LIGGGHTS Documentation, Version 3.X

LIGGGHTS stands for LAMMPS Improved for General Granular and Granular Heat Transfer Simulations. It is part of the [CFDEMproject](#), www.cfdem.com

The core developers of LIGGGHTS are Christoph Kloss (DCS Computing GmbH, Linz and JKU Linz) and Richard Berger (JKU Linz), with major contributions from Philippe Seil, Andreas Aigner and Stefan Amberger (all JKU Linz) and Christoph Goniva (DCS Computing GmbH, Linz and JKU Linz)

[CFDEMproject](#) has more information about the code and its uses. For questions about the code, please use the forums at [CFDEMproject](#).

LIGGGHTS is based on LAMMPS (see below), and so is its manual. So if the manual says 'LAMMPS', you could read 'LIGGGHTS' instead. However, we want to make clear which parts of the code and framework stem from the LAMMPS base.

LIGGGHTS Version info:

All LIGGGHTS versions are based on a specific version of LAMMPS, as printed in the file src/version.h LIGGGHTS version are identidied by a version number (e.g. '3.0'), a branch name (e.g. 'LIGGGHTS-PUBLIC' for the public release of LIGGGHTS), compilation info (date / time stamp and user name), and a LAMMPS version number (which is the LAMMPS version that the LIGGGHTS release is based on). For info on the LAMMPS version, see below.

LAMMPS Version info:

The LAMMPS "version" is the date when it was released, such as 1 May 2010. LAMMPS is updated continuously. Whenever we fix a bug or add a feature, we release it immediately, and post a notice on [this page of the WWW site](#). Each dated copy of LAMMPS contains all the features and bug-fixes up to and including that version date. The version date is printed to the screen and logfile every time you run LAMMPS. It is also in the file src/version.h and in the LAMMPS directory name created when you unpack a tarball, and at the top of the first page of the manual (this page).

- If you browse the HTML doc pages on the LAMMPS WWW site, they always describe the most current version of LAMMPS.
- If you browse the HTML doc pages included in your tarball, they describe the version you have.
- The PDF file on the WWW site or in the tarball is updated about once per month. This is because it is large, and we don't want it to be part of every patch.
- There is also a Developer.pdf file in the doc directory, which describes the internal structure and algorithms of LAMMPS.

LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator.

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier who can be contacted at sjplimp,athomps,pscrozi at sandia.gov. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about the code and its uses.

The LAMMPS documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so we can improve the LAMMPS documentation.

Once you are familiar with LAMMPS, you may want to bookmark this page at Section_commands.html#comm since it gives quick access to documentation for all LAMMPS commands.

PDF file of the entire manual, generated by htmldoc

angle_style charmm command

angle_style charmm/omp command

Syntax:

```
angle_style charmm
```

Examples:

```
angle_style charmm
angle_coeff 1 300.0 107.0 50.0 3.0
```

Description:

The *charmm* angle style uses the potential

$$E = K(\theta - \theta_0)^2 + K_{UB}(r - r_{UB})^2$$

with an additional Urey_Bradley term based on the distance *r* between the 1st and 3rd atoms in the angle. K, theta0, Kub, and Rub are coefficients defined for each angle type.

See [(MacKerell)](#) for a description of the CHARMM force field.

The following coefficients must be defined for each angle type via the [angle_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy/radian^2)
- theta0 (degrees)
- K_ub (energy/distance^2)
- r_ub (distance)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the **Making LAMMPS** section for more info on packages.

Related commands:

angle_coeff

Default: none

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

angle_style class2 command

angle_style class2/omp command

Syntax:

```
angle_style class2
```

Examples:

```
angle_style class2
angle_coeff * 75.0
angle_coeff 1 bb 10.5872 1.0119 1.5228
angle_coeff * ba 3.6551 24.895 1.0119 1.5228
```

Description:

The *class2* angle style uses the potential

$$
\begin{aligned}
E &= E_a + E_{bb} + E_{ba} \\
E_a &= K_2(\theta - \theta_0)^2 + K_3(\theta - \theta_0)^3 + K_4(\theta - \theta_0)^4 \\
E_{bb} &= M(r_{ij} - r_1)(r_{jk} - r_2) \\
E_{ba} &= N_1(r_{ij} - r_1)(\theta - \theta_0) + N_2(r_{jk} - r_2)(\theta - \theta_0)
\end{aligned}
$$

where Ea is the angle term, Ebb is a bond-bond term, and Eba is a bond-angle term. Theta0 is the equilibrium angle and r1 and r2 are the equilibrium bond lengths.

See [(Sun)](#) for a description of the COMPASS class2 force field.

Coefficients for the Ea, Ebb, and Eba formulas must be defined for each angle type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands.

These are the 4 coefficients for the Ea formula:

- theta0 (degrees)
- K2 (energy/radian^2)
- K3 (energy/radian^3)
- K4 (energy/radian^4)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of the various K are in per-radian.

For the Ebb formula, each line in a [bond_coeff](#) command in the input script lists 4 coefficients, the first of which is "bb" to indicate they are BondBond coefficients. In a data file, these coefficients should be listed under a "BondBond Coeffs" heading and you must leave out the "bb", i.e. only list 3 coefficients after the angle type.

- bb
- M (energy/distance^2)
- r1 (distance)
- r2 (distance)

For the Eba formula, each line in a [bond_coeff](#) command in the input script lists 5 coefficients, the first of which is "ba" to indicate they are BondAngle coefficients. In a data file, these coefficients should be listed under a "BondAngle Coeffs" heading and you must leave out the "ba", i.e. only list 4 coefficients after the angle type.

- ba
- N1 (energy/distance^2)
- N2 (energy/distance^2)
- r1 (distance)
- r2 (distance)

The theta0 value in the Eba formula is not specified, since it is the same value from the Ea formula.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the CLASS2 package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[angle_coeff](#)

Default: none

(Sun) Sun, J Phys Chem B 102, 7338-7364 (1998).

angle_coeff command

Syntax:

```
angle_coeff N args
```

- N = angle type (see asterisk form below)
- args = coefficients for one or more angle types

Examples:

```
angle_coeff 1 300.0 107.0
angle_coeff * 5.0
angle_coeff 2*10 5.0
```

Description:

Specify the angle force field coefficients for one or more angle types. The number and meaning of the coefficients depends on the angle style. Angle coefficients can also be set in the data file read by the read_data command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the coefficients for multiple angle types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of angle types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using an angle_coeff command can override a previous setting for the same angle type. For example, these commands set the coeffs for all angle types, then overwrite the coeffs for just angle type 2:

```
angle_coeff * 200.0 107.0 1.2
angle_coeff 2 50.0 107.0
```

A line in a data file that specifies angle coefficients uses the exact same format as the arguments of the angle_coeff command in an input script, except that wild-card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Angle Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 107.0
```

The angle_style class2 is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross-term coefficients. See its doc page for details.

Here is an alphabetic list of angle styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated angle_coeff command.

Note that there are also additional angle styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the angle section of this page.

- angle_style none - turn off angle interactions
- angle_style hybrid - define multiple styles of angle interactions

- angle_style charmm - CHARMM angle

- angle_style class2 - COMPASS (class 2) angle
- angle_style cosine - cosine angle potential
- angle_style cosine/delta - difference of cosines angle potential
- angle_style cosine/periodic - DREIDING angle
- angle_style cosine/squared - cosine squared angle potential
- angle_style harmonic - harmonic angle
- angle_style table - tabulated by angle

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

An angle style must be defined before any angle coefficients are set, either in the input script or in a data file.

Related commands:

angle_style

Default: none

angle_style cosine/delta command

angle_style cosine/delta/omp command

Syntax:

```
angle_style cosine/delta
```

Examples:

```
angle_style cosine/delta
angle_coeff 2*4 75.0 100.0
```

Description:

The *cosine/delta* angle style uses the potential

$$E = K[1 - \cos(\theta - \theta_0)]$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_style cosine/squared

Default: none

angle_style cosine command

angle_style cosine/omp command

Syntax:

```
angle_style cosine
```

Examples:

```
angle_style cosine
angle_coeff * 75.0
```

Description:

The *cosine* angle style uses the potential

$$E = K[1 + \cos(\theta)]$$

where K is defined for each angle type.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style cosine/periodic command

angle_style cosine/periodic/omp command

Syntax:

```
angle_style cosine/periodic
```

Examples:

```
angle_style cosine/periodic
angle_coeff * 75.0 1 6
```

Description:

The *cosine/periodic* angle style uses the following potential, which is commonly used in the DREIDING force field, particularly for organometallic systems where $n = 4$ might be used for an octahedral complex and $n = 3$ might be used for a trigonal center:

$$E = C \left[1 - B(-1)^n cos\left(n\theta\right) \right]$$

where C, B and n are coefficients defined for each angle type.

See (Mayo) for a description of the DREIDING force field

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- C (energy)
- B = 1 or -1
- n = 1, 2, 3, 4, 5 or 6 for periodicity

Note that the prefactor C is specified and not the overall force constant K = C / n^2. When B = 1, it leads to a minimum for the linear geometry. When B = -1, it leads to a maximum for the linear geometry.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[angle_coeff](#)

Default: none

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897-8909 (1990).

angle_style cosine/shift/exp command

angle_style cosine/shift/exp/omp command

Syntax:

```
angle_style cosine/shift/exp
```

Examples:

```
angle_style cosine/shift/exp
angle_coeff * 10.0 45.0 2.0
```

Description:

The *cosine/shift/exp* angle style uses the potential

$$ E = -U_{min} \frac{e^{-aU(\theta,\theta_0)} - 1}{e^a - 1} \quad \text{with} \quad U(\theta,\theta_0) = -0.5\left(1 + \cos(\theta - \theta_0)\right) $$

where Umin, theta, and a are defined for each angle type.

The potential is bounded between [-Umin:0] and the minimum is located at the angle theta0. The a parameter can be both positive or negative and is used to control the spring constant at the equilibrium.

The spring constant is given by k = A exp(A) Umin / [2 (Exp(a)-1)]. For a > 3, k/Umin = a/2 to better than 5% relative error. For negative values of the a parameter, the spring constant is essentially zero, and anharmonic terms takes over. The potential is furthermore well behaved in the limit a -> 0, where it has been implemented to linear order in a for a < 0.001. In this limit the potential reduces to the cosineshifted potential.

The following coefficients must be defined for each angle type via the [angle_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- umin (energy)
- theta (angle)
- A (real number)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER-MISC package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[angle_coeff](#), [angle_cosineshift](#), [dihedral_cosineshift](#)

Default: none

angle_style cosine/shift command

angle_style cosine/shift/omp command

Syntax:

```
angle_style cosine/shift
```

Examples:

```
angle_style cosine/shift
angle_coeff * 10.0 45.0
```

Description:

The *cosine/shift* angle style uses the potential

$$E = -\frac{Umin}{2}\left[1 + Cos(\theta - \theta_0)\right]$$

where theta0 is the equilibrium angle. The potential is bounded between -Umin and zero. In the neighborhood of the minimum E=- Umin + Umin/4(theta-theta0)^2 hence the spring constant is umin/2.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- umin (energy)
- theta (angle)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_cosineshiftexp

Default: none

angle_style cosine/squared command

angle_style cosine/squared/omp command

Syntax:

```
angle_style cosine/squared
```

Examples:

```
angle_style cosine/squared
angle_coeff 2*4 75.0 100.0
```

Description:

The *cosine/squared* angle style uses the potential

$$E = K[\cos(\theta) - \cos(\theta_0)]^2$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style dipole command

angle_style dipole/omp command

Syntax:

```
angle_style dipole
```

Examples:

```
angle_style dipole
angle_coeff 6 2.1 180.0
```

Description:

The *dipole* angle style is used to control the orientation of a dipolar atom within a molecule [(Orsi)](#). Specifically, the *dipole* angle style restrains the orientation of a point dipole mu_j (embedded in atom 'j') with respect to a reference (bond) vector r_ij = r_i - r_j, where 'i' is another atom of the same molecule (typically, 'i' and 'j' are also covalently bonded).

It is convenient to define an angle gamma between the 'free' vector mu_j and the reference (bond) vector r_ij:

$$\cos\gamma = \frac{\vec{\mu_j} \bullet \vec{r_{ij}}}{\mu_j\, r_{ij}}$$

The *dipole* angle style uses the potential:

$$E = K(\cos\gamma - \cos\gamma_0)^2$$

where K is a rigidity constant and gamma0 is an equilibrium (reference) angle.

The torque on the dipole can be obtained by differentiating the potential using the 'chain rule' as in appendix C.3 of [(Allen)](#):

$$\vec{T_j} = \frac{2K(\cos\gamma - \cos\gamma_0)}{\mu_j\, r_{ij}} \vec{r_{ij}} \times \vec{\mu_j}$$

Example: if gamma0 is set to 0 degrees, the torque generated by the potential will tend to align the dipole along the reference direction defined by the (bond) vector r_ij (in other words, mu_j is restrained to point towards atom 'i').

Note that the angle dipole potential does not give rise to any force, because it does not depend on the distance between i and j (it only depends on the angle between mu_j and r_ij).

angle_style dipole command 19

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- gamma0 (degrees)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

IMPORTANT NOTE: In the "Angles" section of the data file, the atom ID 'j' corresponding to the dipole to restrain must come before the atom ID of the reference atom 'i'. A third atom ID 'k' must also be provided, although 'k' is just a 'dummy' atom which can be any atom; it may be useful to choose a convention (e.g., 'k'='i') and adhere to it. For example, if ID=1 for the dipolar atom to restrain, and ID=2 for the reference atom, the corresponding line in the "Angles" section of the data file would read: X X 1 2 2

The "newton" command for intramolecular interactions must be "on" (which is the default).

This angle style should not be used with SHAKE.

Related commands:

angle_coeff, angle_hybrid

Default: none

(Orsi) Orsi & Essex, The ELBA force field for coarse-grain modeling of lipid membranes, PloS ONE 6(12): e28637, 2011.

(Allen) Allen & Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.

angle_style fourier command

angle_style fourier/omp command

Syntax:

```
angle_style fourier
```

Examples:

angle_style fourier angle_coeff 75.0 1.0 1.0 1.0

Description:

The *fourier* angle style uses the potential

$$E = K[C_0 + C_1 \cos(\theta) + C_2 \cos(2\theta)]$$

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- C0 (real)
- C1 (real)
- C2 (real)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style fourier/simple command

angle_style fourier/simple/omp command

Syntax:

```
angle_style fourier/simple
```

Examples:

angle_style fourier/simple angle_coeff 100.0 -1.0 1.0

Description:

The *fourier/simple* angle style uses the potential

$$E = K[1.0 + c\cos(n\theta)]$$

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- c (real)
- n (real)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style harmonic command

angle_style harmonic/omp command

Syntax:

```
angle_style harmonic
```

Examples:

```
angle_style harmonic
angle_coeff 1 300.0 107.0
```

Description:

The *harmonic* angle style uses the potential

$$E = K(\theta - \theta_0)^2$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the [angle_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy/radian^2)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions: none

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

angle_coeff

Default: none

angle_style hybrid command

Syntax:

```
angle_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more angle styles

Examples:

```
angle_style hybrid harmonic cosine
angle_coeff 1 harmonic 80.0 30.0
angle_coeff 2* cosine 50.0
```

Description:

The *hybrid* style enables the use of multiple angle styles in one simulation. An angle style is assigned to each angle type. For example, angles in a polymer flow (of angle type 1) could be computed with a *harmonic* potential and angles in the wall boundary (of angle type 2) could be computed with a *cosine* potential. The assignment of angle type to style is made via the [angle_coeff](#) command or in the data file.

In the angle_coeff commands, the name of an angle style must be added after the angle type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 angle_coeff commands set angles of angle type 1 to be computed with a *harmonic* potential with coefficients 80.0, 30.0 for K, theta0. All other angle types (2-N) are computed with a *cosine* potential with coefficient 50.0 for K.

If angle coefficients are specified in the data file read via the [read_data](#) command, then the same rule applies. E.g. "harmonic" or "cosine", must be added after the angle type, for each line in the "Angle Coeffs" section, e.g.

```
Angle Coeffs

1 harmonic 80.0 30.0
2 cosine 50.0
...
```

If *class2* is one of the angle hybrid styles, the same rule holds for specifying additional BondBond (and BondAngle) coefficients either via the input script or in the data file. I.e. *class2* must be added to each line after the angle type. For lines in the BondBond (or BondAngle) section of the data file for angle types that are not *class2*, you must use an angle style of *skip* as a placeholder, e.g.

```
BondBond Coeffs

1 skip
2 class2 3.6512 1.0119 1.0119
...
```

Note that it is not necessary to use the angle style *skip* in the input script, since BondBond (or BondAngle) coefficients need not be specified at all for angle types that are not *class2*.

An angle style of *none* with no additional coefficients can be used in place of an angle style, either in a input script angle_coeff command or in the data file, if you desire to turn off interactions for specific angle types.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Unlike other angle styles, the hybrid angle style does not store angle coefficient info for individual sub-styles in a [binary restart files](#). Thus when retarting a simulation from a restart file, you need to re-specify angle_coeff commands.

Related commands:

[angle_coeff](#)

Default: none

angle_style none command

Syntax:

```
angle_style none
```

Examples:

```
angle_style none
```

Description:

Using an angle style of none means angle forces are not computed, even if triplets of angle atoms were listed in the data file read by the [read_data](#) command.

Restrictions: none

Related commands: none

Default: none

angle_style quartic command

angle_style quartic/omp command

Syntax:

```
angle_style quartic
```

Examples:

```
angle_style quartic
angle_coeff 1 129.1948 56.8726 -25.9442 -14.2221
```

Description:

The *quartic* angle style uses the potential

$$E = K_2(\theta - \theta_0)^2 + K_3(\theta - \theta_0)^3 + K_4(\theta - \theta_0)^4$$

where theta0 is the equilibrium value of the angle, and K is a prefactor. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- theta0 (degrees)
- K2 (energy/radian^2)
- K3 (energy/radian^3)
- K4 (energy/radian^4)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[angle_coeff](#)

Default: none

angle_style sdk command

Syntax:

```
angle_style sdk
```

```
angle_style sdk/omp
```

Examples:

```
angle_style sdk
angle_coeff 1 300.0 107.0
```

Description:

The *sdk* angle style is a combination of the harmonic angle potential,

$$E = K(\theta - \theta_0)^2$$

where theta0 is the equilibrium value of the angle and K a prefactor, with the *repulsive* part of the non-bonded *lj/sdk* pair style between the atoms 1 and 3. This angle potential is intended for coarse grained MD simulations with the CMM parametrization using the pair_style lj/sdk. Relative to the pair_style *lj/sdk*, however, the energy is shifted by *epsilon*, to avoid sudden jumps. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each angle type via the angle_coeff command as in the example above:

- K (energy/radian^2)
- theta0 (degrees)

Theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2. The also required *lj/sdk* parameters will be extracted automatically from the pair_style.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER-CG-CMM package. See the Making LAMMPS section for more info on packages.

Related commands:

angle_coeff, angle_style harmonic, pair_style lj/sdk, pair_style lj/sdk/coul/long

Default: none

angle_style command

Syntax:

```
angle_style style
```

- style = *none* or *hybrid* or *charmm* or *class2* or *cosine* or *cosine/squared* or *harmonic*

Examples:

```
angle_style harmonic
angle_style charmm
angle_style hybrid harmonic cosine
```

Description:

Set the formula(s) LAMMPS uses to compute angle interactions between triplets of atoms, which remain in force for the duration of the simulation. The list of angle triplets is read in by a read_data or read_restart command from a data or restart file.

Hybrid models where angles are computed using different angle potentials can be setup using the *hybrid* angle style.

The coefficients associated with a angle style can be specified in a data or restart file or via the angle_coeff command.

All angle potentials store their coefficient data in binary restart files which means angle_style and angle_coeff commands do not need to be re-specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that angle_style *hybrid* only stores the list of sub-styles in the restart file; angle coefficients need to be re-specified.

IMPORTANT NOTE: When both an angle and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 3 bonded atoms.

In the formulas listed for each angle style, *theta* is the angle between the 3 atoms in the angle.

Here is an alphabetic list of angle styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated angle_coeff command.

Note that there are also additional angle styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the angle section of this page.

- angle_style none - turn off angle interactions
- angle_style hybrid - define multiple styles of angle interactions

- angle_style charmm - CHARMM angle
- angle_style class2 - COMPASS (class 2) angle
- angle_style cosine - cosine angle potential
- angle_style cosine/delta - difference of cosines angle potential
- angle_style cosine/periodic - DREIDING angle
- angle_style cosine/squared - cosine squared angle potential
- angle_style harmonic - harmonic angle
- angle_style table - tabulated by angle

Restrictions:

Angle styles can only be set for atom_styles that allow angles to be defined.

Most angle styles are part of the MOLECULAR package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual bond potentials tell if it is part of a package.

Related commands:

angle_coeff

Default:

```
angle_style none
```

angle_style table command

angle_style table/omp command

Syntax:

```
angle_style table style N
```

- style = *linear* or *spline* = method of interpolation
- N = use N values in table

Examples:

```
angle_style table linear 1000
angle_coeff 3 file.table ENTRY1
```

Description:

Style *table* creates interpolation tables of length *N* from angle potential and derivative values listed in a file(s) as a function of angle The files are read by the [angle_coeff](#) command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and derivative values at each of *N* angles. During a simulation, these tables are used to interpolate energy and force values on individual atoms as needed. The interpolation is done in one of 2 styles: *linear* or *spline*.

For the *linear* style, the angle is used to find 2 surrounding table values from which an energy or its derivative is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The angle is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or derivative.

The following coefficients must be defined for each angle type via the [angle_coeff](#) command as in the example above.

- filename
- keyword

The filename specifies a file containing tabulated energy and derivative values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Angle potential for harmonic (one or more comment or blank lines)

HAM                             (keyword is the first text on line)
N 181 FP 0 0 EQ 90.0            (N, FP, EQ parameters)
                                (blank line)
N 181 FP 0 0                    (N, FP parameters)
1 0.0 200.5 2.5                 (index, angle, energy, derivative)
2 1.0 198.0 2.5
...
181 180.0 0.0 0.0
```

A section begins with a non-blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the angle_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the angle_style table command. Let Ntable = *N* in the angle_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and derivative values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual angles and their atoms. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile.

The "FP" parameter is optional. If used, it is followed by two values fplo and fphi, which are the 2nd derivatives at the innermost and outermost angle settings. These values are needed by the spline construction routines. If not specified by the "FP" parameter, they are estimated (less accurately) by the first two and last two derivative values in the table.

The "EQ" parameter is also optional. If used, it is followed by a the equilibrium angle value, which is used, for example, by the fix shake command. If not used, the equilibrium angle is set to 180.0.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is the angle value (in degrees), the 3rd value is the energy (in energy units), and the 4th is -dE/d(theta) (also in energy units). The 3rd term is the energy of the 3-atom configuration for the specified angle. The last term is the derivative of the energy with respect to the angle (in degrees, not radians). Thus the units of the last term are still energy, not force. The angle values must increase from one line to the next. The angle values must also begin with 0.0 and end with 180.0, i.e. span the full range of possible angles.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

angle_style table/omp command   36

angle_coeff

Default: none

atom_modify command

Syntax:

```
atom_modify keyword values ...
```

- one or more keyword/value pairs may be appended
- keyword = *map* or *first* or *sort*

```
  map value = array or hash
    first value = group-ID = group whose atoms will appear first in internal atom lists
    sort values = Nfreq binsize
      Nfreq = sort atoms spatially every this many time steps
      binsize = bin size for spatial sorting (distance units)
```

Examples:

```
atom_modify map hash
atom_modify map array sort 10000 2.0
atom_modify first colloid
```

Description:

Modify properties of the atom style selected within LAMMPS.

The *map* keyword determines how atom ID lookup is done for molecular problems. Lookups are performed by bond (angle, etc) routines in LAMMPS to find the local atom index associated with a global atom ID. When the *array* value is used, each processor stores a lookup table of length N, where N is the total # of atoms in the system. This is the fastest method for most simulations, but a processor can run out of memory to store the table for very large simulations. The *hash* value uses a hash table to perform the lookups. This method can be slightly slower than the *array* method, but its memory cost is proportional to N/P on each processor, where P is the total number of processors running the simulation.

The *first* keyword allows a [group](#) to be specified whose atoms will be maintained as the first atoms in each processor's list of owned atoms. This in only useful when the specified group is a small fraction of all the atoms, and there are other operations LAMMPS is performing that will be sped-up significantly by being able to loop over the smaller set of atoms. Otherwise the reordering required by this option will be a net slow-down. The [neigh_modify include](#) and [communicate group](#) commands are two examples of commands that require this setting to work efficiently. Several [fixes](#), most notably time integration fixes like [fix nve](#), also take advantage of this setting if the group they operate on is the group specified by this command. Note that specifying "all" as the group-ID effectively turns off the *first* option.

It is OK to use the *first* keyword with a group that has not yet been defined, e.g. to use the atom_modify first command at the beginning of your input script. LAMMPS does not use the group until a simullation is run.

The *sort* keyword turns on a spatial sorting or reordering of atoms within each processor's sub-domain every *Nfreq* timesteps. If *Nfreq* is set to 0, then sorting is turned off. Sorting can improve cache performance and thus speed-up a LAMMPS simulation, as discussed in a paper by [(Meloni)](#). Its efficacy depends on the problem size (atoms/processor), how quickly the system becomes disordered, and various other factors. As a general rule, sorting is typically more effective at speeding up simulations of liquids as opposed to solids. In tests we have done, the speed-up can range from zero to 3-4x.

Reordering is peformed every *Nfreq* timesteps during a dynamics run or iterations during a minimization. More precisely, reordering occurs at the first reneighboring that occurs after the target timestep. The

reordering is performed locally by each processor, using bins of the specified *binsize*. If *binsize* is set to 0.0, then a binsize equal to half the [neighbor](#) cutoff distance (force cutoff plus skin distance) is used, which is a reasonable value. After the atoms have been binned, they are reordered so that atoms in the same bin are adjacent to each other in the processor's 1d list of atoms.

The goal of this procedure is for atoms to put atoms close to each other in the processor's one-dimensional list of atoms that are also near to each other spatially. This can improve cache performance when pairwise intereractions and neighbor lists are computed. Note that if bins are too small, there will be few atoms/bin. Likewise if bins are too large, there will be many atoms/bin. In both cases, the goal of cache locality will be undermined.

IMPORTANT NOTE: Running a simulation with sorting on versus off should not change the simulation results in a statistical sense. However, a different ordering will induce round-off differences, which will lead to diverging trajectories over time when comparing two simluations. Various commands, particularly those which use random numbers (e.g. [velocity create](#), and [fix langevin](#)), may generate (statistically identical) results which depend on the order in which atoms are processed. The order of atoms in a [dump](#) file will also typically change if sorting is enabled.

Restrictions:

The map keyword can only be used before the simulation box is defined by a [read_data](#) or [create_box](#) command.

The *first* and *sort* options cannot be used together. Since sorting is on by default, it will be turned off if the *first* keyword is used with a group-ID that is not "all".

Related commands: none

Default:

By default, atomic (non-molecular) problems do not allocate maps. For molecular problems, the option default is map = array. By default, a "first" group is not defined. By default, sorting is enabled with a frequency of 1000 and a binsize of 0.0, which means the neighbor cutoff will be used to set the bin size.

(Meloni) Meloni, Rosati and Colombo, J Chem Phys, 126, 121102 (2007).

atom_style command

Syntax:

```
atom_style style args
```

- style = *angle* or *atomic* or *body* or *bond* or *charge* or *dipole* or *electron* or *ellipsoid* or *full* or *line* or *meso* or *molecular* or *peri* or *sphere* or *granular* or *tri* or *hybrid* or *sph*

```
args = none for any style except body and hybrid
body args = bstyle bstyle-args
  bstyle = style of body particles
  bstyle-args = additional arguments specific to the bstyle
               see the body doc page for details
hybrid args = list of one or more sub-styles, each with their args
```

Examples:

```
atom_style atomic
atom_style bond
atom_style full
atom_style body nparticle 2 10
atom_style hybrid charge bond
atom_style hybrid charge body nparticle 2 5
```

Description:

Define what style of atoms to use in a simulation. This determines what attributes are associated with the atoms. This command must be used before a simulation is setup via a read_data, read_restart, or create_box command.

Once a style is assigned, it cannot be changed, so use a style general enough to encompass all attributes. E.g. with style *bond*, angular terms cannot be used or added later to the model. It is OK to use a style more general than needed, though it may be slightly inefficient.

The choice of style affects what quantities are stored by each atom, what quantities are communicated between processors to enable forces to be computed, and what quantities are listed in the data file read by the read_data command.

These are the additional attributes of each style and the typical kinds of physical systems they are used to model. All styles store coordinates, velocities, atom IDs and types. See the read_data, create_atoms, and set commands for info on how to set these various quantities.

angle	bonds and angles	bead-spring polymers with stiffness
atomic	only the default values	coarse-grain liquids, solids, metals
body	mass, inertia moments, quaternion, angular momentum	arbitrary bodies
bond	bonds	bead-spring polymers
charge	charge	atomic system with charges
dipole	charge and dipole moment	system with dipolar particles
electron	charge and spin and eradius	electronic force field
ellipsoid	shape, quaternion, angular momentum	aspherical particles

full	molecular + charge	bio-molecules
line	end points, angular velocity	rigid bodies
meso	rho, e, cv	SPH particles
sph	q(pressure), density	SPH particles
molecular	bonds, angles, dihedrals, impropers	uncharged molecules
peri	mass, volume	mesocopic Peridynamic models
sphere or granular	diameter, mass, angular velocity	granular models
tri	corner points, angular momentum	rigid bodies
wavepacket	charge, spin, eradius, etag, cs_re, cs_im	AWPMD

IMPORTANT NOTE: It is possible to add some attributes, such as a molecule ID, to atom styles that do not have them via the fix property/atom command. This command also allows new custom attributes consisting of extra integer or floating-point values to be added to atoms. See the fix property/atom doc page for examples of cases where this is useful and details on how to initialize, access, and output the custom values.

All of the above styles define point particles, except the *sphere*, *ellipsoid*, *electron*, *peri*, *wavepacket*, *line*, *tri*, and *body* styles, which define finite-size particles. See Section_howto 14 for an overview of using finite-size particle models with LAMMPS.

All of the styles assign mass to particles on a per-type basis, using the mass command, except for the finite-size particle styles. They assign mass to individual particles on a per-particle basis.

For the *sphere* style, the particles are spheres and each stores a per-particle diameter and mass. If the diameter > 0.0, the particle is a finite-size sphere. If the diameter = 0.0, it is a point particle.

For the *ellipsoid* style, the particles are ellipsoids and each stores a flag which indicates whether it is a finite-size ellipsoid or a point particle. If it is an ellipsoid, it also stores a shape vector with the 3 diamters of the ellipsoid and a quaternion 4-vector with its orientation.

For the *electron* style, the particles representing electrons are 3d Gaussians with a specified position and bandwidth or uncertainty in position, which is represented by the eradius = electron size.

For the *peri* style, the particles are spherical and each stores a per-particle mass and volume.

The *meso* style is for smoothed particle hydrodynamics (SPH) particles which store a density (rho), energy (e), and heat capacity (cv).

The *wavepacket* style is similar to *electron*, but the electrons may consist of several Gaussian wave packets, summed up with coefficients cs= (cs_re,cs_im). Each of the wave packets is treated as a separate particle in LAMMPS, wave packets belonging to the same electron must have identical *etag* values.

For the *line* style, the particles are idealized line segments and each stores a per-particle mass and length and orientation (i.e. the end points of the line segment).

For the *tri* style, the particles are planar triangles and each stores a per-particle mass and size and orientation (i.e. the corner points of the triangle).

For the *body* style, the particles are arbitrary bodies with internal attributes defined by the "style" of the bodies, which is specified by the *bstyle* argument. Body particles can represent complex entities, such as surface meshes of discrete points, collections of sub-particles, deformable objects, etc.

The body doc page descibes the body styles LAMMPS currently supports, and provides more details as to the kind of body particles they represent. For all styles, each body particle stores moments of inertia and a quaternion 4-vector, so that its orientation and position can be time integrated due to forces and torques.

Note that there may be additional arguments required along with the *bstyle* specification, in the atom_style body command. These arguments are described in the [body](#) doc page.

Typically, simulations require only a single (non-hybrid) atom style. If some atoms in the simulation do not have all the properties defined by a particular style, use the simplest style that defines all the needed properties by any atom. For example, if some atoms in a simulation are charged, but others are not, use the *charge* style. If some atoms have bonds, but others do not, use the *bond* style.

The only scenario where the *hybrid* style is needed is if there is no single style which defines all needed properties of all atoms. For example, if you want dipolar particles which will rotate due to torque, you would need to use "atom_style hybrid sphere dipole". When a hybrid style is used, atoms store and communicate the union of all quantities implied by the individual styles.

LAMMPS can be extended with new atom styles as well as new body styles; see [this section](#).

Restrictions:

This command cannot be used after the simulation box is defined by a [read_data](#) or [create_box](#) command.

The *angle*, *bond*, *full*, and *molecular* styles are part of the MOLECULAR package. The *line* and *tri* styles are part of the ASPHERE pacakge. The *body* style is part of the BODY package. The *dipole* style is part of the DIPOLE package. The *peri* style is part of the PERI package for Peridynamics. The *electron* style is part of the USER-EFF package for [electronic force fields](#). The *meso* style is part of the USER-SPH package for smoothed particle hydrodyanmics (SPH). See [this PDF guide](#) to using SPH in LAMMPS. The *wavepacket* style is part of the USER-AWPMD package for the [antisymmetrized wave packet MD method](#). They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[read_data](#), [pair_style](#)

Default:

atom_style atomic

Body particles

Overview:

This doc page is not about a LAMMPS input script command, but about body particles, which are generalized finite-size particles. Individual body particles can represent complex entities, such as surface meshes of discrete points, collections of sub-particles, deformable objects, etc. Note that other kinds of finite-size spherical and aspherical particles are also supported by LAMMPS, such as spheres, ellipsoids, line segments, and triangles, but they are simpler entities that body particles. See [Section_howto 14](#) for a general overview of all these particle types.

Body particles are used via the [atom_style body](#) command. It takes a body style as an argument. The current body styles supported by LAMMPS are as follows. The name in the first column is used as the *bstyle* argument for the [atom_style body](#) command.

nparticle	rigid body with N sub-particles

The body style determines what attributes are stored for each body and thus how they can be used to compute pairwise body/body or bond/non-body (point particle) interactions. More details of each style are described below.

We hope to add more styles in the future. See [Section_modify 12](#) for details on how to add a new body style to the code.

When to use body particles:

You should not use body particles to model a rigid body made of simpler particles (e.g. point, sphere, ellipsoid, line segment, triangular particles), if the interaction between pairs of rigid bodies is just the summation of pairwise interactions between the simpler particles. LAMMPS already supports this kind of model via the [fix rigid](#) command. Any of the numerous pair styles that compute interactions between simpler particles can be used. The [fix rigid](#) command time integrates the motion of the rigid bodies. All of the standard LAMMPS commands for thermostatting, adding constraints, performing output, etc will operate as expected on the simple particles.

By contrast, when body particles are used, LAMMPS treats an entire body as a single particle for purposes of computing pairwise interactions, building neighbor lists, migrating particles between processors, outputting particles to a dump file, etc. This means that interactions between pairs of bodies or between a body and non-body (point) particle need to be encoded in an appropriate pair style. If such a pair style were to mimic the [fix rigid](#) model, it would need to loop over the entire collection of interactions between pairs of simple particles within the two bodies, each time a single body/body interaction was computed.

Thus it only makes sense to use body particles and develop such a pair style, when particle/particle interactions are more complex than what the [fix rigid](#) command can already calculate. For example, if particles have one or more of the following attributes:

- represented by a surface mesh
- represented by a collection of geometric entities (e.g. planes + spheres)
- deformable
- internal stress that induces fragmentation

then the interaction between pairs of particles is likely to be more complex than the summation of simple sub-particle interactions. An example is contact or frictional forces between particles with planar sufaces that inter-penetrate.

These are additional LAMMPS commands that can be used with body particles of different styles

fix nve/body	integrate motion of a body particle
compute body/local	store sub-particle attributes of a body particle
dump local	output sub-particle attributes of a body particle

The pair styles defined for use with specific body styles are listed in the sections below.

Specifics of body style nparticle:

The *nparticle* body style represents body particles as a rigid body with a variable number N of sub-particles. It is provided as a vanillia, prototypical example of a body particle, although as mentioned above, the fix rigid command already duplicates its functionality.

The atom_style body command for this body style takes two additional arguments:

```
atom_style body nparticle Nmin Nmax
Nmin = minimum # of sub-particles in any body in the system
Nmax = maximum # of sub-particles in any body in the system
```

The Nmin and Nmax arguments are used to bound the size of data structures used internally by each particle.

When the read_data command reads a data file for this body style, the following information must be provided for each entry in the *Bodies* section of the data file:

```
atom-ID 1 M
N
ixx iyy izz ixy ixz iyz x1 y1 z1 ...
...
... xN yN zN
```

N is the number of sub-particles in the body particle. M = 6 + 3*N. The integer line has a single value N. The floating point line(s) list 6 moments of inertia followed by the coordinates of the N sub-particles (x1 to zN) as 3N values on as many lines as required. Note that this in not N lines, but 10 values per line; see the read_data command for details. The 6 moments of inertia (ixx,iyy,izz,ixy,ixz,iyz) should be the values consistent with the current orientation of the rigid body around its center of mass. The values are with respect to the simulation box XYZ axes, not with respect to the prinicpal axes of the rigid body itself. LAMMPS performs the latter calculation internally. The coordinates of each sub-particle are specified as its x,y,z displacement from the center-of-mass of the body particle. The center-of-mass position of the particle is specified by the x,y,z values in the *Atoms* section of the data file.

The pair_style body command can be used with this body style to compute body/body and body/non-body interactions.

For output purposes via the compute body/local and dump local commands, this body style produces one datum for each of the N sub-particles in a body particle. The datum has 3 values:

```
1 = x position of sub-particle
2 = y position of sub-particle
3 = z position of sub-particle
```

These values are the current position of the sub-particle within the simulation domain, not a displacement from the center-of-mass (COM) of the body particle itself. These values are calculated using the current COM and orientiation of the body particle.

bond_style class2 command

bond_style class2/omp command

Syntax:

```
bond_style class2
```

Examples:

```
bond_style class2
bond_coeff 1 1.0 100.0 80.0 80.0
```

Description:

The *class2* bond style uses the potential

$$E = K_2(r - r_0)^2 + K_3(r - r_0)^3 + K_4(r - r_0)^4$$

where r0 is the equilibrium bond distance.

See [(Sun)](#) for a description of the COMPASS class2 force field.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- R0 (distance)
- K2 (energy/distance^2)
- K3 (energy/distance^3)
- K4 (energy/distance^4)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the CLASS2 package. See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

(Sun) Sun, J Phys Chem B 102, 7338-7364 (1998).

bond_coeff command

Syntax:

```
bond_coeff N args
```

- N = bond type (see asterisk form below)
- args = coefficients for one or more bond types

Examples:

```
bond_coeff 5 80.0 1.2
bond_coeff * 30.0 1.5 1.0 1.0
bond_coeff 1*4 30.0 1.5 1.0 1.0
bond_coeff 1 harmonic 200.0 1.0
```

Description:

Specify the bond force field coefficients for one or more bond types. The number and meaning of the coefficients depends on the bond style. Bond coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the coefficients for multiple bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using a bond_coeff command can override a previous setting for the same bond type. For example, these commands set the coeffs for all bond types, then overwrite the coeffs for just bond type 2:

```
bond_coeff * 100.0 1.2
bond_coeff 2 200.0 1.2
```

A line in a data file that specifies bond coefficients uses the exact same format as the arguments of the bond_coeff command in an input script, except that wild-card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Bond Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
5 80.0 1.2
```

Here is an alphabetic list of bond styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated [bond_coeff](#) command.

Note that here are also additional bond styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the bond section of [this page](#).

- [bond_style none](#) - turn off bonded interactions
- [bond_style hybrid](#) - define multiple styles of bond interactions

- [bond_style class2](#) - COMPASS (class 2) bond
- [bond_style fene](#) - FENE (finite-extensible non-linear elastic) bond
- [bond_style fene/expand](#) - FENE bonds with variable size particles

- bond_style harmonic - harmonic bond
- bond_style morse - Morse bond
- bond_style nonlinear - nonlinear bond
- bond_style quartic - breakable quartic bond
- bond_style table - tabulated by bond length

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

A bond style must be defined before any bond coefficients are set, either in the input script or in a data file.

Related commands:

bond_style

Default: none

bond_style fene/expand command

bond_style fene/expand/omp command

Syntax:

```
bond_style fene/expand
```

Examples:

```
bond_style fene/expand
bond_coeff 1 30.0 1.5 1.0 1.0 0.5
```

Description:

The *fene/expand* bond style uses the potential

$$E = -0.5KR_0^2 \ln\left[1 - \left(\frac{(r-\Delta)}{R_0}\right)^2\right] + 4\epsilon\left[\left(\frac{\sigma}{(r-\Delta)}\right)^{12} - \left(\frac{\sigma}{(r-\Delta)}\right)^6\right] +$$

to define a finite extensible nonlinear elastic (FENE) potential [(Kremer)](#), used for bead-spring polymer models. The first term is attractive, the 2nd Lennard-Jones term is repulsive.

The *fene/expand* bond style is similar to *fene* except that an extra shift factor of delta (positive or negative) is added to *r* to effectively change the bead size of the bonded atoms. The first term now extends to R0 + delta and the 2nd term is cutoff at 2^(1/6) sigma + delta.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)
- delta (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your

input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

You typically should specify [special_bonds fene](#) or [special_bonds lj/coul 0 1 1](#) to use this bond style. LAMMPS will issue a warning it that's not the case.

Related commands:

[bond_coeff](#), [delete_bonds](#)

Default: none

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

bond_style fene command

bond_style fene/omp command

Syntax:

```
bond_style fene
```

Examples:

```
bond_style fene
bond_coeff 1 30.0 1.5 1.0 1.0
```

Description:

The *fene* bond style uses the potential

$$E = -0.5KR_0^2 \ln\left[1 - \left(\frac{r}{R_0}\right)^2\right] + 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] + \epsilon$$

to define a finite extensible nonlinear elastic (FENE) potential [(Kremer)](#), used for bead-spring polymer models. The first term is attractive, the 2nd Lennard-Jones term is repulsive. The first term extends to R0, the maximum extent of the bond. The 2nd term is cutoff at 2^(1/6) sigma, the minimum of the LJ potential.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy/distance^2)
- R0 (distance)
- epsilon (energy)
- sigma (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

You typically should specify special_bonds fene or special_bonds lj/coul 0 1 1 to use this bond style. LAMMPS will issue a warning it that's not the case.

Related commands:

bond_coeff, delete_bonds

Default: none

(**Kremer**) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

bond_style harmonic command

bond_style harmonic/omp command

Syntax:

```
bond_style harmonic
```

Examples:

```
bond_style harmonic
bond_coeff 5 80.0 1.2
```

Description:

The *harmonic* bond style uses the potential

$$E = K(r - r_0)^2$$

where r0 is the equilibrium bond distance. Note that the usual 1/2 factor is included in K.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/distance^2)
- r0 (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

bond_style harmonic command 53

Default: none

bond_style harmonic/shift/cut command

bond_style harmonic/shift/cut/omp command

Syntax:

```
bond_style harmonic/shift/cut
```

Examples:

```
bond_style harmonic/shift/cut
bond_coeff 5 10.0 0.5 1.0
```

Description:

The *harmonic/shift/cut* bond style is a shifted harmonic bond that uses the potential

$$E = \frac{Umin}{(r_0 - r_c)^2} \left[(r - r_0)^2 - (r_c - r_0)^2 \right]$$

where r0 is the equilibrium bond distance, and rc the critical distance. The bond potential is zero for distances r > rc. The potential is -Umin at r0 and zero at rc. The spring constant is k = Umin / [2 (r0-rc)^2].

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- Umin (energy)
- r0 (distance)
- rc (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

[bond_coeff](#), [delete_bonds](#), [bond_harmonic](#), [bond_harmonicshift](#)

Default: none

bond_style harmonic/shift command

bond_style harmonic/shift/omp command

Syntax:

```
bond_style harmonic/shift
```

Examples:

```
bond_style harmonic/shift
bond_coeff 5 10.0 0.5 1.0
```

Description:

The *harmonic/shift* bond style is a shifted harmonic bond that uses the potential

$$E = \frac{Umin}{(r_0 - r_c)^2} \left[(r - r_0)^2 - (r_c - r_0)^2 \right]$$

where r0 is the equilibrium bond distance, and rc the critical distance. The potential is -Umin at r0 and zero at rc. The spring constant is k = Umin / [2 (r0-rc)^2].

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- Umin (energy)

- r0 (distance)

- rc (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds, bond_harmonic

Default: none

bond_style hybrid command

Syntax:

```
bond_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more bond styles

Examples:

```
bond_style hybrid harmonic fene
bond_coeff 1 harmonic 80.0 1.2
bond_coeff 2* fene 30.0 1.5 1.0 1.0
```

Description:

The *hybrid* style enables the use of multiple bond styles in one simulation. A bond style is assigned to each bond type. For example, bonds in a polymer flow (of bond type 1) could be computed with a *fene* potential and bonds in the wall boundary (of bond type 2) could be computed with a *harmonic* potential. The assignment of bond type to style is made via the bond_coeff command or in the data file.

In the bond_coeff commands, the name of a bond style must be added after the bond type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 bond_coeff commands set bonds of bond type 1 to be computed with a *harmonic* potential with coefficients 80.0, 1.2 for K, r0. All other bond types (2-N) are computed with a *fene* potential with coefficients 30.0, 1.5, 1.0, 1.0 for K, R0, epsilon, sigma.

If bond coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "harmonic" or "fene" must be added after the bond type, for each line in the "Bond Coeffs" section, e.g.

```
Bond Coeffs

1 harmonic 80.0 1.2
2 fene 30.0 1.5 1.0 1.0
...
```

A bond style of *none* with no additional coefficients can be used in place of a bond style, either in a input script bond_coeff command or in the data file, if you desire to turn off interactions for specific bond types.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Unlike other bond styles, the hybrid bond style does not store bond coefficient info for individual sub-styles in a binary restart files. Thus when retarting a simulation from a restart file, you need to re-specify bond_coeff commands.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style morse command

bond_style morse/omp command

Syntax:

```
bond_style morse
```

Examples:

```
bond_style morse
bond_coeff 5 1.0 2.0 1.2
```

Description:

The *morse* bond style uses the potential

$$E = D \left[1 - e^{-\alpha(r - r_0)} \right]^2$$

where r0 is the equilibrium bond distance, alpha is a stiffness parameter, and D determines the depth of the potential well.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- D (energy)
- alpha (inverse distance)
- r0 (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[bond_coeff](), [delete_bonds]()

Default: none

bond_style none command

Syntax:

```
bond_style none
```

Examples:

```
bond_style none
```

Description:

Using a bond style of none means bond forces are not computed, even if pairs of bonded atoms were listed in the data file read by the [read_data](#) command.

Restrictions: none

Related commands: none

Default: none

bond_style nonlinear command

bond_style nonlinear/omp command

Syntax:

```
bond_style nonlinear
```

Examples:

```
bond_style nonlinear
bond_coeff 2 100.0 1.1 1.4
```

Description:

The *nonlinear* bond style uses the potential

$$E = \frac{\epsilon(r - r_0)^2}{\left[\lambda^2 - (r - r_0)^2\right]}$$

to define an anharmonic spring [(Rector)](#) of equilibrium length r0 and maximum extension lamda.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- epsilon (energy)
- r0 (distance)
- lamda (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

(Rector) Rector, Van Swol, Henderson, Molecular Physics, 82, 1009 (1994).

bond_style quartic command

bond_style quartic/omp command

Syntax:

```
bond_style quartic
```

Examples:

```
bond_style quartic
bond_coeff 2 1200 -0.55 0.25 1.3 34.6878
```

Description:

The *quartic* bond style uses the potential

$$E = K(r - R_c)^2(r - R_c - B_1)(r - R_c - B_2) + U_0 + 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] +$$

to define a bond that can be broken as the simulation proceeds (e.g. due to a polymer being stretched). The sigma and epsilon used in the LJ portion of the formula are both set equal to 1.0 by LAMMPS.

The following coefficients must be defined for each bond type via the [bond_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy/distance^4)
- B1 (distance)
- B2 (distance)
- Rc (distance)
- U0 (energy)

This potential was constructed to mimic the FENE bond potential for coarse-grained polymer chains. When monomers with sigma = epsilon = 1.0 are used, the following choice of parameters gives a quartic potential that looks nearly like the FENE potential: K = 1200, B1 = -0.55, B2 = 0.25, Rc = 1.3, and U0 = 34.6878. Different parameters can be specified using the [bond_coeff](#) command, but you will need to choose them carefully so they form a suitable bond potential.

Rc is the cutoff length at which the bond potential goes smoothly to a local maximum. If a bond length ever becomes > Rc, LAMMPS "breaks" the bond, which means two things. First, the bond potential is turned off by setting its type to 0, and is no longer computed. Second, a pairwise interaction between the two atoms is turned on, since they are no longer bonded.

LAMMPS does the second task via a computational sleight-of-hand. It subtracts the pairwise interaction as part of the bond computation. When the bond breaks, the subtraction stops. For this to work, the pairwise interaction must always be computed by the [pair_style](#) command, whether the bond is broken or not. This means that [special_bonds](#) must be set to 1,1,1, as indicated as a restriction below.

Note that when bonds are dumped to a file via the dump local command, bonds with type 0 are not included. The delete_bonds command can also be used to query the status of broken bonds or permanently delete them, e.g.:

```
delete_bonds all stats
delete_bonds all bond 0 remove
```

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

The *quartic* style requires that special_bonds parameters be set to 1,1,1. Three- and four-body interactions (angle, dihedral, etc) cannot be used with *quartic* bonds.

Related commands:

bond_coeff, delete_bonds

Default: none

bond_style command

Syntax:

```
bond_style style args
```

> • style = *none* or *hybrid* or *class2* or *fene* or *fene/expand* or *harmonic* or *morse* or *nonlinear* or *quartic*

```
  args = none for any style except hybrid
  hybrid args = list of one or more styles
```

Examples:

```
bond_style harmonic
bond_style fene
bond_style hybrid harmonic fene
```

Description:

Set the formula(s) LAMMPS uses to compute bond interactions between pairs of atoms. In LAMMPS, a bond differs from a pairwise interaction, which are set via the [pair_style](#) command. Bonds are defined between specified pairs of atoms and remain in force for the duration of the simulation (unless the bond breaks which is possible in some bond potentials). The list of bonded atoms is read in by a [read_data](#) or [read_restart](#) command from a data or restart file. By contrast, pair potentials are typically defined between all pairs of atoms within a cutoff distance and the set of active interactions changes over time.

Hybrid models where bonds are computed using different bond potentials can be setup using the *hybrid* bond style.

The coefficients associated with a bond style can be specified in a data or restart file or via the [bond_coeff](#) command.

All bond potentials store their coefficient data in binary restart files which means bond_style and [bond_coeff](#) commands do not need to be re-specified in an input script that restarts a simulation. See the [read_restart](#) command for details on how to do this. The one exception is that bond_style *hybrid* only stores the list of sub-styles in the restart file; bond coefficients need to be re-specified.

IMPORTANT NOTE: When both a bond and pair style is defined, the [special_bonds](#) command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 2 bonded atoms.

In the formulas listed for each bond style, *r* is the distance between the 2 atoms in the bond.

Here is an alphabetic list of bond styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated [bond_coeff](#) command.

Note that there are also additional bond styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the bond section of [this page](#).

- [bond_style none](#) - turn off bonded interactions
- [bond_style hybrid](#) - define multiple styles of bond interactions

- [bond_style class2](#) - COMPASS (class 2) bond
- [bond_style fene](#) - FENE (finite-extensible non-linear elastic) bond

- **bond_style fene/expand** - FENE bonds with variable size particles
- **bond_style harmonic** - harmonic bond
- **bond_style morse** - Morse bond
- **bond_style nonlinear** - nonlinear bond
- **bond_style quartic** - breakable quartic bond
- **bond_style table** - tabulated by bond length

Restrictions:

Bond styles can only be set for atom styles that allow bonds to be defined.

Most bond styles are part of the MOLECULAR package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual bond potentials tell if it is part of a package.

Related commands:

bond_coeff, delete_bonds

Default:

bond_style none

bond_style table command

bond_style table/omp command

Syntax:

```
bond_style table style N
```

- style = *linear* or *spline* = method of interpolation
- N = use N values in table

Examples:

```
bond_style table linear 1000
bond_coeff 1 file.table ENTRY1
```

Description:

Style *table* creates interpolation tables of length *N* from bond potential and force values listed in a file(s) as a function of bond length. The files are read by the bond_coeff command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and force values at each of *N* distances. During a simulation, these tables are used to interpolate energy and force values as needed. The interpolation is done in one of 2 styles: *linear* or *spline*.

For the *linear* style, the bond length is used to find 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The bond length is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

The following coefficients must be defined for each bond type via the bond_coeff command as in the example above.

- filename
- keyword

The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Bond potential for harmonic (one or more comment or blank lines)

HAM                            (keyword is the first text on line)
N 101 FP 0 0 EQ 0.5            (N, FP, EQ  parameters)
                               (blank line)
1 0.00 338.0000 1352.0000      (index, bond-length, energy, force)
2 0.01 324.6152 1324.9600
...
101 1.00 338.0000 -1352.0000
```

A section begins with a non-blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the bond_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the bond_style table command. Let Ntable = *N* in the bond_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and force values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual bond lengths. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile.

The "FP" parameter is optional. If used, it is followed by two values fplo and fphi, which are the derivatives of the force at the innermost and outermost bond lengths. These values are needed by the spline construction routines. If not specified by the "FP" parameter, they are estimated (less accurately) by the first two and last two force values in the table.

The "EQ" parameter is also optional. If used, it is followed by a the equilibrium bond length, which is used, for example, by the fix shake command. If not used, the equilibrium bond length is set to 0.0.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is the bond length r (in distance units), the 3rd value is the energy (in energy units), and the 4th is the force (in force units). The bond lengths must range from a LO value to a HI value, and increase from one line to the next. If the actual bond length is ever smaller than the LO value or larger than the HI value, then the bond energy and force is evaluated as if the bond were the LO or HI length.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This bond style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

bond_coeff, delete_bonds

Default: none

boundary command

Syntax:

```
boundary x y z
```

- x,y,z = *p* or *s* or *f* or *m*, one or two letters

  ```
  p is periodic
    f is non-periodic and fixed
    s is non-periodic and shrink-wrapped
    m is non-periodic and shrink-wrapped with a minimum value
  ```

Examples:

```
boundary p p f
boundary p fs p
boundary s f fm
```

Description:

Set the style of boundaries for the global simulation box in each dimension. A single letter assigns the same style to both the lower and upper face of the box. Two letters assigns the first style to the lower face and the second style to the upper face. The initial size of the simulation box is set by the read_data, read_restart, or create_box commands.

The style *p* means the box is periodic, so that particles interact across the boundary, and they can exit one end of the box and re-enter the other end. A periodic dimension can change in size due to constant pressure boundary conditions or box deformation (see the fix npt and fix deform commands). The *p* style must be applied to both faces of a dimension.

The styles *f*, *s*, and *m* mean the box is non-periodic, so that particles do not interact across the boundary and do not move from one side of the box to the other. For style *f*, the position of the face is fixed. If an atom moves outside the face it may be lost. For style *s*, the position of the face is set so as to encompass the atoms in that dimension (shrink-wrapping), no matter how far they move. For style *m*, shrink-wrapping occurs, but is bounded by the value specified in the data or restart file or set by the create_box command. For example, if the upper z face has a value of 50.0 in the data file, the face will always be positioned at 50.0 or above, even if the maximum z-extent of all the atoms becomes less than 50.0.

For triclinic (non-orthogonal) simulation boxes, if the 2nd dimension of a tilt factor (e.g. y for xy) is periodic, then the periodicity is enforced with the tilt factor offset. If the 1st dimension is shrink-wrapped, then the shrink wrapping is applied to the tilted box face, to encompass the atoms. E.g. for a positive xy tilt, the xlo and xhi faces of the box are planes tilting in the +y direction as y increases. These tilted planes are shrink-wrapped around the atoms to determine the x extent of the box.

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

IMPORTANT NOTE: If mesh walls (e.g. fix mesh/surface) are used, not only atom positions, but also the mesh nodes are used for setting the boundaries.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command or read_restart command. See the change_box command for how to change the simulation box boundaries after it has been defined.

For 2d simulations, the z dimension must be periodic.

Related commands:

See the thermo_modify command for a discussion of lost atoms.

Default:

```
boundary p p p
```

box command

Syntax:

```
box keyword value ...
```

- one or more keyword/value pairs may be appended
- keyword = *tilt*

  tilt value = *small* or *large*

Examples:

```
box tilt large
box tilt small
```

Description:

Set attributes of the simulation box.

For triclinic (non-orthogonal) simulation boxes, the *tilt* keyword allows simulation domains to be created with arbitrary tilt factors, e.g. via the create_box or read_data commands. Tilt factors determine how skewed the triclinic box is; see this section of the manual for a discussion of triclinic boxes in LAMMPS.

LAMMPS normally requires that no tilt factor can skew the box more than half the distance of the parallel box length, which is the 1st dimension in the tilt factor (x for xz). If *tilt* is set to *small*, which is the default, then an error will be generated if a box is created which exceeds this limit. If *tilt* is set to *large*, then no limit is enforced. You can create a box with any tilt factors you wish.

Note that if a simulation box has a large tilt factor, LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command or read_restart command.

Related commands: none

Default:

The default value is tilt = small.

change_box command

Syntax:

```
change_box group-ID parameter args ... keyword args ...
```

- group-ID = ID of group of atoms to (optionally) displace
- one or more parameter/arg pairs may be appended

```
  parameter = x or y or z or xy or xz or yz or boundary or ortho or triclinic or set or remap
    x, y, z args = style value(s)
      style = final or delta or scale or volume
        final values = lo hi
          lo hi = box boundaries after displacement (distance units)
        delta values = dlo dhi
          dlo dhi = change in box boundaries after displacement (distance units)
        scale values = factor
          factor = multiplicative factor for change in box length after displacement
        volume value = none = adjust this dim to preserve volume of system
    xy, xz, yz args = style value
      style = final or delta
        final value = tilt
          tilt = tilt factor after displacement (distance units)
        delta value = dtilt
          dtilt = change in tilt factor after displacement (distance units)
    boundary args = x y z
      x,y,z = p or s or f or m, one or two letters
      p is periodic
      f is non-periodic and fixed
      s is non-periodic and shrink-wrapped
      m is non-periodic and shrink-wrapped with a minimum value
    ortho args = none = change box to orthogonal
    triclinic args = none = change box to triclinic
    set args = none = store state of current box
    remap args = none = remap atom coords from last saved state to current box
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
  units value = lattice or box
      lattice = distances are defined in lattice units
      box = distances are defined in simulation box units
```

Examples:

```
change_box all xy final -2.0 z final 0.0 5.0 boundary p p f remap units box
change_box all x scale 1.1 y volume z volume remap
```

Description:

Change the volume and/or shape and/or boundary conditions for the simulation box. Orthogonal simulation boxes have 3 adjustable size parameters (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable size/shape parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently by this command. Thus it can be used to expand or contract a box, or to apply a shear strain to a non-orthogonal box. It can also be used to change the boundary conditions for the simulation box, similar to the boundary command.

The size and shape of the initial simulation box are specified by the create_box or read_data or read_restart command used to setup the simulation. The size and shape may be altered by subsequent runs, e.g. by use of the fix npt or fix deform commands. The create_box, read data, and read_restart commands also determine

whether the simulation box is orthogonal or triclinic and their doc pages explain the meaning of the xy,xz,yz tilt factors.

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

The keywords used in this command are applied sequentially to the simulation box and the atoms in it, in the order specified.

Before the sequence of keywords are invoked, the current box size/shape is stored, in case a *remap* keyword is used to map the atom coordinates from a previously stored box size/shape to the current one.

After all the keywords have been processed, any shrink-wrap boundary conditions are invoked (see the boundary command) which may change simulation box boundaries, and atoms are migrated to new owning processors.

IMPORTANT NOTE: Unlike the earlier "displace_box" version of this command, atom remapping is NOT performed by default. This command allows remapping to be done in a more general way, exactly when you specify it (zero or more times) in the sequence of transformations. Thus if you do not use the *remap* keyword, atom coordinates will not be changed even if the box size/shape changes. If a uniformly strained state is desired, the *remap* keyword should be specified.

IMPORTANT NOTE: It is possible to lose atoms with this command. E.g. by changing the box without remapping the atoms, and having atoms end up outside of non-periodic boundaries. It is also possible to alter bonds between atoms straddling a boundary in bad ways. E.g. by converting a boundary from periodic to non-periodic. It is also possible when remapping atoms to put them (nearly) on top of each other. E.g. by converting a boundary from non-periodic to periodic. All of these will typically lead to bad dynamics and/or generate error messages.

IMPORTANT NOTE: The simulation box size/shape can be changed by arbitrarily large amounts by this command. This is not a problem, except that the mapping of processors to the simulation box is not changed from its initial 3d configuration; see the processors command. Thus, if the box size/shape changes dramatically, the mapping of processors to the simulation box may not end up as optimal as the initial mapping attempted to be.

IMPORTANT NOTE: Because the keywords used in this command are applied one at a time to the simulation box and the atoms in it, care must be taken with triclinic cells to avoid exceeding the limits on skew after each transformation in the sequence. If skew is exceeded before the final transformation this can be avoided by changing the order of the sequence, or breaking the transformation into two or more smaller transformations. For more information on the allowed limits for box skew see the discussion on triclinic boxes on this page.

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

The *volume* style changes the specified dimension in such a way that the overall box volume remains constant

with respect to the operation performed by the preceding keyword. The *volume* style can only be used following a keyword that changed the volume, which is any of the *x*, *y*, *z* keywords. If the preceding keyword "key" had a *volume* style, then both it and the current keyword apply to the keyword preceding "key". I.e. this sequence of keywords is allowed:

```
change_box all x scale 1.1 y volume z volume
```

The *volume* style changes the associated dimension so that the overall box volume is unchanged relative to its value before the preceding keyword was invoked.

If the following command is used, then the z box length will shrink by the same 1.1 factor the x box length was increased by:

```
change_box all x scale 1.1 z volume
```

If the following command is used, then the y,z box lengths will each shrink by sqrt(1.1) to keep the volume constant. In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant:

```
change_box all"x scale 1.1 y volume z volume
```

If the following command is used, then the final box will be a factor of 10% larger in x and y, and a factor of 21% smaller in z, so as to keep the volume constant:

```
change_box all x scale 1.1 z volume y scale 1.1 z volume
```

IMPORTANT NOTE: For solids or liquids, when one dimension of the box is expanded, it may be physically undesirable to hold the other 2 box lengths constant since that implies a density change. For solids, adjusting the other dimensions via the *volume* style may make physical sense (just as for a liquid), but may not be correct for materials and potentials whose Poisson ratio is not 0.5.

For the *scale* and *volume* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

All of these styles change the xy, xz, yz tilt factors. In LAMMPS, tilt factors (xy,xz,yz) for triclinic boxes are required to be no more than half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all equivalent. Any tilt factor specified by this command must be within these limits.

The *boundary* keyword takes arguments that have exactly the same meaning as they do for the boundary command. In each dimension, a single letter assigns the same style to both the lower and upper face of the box. Two letters assigns the first style to the lower face and the second style to the upper face.

The style *p* means the box is periodic; the other styles mean non-periodic. For style *f*, the position of the face is fixed. For style *s*, the position of the face is set so as to encompass the atoms in that dimension (shrink-wrapping), no matter how far they move. For style *m*, shrink-wrapping occurs, but is bounded by the

current box edge in that dimension, so that the box will become no smaller. See the [boundary](#) command for more explanation of these style options.

Note that the "boundary" command itself can only be used before the simulation box is defined via a [read_data](#) or [create_box](#) or [read_restart](#) command. This command allows the boundary conditions to be changed later in your input script. Also note that the [read_restart](#) will change boundary conditions to match what is stored in the restart file. So if you wish to change them, you should use the change_box command after the read_restart command.

The *ortho* and *triclinic* keywords convert the simulation box to be orthogonal or triclinic (non-orthongonal). See [this section](#) for a discussion of how non-orthongal boxes are represented in LAMMPS.

The simulation box is defined as either orthogonal or triclinic when it is created via the [create_box](#), [read_data](#), or [read_restart](#) commands.

These keywords allow you to toggle the existing simulation box from orthogonal to triclinic and vice versa. For example, an initial equilibration simulation can be run in an orthogonal box, the box can be toggled to triclinic, and then a [non-equilibrium MD (NEMD) simulation](#) can be run with deformation via the [fix deform](#) command.

If the simulation box is currently triclinic and has non-zero tilt in xy, yz, or xz, then it cannot be converted to an orthogonal box.

The *set* keyword saves the current box size/shape. This can be useful if you wish to use the *remap* keyword more than once or if you wish it to be applied to an intermediate box size/shape in a sequence of keyword operations. Note that the box size/shape is saved before any of the keywords are processed, i.e. the box size/shape at the time the create_box command is encountered in the input script.

The *remap* keyword remaps atom coordinates from the last saved box size/shape to the current box state. For example, if you stretch the box in the x dimension or tilt it in the xy plane via the *x* and *xy* keywords, then the *remap* commmand will dilate or tilt the atoms to conform to the new box size/shape, as if the atoms moved with the box as it deformed.

Note that this operation is performed without regard to periodic boundaries. Also, any shrink-wrapping of non-periodic boundaries (see the [boundary](#) command) occurs after all keywords, including this one, have been processed.

Only atoms in the specified group are remapped.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing.

Restrictions:

If you use the *ortho* or *triclinic* keywords, then at the point in the input script when this command is issued, no [dumps](#) can be active, nor can a [fix ave/spatial](#) or [fix deform](#) be active. This is because these commands test whether the simulation box is orthogonal when they are first issued. Note that these commands can be used in your script before a change_box command is issued, so long as an [undump](#) or [unfix](#) command is also used to turn them off.

Related commands:

change_box command   78

fix deform, boundary

Default:

The option default is units = lattice.

clear command

Syntax:

```
clear
```

Examples:

```
(commands for 1st simulation)
clear
(commands for 2nd simulation)
```

Description:

This command deletes all atoms, restores all settings to their default values, and frees all memory allocated by LAMMPS. Once a clear command has been executed, it is as if LAMMPS were starting over, with only the exceptions noted below. This command enables multiple jobs to be run sequentially from one input script.

These settings are not affected by a clear command: the working directory (shell command), log file status (log command), echo status (echo command), and input script variables (variable command).

Restrictions: none

Related commands: none

Default: none

communicate command

Syntax:

```
communicate style keyword value ...
```

- style = *single* or *multi*
- zero or more keyword/value pairs may be appended
- keyword = *cutoff* or *group* or *vel*

```
  cutoff value = Rcut (distance units) = communicate atoms from this far away
    group value = group-ID = only communicate atoms in the group
      vel value = yes or no = do or do not communicate velocity info with ghost atoms
```

Examples:

```
communicate multi
communicate multi group solvent
communicate single vel yes
communicate single cutoff 5.0 vel yes
```

Description:

This command sets the style of inter-processor communication that occurs each timestep as atom coordinates and other properties are exchanged between neighboring processors and stored as properties of ghost atoms.

The default style is *single* which means each processor acquires information for ghost atoms that are within a single distance from its sub-domain. The distance is the maximum of the neighbor cutoff for all atom type pairs.

For many systems this is an efficient algorithm, but for systems with widely varying cutoffs for different type pairs, the *multi* style can be faster. In this case, each atom type is assigned its own distance cutoff for communication purposes, and fewer atoms will be communicated. However, for granular systems optimization is automatically performed with the *single* style, so *multi* is not necessary/available for granular systems. See the [neighbor multi](#) command for a neighbor list construction option that may also be beneficial for simulations of this kind.

The *cutoff* option allows you to set a ghost cutoff distance, which is the distance from the borders of a processor's sub-domain at which ghost atoms are acquired from other processors. By default the ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin. See the [neighbor](#) command for more information about the skin distance. If the specified Rcut is greater than the neighbor cutoff, then extra ghost atoms will be acquired. If it is smaller, the ghost cutoff is set to the neighbor cutoff.

These are simulation scenarios in which it may be useful or even necessary to set a ghost cutoff > neighbor cutoff:

- a single polymer chain with bond interactions, but no pairwise interactions
- bonded interactions (e.g. dihedrals) extend further than the pairwise cutoff
- ghost atoms beyond the pairwise cutoff are needed for some computation

In the first scenario, a pairwise potential is not defined. Thus the pairwise neighbor cutoff will be 0.0. But ghost atoms are still needed for computing bond, angle, etc interactions between atoms on different processors, or when the interaction straddles a periodic boundary.

The appropriate ghost cutoff depends on the newton bond setting. For newton bond *off*, the distance needs to be the furthest distance between any two atoms in the bond, angle, etc. E.g. the distance between 1-4 atoms in a dihedral. For newton bond *on*, the distance between the central atom in the bond, angle, etc and any other atom is sufficient. E.g. the distance between 2-4 atoms in a dihedral.

In the second scenario, a pairwise potential is defined, but its neighbor cutoff is not sufficiently long enough to enable bond, angle, etc terms to be computed. As in the previous scenario, an appropriate ghost cutoff should be set.

In the last scenario, a fix or compute or pairwise potential needs to calculate with ghost atoms beyond the normal pairwise cutoff for some computation it performs (e.g. locate neighbors of ghost atoms in a multibody pair potential). Setting the ghost cutoff appropriately can insure it will find the needed atoms.

IMPORTANT NOTE: In these scenarios, if you do not set the ghost cutoff long enough, and if there is only one processor in a periodic dimension (e.g. you are running in serial), then LAMMPS may "find" the atom it is looking for (e.g. the partner atom in a bond), that is on the far side of the simulation box, across a periodic boundary. This will typically lead to bad dynamics (i.e. the bond length is now the simulation box length). To detect if this is happening, see the neigh_modify cluster command.

The *group* option will limit communication to atoms in the specified group. This can be useful for models where no ghost atoms are needed for some kinds of particles. All atoms (not just those in the specified group) will still migrate to new processors as they move. The group specified with this option must also be specified via the atom_modify first command.

The *vel* option enables velocity information to be communicated with ghost particles. Depending on the atom_style, velocity info includes the translational velocity, angular velocity, and angular momentum of a particle. If the *vel* option is set to *yes*, then ghost atoms store these quantities; if *no* then they do not. The *yes* setting is needed by some pair styles which require the velocity state of both the I and J particles to compute a pairwise I,J interaction.

Note that if the fix deform command is being used with its "remap v" option enabled, then the velocities for ghost atoms (in the fix deform group) mirrored across a periodic boundary will also include components due to any velocity shift that occurs across that boundary (e.g. due to dilation or shear).

Restrictions: none

Related commands:

neighbor

Default:

The default settings are style = single, group = all, cutoff = 0.0, vel = no. The cutoff default of 0.0 means that ghost cutoff = neighbor cutoff = pairwise force cutoff + neighbor skin.

compute ackland/atom command

Syntax:

```
compute ID group-ID ackland/atom
```

- ID, group-ID are documented in compute command
- ackland/atom = style name of this compute command

Examples:

```
compute 1 all ackland/atom
```

Description:

Defines a computation that calculates the local lattice structure according to the formulation given in (Ackland).

In contrast to the centro-symmetry parameter this method is stable against temperature boost, because it is based not on the distance between particles but the angles. Therefore statistical fluctuations are averaged out a little more. A comparison with the Common Neighbor Analysis metric is made in the paper.

The result is a number which is mapped to the following different lattice structures:

- 0 = UNKNOWN
- 1 = BCC
- 2 = FCC
- 3 = HCP
- 4 = ICO

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of which computes this quantity.-

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

Restrictions:

This compute is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The per-atom vector values will be unitless since they are the integers defined above.

Related commands:

compute centro/atom

Default: none

(Ackland) Ackland, Jones, Phys Rev B, 73, 054104 (2006).

compute angle/local command

Syntax:

```
compute ID group-ID angle/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- angle/local = style name of this compute command
- one or more keywords may be appended
- keyword = *theta* or *eng*

```
  theta = tabulate angles
    eng = tabulate angle energies
```

Examples:

```
compute 1 all angle/local theta
compute 1 all angle/local eng theta
```

Description:

Define a computation that calculates properties of individual angle interactions. The number of datums generated, aggregated across all processors, equals the number of angles in the system, modified by the group parameter as explained below.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their angles. An angle will only be included if all 3 atoms in the angle are in the specified compute group. Any angles that have been broken (see the [angle_style](#) command) by setting their angle type to 0 are not included. Angles that have been turned off (see the [fix shake](#) or [delete_bonds](#) commands) by setting their angle type negative are written into the file, but their energy will be 0.0.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, angle output from the [compute property/local](#) command can be combined with data from this command and output by the [dump local](#) command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of angles. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The output for *theta* will be in degrees. The output for *eng* will be in energy [units](#).

Restrictions: none

Related commands:

[dump local](#), [compute property/local](#)

Default: none

compute atom/molecule command

Syntax:

```
compute ID group-ID atom/molecule input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- atom/molecule = style name of this compute command
- one or more inputs can be listed
- input = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    c_ID = per-atom vector calculated by a compute with ID
    c_ID[I] = Ith column of per-atom array calculated by a compute with ID
    f_ID = per-atom vector calculated by a fix with ID
    f_ID[I] = Ith column of per-atom array calculated by a fix with ID
    v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
compute 1 all atom/molecule c_ke c_pe
compute 1 top atom/molecule v_myFormula c_stress3
```

Description:

Define a calculation that sums per-atom values on a per-molecule basis, one per listed input. The inputs can be [computes](#), [fixes](#), or [variables](#) that generate per-atom quantities. Note that attributes stored by atoms, such as mass or force, can also be summed on a per-molecule basis, by accessing these quantities via the [compute property/atom](#) command.

Each listed input is operated on independently. Only atoms within the specified group contribute to the per-molecule sum. Note that compute or fix inputs define their own group which may affect the quantities they return. For example, if a compute is used as an input which generates a per-atom vector, it will generate values of 0.0 for atoms that are not in the group specified for that compute.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

If an input begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates per-atom quantities. See the individual [compute](#) doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LAMMPS](#).

If an input begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates per-atom quantities. See the individual [fix](#) doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute atom/molecule references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and [add them to LAMMPS](#).

If an input begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an [atom-style variable](#). Atom-style variables can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a

very general means of generating per-atom quantities to sum on a per-molecule basis.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LAMMPS output options.

All the vector or array values calculated by this compute are "extensive".

The vector or array values will be in whatever units the input quantities are in.

Restrictions: none

Related commands:

compute, fix, variable

Default: none

compute basal/atom command

Syntax:

```
compute ID group-ID basal/atom
```

- ID, group-ID are documented in compute command
- basal/atom = style name of this compute command

Examples:

```
compute 1 all basal/atom
```

Description:

Defines a computation that calculates the hexagonal close-packed "c" lattice vector for each atom in the group. It does this by calculating the normal unit vector to the basal plane for each atom. The results enable efficient identification and characterization of twins and grains in hexagonal close-packed structures.

The output of the compute is thus the 3 components of a unit vector associate with each atom. The components are set to 0.0 for atoms not in the group.

Details of the calculation are given in (Barrett).

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of which computes this quantity.

An example input script that uses this compute is provided in examples/USER/misc/basal.

Output info:

This compute calculates a per-atom array with 3 columns, which can be accessed by indices 1-3 by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom vector values are unitless since the 3 columns represent components of a unit vector.

Restrictions:

This compute is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The output of this compute will be meaningless unless the atoms are on (or near) hcp lattice sites, since the calculation assumes a well-defined basal plane.

Related commands:

compute centro/atom, compute ackland/atom

Default: none

(Barrett) Barrett, Tschopp, El Kadiri, Scripta Mat. 66, p.666 (2012).

compute body/local command

Syntax:

```
compute ID group-ID body/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- body/local = style name of this compute command
- one or more keywords may be appended
- keyword = *type* or *integer*

```
  type = atom type of the body particle
    integer = 1,2,3,etc = index of fields defined by body style
```

Examples:

```
compute 1 all body/local type 1 2 3
compute 1 all body/local 3 6
```

Description:

Define a computation that calculates properties of individual body sub-particles. The number of datums generated, aggregated across all processors, equals the number of body sub-particles plus the number of non-body particles in the system, modified by the group parameter as explained below. See [Section_howto 14](#) of the manual and the [body](#) doc page for more details on using body particles.

The local data stored by this command is generated by looping over all the atoms. An atom will only be included if it is in the group. If the atom is a body particle, then its N sub-particles will be looped over, and it will contribute N datums to the count of datums. If it is not a body particle, it will contribute 1 datum.

For both body particles and non-body particles, the *type* keyword will store the type of the atom.

The *integer* keywords mean different things for body and non-body particles. If the atom is not a body particle, only its *x*, *y*, *z* coordinates can be referenced, using the *integer* keywords 1,2,3. Note that this means that if you want to access more fields than this for body particles, then you cannot include non-body particles in the group.

For a body particle, the *integer* keywords refer to fields calculated by the body style for each sub-particle. The body style, as specified by the [atom_style body](#), determines how many fields exist and what they are. See the [body](#) doc page for details of the different styles.

Here is an example of how to output body information using the [dump local](#) command with this compute. If fields 1,2,3 for the body sub-particles are x,y,z coordinates, then the dump file will be formatted similar to the output of a [dump atom or custom](#) command.

```
compute 1 all body/local type 1 2 3
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_1[4]
```

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of datums as described above. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where

the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The units for output values depend on the body style.

Restrictions: none

Related commands:

dump local

Default: none

compute bond/local command

Syntax:

```
compute ID group-ID bond/local input1 input2 ...
```

- ID, group-ID are documented in compute command
- bond/local = style name of this compute command
- one or more keywords may be appended
- keyword = *dist* or *eng*

```
 dist = bond distance
  eng = bond energy
force = bond force
```

Examples:

```
compute 1 all bond/local eng
compute 1 all bond/local dist eng force
```

Description:

Define a computation that calculates properties of individual bond interactions. The number of datums generated, aggregated across all processors, equals the number of bonds in the system, modified by the group parameter as explained below.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their bonds. A bond will only be included if both atoms in the bond are in the specified compute group. Any bonds that have been broken (see the bond_style command) by setting their bond type to 0 are not included. Bonds that have been turned off (see the fix shake or delete_bonds commands) by setting their bond type negative are written into the file, but their energy will be 0.0.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, bond output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Here is an example of how to do this:

```
compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local dist eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_2[1] c_2[2]
```

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of bonds. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *dist* will be in distance units. The output for *eng* will be in energy units. The output for *force* will be in force units.

Restrictions: none

Related commands:

[dump local](), [compute property/local]()

Default: none

compute centro/atom command

Syntax:

```
compute ID group-ID centro/atom lattice
```

- ID, group-ID are documented in [compute](#) command
- centro/atom = style name of this compute command
- lattice = *fcc* or *bcc* or N = # of neighbors per atom to include

Examples:

```
compute 1 all centro/atom fcc

compute 1 all centro/atom 8
```

Description:

Define a computation that calculates the centro-symmetry parameter for each atom in the group. In solid-state systems the centro-symmetry parameter is a useful measure of the local lattice disorder around an atom and can be used to characterize whether the atom is part of a perfect lattice, a local defect (e.g. a dislocation or stacking fault), or at a surface.

The value of the centro-symmetry parameter will be 0.0 for atoms not in the specified compute group.

This parameter is computed using the following formula from [(Kelchner)](#)

$$CS = \sum_{i=1}^{N/2} |\vec{R}_i + \vec{R}_{i+N/2}|^2$$

where the *N* nearest neighbors or each atom are identified and Ri and Ri+N/2 are vectors from the central atom to a particular pair of nearest neighbors. There are N*(N-1)/2 possible neighbor pairs that can contribute to this formula. The quantity in the sum is computed for each, and the N/2 smallest are used. This will typically be for pairs of atoms in symmetrically opposite positions with respect to the central atom; hence the i+N/2 notation.

N is an input parameter, which should be set to correspond to the number of nearest neighbors in the underlying lattice of atoms. If the keyword *fcc* or *bcc* is used, *N* is set to 12 and 8 respectively. More generally, *N* can be set to a positive, even integer.

For an atom on a lattice site, surrounded by atoms on a perfect lattice, the centro-symmetry parameter will be 0. It will be near 0 for small thermal perturbations of a perfect lattice. If a point defect exists, the symmetry is broken, and the parameter will be a larger positive value. An atom at a surface will have a large positive parameter. If the atom does not have *N* neighbors (within the potential cutoff), then its centro-symmetry parameter is set to 0.0.

Only atoms within the cutoff of the pairwise neighbor list are considered as possible neighbors. Atoms not in the compute group are included in the *N* neighbors used in this calculation.

compute centro/atom command

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *centro/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom vector values are unitless values >= 0.0. Their magnitude depends on the lattice style due to the number of contibuting neighbor pairs in the summation in the formula above. And it depends on the local defects surrounding the central atom, as described above.

Here are typical centro-symmetry values, from a a nanoindentation simulation into gold (FCC). These were provided by Jon Zimmerman (Sandia):

```
Bulk lattice = 0
Dislocation core ~ 1.0 (0.5 to 1.25)
Stacking faults ~ 5.0 (4.0 to 6.0)
Free surface ~ 23.0
```

These values are *not* normalized by the square of the lattice parameter. If they were, normalized values would be:

```
Bulk lattice = 0
Dislocation core ~ 0.06 (0.03 to 0.075)
Stacking faults ~ 0.3 (0.24 to 0.36)
Free surface ~ 1.38
```

For BCC materials, the values for dislocation cores and free surfaces would be somewhat different, due to their being only 8 neighbors instead of 12.

Restrictions: none

Related commands:

compute cna/atom

Default: none

(Kelchner) Kelchner, Plimpton, Hamilton, Phys Rev B, 58, 11085 (1998).

compute cluster/atom command

Syntax:

```
compute ID group-ID cluster/atom cutoff
```

- ID, group-ID are documented in [compute](#) command
- cluster/atom = style name of this compute command
- cutoff = distance within which to label atoms as part of same cluster (distance units)

Examples:

```
compute 1 all cluster/atom 1.0
```

Description:

Define a computation that assigns each atom a cluster ID.

A cluster is defined as a set of atoms, each of which is within the cutoff distance from one or more other atoms in the cluster. If an atom has no neighbors within the cutoff distance, then it is a 1-atom cluster. The ID of every atom in the cluster will be the smallest atom ID of any atom in the cluster.

Only atoms in the compute group are clustered and assigned cluster IDs. Atoms not in the compute group are assigned a cluster ID = 0.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each of a *clsuter/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be an ID > 0, as explained above.

Restrictions: none

Related commands:

[compute coord/atom](#)

Default: none

compute cna/atom command

Syntax:

```
compute ID group-ID cna/atom cutoff
```

- ID, group-ID are documented in <u>compute</u> command
- cna/atom = style name of this compute command
- cutoff = cutoff distance for nearest neighbors (distance units)

Examples:

```
compute 1 all cna/atom 3.08
```

Description:

Define a computation that calculates the CNA (Common Neighbor Analysis) pattern for each atom in the group. In solid-state systems the CNA pattern is a useful measure of the local crystal structure around an atom. The CNA methodology is described in <u>(Faken)</u> and <u>(Tsuzuki)</u>.

Currently, there are five kinds of CNA patterns LAMMPS recognizes:

- fcc = 1
- hcp = 2
- bcc = 3
- icosohedral = 4
- unknown = 5

The value of the CNA pattern will be 0 for atoms not in the specified compute group. Note that normally a CNA calculation should only be performed on mono-component systems.

The CNA calculation can be sensitive to the specified cutoff value. You should insure the appropriate nearest neighbors of an atom are found within the cutoff distance for the presumed crystal strucure. E.g. 12 nearest neighbor for perfect FCC and HCP crystals, 14 nearest neighbors for perfect BCC crystals. These formulas can be used to obtain a good cutoff distance:

$$
\begin{aligned}
r_c^{fcc} &= \frac{1}{2}\left(\frac{\sqrt{2}}{2}+1\right)a \simeq 0.8536\,a \\
r_c^{bcc} &= \frac{1}{2}(\sqrt{2}+1)a \simeq 1.207\,a \\
r_c^{hcp} &= \frac{1}{2}\left(1+\sqrt{\frac{4+2x^2}{3}}\right)a
\end{aligned}
$$

where a is the lattice constant for the crystal structure concerned and in the HCP case, x = (c/a) / 1.633, where 1.633 is the ideal c/a for HCP crystals.

Also note that since the CNA calculation in LAMMPS uses the neighbors of an owned atom to find the nearest neighbors of a ghost atom, the following relation should also be satisfied:

$$Rc + Rs > 2 * \text{cutoff}$$

where Rc is the cutoff distance of the potential, Rs is the skin distance as specified by the [neighbor](#) command, and cutoff is the argument used with the compute cna/atom command. LAMMPS will issue a warning if this is not the case.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (e.g. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently or to have multiple compute/dump commands, each with a *cna/atom* style.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be a number from 0 to 5, as explained above.

Restrictions: none

Related commands:

[compute centro/atom](#)

Default: none

(Faken) Faken, Jonsson, Comput Mater Sci, 2, 279 (1994).

(Tsuzuki) Tsuzuki, Branicio, Rino, Comput Phys Comm, 177, 518 (2007).

compute com command

Syntax:

```
compute ID group-ID com
```

- ID, group-ID are documented in compute command
- com = style name of this compute command

Examples:

```
compute 1 all com
```

Description:

Define a computation that calculates the center-of-mass of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of three quantites is calculated by this compute, which are the x,y,z coordinates of the center of mass.

IMPORTANT NOTE: The coordinates of an atom contribute to the center-of-mass in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the center-of-mass may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the center-of-mass of rigid bodies as they cross periodic boundaries, you will need to post-process a dump file containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global vector of length 3, which can be accessed by indices 1-3 by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values are "intensive". The vector values will be in distance units.

Restrictions: none

Related commands:

compute com/molecule

Default: none

compute com/molecule command

Syntax:

```
compute ID group-ID com/molecule
```

- ID, group-ID are documented in compute command
- com/molecule = style name of this compute command

Examples:

```
compute 1 fluid com/molecule
```

Description:

Define a computation that calculates the center-of-mass of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

The x,y,z coordinates of the center-of-mass for a particular molecule are only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the center-of-mass calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to the molecule's center-of-mass in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the center-of-mass may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the center-of-mass of rigid bodies as they cross periodic boundaries, you will need to post-process a dump file containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 3 for the x,y,z center-of-mass coordinates of each molecule. These values can be accessed by any command that uses global array values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in distance units.

Restrictions: none

Related commands:

compute com

Default: none

compute contact/atom command

Syntax:

```
compute ID group-ID contact/atom
```

- ID, group-ID are documented in [compute](#) command
- contact/atom = style name of this compute command

Examples:

```
compute 1 all contact/atom
```

Description:

Define a computation that calculates the number of contacts for each atom in a group.

The contact number is defined for finite-size spherical particles as the number of neighbor atoms which overlap the central particle, meaning that their distance of separation is less than or equal to the sum of the radii of the two particles.

The value of the contact number will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, whose values can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be a number >= 0.0, as explained above.

Restrictions:

This compute requires that atoms store a radius as defined by the [atom_style sphere](#) command.

Related commands:

[compute coord/atom](#)

Default: none

compute coord/atom command

Syntax:

```
compute ID group-ID coord/atom cutoff keyword value
```

- ID, group-ID are documented in [compute](#) command
- coord/atom = style name of this compute command cutoff = distance within which to count coordination neighbors (distance units) zero or more keyword/value pairs may be appended to args
- keyword = *mix* or *type1*, *type2*, ...

```
mix value = yes or no -ID
    no = count all neighbors
    yes = count only neighbors that have same atom type

typeN = atom type for Nth coordination count (see asterisk form below)
```

Examples:

```
compute 1 all coord/atom 0.003 mix
compute 1 all coord/atom 2.0
compute 1 all coord/atom 6.0 1 2
compute 1 all coord/atom 6.0 2*4 5*8 *
```

Description:

Define a computation that calculates one or more coordination numbers for each atom in a group.

A coordination number is defined as the number of neighbor atoms with specified atom type(s) that are within the specified cutoff distance from the central atom. Atoms not in the group are included in a coordination number of atoms in the group.

The *typeN* keywords allow you to specify which atom types contribute to each coordination number. One coordination number is computed for each of the *typeN* keywords listed. If no *typeN* keywords are listed, a single coordination number is calculated, which includes atoms of all types (same as the "*" format, see below).

The *typeN* keywords can be specified in one of two ways. An explicit numeric value can be used, as in the 2nd example above. Or a wild-card asterisk can be used to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The value of all coordination numbers will be 0.0 for atoms not in the specified compute group.

The neighbor list needed to compute this quantity is constructed each time the calculation is performed (i.e. each time a snapshot of atoms is dumped). Thus it can be inefficient to compute/dump this quantity too frequently.

Keyword *mix* controlls if all neighbors are counted or if only neighbors with same atom type are counted. The latter can be useful to quanitfy mixture of different species.

IMPORTANT NOTE: If you have a bonded system, then the settings of [special_bonds](#) command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the

[special_bonds](#) command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the coordination count. One way to get around this, is to write a dump file, and use the [rerun](#) command to compute the coordination for snapshots in the dump file. The rerun script can use a [special_bonds](#) command that includes all pairs in the neighbor list.

Output info:

If single *type1* keyword is specified (or if none are specified), or the *mix* keyword is used, this compute calculates a per-atom vector. If multiple *typeN* keywords are specified, this compute calculates a per-atom array, with N columns. These values can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector or array values will be a number >= 0.0, as explained above.

Restrictions: none

Related commands:

[compute cluster/atom](#)

Default: none

compute damage/atom command

Syntax:

```
compute ID group-ID damage/atom
```

- ID, group-ID are documented in [compute](#) command
- damage/atom = style name of this compute command

Examples:

```
compute 1 all damage/atom
```

Description:

Define a computation that calculates the per-atom damage for each atom in a group. Please see the [PDLAMMPS user guide](#) for a formal definition of "damage" and more details about Peridynamics as it is implemented in LAMMPS.

The value of the damage will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be a number >= 0.0, as explained above.

Restrictions:

This compute is part of the PERI package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[dump custom](#)

Default: none

compute dihedral/local command

Syntax:

```
compute ID group-ID dihedral/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- dihedral/local = style name of this compute command
- one or more keywords may be appended
- keyword = *phi*

```
  phi = tabulate dihedral angles
```

Examples:

```
compute 1 all dihedral/local phi
```

Description:

Define a computation that calculates properties of individual dihedral interactions. The number of datums generated, aggregated across all processors, equals the number of angles in the system, modified by the group parameter as explained below.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their dihedrals. A dihedral will only be included if all 4 atoms in the dihedral are in the specified compute group.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, dihedral output from the [compute property/local](#) command can be combined with data from this command and output by the [dump local](#) command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of dihedrals. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The output for *phi* will be in degrees.

Restrictions: none

Related commands:

[dump local](#), [compute property/local](#)

Default: none

compute displace/atom command

Syntax:

```
compute ID group-ID displace/atom
```

- ID, group-ID are documented in compute command
- displace/atom = style name of this compute command

Examples:

```
compute 1 all displace/atom
```

Description:

Define a computation that calculates the current displacement of each atom in the group from its original coordinates, including all effects due to atoms passing thru periodic boundaries.

A vector of four quantites per atom is calculated by this compute. The first 3 elements of the vector are the dx,dy,dz displacements. The 4th component is the total displacement, i.e. sqrt(dx*dx + dy*dy + dz*dz).

The displacement of an atom is from its original position at the time the compute command was issued. The value of the displacement will be 0.0 for atoms not in the specified compute group.

IMPORTANT NOTE: Initial coordinates are stored in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and the computed displacement may not reflect its true displacement. See the fix rigid command for details. Thus, to compute the displacement of rigid bodies as they cross periodic boundaries, you will need to post-process a dump file containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a restart file, then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

This compute calculates a per-atom array with 4 columns, which can be accessed by indices 1-4 by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom array values will be in distance units.

Restrictions: none

Related commands:

compute msd, dump custom, fix store/state

Default: none

compute erotate/asphere command

Syntax:

```
compute ID group-ID erotate/asphere
```

- ID, group-ID are documented in compute command
- erotate/asphere = style name of this compute command

Examples:

```
compute 1 all erotate/asphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of aspherical particles. The aspherical particles can be ellipsoids, or line segments, or triangles. See the atom_style and read_data commands for descriptions of these options.

For all 3 types of particles, the rotational kinetic energy is computed as 1/2 I w^2, where I is the inertia tensor for the aspherical particle and w is its angular velocity, which is computed from its angular momentum if needed.

IMPORTANT NOTE: For 2d models, ellipsoidal particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions:

This compute requires that ellipsoidal particles atoms store a shape and quaternion orientation and angular momentum as defined by the atom_style ellipsoid command.

This compute requires that line segment particles atoms store a length and orientation and angular velocity as defined by the atom_style line command.

This compute requires that triangular particles atoms store a size and shape and quaternion orientation and angular momentum as defined by the atom_style tri command.

All particles in the group must be finite-size. They cannot be point particles.

Related commands: none

compute erotate/sphere

Default: none

compute erotate/sphere/atom command

Syntax:

```
compute ID group-ID erotate/sphere/atom
```

- ID, group-ID are documented in **compute** command
- erotate/sphere/atom = style name of this compute command

Examples:

```
compute 1 all erotate/sphere/atom
```

Description:

Define a computation that calculates the rotational kinetic energy for each particle in a group.

The rotational energy is computed as 1/2 I w^2, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For **2d models**, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

The value of the rotational kinetic energy will be 0.0 for atoms not in the specified compute group or for point particles with a radius = 0.0.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See **Section_howto 15** for an overview of LAMMPS output options.

The per-atom vector values will be in energy **units**.

Restrictions: none

Related commands:

dump custom

Default: none

compute erotate/sphere command

Syntax:

```
compute ID group-ID erotate/sphere
```

- ID, group-ID are documented in compute command
- erotate/sphere = style name of this compute command

Examples:

```
compute 1 all erotate/sphere
```

Description:

Define a computation that calculates the rotational kinetic energy of a group of spherical particles.

The rotational energy is computed as 1/2 I w^2, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For 2d models, particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions:

This compute requires that atoms store a radius and angular velocity (omega) as defined by the atom_style sphere command.

All particles in the group must be finite-size spheres or point particles. They cannot be aspherical. Point particles will not contribute to the rotational energy.

Related commands:

compute erotate/asphere

Default: none

compute event/displace command

Syntax:

```
compute ID group-ID event/displace threshold
```

- ID, group-ID are documented in [compute](#) command
- event/displace = style name of this compute command
- threshold = minimum distance anyparticle must move to trigger an event (distance units)

Examples:

```
compute 1 all event/displace 0.5
```

Description:

Define a computation that flags an "event" if any particle in the group has moved a distance greater than the specified threshold distance when compared to a previously stored reference state (i.e. the previous event). This compute is typically used in conjunction with the [prd](#) and [tad](#) commands, to detect if a transition to a new minimum energy basin has occurred.

This value calculated by the compute is equal to 0 if no particle has moved far enough, and equal to 1 if one or more particles have moved further than the threshold distance.

NOTE: If the system is undergoing significant center-of-mass motion, due to thermal motion, an external force, or an initial net momentum, then this compute will not be able to distinguish that motion from local atom displacements and may generate "false postives."

Output info:

This compute calculates a global scalar (the flag). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The scalar value will be a 0 or 1 as explained above.

Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[prd](#), [tad](#)

Default: none

compute group/group command

Syntax:

```
compute ID group-ID group/group group2-ID keyword value ...
```

- ID, group-ID are documented in [compute](#) command
- group/group = style name of this compute command
- group2-ID = group ID of second (or same) group
- zero or more keyword/value pairs may be appended
- keyword = *pair* or *kspace* or *boundary*

```
  pair value = yes or no
    kspace value = yes or no
    boundary value = yes or no
```

Examples:

```
compute 1 lower group/group upper
compute 1 lower group/group upper kspace yes
compute mine fluid group/group wall
```

Description:

Define a computation that calculates the total energy and force interaction between two groups of atoms: the compute group and the specified group2. The two groups can be the same.

If the *pair* keyword is set to *yes*, which is the default, then the the interaction energy will include a pair component which is defined as the pairwise energy between all pairs of atoms where one atom in the pair is in the first group and the other is in the second group. Likewise, the interaction force calculated by this compute will include the force on the compute group atoms due to pairwise interactions with atoms in the specified group2.

If the *kspace* keyword is set to *yes*, which is not the default, and if a [kspace_style](#) is defined, then the interaction energy will include a Kspace component which is the long-range Coulombic energy between all the atoms in the first group and all the atoms in the 2nd group. Likewise, the interaction force calculated by this compute will include the force on the compute group atoms due to long-range Coulombic interactions with atoms in the specified group2.

Normally the long-range Coulombic energy converges only when the net charge of the unit cell is zero. However, one can assume the net charge of the system is neutralized by a uniform background plasma, and a correction to the system energy can be applied to reduce artifacts. For more information see [(Bogusz)](#). If the *boundary* keyword is set to *yes*, which is the default, and *kspace* contributions are included, then this energy correction term will be added to the total group-group energy. This correction term does not affect the force calculation and will be zero if one or both of the groups are charge neutral. This energy correction term is the same as that included in the regular Ewald and PPPM routines.

This compute does not calculate any bond or angle or dihedral or improper interactions between atoms in the two groups.

The pairwise contributions to the group-group interactions are calculated by looping over a neighbor list. The Kspace contribution to the group-group interactions require essentially the same amount of work (FFTs, Ewald summation) as computing long-range forces for the entire system. Thus it can be costly to invoke this

compute too frequently.

If you desire a breakdown of the interactions into a pairwise and Kspace component, simply invoke the compute twice with the appropriate yes/no settings for the *pair* and *kspace* keywords. This is no more costly than using a single compute with both keywords set to *yes*. The individual contributions can be summed in a [variable](#) if desired.

This [document](#) describes how the long-range group-group calculations are performed.

Output info:

This compute calculates a global scalar (the energy) and a global vector of length 3 (force), which can be accessed by indices 1-3. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

Both the scalar and vector values calculated by this compute are "extensive". The scalar value will be in energy [units](#). The vector values will be in force [units](#).

Restrictions:

Not all pair styles can be evaluated in a pairwise mode as required by this compute. For example, 3-body and other many-body potentials, such as [Tersoff](#) and [Stillinger-Weber](#) cannot be used. [EAM](#) potentials only include the pair potential portion of the EAM interaction when used by this compute, not the embedding term.

Not all Kspace styles support calculation of group/group interactions. The *ewald* and *pppm* styles do.

Related commands: none

Default:

The option defaults are pair = yes, kspace = no, and boundary = yes.

Bogusz et al, J Chem Phys, 108, 7070 (1998)

compute gyration command

Syntax:

```
compute ID group-ID gyration
```

- ID, group-ID are documented in **compute** command
- gyration = style name of this compute command

Examples:

```
compute 1 molecule gyration
```

Description:

Define a computation that calculates the radius of gyration Rg of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of the group of atoms, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the group, Rcm is the center-of-mass position of the group, and the sum is over all atoms in the group.

A Rg tensor, stored as a 6-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that (Ri - Rcm)^2 is replaced by (Rix - Rcmx) * (Riy - Rcmy) for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the **dump custom** command for a discussion of "unwrapped" coordinates. See the Atoms section of the **read_data** command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the **set image** command.

Output info:

This compute calculates a global scalar (Rg) and a global vector of length 6 (Rg tensor), which can be accessed by indices 1-6. These values can be used by any command that uses a global scalar value or vector values from a compute as input. See **Section_howto 15** for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in distance **units**.

Restrictions: none

Related commands:

compute gyration/molecule

Default: none

compute gyration/molecule command

Syntax:

```
compute ID group-ID gyration/molecule keyword value ...
```

- ID, group-ID are documented in compute command
- gyration/molecule = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *tensor*

```
  tensor value = none
```

Examples:

```
compute 1 molecule gyration/molecule
compute 2 molecule gyration/molecule tensor
```

Description:

Define a computation that calculates the radius of gyration Rg of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Rg is a measure of the size of a molecule, and is computed by this formula

$$R_g{}^2 = \frac{1}{M} \sum_i m_i (r_i - r_{cm})^2$$

where M is the total mass of the molecule, Rcm is the center-of-mass position of the molecule, and the sum is over all atoms in the molecule and in the group.

If the *tensor* keyword is specified, then the scalar Rg value is not calculated, but an Rg tensor is instead calculated for each molecule. The formula for the components of the tensor is the same as the above formula, except that (Ri - Rcm)^2 is replaced by (Rix - Rcmx) * (Riy - Rcmy) for the xy component, etc. The 6 components of the tensor are ordered xx, yy, zz, xy, xz, yz.

Rg for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the Rg calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to Rg in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

Output info:

This compute calculates a global vector if the *tensor* keyword is not specified and a global array if it is. The length of the vector or number of rows in the array is the number of molecules. If the *tensor* keyword is specified, the global array has 6 columns. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LAMMPS output options.

All the vector or array values calculated by this compute are "intensive". The vector or array values will be in distance units.

Restrictions: none

Related commands: none

compute gyration

Default: none

compute heat/flux command

Syntax:

```
compute ID group-ID heat/flux ke-ID pe-ID stress-ID
```

- ID, group-ID are documented in compute command
- heat/flux = style name of this compute command
- ke-ID = ID of a compute that calculates per-atom kinetic energy
- pe-ID = ID of a compute that calculates per-atom potential energy
- stress-ID = ID of a compute that calculates per-atom stress

Examples:

```
compute myFlux all heat/flux myKE myPE myStress
```

Description:

Define a computation that calculates the heat flux vector based on contributions from atoms in the specified group. This can be used by itself to measure the heat flux into or out of a reservoir of atoms, or to calculate a thermal conductivity using the Green-Kubo formalism.

See the fix thermal/conductivity command for details on how to compute thermal conductivity in an alternate way, via the Muller-Plathe method. See the fix heat command for a way to control the heat added or subtracted to a group of atoms.

The compute takes three arguments which are IDs of other computes. One calculates per-atom kinetic energy (*ke-ID*), one calculates per-atom potential energy (*pe-ID*), and the third calcualtes per-atom stress (*stress-ID*). These should be defined for the same group used by compute heat/flux, though LAMMPS does not check for this.

The Green-Kubo formulas relate the ensemble average of the auto-correlation of the heat flux J to the thermal conductivity kappa:

$$
\begin{aligned}
\mathbf{J} &= \frac{1}{V}\left[\sum_i e_i \mathbf{v}_i - \sum_i \mathbf{S}_i \mathbf{v}_i\right] \\
&= \frac{1}{V}\left[\sum_i e_i \mathbf{v}_i + \sum_{i<j}(\mathbf{f}_{ij}\cdot\mathbf{v}_j)\,\mathbf{x}_{ij}\right] \\
&= \frac{1}{V}\left[\sum_i e_i \mathbf{v}_i + \frac{1}{2}\sum_{i<j}(\mathbf{f}_{ij}\cdot(\mathbf{v}_i+\mathbf{v}_j))\,\mathbf{x}_{ij}\right]
\end{aligned}
$$

$$\kappa = \frac{V}{k_B T^2} \int_0^\infty \langle J_x(0) J_x(t) \rangle \, dt = \frac{V}{3k_B T^2} \int_0^\infty \langle \mathbf{J}(0) \cdot \mathbf{J}(t) \rangle \, dt$$

Ei in the first term of the equation for J is the per-atom energy (potential and kinetic). This is calculated by the computes *ke-ID* and *pe-ID*. Si in the second term of the equation for J is the per-atom stress tensor calculated by the compute *stress-ID*. The tensor multiplies Vi as a 3x3 matrix-vector multiply to yield a vector. Note that as discussed below, the 1/V scaling factor in the equation for J is NOT included in the calculation performed by this compute; you need to add it for a volume appropriate to the atoms included in the calculation.

IMPORTANT NOTE: The compute pe/atom and compute stress/atom commands have options for which terms to include in their calculation (pair, bond, etc). The heat flux calculation will thus include exactly the same terms. Normally you should use compute stress/atom virial so as not to include a kinetic energy term in the heat flux.

This compute calculates 6 quantities and stores them in a 6-component vector. The first 3 components are the x, y, z components of the full heat flux vector, i.e. (Jx, Jy, Jz). The next 3 components are the x, y, z components of just the convective portion of the flux, i.e. the first term in the equation for J above.

The heat flux can be output every so many timesteps (e.g. via the thermo_style custom command). Then as a post-processing operation, an autocorrelation can be performed, its integral estimated, and the Green-Kubo formula above evaluated.

The fix ave/correlate command can calclate the autocorrelation. The trap() function in the variable command can calculate the integral.

An example LAMMPS input script for solid Ar is appended below. The result should be: average conductivity ~0.29 in W/mK.

Output info:

This compute calculates a global vector of length 6 (total heat flux vector, followed by conductive heat flux vector), which can be accessed by indices 1-6. These values can be used by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values calculated by this compute are "extensive", meaning they scale with the number of atoms in the simulation. They can be divided by the appropriate volume to get a flux, which would then be an "intensive" value, meaning independent of the number of atoms in the simulation. Note that if the compute is "all", then the appropriate volume to divide by is the simulation box volume. However, if a sub-group is used, it should be the volume containing those atoms.

The vector values will be in energy*velocity units. Once divided by a volume the units will be that of flux, namely energy/area/time units

Restrictions: none

Related commands:

fix thermal/conductivity, fix ave/correlate, variable

Default: none

```
# Sample LAMMPS input script for thermal conductivity of solid Ar


units       real
variable    T equal 70
variable    V equal vol
variable    dt equal 4.0
variable    p equal 200     # correlation length
variable    s equal 10      # sample interval
variable    d equal $p*$s   # dump interval


# convert from LAMMPS real units to SI


variable    kB equal 1.3806504e-23    # [J/K] Boltzmann
variable    kCal2J equal 4186.0/6.02214e23
variable    A2m equal 1.0e-10
variable    fs2s equal 1.0e-15
variable    convert equal ${kCal2J}*${kCal2J}/${fs2s}/${A2m}


# setup problem


dimension   3
boundary    p p p
lattice     fcc 5.376 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
region      box block 0 4 0 4 0 4
create_box  1 box
create_atoms 1 box
mass        1 39.948
pair_style  lj/cut 13.0
pair_coeff  * * 0.2381 3.405
timestep    ${dt}
thermo      $d


# equilibration and thermalization


velocity    all create $T 102486 mom yes rot yes dist gaussian
fix         NVT all nvt temp $T $T 10 drag 0.2
run         8000


# thermal conductivity calculation, switch to NVE if desired


#unfix      NVT
#fix        NVE all nve


reset_timestep 0
compute     myKE all ke/atom
compute     myPE all pe/atom
compute     myStress all stress/atom virial
compute     flux all heat/flux myKE myPE myStress
variable    Jx equal c_flux[1]/vol
variable    Jy equal c_flux[2]/vol
variable    Jz equal c_flux[3]/vol
fix         JJ all ave/correlate $s $p $d &
            c_flux[1] c_flux[2] c_flux[3] type auto file J0Jt.dat ave running
variable    scale equal ${convert}/${kB}/$T/$T/$V*$s*${dt}
variable    k11 equal trap(f_JJ[3])*${scale}
variable    k22 equal trap(f_JJ[4])*${scale}
variable    k33 equal trap(f_JJ[5])*${scale}
thermo_style custom step temp v_Jx v_Jy v_Jz v_k11 v_k22 v_k33
run         100000
variable    k equal (v_k11+v_k22+v_k33)/3.0
variable    ndens equal count(all)/vol
print       "average conductivity: $k[W/mK] @ $T K, ${ndens} /A^3"
```

compute heat/flux command   122

compute command

Syntax:

```
compute ID group-ID style args
```

- ID = user-assigned name for the computation
- group-ID = ID of the group of atoms to perform the computation on
- style = one of a list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
compute 1 all temp
compute newtemp flow temp/partial 1 1 0
compute 3 all ke/atom
```

Description:

Define a computation that will be performed on a group of atoms. Quantities calculated by a compute are instantaneous values, meaning they are calculated from information about atoms on the current timestep or iteration, though a compute may internally store some information about a previous state of the system. Defining a compute does not perform a computation. Instead computes are invoked by other LAMMPS commands as needed, e.g. to calculate a temperature needed for a thermostat fix or to generate thermodynamic or dump file output. See this [howto section](#) for a summary of various LAMMPS output options, many of which involve computes.

The ID of a compute can only contain alphanumeric characters and underscores.

Computes calculate one of three styles of quantities: global, per-atom, or local. A global quantity is one or more system-wide values, e.g. the temperature of the system. A per-atom quantity is one or more values per atom, e.g. the kinetic energy of each atom. Per-atom values are set to 0.0 for atoms not in the specified compute group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atom, e.g. a list of bond distances. Computes that produce per-atom quantities have the word "atom" in their style, e.g. *ke/atom*. Computes that produce local quantities have the word "local" in their style, e.g. *bond/local*. Styles with neither "atom" or "local" in their style produce global quantities.

Note that a single compute produces either global or per-atom or local quantities, but never more than one of these.

Global, per-atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each compute describes the style and kind of values it produces, e.g. a per-atom vector. Some computes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a compute quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the compute:

c_ID	entire scalar, vector, or array
c_ID[I]	one element of vector, one column of array
c_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the quantity once (vector -> scalar, array -> vector). Using two brackets reduces the dimension twice (array -> scalar). Thus a command that uses scalar compute values as input can also process elements of a vector or array.

Note that commands and variables which use compute quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a compute quantity as c_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LAMMPS, the values generated by a compute can be used in several ways:

- The results of computes that calculate a global temperature or pressure can be used by fixes that do thermostatting or barostatting or when atom velocities are created.
- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per-atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time-averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per-atom values can be referenced in an atom-style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command, or output by the dump local command.

The results of computes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a compute value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual computes for further info.

LAMMPS creates its own computes internally for thermodynamic output. Three computes are always created, named "thermo_temp", "thermo_press", and "thermo_pe", as if these commands had been invoked in the input script:

```
compute thermo_temp all temp
compute thermo_press all pressure thermo_temp
compute thermo_pe all pe
```

Additional computes for other quantities are created if the thermo style requires it. See the documentation for the thermo_style command.

Fixes that calculate temperature or pressure, i.e. for thermostatting or barostatting, may also create computes. These are discussed in the documentation for specific fix commands.

In all these cases, the default computes LAMMPS creates can be replaced by computes defined by the user in the input script, as described by the thermo_modify and fix modify commands.

Properties of either a default or user-defined compute can be modified via the compute_modify command.

Computes can be deleted with the uncompute command.

Code for new computes can be added to LAMMPS (see this section of the manual) and the results of their calculations accessed in the various ways described above.

Each compute style has its own doc page which describes its arguments and what it does. Here is an alphabetic list of compute styles available in LAMMPS:

- [angle/local](#) - theta and energy of each angle
- [atom/molecule](#) - sum per-atom properties for each molecule
- [body/local](#) - attributes of body sub-particles
- [bond/local](#) - distance and energy of each bond
- [centro/atom](#) - centro-symmetry parameter for each atom
- [cluster/atom](#) - cluster ID for each atom
- [cna/atom](#) - common neighbor analysis (CNA) for each atom
- [com](#) - center-of-mass of group of atoms
- [com/molecule](#) - center-of-mass for each molecule
- [contact/atom](#) - contact count for each spherical particle
- [coord/atom](#) - coordination number for each atom
- [damage/atom](#) - Peridynamic damage for each atom
- [dihedral/local](#) - angle of each dihedral
- [displace/atom](#) - displacement of each atom
- [erotate/asphere](#) - rotational energy of aspherical particles
- [erotate/rigid](#) - rotational energy of rigid bodies
- [erotate/sphere](#) - rotational energy of spherical particles
- [erotate/sphere/atom](#) - rotational energy for each spherical particle
- [event/displace](#) - detect event on atom displacement
- [group/group](#) - energy/force between two groups of atoms
- [gyration](#) - radius of gyration of group of atoms
- [gyration/molecule](#) - radius of gyration for each molecule
- [heat/flux](#) - heat flux through a group of atoms
- [improper/local](#) - angle of each improper
- [inertia/molecule](#) - inertia tensor for each molecule
- [ke](#) - translational kinetic energy
- [ke/atom](#) - kinetic energy for each atom
- [ke/rigid](#) - translational kinetic energy of rigid bodies
- [msd](#) - mean-squared displacement of group of atoms
- [msd/molecule](#) - mean-squared displacement for each molecule
- [pair](#) - values computed by a pair style
- [pair/local](#) - distance/energy/force of each pairwise interaction
- [pe](#) - potential energy
- [pe/atom](#) - potential energy for each atom
- [pressure](#) - total pressure and pressure tensor
- [property/atom](#) - convert atom attributes to per-atom vectors/arrays
- [property/local](#) - convert local attributes to localvectors/arrays
- [property/molecule](#) - convert molecule attributes to localvectors/arrays
- [rdf](#) - radial distribution function g(r) histogram of group of atoms
- [reduce](#) - combine per-atom quantities into a single global value
- [reduce/region](#) - same as compute reduce, within a region
- [slice](#) - extract values from global vector or array
- [stress/atom](#) - stress tensor for each atom
- [temp](#) - temperature of group of atoms
- [temp/asphere](#) - temperature of aspherical particles
- [temp/com](#) - temperature after subtracting center-of-mass velocity
- [temp/deform](#) - temperature excluding box deformation velocity
- [temp/partial](#) - temperature excluding one or more dimensions of velocity
- [temp/profile](#) - temperature excluding a binned velocity profile
- [temp/ramp](#) - temperature excluding ramped velocity component
- [temp/region](#) - temperature of a region of atoms
- [temp/sphere](#) - temperature of spherical particles
- [ti](#) - thermodyanmic integration free energy values
- [voronoi/atom](#) - Voronoi volume and neighbors for each atom

There are also additional compute styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the compute section of this page.

There are also additional accelerated compute styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions: none

Related commands:

uncompute, compute_modify, fix ave/atom, fix ave/spatial, fix ave/time, fix ave/histo

Default: none

compute improper/local command

Syntax:

```
compute ID group-ID improper/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- improper/local = style name of this compute command
- one or more keywords may be appended
- keyword = *chi*

```
  chi = tabulate improper angles
```

Examples:

```
compute 1 all improper/local chi
```

Description:

Define a computation that calculates properties of individual improper interactions. The number of datums generated, aggregated across all processors, equals the number of impropers in the system, modified by the group parameter as explained below.

The local data stored by this command is generated by looping over all the atoms owned on a processor and their impropers. An improper will only be included if all 4 atoms in the improper are in the specified compute group.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, improper output from the [compute property/local](#) command can be combined with data from this command and output by the [dump local](#) command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of impropers. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The output for *chi* will be in degrees.

Restrictions: none

Related commands:

[dump local](#), [compute property/local](#)

Default: none

compute inertia/molecule command

Syntax:

```
compute ID group-ID inertia/molecule
```

- ID, group-ID are documented in compute command
- inertia/molecule = style name of this compute command

Examples:

```
compute 1 fluid inertia/molecule
```

Description:

Define a computation that calculates the inertia tensor of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

The symmetric intertia tensor has 6 components, ordered Ixx,Iyy,Izz,Ixy,Iyz,Ixz. The tensor for a particular molecule is only computed if one or more of its atoms is in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the inertia tensor and associated center-of-mass calculation for the molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, the molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The coordinates of an atom contribute to the molecule's inertia tensor in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the inertia tensor may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the inertia tensor of rigid bodies as they cross periodic boundaries, you will need to post-process a dump file containing coordinates of the atoms in the bodies.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 6 for the 6 components of the inertia tensor of each molecule, ordered as listed above. These values can be accessed by any command that uses global array values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in distance units.

Restrictions: none

Related commands:

variable inertia() function

Default: none

compute ke/atom/eff command

Syntax:

```
compute ID group-ID ke/atom/eff
```

- ID, group-ID are documented in [compute](#) command
- ke/atom/eff = style name of this compute command

Examples:

```
compute 1 all ke/atom/eff
```

Description:

Define a computation that calculates the per-atom translational (nuclei and electrons) and radial kinetic energy (electron only) in a group. The particles are assumed to be nuclei and electrons modeled with the [electronic force field](#).

The kinetic energy for each nucleus is computed as $1/2\ m\ v^2$, where m corresponds to the corresponding nuclear mass, and the kinetic energy for each electron is computed as $1/2\ (m_e\ v^2 + 3/4\ m_e\ s^2)$, where m_e and v correspond to the mass and translational velocity of each electron, and s to its radial velocity, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the [thermo_style](#) command. For this compute, kinetic energy is "translational" plus electronic "radial" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with $1/2\ K_b\ T$ of energy for each (nuclear-only) degree of freedom in eFF.

IMPORTANT NOTE: The temperature in eFF should be monitored via the [compute temp/eff](#) command, which can be printed with thermodynamic output by using the [thermo_modify](#) command, as shown in the following example:

```
compute         effTemp all temp/eff
thermo_style    custom step etotal pe ke temp press
thermo_modify   temp effTemp
```

The value of the kinetic energy will be 0.0 for atoms (nuclei or electrons) not in the specified compute group.

Output info:

This compute calculates a scalar quantity for each atom, which can be accessed by any command that uses per-atom computes as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be in energy [units](#).

Restrictions:

This compute is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

compute ke/atom/eff command 130

[dump custom](dump_custom)

Default: none

compute ke/atom command

Syntax:

```
compute ID group-ID ke/atom
```

- ID, group-ID are documented in [compute](#) command
- ke/atom = style name of this compute command

Examples:

```
compute 1 all ke/atom
```

Description:

Define a computation that calculates the per-atom translational kinetic energy for each atom in a group.

The kinetic energy is simply 1/2 m v^2, where m is the mass and v is the velocity of each atom.

The value of the kinetic energy will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be in energy [units](#).

Restrictions: none

Related commands:

[dump custom](#)

Default: none

compute ke/eff command

Syntax:

```
compute ID group-ID ke/eff
```

- ID, group-ID are documented in [compute](#) command
- ke/eff = style name of this compute command

Examples:

```
compute 1 all ke/eff
```

Description:

Define a computation that calculates the kinetic energy of motion of a group of eFF particles (nuclei and electrons), as modeled with the [electronic force field](#).

The kinetic energy for each nucleus is computed as $1/2\ m\ v^2$ and the kinetic energy for each electron is computed as $1/2(m_e\ v^2 + 3/4\ m_e\ s^2)$, where m corresponds to the nuclear mass, m_e to the electron mass, v to the translational velocity of each particle, and s to the radial velocity of the electron, respectively.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the [thermo_style](#) command. For this compute, kinetic energy is "translational" and "radial" (only for electrons) kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with $1/2\ K_b\ T$ of energy for each degree of freedom. For the eFF temperature computation via the [compute temp_eff](#) command, these are the same. But different computes that calculate temperature can subtract out different non-thermal components of velocity and/or include other degrees of freedom.

IMPORTANT NOTE: The temperature in eFF models should be monitored via the [compute temp/eff](#) command, which can be printed with thermodynamic output by using the [thermo_modify](#) command, as shown in the following example:

```
compute        effTemp all temp/eff
thermo_style   custom step etotal pe ke temp press
thermo_modify  temp effTemp
```

See [compute temp/eff](#).

Output info:

This compute calculates a global scalar (the KE). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions:

This compute is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands: none

Default: none

compute ke command

Syntax:

```
compute ID group-ID ke
```

- ID, group-ID are documented in [compute](#) command
- ke = style name of this compute command

Examples:

```
compute 1 all ke
```

Description:

Define a computation that calculates the translational kinetic energy of a group of particles.

The kinetic energy of each particle is computed as 1/2 m v^2, where m and v are the mass and velocity of the particle.

There is a subtle difference between the quantity calculated by this compute and the kinetic energy calculated by the *ke* or *etotal* keyword used in thermodynamic output, as specified by the [thermo_style](#) command. For this compute, kinetic energy is "translational" kinetic energy, calculated by the simple formula above. For thermodynamic output, the *ke* keyword infers kinetic energy from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. For the default temperature computation via the [compute temp](#) command, these are the same. But different computes that calculate temperature can subtract out different non-thermal components of velocity and/or include different degrees of freedom (translational, rotational, etc).

Output info:

This compute calculates a global scalar (the summed KE). This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy [units](#).

Restrictions: none

Related commands:

[compute erotate/sphere](#)

Default: none

compute ke command

compute meso_e/atom command

Syntax:

```
compute ID group-ID meso_e/atom
```

- ID, group-ID are documented in compute command
- meso_e/atom = style name of this compute command

Examples:

```
compute 1 all meso_e/atom
```

Description:

Define a computation that calculates the per-atom internal energy for each atom in a group.

The internal energy is the energy associated with the internal degrees of freedom of a mesoscopic particles, e.g. a Smooth-Particle Hydrodynamics particle.

See this PDF guide to using SPH in LAMMPS.

The value of the internal energy will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom vector values will be in energy units.

Restrictions:

This compute is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

dump custom

Default: none

compute meso_rho/atom command

Syntax:

```
compute ID group-ID meso_rho/atom
```

- ID, group-ID are documented in **compute** command
- meso_rho/atom = style name of this compute command

Examples:

```
compute 1 all meso_rho/atom
```

Description:

Define a computation that calculates the per-atom mesoscopic density for each atom in a group.

The mesoscopic density is the mass density of a mesoscopic particle, calculated by kernel function interpolation using "pair style sph/rhosum".

See this PDF guide to using SPH in LAMMPS.

The value of the mesoscopic density will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom vector values will be in mass/volume units.

Restrictions:

This compute is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

dump custom

Default: none

compute meso_t/atom command

Syntax:

```
compute ID group-ID meso_t/atom
```

- ID, group-ID are documented in [compute](#) command
- meso_t/atom = style name of this compute command

Examples:

```
compute 1 all meso_t/atom
```

Description:

Define a computation that calculates the per-atom internal temperature for each atom in a group.

The internal temperature is the ratio of internal energy over the heat capacity associated with the internal degrees of freedom of a mesoscopic particles, e.g. a Smooth-Particle Hydrodynamics particle.

T_*int* = E_*int* / C_*V, int*

See [this PDF guide](#) to using SPH in LAMMPS.

The value of the internal energy will be 0.0 for atoms not in the specified compute group.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom vector values will be in temperature [units](#).

Restrictions:

This compute is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[dump custom](#)

Default: none

compute_modify command

Syntax:

```
compute_modify compute-ID keyword value ...
```

- compute-ID = ID of the compute to modify
- one or more keyword/value pairs may be listed
- keyword = *extra* or *dynamic*

```
    extra value = N
        N = # of extra degrees of freedom to subtract
    dynamic value = yes or no
      yes/no = do or do not recompute the number of atoms contributing to the temperature
    thermo value = yes or no
      yes/no = do or do not add contributions from fixes to the potential energy
```

Examples:

```
compute_modify myTemp extra 0
compute_modify newtemp dynamic yes extra 600
```

Description:

Modify one or more parameters of a previously defined compute. Not all compute styles support all parameters.

The *extra* keyword refers to how many degrees-of-freedom are subtracted (typically from 3N) as a normalizing factor in a temperature computation. Only computes that compute a temperature use this option. The default is 2 or 3 for 2d or 3d systems which is a correction factor for an ensemble of velocities with zero total linear momentum. You can use a negative number for the *extra* parameter if you need to add degrees-of-freedom. See the compute temp/asphere command for an example.

The *dynamic* keyword determines whether the number of atoms N in the compute group is re-computed each time a temperature is computed. Only compute styles that compute a temperature use this option. By default, N is assumed to be constant. If you are adding atoms to the system (see the fix pour or fix deposit commands) or expect atoms to be lost (e.g. due to evaporation), then this option can be used to insure the temperature is correctly normalized.

The *thermo* keyword determines whether the potential energy contribution calculated by some fixes is added to the potential energy calculated by the compute. Currently, only the compute of style *pe* uses this option. See the doc pages for individual fixes for details.

Restrictions: none

Related commands:

compute

Default:

The option defaults are extra = 2 or 3 for 2d or 3d systems and dynamic = no. Thermo is *yes* if the compute of style *pe* was defined with no extra keywords; otherwise it is *no*.

compute msd command

Syntax:

```
compute ID group-ID msd keyword values ...
```

- ID, group-ID are documented in [compute](#) command
- msd = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *com*

  com value = *yes* or *no*

Examples:

```
compute 1 all msd
compute 1 upper msd com yes
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) of the group of atoms, including all effects due to atoms passing thru periodic boundaries. For computation of the non-Gaussian parameter of mean-squared displacement, see the [compute msd/nongauss](#) command.

A vector of four quantites is calculated by this compute. The first 3 elements of the vector are the squared dx,dy,dz displacements, summed and averaged over atoms in the group. The 4th element is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz), summed and averaged over atoms in the group.

The slope of the mean-squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing atoms.

The displacement of an atom is from its original position at the time the compute command was issued. The value of the displacement will be 0.0 for atoms not in the specified compute group.

If the *com* option is set to *yes* then the effect of any drift in the center-of-mass of the group of atoms is subtracted out before the displacment of each atom is calcluated.

IMPORTANT NOTE: Initial coordinates are stored in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the [set image](#) command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the [fix rigid](#) command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the [fix rigid](#) command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post-process a [dump file](#) containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: If you want the quantities calculated by this compute to be continuous when running from a [restart file](#), then you should use the same ID for this compute, as in the original run. This is so that the created fix will also have the same ID, and thus be initialized correctly with atom coordinates from the restart file.

Output info:

compute msd command   140

This compute calculates a global vector of length 4, which can be accessed by indices 1-4 by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values are "intensive". The vector values will be in distance^2 units.

Restrictions: none

Related commands:

compute msd/nongauss, compute displace_atom, fix store/state, compute msd/molecule

Default:

The option default is com = no.

compute msd/molecule command

Syntax:

```
compute ID group-ID msd/molecule
```

- ID, group-ID are documented in compute command
- msd/molecule = style name of this compute command

Examples:

```
compute 1 all msd/molecule
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) of individual molecules. The calculation includes all effects due to atoms passing thru periodic boundaries.

Four quantites are calculated by this compute for each molecule. The first 3 quantities are the squared dx,dy,dz displacements of the center-of-mass. The 4th component is the total squared displacement, i.e. (dx*dx + dy*dy + dz*dz) of the center-of-mass.

The slope of the mean-squared displacement (MSD) versus time is proportional to the diffusion coefficient of the diffusing molecules.

The displacement of the center-of-mass of the molecule is from its original center-of-mass position at the time the compute command was issued.

The MSD for a particular molecule is only computed if one or more of its atoms are in the specified group. Normally all atoms in the molecule should be in the group, however this is not required. LAMMPS will warn you if this is not the case. Only atoms in the group contribute to the center-of-mass calculation for the molecule, which is used to caculate its initial and current position.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

IMPORTANT NOTE: The initial coordinates of each molecule are stored in "unwrapped" form, by using the image flags associated with each atom. See the dump custom command for a discussion of "unwrapped" coordinates. See the Atoms section of the read_data command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this compute by using the set image command.

IMPORTANT NOTE: If an atom is part of a rigid body (see the fix rigid command), it's periodic image flags are altered, and its contribution to the MSD may not reflect its true contribution. See the fix rigid command for details. Thus, to compute the MSD of rigid bodies as they cross periodic boundaries, you will need to post-process a dump file containing coordinates of the atoms in the bodies.

IMPORTANT NOTE: Unlike the compute msd command, this compute does not store the initial center-of-mass coorindates of its molecules in a restart file. Thus you cannot continue the MSD per molecule calculation of this compute when running from a restart file.

Output info:

This compute calculates a global array where the number of rows = Nmolecules and the number of columns = 4 for dx,dy,dz and the total displacement. These values can be accessed by any command that uses global array values from a compute as input. See this section for an overview of LAMMPS output options.

The array values are "intensive". The array values will be in distance^2 units.

Restrictions: none

Related commands:

compute msd

Default: none

compute msd/nongauss command

Syntax:

```
compute ID group-ID msd/nongauss keyword values ...
```

- ID, group-ID are documented in compute command
- msd/nongauss = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *com*

```
com value = yes or no
```

Examples:

```
compute 1 all msd/nongauss
compute 1 upper msd/nongauss com yes
```

Description:

Define a computation that calculates the mean-squared displacement (MSD) and non-Gaussian parameter (NGP) of the group of atoms, including all effects due to atoms passing thru periodic boundaries.

A vector of three quantites is calculated by this compute. The first element of the vector is the total squared dx,dy,dz displacements drsquared = (dx*dx + dy*dy + dz*dz) of atoms, and the second is the fourth power of these displacements drfourth = (dx*dx + dy*dy + dz*dz)*(dx*dx + dy*dy + dz*dz), summed and averaged over atoms in the group. The 3rd component is the nonGaussian diffusion paramter NGP = 3*drfourth/(5*drsquared*drsquared), i.e.

$$NGP(t) = 3 < (r(t) - r(0))^4 > / (5 < (r(t) - r(0))^2 >^2) - 1$$

The NGP is a commonly used quantity in studies of dynamical heterogeneity. Its minimum theoretical value (-0.4) occurs when all atoms have the same displacement magnitude. NGP=0 for Brownian diffusion, while NGP > 0 when some mobile atoms move faster than others.

If the *com* option is set to *yes* then the effect of any drift in the center-of-mass of the group of atoms is subtracted out before the displacment of each atom is calcluated.

See the compute msd doc page for further IMPORTANT NOTES, which also apply to this compute.

Output info:

This compute calculates a global vector of length 3, which can be accessed by indices 1-3 by any command that uses global vector values from a compute as input. See this section for an overview of LAMMPS output options.

The vector values are "intensive". The first vector value will be in distance^2 units, the second is in distance^4 units, and the 3rd is dimensionless.

Restrictions:

This compute is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute msd

Default:

The option default is com = no.

compute nparticles/tracer/region command

Syntax:

```
compute ID group-ID nparticles/tracer/region
```

- ID, group-ID are documented in <u>compute</u> command
- nparticles/tracer/region = style name of this compute command
- region_count = obligatory keyword
- region-ID = ID of region atoms must be in to be counted
- tracer = obligatory keyword
- tracer-ID = ID of a fix of type <u>fix property/atom/tracer</u>
- zero or more keyword/value pairs may be appended to args
- keyword = *periodic* or *check_mark_every*

```
periodic value = dim image
    dim = x or y or z
    image = image that a particle has to be in to be counted (any integer number or all)
  reset_marker value = yes or no
    yes = un-mark particles after counting them
    no = do not un-mark particles after counting them
```

Examples:

```
compute nparticles all nparticles/tracer/region region_count count tracer tr periodic z -1
```

Description:

Define a computation that calculates the number and mass of marked and un-marked particles that are in the region speficied via the *region_count* keyword. Particles have to be in the group "group-ID" to be counted.

Note that only particles marked by a <u>fix property/atom/tracer</u> or <u>fix property/atom/tracer/stream</u> command are counted - therefore, a valid ID of such a fix has to be provided via the *tracer* keyword.

The *reset_marker* keyword controls if particles are un-marked (default) after they have been counted once by this command.

IMPORTANT NOTE: If multiple compute nparticles/tracer/region commands are operating on the same <u>fix property/atom/tracer</u> commands, and the first compute resets the marker value, the second compute will not count them.

With the *periodic* keyword, you can restrict counting/unmarking to particles which are in a specified image in a periodic simulation. For example, using

```
periodic z +2
```

means that particles are only counted if they are in z-image #2. By default, all particles are counted/unmarked regardless in which periodic image they are.

IMPORTANT NOTE: Currently, this command only supports one periodic boundary restriction via the *periodic* keyword. If keyword *periodic* is used multiple times, the last setting will be applied.

Output info:

This this compute calculates a global vector containing the following information (the number in brackets corresponds to the vector id):

- **(1)** total number of (marked + un-marked) particles in region
- **(2)** number of marked particles in region
- **(3)** total mass of (marked + un-marked) particles in region
- **(4)** mass of marked particles in region

See this section for an overview of LAMMPS output options.

Restrictions:

Currently, only one periodic restriction via the *periodic* keyword can be used.

Related commands:

fix property/atom/tracer

Default: *reset_marker* = yes, *periodic* is off per default

compute pair/gran/local command

compute wall/gran/local command

Syntax:

```
compute ID group-ID pair/gran/local keywords ...
compute ID group-ID wall/gran/local keywords ...
```

- ID, group-ID are documented in compute command
- pair/gran/local or wall/gran/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *pos* or *vel* or *id* or *force* or *torque* or *history* or *contactArea*

```
pos = positions of particles in contact (6 values)
  vel = velocities of particles in contact (6 values)
  id = IDs of particles in contact and a periodicity flag (3 values) or IDs of the mesh, t
  force = contact force (3 values)
  torque = torque divided by particle diameter (3 values)
  history = contact history (# depends on pair style, e.g. 3 shear history values)
  contactArea = area of the contact (1 value)
  heatFlux = conductive heat flux of the contact (1 value)
```

Examples:

```
compute 1 all pair/gran/local
compute 1 all pair/gran/local pos force
compute 1 all wall/gran/local
```

Description:

Define a computation that calculates properties of individual pairwise or particle-wall interactions of a granular pair style. The number of datums generated, aggregated across all processors, equals the number of pairwise interactions or particle-wall interactions in the system.

The local data stored by this command is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group, and if the current pairwise distance is less than the force cutoff distance for that interaction, as defined by the pair_style and pair_coeff commands.

IMPORTANT NOTE: For accessing particle-wall contact data, only mesh walls (see fix mesh) can be used. For computing particle-wall (compute wall/gran/local), the code will automatically look for a fix wall/gran command that uses mesh walls. The order of the meshes in the fix wall/gran command is called the mesh id (starting with 0), and the triangle id reflects the order of the triangles in the STL/VTK file read via the dedicated fix mesh command. For how to output the trangle id, see "dump mesh/gran/VTK command"dump.html.

The output *pos* is the particle positions (6 values) in distance units. Keyword *vel* will do the same for velocities. For computing pairwise data, the output *id* will be the two particle IDs (using this option requires to use an atom map) and a flag that is 1 for interaction over a periodic boundary and 0 otherwise. For computing particle-wall data, the output *id* will be the mesh id, the triangle id and the particle id. The output *force* and *torque* are the contact force and the torque divided by the particle radius, both in force units. Note that the torque does NOT contain any rolling friction torque. The output *history* will depend on what this history represents, according to the granular pair style used. The output *contactArea* will output the contact

area, in distance^2 units. The output *heatFlux* (available only if a fix heat/gran is used to compute heat fluxes) will output the per-contact conductive heat flux area, in energy/time units.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, pair output from the compute property/local command can be combined with data from this command and output by the dump local command in a consistent way.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of pairs. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

For information on the units of the output, see above.

Restrictions:

Can only be used together with a granular pair style. For accessing particle-wall contact data, only mesh walls can be used.

Related commands:

dump local, compute property/local, compute pair/local

Default:

By default, all of the outputs keywords (except the heat flux) are activated, i.e. when no keyword is used, positions velocities, ids, forces, torques, history and contact area are output.

compute pair command

Syntax:

```
compute ID group-ID pair pstyle evalue
```

- ID, group-ID are documented in [compute](#) command
- pair = style name of this compute command
- pstyle = style name of a pair style that calculates additional values
- evalue = *epair* or *evdwl* or *ecoul* or blank (optional setting)

Examples:

```
compute 1 all pair gauss
compute 1 all pair lj/cut/coul/cut ecoul
compute 1 all pair reax
```

Description:

Define a computation that extracts additional values calculated by a pair style, sums them across processors, and makes them accessible for output or further processing by other commands. The group specified for this command is ignored.

The specified *pstyle* must be a pair style used in your simulation either by itself or as a sub-style in a [pair_style hybrid or hybrid/overlay](#) command.

The *evalue* setting is optional; it may be left off the command. All pair styles tally a potential energy *epair* which may be broken into two parts: *evdwl* and *ecoul* such that *epair = evdwl + ecoul*. If the pair style calculates Coulombic interactions, their energy will be tallied in *ecoul*. Everything else (whether it is a Lennard-Jones style van der Waals interaction or not) is tallied in *evdwl*. If *evalue* is specified as *epair* or left out, then *epair* is stored as a global scalar by this compute. This is useful when using [pair_style hybrid](#) if you want to know the portion of the total energy contributed by one sub-style. If *evalue* is specfied as *evdwl* or *ecoul*, then just that portion of the energy is stored as a global scalar.

Some pair styles tally additional quantities, e.g. a breakdown of potential energy into a dozen or so components is tallied by the [pair_style reax](#) commmand. These values (1 or more) are stored as a global vector by this compute. See the doc page for [individual pair styles](#) for info on these values.

Output info:

This compute calculates a global scalar which is *epair* or *evdwl* or *ecoul*. If the pair style supports it, it also calculates a global vector of length >= 1, as determined by the pair style. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "extensive".

The scalar value will be in energy [units](#). The vector values will typically also be in energy [units](#), but see the doc page for the pair style for details.

Restrictions: none

Related commands:

compute pair command   150

[compute pe](#)

Default:

The default for *evalue* is *epair*.

[compute pe](#)

Default:

compute pair/local command

Syntax:

```
compute ID group-ID pair/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- pair/local = style name of this compute command
- zero or more keywords may be appended
- keyword = *dist* or *eng* or *force* or *fx* or *fy* or *fz* or *pN*

```
 dist = pairwise distance
   eng = pairwise energy
   force = pairwise force
   fx,fy,fz = components of pairwise force
   pN = pair style specific quantities for allowed N values
```

Examples:

```
compute 1 all pair/local eng
compute 1 all pair/local dist eng force
compute 1 all pair/local dist eng fx fy fz
compute 1 all pair/local dist fx fy fz p1 p2 p3
```

Description:

Define a computation that calculates properties of individual pairwise interactions. The number of datums generated, aggregated across all processors, equals the number of pairwise interactions in the system.

The local data stored by this command is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group, and if the current pairwise distance is less than the force cutoff distance for that interaction, as defined by the [pair_style](#) and [pair_coeff](#) commands.

The output *dist* is the distance bewteen the pair of atoms.

The output *eng* is the interaction energy for the pair of atoms.

The output *force* is the force acting between the pair of atoms, which is positive for a repulsive force and negative for an attractive force. The outputs *fx*, *fy*, and *fz* are the xyz components of *force* on atom I.

A pair style may define additional pairwise quantities which can be accessed as *p1* to *pN*, where N is defined by the pair style. Most pair styles do not define any additional quantities, so N = 0. An example of ones that do are the [granular pair styles](#) which calculate the tangential force between two particles and return its components and magnitude acting on atom I for N = 1,2,3,4. See individual pair styles for detils.

The output *dist* will be in distance [units](#). The output *eng* will be in energy [units](#). The outputs *force*, *fx*, *fy*, and *fz* will be in force [units](#). The output *pN* will be in whatever units the pair style defines.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, pair output from the [compute property/local](#) command can be combined with data from this command and output by the [dump local](#) command in a consistent way.

IMPORTANT NOTE: For pairs, if two atoms I,J are involved in 1-2, 1-3, 1-4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this may be true of I,J pairs with a weighting factor of 0.0; pairs with a non-zero weighting factor are included. The weighting factors for 1-2, 1-3, and 1-4 pairwise interactions are set by the special_bonds command.

Output info:

This compute calculates a local vector or local array depending on the number of keywords. The length of the vector or number of rows in the array is the number of pairs. If a single keyword is specified, a local vector is produced. If two or more keywords are specified, a local array is produced where the number of columns = the number of keywords. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The output for *dist* will be in distance units. The output for *eng* will be in energy units. The output for *force* will be in force units.

Restrictions: none

Related commands:

dump local, compute property/local

Default: none

compute pe/atom command

Syntax:

```
compute ID group-ID pe/atom keyword ...
```

- ID, group-ID are documented in [compute](#) command
- pe/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace*

Examples:

```
compute 1 all pe/atom
compute 1 all pe/atom pair
compute 1 all pe/atom pair bond
```

Description:

Define a computation that computes the per-atom potential energy for each atom in a group. See the [compute pe](#) command if you want the potential energy of the entire system.

The per-atom energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral,improper, and kspace energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

Note that the energy of each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

For an energy contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3-body interaction), that energy is assigned in equal portions to each atom in the set. E.g. 1/4 of the dihedral energy to each of the 4 atoms.

The [dihedral_style charmm](#) style calculates pairwise interactions between 1-4 atoms. The energy contribution of these terms is included in the pair energy, not the dihedral energy.

The KSpace contribution is calculated using the method in [(Heyes)](#) for the Ewald method and a related method for PPPM, as specified by the [kspace_style pppm](#) command. For PPPM, the calcluation requires 1 extra FFT each timestep that per-atom energy is calculated. Thie [document](#) describes how the long-range per-atom energy calculation is performed.

As an example of per-atom potential energy compared to total potential energy, these lines in an input script should yield the same result in the last 2 columns of thermo output:

```
compute        peratom all pe/atom
compute        pe all reduce sum c_peratom
thermo_style   custom step temp etotal press pe c_pe
```

IMPORTANT NOTE: The per-atom energy does not any Lennard-Jones tail corrections invoked by the [pair_modify tail yes](#) command, since those are global contributions to the system energy.

Output info:

This compute calculates a per-atom vector, which can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The per-atom vector values will be in energy units.

Restrictions:

Related commands:

compute pe, compute stress/atom

Default: none

(Heyes) Heyes, Phys Rev B 49, 755 (1994),

compute pe command

compute pe/cuda command

Syntax:

```
compute ID group-ID pe keyword ...
```

- ID, group-ID are documented in [compute](#) command
- pe = style name of this compute command
- zero or more keywords may be appended
- keyword = *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace*

Examples:

```
compute 1 all pe
compute molPE all pe bond angle dihedral improper
```

Description:

Define a computation that calculates the potential energy of the entire system of atoms. The specified group must be "all". See the [compute pe/atom](#) command if you want per-atom energies. These per-atom values could be summed for a group of atoms via the [compute reduce](#) command.

The energy is calculated by the various pair, bond, etc potentials defined for the simulation. If no extra keywords are listed, then the potential energy is the sum of pair, bond, angle, dihedral, improper, and kspace (long-range) energy. If any extra keywords are listed, then only those components are summed to compute the potential energy.

The Kspace contribution requires 1 extra FFT each timestep the energy is calculated, if using the PPPM solver via the [kspace_style pppm](#) command. Thus it can increase the cost of the PPPM calculation if it is needed on a large fraction of the simulation timesteps.

Various fixes can contribute to the total potential energy of the system. See the doc pages for [individual fixes](#) for details. The *thermo* option of the [compute_modify](#) command determines whether these contributions are added into the computed potential energy. If no keywords are specified the default is *yes*. If any keywords are specified, the default is *no*.

A compute of this style with the ID of "thermo_pe" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_pe all pe
```

See the "thermo_style" command for more details.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the potential energy). This value can be used by any command that uses a global scalar value from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive". The scalar value will be in energy units.

Restrictions: none

Related commands:

compute pe/atom

Default: none

compute pressure command

compute pressure/cuda command

Syntax:

```
compute ID group-ID pressure temp-ID keyword ...
```

- ID, group-ID are documented in [compute](#) command
- pressure = style name of this compute command
- temp-ID = ID of compute that calculates temperature
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

Examples:

```
compute 1 all pressure myTemp
compute 1 all pressure thermo_temp pair bond
```

Description:

Define a computation that calculates the pressure of the entire system of atoms. The specified group must be "all". See the [compute stress/atom](#) command if you want per-atom pressure (stress). These per-atom values could be summed for a group of atoms via the [compute reduce](#) command.

The pressure is computed by the formula

$$P = \frac{Nk_BT}{V} + \frac{\sum_i^N r_i \bullet f_i}{dV}$$

where N is the number of atoms in the system (see discussion of DOF below), Kb is the Boltzmann constant, T is the temperature, d is the dimensionality of the system (2 or 3 for 2d/3d), V is the system volume (or area in 2d), and the second term is the virial, computed within LAMMPS for all pairwise as well as 2-body, 3-body, and 4-body, and long-range interactions. [Fixes](#) that impose constraints (e.g. the [fix shake](#) command) also contribute to the virial term.

A symmetric pressure tensor, stored as a 6-element vector, is also calculated by this compute. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz. The equation for the I,J components (where I and J = x,y,z) is similar to the above formula, except that the first term uses components of the kinetic energy tensor and the second term uses components of the virial tensor:

$$P_{IJ} = \frac{\sum_k^N m_k v_{k_I} v_{k_J}}{V} + \frac{\sum_k^N r_{k_I} f_{k_J}}{V}$$

If no extra keywords are listed, the entire equations above are calculated which include a kinetic energy (temperature) term and the virial as the sum of pair, bond, angle, dihedral, improper, kspace (long-range), and fix contributions to the force on each atom. If any extra keywords are listed, then only those components are summed to compute temperature or ke and/or the virial. The *virial* keyword means include all terms except the kinetic energy *ke*.

The temperature and kinetic energy tensor is not calculated by this compute, but rather by the temperature compute specified with the command. Normally this compute should calculate the temperature of all atoms for consistency with the virial term, but any compute style that calculates temperature can be used, e.g. one that excludes frozen atoms or other degrees of freedom.

Note that the N in the first formula above is really degrees-of-freedom divided by d = dimensionality, where the DOF value is calcluated by the temperature compute. See the various compute temperature styles for details.

A compute of this style with the ID of "thermo_press" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_press all pressure thermo_temp
```

where "thermo_temp" is the ID of a similarly defined compute of style "temp". See the "thermo_style" command for more details.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the pressure) and a global vector of length 6 (pressure tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar and vector values calculated by this compute are "intensive". The scalar and vector values will be in pressure units.

Restrictions: none

Related commands:

compute temp, compute stress/atom, thermo_style,

Default: none

compute property/atom command

Syntax:

```
compute ID group-ID property/atom input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- property/atom = style name of this compute command
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz, mu,
                      radius, diameter, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz,
                      shapex,shapey, shapez,
                      quatw, quati, quatj, quatk, tqx, tqy, tqz,
                      spin, eradius, ervel, erforce
                      end1x, end1y, end1z, end2x, end2y, end2z,
                      corner1x, corner1y, corner1z,
                      corner2x, corner2y, corner2z,
                      corner3x, corner3y, corner3z,
                      i_name, d_name

      id = atom ID
      mol = molecule ID
      type = atom type
      mass = atom mass
      x,y,z = unscaled atom coordinates
      xs,ys,zs = scaled atom coordinates
      xu,yu,zu = unwrapped atom coordinates
      ix,iy,iz = box image that the atom is in
      vx,vy,vz = atom velocities
      fx,fy,fz = forces on atoms
      q = atom charge
      mux,muy,muz = orientation of dipole moment of atom
      mu = magnitude of dipole moment of atom
      radius,diameter = radius,diameter of spherical particle
      omegax,omegay,omegaz = angular velocity of spherical particle
      angmomx,angmomy,angmomz = angular momentum of aspherical particle
      shapex,shapey,shapez = 3 diameters of aspherical particle
      quatw,quati,quatj,quatk = quaternion components for aspherical or body particles
      tqx,tqy,tqz = torque on finite-size particles
      spin = electron spin
      eradius = electron radius
      ervel = electron radial velocity
      erforce = electron radial force
      end12x, end12y, end12z = end points of line segment
      coner123x, corner123y, corner123z = corner points of triangle
      i_name = custom integer vector with name
      d_name = custom integer vector with name
```

Examples:

```
compute 1 all property/atom xs vx fx mux
compute 2 all property/atom type
compute 1 all property/atom ix iy iz
```

Description:

Define a computation that simply stores atom attributes for each atom in the group. This is useful so that the values can be used by other [output commands](#) that take computes as inputs. See for example, the [compute reduce](#), [fix ave/atom](#), [fix ave/histo](#), [fix ave/spatial](#), and [atom-style variable](#) commands.

The list of possible attributes is the same as that used by the [dump custom](#) command, which describes their meaning, with some additional quantities that are only defined for certain [atom styles](#). Basically, this list gives your input script access to any per-atom quantity stored by LAMMPS.

The values are stored in a per-atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group or for quantities that are not defined for a particular particle in the group (e.g. *shapex* if the particle is not an ellipsoid).

The additional quantities only accessible via this command, and not directly via the [dump custom](#) command, are as follows.

Shapex, *shapey*, and *shapez* are defined for ellipsoidal particles and define the 3d shape of each particle.

Quatw, *quati*, *quatj*, and *quatk* are defined for ellipsoidal particles and body particles and store the 4-vector quaternion representing the orientation of each particle. See the [set](#) command for an explanation of the quaternion vector.

End1x, *end1y*, *end1z*, *end2x*, *end2y*, *end2z*, are defined for line segment particles and define the end points of each line segment.

Corner1x, *corner1y*, *corner1z*, *corner2x*, *corner2y*, *corner2z*, *corner3x*, *corner3y*, *corner3z*, are defined for triangular particles and define the corner points of each triangle.

The *i_name* and *d_name* attributes refer to custom integer and floating-point properties that have been added to each atom via the [fix property/atom](#) command. When that command is used specific names are given to each attribute which are what is specified as the "name" portion of *i_name* or *d_name*.

Output info:

This compute calculates a per-atom vector or per-atom array depending on the number of input values. If a single input is specified, a per-atom vector is produced. If two or more inputs are specified, a per-atom array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses per-atom values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The vector or array values will be in whatever [units](#) the corresponding attribute is in, e.g. velocity units for vx, charge units for q, etc.

Restrictions: none

Related commands:

[dump custom](#), [compute reduce](#), [fix ave/atom](#), [fix ave/spatial](#), [fix property/atom](#)

Default: none

compute property/local command

Syntax:

```
compute ID group-ID property/local input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- property/local = style name of this compute command
- input = one or more attributes

```
possible attributes = natom1 natom2 ntype1 ntype2
                      patom1 patom2 ptype1 ptype2
                      batom1 batom2 btype
                      aatom1 aatom2 aatom3 atype
                      datom1 datom2 datom3 dtype
                      iatom1 iatom2 iatom3 itype

    natom1, natom2 = IDs of 2 atoms in each pair (within neighbor cutoff)
    ntype1, ntype2 = type of 2 atoms in each pair (within neighbor cutoff)
    patom1, patom2 = IDs of 2 atoms in each pair (within force cutoff)
    ptype1, ptype2 = type of 2 atoms in each pair (within force cutoff)
    batom1, batom2 = IDs of 2 atoms in each bond
    btype = bond type of each bond
    aatom1, aatom2, aatom3 = IDs of 3 atoms in each angle
    atype = angle type of each angle
    datom1, datom2, datom3, datom4 = IDs of 4 atoms in each dihedral
    dtype = dihedral type of each dihedral
    iatom1, iatom2, iatom3, iatom4 = IDs of 4 atoms in each improper
    itype = improper type of each improper
```

Examples:

```
compute 1 all property/local btype batom1 batom2
compute 1 all property/local atype aatom2
```

Description:

Define a computation that stores the specified attributes as local data so it can be accessed by other [output commands](#). If the input attributes refer to bond information, then the number of datums generated, aggregated across all processors, equals the number of bonds in the system. Ditto for pairs, angles, etc.

If multiple input attributes are specified then they must all generate the same amount of information, so that the resulting local array has the same number of rows for each column. This means that only bond attributes can be specified together, or angle attributes, etc. Bond and angle attributes can not be mixed in the same compute property/local command.

If the inputs are pair attributes, the local data is generated by looping over the pairwise neighbor list. Info about an individual pairwise interaction will only be included if both atoms in the pair are in the specified compute group. For *natom1* and *natom2*, all atom pairs in the neighbor list are considered (out to the neighbor cutoff = force cutoff + [neighbor skin](#)). For *patom1* and *patom2*, the distance between the atoms must be less than the force cutoff distance for that pair to be included, as defined by the [pair_style](#) and [pair_coeff](#) commands.

If the inputs are bond, angle, etc attributes, the local data is generated by looping over all the atoms owned on a processor and extracting bond, angle, etc info. For bonds, info about an individual bond will only be included if both atoms in the bond are in the specified compute group. Likewise for angles, dihedrals, etc.

Note that as atoms migrate from processor to processor, there will be no consistent ordering of the entries within the local vector or array from one timestep to the next. The only consistency that is guaranteed is that the ordering on a particular timestep will be the same for local vectors or arrays generated by other compute commands. For example, output from the compute bond/local command can be combined with bond atom indices from this command and output by the dump local command in a consistent way.

The *natom1* and *natom2*, or *patom1* and *patom2* attributes refer to the atom IDs of the 2 atoms in each pairwise interaction computed by the pair_style command. The *ntype1* and *ntype2*, or *ptype1* and *ptype2* attributes refer to the atom types of the 2 atoms in each pairwise interaction.

IMPORTANT NOTE: For pairs, if two atoms I,J are involved in 1-2, 1-3, 1-4 interactions within the molecular topology, their pairwise interaction may be turned off, and thus they may not appear in the neighbor list, and will not be part of the local data created by this command. More specifically, this may be true of I,J pairs with a weighting factor of 0.0; pairs with a non-zero weighting factor are included. The weighting factors for 1-2, 1-3, and 1-4 pairwise interactions are set by the special_bonds command.

The *batom1* and *batom2* attributes refer to the atom IDs of the 2 atoms in each bond. The *btype* attribute refers to the type of the bond, from 1 to Nbtypes = # of bond types. The number of bond types is defined in the data file read by the read_data command.

The attributes that start with "a", "d", "i", refer to similar values for angles, dihedrals, and impropers.

Output info:

This compute calculates a local vector or local array depending on the number of input values. The length of the vector or number of rows in the array is the number of bonds, angles, etc. If a single input is specified, a local vector is produced. If two or more inputs are specified, a local array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses local values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands:

dump local, compute reduce

Default: none

compute property/molecule command

Syntax:

```
compute ID group-ID property/molecule input1 input2 ...
```

- ID, group-ID are documented in compute command
- property/molecule = style name of this compute command
- input = one or more attributes

```
possible attributes = mol cout
  mol = molecule ID
  count = # of atoms in molecule
```

Examples:

```
compute 1 all property/molecule mol
```

Description:

Define a computation that stores the specified attributes as global data so it can be accessed by other output commands and used in conjunction with other commands that generate per-molecule data, such as compute com/molecule and compute msd/molecule.

The ordering of per-molecule quantities produced by this compute is consistent with the ordering produced by other compute commands that generate per-molecule datums. Conceptually, them molecule IDs will be in ascending order for any molecule with one or more of its atoms in the specified group.

The *mol* attribute is the molecule ID. This attribute can be used to produce molecule IDs as labels for per-molecule datums generated by other computes or fixes when they are output to a file, e.g. by the fix ave/time command.

The *count* attribute is the number of atoms in the molecule.

Output info:

This compute calculates a global vector or global array depending on the number of input values. The length of the vector or number of rows in the array is the number of molecules. If a single input is specified, a global vector is produced. If two or more inputs are specified, a global array is produced where the number of columns = the number of inputs. The vector or array can be accessed by any command that uses global values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values will be integers that correspond to the specified attribute.

Restrictions: none

Related commands: none

Default: none

compute rdf command

Syntax:

```
compute ID group-ID rdf Nbin itype1 jtype1 itype2 jtype2 ...
```

- ID, group-ID are documented in [compute](#) command
- rdf = style name of this compute command
- Nbin = number of RDF bins
- itypeN = central atom type for Nth RDF histogram (see asterisk form below)
- jtypeN = distribution atom type for Nth RDF histogram (see asterisk form below)

Examples:

```
compute 1 all rdf 100
compute 1 all rdf 100 1 1
compute 1 all rdf 100 * 3
compute 1 fluid rdf 500 1 1 1 2 2 1 2 2
compute 1 fluid rdf 500 1*3 2 5 *10
```

Description:

Define a computation that calculates the radial distribution function (RDF), also called g(r), and the coordination number for a group of particles. Both are calculated in histogram form by binning pairwise distances into *Nbin* bins from 0.0 to the maximum force cutoff defined by the [pair_style](#) command. The bins are of uniform size in radial distance. Thus a single bin encompasses a thin shell of distances in 3d and a thin ring of distances in 2d.

IMPORTANT NOTE: If you have a bonded system, then the settings of [special_bonds](#) command can remove pairwise interactions between atoms in the same bond, angle, or dihedral. This is the default setting for the [special_bonds](#) command, and means those pairwise interactions do not appear in the neighbor list. Because this fix uses the neighbor list, it also means those pairs will not be included in the RDF. One way to get around this, is to write a dump file, and use the [rerun](#) command to compute the RDF for snapshots in the dump file. The rerun script can use a [special_bonds](#) command that includes all pairs in the neighbor list.

The *itypeN* and *jtypeN* arguments are optional. These arguments must come in pairs. If no pairs are listed, then a single histogram is computed for g(r) between all atom types. If one or more pairs are listed, then a separate histogram is generated for each *itype*,*jtype* pair.

The *itypeN* and *jtypeN* settings can be specified in one of two ways. An explicit numeric value can be used, as in the 4th example above. Or a wild-card asterisk can be used to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

If both *itypeN* and *jtypeN* are single values, as in the 4th example above, this means that a g(r) is computed where atoms of type *itypeN* are the central atom, and atoms of type *jtypeN* are the distribution atom. If either *itypeN* and *jtypeN* represent a range of values via the wild-card asterisk, as in the 5th example above, this means that a g(r) is computed where atoms of any of the range of types represented by *itypeN* are the central atom, and atoms of any of the range of types represented by *jtypeN* are the distribution atom.

Pairwise distances are generated by looping over a pairwise neighbor list, just as they would be in a [pair_style](#) computation. The distance between two atoms I and J is included in a specific histogram if the following

criteria are met:

- atoms I,J are both in the specified compute group
- the distance between atoms I,J is less than the maximum force cutoff
- the type of the I atom matches itypeN (one or a range of types)
- the type of the J atom matches jtypeN (one or a range of types)

It is OK if a particular pairwise distance is included in more than one individual histogram, due to the way the *itypeN* and *jtypeN* arguments are specified.

The g(r) value for a bin is calculated from the histogram count by scaling it by the idealized number of how many counts there would be if atoms of type *jtypeN* were uniformly distributed. Thus it involves the count of *itypeN* atoms, the count of *jtypeN* atoms, the volume of the entire simulation box, and the volume of the bin's thin shell in 3d (or the area of the bin's thin ring in 2d).

A coordination number coord(r) is also calculated, which is the number of atoms of type *jtypeN* within the current bin or closer, averaged over atoms of type *itypeN*. This is calculated as the area- or volume-weighted sum of g(r) values over all bins up to and including the current bin, multiplied by the global average volume density of atoms of type jtypeN.

The simplest way to output the results of the compute rdf calculation to a file is to use the [fix ave/time](#) command, for example:

```
compute myRDF all rdf 50
fix 1 all ave/time 100 1 100 c_myRDF file tmp.rdf mode vector
```

Output info:

This compute calculates a global array with the number of rows = *Nbins*, and the number of columns = 1 + 2*Npairs, where Npairs is the number of I,J pairings specified. The first column has the bin coordinate (center of the bin), Each successive set of 2 columns has the g(r) and coord(r) values for a specific set of *itypeN* versus *jtypeN* interactions, as described above. These values can be used by any command that uses a global values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The array values calculated by this compute are all "intensive".

The first column of array values will be in distance [units](#). The g(r) columns of array values are normalized numbers >= 0.0. The coordination number columns of array values are also numbers >= 0.0.

Restrictions:

The RDF is not computed for distances longer than the force cutoff, since processors (in parallel) don't know about atom coordinates for atoms further away than that distance. If you want an RDF for larger distances, you can use the [rerun](#) command to post-process a dump file. The definition of g(r) used by LAMMPS is only appropriate for characterizing atoms that are uniformly distributed throughout the simulation cell. In such cases, the coordination number is still correct and meaningful. As an example, if a large simulation cell contains only one atom of type *itypeN* and one of *jtypeN*, then g(r) will register an arbitrarily large spike at whatever distance they happen to be at, and zero everywhere else. coord(r) will show a step change from zero to one at the location of the spike in g(r).

Related commands:

[fix ave/time](#)

Default: none

compute rdf command   166

compute reduce command

compute reduce/region command

Syntax:

```
compute ID group-ID style arg mode input1 input2 ... keyword args ...
```

- ID, group-ID are documented in [compute](#) command
- style = *reduce* or *reduce/region*

```
  reduce arg = none
    reduce/region arg = region-ID
      region-ID = ID of region to use for choosing atoms
```
- mode = *sum* or *min* or *max* or *ave*
- one or more inputs can be listed
- input = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
  x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
  c_ID = per-atom or local vector calculated by a compute with ID
  c_ID[I] = Ith column of per-atom or local array calculated by a compute with ID
  f_ID = per-atom or local vector calculated by a fix with ID
  f_ID[I] = Ith column of per-atom or local array calculated by a fix with ID
  v_name = per-atom vector calculated by an atom-style variable with name
```
- zero or more keyword/args pairs may be appended
- keyword = *replace*

```
  replace args = vec1 vec2
      vec1 = reduced value from this input vector will be replaced
      vec2 = replace it with vec1[N] where N is index of max/min value from vec2
```

Examples:

```
compute 1 all reduce sum c_force
compute 1 all reduce/region subbox sum c_force
compute 2 all reduce min c_press2 f_ave v_myKE
compute 3 fluid reduce max c_index1 c_index2 c_dist replace 1 3 replace 2 3
```

Description:

Define a calculation that "reduces" one or more vector inputs into scalar values, one per listed input. The inputs can be per-atom or local quantities; they cannot be global quantities. Atom attributes are per-atom quantities, [computes](#) and [fixes](#) may generate any of the three kinds of quantities, and [atom-style variables](#) generate per-atom quantities. See the [variable](#) command and its special functions which can perform the same operations as the compute reduce command on global vectors.

The reduction operation is specified by the *mode* setting. The *sum* option adds the values in the vector into a global total. The *min* or *max* options find the minimum or maximum value across all vector values. The *ave* setting adds the vector values into a global total, then divides by the number of values in the vector.

Each listed input is operated on independently. For per-atom inputs, the group specified with this command means only atoms within the group contribute to the result. For per-atom inputs, if the compute reduce/region command is used, the atoms must also currently be within the region. Note that an input that produces per-atom quantities may define its own group which affects the quantities it returns. For example, if a compute is used as an input which generates a per-atom vector, it will generate values of 0.0 for atoms that are

not in the group specified for that compute.

Each listed input can be an atom attribute (position, velocity, force component) or can be the result of a [compute](#) or [fix](#) or the evaluation of an atom-style [variable](#).

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the [compute property/atom](#) command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. Computes can generate per-atom or local quantities. See the individual [compute](#) doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LAMMPS](#).

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. Fixes can generate per-atom or local quantities. See the individual [fix](#) doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute reduce references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and [add them to LAMMPS](#).

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. It must be an [atom-style variable](#). Atom-style variables can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to reduce.

If the *replace* keyword is used, two indices *vec1* and *vec2* are specified, where each index ranges from 1 to the # of input values. The replace keyword can only be used if the *mode* is *min* or *max*. It works as follows. A min/max is computed as usual on the *vec2* input vector. The index N of that value within *vec2* is also stored. Then, instead of performing a min/max on the *vec1* input vector, the stored index is used to select the Nth element of the *vec1* vector.

Thus, for example, if you wish to use this compute to find the bond with maximum stretch, you can do it as follows:

```
compute 1 all property/local batom1 batom2
compute 2 all bond/local dist
compute 3 all reduce max c_1[1] c_1[2] c_2 replace 1 3 replace 2 3
thermo_style custom step temp c_3[1] c_3[2] c_3[3]
```

The first two input values in the compute reduce command are vectors with the IDs of the 2 atoms in each bond, using the [compute property/local](#) command. The last input value is bond distance, using the [compute bond/local](#) command. Instead of taking the max of the two atom ID vectors, which does not yield useful information in this context, the *replace* keywords will extract the atom IDs for the two atoms in the bond of maximum stretch. These atom IDs and the bond stretch will be printed with thermodynamic output.

If a single input is specified this compute produces a global scalar value. If multiple inputs are specified, this compute produces a global vector of values, the length of which is equal to the number of inputs specified.

As discussed below, for *sum* mode, the value(s) produced by this compute are all "extensive", meaning their value scales linearly with the number of atoms involved. If normalized values are desired, this compute can be accessed by the [thermo_style custom](#) command with [thermo_modify norm yes](#) set as an option. Or it can be accessed by a [variable](#) that divides by the appropriate atom count.

Output info:

This compute calculates a global scalar if a single input value is specified or a global vector of length N where N is the number of inputs, and which can be accessed by indices 1 to N. These values can be used by any command that uses global scalar or vector values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

All the scalar or vector values calculated by this compute are "intensive", except when the *sum* mode is used on per-atom or local vectors, in which case the calculated values are "extensive".

The scalar or vector values will be in whatever [units](#) the quantities being reduced are in.

Restrictions: none

Related commands:

[compute](#), [fix](#), [variable](#)

Default: none

compute slice command

Syntax:

```
compute ID group-ID slice Nstart Nstop Nskip input1 input2 ...
```

- ID, group-ID are documented in [compute](#) command
- slice = style name of this compute command
- Nstart = starting index within input vector(s)
- Nstop = stopping index within input vector(s)
- Nskip = extract every Nskip elements from input vector(s)
- input = c_ID, c_ID[N], f_ID, f_ID[N]

```
    c_ID = global vector calculated by a compute with ID
    c_ID[I] = Ith column of global array calculated by a compute with ID
    f_ID = global vector calculated by a fix with ID
    f_ID[I] = Ith column of global array calculated by a fix with ID
```

Examples:

```
compute 1 all slice 1 100 10 c_msdmol[4]
compute 1 all slice 301 400 1 c_msdmol[4]
```

Description:

Define a calculation that "slices" one or more vector inputs into smaller vectors, one per listed input. The inputs can be global quantities; they cannot be per-atom or local quantities. [Computes](#) and [fixes](#) may generate any of the three kinds of quantities. [Variables](#) do not generate global vectors. The group specified with this command is ignored.

The values extracted from the input vector(s) are determined by the *Nstart*, *Nstop*, and *Nskip* parameters. The elements of an input vector of length N are indexed from 1 to N. Starting at element *Nstart*, every Mth element is extracted, where M = *Nskip*, until element *Nstop* is reached. The extracted quantities are stored as a vector, which is typically shorter than the input vector.

Each listed input is operated on independently to produce one output vector. Each listed input must be a global vector or column of a global array calculated by another [compute](#) or [fix](#).

If an input value begins with "c_", a compute ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual [compute](#) doc page for details. If no bracketed integer is appended, the vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the array calculated by the compute is used. Users can also write code for their own compute styles and [add them to LAMMPS](#).

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script and which generates a global vector or array. See the individual [fix](#) doc page for details. Note that some fixes only produce their values on certain timesteps, which must be compatible with when compute slice references the values, else an error results. If no bracketed integer is appended, the vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the array calculated by the fix is used. Users can also write code for their own fix style and [add them to LAMMPS](#).

If a single input is specified this compute produces a global vector, even if the length of the vector is 1. If multiple inputs are specified, then a global array of values is produced, with the number of columns equal to

the number of inputs specified.

Output info:

This compute calculates a global vector if a single input value is specified or a global array with N columns where N is the number of inputs. The length of the vector or the number of rows in the array is equal to the number of values extracted from each input vector. These values can be used by any command that uses global vector or array values from a compute as input. See this section for an overview of LAMMPS output options.

The vector or array values calculated by this compute are simply copies of values generated by computes or fixes that are input vectors to this compute. If there is a single input vector of intensive and/or extensive values, then each value in the vector of values calculated by this compute will be "intensive" or "extensive", depending on the corresponding input value. If there are multiple input vectors, and all the values in them are intensive, then the array values calculated by this compute are "intensive". If there are multiple input vectors, and any value in them is extensive, then the array values calculated by this compute are "extensive".

The vector or array values will be in whatever units the input quantities are in.

Restrictions: none

Related commands:

compute, fix, compute reduce

Default: none

compute stress/atom command

Syntax:

```
compute ID group-ID stress/atom keyword ...
```

- ID, group-ID are documented in compute command
- stress/atom = style name of this compute command
- zero or more keywords may be appended
- keyword = *ke* or *pair* or *bond* or *angle* or *dihedral* or *improper* or *kspace* or *fix* or *virial*

Examples:

```
compute 1 mobile stress/atom
compute 1 all stress/atom pair bond
```

Description:

Define a computation that computes the symmetric per-atom stress tensor for each atom in a group. The tensor for each atom has 6 components and is stored as a 6-element vector in the following order: xx, yy, zz, xy, xz, yz. See the compute pressure command if you want the stress tensor (pressure) of the entire system.

The stress tensor for atom *I* is given by the following formula, where *a* and *b* take on values x,y,z to generate the 6 components of the symmetric tensor:

$$
\begin{aligned}
S_{ab} = -\Bigg[& mv_a v_b + \frac{1}{2} \sum_{n=1}^{N_p} (r_{1_a} F_{1_b} + r_{2_a} F_{2_b}) + \frac{1}{2} \sum_{n=1}^{N_b} (r_{1_a} F_{1_b} + r_{2_a} F_{2_b}) + \\
& \frac{1}{3} \sum_{n=1}^{N_a} (r_{1_a} F_{1_b} + r_{2_a} F_{2_b} + r_{3_a} F_{3_b}) + \frac{1}{4} \sum_{n=1}^{N_d} (r_{1_a} F_{1_b} + r_{2_a} F_{2_b} + r_{3_a} F_{3_b} + r_{4_a} F_{4_b}) + \\
& \frac{1}{4} \sum_{n=1}^{N_i} (r_{1_a} F_{1_b} + r_{2_a} F_{2_b} + r_{3_a} F_{3_b} + r_{4_a} F_{4_b}) + \text{Kspace}(r_{i_a}, F_{i_b}) + \sum_{n=1}^{N_f} r_{i_a} F_{i_b} \Bigg]
\end{aligned}
$$

The first term is a kinetic energy contribution for atom *I*. The second term is a pairwise energy contribution where *n* loops over the *Np* neighbors of atom *I*, *r1* and *r2* are the positions of the 2 atoms in the pairwise interaction, and *F1* and *F2* are the forces on the 2 atoms resulting from the pairwise interaction. The third term is a bond contribution of similar form for the *Nb* bonds which atom *I* is part of. There are similar terms for the *Na* angle, *Nd* dihedral, and *Ni* improper interactions atom *I* is part of. There is also a term for the KSpace contribution from long-range Coulombic interactions, if defined. Finally, there is a term for the *Nf* fixes that apply internal constraint forces to atom *I*. Currently, only the fix shake and fix rigid commands contribute to this term.

IMPORTANT NOTE: For granular systems, this formular neglects the contribution of average velocity in the kinetic energy contribution. This is corrected in the compute ave/euler command (currently no doc available).

As the coefficients in the formula imply, a virial contribution produced by a small set of atoms (e.g. 4 atoms in a dihedral or 3 atoms in a Tersoff 3-body interaction) is assigned in equal portions to each atom in the set.

E.g. 1/4 of the dihedral virial to each of the 4 atoms, or 1/3 of the fix virial due to SHAKE constraints applied to atoms in a a water molecule via the [fix shake](#) command.

If no extra keywords are listed, all of the terms in this formula are included in the per-atom stress tensor. If any extra keywords are listed, only those terms are summed to compute the tensor. The *virial* keyword means include all terms except the kinetic energy *ke*.

Note that the stress for each atom is due to its interaction with all other atoms in the simulation, not just with other atoms in the group.

The [dihedral_style charmm](#) style calculates pairwise interactions between 1-4 atoms. The virial contribution of these terms is included in the pair virial, not the dihedral virial.

The KSpace contribution is calculated using the method in [(Heyes)](#) for the Ewald method and by the methodology described in [(Sirk)](#) for PPPM. The choice of KSpace solver is specified by the [kspace_style pppm](#) command. Note that for PPPM, the calcluation requires 6 extra FFTs each timestep that per-atom stress is calculated. Thus it can significantly increase the cost of the PPPM calculation if it is needed on a large fraction of the simulation timesteps.

Note that as defined in the formula, per-atom stress is the negative of the per-atom pressure tensor. It is also really a stress*volume formulation, meaning the computed quantity is in units of pressure*volume. It would need to be divided by a per-atom volume to have units of stress (pressure), but an individual atom's volume is not well defined or easy to compute in a deformed solid or a liquid. Thus, if the diagonal components of the per-atom stress tensor are summed for all atoms in the system and the sum is divided by dV, where d = dimension and V is the volume of the system, the result should be -P, where P is the total pressure of the system.

These lines in an input script for a 3d system should yield that result. I.e. the last 2 columns of thermo output will be the same:

```
compute         peratom all stress/atom
compute         p all reduce sum c_peratom[1] c_peratom[2] c_peratom[3]
variable        press equal -(c_p[1]+c_p[2]+c_p[3])/(3*vol)
thermo_style    custom step temp etotal press v_press
```

Output info:

This compute calculates a per-atom array with 6 columns, which can be accessed by indices 1-6 by any command that uses per-atom values from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The per-atom array values will be in pressure*volume [units](#) as discussed above.

Restrictions: none

Related commands:

[compute pe](#), [compute pressure](#)

Default: none

(Heyes) Heyes, Phys Rev B 49, 755 (1994),

(Sirk) Sirk, Moore, Brown, J Chem Phys, 138, 064505 (2013).

compute temp/asphere command

Syntax:

```
compute ID group-ID temp/asphere keyword value ...
```

- ID, group-ID are documented in compute command
- temp/asphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

```
  bias value = bias-IDuniform or gaussian
      bias-ID = ID of a temperature compute that removes a velocity bias
    dof value = all or rotate
      all = compute temperature of translational and rotational degrees of freedom
      rotate = compute temperature of just rotational degrees of freedom
```

Examples:

```
compute 1 all temp/asphere
compute myTemp mobile temp/asphere bias tempCOM
compute myTemp mobile temp/asphere dof rotate
```

Description:

Define a computation that calculates the temperature of a group of aspherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual compute temp command, which assumes point particles with only translational kinetic energy.

Only finite-size particles (aspherical or spherical) can be included in the group. For 3d finite-size particles, each has 6 degrees of freedom (3 translational, 3 rotational). For 2d finite-size particles, each has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: This choice for degrees of freedom (dof) assumes that all finite-size aspherical or spherical particles in your model will freely rotate, sampling all their rotational dof. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less dof and you should use the compute_modify extra command to adjust the dof accordingly.

For example, an aspherical particle with all three of its shape parameters the same is a sphere. If it does not rotate, then it should have 3 dof instead of 6 in 3d (or 2 instead of 3 in 2d). A uniaxial aspherical particle has two of its three shape parameters the same. If it does not rotate around the axis perpendicular to its circular cross section, then it should have 5 dof instead of 6 in 3d. The latter is the case for uniaxial ellipsoids in a GayBerne model since there is no induced torque around the optical axis. It will also be the case for biaxial ellipsoids when exactly two of the semiaxes have the same length and the corresponding relative well depths are equal.

The translational kinetic energy is computed the same as is described by the compute temp command. The rotational kinetic energy is computed as 1/2 I w^2, where I is the inertia tensor for the aspherical particle and w is its angular velocity, which is computed from its angular momentum.

IMPORTANT NOTE: For 2d models, particles are treated as ellipsoids, not ellipses, meaning their moments of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formula, except that v^2 and w^2 are replaced by vx*vy and wx*wy for the xy component, and the appropriate elements of the inertia tensor are used. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

This compute subtracts out translational degrees-of-freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way, e.g. to remove a velocity profile. Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This compute requires that atoms store angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

compute temp

Default: none

compute temp/com command

Syntax:

```
compute ID group-ID temp/com
```

- ID, group-ID are documented in [compute](#) command
- temp/com = style name of this compute command

Examples:

```
compute 1 all temp/com
compute myTemp mobile temp/com
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center-of-mass velocity of the group. This is useful if the group is expected to have a non-zero net velocity for some reason. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

After the center-of-mass velocity has been subtracted from each atom, the temperature is calculated by the formula $KE = dim/2 \, N \, k \, T$, where KE = total kinetic energy of the group of atoms (sum of $1/2 \, m \, v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by v_x*v_y for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

The removal of the center-of-mass velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#).

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature [units](). The vector values will be in energy [units]().

Restrictions: none

Related commands:

[compute temp]()

Default: none

compute temp/deform/eff command

Syntax:

```
compute ID group-ID temp/deform/eff
```

- ID, group-ID are documented in compute command
- temp/deform/eff = style name of this compute command

Examples:

```
compute myTemp all temp/deform/eff
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the electron force field model, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the fix deform/eff command. A compute of this style is created by the fix nvt/sllod/eff command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature, e.g. thermo_modify, fix npt/eff, etc.

The calculation performed by this compute is exactly like that described by the compute temp/deform command, except that the formula for the temperature includes the radial electron velocity contributions, as discussed by the compute temp/eff command. Note that only the translational degrees of freedom for each nuclei or electron are affected by the streaming velocity adjustment. The radial velocity component of the electrons is not affected.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/ramp, fix deform/eff, fix nvt/sllod/eff

Default: none

compute temp/deform command

Syntax:

```
compute ID group-ID temp/deform
```

- ID, group-ID are documented in [compute](#) command
- temp/deform = style name of this compute command

Examples:

```
compute myTemp all temp/deform
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out a streaming velocity induced by the simulation box changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the [fix deform](#) command. A compute of this style is created by the [fix nvt/sllod](#) command to compute the thermal temperature of atoms for thermostatting purposes. A compute of this style can also be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

The deformation fix changes the box size and/or shape over time, so each atom in the simulation box can be thought of as having a "streaming" velocity. For example, if the box is being sheared in x, relative to y, then atoms at the bottom of the box (low y) have a small x velocity, while atoms at the top of the box (hi y) have a large x velocity. This position-dependent streaming velocity is subtracted from each atom's actual velocity to yield a thermal velocity which is used to compute the temperature.

IMPORTANT NOTE: [Fix deform](#) has an option for remapping either atom coordinates or velocities to the changing simulation box. When using this compute in conjunction with a deforming box, fix deform should NOT remap atom positions, but rather should let atoms respond to the changing box by adjusting their own velocities (or let [fix deform](#) remap the atom velocities, see it's remap option). If fix deform does remap atom positions, then they appear to move with the box but their velocity is not changed, and thus they do NOT have the streaming velocity assumed by this compute. LAMMPS will warn you if fix deform is defined and its remap setting is not consistent with this compute.

After the streaming velocity has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of $1/2\ m\ v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature. Note that v in the kinetic energy formula is the atom's thermal velocity.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by v_x*v_y for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

The removal of the box deformation velocity component by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining

thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

IMPORTANT NOTE: The temperature calculated by this compute is only accurate if the atoms are indeed moving with a stream velocity profile that matches the box deformation. If not, then the compute will subtract off an incorrect stream velocity, yielding a bogus thermal temperature. You should NOT assume that your atoms are streaming at the same rate the box is deforming. Rather, you should monitor their velocity profile, e.g. via the fix ave/spatial command. And you can compare the results of this compute to compute temp/profile, which actually calculates the stream profile before subtracting it. If the two computes do not give roughly the same temperature, then your atoms are not streaming consistent with the box deformation. See the fix deform command for more details on ways to get atoms to stream consistently with the box deformation.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp/ramp, compute temp/profile, fix deform, fix nvt/sllod

Default: none

compute temp/eff command

Syntax:

```
compute ID group-ID temp/eff
```

- ID, group-ID are documented in [compute](#) command
- temp/eff = style name of this compute command

Examples:

```
compute 1 all temp/eff
compute myTemp mobile temp/eff
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the [electron force field](#) model. A compute of this style can be used by commands that compute a temperature, e.g. [thermo_modify](#), [fix npt/eff](#), etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of $1/2 \, m \, v^2$ for nuclei and sum of $1/2 \, (m \, v^2 + 3/4 \, m \, s^2)$ for electrons, where s includes the radial electron velocity contributions), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms (only total number of nuclei in the eFF (see the [pair_eff](#) command) in the group, k = Boltzmann constant, and T = temperature. This expression is summed over all nuclear and electronic degrees of freedom, essentially by setting the kinetic contribution to the heat capacity to 3/2k (where only nuclei contribute). This subtlety is valid for temperatures well below the Fermi temperature, which for densities two to five times the density of liquid H2 ranges from 86,000 to 170,000 K.

IMPORTANT NOTE: For eFF models, in order to override the default temperature reported by LAMMPS in the thermodynamic quantities reported via the [thermo](#) command, the user should apply a [thermo_modify](#) command, as shown in the following example:

```
compute        effTemp all temp/eff
thermo_style   custom step etotal pe ke temp press
thermo_modify  temp effTemp
```

A 6-component kinetic energy tensor is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx * vy for the xy component, etc. For the eFF, again, the radial electronic velocities are also considered.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

The scalar value calculated by this compute is "intensive", meaning it is independent of the number of atoms in the simulation. The vector values are "extensive", meaning they scale with the number of atoms in the simulation.

Restrictions:

This compute is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/partial, compute temp/region, compute pressure

Default: none

compute temp command

compute temp/cuda command

Syntax:

```
compute ID group-ID temp
```

- ID, group-ID are documented in [compute](#) command
- temp = style name of this compute command

Examples:

```
compute 1 all temp
compute myTemp mobile temp
```

Description:

Define a computation that calculates the temperature of a group of atoms. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of $1/2\ m\ v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

A compute of this style with the ID of "thermo_temp" is created when LAMMPS starts up, as if this command were in the input script:

```
compute thermo_temp all temp
```

See the "thermo_style" command for more details.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp/partial, compute temp/region, compute pressure

Default: none

compute temp/partial command

compute temp/partial/cuda command

Syntax:

```
compute ID group-ID temp/partial xflag yflag zflag
```

- ID, group-ID are documented in [compute](#) command
- temp/partial = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension

Examples:

```
compute newT flow temp/partial 1 1 0
```

Description:

Define a computation that calculates the temperature of a group of atoms, after excluding one or more velocity components. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of $1/2 m v^2$), dim = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature. The calculation of KE excludes the x, y, or z dimensions if xflag, yflag, or zflag = 0. The dim parameter is adjusted to give the correct number of degrees of freedom.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the calculation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

The removal of velocity components by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#).

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for

round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute temp/region, compute pressure

Default: none

compute temp/profile command

Syntax:

```
compute ID group-ID temp/profile xflag yflag zflag binstyle args
```

- ID, group-ID are documented in [compute](#) command
- temp/profile = style name of this compute command
- xflag,yflag,zflag = 0/1 for whether to exclude/include this dimension
- binstyle = *x* or *y* or *z* or *xy* or *yz* or *xz* or *xyz*

```
  x arg = Nx
    y arg = Ny
    z arg = Nz
    xy args = Nx Ny
    yz args = Ny Nz
    xz args = Nx Nz
    xyz args = Nx Ny Nz
      Nx,Ny,Nz = number of velocity bins in x,y,z dimensions
```
- zero or more keyword/value pairs may be appended
- keyword = *out*

```
  out value = tensor or bin
```

Examples:

```
compute myTemp flow temp/profile 1 1 1 x 10
compute myTemp flow temp/profile 1 1 1 x 10 out bin
compute myTemp flow temp/profile 0 1 1 xyz 20 20 20
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out a spatially-averaged center-of-mass velocity field, before computing the kinetic energy. This can be useful for thermostatting a collection of atoms undergoing a complex flow, e.g. via a profile-unbiased thermostat (PUT) as described in [(Evans)](#). A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

The *xflag*, *yflag*, *zflag* settings determine which components of average velocity are subtracted out.

The *binstyle* setting and its *Nx*, *Ny*, *Nz* arguments determine how bins are setup to perform spatial averaging. "Bins" can be 1d slabs, 2d pencils, or 3d bricks depending on which *binstyle* is used. The simulation box is partitioned conceptually into *Nx* by *Ny* by *Nz* bins. Depending on the *binstyle*, you may only specify one or two of these values; the others are effectively set to 1 (no binning in that dimension). For non-orthogonal (triclinic) simulation boxes, the bins are "tilted" slabs or pencils or bricks that are parallel to the tilted faces of the box. See the [region prism](#) command for a discussion of the geometry of tilted boxes in LAMMPS.

When a temperature is computed, the center-of-mass velocity for the set of atoms that are both in the compute group and in the same spatial bin is calculated. This bias velocity is then subtracted from the velocities of individual atoms in the bin to yield a thermal velocity for each atom. Note that if there is only one atom in the bin, its thermal velocity will thus be 0.0.

After the spatially-averaged velocity field has been subtracted from each atom, the temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of 1/2 m v^2),

dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

If the *out* keyword is used with a *tensor* value, which is the default, a kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

If the *out* keyword is used with a *bin* value, the count of atoms and computed temperature for each bin are stored for output, as an array of values, as described below. The temperature of each bin is calculated as described above, where the bias velocity is subtracted and only the remaining thermal velocity of atoms in the bin contributes to the temperature. See the note below for how the temperature is normalized by the degrees-of-freedom of atoms in the bin.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the compute_modify command if this is not the case.

The removal of the spatially-averaged velocity field by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include fix nvt, fix temp/rescale, fix temp/berendsen, and fix langevin.

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the compute_modify command.

IMPORTANT NOTE: When using the *out* keyword with a value of *bin*, the calculated temperature for each bin does not include the degrees-of-freedom adjustment described in the preceeding paragraph, for fixes that constrain molecular motion. It does include the adjustment due to the *extra* option, which is applied to each bin.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting. Using this compute in conjunction with a thermostatting fix, as explained there, will effectively implement a profile-unbiased thermostat (PUT), as described in (Evans).

Output info:

This compute calculates a global scalar (the temperature). Depending on the setting of the *out* keyword, it also calculates a global vector or array. For *out = tensor*, it calculates a vector of length 6 (KE tensor), which can be accessed by indices 1-6. For *out = bin* it calculates a global array which has 2 columns and N rows, where N is the number of bins. The first column contains the number of atoms in that bin. The second contains the temperature of that bin, calculated as described above. The ordering of rows in the array is as follows. Bins in x vary fastest, then y, then z. Thus for a 10x10x10 3d array of bins, there will be 1000 rows. The bin with indices ix,iy,iz = 2,3,4 would map to row M = (iz-1)*10*10 + (iy-1)*10 + ix = 322, where the rows are numbered from 1 to 1000 and the bin indices are numbered from 1 to 10 in each dimension.

These values can be used by any command that uses global scalar or vector or array values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive". The array values are "intensive".

The scalar value will be in temperature units. The vector values will be in energy units. The first column of array values are counts; the values in the second column will be in temperature units.

Restrictions:

You should not use too large a velocity-binning grid, especially in 3d. In the current implementation, the binned velocity averages are summed across all processors, so this will be inefficient if the grid is too large, and the operation is performed every timestep, as it will be for most thermostats.

Related commands:

compute temp, compute temp/ramp, compute temp/deform, compute pressure

Default:

The option default is out = tensor.

(Evans) Evans and Morriss, Phys Rev Lett, 56, 2172-2175 (1986).

compute temp/ramp command

Syntax:

```
compute ID group-ID temp/ramp vdim vlo vhi dim clo chi keyword value ...
```

- ID, group-ID are documented in [compute](#) command
- temp/ramp = style name of this compute command
- vdim = *vx* or *vy* or *vz*
- vlo,vhi = subtract velocities between vlo and vhi (velocity units)
- dim = *x* or *y* or *z*
- clo,chi = lower and upper bound of domain to subtract from (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
```

Examples:

```
compute 2nd middle temp/ramp vx 0 8 y 2 12 units lattice
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out an ramped velocity profile before computing the kinetic energy. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

The meaning of the arguments for this command which define the velocity ramp are the same as for the [velocity ramp](#) command which was presumably used to impose the velocity.

After the ramp velocity has been subtracted from the specified dimension for each atom, the temperature is calculated by the formula $KE = dim/2\ N\ k\ T$, where KE = total kinetic energy of the group of atoms (sum of $1/2\ m\ v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

The *units* keyword determines the meaning of the distance units used for coordinates (c1,c2) and velocities (vlo,vhi). A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings; e.g. velocity = lattice spacings / tau. The [lattice](#) command must have been previously used to define the lattice spacing.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by $vx*vy$ for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

The removal of the ramped velocity component by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#).

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as fix shake and fix rigid. This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the compute_modify command.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute temp/profie, compute temp/deform, compute pressure

Default:

The option default is units = lattice.

compute temp/region/eff command

Syntax:

```
compute ID group-ID temp/region/eff region-ID
```

- ID, group-ID are documented in compute command
- temp/region/eff = style name of this compute command
- region-ID = ID of region to use for choosing atoms

Examples:

```
compute mine flow temp/region/eff boundary
```

Description:

Define a computation that calculates the temperature of a group of nuclei and electrons in the electron force field model, within a geometric region using the electron force field. A compute of this style can be used by commands that compute a temperature, e.g. thermo_modify.

The operation of this compute is exactly like that described by the compute temp/region command, except that the formula for the temperature itself includes the radial electron velocity contributions, as discussed by the compute temp/eff command.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions:

This compute is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute temp/region, compute temp/eff, compute pressure

Default: none

compute temp/region command

Syntax:

```
compute ID group-ID temp/region region-ID
```

- ID, group-ID are documented in [compute](#) command
- temp/region = style name of this compute command
- region-ID = ID of region to use for choosing atoms

Examples:

```
compute mine flow temp/region boundary
```

Description:

Define a computation that calculates the temperature of a group of atoms in a geometric region. This can be useful for thermostatting one portion of the simulation box. E.g. a McDLT simulation where one side is cooled, and the other side is heated. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), etc.

Note that a *region*-style temperature can be used to thermostat with [fix temp/rescale](#) or [fix langevin](#), but should probably not be used with Nose/Hoover style fixes ([fix nvt](#), [fix npt](#), or [fix nph](#)), if the degrees-of-freedom included in the computed T varies with time.

The temperature is calculated by the formula KE = dim/2 N k T, where KE = total kinetic energy of the group of atoms (sum of $1/2$ m v^2), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in both the group and region, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by vx*vy for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is compute each time the temperature is evaluated since it is assumed atoms can enter/leave the region. Thus there is no need to use the *dynamic* option of the [compute_modify](#) command for this compute style.

The removal of atoms outside the region by this fix is essentially computing the temperature after a "bias" has been removed, which in this case is the velocity of any atoms outside the region. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#). This means any of the thermostatting fixes can operate on a geometric region of atoms, as defined by this compute.

Unlike other compute styles that calculate temperature, this compute does not subtract out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This is because those degrees of freedom (e.g. a constrained bond) can straddle the region boundary, and hence the concept is somewhat ill-defined. If needed the number of subtracted degrees-of-freedom can be set explicitly using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform

thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See this section for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature units. The vector values will be in energy units.

Restrictions: none

Related commands:

compute temp, compute pressure

Default: none

compute temp/rotate command

Syntax:

```
compute ID group-ID temp/rotate
```

- ID, group-ID are documented in [compute](#) command
- temp/rotate = style name of this compute command

Examples:

```
compute Tbead bead temp/rotate
```

Description:

Define a computation that calculates the temperature of a group of atoms, after subtracting out the center-of-mass velocity and angular velocity of the group. This is useful if the group is expected to have a non-zero net velocity and/or global rotation motion for some reason. A compute of this style can be used by any command that computes a temperature, e.g. [thermo_modify](#), [fix temp/rescale](#), [fix npt](#), etc.

After the center-of-mass velocity and angular velocity has been subtracted from each atom, the temperature is calculated by the formula $KE = dim/2 \, N \, k \, T$, where KE = total kinetic energy of the group of atoms (sum of $1/2 \, m \, v^2$), dim = 2 or 3 = dimensionality of the simulation, N = number of atoms in the group, k = Boltzmann constant, and T = temperature.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute for use in the computation of a pressure tensor. The formula for the components of the tensor is the same as the above formula, except that v^2 is replaced by v_x*v_y for the xy component, etc. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

The removal of the center-of-mass velocity and angular velocity by this fix is essentially computing the temperature after a "bias" has been removed from the velocity of the atoms. If this compute is used with a fix command that performs thermostatting then this bias will be subtracted from each atom, thermostatting of the remaining thermal velocity will be performed, and the bias will be added back in. Thermostatting fixes that work in this way include [fix nvt](#), [fix temp/rescale](#), [fix temp/berendsen](#), and [fix langevin](#).

This compute subtracts out degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature [units](). The vector values will be in energy [units]().

Restrictions:

This compute is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS]() section for more info.

Related commands:

[compute temp]()

Default: none

compute temp/sphere command

Syntax:

```
compute ID group-ID temp/sphere keyword value ...
```

- ID, group-ID are documented in [compute](#) command
- temp/sphere = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *bias* or *dof*

```
  bias value = bias-IDuniform or gaussian
      bias-ID = ID of a temperature compute that removes a velocity bias
    dof value = all or rotate
      all = compute temperature of translational and rotational degrees of freedom
      rotate = compute temperature of just rotational degrees of freedom
```

Examples:

```
compute 1 all temp/sphere
compute myTemp mobile temp/sphere bias tempCOM
compute myTemp mobile temp/sphere dof rotate
```

Description:

Define a computation that calculates the temperature of a group of spherical particles, including a contribution from both their translational and rotational kinetic energy. This differs from the usual [compute temp](#) command, which assumes point particles with only translational kinetic energy.

Both point and finite-size particles can be included in the group. Point particles do not rotate, so they have only 3 translational degrees of freedom. For 3d spherical particles, each has 6 degrees of freedom (3 translational, 3 rotational). For 2d spherical particles, each has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: This choice for degrees of freedom (dof) assumes that all finite-size spherical particles in your model will freely rotate, sampling all their rotational dof. It is possible to use a combination of interaction potentials and fixes that induce no torque or otherwise constrain some of all of your particles so that this is not the case. Then there are less dof and you should use the [compute_modify extra](#) command to adjust the dof accordingly.

The translational kinetic energy is computed the same as is described by the [compute temp](#) command. The rotational kinetic energy is computed as $1/2\ I\ w^2$, where I is the moment of inertia for a sphere and w is the particle's angular velocity.

IMPORTANT NOTE: For [2d models](#), particles are treated as spheres, not disks, meaning their moment of inertia will be the same as in 3d.

A kinetic energy tensor, stored as a 6-element vector, is also calculated by this compute. The formula for the components of the tensor is the same as the above formulas, except that v^2 and w^2 are replaced by v_x*v_y and w_x*w_y for the xy component. The 6 components of the vector are ordered xx, yy, zz, xy, xz, yz.

The number of atoms contributing to the temperature is assumed to be constant for the duration of the run; use the *dynamic* option of the [compute_modify](#) command if this is not the case.

This compute subtracts out translational degrees-of-freedom due to fixes that constrain molecular motion, such as [fix shake](#) and [fix rigid](#). This means the temperature of groups of atoms that include these constraints will be computed correctly. If needed, the subtracted degrees-of-freedom can be altered using the *extra* option of the [compute_modify](#) command.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The keyword/value option pairs are used in the following ways.

For the *bias* keyword, *bias-ID* refers to the ID of a temperature compute that removes a "bias" velocity from each atom. This allows compute temp/sphere to compute its thermal temperature after the translational kinetic energy components have been altered in a prescribed way, e.g. to remove a velocity profile. Thermostats that use this compute will work with this bias term. See the doc pages for individual computes that calculate a temperature and the doc pages for fixes that perform thermostatting for more details.

For the *dof* keyword, a setting of *all* calculates a temperature that includes both translational and rotational degrees of freedom. A setting of *rotate* calculates a temperature that includes only rotational degrees of freedom.

Output info:

This compute calculates a global scalar (the temperature) and a global vector of length 6 (KE tensor), which can be accessed by indices 1-6. These values can be used by any command that uses global scalar or vector values from a compute as input. See [this section](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "intensive". The vector values are "extensive".

The scalar value will be in temperature [units](#). The vector values will be in energy [units](#).

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the [atom_style sphere](#) command.

All particles in the group must be finite-size spheres, or point particles with radius = 0.0.

Related commands:

[compute temp](#), [compute temp/asphere](#)

Default:

The option defaults are no bias and dof = all.

compute ti command

Syntax:

```
compute ID group ti keyword args ...
```

- ID, group-ID are documented in [compute](#) command
- ti = style name of this compute command
- one or more attribute/arg pairs may be appended
- keyword = pair style (lj/cut, gauss, born, etc) or *tail* or *kspace*

```
    pair style args = atype v_name1 v_name2
      atype = atom type (see asterisk form below)
      v_name1 = variable with name1 that is energy scale factor and function of lambda
      v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
    tail args = atype v_name1 v_name2
      atype = atom type (see asterisk form below)
      v_name1 = variable with name1 that is energy tail correction scale factor and function
      v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
    kspace args = atype v_name1 v_name2
      atype = atom type (see asterisk form below)
      v_name1 = variable with name1 that is K-Space scale factor and function of lambda
      v_name2 = variable with name2 that is derivative of v_name1 with respect to lambda
```

Examples:

```
compute 1 all ti lj/cut 1 v_lj v_dlj coul/long 2 v_c v_dc kspace 1 v_ks v_dks
compute 1 all ti lj/cut 1*3 v_lj v_dlj coul/long * v_c v_dc kspace * v_ks v_dks
```

Description:

Define a computation that calculates the derivative of the interaction potential with respect to *lambda*, the coupling parameter used in a thermodynamic integration. This derivative can be used to infer a free energy difference resulting from an alchemical simulation, as described in [Eike](#).

Typically this compute will be used in conjunction with the [fix adapt](#) command which can perform alchemical transformations by adusting the strength of an interaction potential as a simulation runs, as defined by one or more [pair_style](#) or [kspace_style](#) commands. This scaling is done via a prefactor on the energy, forces, virial calculated by the pair or K-Space style. The prefactor is often a function of a *lambda* parameter which may be adjusted from 0 to 1 (or vice versa) over the course of a [run](#). The time-dependent adjustment is what the [fix adapt](#) command does.

Assume that the unscaled energy of a pair_style or kspace_style is given by U. Then the scaled energy is

```
Us = f(lambda) U
```

where f() is some function of lambda. What this compute calculates is

```
dUs / d(lambda) = U df(lambda)/dlambda = Us / f(lambda) df(lambda)/dlambda
```

which is the derivative of the system's scaled potential energy Us with respect to *lambda*.

To perform this calculation, you provide one or more atom types as *atype*. *Atype* can be specified in one of two ways. An explicit numeric values can be used, as in the 1st example above. Or a wildcard asterisk can be used in place of or in conjunction with the *atype* argument to select multiple atom types. This takes the form

"*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

You also specify two functions, as [equal-style variables](#). The first is specified as *v_name1*, where *name1* is the name of the variable, and is f(lambda) in the notation above. The second is specified as *v_name2*, where *name2* is the name of the variable, and is df(lambda) / dlambda in the notation above. I.e. it is the analytic derivative of f() with respect to lambda. Note that the *name1* variable is also typically given as an argument to the [fix adapt](#) command.

An alchemical simulation may use several pair potentials together, invoked via the [pair_style hybrid or hybrid/overlay](#) command. The total dUs/dlambda for the overall system is calculated as the sum of each contributing term as listed by the keywords in the compute ti command. Individual pair potentials can be listed, which will be sub-styles in the hybrid case. You can also include a K-space term via the *kspace* keyword. You can also include a pairwise long-range tail correction to the energy via the *tail* keyword.

For each term you can specify a different (or the same) scale factor by the two variables that you list. Again, these will typically correspond toe the scale factors applied to these various potentials and the K-Space contribution via the [fix_adapt](#) command.

More details about the exact functional forms for the computation of du/dl can be found in the paper by [Eike](#).

Output info:

This compute calculates a global scalar, namely dUs/dlambda. This value can be used by any command that uses a global scalar value from a compute as input. See [Section_howto 15](#) for an overview of LAMMPS output options.

The scalar value calculated by this compute is "extensive".

The scalar value will be in energy [units](#).

Restrictions:

This compute is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[fix adapt](#)

Default: none

(Eike) Eike and Maginn, Journal of Chemical Physics, 124, 164503 (2006).

compute voronoi/atom command

Syntax:

```
compute ID group-ID voronoi/atom keyword arg ...
```

- ID, group-ID are documented in compute command
- voronoi/atom = style name of this compute command
- zero or more keyword/value pairs may be appended
- keyword = *only_group* or *surface* or *radius* or *edge_histo* or *edge_threshold* or *face_threshold*

```
  only_group = no arg
    surface arg = sgroup-ID
      sgroup-ID = compute the dividing surface between group-ID and sgroup-ID
        this keyword adds a third column to the compute output
    radius arg = v_r
      v_r = radius atom style variable for a poly-disperse voronoi tessellation
    edge_histo arg = maxedge
      maxedge = maximum number of voronoi cell edges to be accounted in the histogram
    edge_threshold arg = minlength
      minlength = minimum length for an edge to be counted
    face_threshold arg = minarea
      minarea = minimum area for a face to be counted
```

Examples:

```
compute 1 all voronoi/atom
compute 2 precipitate voronoi/atom surface matrix
compute 3b precipitate voronoi/atom radius v_r
compute 4 solute voronoi/atom only_group
```

Description:

Define a computation that calculates the Voronoi tessellation of the atoms in the simulation box. The tessellation is calculated using all atoms in the simulation, but non-zero values are only stored for atoms in the group.

By default two quantities per atom are calculated by this compute. The first is the volume of the Voronoi cell around each atom. Any point in an atom's Voronoi cell is closer to that atom than any other. The second is the number of faces of the Voronoi cell, which is also the number of nearest neighbors of the atom in the middle of the cell.

If the *only_group* keyword is specified the tessellation is performed only with respect to the atoms contained in the compute group. This is equivalent to deleting all atoms not contained in the group prior to evaluating the tessellation.

If the *surface* keyword is specified a third quantity per atom is computed: the voronoi cell surface of the given atom. *surface* takes a group ID as an argument. If a group other than *all* is specified, only the voronoi cell facets facing a neighbor atom from the specified group are counted towards the surface area.

In the example above, a precipitate embedded in a matrix, only atoms at the surface of the precipitate will have non-zero surface area, and only the outward facing facets of the voronoi cells are counted (the hull of the precipitate). The total surface area of the precipitate can be obtained by running a "reduce sum" compute on c_2[3]

If the *radius* keyword is specified with an atom style variable as the argument, a poly-disperse voronoi tessellation is performed. Examples for radius variables are

```
variable r1 atom (type==1)*0.1+(type==2)*0.4
compute radius all property/atom radius
variable r2 atom c_radius
```

Here v_r1 specifies a per-type radius of 0.1 units for type 1 atoms and 0.4 units for type 2 atoms, and v_r2 accesses the radius property present in atom_style sphere for granular models.

The *edge_histo* keyword activates the compilation of a histogram of number of edges on the faces of the voronoi cells in the compute group. The argument maxedge of the this keyword is the largest number of edges on a single voronoi cell face expected to occur in the sample. This keyword adds the generation of a global vector with maxedge+1 entries. The last entry in the vector contains the number of faces with with more than maxedge edges. Since the polygon with the smallest amount of edges is a triangle, entries 1 and 2 of the vector will always be zero.

The *edge_threshold* and *face_threshold* keywords allow the suppression of edges below a given minimum length and faces below a given minimum area. Ultra short edges and ultra small faces can occur as artifacts of the voronoi tessellation. These keywords will affect the neighbor count and edge histogram outputs.

The Voronoi calculation is performed by the freely available Voro++ package, written by Chris Rycroft at UC Berkeley and LBL, which must be installed on your system when building LAMMPS for use with this compute. See instructions on obtaining and installing the Voro++ software in the src/VORONOI/README file.

IMPORTANT NOTE: The calculation of Voronoi volumes is performed by each processor for the atoms it owns, and includes the effect of ghost atoms stored by the processor. This assumes that the Voronoi cells of owned atoms are not affected by atoms beyond the ghost atom cut-off distance. This is usually a good assumption for liquid and solid systems, but may lead to underestimation of Voronoi volumes in low density systems. By default, the set of ghost atoms stored by each processor is determined by the cutoff used for pair_style interactions. The cutoff can be set explicitly via the communicate cutoff command.

IMPORTANT NOTE: The Voro++ package performs its calculation in 3d. This should still work for a 2d LAMMPS simulation, to effectively compute Voronoi "areas", so long as the z-dimension of the box is roughly the same (or smaller) compared to the separation of the atoms. Typical values for the z box dimensions in a 2d LAMMPS model are -0.5 to 0.5, which satisfies the criterion for most units systems. Note that you define the z extent of the simulation box for 2d simulations when using the create_box or read_data commands.

Output info:

This compute calculates a per-atom array with 2 columns. The first column is the Voronoi volume, the second is the neighbor count, as described above. These values can be accessed by any command that uses per-atom values from a compute as input. See Section_howto 15 for an overview of LAMMPS output options.

The Voronoi cell volume will be in distance units cubed.

Restrictions:

This compute is part of the VORONOI package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

compute voronoi/atom command   203

dump custom

Default: none

create_atoms command

Syntax:

```
create_atoms type style args keyword values ...
```

- type = atom type (1-Ntypes) of atoms to create
- style = *box* or *region* or *single* or *random*

```
  box args = none
    region args = region-ID
      region-ID = atoms will only be created if contained in the region
    single args = x y z
      x,y,z = coordinates of a single atom (distance units)
    random args = N seed region-ID
      N = number of atoms to create
      seed = random # seed (positive integer)
      region-ID = create atoms within this region, use NULL for entire simulation box
```

- zero or more keyword/value pairs may be appended
- keyword = *basis* or *remap* or *units*

```
  basis values = M itype
      M = which basis atom
      itype = atom type (1-N) to assign to this basis atom
    remap value = yes or no
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
create_atoms 1 box
create_atoms 3 region regsphere basis 2 3
create_atoms 3 single 0 0 5
```

Description:

This command creates atoms on a lattice, or a single atom, or a random collection of atoms, as an alternative to reading in their coordinates explicitly via a [read_data](#) or [read_restart](#) command. A simulation box must already exist, which is typically created via the [create_box](#) command. Before using this command, a lattice must also be defined using the [lattice](#) command. The only exceptions are for the *single* style with units = box or the *random* style.

For the *box* style, the create_atoms command fills the entire simulation box with atoms on the lattice. If your simulation box is periodic, you should insure its size is a multiple of the lattice spacings, to avoid unwanted atom overlaps at the box boundaries. If your box is periodic and a multiple of the lattice spacing in a particular dimension, LAMMPS is careful to put exactly one atom at the boundary (on either side of the box), not zero or two.

For the *region* style, the geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the [region](#) command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary. Also note that if your region is the same size as a periodic simulation box (in some dimension), LAMMPS does not implement the same logic as with the *box* style, to insure exactly one atom at the boundary. if this is what you desire, you should either use the *box* style, or tweak the region size to get precisely the atoms you want.

For the *single* style, a single atom is added to the system at the specified coordinates. This can be useful for debugging purposes or to create a tiny system with a handful of atoms at specified positions.

For the *random* style, N atoms are added to the system at randomly generated coordinates, which can be useful for generating an amorphous system. The atoms are created one by one using the speficied random number *seed*, resulting in the same set of atom coordinates, independent of how many processors are being used in the simulation. If the *region-ID* argument is specified as NULL, then the created atoms will be anywhere in the simulation box. If a *region-ID* is specified, a geometric volume is filled that is inside the simulation box and is also consistent with the region volume. See the region command for details. Note that a region can be specified so that its "volume" is either inside or outside a geometric boundary.

IMPORTANT NOTE: The atoms generated by the *random* style will typically be highly overlapped which will cause many interatomic potentials to compute large energies and forces. Thus you should either perform an energy minimization or run dynamics with fix nve/limit to equilibrate such a system, before running normal dynamics.

The *basis* keyword specifies an atom type that will be assigned to specific basis atoms as they are created. See the lattice command for specifics on how basis atoms are defined for the unit cell of the lattice. By default, all created atoms are assigned the argument *type* as their atom type.

The *remap* keyword only applies to the *single* style. If it is set to *yes*, then if the specified position is outside the simulation box, it will mapped back into the box, assuming the relevant dimensions are periodic. If it is set to *no*, no remapping is done and no atom is created if its position is outside the box.

The *units* keyword determines the meaning of the distance units used to specify the coordinates of the one atom created by the *single* style. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings.

Note that this command adds atoms to those that already exist. By using the create_atoms command multiple times, multiple sets of atoms can be added to the simulation. For example, interleaving create_atoms with lattice commands specifying different orientations, grain boundaries can be created. By using the create_atoms command in conjunction with the delete_atoms command, reasonably complex geometries can be created. The create_atoms command can also be used to add atoms to a system previously read in from a data or restart file. In all these cases, care should be taken to insure that new atoms do not overlap existing atoms inappropriately. The delete_atoms command can be used to handle overlaps.

Atom IDs are assigned to created atoms in the following way. The collection of created atoms are assigned consecutive IDs that start immediately following the largest atom ID existing before the create_atoms command was invoked. When a simulation is performed on different numbers of processors, there is no guarantee a particular created atom will be assigned the same ID.

Aside from their ID, atom type, and xyz position, other properties of created atoms are set to default values, depending on which quantities are defined by the chosen atom style. See the atom style command for more details. See the set and velocity commands for info on how to change these values.

- charge = 0.0
- dipole moment magnitude = 0.0
- diameter = 1.0
- shape = 0.0 0.0 0.0
- density = 1.0
- volume = 1.0
- velocity = 0.0 0.0 0.0
- angular velocity = 0.0 0.0 0.0
- angular momentum = 0.0 0.0 0.0

- quaternion = (1,0,0,0)
- bonds, angles, dihedrals, impropers = none

Note that the *sphere* atom style sets the default particle diameter to 1.0 as well as the density. This means the mass for the particle is not 1.0, but is PI/6 * diameter^3 = 0.5236.

Note that the *ellipsoid* atom style sets the default particle shape to (0.0 0.0 0.0) and the density to 1.0 which means it is a point particle, not an ellipsoid, and has a mass of 1.0.

Note that the *peri* style sets the default volume and density to 1.0 and thus also set the mass for the particle to 1.0.

The set command can be used to override many of these default settings.

Restrictions:

An atom_style must be previously defined to use this command.

Related commands:

lattice, region, create_box, read_data, read_restart

Default:

The default for the *basis* keyword is that all created atoms are assigned the argument *type* as their atom type. The default for *remap* = no and for *units* = lattice.

create_box command

Syntax:

```
create_box N region-ID
```

- N = # of atom types to use in this simulation
- region-ID = ID of region to use as simulation domain

Examples:

```
create_box 2 mybox
```

Description:

This command creates a simulation box based on the specified region. Thus a [region](#) command must first be used to define a geometric domain.

The argument N is the number of atom types that will be used in the simulation.

If the region is not of style *prism*, then LAMMPS encloses the region (block, sphere, etc) with an axis-aligned orthogonal bounding box which becomes the simulation domain.

If the region is of style *prism*, LAMMPS creates a non-orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. As defined by the [region prism](#) command, the parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

A *prism* region used with the create_box command must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

See [Section_howto 12](#) of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

When a prism region is used, the simulation domain must be periodic in any dimensions with a non-zero tilt factor, as defined by the [boundary](#) command. I.e. if the xy tilt factor is non-zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non-zero and y and z must be periodic if yz is non-zero. Also note that if your simulation will tilt the box, e.g. via the [fix deform](#) command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

IMPORTANT NOTE: If the system is non-periodic (in a dimension), then you should not make the lo/hi box dimensions (as defined in your [region](#) command) radically smaller/larger than the extent of the atoms you eventually plan to create, e.g. via the [create_atoms](#) command. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as -10000 and 10000. This is because LAMMPS uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub-optimal for performance when using "fixed" boundary conditions (see the [boundary](#) command). When using "shrink-wrap" boundary conditions (see the [boundary](#) command), a huge (mostly empty) box may cause a parallel simulation to lose

atoms the first time that LAMMPS shrink-wraps the box around the atoms.

Restrictions:

An atom_style and region must have been previously defined to use this command.

Related commands:

create_atoms, region

Default: none

delete_atoms command

Syntax:

```
delete_atoms style args keyword value ...
```

- style = *group* or *region* or *overlap* or *porosity*

```
group args = group-ID
  region args = region-ID
  overlap args = cutoff group1-ID group2-ID
    cutoff = delete one atom from pairs of atoms within the cutoff (distance units)
    group1-ID = one atom in pair must be in this group
    group2-ID = other atom in pair must be in this group
  porosity args = region-ID fraction seed
    region-ID = region within which to perform deletions
    fraction = delete this fraction of atoms
    seed = random number seed (positive integer)
```
- zero or more keyword/value pairs may be appended
- keyword = *compress* or *mol*

```
compress value = no or yes
  mol value = no or yes
```

Examples:

```
delete_atoms group edge
delete_atoms region sphere compress no
delete_atoms overlap 0.3 all all
delete_atoms overlap 0.5 solvent colloid
delete_atoms porosity cube 0.1 482793
```

Description:

Delete the specified atoms. This command can be used to carve out voids from a block of material or to delete created atoms that are too close to each other (e.g. at a grain boundary).

For style *group*, all atoms belonging to the group are deleted.

For style *region*, all atoms in the region volume are deleted. Additional atoms can be deleted if they are in a molecule for which one or more atoms were deleted within the region; see the *mol* keyword discussion below.

For style *overlap* pairs of atoms whose distance of separation is within the specified cutoff distance are searched for, and one of the 2 atoms is deleted. Only pairs where one of the two atoms is in the first group specified and the other atom is in the second group are considered. The atom that is in the first group is the one that is deleted.

Note that it is OK for the two group IDs to be the same (e.g. group *all*), or for some atoms to be members of both groups. In these cases, either atom in the pair may be deleted. Also note that if there are atoms which are members of both groups, the only guarantee is that at the end of the deletion operation, enough deletions will have occurred that no atom pairs within the cutoff will remain (subject to the group restriction). There is no guarantee that the minimum number of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

For style *porosity* a specified *fraction* of atoms are deleted within the specified region. For example, if

fraction is 0.1, then 10% of the atoms will be deleted. The atoms to delete are chosen randomly. There is no guarantee that the exact fraction of atoms will be deleted, or that the same atoms will be deleted when running on different numbers of processors.

If the *compress* keyword is set to *yes*, then after atoms are deleted, then atom IDs are re-assigned so that they run from 1 to the number of atoms in the system. This is not done for molecular systems (see the atom_style command), regardless of the *compress* setting, since it would foul up the bond connectivity that has already been assigned.

It the *mol* keyword is set to *yes*, then for every atom that is deleted, all other atoms in the same molecule will also be deleted. This keyword is only used by the *region* style. It is a way to insure that entire molecules are deleted instead of only a subset of atoms in a bond or angle or dihedral interaction.

Restrictions:

The *overlap* styles requires inter-processor communication to acquire ghost atoms and build a neighbor list. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc). Since a neighbor list is used to find overlapping atom pairs, it also means that you must define a pair style with force cutoffs greater than or equal to the desired overlap cutoff between pairs of relevant atom types, even though the pair potential will not be evaluated.

If the special_bonds command is used with a setting of 0, then a pair of bonded atoms (1-2, 1-3, or 1-4) will not appear in the neighbor list, and thus will not be considered for deletion by the *overlap* styles. You probably don't want to be deleting one atom in a bonded pair anyway.

Related commands:

create_atoms

Default:

The option defaults are compress = yes and mol = no.

delete_bonds command

Syntax:

```
delete_bonds group-ID style args keyword ...
```

- group-ID = group ID
- style = *multi* or *atom* or *bond* or *angle* or *dihedral* or *improper* or *stats*

```
  multi args = none
    atom args = an atom type
    bond args = a bond type
    angle args = an angle type
    dihedral args = a dihedral type
    improper args = an improper type
    stats args = none
```

- zero or more keywords may be appended
- keyword = *any* or *undo* or *remove* or *special*

Examples:

```
delete_bonds frozen multi remove
delete_bonds all atom 4 special
delete_bonds all stats
```

Description:

Turn off (or on) molecular topology interactions, i.e. bonds, angles, dihedrals, impropers. This command is useful for deleting interactions that have been previously turned off by bond-breaking potentials. It is also useful for turning off topology interactions between frozen or rigid atoms. Pairwise interactions can be turned off via the [neigh_modify exclude](#) command. The [fix shake](#) command also effectively turns off certain bond and angle interactions.

For all styles, by default, an interaction is only turned off (or on) if all the atoms involved are in the specified group. See the *any* keyword to change the behavior.

For style *multi* all bond, angle, dihedral, and improper interactions of any type, involving atoms in the group, are turned off.

Style *atom* is the same as style *multi* except that in addition, one or more of the atoms involved in the bond, angle, dihedral, or improper interaction must also be of the specified atom type.

For style *bond*, only bonds are candidates for turn-off, and the bond must also be of the specified type. Styles *angle*, *dihedral*, and *improper* are treated similarly.

For style *bond*, you can set the type to 0 to delete bonds that have been previously broken by a bond-breaking potential (which sets the bond type to 0 when a bond is broken); e.g. see the [bond_style quartic](#) command.

For style *stats* no interactions are turned off (or on); the status of all interactions in the specified group is simply reported. This is useful for diagnostic purposes if bonds have been turned off by a bond-breaking potential during a previous run.

The default behavior of the delete_bonds command is to turn off interactions by toggling their type to a negative value, but not to permanently remove the interaction. E.g. a bond_type of 2 is set to -2. The neighbor

list creation routines will not include such an interaction in their interaction lists. The default is also to not alter the list of 1-2, 1-3, 1-4 neighbors computed by the special_bonds command and used to weight pairwise force and energy calculations. This means that pairwise computations will proceed as if the bond (or angle, etc) were still turned on.

Several keywords can be appended to the argument list to alter the default behaviors.

The *any* keyword changes the requirement that all atoms in the bond (angle, etc) must be in the specified group in order to turn-off the interaction. Instead, if any of the atoms in the interaction are in the specified group, it will be turned off (or on if the *undo* keyword is used).

The *undo* keyword inverts the delete_bonds command so that the specified bonds, angles, etc are turned on if they are currently turned off. This means a negative value is toggled to positive. Note that the fix shake command also sets bond and angle types negative, so this option should not be used on those interactions.

The *remove* keyword is invoked at the end of the delete_bonds operation. It causes turned-off bonds (angles, etc) to be removed from each atom's data structure and then adjusts the global bond (angle, etc) counts accordingly. Removal is a permanent change; removed bonds cannot be turned back on via the *undo* keyword. Removal does not alter the pairwise 1-2, 1-3, 1-4 weighting list.

The *special* keyword is invoked at the end of the delete_bonds operation, after (optional) removal. It re-computes the pairwise 1-2, 1-3, 1-4 weighting list. The weighting list computation treats turned-off bonds the same as turned-on. Thus, turned-off bonds must be removed if you wish to change the weighting list.

Note that the choice of *remove* and *special* options affects how 1-2, 1-3, 1-4 pairwise interactions will be computed across bonds that have been modified by the delete_bonds command.

Restrictions:

This command requires inter-processor communication to coordinate the deleting of bonds. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses set, etc).

If deleted bonds (angles, etc) are removed but the 1-2, 1-3, 1-4 weighting list is not recomputed, this can cause a later fix shake command to fail due to an atom's bonds being inconsistent with the weighting list. This should only happen if the group used in the fix command includes both atoms in the bond, in which case you probably should be recomputing the weighting list.

Related commands:

neigh_modify exclude, special_bonds, fix shake

Default: none

dielectric command

Syntax:

```
dielectric value
```

- value = dielectric constant

Examples:

```
dielectric 2.0
```

Description:

Set the dielectric constant for Coulombic interactions (pairwise and long-range) to this value. The constant is unitless, since it is used to reduce the strength of the interactions. The value is used in the denominator of the formulas for Coulombic interactions - e.g. a value of 4.0 reduces the Coulombic interactions to 25% of their default strength. See the pair_style command for more details.

Restrictions: none

Related commands:

pair_style

Default:

```
dielectric 1.0
```

dihedral_style charmm command

dihedral_style charmm/omp command

Syntax:

```
dihedral_style charmm
```

Examples:

```
dihedral_style charmm
dihedral_coeff 1 120.0 1 60 0.5
```

Description:

The *charmm* dihedral style uses the potential

$$E = K[1 + \cos(n\phi - d)]$$

See [(MacKerell)](#) for a description of the CHARMM force field. This dihedral style can also be used for the AMBER force field (see comment on weighting factors below). See [(Cornell)](#) for a description of the AMBER force field.

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy)
- n (integer >= 0)
- d (integer value of degrees)
- weighting factor (0.0 to 1.0)

The weighting factor is applied to pairwise interaction between the 1st and 4th atoms in the dihedral, which are computed by a CHARMM [pair_style](#) with epsilon and sigma values specified with a [pair_coeff](#) command. Note that this weighting factor is unrelated to the weighting factor specified by the [special_bonds](#) command which applies to all 1-4 interactions in the system.

For CHARMM force fields, the special_bonds 1-4 weighting factor should be set to 0.0. This is because the pair styles that contain "charmm" (e.g. [pair_style lj/charmm/coul/long](#)) define extra 1-4 interaction coefficients that are used by this dihedral style to compute those interactions explicitly. This means that if any of the weighting factors defined as dihedral coefficients (4th coeff above) are non-zero, then you must use a charmm pair style. Note that if you do not set the special_bonds 1-4 weighting factor to 0.0 (which is the default) then 1-4 interactions in dihedrals will be computed twice, once by the pair routine and once by the dihedral routine, which is probably not what you want.

For AMBER force fields, the special_bonds 1-4 weighting factor should be set to the AMBER defaults (1/2 and 5/6) and all the dihedral weighting factors (4th coeff above) should be set to 0.0. In this case, you can use any pair style you wish, since the dihedral does not need any 1-4 information.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in

dihedral_style charmm command   215

Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179-5197 (1995).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem B, 102, 3586 (1998).

dihedral_style class2 command

dihedral_style class2/omp command

Syntax:

```
dihedral_style class2
```

Examples:

```
dihedral_style class2
dihedral_coeff 1 100 75 100 70 80 60
dihedral_coeff * mbt 3.5945 0.1704 -0.5490 1.5228
dihedral_coeff * ebt 0.3417 0.3264 -0.9036 0.1368 0.0 -0.8080 1.0119 1.1010
dihedral_coeff 2 at 0.0 -0.1850 -0.7963 -2.0220 0.0 -0.3991 110.2453 105.1270
dihedral_coeff * aat -13.5271 110.2453 105.1270
dihedral_coeff * bb13 0.0 1.0119 1.1010
```

Description:

The *class2* dihedral style uses the potential

$$
\begin{aligned}
E &= E_d + E_{mbt} + E_{ebt} + E_{at} + E_{aat} + E_{bb13} \\
E_d &= \sum_{n=1}^{3} K_n[1 - \cos(n\phi - \phi_n)] \\
E_{mbt} &= (r_{jk} - r_2)[A_1\cos(\phi) + A_2\cos(2\phi) + A_3\cos(3\phi)] \\
E_{ebt} &= (r_{ij} - r_1)[B_1\cos(\phi) + B_2\cos(2\phi) + B_3\cos(3\phi)] + \\
&\quad (r_{kl} - r_3)[C_1\cos(\phi) + C_2\cos(2\phi) + C_3\cos(3\phi)] \\
E_{at} &= (\theta_{ijk} - \theta_1)[D_1\cos(\phi) + D_2\cos(2\phi) + D_3\cos(3\phi)] + \\
&\quad (\theta_{jkl} - \theta_2)[E_1\cos(\phi) + E_2\cos(2\phi) + E_3\cos(3\phi)] \\
E_{aat} &= M(\theta_{ijk} - \theta_1)(\theta_{jkl} - \theta_2)\cos(\phi) \\
E_{bb13} &= N(r_{ij} - r_1)(r_{kl} - r_3)
\end{aligned}
$$

where Ed is the dihedral term, Embt is a middle-bond-torsion term, Eebt is an end-bond-torsion term, Eat is an angle-torsion term, Eaat is an angle-angle-torsion term, and Ebb13 is a bond-bond-13 term.

Theta1 and theta2 are equilibrium angles and r1 r2 r3 are equilibrium bond lengths.

See [(Sun)](#) for a description of the COMPASS class2 force field.

Coefficients for the Ed, Embt, Eebt, Eat, Eaat, and Ebb13 formulas must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands.

dihedral_style class2 command   217

These are the 6 coefficients for the Ed formula:

- K1 (energy)
- phi1 (degrees)
- K2 (energy)
- phi2 (degrees)
- K3 (energy)
- phi3 (degrees)

For the Embt formula, each line in a [dihedral_coeff](#) command in the input script lists 5 coefficients, the first of which is "mbt" to indicate they are MiddleBondTorsion coefficients. In a data file, these coefficients should be listed under a "MiddleBondTorsion Coeffs" heading and you must leave out the "mbt", i.e. only list 4 coefficients after the dihedral type.

- mbt
- A1 (energy/distance)
- A2 (energy/distance)
- A3 (energy/distance)
- r2 (distance)

For the Eebt formula, each line in a [dihedral_coeff](#) command in the input script lists 9 coefficients, the first of which is "ebt" to indicate they are EndBondTorsion coefficients. In a data file, these coefficients should be listed under a "EndBondTorsion Coeffs" heading and you must leave out the "ebt", i.e. only list 8 coefficients after the dihedral type.

- ebt
- B1 (energy/distance)
- B2 (energy/distance)
- B3 (energy/distance)
- C1 (energy/distance)
- C2 (energy/distance)
- C3 (energy/distance)
- r1 (distance)
- r3 (distance)

For the Eat formula, each line in a [dihedral_coeff](#) command in the input script lists 9 coefficients, the first of which is "at" to indicate they are AngleTorsion coefficients. In a data file, these coefficients should be listed under a "AngleTorsion Coeffs" heading and you must leave out the "at", i.e. only list 8 coefficients after the dihedral type.

- at
- D1 (energy/radian)
- D2 (energy/radian)
- D3 (energy/radian)
- E1 (energy/radian)
- E2 (energy/radian)
- E3 (energy/radian)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of D and E are in energy/radian.

For the Eaat formula, each line in a [dihedral_coeff](#) command in the input script lists 4 coefficients, the first of

dihedral_style class2/omp command

which is "aat" to indicate they are AngleAngleTorsion coefficients. In a data file, these coefficients should be listed under a "AngleAngleTorsion Coeffs" heading and you must leave out the "aat", i.e. only list 3 coefficients after the dihedral type.

- aat
- M (energy/radian^2)
- theta1 (degrees)
- theta2 (degrees)

Theta1 and theta2 are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

For the Ebb13 formula, each line in a dihedral_coeff command in the input script lists 4 coefficients, the first of which is "bb13" to indicate they are BondBond13 coefficients. In a data file, these coefficients should be listed under a "BondBond13 Coeffs" heading and you must leave out the "bb13", i.e. only list 3 coefficients after the dihedral type.

- bb13
- N (energy/distance^2)
- r1 (distance)
- r3 (distance)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the CLASS2 package. See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338-7364 (1998).

dihedral_coeff command

Syntax:

```
dihedral_coeff N args
```

- N = dihedral type (see asterisk form below)
- args = coefficients for one or more dihedral types

Examples:

```
dihedral_coeff 1 80.0 1 3
dihedral_coeff * 80.0 1 3 0.5
dihedral_coeff 2* 80.0 1 3 0.5
```

Description:

Specify the dihedral force field coefficients for one or more dihedral types. The number and meaning of the coefficients depends on the dihedral style. Dihedral coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the coefficients for multiple dihedral types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of dihedral types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using a dihedral_coeff command can override a previous setting for the same dihedral type. For example, these commands set the coeffs for all dihedral types, then overwrite the coeffs for just dihedral type 2:

```
dihedral_coeff * 80.0 1 3
dihedral_coeff 2 200.0 1 3
```

A line in a data file that specifies dihedral coefficients uses the exact same format as the arguments of the dihedral_coeff command in an input script, except that wild-card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Dihedral Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 80.0 1 3
```

The [dihedral_style class2](#) is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross-term coefficients. See its doc page for details.

IMPORTANT NOTE: When comparing the formulas and coefficients for various LAMMPS dihedral styles with dihedral equations defined by other force fields, note that some force field implementations divide/multiply the energy prefactor K by the multiple number of torsions that contain the J-K bond in an I-J-K-L torsion. LAMMPS does not do this, i.e. the listed dihedral equation applies to each individual dihedral. Thus you need to define K appropriately to account for this difference if necessary.

Here is an alphabetic list of dihedral styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated [dihedral_coeff](#) command.

Note that there are also additional dihedral styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the dihedral section of [this page](#).

- [dihedral_style none](#) - turn off dihedral interactions
- [dihedral_style hybrid](#) - define multiple styles of dihedral interactions

- [dihedral_style charmm](#) - CHARMM dihedral
- [dihedral_style class2](#) - COMPASS (class 2) dihedral
- [dihedral_style harmonic](#) - harmonic dihedral
- [dihedral_style helix](#) - helix dihedral
- [dihedral_style multi/harmonic](#) - multi-harmonic dihedral
- [dihedral_style opls](#) - OPLS dihedral

Restrictions:

This command must come after the simulation box is defined by a [read_data](#), [read_restart](#), or [create_box](#) command.

A dihedral style must be defined before any dihedral coefficients are set, either in the input script or in a data file.

Related commands:

[dihedral_style](#)

Default: none

dihedral_style cosine/shift/exp command

dihedral_style cosine/shift/exp/omp command

Syntax:

```
dihedral_style cosine/shift/exp
```

Examples:

```
dihedral_style cosine/shift/exp
dihedral_coeff 1 10.0 45.0 2.0
```

Description:

The *cosine/shift/exp* dihedral style uses the potential

$$E = -U_{min}\frac{e^{-aU(\theta,\theta_0)}-1}{e^a-1} \quad \text{with} \quad U(\theta,\theta_0) = -0.5\left(1+\cos(\theta-\theta_0)\right)$$

where Umin, theta, and a are defined for each dihedral type.

The potential is bounded between [-Umin:0] and the minimum is located at the angle theta0. The a parameter can be both positive or negative and is used to control the spring constant at the equilibrium.

The spring constant is given by k=a exp(a) Umin/ [2 (Exp(a)-1)]. For a>3 k/Umin = a/2 to better than 5% relative error. For negative values of the a parameter, the spring constant is essentially zero, and anharmonic terms takes over. The potential is furthermore well behaved in the limit a->0, where it has been implemented to linear order in a for a < 0.001.

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- umin (energy)
- theta (angle)
- A (real number)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the USER-MISC package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[dihedral_coeff](#), [angle_cosineshiftexp](#)

Default: none

dihedral_style fourier command

dihedral_style fourier/omp command

Syntax:

```
dihedral_style fourier
```

Examples:

```
dihedral_style fourier
dihedral_coeff 1 3 -0.846200 3 0.0 7.578800 1 0 0.138000 2 -180.0
```

Description:

The *fourier* dihedral style uses the potential:

$$E = \sum_{i=1,m} K_i[1.0 + \cos(n_i\phi - d_i)]$$

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- m (integer >=1)
- K1 (energy)
- n1 (integer >= 0)
- d1 (degrees)
- ....
- Km (energy)
- nm (integer >= 0)
- dm (degrees)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[dihedral_coeff](#)

Default: none

dihedral_style harmonic command

dihedral_style harmonic/omp command

Syntax:

```
dihedral_style harmonic
```

Examples:

```
dihedral_style harmonic
dihedral_coeff 1 80.0 1 2
```

Description:

The *harmonic* dihedral style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy)
- d (+1 or -1)
- n (integer >= 0)

IMPORTANT NOTE: Here are important points to take note of when defining LAMMPS dihedral coefficients for the harmonic style, so that they are compatible with how harmonic dihedrals are defined by other force fields:

- The LAMMPS convention is that the trans position = 180 degrees, while in some force fields trans = 0 degrees.
- Some force fields reverse the sign convention on *d*.
- Some force fields let *n* be positive or negative which corresponds to *d* = 1 or -1 for the harmonic style.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the **Making LAMMPS** section for more info on packages.

Related commands:

dihedral_coeff

Default: none

dihedral_style helix command

dihedral_style helix/omp command

Syntax:

```
dihedral_style helix
```

Examples:

```
dihedral_style helix
dihedral_coeff 1 80.0 100.0 40.0
```

Description:

The *helix* dihedral style uses the potential

$$E = A[1 - \cos(\theta)] + B[1 + \cos(3\theta)] + C[1 + \cos(\theta + \frac{\pi}{4})]$$

This coarse-grain dihedral potential is described in (Guo). For dihedral angles in the helical region, the energy function is represented by a standard potential consisting of three minima, one corresponding to the trans (t) state and the other to gauche states (g+ and g-). The paper describes how the A,B,C parameters are chosen so as to balance secondary (largely driven by local interactions) and tertiary structure (driven by long-range interactions).

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy)
- B (energy)
- C (energy)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the **Making LAMMPS** section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Guo) Guo and Thirumalai, Journal of Molecular Biology, 263, 323-43 (1996).

dihedral_style hybrid command

Syntax:

```
dihedral_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more dihedral styles

Examples:

```
dihedral_style hybrid harmonic helix
dihedral_coeff 1 harmonic 6.0 1 3
dihedral_coeff 2* helix 10 10 10
```

Description:

The *hybrid* style enables the use of multiple dihedral styles in one simulation. An dihedral style is assigned to each dihedral type. For example, dihedrals in a polymer flow (of dihedral type 1) could be computed with a *harmonic* potential and dihedrals in the wall boundary (of dihedral type 2) could be computed with a *helix* potential. The assignment of dihedral type to style is made via the [dihedral_coeff](#) command or in the data file.

In the dihedral_coeff commands, the name of a dihedral style must be added after the dihedral type, with the remaining coefficients being those appropriate to that style. In the example above, the 2 dihedral_coeff commands set dihedrals of dihedral type 1 to be computed with a *harmonic* potential with coefficients 6.0, 1, 3 for K, d, n. All other dihedral types (2-N) are computed with a *helix* potential with coefficients 10, 10, 10 for A, B, C.

If dihedral coefficients are specified in the data file read via the [read_data](#) command, then the same rule applies. E.g. "harmonic" or "helix", must be added after the dihedral type, for each line in the "Dihedral Coeffs" section, e.g.

```
Dihedral Coeffs

1 harmonic 6.0 1 3
2 helix 10 10 10
...
```

If *class2* is one of the dihedral hybrid styles, the same rule holds for specifying additional AngleTorsion (and EndBondTorsion, etc) coefficients either via the input script or in the data file. I.e. *class2* must be added to each line after the dihedral type. For lines in the AngleTorsion (or EndBondTorsion, etc) section of the data file for dihedral types that are not *class2*, you must use an dihedral style of *skip* as a placeholder, e.g.

```
AngleTorsion Coeffs

1 skip
2 class2 1.0 1.0 1.0 3.0 3.0 3.0 30.0 50.0
...
```

Note that it is not necessary to use the dihedral style *skip* in the input script, since AngleTorsion (or EndBondTorsion, etc) coefficients need not be specified at all for dihedral types that are not *class2*.

A dihedral style of *none* with no additional coefficients can be used in place of a dihedral style, either in a input script dihedral_coeff command or in the data file, if you desire to turn off interactions for specific dihedral types.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the <u>Making LAMMPS</u> section for more info on packages.

Unlike other dihedral styles, the hybrid dihedral style does not store dihedral coefficient info for individual sub-styles in a <u>binary restart files</u>. Thus when retarting a simulation from a restart file, you need to re-specify dihedral_coeff commands.

Related commands:

<u>dihedral_coeff</u>

Default: none

dihedral_style multi/harmonic command

dihedral_style multi/harmonic/omp command

Syntax:

```
dihedral_style multi/harmonic
```

Examples:

```
dihedral_style multi/harmonic
dihedral_coeff 1 20 20 20 20 20
```

Description:

The *multi/harmonic* dihedral style uses the potential

$$E = \sum_{n=1,5} A_n \cos^{n-1}(\phi)$$

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- A1 (energy)
- A2 (energy)
- A3 (energy)
- A4 (energy)
- A5 (energy)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[dihedral_coeff](#)

Default: none

dihedral_style nharmonic command

dihedral_style nharmonic/omp command

Syntax:

```
dihedral_style nharmonic
```

Examples:

```
dihedral_style nharmonic
dihedral_coeff 3 10.0 20.0 30.0
```

Description:

The *nharmonic* dihedral style uses the potential:

$$ E = \sum_{n=1,n} A_n \cos^{n-1}(\phi) $$

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- n (integer >=1)
- A1 (energy)
- A2 (energy)
- ...
- An (energy)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the [Making LAMMPS](#) section for more info on packages.

Related commands:

dihedral_style nharmonic command 234

[dihedral_coeff](#)

Default: none

dihedral_style none command

Syntax:

```
dihedral_style none
```

Examples:

```
dihedral_style none
```

Description:

Using an dihedral style of none means dihedral forces are not computed, even if quadruplets of dihedral atoms were listed in the data file read by the [read_data](#) command.

Restrictions: none

Related commands: none

Default: none

dihedral_style opls command

dihedral_style opls/omp command

Syntax:

```
dihedral_style opls
```

Examples:

```
dihedral_style opls
dihedral_coeff 1 90.0 90.0 90.0 70.0
```

Description:

The *opls* dihedral style uses the potential

$$ E = \frac{1}{2}K_1[1+\cos(\phi)] + \frac{1}{2}K_2[1-\cos(2\phi)] + \frac{1}{2}K_3[1+\cos(3\phi)] + \frac{1}{2}K_4[1-\cos(4\phi)] $$

Note that the usual 1/2 factor is not included in the K values.

This dihedral potential is used in the OPLS force field and is described in [(Watkins)](#).

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K1 (energy)
- K2 (energy)
- K3 (energy)
- K4 (energy)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the **Making LAMMPS** section for more info on packages.

Related commands:

dihedral_coeff

Default: none

(Watkins) Watkins and Jorgensen, J Phys Chem A, 105, 4118-4125 (2001).

dihedral_style quadratic command

dihedral_style quadratic/omp command

Syntax:

```
dihedral_style quadratic
```

Examples:

```
dihedral_style quadratic
dihedral_coeff 100.0 80.0
```

Description:

The *quadratic* dihedral style uses the potential:

$$E = K(\phi - \phi_0)^2$$

This dihedral potential can be used to keep a dihedral in a predefined value.

The following coefficients must be defined for each dihedral type via the dihedral_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy)
- phi0 (degrees)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

dihedral_style quadratic command 239

Default: none

dihedral_style command

Syntax:

```
dihedral_style style
```

- style = *none* or *hybrid* or *charmm* or *class2* or *harmonic* or *helix* or *multi/harmonic* or *opls*

Examples:

```
dihedral_style harmonic
dihedral_style multi/harmonic
dihedral_style hybrid harmonic charmm
```

Description:

Set the formula(s) LAMMPS uses to compute dihedral interactions between quadruplets of atoms, which remain in force for the duration of the simulation. The list of dihedral quadruplets is read in by a read_data or read_restart command from a data or restart file.

Hybrid models where dihedrals are computed using different dihedral potentials can be setup using the *hybrid* dihedral style.

The coefficients associated with a dihedral style can be specified in a data or restart file or via the dihedral_coeff command.

All dihedral potentials store their coefficient data in binary restart files which means dihedral_style and dihedral_coeff commands do not need to be re-specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that dihedral_style *hybrid* only stores the list of sub-styles in the restart file; dihedral coefficients need to be re-specified.

IMPORTANT NOTE: When both a dihedral and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between 4 bonded atoms.

In the formulas listed for each dihedral style, *phi* is the torsional angle defined by the quadruplet of atoms. This angle has a sign convention as shown in this diagram:
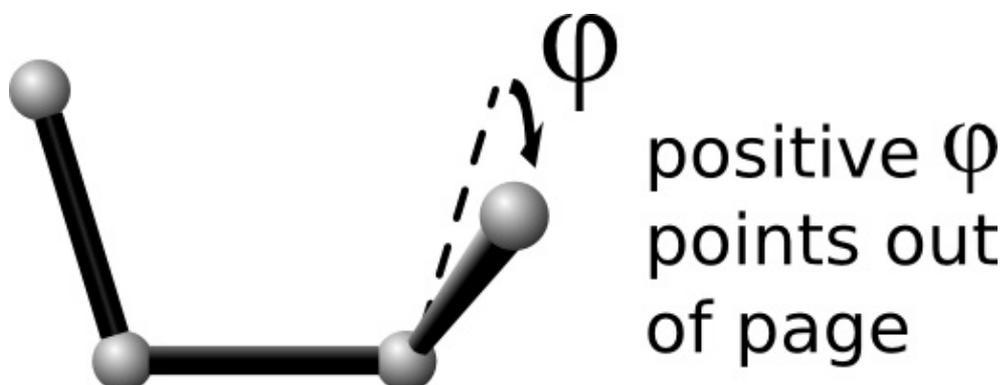
where the I,J,K,L ordering of the 4 atoms that define the dihedral is from left to right.

This sign convention effects several of the dihedral styles listed below (e.g. charmm, helix) in the sense that the energy formula depends on the sign of phi, which may be reflected in the value of the coefficients you specify.

IMPORTANT NOTE: When comparing the formulas and coefficients for various LAMMPS dihedral styles with dihedral equations defined by other force fields, note that some force field implementations divide/multiply the energy prefactor K by the multiple number of torsions that contain the J-K bond in an I-J-K-L torsion. LAMMPS does not do this, i.e. the listed dihedral equation applies to each individual dihedral. Thus you need to define K appropriately via the dihedral_coeff command to account for this difference if necessary.

Here is an alphabetic list of dihedral styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated dihedral_coeff command.

Note that there are also additional dihedral styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the dihedral section of this page.

- dihedral_style none - turn off dihedral interactions
- dihedral_style hybrid - define multiple styles of dihedral interactions

- dihedral_style charmm - CHARMM dihedral
- dihedral_style class2 - COMPASS (class 2) dihedral
- dihedral_style harmonic - harmonic dihedral
- dihedral_style helix - helix dihedral
- dihedral_style multi/harmonic - multi-harmonic dihedral
- dihedral_style opls - OPLS dihedral

Restrictions:

Dihedral styles can only be set for atom styles that allow dihedrals to be defined.

Most dihedral styles are part of the MOLECULAR package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual dihedral potentials tell if it is part of a package.

Related commands:

dihedral_coeff

Default:

dihedral_style none

dihedral_style table command

dihedral_style table/omp command

Syntax:

```
dihedral_style table style Ntable
```

- style = *linear* or *spline* = method of interpolation
- Ntable = size of the internal lookup table

Examples:

```
dihedral_style table spline 400
dihedral_style table linear 1000
dihedral_coeff 1 file.table DIH_TABLE1
dihedral_coeff 2 file.table DIH_TABLE2
```

Description:

The *table* dihedral style creates interpolation tables of length *Ntable* from dihedral potential and derivative values listed in a file(s) as a function of the dihedral angle "phi". The files are read by the [dihedral_coeff](#) command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and derivative values at each of *Ntable* dihedral angles. During a simulation, these tables are used to interpolate energy and force values on individual atoms as needed. The interpolation is done in one of 2 styles: *linear* or *spline*.

For the *linear* style, the dihedral angle (phi) is used to find 2 surrounding table values from which an energy or its derivative is computed by linear interpolation.

For the *spline* style, cubic spline coefficients are computed and stored at each of the *Ntable* evenly-spaced values in the interpolated table. For a given dihedral angle (phi), the appropriate coefficients are chosen from this list, and a cubic polynomial is used to compute the energy and the derivative at this angle.

The following coefficients must be defined for each dihedral type via the [dihedral_coeff](#) command as in the example above.

- filename
- keyword

The filename specifies a file containing tabulated energy and derivative values. The keyword specifies a section of the file. The format of this file is described below.

The format of a tabulated file is as follows (without the parenthesized comments). It can begin with one or more comment or blank lines.

```
# Table of the potential and its negative derivative

DIH_TABLE1                   (keyword is the first text on line)
N 30 DEGREES                 (N, NOF, DEGREES, RADIANS, CHECKU/F)
                             (blank line)
1 -168.0 -1.40351172223 -0.0423346818422
```

```
2 -156.0 -1.70447981034 -0.00811786522531
3 -144.0 -1.62956100432 0.0184129719987
...
30 180.0 -0.707106781187 -0.0719306095245


# Example 2: table of the potential. Forces omitted

DIH_TABLE2
N 30 NOF CHECKU testU.dat CHECKF testF.dat


1 -168.0 -1.40351172223
2 -156.0 -1.70447981034
3 -144.0 -1.62956100432
...
30 180.0 -0.707106781187
```

A section begins with a non-blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the dihedral_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is the angle value, the 3rd value is the energy (in energy units), and the 4th is -dE/d(phi) also in energy units). The 3rd term is the energy of the 4-atom configuration for the specified angle. The 4th term (when present) is the negative derivative of the energy with respect to the angle (in degrees, or radians depending on whether the user selected DEGREES or RADIANS). Thus the units of the last term are still energy, not force. The dihedral angle values must increase from one line to the next.

Dihedral table splines are cyclic. There is no discontinuity at 180 degrees (or at any other angle). Although in the examples above, the angles range from -180 to 180 degrees, in general, the first angle in the list can have any value (positive, zero, or negative). However the *range* of angles represented in the table must be *strictly* less than 360 degrees (2pi radians) to avoid angle overlap. (You may not supply entries in the table for both 180 and -180, for example.) If the user's table covers only a narrow range of dihedral angles, strange numerical behavior can occur in the large remaining gap.

Parameters:

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the N specified in the dihedral_style table command. Let *Ntable* is the number of table entries requested dihedral_style command, and let *Nfile* be the parameter following "N" in the tabulated file ("30" in the sparse example above). What LAMMPS does is a preliminary interpolation by creating splines using the *Nfile* tabulated values as nodal points. It uses these to interpolate as needed to generate energy and derivative values at *Ntable* different points (which are evenly spaced over a 360 degree range, even if the angles in the file are not). The resulting tables of length *Ntable* are then used as described above, when computing energy and force for individual dihedral angles and their atoms. This means that if you want the interpolation tables of length *Ntable* to match exactly what is in the tabulated file (with effectively nopreliminary interpolation), you should set *Ntable = Nfile*. To insure the nodal points in the user's file are aligned with the interpolated table entries, the angles in the table should be integer multiples of 360/*Ntable* degrees, or 2*PI/*Ntable* radians (depending on your choice of angle units).

The optional "NOF" keyword allows the user to omit the forces (negative energy derivatives) from the table file (normally located in the 4th column). In their place, forces will be calculated automatically by differentiating the potential energy function indicated by the 3rd column of the table (using either linear or spline interpolation).

The optional "DEGREES" keyword allows the user to specify angles in degrees instead of radians (default).

The optional "RADIANS" keyword allows the user to specify angles in radians instead of degrees. (Note: This changes the way the forces are scaled in the 4th column of the data file.)

The optional "CHECKU" keyword is followed by a filename. This allows the user to save all of the the *Ntable* different entries in the interpolated energy table to a file to make sure that the interpolated function agrees with the user's expectations. (Note: You can temporarily increase the *Ntable* parameter to a high value for this purpose. "*Ntable*" is explained above.)

The optional "CHECKF" keyword is analogous to the "CHECKU" keyword. It is followed by a filename, and it allows the user to check the interpolated force table. This option is available even if the user selected the "NOF" option.

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This dihedral style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

dihedral_coeff

Default: none

dimension command

Syntax:

```
dimension N
```

- N = 2 or 3

Examples:

```
dimension 2
```

Description:

Set the dimensionality of the simulation. By default LAMMPS runs 3d simulations. To run a 2d simulation, this command should be used prior to setting up a simulation box via the [create_box](#) or [read_data](#) commands. Restart files also store this setting.

See the discussion in [Section_howto](#) for additional instructions on how to run 2d simulations.

IMPORTANT NOTE: Some models in LAMMPS treat particles as finite-size spheres or ellipsoids, as opposed to point particles. In 2d, the particles will still be spheres or ellipsoids, not circular disks or ellipses, meaning their moment of inertia will be the same as in 3d.

Restrictions:

This command must be used before the simulation box is defined by a [read_data](#) or [create_box](#) command.

Related commands:

[fix enforce2d](#)

Default:

```
dimension 3
```

displace_atoms command

Syntax:

```
displace_atoms group-ID style args keyword value ...
```

- group-ID = ID of group of atoms to displace
- style = *move* or *ramp* or *random* or *rotate*

```
  move args = delx dely delz
      delx,dely,delz = distance to displace in each dimension (distance units)
    ramp args = ddim dlo dhi dim clo chi
      ddim = x or y or z
      dlo,dhi = displacement distance between dlo and dhi (distance units)
      dim = x or y or z
      clo,chi = lower and upper bound of domain to displace (distance units)
    random args = dx dy dz seed
      dx,dy,dz = random displacement magnitude in each dimension (distance units)
      seed = random # seed (positive integer)
    rotate args = Px Py Pz Rx Ry Rz theta
      Px,Py,Pz = origin point of axis of rotation (distance units)
      Rx,Ry,Rz = axis of rotation vector
      theta = angle of rotation (degrees)
```

- zero or more keyword/value pairs may be appended

```
      keyword = units
        value = box or lattice
```

Examples:

```
displace_atoms top move 0 -5 0 units box
displace_atoms flow ramp x 0.0 5.0 y 2.0 20.5
```

Description:

Displace a group of atoms. This can be used to move atoms a large distance before beginning a simulation or to randomize atoms initially on a lattice. For example, in a shear simulation, an initial strain can be imposed on the system. Or two groups of atoms can be brought into closer proximity.

The *move* style displaces the group of atoms by the specified 3d distance.

The *ramp* style displaces atoms a variable amount in one dimension depending on the atom's coordinate in a (possibly) different dimension. For example, the second example command displaces atoms in the x-direction an amount between 0.0 and 5.0 distance units. Each atom's displacement depends on the fractional distance its y coordinate is between 2.0 and 20.5. Atoms with y-coordinates outside those bounds will be moved the minimum (0.0) or maximum (5.0) amount.

The *random* style independently moves each atom in the group by a random displacement, uniformly sampled from a value between -dx and +dx in the x dimension, and similarly for y and z. Random numbers are used in such a way that the displacement of a particular atom is the same, regardless of how many processors are being used.

The *rotate* style rotates each atom in the group by the angle *theta* around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The direction of rotation for the atoms around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along R, then your fingers wrap around the axis in the direction of positive theta.

displace_atoms command   247

Distance units for displacements and the origin point of the *rotate* style are determined by the setting of *box* or *lattice* for the *units* keyword. *Box* means distance units as defined by the units command - e.g. Angstroms for *real* units. *Lattice* means distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing.

IMPORTANT NOTE: Care should be taken not to move atoms on top of other atoms. After the move, atoms are remapped into the periodic simulation box if needed, and any shrink-wrap boundary conditions (see the boundary command) are enforced which may change the box size. Other than this effect, this command does not change the size or shape of the simulation box. See the change_box command if that effect is desired.

IMPORTANT NOTE: Atoms can be moved arbitrarily long distances by this command. If the simulation box is non-periodic and shrink-wrapped (see the boundary command), this can change its size or shape. This is not a problem, except that the mapping of processors to the simulation box is not changed by this command from its initial 3d configuration; see the processors command. Thus, if the box size/shape changes dramatically, the mapping of processors to the simulation box may not end up as optimal as the initial mapping attempted to be.

Restrictions:

You cannot rotate around any rotation vector except the z-axis for a 2d simulation.

Related commands:

lattice, change_box, fix_move

Default:

The option defaults are units = lattice.

---

dump command

[dump image](#) command

[dump movie](#) command

[dump molfile](#) command

Syntax:

```
dump ID group-ID style N file args
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be dumped
- style = *atom* or *atom/vtk* or *cfg* or *dcd* or *xtc* or *xyz* or *image* or *molfile* or *local* or *custom* or *mesh/stl*
  or *mesh/vtk* or *mesh/vtk* or *decomposition/vtk* or *euler/vtk*
- N = dump every this many timesteps
- file = name of file to write dump info to
- args = list of arguments for a particular style

```
atom args = none
  atom/vtk args = none
  cfg args = same as custom args, see below
  dcd args = none
  xtc args = none
  xyz args = none

image args = discussed on dump image doc page

molfile args = discussed on dump molfile doc page

mesh/stl args = 'local' or 'ghost' or 'all' or 'region' or any ID of a fix mesh/surface
     region values = ID for region threshold
  mesh/vtk args =  zero or more keyword/ value pairs and one or more dump-identifiers
     keywords = output
     output values = face or interpolate
     dump-identifier = 'stress' or 'id' or 'wear' or 'vel' or 'stresscomponents' or 'owne
  euler/vtk args = none
  decomposition/vtk args = none

local args = list of local attributes
    possible attributes = index, c_ID, c_ID[N], f_ID, f_ID[N]
       index = enumeration of local values
       c_ID = local vector calculated by a compute with ID
       c_ID[N] = Nth column of local array calculated by a compute with ID
       f_ID = local vector calculated by a fix with ID
       f_ID[N] = Nth column of local array calculated by a fix with ID

custom args = list of atom attributes
    possible attributes = id, mol, type, element, mass,
                          x, y, z, xs, ys, zs, xu, yu, zu,
                          xsu, ysu, zsu, ix, iy, iz,
                          vx, vy, vz, fx, fy, fz,
                          q, mux, muy, muz, mu,
                          radius, diameter, omegax, omegay, omegaz,
                          angmomx, angmomy, angmomz, tqx, tqy, tqz,
                          spin, eradius, ervel, erforce,
                          c_ID, c_ID[N], f_ID, f_ID[N], v_name
```

dump command   249

```
id = atom ID
mol = molecule ID
type = atom type
element = name of atom element, as defined by dump_modify command
mass = atom mass
x,y,z = unscaled atom coordinates
xs,ys,zs = scaled atom coordinates
xu,yu,zu = unwrapped atom coordinates
xsu,ysu,zsu = scaled unwrapped atom coordinates
ix,iy,iz = box image that the atom is in
vx,vy,vz = atom velocities
fx,fy,fz = forces on atoms
q = atom charge
mux,muy,muz = orientation of dipole moment of atom
mu = magnitude of dipole moment of atom
radius,diameter = radius,diameter of spherical particle
omegax,omegay,omegaz = angular velocity of spherical particle
angmomx,angmomy,angmomz = angular momentum of aspherical particle
tqx,tqy,tqz = torque on finite-size particles
spin = electron spin
eradius = electron radius
ervel = electron radial velocity
erforce = electron radial force
c_ID = per-atom vector calculated by a compute with ID
c_ID[N] = Nth column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[N] = Nth column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
dump myDump all atom 100 dump.atom
dump 2 subgroup atom 50 dump.run.bin
dump 4a all custom 100 dump.myforce.* id type x y vx fx
dump 4b flow custom 100 dump.%.myforce id type c_myF[3] v_ke
dump 2 inner cfg 10 dump.snap.*.cfg mass type xs ys zs vx vy vz
dump snap all cfg 100 dump.config.*.cfg mass type xs ys zs id type c_Stress2
dump 1 all xtc 1000 file.xtc
dump e_data all custom 100 dump.eff id type x y z spin eradius fx fy fz eforce
```

LIGGGHTS vs. LAMMPS Info:

Two new styles (*mesh/stl* and *mesh/vtk*) are available for dumping granular mesh geometry into STL files or VTK files. The former is used for dumping only the geometry, while the latter command is used to dump the mesh IDs, stress etc- calculated on a granular mesh using the fix mesh/surface/stress command. Furthermore, style *decomposition/vtk* can be used to dump the current parallel domain decomposition to a VTK file. Style *euler/vtk* can be used to dump cell-based averages to a VTK file.

Description:

Dump a snapshot of atom quantities to one or more files every N timesteps in one of several styles. The *image* style is the exception; it creates a JPG or PPM image file of the atom configuration every N timesteps, as discussed on the dump image doc page. The timesteps on which dump output is written can also be controlled by a variable; see the dump_modify every command for details.

Only information for atoms in the specified group is dumped. The dump_modify thresh and region commands can also alter what atoms are included. Not all styles support all these options; see details below.

As described below, the filename determines the kind of output (text or binary or gzipped, one big file or one per timestep, one big file or multiple smaller files).

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

IMPORTANT NOTE: Unless the dump_modify sort option is invoked, the lines of atom information written to dump files (typically one line per atom) will be in an indeterminate order for each snapshot. This is even true when running on a single processor, if the atom_modify sort option is on, which it is by default. In this case atoms are re-ordered periodically during a simulation, due to spatial sorting. It is also true when running in parallel, because data for a single snapshot is collected from multiple processors, each of which owns a subset of the atoms.

For the *atom*, *custom*, *cfg*, and *local* styles, sorting is off by default. For the *dcd*, *xtc*, *xyz*, and *molfile* styles, sorting by atom ID is on by default. See the dump_modify doc page for details.

The *style* keyword determines what atom quantities are written to the file and in what format. Settings made via the dump_modify command can also alter the format of individual values and the file itself.

The *atom*, *local*, and *custom* styles create files in a simple text format that is self-explanatory when viewing a dump file. Many of the LAMMPS post-processing tools, including Pizza.py, work with this format, as does the rerun command.

For post-processing purposes the *atom*, *local*, and *custom* text files are self-describing in the following sense.

The dimensions of the simulation box are included in each snapshot. For an orthogonal simulation box this information is is formatted as:

```
ITEM: BOX BOUNDS xx yy zz
xlo xhi
ylo yhi
zlo zhi
```

where xlo,xhi are the maximum extents of the simulation box in the x-dimension, and similarly for y and z. The "xx yy zz" represent 6 characters that encode the style of boundary for each of the 6 simulation box boundaries (xlo,xhi and ylo,yhi and zlo,zhi). Each of the 6 characters is either p = periodic, f = fixed, s = shrink wrap, or m = shrink wrapped with a minimum value. See the boundary command for details.

For triclinic simulation boxes (non-orthogonal), an orthogonal bounding box which encloses the triclinic simulation box is output, along with the 3 tilt factors (xy, xz, yz) of the triclinic box, formatted as follows:

```
ITEM: BOX BOUNDS xy xz yz xx yy zz
xlo_bound xhi_bound xy
ylo_bound yhi_bound xz
zlo_bound zhi_bound yz
```

The presence of the text "xy xz yz" in the ITEM line indicates that the 3 tilt factors will be included on each of the 3 following lines. This bounding box is convenient for many visualization programs. The meaning of the 6 character flags for "xx yy zz" is the same as above.

Note that the first two numbers on each line are now xlo_bound instead of xlo, etc, since they repesent a bounding box. See this section of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, simple formulas for how the 6 bounding box extents (xlo_bound,xhi_bound,etc) are calculated from the triclinic parameters, and how to transform those parameters to and from other commonly used triclinic representations.

The "ITEM: ATOMS" line in each snapshot lists column descriptors for the per-atom lines that follow. For example, the descriptors would be "id type xs ys zs" for the default *atom* style, and would be the atom attributes you specify in the dump command for the *custom* style.

dump molfile command

For style *atom*, atom coordinates are written to the file, along with the atom ID and atom type. By default, atom coords are written in a scaled format (from 0 to 1). I.e. an x value of 0.25 means the atom is at a location 1/4 of the distance from xlo to xhi of the box boundaries. The format can be changed to unscaled coords via the dump_modify settings. Image flags can also be added for each atom via dump_modify.

For style *atom/vtk*, atom coordinates, velocity, rotational velocity, force, atom ID, atom radius and atom type are written to VTK files. Note that you have to link against VTK libraries to use this functionality.

Style *custom* allows you to specify a list of atom attributes to be written to the dump file for each atom. Possible attributes are listed above and will appear in the order specified. You cannot specify a quantity that is not defined for a particular simulation - such as *q* for atom style *bond*, since that atom style doesn't assign charges. Dumps occur at the very end of a timestep, so atom attributes will include effects due to fixes that are applied during the timestep. An explanation of the possible dump custom attributes is given below.

For style *local*, local output generated by computes and fixes is used to generate lines of output that is written to the dump file. This local data is typically calculated by each processor based on the atoms it owns, but there may be zero or more entities per atom, e.g. a list of bond distances. An explanation of the possible dump local attributes is given below. Note that by using input from the compute property/local command with dump local, it is possible to generate information on bonds, angles, etc that can be cut and pasted directly into a data file read by the read_data command.

Style *cfg* has the same command syntax as style *custom* and writes extended CFG format files, as used by the AtomEye visualization package. Since the extended CFG format uses a single snapshot of the system per file, a wildcard "*" must be included in the filename, as discussed below. The list of atom attributes for style *cfg* must begin with either "mass type xs ys zs" or "mass type xsu ysu zsu" since these quantities are needed to write the CFG files in the appropriate format (though the "mass" and "type" fields do not appear explicitly in the file). Any remaining attributes will be stored as "auxiliary properties" in the CFG files. Note that you will typically want to use the dump_modify element command with CFG-formatted files, to associate element names with atom types, so that AtomEye can render atoms appropriately. When unwrapped coordinates *xsu*, *ysu*, and *zsu* are requested, the nominal AtomEye periodic cell dimensions are expanded by a large factor UNWRAPEXPAND = 10.0, which ensures atoms that are displayed correctly for up to UNWRAPEXPAND/2 periodic boundary crossings in any direction. Beyond this, AtomEye will rewrap the unwrapped coordinates. The expansion causes the atoms to be drawn farther away from the viewer, but it is easy to zoom the atoms closer, and the interatomic distances are unaffected.

The *dcd* style writes DCD files, a standard atomic trajectory format used by the CHARMM, NAMD, and XPlor molecular dynamics packages. DCD files are binary and thus may not be portable to different machines. The number of atoms per snapshot cannot change with the *dcd* style. The *unwrap* option of the dump_modify command allows DCD coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xtc* style writes XTC files, a compressed trajectory format used by the GROMACS molecular dynamics package, and described here. The precision used in XTC files can be adjusted via the dump_modify command. The default value of 1000 means that coordinates are stored to 1/1000 nanometer accuracy. XTC files are portable binary files written in the NFS XDR data format, so that any machine which supports XDR should be able to read them. The number of atoms per snapshot cannot change with the *xtc* style. The *unwrap* option of the dump_modify command allows XTC coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

The *xyz* style writes XYZ files, which is a simple text-based coordinate format that many codes can read.

Specifically it has a line with the number of atoms, then a comment line that is usually ignored followed by one line per atom with the atom type and the x-, y-, and z-coordinate of that atom. You can use the dump_modify element option to change the output from using the (numerical) atom type to an element name (or some other label). This will help many visualization programs to guess bonds and colors.

Note that *atom*, *custom*, *dcd*, *xtc*, and *xyz* style dump files can be read directly by VMD, a popular molecular viewing program. See Section tools of the manual and the tools/lmp2vmd/README.txt file for more information about support in VMD for reading and visualizing LAMMPS dump files.

The *mesh/stl* style dumps active STL geometries defined via fix mesh commands into the specified file. If you do not supply the optional list of mesh IDs, all meshes are dumped, irrespective of whether they are used in a fix wall/gran command or not. By specifying a list of mesh IDs you can explicitly choose which meshes to dump. The group-ID is ignored, because the command is not applied to particles, but to mesh geometries. With keywords 'local', 'owned' or 'ghost' you can decide which parts of the parallel meshes you want to dump (default is 'local'). If the multiprocessor option is not used (no '%' in filename), data is gathered from all processors, so using the default will output the whole mesh data across all processors.

Examples:

```
dump stl1 all mesh/stl 300 post/dump*.stl
dump stl2 all mesh/stl 300 post/dump_proc%_local*.stl local
dump stl3 all mesh/stl 300 post/dump_proc%_ghost*.stl ghost
dump stl4 all mesh/stl 300 post/dump_proc_all_ghost*.stl ghost
```

The first command will write one file per time-step containing the complete mesh. The second command will output one file per time-step per processor containing the local (owned) mesh elements of each processor. The third command will output one file per time-step per processor containing the ghost (corona) mesh elements of each processor. The third command will output one file per time-step containing the ghost (corona) mesh elements of all processors.

With the *region* keyword, just those mesh element where the element center (arithmetic average of all nodes) is in the specified region, will be dumped.

This dump is especially useful if a fix move/mesh is registered. If the position of the mesh is changed over time and you want to dump one file for each dump timestep for post-processing together with the particle data, you should use a filename like 'mydumpfile*.stl'. Note: This series of files can then be post-processed together with the particle dump file converted to VTK in Paraview , www.paraview.org

By providing any ID (or a list of IDs) of fix mesh/surface commands, you can specify which meshes to dump. If no meshes are specified, all meshes used in the simulation are dumped.

The *mesh/vtk* style can be used to dump active mesh geometries defined via fix mesh commands to a series of VTK files. Different keywords can be used to dump the per-triangle stress (force magnitude / element area), id, velocity, wear, stress components (fx / element area, fy / element area, fz / element area), area (area of each element) or the process which owns the element (visulatisation of the parallel decomposition) into the specified file using a VTK file format. The list of mesh IDs is optional. As with the stl style, all active meshes are dumped if you do not supply the optional list of mesh IDs. By specifying list of mesh IDs you can explicitly choose which meshes to dump. The group-ID is ignored. Again, a series of files can be post-processed in Paraview , www.paraview.org Most keywords as used for the *mesh/vtk* style are self-explanatory. Keyword *output* controlls if the data is written in a per-face manner or as interpolated values to VTK. Keywords *aedges* and *acorners* dump the number of active edges/corners per face. Keyword *nneighs* dumps the number of face neighbors LIGGGHTS has recognized for each face.

By providing any ID (or a list of IDs) of fix mesh/surface commands, you can specify which meshes to dump. If no meshes are specified, all meshes used in the simulation are dumped.

dump molfile command   253

The *euler/vtk* style dumps the output of a [fix ave/euler](#) command into a series of VTK files. No further args are expected.

The *decomposition/vtk* style dumps the processor grid decomposition into a series of VTK files. No further args are expected.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the [dump_modify first](#) command, which can also be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the [dump_modify every](#) command (not allowed for *dcd* style). The [dump_modify every](#) command also allows a variable to be used to determine the sequence of timesteps on which dump files are written. In this mode a dump on the first timestep of a run will also not be written unless the [dump_modify first](#) command is used.

The specified filename determines how the dump file(s) is written. The default is to write one large text file, which is opened when the dump command is invoked and closed when an [undump](#) command is used or when LAMMPS exits. For the *dcd* and *xtc* styles, this is a single large binary file.

Dump filenames can contain two wildcard characters. If a "*" character appears in the filename, then one file per snapshot is written and the "*" character is replaced with the timestep value. For example, tmp.dump.* becomes tmp.dump.0, tmp.dump.10000, tmp.dump.20000, etc. This option is not available for the *dcd* and *xtc* styles. Note that the [dump_modify pad](#) command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to read a series of dump files in order with some post-processing tools.

If a "%" character appears in the filename, then each of P processors writes a portion of the dump file, and the "%" character is replaced with the processor ID from 0 to P-1. For example, tmp.dump.% becomes tmp.dump.0, tmp.dump.1, ... tmp.dump.P-1, etc. This creates smaller files and can be a fast mode of output on parallel machines that support parallel I/O for output. This option is not available for the *dcd*, *xtc*, and *xyz* styles.

By default, P = the number of processors meaning one file per processor, but P can be set to a smaller value via the *nfile* or *fileper* keywords of the [dump_modify](#) command. These options can be the most efficient way of writing out dump files when running on large numbers of processors.

Note that using the "*" and "%" characters together can produce a large number of small dump files!

If the filename ends with ".bin", the dump file (or files, if "*" or "%" is also used) is written in binary format. A binary dump file will be about the same size as a text version, but will typically write out much faster. Of course, when post-processing, you will need to convert it back to text format (see the [binary2txt tool](#)) or write your own code to read the binary file. The format of the binary file can be understood by looking at the tools/binary2txt.cpp file. This option is only available for the *atom* and *custom* styles.

If the filename ends with ".gz", the dump file (or files, if "*" or "%" is also used) is written in gzipped format. A gzipped dump file will be about 3x smaller than the text version, but will also take longer to write. This option is not available for the *dcd* and *xtc* styles.

This section explains the local attributes that can be specified as part of the *local* style.

The *index* attribute can be used to generate an index number from 1 to N for each line written into the dump file, where N is the total number of local datums from all processors, or lines of output that will appear in the snapshot. Note that because data from different processors depend on what atoms they currently own, and atoms migrate between processor, there is no guarantee that the same index will be used for the same info

(e.g. a particular bond) in successive snapshots.

The *c_ID* and *c_ID[N]* attributes allow local vectors or arrays calculated by a [compute](#) to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the [compute](#) command for details. There are computes for calculating local information such as indices, types, and energies for bonds and angles.

Note that computes which calculate global or per-atom quantities, as opposed to local quantities, cannot be output in a dump local command. Instead, global quantities can be output by the [thermo_style custom](#) command, and per-atom quantities can be output by the dump custom command.

If *c_ID* is used as a attribute, then the local vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length local array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow local vectors or arrays calculated by a [fix](#) to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script.

If *f_ID* is used as a attribute, then the local vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length local array calculated by the fix.

Here is an example of how to dump bond info for a system, including the distance and energy of each bond:

```
compute 1 all property/local batom1 batom2 btype
compute 2 all bond/local dist eng
dump 1 all local 1000 tmp.dump index c_1[1] c_1[2] c_1[3] c_2[1] c_2[2]
```

This section explains the atom attributes that can be specified as part of the *custom* and *cfg* styles.

The *id*, *mol*, *type*, *element*, *mass*, *vx*, *vy*, *vz*, *fx*, *fy*, *fz*, *q* attributes are self-explanatory.

Id is the atom ID. *Mol* is the molecule ID, included in the data file for molecular systems. *Type* is the atom type. *Element* is typically the chemical name of an element, which you must assign to each type via the [dump_modify element](#) command. More generally, it can be any string you wish to associated with an atom type. *Mass* is the atom mass. *Vx*, *vy*, *vz*, *fx*, *fy*, *fz*, and *q* are components of atom velocity and force and atomic charge.

There are several options for outputting atom coordinates. The *x*, *y*, *z* attributes write atom coordinates "unscaled", in the appropriate distance [units](#) (Angstroms, sigma, etc). Use *xs*, *ys*, *zs* if you want the coordinates "scaled" to the box size, so that each value is 0.0 to 1.0. If the simulation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. Use *xu*, *yu*, *zu* if you want the coordinates "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that using *xu*, *yu*, *zu* means that the coordinate values may be far outside the box bounds printed with the snapshot. Using *xsu*, *ysu*, *zsu* is similar to using *xu*, *yu*, *zu*, except that the unwrapped coordinates are scaled by the box size. Atoms that have passed through a periodic boundary will have the corresponding cooordinate increased or decreased by 1.0.

The image flags can be printed directly using the *ix*, *iy*, *iz* attributes. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation.

dump molfile command

The *mux*, *muy*, *muz* attributes are specific to dipolar systems defined with an atom style of *dipole*. They give the orientation of the atom's point dipole moment. The *mu* attribute gives the magnitude of the atom's dipole moment.

The *radius* and *diameter* attributes are specific to spherical particles that have a finite size, such as those defined with an atom style of *sphere*.

The *omegax*, *omegay*, and *omegaz* attributes are specific to finite-size spherical particles that have an angular velocity. Only certain atom styles, such as *sphere* define this quantity.

The *angmomx*, *angmomy*, and *angmomz* attributes are specific to finite-size aspherical particles that have an angular momentum. Only the *ellipsoid* atom style defines this quantity.

The *tqx*, *tqy*, *tqz* attributes are for finite-size particles that can sustain a rotational torque due to interactions with other particles.

The *spin*, *eradius*, *ervel*, and *erforce* attributes are for particles that represent nuclei and electrons modeled with the electronic force field (EFF). See atom_style electron and pair_style eff for more details.

The *c_ID* and *c_ID[N]* attributes allow per-atom vectors or arrays calculated by a compute to be output. The ID in the attribute should be replaced by the actual ID of the compute that has been defined previously in the input script. See the compute command for details. There are computes for calculating the per-atom energy, stress, centro-symmetry parameter, and coordination number of individual atoms.

Note that computes which calculate global or local quantities, as opposed to per-atom quantities, cannot be output in a dump custom command. Instead, global quantities can be output by the thermo_style custom command, and local quantities can be output by the dump local command.

If *c_ID* is used as a attribute, then the per-atom vector calculated by the compute is printed. If *c_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the compute.

The *f_ID* and *f_ID[N]* attributes allow vector or array per-atom quantities calculated by a fix to be output. The ID in the attribute should be replaced by the actual ID of the fix that has been defined previously in the input script. The fix ave/atom command is one that calculates per-atom quantities. Since it can time-average per-atom quantities produced by any compute, fix, or atom-style variable, this allows those time-averaged results to be written to a dump file.

If *f_ID* is used as a attribute, then the per-atom vector calculated by the fix is printed. If *f_ID[N]* is used, then N must be in the range from 1-M, which will print the Nth column of the M-length per-atom array calculated by the fix.

The *v_name* attribute allows per-atom vectors calculated by a variable to be output. The name in the attribute should be replaced by the actual name of the variable that has been defined previously in the input script. Only an atom-style variable can be referenced, since it is the only style that generates per-atom values. Variables of style *atom* can reference individual atom attributes, per-atom atom attributes, thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of creating quantities to output to a dump file.

See Section_modify of the manual for information on how to add new compute and fix styles to LAMMPS to calculate per-atom quantities which could then be output into dump files.

Restrictions:

To write gzipped dump files, you must compile LAMMPS with the -DLAMMPS_GZIP option - see the Making LAMMPS section of the documentation.

To be able to use *atom/vtk*, you have to link to VTK libraries, please adapt your Makefile accordingly.

The *xtc* style is part of the XTC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info. This is because some machines may not support the low-level XDR data format that XTC files are written with, which will result in a compile-time error when a low-level include file is not found. Putting this style in a package makes it easy to exclude from a LAMMPS build for those machines. However, the XTC package also includes two compatibility header files and associated functions, which should be a suitable substitute on machines that do not have the appropriate native header files. This option can be invoked at build time by adding -DLAMMPS_XDR to the CCFLAGS variable in the appropriate low-level Makefile, e.g. src/MAKE/Makefile.foo. This compatibility mode has been tested successfully on Cray XT3/XT4/XT5 and IBM BlueGene/L machines and should also work on IBM BG/P, and Windows XP/Vista/7 machines.

Related commands:

dump image, dump_modify, undump

Default:

The defaults for the image style are listed on the dump image doc page.

dump image command

dump movie command

Syntax:

```
dump ID group-ID style N file color diameter keyword value ...
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- style = *image* or *movie* = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the [dump](#) doc page)
- N = dump every this many timesteps
- file = name of file to write image to
- color = atom attribute that determines color of each atom
- diameter = atom attribute that determines size of each atom
- zero or more keyword/value pairs may be appended
- keyword = *adiam* or *atom* or *bond* or *size* or *view* or *center* or *up* or *zoom* or *persp* or *box* or *axes* or *shiny* or *ssao*

```
  adiam value = number = numeric value for atom diameter (distance units)
    atom = yes/no = do or do not draw atoms
    bond values = color width = color and width of bonds
      color = atom or type or none
      width = number or atom or type or none
        number = numeric value for bond width (distance units)
    size values = width height = size of images
      width = width of image in # of pixels
      height = height of image in # of pixels
    view values = theta phi = view of simulation box
      theta = view angle from +z axis (degrees)
      phi = azimuthal view angle (degrees)
      theta or phi can be a variable (see below)
    center values = flag Cx Cy Cz = center point of image
      flag = "s" for static, "d" for dynamic
      Cx,Cy,Cz = center point of image as fraction of box dimension (0.5 = center of box)
      Cx,Cy,Cz can be variables (see below)
    up values = Ux Uy Uz = direction that is "up" in image
      Ux,Uy,Uz = components of up vector
      Ux,Uy,Uz can be variables (see below)
    zoom value = zfactor = size that simulation box appears in image
      zfactor = scale image size by factor > 1 to enlarge, factor <1 to shrink
      zfactor can be a variable (see below)
    persp value = pfactor = amount of "perspective" in image
      pfactor = amount of perspective (0 = none,  <1 = some, > 1 = highly skewed)
      pfactor can be a variable (see below)
    box values = yes/no diam = draw outline of simulation box
      yes/no = do or do not draw simulation box lines
      diam = diameter of box lines as fraction of shortest box length
    axes values = yes/no length diam = draw xyz axes
      yes/no = do or do not draw xyz axes lines next to simulation box
      length = length of axes lines as fraction of respective box lengths
      diam = diameter of axes lines as fraction of shortest box length
    shiny value = sfactor = shinyness of spheres and cylinders
      sfactor = shinyness of spheres and cylinders from 0.0 to 1.0
    ssao value = yes/no seed dfactor = SSAO depth shading
      yes/no = turn depth shading on/off
      seed = random # seed (positive integer)
      dfactor = strength of shading from 0.0 to 1.0
```

Examples:

```
dump d0 all image 100 dump.*.jpg type type
dump d1 mobile image 500 snap.*.png element element ssao yes 4539 0.6
dump d2 all image 200 img-*.ppm type type zoom 2.5 adiam 1.5 size 1280 720
dump m0 all movie 1000 movie.mpg type type size 640 480
dump m1 all movie 1000 movie.avi type type size 640 480
dump m2 all movie 100 movie.m4v type type zoom 1.8 adiam v_value size 1280 720
```

Description:

Dump a high-quality rendered image of the atom configuration every N timesteps and save the images either as a sequence of JPG or PNG, or PPM files, or as a single movie file. The options for this command as well as the dump_modify command control what is included in the image or movie and how it appears. A series of such images can easily be manually converted into an animated movie of your simulation or the process can be automated without writing the intermediate files using the dump movie style; see further details below. Other dump styles store snapshots of numerical data asociated with atoms in various formats, as discussed on the dump doc page.

Note that a set of images or a movie can be made after a simulation has been run, using the rerun command to read snapshots from an existing dump file, and using these dump commands in the rerun script to generate the images/movie.

Here are two sample images, rendered as 1024x1024 JPG files. Click to see the full-size images:
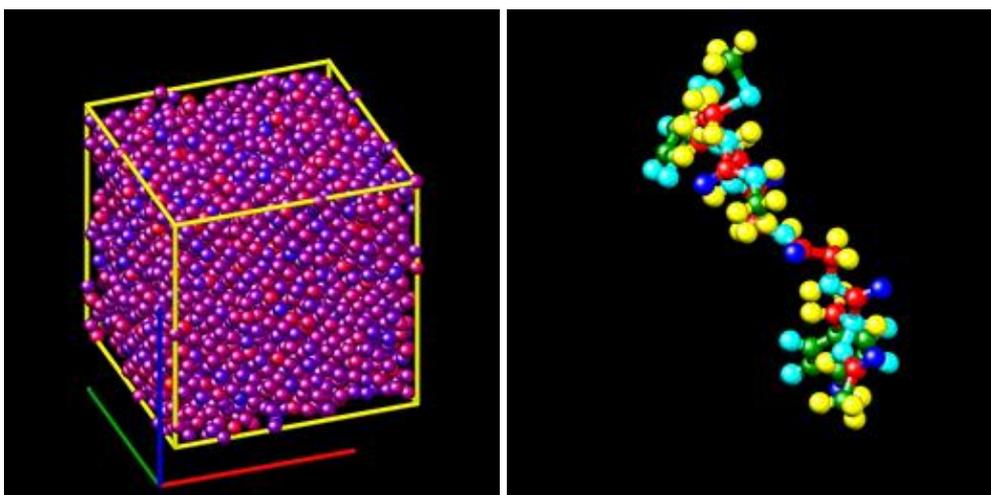
Only atoms in the specified group are rendered in the image. The dump_modify region and thresh commands can also alter what atoms are included in the image.

The filename suffix determines whether a JPEG, PNG, or PPM file is created with the *image* dump style. If the suffix is ".jpg" or ".jpeg", then a JPEG format file is created, if the suffix is ".png", then a PNG format is created, else a PPM (aka NETPBM) format file is created. The JPG and PNG files are binary; PPM has a text mode header followed by binary data. JPG images have lossy compression; PNG has lossless compression; and PPM files are uncompressed but can be compressed with gzip, if LAMMPS has been compiled with -DLAMMPS_GZIP and a ".gz" suffix is used.

Similarly, the format of the resulting movie is chosen with the *movie* dump style. This is handled by the underlying FFmpeg converter and thus details have to be looked up in the FFmpeg documentation. Typical examples are: .avi, .mpg, .m4v, .mp4, .mkv, .flv, .mov, .gif Additional settings of the movie compression like bitrate and framerate can be set using the dump_modify command.

To write out JPEG and PNG format files, you must build LAMMPS with support for the corresponding JPEG or PNG library. To convert images into movies, LAMMPS has to be compiled with the

dump movie command

-DLAMMPS_FFMPEG flag. See this section of the manual for instructions on how to do this.

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom in the image may be slightly outside the simulation box.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command.

Dump *image* filenames must contain a wildcard character "*", so that one image file per snapshot is written. The "*" character is replaced with the timestep value. For example, tmp.dump.*.jpg becomes tmp.dump.0.jpg, tmp.dump.10000.jpg, tmp.dump.20000.jpg, etc. Note that the dump_modify pad command can be used to insure all timestep numbers are the same length (e.g. 00010), which can make it easier to convert a series of images into a movie in the correct ordering.

Dump *movie* filenames on the other hand, must not have any wildcard character since only one file combining all images into a single movie will be written by the movie encoder.

The *color* and *diameter* settings determine the color and size of atoms rendered in the image. They can be any atom attribute defined for the dump custom command, including *type* and *element*. This includes per-atom quantities calculated by a compute, fix, or variable, which are prefixed by "c_", "f_", or "v_" respectively. Note that the *diameter* setting can be overridden with a numeric value by the optional *adiam* keyword, in which case you can specify the *diameter* setting with any valid atom attribute.

If *type* is specified for the *color* setting, then the color of each atom is determined by its atom type. By default the mapping of types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for types > 6. This mapping can be changed by the dump_modify acolor command.

If *type* is specified for the *diameter* setting then the diameter of each atom is determined by its atom type. By default all types have diameter 1.0. This mapping can be changed by the dump_modify adiam command.

If *element* is specified for the *color* and/or *diameter* setting, then the color and/or diameter of each atom is determined by which element it is, which in turn is specified by the element-to-type mapping specified by the "dump_modify element" command. By default every atom type is C (carbon). Every element has a color and diameter associated with it, which is the same as the colors and sizes used by the AtomEye visualization package.

If other atom attributes are used for the *color* or *diameter* settings, they are interpreted in the following way.

If "vx", for example, is used as the *color* setting, then the color of the atom will depend on the x-component of its velocity. The association of a per-atom value with a specific color is determined by a "color map", which can be specified via the dump_modify command. The basic idea is that the atom-attribute will be within a range of values, and every value within the range is mapped to a specific color. Depending on how the color

map is defined, that mapping can take place via interpolation so that a value of -3.2 is halfway between "red" and "blue", or discretely so that the value of -3.2 is "orange".

If "vx", for example, is used as the *diameter* setting, then the atom will be rendered using the x-component of its velocity as the diameter. If the per-atom value <= 0.0, them the atom will not be drawn. Note that finite-size spherical particles, as defined by atom_style sphere define a per-particle radius or diameter, which can be used as the *diameter* setting.

The various kewords listed above control how the image is rendered. As listed below, all of the keywords have defaults, most of which you will likely not need to change. The dump modify also has options specific to the dump image style, particularly for assigning colors to atoms, bonds, and other image features.

The *adiam* keyword allows you to override the *diameter* setting to a per-atom attribute with a specified numeric value. All atoms will be drawn with that diameter, e.g. 1.5, which is in whatever distance units the input script defines, e.g. Angstroms.

The *atom* keyword allow you to turn off the drawing of all atoms, if the specified value is *no*.

The *bond* keyword allows to you to alter how bonds are drawn. A bond is only drawn if both atoms in the bond are being drawn due to being in the specified group and due to other selection criteria (e.g. region, threshhold settings of the dump_modify command). By default, bonds are drawn if they are defined in the input data file as read by the read_data command. Using *none* for both the bond *color* and *width* value will turn off the drawing of all bonds.

If *atom* is specified for the bond *color* value, then each bond is drawn in 2 halves, with the color of each half being the color of the atom at that end of the bond.

If *type* is specified for the *color* value, then the color of each bond is determined by its bond type. By default the mapping of bond types to colors is as follows:

- type 1 = red
- type 2 = green
- type 3 = blue
- type 4 = yellow
- type 5 = aqua
- type 6 = cyan

and repeats itself for bond types > 6. This mapping can be changed by the dump_modify bcolor command.

The bond *width* value can be a numeric value or *atom* or *type* (or *none* as indicated above).

If a numeric value is specified, then all bonds will be drawn as cylinders with that diameter, e.g. 1.0, which is in whatever distance units the input script defines, e.g. Angstroms.

If *atom* is specified for the *width* value, then each bond will be drawn with a width corresponding to the minimum diameter of the 2 atoms in the bond.

If *type* is specified for the *width* value then the diameter of each bond is determined by its bond type. By default all types have diameter 0.5. This mapping can be changed by the dump_modify bdiam command.

The *size* keyword sets the width and height of the created images, i.e. the number of pixels in each direction.

The *view*, *center*, *up*, *zoom*, and *persp* values determine how 3d simulation space is mapped to the 2d plane of the image. Basically they control how the simulation box appears in the image.

dump movie command   261

All of the *view*, *center*, *up*, *zoom*, and *persp* values can be specified as numeric quantities, whose meaning is explained below. Any of them can also be specified as an equal-style variable, by using v_name as the value, where "name" is the variable name. In this case the variable will be evaluated on the timestep each image is created to create a new value. If the equal-style variable is time-dependent, this is a means of changing the way the simulation box appears from image to image, effectively doing a pan or fly-by view of your simulation.

The *view* keyword determines the viewpoint from which the simulation box is viewed, looking towards the *center* point. The *theta* value is the vertical angle from the +z axis, and must be an angle from 0 to 180 degrees. The *phi* value is an azimuthal angle around the z axis and can be positive or negative. A value of 0.0 is a view along the +x axis, towards the *center* point. If *theta* or *phi* are specified via variables, then the variable values should be in degrees.

The *center* keyword determines the point in simulation space that will be at the center of the image. *Cx*, *Cy*, and *Cz* are speficied as fractions of the box dimensions, so that (0.5,0.5,0.5) is the center of the simulation box. These values do not have to be between 0.0 and 1.0, if you want the simulation box to be offset from the center of the image. Note, however, that if you choose strange values for *Cx*, *Cy*, or *Cz* you may get a blank image. Internally, *Cx*, *Cy*, and *Cz* are converted into a point in simulation space. If *flag* is set to "s" for static, then this conversion is done once, at the time the dump command is issued. If *flag* is set to "d" for dynamic then the conversion is performed every time a new image is created. If the box size or shape is changing, this will adjust the center point in simulation space.

The *up* keyword determines what direction in simulation space will be "up" in the image. Internally it is stored as a vector that is in the plane perpendicular to the view vector implied by the *theta* and *pni* values, and which is also in the plane defined by the view vector and user-specified up vector. Thus this internal vector is computed from the user-specified *up* vector as

```
up_internal = view cross (up cross view)
```

This means the only restriction on the specified *up* vector is that it cannot be parallel to the *view* vector, implied by the *theta* and *phi* values.

The *zoom* keyword scales the size of the simulation box as it appears in the image. The default *zfactor* value of 1 should display an image mostly filled by the atoms in the simulation box. A *zfactor* > 1 will make the simulation box larger; a *zfactor* < 1 will make it smaller. *Zfactor* must be a value > 0.0.

The *persp* keyword determines how much depth perspective is present in the image. Depth perspective makes lines that are parallel in simulation space appear non-parallel in the image. A *pfactor* value of 0.0 means that parallel lines will meet at infinty (1.0/pfactor), which is an orthographic rendering with no persepctive. A *pfactor* value between 0.0 and 1.0 will introduce more perspective. A *pfactor* value > 1 will create a highly skewed image with a large amount of perspective.

IMPORTANT NOTE: The *persp* keyword is not yet supported as an option.

The *box* keyword determines how the simulation box boundaries are rendered as thin cylinders in the image. If *no* is set, then the box boundaries are not drawn and the *diam* setting is ignored. If *yes* is set, the 12 edges of the box are drawn, with a diameter that is a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d). The color of the box boundaries can be set with the dump_modify boxcolor command.

The *axes* keyword determines how the coordinate axes are rendered as thin cylinders in the image. If *no* is set, then the axes are not drawn and the *length* and *diam* settings are ignored. If *yes* is set, 3 thin cylinders are drawn to represent the x,y,z axes in colors red,green,blue. The origin of these cylinders will be offset from the lower left corner of the box by 10%. The *length* setting determines how long the cylinders will be as a fraction of the respective box lengths. The *diam* setting determines their thickness as a fraction of the shortest box length in x,y,z (for 3d) or x,y (for 2d).

The *shiny* keyword determines how shiny the objects rendered in the image will appear. The *sfactor* value must be a value 0.0 <= *sfactor* <= 1.0, where *sfactor* = 1 is a highly reflective surface and *sfactor* = 0 is a rough non-shiny surface.

The *ssao* keyword turns on/off a screen space ambient occlusion (SSAO) model for depth shading. If *yes* is set, then atoms further away from the viewer are darkened via a randomized process, which is perceived as depth. The calculation of this effect can increase the cost of computing the image by roughly 2x. The strength of the effect can be scaled by the *dfactor* parameter. If *no* is set, no depth shading is performed.

A series of JPG, PNG, or PPM images can be converted into a movie file and then played as a movie using commonly available tools. Using dump style *movie* automates this step and avoids the intermediate step of writing (many) image snapshot file. But LAMMPS has to be compiled with -DLAMMPS_FFMPEG and an FFmpeg executable have to be installed.

To manually convert JPG, PNG or PPM files into an animated GIF or MPEG or other movie file you can use:

- a) Use the ImageMagick convert program.

  ```
  % convert *.jpg foo.gif
  % convert -loop 1 *.ppm foo.mpg
  ```

  Animated GIF files from ImageMagick are unoptimized. You can use a program like gifsicle to optimize and massively shrink them. MPEG files created by ImageMagick are in MPEG-1 format with rather inefficient compression and low quality.
- b) Use QuickTime.

  Select "Open Image Sequence" under the File menu Load the images into QuickTime to animate them Select "Export" under the File menu Save the movie as a QuickTime movie (*.mov) or in another format. QuickTime can generate very high quality and efficiently compressed movie files. Some of the supported formats require to buy a license and some are not readable on all platforms until specific runtime libraries are installed.
- c) Use FFmpeg

FFmpeg is a command line tool that is available on many platforms and allows extremely flexible encoding and decoding of movies.

```
cat snap.*.jpg | ffmpeg -y -f image2pipe -c:v mjpeg -i - -b:v 2000k movie.m4v
cat snap.*.ppm | ffmpeg -y -f image2pipe -c:v ppm -i - -b:v 2400k movie.avi
```

Frontends for FFmpeg exist for multiple platforms. For more information see the [FFmpeg homepage](#)

Play the movie:

- a) Use your browser to view an animated GIF movie.

  Select "Open File" under the File menu Load the animated GIF file
- b) Use the freely available mplayer or ffplay tool to view a movie. Both are available for multiple OSes and support a large variety of file formats and decoders.

  ```
  % mplayer foo.mpg
  % ffplay bar.avi
  ```
- c) Use the [Pizza.py](#) [animate tool](#), which works directly on a series of image files.

  ```
  a = animate("foo*.jpg")
  ```

dump movie command   263

- d) QuickTime and other Windows- or MacOS-based media players can obviously play movie files directly. Similarly the corresponding tools bundled with Linux desktop environments, however, due to licensing issues of some of the file formats, some formats may require installing additional libraries, purchasing a license, or are not supported.

See Section_modify of the manual for information on how to add new compute and fix styles to LAMMPS to calculate per-atom quantities which could then be output into dump files.

Restrictions:

To write JPG images, you must use the -DLAMMPS_JPEG switch when building LAMMPS and link with a JPEG library. To write PNG images, you must use the -DLAMMPS_PNG switch when building LAMMPS and link with a PNG library.

To write *movie* dumps, you must use the -DLAMMPS_FFMPEG switch when building LAMMPS and have the FFmpeg executable available on the machine where LAMMPS is being run.

See the Making LAMMPS section of the documentation for details on how to configure and compile optional in LAMMPS.

Related commands:

dump, dump_modify, undump

Default:

The defaults for the keywords are as follows:

- adiam = not specified (use diameter setting)
- atom = yes
- bond = none none (if no bonds in system)
- bond = atom 0.5 (if bonds in system)
- size = 512 512
- view = 60 30 (for 3d)
- view = 0 0 (for 2d)
- center = s 0.5 0.5 0.5
- up = 0 0 1 (for 3d)
- up = 0 1 0 (for 2d)
- zoom = 1.0
- persp = 0.0
- box = yes 0.02
- axes = no 0.0 0.0
- shiny = 1.0
- ssao = no

dump_modify command

Syntax:

```
dump_modify dump-ID keyword values ...
```

- dump-ID = ID of dump to modify
- one or more keyword/value pairs may be appended
- these keywords apply to various dump styles
- keyword = *append* or *buffer* or *element* or *every* or *fileper* or *first* or *flush* or *format* or *image* or *label* or *nfile* or *pad* or *precision* or *region* or *scale* or *sort* or *thresh* or *unwrap*

```
append arg = yes or no
  buffer arg = yes or no
  element args = E1 E2 ... EN, where N = # of atom types
    E1,...,EN = element name, e.g. C or Fe or Ga
  every arg = N
    N = dump every this many timesteps
    N can be a variable (see below)
  fileper arg = Np
    Np = write one file for every this many processors
  first arg = yes or no
  format arg = C-style format string for one line of output
  flush arg = yes or no
  image arg = yes or no
  label arg = string
    string = character string (e.g. BONDS) to use in header of dump local file
  nfile arg = Nf
    Nf = write this many files, one from each of Nf processors
  pad arg = Nchar = # of characters to convert timestep to
  precision arg = power-of-10 value from 10 to 1000000
  region arg = region-ID or "none"
  scale arg = yes or no
  sort arg = off or id or N or -N
      off = no sorting of per-atom lines within a snapshot
      id = sort per-atom lines by atom ID
      N = sort per-atom lines in ascending order by the Nth column
      -N = sort per-atom lines in descending order by the Nth column
  thresh args = attribute operation value
    attribute = same attributes (x,fy,etotal,sxx,etc) used by dump custom style
    operation = "" or ">=" or "==" or "!="
    value = numeric value to compare to
    these 3 args can be replaced by the word "none" to turn off thresholding
  unwrap arg = yes or no
```

- these keywords apply only to the *image* and *movie* styles
- keyword = *acolor* or *adiam* or *amap* or *bcolor* or *bdiam* or *backcolor* or *boxcolor* or *color* or *bitrate* or *framerate*

```
acolor args = type color
    type = atom type or range of types (see below)
    color = name of color or color1/color2/...
  adiam args = type diam
    type = atom type or range of types (see below)
    diam = diameter of atoms of that type (distance units)
  amap args = lo hi style delta N entry1 entry2 ... entryN
    lo = number or min = lower bound of range of color map
    hi = number or max = upper bound of range of color map
    style = 2 letters = "c" or "d" or "s" plus "a" or "f"
      "c" for continuous
      "d" for discrete
      "s" for sequential
```

```
       "a" for absolute
       "f" for fractional
     delta = binsize (only used for style "s", otherwise ignored)
       binsize = range is divided into bins of this width
     N = # of subsequent entries
     entry = value color (for continuous style)
       value = number or min or max = single value within range
       color = name of color used for that value
     entry = lo hi color (for discrete style)
       lo/hi = number or min or max = lower/upper bound of subset of range
       color = name of color used for that subset of values
     entry = color (for sequential style)
       color = name of color used for a bin of values
   backcolor arg = color
     color = name of color for background
   bcolor args = type color
     type = bond type or range of types (see below)
     color = name of color or color1/color2/...
   bdiam args = type diam
     type = bond type or range of types (see below)
     diam = diameter of bonds of that type (distance units)
   bitrate arg = rate
     rate = target bitrate for movie in kbps
   boxcolor arg = color
     color = name of color for box lines
   color args = name R G B
     name = name of color
     R,G,B = red/green/blue numeric values from 0.0 to 1.0
   framerate arg = fps
     fps = frames per second for movie
```

Examples:

```
dump_modify 1 format "%d %d %20.15g %g %g" scale yes
dump_modify myDump image yes scale no flush yes
dump_modify 1 region mySphere thresh x <0.0 thresh epair >= 3.2
dump_modify xtcdump precision 10000
dump_modify 1 every 1000 nfile 20
dump_modify 1 every v_myVar
dump_modify 1 amap min max cf 0.0 3 min green 0.5 yellow max blue boxcolor red
```

Description:

Modify the parameters of a previously defined dump command. Not all parameters are relevant to all dump styles.

These keywords apply to various dump styles, including the dump image and dump movie styles. The description gives details.

The *append* keyword applies to all dump styles except *cfg* and *xtc* and *dcd*. It also applies only to text output files, not to binary or gzipped or image/movie files. If specified as *yes*, then dump snapshots are appended to the end of an existing dump file. If specified as *no*, then a new dump file will be created which will overwrite an existing file with the same name. This keyword can only take effect if the dump_modify command is used after the dump command, but before the first command that causes dump snapshots to be output, e.g. a run or minimize command. Once the dump file has been opened, this keyword has no further effect.

The *buffer* keyword applies only to dump styles *atom*, *custom*, *local*, and *xyz*. It also applies only to text output files, not to binary or gzipped files. If specified as *yes*, which is the default, then each processor writes its output into an internal text buffer, which is then sent to the processor(s) which perform file writes, and written by those processors(s) as one large chunk of text. If specified as *no*, each processor sends its per-atom

data in binary format to the processor(s) which perform file wirtes, and those processor(s) format and write it line by line into the output file.

The buffering mode is typically faster since each processor does the relatively expensive task of formatting the output for its own atoms. However it requires about twice the memory (per processor) for the extra buffering.

The *element* keyword applies only to the the dump *cfg*, *xyz*, and *image* styles. It associates element names (e.g. H, C, Fe) with LAMMPS atom types. See the list of element names at the bottom of this page.

In the case of dump *cfg*, this allows the AtomEye visualization package to read the dump file and render atoms with the appropriate size and color.

In the case of dump *image*, the output images will follow the same AtomEye convention. An element name is specified for each atom type (1 to Ntype) in the simulation. The same element name can be given to multiple atom types.

In the case of *xyz* format dumps, there are no restrictions to what label can be used as an element name. Any whitespace separated text will be accepted.

The *every* keyword changes the dump frequency originally specified by the dump command to a new value. The every keyword can be specified in one of two ways. It can be a numeric value in which case it must be > 0. Or it can be an equal-style variable, which should be specified as v_name, where name is the variable name.

In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a dump snapshot will be written out. On that timestep the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for equal-style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal-style variables. Also see the next() function, which allows use of a file-style variable which reads successive values from a file, each time the variable is evaluated. Used with the *every* keyword, if the file contains a list of ascending timesteps, you can output snapshots whenever you wish.

Note that when using the variable option with the *every* keyword, you need to use the *first* option if you want an initial snapshot written to the dump file. The *every* keyword cannot be used with the dump *dcd* style.

For example, the following commands will write snapshots at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s first yes
```

The following commands would write snapshots at the timesteps listed in file tmp.times:

```
variable        f file tmp.times
variable        s equal next(f)
dump            1 all atom 100 tmp.dump
dump_modify     1 every v_s
```

IMPORTANT NOTE: When using a file-style variable with the *every* keyword, the file of timesteps must list a first timestep that is beyond the current timestep (e.g. it cannot be 0). And it must list one or more timesteps beyond the length of the run you perform. This is because the dump command will generate an error if the next timestep it reads from the file is not a value greater than the current timestep. Thus if you wanted output

on steps 0,15,100 of a 100-timestep run, the file should contain the values 15,100,101 and you should also use the dump_modify first command. Any final value > 100 could be used in place of 101.

The *first* keyword determines whether a dump snapshot is written on the very first timestep after the dump command is invoked. This will always occur if the current timestep is a multiple of N, the frequency specified in the [dump](#) command, including timestep 0. But if this is not the case, a dump snapshot will only be written if the setting of this keyword is *yes*. If it is *no*, which is the default, then it will not be written.

The *flush* keyword determines whether a flush operation is invoked after a dump snapshot is written to the dump file. A flush insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes. Flushes cannot be performed with dump style *xtc*.

The text-based dump styles have a default C-style format string which simply specifies %d for integers and %g for real values. The *format* keyword can be used to override the default with a new C-style format string. Do not include a trailing "\n" newline character in the format string. This option has no effect on the *dcd* and *xtc* dump styles since they write binary files. Note that for the *cfg* style, the first two fields (atom id and type) are not actually written into the CFG file, though you must include formats for them in the format string.

The *fileper* keyword is documented below with the *nfile* keyword.

The *image* keyword applies only to the dump *atom* style. If the image value is *yes*, 3 flags are appended to each atom's coords which are the absolute box image of the atom in each dimension. For example, an x image flag of -2 with a normalized coord of 0.5 means the atom is in the center of the box, but has passed thru the box boundary 2 times and is really 2 box lengths to the left of its current coordinate. Note that for dump style *custom* these various values can be printed in the dump file by using the appropriate atom attributes in the dump command itself.

The *label* keyword applies only to the dump *local* style. When it writes local information, such as bond or angle topology to a dump file, it will use the specified *label* to format the header. By default this includes 2 lines:

```
ITEM: NUMBER OF ENTRIES
ITEM: ENTRIES ...
```

The word "ENTRIES" will be replaced with the string specified, e.g. BONDS or ANGLES.

The *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified dump file name, for all dump styles except the *dcd*, *image*, *movie*, *xtc*, and *xyz* styles (for which "%" is not allowed). As explained on the [dump](#) command doc page, the "%" character causes the dump file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a dump file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a dump file.

The *pad* keyword only applies when the dump filename is specified with a wildcard "*" character which becomes the timestep. If *pad* is 0, which is the default, the timestep is converted into a string of unpadded length, e.g. 100 or 12000 or 2000000. When *pad* is specified with *Nchar* > 0, the string is padded with leading

zeroes so they are all the same length = *Nchar*. For example, pad 7 would yield 0000100, 0012000, 2000000. This can be useful so that post-processing programs can easily read the files in ascending timestep order.

The *precision* keyword only applies to the dump *xtc* style. A specified value of N means that coordinates are stored to 1/N nanometer accuracy, e.g. for N = 1000, the coordinates are written to 1/1000 nanometer accuracy.

The *region* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. If specified, only atoms in the region will be written to the dump file or included in the image/movie. Only one region can be applied as a filter (the last one specified). See the region command for more details. Note that a region can be defined as the "inside" or "outside" of a geometric shape, and it can be the "union" or "intersection" of a series of simpler regions.

The *scale* keyword applies only to the dump *atom* style. A scale value of *yes* means atom coords are written in normalized units from 0.0 to 1.0 in each box dimension. If the simluation box is triclinic (tilted), then all atom coords will still be between 0.0 and 1.0. A value of *no* means they are written in absolute distance units (e.g. Angstroms or sigma).

The *sort* keyword determines whether lines of per-atom output in a snapshot are sorted or not. A sort value of *off* means they will typically be written in indeterminate order, either in serial or parallel. This is the case even in serial if the atom_modify sort option is turned on, which it is by default, to improve performance. A sort value of *id* means sort the output by atom ID. A sort value of N or -N means sort the output by the value in the Nth column of per-atom info in either ascending or descending order.

The dump *local* style cannot be sorted by atom ID, since there are typically multiple lines of output per atom. Some dump styles, such as *dcd* and *xtc*, require sorting by atom ID to format the output file correctly. If multiple processors are writing the dump file, via the "%" wildcard in the dump filename, then sorting cannot be performed.

IMPORTANT NOTE: Unless it is required by the dump style, sorting dump file output requires extra overhead in terms of CPU and communication cost, as well as memory, versus unsorted output.

The *thresh* keyword only applies to the dump *custom*, *cfg*, *image*, and *movie* styles. Multiple thresholds can be specified. Specifying "none" turns off all threshold criteria. If thresholds are specified, only atoms whose attributes meet all the threshold criteria are written to the dump file or included in the image. The possible attributes that can be tested for are the same as those that can be specified in the dump custom command, with the exception of the *element* attribute, since it is not a numeric value. Note that different attributes can be output by the dump custom command than are used as threshold criteria by the dump_modify command. E.g. you can output the coordinates and stress of atoms whose energy is above some threshold.

The *unwrap* keyword only applies to the dump *dcd* and *xtc* styles. If set to *yes*, coordinates will be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed thru a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

These keywords apply only to the dump image and dump movie styles. Any keyword that affects an image, also affects a movie, since the movie is simply a collection of images. Some of the keywords only affect the dump movie style. The description gives details.

The *acolor* keyword can be used with the dump image command, when its atom color setting is *type*, to set the color that atoms of each type will be drawn in the image.

---

dump_modify command   269

The specified *type* should be an integer from 1 to Ntypes = the number of atom types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified atom types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified atom types.

The *adiam* keyword can be used with the [dump image](#) command, when its atom diameter setting is *type*, to set the size that atoms of each type will be drawn in the image. The specified *type* should be an integer from 1 to Ntypes. As with the *acolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of atomt types. The specified *diam* is the size in whatever distance [units](#) the input script is using, e.g. Angstroms.

The *amap* keyword can be used with the [dump image](#) command, with its *atom* keyword, when its atom setting is an atom-attribute, to setup a color map. The color map is used to assign a specific RGB (red/green/blue) color value to an individual atom when it is drawn, based on the atom's attribute, which is a numeric value, e.g. its x-component of velocity if the atom-attribute "vx" was specified.

The basic idea of a color map is that the atom-attribute will be within a range of values, and that range is associated with a a series of colors (e.g. red, blue, green). An atom's specific value (vx = -3.2) can then mapped to the series of colors (e.g. halfway between red and blue), and a specific color is determined via an interpolation procedure.

There are many possible options for the color map, enabled by the *amap* keyword. Here are the details.

The *lo* and *hi* settings determine the range of values allowed for the atom attribute. If numeric values are used for *lo* and/or *hi*, then values that are lower/higher than that value are set to the value. I.e. the range is static. If *lo* is specified as *min* or *hi* as *max* then the range is dynamic, and the lower and/or upper bound will be calculated each time an image is drawn, based on the set of atoms being visualized.

The *style* setting is two letters, such as "ca". The first letter is either "c" for continuous, "d" for discrete, or "s" for sequential. The second letter is either "a" for absolute, or "f" for fractional.

A continuous color map is one in which the color changes continuously from value to value within the range. A discrete color map is one in which discrete colors are assigned to sub-ranges of values within the range. A sequential color map is one in which discrete colors are assigned to a sequence of sub-ranges of values covering the entire range.

An absolute color map is one in which the values to which colors are assigned are specified explicitly as values within the range. A fractional color map is one in which the values to which colors are assigned are specified as a fractional portion of the range. For example if the range is from -10.0 to 10.0, and the color red is to be assigned to atoms with a value of 5.0, then for an absolute color map the number 5.0 would be used. But for a fractional map, the number 0.75 would be used since 5.0 is 3/4 of the way from -10.0 to 10.0.

The *delta* setting must be specified for all styles, but is only used for the sequential style; otherwise the value is ignored. It specifies the bin size to use within the range for assigning consecutive colors to. For example, if the range is from -10.0 to 10.0 and a *delta* of 1.0 is used, then 20 colors will be assigned to the range. The first will be from -10.0 <= color1 < -9.0, then 2nd from -9.0 <= color2 < -8.0, etc.

The *N* setting is how many entries follow. The format of the entries depends on whether the color map style is continuous, discrete or sequential. In all cases the *color* setting can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

For continuous color maps, each entry has a *value* and a *color*. The *value* is either a number within the range of values or *min* or *max*. The *value* of the first entry must be *min* and the *value* of the last entry must be *max*. Any entries in between must have increasing values. Note that numeric values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. X will fall between 2 of the entry values. The color of the atom is linearly interpolated (in each of the RGB values) between the 2 colors associated with those entries. For example, if X = -5.0 and the 2 surrounding entries are "red" at -10.0 and "blue" at 0.0, then the atom's color will be halfway between "red" and "blue", which happens to be "purple".

For discrete color maps, each entry has a *lo* and *hi* value and a *color*. The *lo* and *hi* settings are either numbers within the range of values or *lo* can be *min* or *hi* can be *max*. The *lo* and *hi* settings of the last entry must be *min* and *max*. Other entries can have any *lo* and *hi* values and the sub-ranges of different values can overlap. Note that numeric *lo* and *hi* values can be specified either as absolute numbers or as fractions (0.0 to 1.0) of the range, depending on the "a" or "f" in the style setting for the color map.

Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The entries are scanned from first to last. The first time that *lo* <= X <= *hi*, X is assigned the color associated with that entry. You can think of the last entry as assigning a default color (since it will always be matched by X), and the earlier entries as colors that override the default. Also note that no interpolation of a color RGB is done. All atoms will be drawn with one of the colors in the list of entries.

For sequential color maps, each entry has only a *color*. Here is how the entries are used to determine the color of an individual atom, given the value X of its atom attribute. The range is partitioned into N bins of width *binsize*. Thus X will fall in a specific bin from 1 to N, say the Mth bin. If it falls on a boundary between 2 bins, it is considered to be in the higher of the 2 bins. Each bin is assigned a color from the E entries. If E < N, then the colors are repeated. For example if 2 entries with colors red and green are specified, then the odd numbered bins will be red and the even bins green. The color of the atom is the color of its bin. Note that the sequential color map is really a shorthand way of defining a discrete color map without having to specify where all the bin boundaries are.

The *backcolor* sets the background color of the images. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *bcolor* keyword can be used with the [dump image](#) command, with its *bond* keyword, when its color setting is *type*, to set the color that bonds of each type will be drawn in the image.

The specified *type* should be an integer from 1 to Nbondtypes = the number of bond types. A wildcard asterisk can be used in place of or in conjunction with the *type* argument to specify a range of bond types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of bond types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

The specified *color* can be a single color which is any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option. Or it can be two or more colors separated by a "/" character, e.g. red/green/blue. In the former case, that color is assigned to all the specified bond types. In the latter case, the list of colors are assigned in a round-robin fashion to each of the specified bond types.

The *bdiam* keyword can be used with the dump image command, with its *bond* keyword, when its diam setting is *type*, to set the diameter that bonds of each type will be drawn in the image. The specified *type* should be an integer from 1 to Nbondtypes. As with the *bcolor* keyword, a wildcard asterisk can be used as part of the *type* argument to specify a range of bond types. The specified *diam* is the size in whatever distance units you are using, e.g. Angstroms.

The *bitrate* keyword can be used with the dump movie command to define the size of the resulting movie file and its quality via setting how many kbits per second are to be used for the movie file. Higher bitrates require less compression and will result in higher quality movies. The quality is also determined by the compression format and encoder. The default setting is 2000 kbit/s, which will result in average quality with older compression formats.

IMPORTANT NOTE: Not all movie file formats supported by dump movie allow the bitrate to be set. If not, the setting is silently ignored.

The *boxcolor* keyword sets the color of the simulation box drawn around the atoms in each image. See the "dump image box" command for how to specify that a box be drawn. The color name can be any of the 140 pre-defined colors (see below) or a color name defined by the dump_modify color option.

The *color* keyword allows definition of a new color name, in addition to the 140-predefined colors (see below), and associates 3 red/green/blue RGB values with that color name. The color name can then be used with any other dump_modify keyword that takes a color name as a value. The RGB values should each be floating point values between 0.0 and 1.0 inclusive.

When a color name is converted to RGB values, the user-defined color names are searched first, then the 140 pre-defined color names. This means you can also use the *color* keyword to overwrite one of the pre-defined color names with new RBG values.

The *framerate* keyword can be used with the dump movie command to define the duration of the resulting movie file. Movie files written by the dump *movie* command have a default frame rate of 24 frames per second and the images generated will be converted at that rate. Thus a sequence of 1000 dump images will result in a movie of about 42 seconds. To make a movie run longer you can either generate images more frequently or lower the frame rate. To speed a movie up, you can do the inverse. Using a frame rate higher than 24 is not recommended, as it will result in simply dropping the rendered images. It is more efficient to dump images less frequently.

Restrictions: none

Related commands:

dump, dump image, undump

Default:

The option defaults are

- append = no
- buffer = yes for dump styles *atom*, *custom*, *loca*, and *xyz*
- element = "C" for every atom type
- every = whatever it was set to via the dump command
- fileper = # of processors

- first = no
- flush = yes
- format = %d and %g for each integer or floating point value
- image = no
- label = ENTRIES
- nfile = 1
- pad = 0
- precision = 1000
- region = none
- scale = yes
- sort = off for dump styles *atom*, *custom*, *cfg*, and *local*
- sort = id for dump styles *dcd*, *xtc*, and *xyz*
- thresh = none
- unwrap = no

- acolor = * red/green/blue/yellow/aqua/cyan
- adiam = * 1.0
- amap = min max cf 0.0 2 min blue max red
- backcolor = black
- bcolor = * red/green/blue/yellow/aqua/cyan
- bdiam = * 0.5
- bitrate = 2000
- boxcolor = yellow
- color = 140 color names are pre-defined as listed below
- framerate = 24

These are the standard 109 element names that LAMMPS pre-defines for use with the dump image and dump_modify commands.

- 1-10 = "H", "He", "Li", "Be", "B", "C", "N", "O", "F", "Ne"
- 11-20 = "Na", "Mg", "Al", "Si", "P", "S", "Cl", "Ar", "K", "Ca"
- 21-30 = "Sc", "Ti", "V", "Cr", "Mn", "Fe", "Co", "Ni", "Cu", "Zn"
- 31-40 = "Ga", "Ge", "As", "Se", "Br", "Kr", "Rb", "Sr", "Y", "Zr"
- 41-50 = "Nb", "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "In", "Sn"
- 51-60 = "Sb", "Te", "I", "Xe", "Cs", "Ba", "La", "Ce", "Pr", "Nd"
- 61-70 = "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm", "Yb"
- 71-80 = "Lu", "Hf", "Ta", "W", "Re", "Os", "Ir", "Pt", "Au", "Hg"
- 81-90 = "Tl", "Pb", "Bi", "Po", "At", "Rn", "Fr", "Ra", "Ac", "Th"
- 91-100 = "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf", "Es", "Fm"
- 101-109 = "Md", "No", "Lr", "Rf", "Db", "Sg", "Bh", "Hs", "Mt"

These are the 140 colors that LAMMPS pre-defines for use with the dump image and dump_modify commands. Additional colors can be defined with the dump_modify color command. The 3 numbers listed for each name are the RGB (red/green/blue) values. Divide each value by 255 to get the equivalent 0.0 to 1.0 value.

aliceblue = 240, 248, 255	antiquewhite = 250, 235, 215	aqua = 0, 255, 255	aquamarine = 127, 255, 212	azure = 240, 255, 255
beige = 245, 245, 220	bisque = 255, 228, 196	black = 0, 0, 0	blanchedalmond = 255, 255, 205	blue = 0, 0, 255
blueviolet = 138, 43, 226	brown = 165, 42, 42	burlywood = 222, 184, 135	cadetblue = 95, 158, 160	chartreuse = 127, 255, 0
	coral = 255, 127, 80			

chocolate = 210, 105, 30		cornflowerblue = 100, 149, 237	cornsilk = 255, 248, 220	crimson = 220, 20, 60
cyan = 0, 255, 255	darkblue = 0, 0, 139	darkcyan = 0, 139, 139	darkgoldenrod = 184, 134, 11	darkgray = 169, 169, 169
darkgreen = 0, 100, 0	darkkhaki = 189, 183, 107	darkmagenta = 139, 0, 139	darkolivegreen = 85, 107, 47	darkorange = 255, 140, 0
darkorchid = 153, 50, 204	darkred = 139, 0, 0	darksalmon = 233, 150, 122	darkseagreen = 143, 188, 143	darkslateblue = 72, 61, 139
darkslategray = 47, 79, 79	darkturquoise = 0, 206, 209	darkviolet = 148, 0, 211	deeppink = 255, 20, 147	deepskyblue = 0, 191, 255
dimgray = 105, 105, 105	dodgerblue = 30, 144, 255	firebrick = 178, 34, 34	floralwhite = 255, 250, 240	forestgreen = 34, 139, 34
fuchsia = 255, 0, 255	gainsboro = 220, 220, 220	ghostwhite = 248, 248, 255	gold = 255, 215, 0	goldenrod = 218, 165, 32
gray = 128, 128, 128	green = 0, 128, 0	greenyellow = 173, 255, 47	honeydew = 240, 255, 240	hotpink = 255, 105, 180
indianred = 205, 92, 92	indigo = 75, 0, 130	ivory = 255, 240, 240	khaki = 240, 230, 140	lavender = 230, 230, 250
lavenderblush = 255, 240, 245	lawngreen = 124, 252, 0	lemonchiffon = 255, 250, 205	lightblue = 173, 216, 230	lightcoral = 240, 128, 128
lightcyan = 224, 255, 255	lightgoldenrodyellow = 250, 250, 210	lightgreen = 144, 238, 144	lightgrey = 211, 211, 211	lightpink = 255, 182, 193
lightsalmon = 255, 160, 122	lightseagreen = 32, 178, 170	lightskyblue = 135, 206, 250	lightslategray = 119, 136, 153	lightsteelblue = 176, 196, 222
lightyellow = 255, 255, 224	lime = 0, 255, 0	limegreen = 50, 205, 50	linen = 250, 240, 230	magenta = 255, 0, 255
maroon = 128, 0, 0	mediumaquamarine = 102, 205, 170	mediumblue = 0, 0, 205	mediumorchid = 186, 85, 211	mediumpurple = 147, 112, 219
mediumseagreen = 60, 179, 113	mediumslateblue = 123, 104, 238	mediumspringgreen = 0, 250, 154	mediumturquoise = 72, 209, 204	mediumvioletred = 199, 21, 133
midnightblue = 25, 25, 112	mintcream = 245, 255, 250	mistyrose = 255, 228, 225	moccasin = 255, 228, 181	navajowhite = 255, 222, 173
navy = 0, 0, 128	oldlace = 253, 245, 230	olive = 128, 128, 0	olivedrab = 107, 142, 35	orange = 255, 165, 0
orangered = 255, 69, 0	orchid = 218, 112, 214	palegoldenrod = 238, 232, 170	palegreen = 152, 251, 152	paleturquoise = 175, 238, 238
palevioletred = 219, 112, 147	papayawhip = 255, 239, 213	peachpuff = 255, 239, 213	peru = 205, 133, 63	pink = 255, 192, 203
plum = 221, 160, 221	powderblue = 176, 224, 230	purple = 128, 0, 128	red = 255, 0, 0	rosybrown = 188, 143, 143
royalblue = 65, 105, 225	saddlebrown = 139, 69, 19	salmon = 250, 128, 114	sandybrown = 244, 164, 96	seagreen = 46, 139, 87
seashell = 255, 245, 238	sienna = 160, 82, 45	silver = 192, 192, 192	skyblue = 135, 206, 235	slateblue = 106, 90, 205
slategray = 112, 128, 144	snow = 255, 250, 250	springgreen = 0, 255, 127	steelblue = 70, 130, 180	tan = 210, 180, 140
teal = 0, 128, 128	thistle = 216, 191, 216	tomato = 253, 99, 71	turquoise = 64, 224, 208	violet = 238, 130, 238
	white = 255, 255, 255		yellow = 255, 255, 0	

wheat = 245, 222, 179		whitesmoke = 245, 245, 245		yellowgreen = 154, 205, 50

dump molfile command

Syntax:

```
dump ID group-ID molfile N file format path
```

- ID = user-assigned name for the dump
- group-ID = ID of the group of atoms to be imaged
- molfile = style of dump command (other styles *atom* or *cfg* or *dcd* or *xtc* or *xyz* or *local* or *custom* are discussed on the dump doc page)
- N = dump every this many timesteps
- file = name of file to write to
- format = file format to be used
- path = file path with plugins (optional)

Examples:

```
dump mf1 all molfile 10 melt1.xml hoomd
dump mf2 all molfile 10 melt2-*.pdb pdb .
dump mf3 all molfile 50 melt3.xyz xyz .:/home/akohlmey/vmd/plugins/LINUX/molfile
```

Description:

Dump a snapshot of atom coordinates and selected additional quantities to one or more files every N timesteps in one of several formats. Only information for atoms in the specified group is dumped. This specific dump style uses molfile plugins that are bundled with the VMD molecular visualization and analysis program. See Section tools of the manual and the tools/lmp2vmd/README.txt file for more information about support in VMD for reading and visualizing native LAMMPS dump files.

Unless the filename contains a * character, the output will be written to one single file with the specified format. Otherwise there will be one file per snapshot and the * will be replaced by the time step number when the snapshot is written.

IMPORTANT NOTE: Because periodic boundary conditions are enforced only on timesteps when neighbor lists are rebuilt, the coordinates of an atom written to a dump file may be slightly outside the simulation box.

The molfile plugin API has a few restrictions that have to be honored by this dump style: the number of atoms must not change, the atoms must be sorted, outside of the coordinates no change in atom properties (like type, mass, charge) will be recorded.

The *format* keyword determines what format is used to write out the dump. For this to work, LAMMPS must be able to find and load a compatible molfile plugin that supports this format. Settings made via the dump_modify command can alter per atom properties like element names.

The *path* keyword determines which in directories. This is a "path" like other search paths, i.e. it can contain multiple directories separated by a colon (or semi-colon on windows). This keyword is optional and default to ".", the current directory.

The *unwrap* option of the dump_modify command allows coordinates to be written "unwrapped" by the image flags for each atom. Unwrapped means that if the atom has passed through a periodic boundary one or more times, the value is printed for what the coordinate would be if it had not been wrapped back into the periodic box. Note that these coordinates may thus be far outside the box size stored with the snapshot.

Dumps are performed on timesteps that are a multiple of N (including timestep 0) and on the last timestep of a minimization if the minimization converges. Note that this means a dump will not be performed on the initial timestep after the dump command is invoked, if the current timestep is not a multiple of N. This behavior can be changed via the dump_modify first command, which can be useful if the dump command is invoked after a minimization ended on an arbitrary timestep. N can be changed between runs by using the dump_modify every command. The dump_modify every command also allows a variable to be used to determine the sequence of timesteps on which dump files are written.

Restrictions:

The *molfile* dump style is part of the USER-MOLFILE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Molfile plugins provide a consistent programming interface to read and write file formats commonly used in molecular simulations. The USER-MOLFILE package only provides the interface code, not the plugins. These can be obtained from a VMD installation which has to match the platform that you are using to compile LAMMPS for. By adding plugins to VMD, support for new file formats can be added to LAMMPS (or VMD or other programs that use them) without having to recompile the application itself. The plugins are installed in the directory: /plugins//molfile

NOTE: while the programming interface (API) to the plugins is backward compatible, the binary interface (ABI) has been changing over time, so it is necessary to compile this package with the plugin header files from VMD that match the binary plugins. These header files in the directory: /plugins/include For convenience, the package ships with a set of header files that are compatible with VMD 1.9 and 1.9.1 (June 2012)

Related commands:

dump, dump_modify, undump

Default:

The default path is ".". All other properties have to be specified.

echo command

Syntax:

```
echo style
```

- style = *none* or *screen* or *log* or *both*

Examples:

```
echo both
echo log
```

Description:

This command determines whether LAMMPS echoes each input script command to the screen and/or log file as it is read and processed. If an input script has errors, it can be useful to look at echoed output to see the last command processed.

The [command-line switch](#) -echo can be used in place of this command.

Restrictions: none

Related commands: none

Default:

```
echo log
```

fix adapt command

Syntax:

```
fix ID group-ID adapt N attribute args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- adapt = style name of this fix command
- N = adapt simulation settings every this many timesteps
- one or more attribute/arg pairs may be appended
- attribute = *pair* or *kspace* or *atom*

```
  pair args = pstyle pparam I J v_name
      pstyle = pair style name, e.g. lj/cut
      pparam = parameter to adapt over time
      I,J = type pair(s) to set parameter for
      v_name = variable with name that calculates value of pparam
    kspace arg = v_name
      v_name = variable with name that calculates scale factor on K-space terms
    atom args = aparam v_name
      aparam = parameter to adapt over time
      v_name = variable with name that calculates value of aparam
```
- zero or more keyword/value pairs may be appended
- keyword = *scale* or *reset*

```
  scale value = no or yes
      no = the variable value is the new setting
      yes = the variable value multiplies the original setting
    reset value = no or yes
      no = values will remain altered at the end of a run
      yes = reset altered values to their original values at the end of a run
```

Examples:

```
fix 1 all adapt 1 pair soft a 1 1 v_prefactor
fix 1 all adapt 1 pair soft a 2* 3 v_prefactor
fix 1 all adapt 1 pair lj/cut epsilon * * v_scale1 coul/cut scale 3 3 v_scale2 scale yes reset ye
fix 1 all adapt 10 atom diameter v_size
```

Description:

Change or adapt one or more specific simulation attributes or settings over time as a simulation runs. Pair potential and K-space and atom attributes which can be varied by this fix are discussed below. Many other fixes can also be used to time-vary simulation parameters, e.g. the "fix deform" command will change the simulation box size/shape and the "fix move" command will change atom positions and velocities in a prescribed manner. Also note that many commands allow variables as arguments for specific parameters, if described in that manner on their doc pages. An equal-style variable can calculate a time-dependent quantity, so this is another way to vary a simulation parameter over time.

If *N* is specified as 0, the specified attributes are only changed once, before the simulation begins. This is all that is needed if the associated variables are not time-dependent. If *N* > 0, then changes are made every *N* steps during the simulation, presumably with a variable that is time-dependent.

Depending on the value of the *reset* keyword, attributes changed by this fix will or will not be reset back to their original values at the end of a simulation. Even if *reset* is specified as *yes*, a restart file written during a simulation will contain the modified settings.

If the *scale* keyword is set to *no*, then the value the parameter is set to will be whatever the variable generates. If the *scale* keyword is set to *yes*, then the value of the altered parameter will be the initial value of that parameter multiplied by whatever the variable generates. I.e. the variable is now a "scale factor" applied in (presumably) a time-varying fashion to the parameter. Internally, the parameters themselves are actually altered; make sure you use the *reset yes* option if you want the parameters to be restored to their initial values after the run.

The *pair* keyword enables various parameters of potentials defined by the pair_style command to be changed, if the pair style supports it. Note that the pair_style and pair_coeff commands must be used in the usual manner to specify these parameters initially; the fix adapt command simply overrides the parameters.

The *pstyle* argument is the name of the pair style. If pair_style hybrid or hybrid/overlay is used, *pstyle* should be a sub-style name. For example, *pstyle* could be specified as "soft" or "lubricate". The *pparam* argument is the name of the parameter to change. This is the current list of pair styles and parameters that can be varied by this fix. See the doc pages for individual pair styles and their energy formulas for the meaning of these parameters:

born	a,b,c	type pairs
buck	a,c	type pairs
coul/cut	scale	type pairs
coul/debye	scale	type pairs
coul/long	scale	type pairs
lj/cut	epsilon,sigma	type pairs
lj/expand	epsilon,sigma,delta	type pairs
lubricate	mu	global
gauss	a	type pairs
soft	a	type pairs

IMPORTANT NOTE: It is easy to add new potentials and their parameters to this list. All it typically takes is adding an extract() method to the pair_*.cpp file associated with the potential.

Some parameters are global settings for the pair style, e.g. the viscosity setting "mu" for pair_style lubricate. Other parameters apply to atom type pairs within the pair style, e.g. the prefactor "a" for pair_style soft.

Note that for many of the potentials, the parameter that can be varied is effectively a prefactor on the entire energy expression for the potential, e.g. the lj/cut epsilon. The parameters listed as "scale" are exactly that, since the energy expression for the coul/cut potential (for example) has no labeled prefactor in its formula. To apply an effective prefactor to some potentials, multiple parameters need to be altered. For example, the Buckingham potential needs both the A and C terms altered together. To scale the Buckingham potential, you should thus list the pair style twice, once for A and once for C.

If a type pair parameter is specified, the *I* and *J* settings should be specified to indicate which type pairs to apply it to. If a global parameter is specified, the *I* and *J* settings still need to be specified, but are ignored.

Similar to the pair_coeff command, I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wild-card asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type

pairs where J < I, they are ignored.

IMPROTANT NOTE: If pair_style hybrid or hybrid/overlay is being used, then the *pstyle* will be a sub-style name. You must specify I,J arguments that correspond to type pair values defined (via the pair_coeff command) for that sub-style.

The *v_name* argument for keyword *pair* is the name of an equal-style variable which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify parameters that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the run command and the *elaplong* keyword of thermo_style custom for details.

For example, these commands would change the prefactor coefficient of the pair_style soft potential from 10.0 to 30.0 in a linear fashion over the course of a simulation:

```
variable prefactor equal ramp(10,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

The *kspace* keyword used the specified variable as a scale factor on the energy, forces, virial calculated by whatever K-Space solver is defined by the kspace_style command. If the variable has a value of 1.0, then the solver is unaltered.

The *kspace* keyword works this way whether the *scale* keyword is set to *no* or *yes*.

The *atom* keyword enables various atom properties to be changed. The *aparam* argument is the name of the parameter to change. This is the current list of atom parameters that can be varied by this fix:

- charge = charge on particle
- diameter = diameter of particle

The *v_name* argument of the *atom* keyword is the name of an equal-style variable which will be evaluated each time this fix is invoked to set the parameter to a new value. It should be specified as v_name, where name is the variable name. See the discussion above describing the formulas associated with equal-style variables. The new value is assigned to the corresponding attribute for all atoms in the fix group.

If the atom parameter is *diameter* and per-atom density and per-atom mass are defined for particles (e.g. atom_style granular), then the mass of each particle is also changed when the diameter changes (density is assumed to stay constant).

For example, these commands would shrink the diameter of all granular particles in the "center" group from 1.0 to 0.1 in a linear fashion over the course of a 1000-step simulation:

```
variable size equal ramp(1.0,0.1)
fix 1 center adapt 10 atom diameter v_size
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

fix adapt command   281

Related commands:

compute ti

Default:

The option defaults are scale = no, reset = no.

fix addforce command

fix addforce/cuda command

Syntax:

```
fix ID group-ID addforce fx fy fz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- addforce = style name of this fix command
- fx,fy,fz = force component values (force units)

  ```
    any of fx,fy,fz can be a variable (see below)
  ```
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *energy*

```
  region value = region-ID
     region-ID = ID of region atoms must be in to have added force
  energy value = v_name
     v_name = variable with name that calculates the potential energy of each atom in the a
```

Examples:

```
fix kick flow addforce 1.0 0.0 0.0
fix kick flow addforce 1.0 0.0 v_oscillate
fix ff boundary addforce 0.0 0.0 v_push energy v_espace
```

Description:

Add fx,fy,fz to the corresponding component of force for each atom in the group. This command can be used to give an additional push to atoms in a simulation, such as for a simulation of Poiseuille flow in a channel.

Any of the 3 quantities defining the force components can be specified as an equal-style or atom-style [variable](#), namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value(s) used to determine the force component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent force field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent force field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the "run" command, this energy can be optionally added to the system's potential energy for thermodynamic output (see below). For energy minimization via the "minimize" command, this energy must be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

The *energy* keyword is not allowed if the added force is a constant vector F = (fx,fy,fz), with all components defined as numeric constants and not as variables. This is because LAMMPS can compute the energy for each atom directly as E = -x dot F = -(x*fx + y*fy + z*fz), so that -Grad(E) = F.

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the [run](#) command. If the keyword is not used, LAMMPS will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the "minimize" command. The keyword specifies the name of an atom-style [variable](#) which is used to compute the energy of each atom as function of its position. Like variables used for *fx*, *fy*, *fz*, the energy variable is specified as v_name, where name is the variable name.

Note that when the *energy* keyword is used during an energy minimization, you must insure that the formula defined for the atom-style [variable](#) is consistent with the force variable formulas, i.e. that -Grad(E) = F. For example, if the force were a spring-like F = kx, then the energy formula should be E = -0.5kx^2. If you don't do this correctly, the minimization will not converge properly.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the potential "energy" inferred by the added force to the system's potential energy as part of [thermodynamic output](#). This is a fictitious quantity but is needed so that the [minimize](#) command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force.

This fix computes a global scalar and a global 3-vector of forces, which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The vector is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the

fix addforce/cuda command   284

fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix setforce, fix aveforce

Default: none

fix addtorque command

Syntax:

```
fix ID group-ID addtorque Tx Ty Tz
```

- ID, group-ID are documented in [fix](#) command
- addtorque = style name of this fix command
- Tx,Ty,Tz = torque component values (torque units)
- any of Tx,Ty,Tz can be a variable (see below)

Examples:

```
fix kick bead addtorque 2.0 3.0 5.0
fix kick bead addtorque 0.0 0.0 v_oscillate
```

Description:

Add a set of forces to each atom in the group such that:

- the components of the total torque applied on the group (around its center of mass) are Tx,Ty,Tz
- the group would move as a rigid body in the absence of other forces.

This command can be used to drive a group of atoms into rotation.

Any of the 3 quantities defining the torque components can be specified as an equal-style [variable](#), namely *Tx*, *Ty*, *Tz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the torque component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent torque.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the potential "energy" inferred by the added forces to the system's potential energy as part of [thermodynamic output](#). This is a fictitious quantity but is needed so that the [minimize](#) command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added forces.

This fix computes a global scalar and a global 3-vector, which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The vector is the total torque on the group of atoms before the forces on individual atoms are changed by the fix. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

Restrictions:

This fix is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[fix addforce](#)

Default: none

fix append/atoms command

Syntax:

```
fix ID group-ID append/atoms face ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- append/atoms = style name of this fix command
- face = *zhi*
- zero or more keyword/value pairs may be appended
- keyword = *basis* or *size* or *freq* or *temp* or *random* or *units*

```
  basis values = M itype
     M = which basis atom
     itype = atom type (1-N) to assign to this basis atom
   size args = Lz
     Lz = z size of lattice region appended in a single event(distance units)
   freq args = freq
     freq = the number of timesteps between append events
   temp args = target damp seed extent
     target = target temperature for the region between zhi-extent and zhi (temperature uni
     damp = damping parameter (time units)
     seed = random number seed for langevin kicks
     extent = extent of thermostated region (distance units)
   random args = xmax ymax zmax seed
     xmax, ymax, zmax = maximum displacement in particular direction (distance units)
     seed = random number seed for random displacement
   units value = lattice or box
     lattice = the wall position is defined in lattice units
     box = the wall position is defined in simulation box units
```

Examples:

```
fix 1 all append/atoms zhi size 5.0 freq 295 units lattice
fix 4 all append/atoms zhi size 15.0 freq 5 units box
fix A all append/atoms zhi size 1.0 freq 1000 units lattice
```

Description:

This fix creates atoms on a lattice, appended on the zhi edge of the system box. This can be useful when a shock or wave is propagating from zlo. This allows the system to grow with time to accommodate an expanding wave. A simulation box must already exist, which is typically created via the [create_box](#) command. Before using this command, a lattice must also be defined using the [lattice](#) command.

This fix will automatically freeze atoms on the zhi edge of the system, so that overlaps are avoided when new atoms are appended.

The *basis* keyword specifies an atom type that will be assigned to specific basis atoms as they are created. See the [lattice](#) command for specifics on how basis atoms are defined for the unit cell of the lattice. By default, all created atoms are assigned type = 1 unless this keyword specifies differently.

The *size* keyword defines the size in z of the chunk of material to be added.

The *random* keyword will give the atoms random displacements around their lattice points to simulate some initial temperature.

The *temp* keyword will cause a region to be thermostated with a Langevin thermostat on the zhi boundary. The size of the region is measured from zhi and is set with the *extent* argument.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The boundary on which atoms are added with append/atoms must be shrink/minimum. The opposite boundary may be any boundary type other than periodic.

Related commands:

fix wall/piston command

Default:

The keyword defaults are size = 0.0, freq = 0, units = lattice. All added atoms are of type 1 unless the basis keyword is used.

fix atc command

Syntax:

```
fix   atc
```

- fixID = name of fix
- group = name of group fix is to be applied
- type = *thermal* or *two_temperature* or *hardy* or *field*

  thermal = thermal coupling with fields: temperature
   two_temperature = electron-phonon coupling with field: temperature and electron_temperature
   hardy = on-the-fly post-processing using kernel localization functions (see "related" section
   field = on-the-fly post-processing using mesh-based localization functions (see "related" sect
- parameter_file = name of the file with material parameters. Note: Neither hardy nor field requires a
  parameter file

Examples:

```
fix AtC internal atc thermal Ar_thermal.dat
fix AtC internal atc two_temperature Ar_ttm.mat
fix AtC internal atc hardy
fix AtC internal atc field
```

Description:

This fix is the beginning to creating a coupled FE/MD simulation and/or an on-the-fly estimation of continuum fields. The coupled versions of this fix do Verlet integration and the post-processing does not. After instantiating this fix, several other fix_modify commands will be needed to set up the problem, e.g. define the finite element mesh and prescribe initial and boundary conditions.
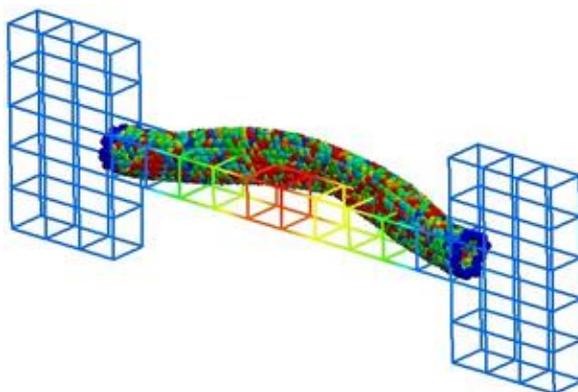

```
The following coupling example is typical, but non-exhaustive:
 # ... commands to create and initialize the MD system

 # initial fix to designate coupling type and group to apply it to
 # tag group physics material_file
 fix AtC internal atc thermal Ar_thermal.mat

 # create a uniform 12 x 2 x 2 mesh that covers region contain the group
 # nx ny nz region periodicity
 fix_modify AtC mesh create 12 2 2 mdRegion f p p

 # specify the control method for the type of coupling
```

```
# physics control_type
fix_modify AtC thermal control flux


# specify the initial values for the empirical field "temperature"
# field node_group value
fix_modify AtC initial temperature all 30


# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC output atc_fe_output 100


run 1000
```

likewise for this post-processing example:

```
# ... commands to create and initialize the MD system


# initial fix to designate post-processing and the group to apply it to
# no material file is allowed nor required
fix AtC internal atc hardy


# for hardy fix, specific kernel function (function type and range) to # be used as a localiza
fix AtC kernel quartic_sphere 10.0


# create a uniform 1 x 1 x 1 mesh that covers region contain the group
# with periodicity this effectively creats a system average
fix_modify AtC mesh create 1 1 1 box p p p


# change from default lagrangian map to eulerian
# refreshed every 100 steps
fix_modify AtC atom_element_map eulerian 100


# start with no field defined
# add mass density, potential energy density, stress and temperature
fix_modify AtC fields add density energy stress temperature


# create an output stream for nodal fields
# filename output_frequency
fix_modify AtC output nvtFE 100 text


run 1000
```

the mesh's linear interpolation functions can be used as the localization function by using the field option:

fix AtC internal atc field

fix_modify AtC mesh create 1 1 1 box p p p

...

Note coupling and post-processing can be combined in the same simulations using separate fixes.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. The fix_modify options relevant to this fix are listed below. No global scalar or vector or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Thermal and two_temperature (coupling) types use a Verlet time-integration algorithm. The hardy type does not contain its own time-integrator and must be used with a separate fix that does contain one, e.g. nve, nvt, etc.

- Currently,
- - the coupling is restricted to thermal physics
- - the FE computations are done in serial on each processor.

Related commands:

After specifying this fix in your input script, several other [fix_modify](#) commands are used to setup the problem, e.g. define the finite element mesh and prescribe initial and boundary conditions.

fix_modify commands for setup:

- [fix_modify AtC mesh create](#)
- [fix_modify AtC mesh quadrature](#)
- [fix_modify AtC mesh read](#)
- [fix_modify AtC mesh write](#)
- [fix_modify AtC mesh create_nodeset](#)
- [fix_modify AtC mesh add_to_nodeset](#)
- [fix_modify AtC mesh create_faceset box](#)
- [fix_modify AtC mesh create_faceset plane](#)
- [fix_modify AtC mesh create_elementset](#)
- [fix_modify AtC mesh delete_elements](#)
- [fix_modify AtC mesh nodeset_to_elementset](#)
- [fix_modify AtC boundary](#)
- [fix_modify AtC internal_quadrature](#)
- [fix_modify AtC time_integration (thermal)](#)
- [fix_modify AtC time_integration (momentum)](#)
- [fix_modify AtC extrinsic electron_integration](#)
- [fix_modify AtC internal_element_set](#)
- [fix_modify AtC decomposition](#)

fix_modify commands for boundary and initial conditions:

- [fix_modify AtC initial](#)
- [fix_modify AtC fix](#)
- [fix_modify AtC unfix](#)
- [fix_modify AtC fix_flux](#)
- [fix_modify AtC unfix_flux](#)
- [fix_modify AtC source](#)
- [fix_modify AtC remove_source](#)

fix_modify commands for control and filtering:

- [fix_modify AtC control](#)
- [fix_modify AtC control thermal](#)
- [fix_modify AtC control thermal correction_max_iterations](#)
- [fix_modify AtC control momentum](#)
- [fix_modify AtC control localized_lambda](#)
- [fix_modify AtC control lumped_lambda_solve](#)

fix atc command

- [fix_modify AtC control mask_direction](#) control
- [fix_modify AtC filter](#)
- [fix_modify AtC filter scale](#)
- [fix_modify AtC filter type](#)
- [fix_modify AtC equilibrium_start](#)
- [fix_modify AtC extrinsic exchange](#)
- [fix_modify AtC poisson_solver](#)

fix_modify commands for output:

- [fix_modify AtC output](#)
- [fix_modify AtC output nodeset](#)
- [fix_modify AtC output elementset](#)
- [fix_modify AtC output boundary_integral](#)
- [fix_modify AtC output contour_integral](#)
- [fix_modify AtC mesh output](#)
- [fix_modify AtC write_restart](#)
- [fix_modify AtC read_restart](#)

fix_modify commands for post-processing:

- [fix_modify AtC kernel](#)
- [fix_modify AtC fields](#)
- [fix_modify AtC grdients](#)
- [fix_modify AtC rates](#)
- [fix_modify AtC computes](#)
- [fix_modify AtC on_the_fly](#)
- [fix_modify AtC pair_interactions/bond_interactions](#)
- [fix_modify AtC sample_frequency](#)
- [fix_modify AtC set](#)

miscellaneous fix_modify commands:

- [fix_modify AtC atom_element_map](#)
- [fix_modify AtC atom_weight](#)
- [fix_modify AtC write_atom_weights](#)
- [fix_modify AtC reset_time](#)
- [fix_modify AtC reset_atomic_reference_positions](#)
- [fix_modify AtC fe_md_boundary](#)
- [fix_modify AtC boundary_faceset](#)
- [fix_modify AtC consistent_fe_initialization](#)
- [fix_modify AtC mass_matrix](#)
- [fix_modify AtC material](#)
- [fix_modify AtC atomic_charge](#)
- [fix_modify AtC source_integration](#)
- [fix_modify AtC temperature_definition](#)
- [fix_modify AtC track_displacement](#)
- [fix_modify AtC boundary_dynamics](#)
- [fix_modify AtC add_species](#)
- [fix_modify AtC add_molecule](#)
- [fix_modify AtC remove_species](#)
- [fix_modify AtC remove_molecule](#)

Note: a set of example input files with the attendant material files are included with this package

fix atc command   293

Default: None

For detailed exposition of the theory and algorithms please see:

(Wagner) Wagner, GJ; Jones, RE; Templeton, JA; Parks, MA, "An atomistic-to-continuum coupling method for heat transfer in solids." Special Issue of Computer Methods and Applied Mechanics (2008) 197:3351.

(Zimmerman2004) Zimmerman, JA; Webb, EB; Hoyt, JJ;. Jones, RE; Klein, PA; Bammann, DJ, "Calculation of stress in atomistic simulation." Special Issue of Modelling and Simulation in Materials Science and Engineering (2004), 12:S319.

(Zimmerman2010) Zimmerman, JA; Jones, RE; Templeton, JA, "A material frame approach for evaluating continuum variables in atomistic simulations." Journal of Computational Physics (2010), 229:2364.

(Templeton2010) Templeton, JA; Jones, RE; Wagner, GJ, "Application of a field-based method to spatially varying thermal transport problems in molecular dynamics." Modelling and Simulation in Materials Science and Engineering (2010), 18:085007.

(Jones) Jones, RE; Templeton, JA; Wagner, GJ; Olmsted, D; Modine, JA, "Electron transport enhanced molecular dynamics for metals and semi-metals." International Journal for Numerical Methods in Engineering (2010), 83:940.

(Templeton2011) Templeton, JA; Jones, RE; Lee, JW; Zimmerman, JA; Wong, BM, "A long-range electric field solver for molecular dynamics based on atomistic-to-continuum modeling." Journal of Chemical Theory and Computation (2011), 7:1736.

(Mandadapu) Mandadapu, KK; Templeton, JA; Lee, JW, "Polarization as a field variable from molecular dynamics simulations." Journal of Chemical Physics (2013), 139:054115.

Please refer to the standard finite element (FE) texts, e.g. T.J.R Hughes " The finite element method ", Dover 2003, for the basics of FE simulation.

fix ave/atom command

Syntax:

```
fix ID group-ID ave/atom Nevery Nrepeat Nfreq value1 value2 ...
```

- ID, group-ID are documented in <u>fix</u> command
- ave/atom = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[i], f_ID, f_ID[i], v_name

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```

Examples:

```
fix 1 all ave/atom 1 100 100 vx vy vz
fix 1 all ave/atom 10 20 1000 c_my_stress[1]
```

Description:

Use one or more per-atom vectors as inputs every few timesteps, and average them atom by atom over longer timescales. The resulting per-atom averages can be used by other <u>output commands</u> such as the <u>fix ave/spatial</u> or <u>dump custom</u> commands.

The group specified with the command means only atoms within the group have their averages computed. Results are set to 0.0 for atoms not in the group.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a <u>compute</u> or <u>fix</u> or the evaluation of an atom-style <u>variable</u>. In the latter cases, the compute, fix, or variable must produce a per-atom vector, not a global quantity or local quantity. If you wish to time-average global quantities from a compute, fix, or variable, then see the <u>fix ave/time</u> command.

<u>Computes</u> that produce per-atom vectors or arrays are those which have the word *atom* in their style name. See the doc pages for individual <u>fixes</u> to determine which ones produce per-atom vectors or arrays. <u>Variables</u> of style *atom* are the only ones that can be used with this fix since they produce per-atom vectors.

Each per-atom value of each input vector is averaged independently.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a multiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

IMPORTANT NOTE: The x,y,z attributes are values that are re-wrapped inside the periodic box whenever an atom crosses a periodic boundary. Thus if you time average an atom that spends half its time on either side of the periodic box, you will get a value in the middle of the box. If this is not what you want, consider averaging unwrapped coordinates, which can be provided by the compute property/atom command via its xu,yu,zu attributes.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per-atom vector calculated by the compute is used. If a bracketed term containing an index I is appended, the Ith column of the per-atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the per-atom vector calculated by the fix is used. If a bracketed term containing an index I is appended, the Ith column of the per-atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script as an atom-style variable Variables of style *atom* can reference thermodynamic keywords, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to time average.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global scalar or vector quantities are stored by this fix for access by various output commands.

This fix produces a per-atom vector or array which can be accessed by various output commands. A vector is produced if only a single quantity is averaged by this fix. If two or more quantities are averaged, then an array of values is produced. The per-atom values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/histo, fix ave/spatial, fix ave/time, variable,

Default: none

fix ave/correlate command

Syntax:

```
fix ID group-ID ave/correlate Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/correlate = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of correlation time windows to accumulate
- Nfreq = calculate tine window averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
c_ID = global scalar calculated by a compute with ID
c_ID[I] = Ith component of global vector calculated by a compute with ID
f_ID = global scalar calculated by a fix with ID
f_ID[I] = Ith component of global vector calculated by a fix with ID
v_name = global value calculated by an equal-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *type* or *ave* or *start* or *prefactor* or *file* or *overwrite* or *title1* or *title2* or *title3*

```
  type arg = auto or upper or lower or auto/upper or auto/lower or full
     auto = correlate each value with itself
     upper = correlate each value with each succeeding value
     lower = correlate each value with each preceding value
     auto/upper = auto + upper
     auto/lower = auto + lower
     full = correlate each value with every other value, including itself = auto + upper +
  ave args = one or running
     one = zero the correlation accumulation every Nfreq steps
     running = accumulate correlations continuously
  start args = Nstart
     Nstart = start accumulating correlations on this timestep
  prefactor args = value
     value = prefactor to scale all the correlation data by
  file arg = filename
     filename = name of file to output correlation data to
  overwrite arg = none = overwrite output file with only latest output
  title1 arg = string
     string = text to print as 1st line of output file
  title2 arg = string
     string = text to print as 2nd line of output file
  title3 arg = string
     string = text to print as 3rd line of output file
```

Examples:

```
fix 1 all ave/correlate 5 100 1000 c_myTemp file temp.correlate
fix 1 all ave/correlate 1 50 10000 &
        c_thermo_press[1] c_thermo_press[2] c_thermo_press[3] &
        type upper ave running title1 "My correlation data"
```

Description:

Use one or more global scalar values as inputs every few timesteps, calculate time correlations bewteen them at varying time intervals, and average the correlation data over longer timescales. The resulting correlation values can be time integrated by [variables](#) or used by other [output commands](#) such as [thermo_style custom](#),

and can also be written to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a compute or fix or the evaluation of an equal-style variable. In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity. If you wish to spatial- or time-average or histogram per-atom quantities from a compute, fix, or variable, then see the fix ave/spatial, fix ave/atom, or fix ave/histo commands. If you wish to sum a per-atom quantity into a single global quantity, see the compute reduce command.

Computes that produce global quantities are those which do not have the word *atom* in their style name. Only a few fixes produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. Variables of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

The input values must either be all scalars. What kinds of correlations between input values are calculated is determined by the *type* keyword as discussed below.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to calculate correlation data. The input values are sampled every *Nevery* timesteps. The correlation data for the preceding samples is computed on timesteps that are a multiple of *Nfreq*. Consider a set of samples from some initial time up to an output timestep. The initial time could be the beginning of the simulation or the last output time; see the *ave* keyword for options. For the set of samples, the correlation value Cij is calculated as:

```
Cij(delta) = ave(Vi(t)*Vj(t+delta))
```

which is the correlation value between input values Vi and Vj, separated by time delta. Note that the second value Vj in the pair is always the one sampled at the later time. The ave() represents an average over every pair of samples in the set that are separated by time delta. The maximum delta used is of size (*Nrepeat*-1)*Nevery*. Thus the correlation between a pair of input values yields *Nrepeat* correlation datums:

```
Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij((Nrepeat-1)*Nevery)
```

For example, if Nevery=5, Nrepeat=6, and Nfreq=100, then values on timesteps 0,5,10,15,...,100 will be used to compute the final averages on timestep 100. Six averages will be computed: Cij(0), Cij(5), Cij(10), Cij(15), Cij(20), and Cij(25). Cij(10) on timestep 100 will be the average of 19 samples, namely Vi(0)*Vj(10), Vi(5)*Vj(15), Vi(10)*V j20), Vi(15)*Vj(25), ..., Vi(85)*Vj(95), Vi(90)*Vj(100).

Nfreq must be a multiple of *Nevery*; *Nevery* and *Nrepeat* must be non-zero. Also, if the *ave* keyword is set to *one* which is the default, then *Nfreq* >= (*Nrepeat*-1)*Nevery* is required.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used.

Note that there is a compute reduce command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/correlate. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Only equal-style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time correlate.

Additional optional keywords also affect the operation of this fix.

The *type* keyword determines which pairs of input values are correlated with each other. For N input values Vi, for i = 1 to N, let the number of pairs = Npair. Note that the second value in the pair Vi(t)*Vj(t+delta) is always the one sampled at the later time.

- If *type* is set to *auto* then each input value is correlated with itself. I.e. Cii = Vi*Vi, for i = 1 to N, so Npair = N.
- If *type* is set to *upper* then each input value is correlated with every succeeding value. I.e. Cij = Vi*Vj, for i < j, so Npair = N*(N-1)/2.
- If *type* is set to *lower* then each input value is correlated with every preceeding value. I.e. Cij = Vi*Vj, for i > j, so Npair = N*(N-1)/2.
- If *type* is set to *auto/upper* then each input value is correlated with itself and every succeeding value. I.e. Cij = Vi*Vj, for i >= j, so Npair = N*(N+1)/2.
- If *type* is set to *auto/lower* then each input value is correlated with itself and every preceding value. I.e. Cij = Vi*Vj, for i <= j, so Npair = N*(N+1)/2.
- If *type* is set to *full* then each input value is correlated with itself and every other value. I.e. Cij = Vi*Vj, for i,j = 1,N so Npair = N^2.

The *ave* keyword determines what happens to the accumulation of correlation samples every *Nfreq* timesteps. If the *ave* setting is *one*, then the accumulation is restarted or zeroed every *Nfreq* timesteps. Thus the outputs on successive *Nfreq* timesteps are essentially independent of each other. The exception is that the Cij(0) = Vi(T)*Vj(T) value at a timestep T, where T is a multiple of *Nfreq*, contributes to the correlation output both at time T and at time T+Nfreq.

If the *ave* setting is *running*, then the accumulation is never zeroed. Thus the output of correlation data at any timestep is the average over samples accumulated every *Nevery* steps since the fix was defined. it can only be restarted by deleting the fix via the unfix command, or by re-defining the fix by re-specifying it.

The *start* keyword specifies what timestep the accumulation of correlation samples will begin on. The default is step 0. Setting it to a larger value can avoid adding non-equilibrated data to the correlation averages.

The *prefactor* keyword specifies a constant which will be used as a multiplier on the correlation data after it is averaged. It is effectively a scale factor on Vi*Vj, which can be used to account for the size of the time window or other unit conversions.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, an array of correlation data is written to the file. The number of rows is *Nrepeat*, as described above. The number of columns is the Npair+2, also as described above. Thus the file ends up to be a series of these array sections.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these,

so they do not need to be specified.

By default, these header lines are as follows:

```
# Time-correlated data for fix ID
# TimeStep Number-of-time-windows
# Index TimeDelta Ncount valueI*valueJ valueI*valueJ ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the value pairs are replaced with the appropriate fields from the fix ave/correlate command.

Let Sij = a set of time correlation data for input values I and J, namely the *Nrepeat* values:

```
Sij = Cij(0), Cij(Nevery), Cij(2*Nevery), ..., Cij(*Nrepeat-1)*Nevery)
```

As explained below, these datums are output as one column of a global array, which is effectively the correlation matrix.

The *trap* function defined for [equal-style variables](#) can be used to perform a time integration of this vector of datums, using a trapezoidal rule. This is useful for calculating various quantities which can be derived from time correlation data. If a normalization factor is needed for the time integration, it can be included in the variable formula or via the *prefactor* keyword.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global array of values which can be accessed by various [output commands](#). The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = *Nrepeat* and # of columns = Npair+2. The first column has the time delta (in timesteps) between the pairs of input values used to calculate the correlation, as described above. The 2nd column has the number of samples contributing to the correlation average, as described above. The remaining Npair columns are for I,J pairs of the N input values, as determined by the *type* keyword, as described above.

- For *type = auto*, the Npair = N columns are ordered: C11, C22, ..., CNN.
- For *type = upper*, the Npair = N*(N-1)/2 columns are ordered: C12, C13, ..., C1N, C23, ..., C2N, C34, ..., CN-1N.
- For *type = lower*, the Npair = N*(N-1)/2 columns are ordered: C21, C31, C32, C41, C42, C43, ..., CN1, CN2, ..., CNN-1.
- For *type = auto/upper*, the Npair = N*(N+1)/2 columns are ordered: C11, C12, C13, ..., C1N, C22, C23, ..., C2N, C33, C34, ..., CN-1N, CNN.
- For *type = auto/lower*, the Npair = N*(N+1)/2 columns are ordered: C11, C21, C22, C31, C32, C33, C41, ..., C44, CN1, CN2, ..., CNN-1, CNN.
- For *type = full*, the Npair = N^2 columns are ordered: C11, C12, ..., C1N, C21, C22, ..., C2N, C31, ..., C3N, ..., CN1, ..., CNN-1, CNN.

The array values calculated by this fix are treated as "intensive". If you need to divide them by the number of atoms, you must do this in a later processing step, e.g. when using them in a [variable](#).

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

compute, fix ave/time, fix ave/atom, fix ave/spatial, fix ave/histo, variable

Default: none

The option defaults are ave = one, type = auto, start = 0, no file output, title 1,2,3 = strings as described above, and prefactor = 1.0.

fix ave/euler command

Syntax:

```
fix ID group-ID ave/euler nevery N cell_size_relative c
```

- ID, group-ID are documented in fix command
- ave/euler = style name of this fix command
- nevery = obligatory keyword
- n = calculate average values every this many timesteps
- cell_size_relative = obligatory keyword
- c = cell size in multiples of max cutoff

Examples:

```
fix 1 all ave/euler nevery 100 cell_size_relative 4.5
```

Description:

Calculate cell-based averages of velocity, radius, volume fraction, and pressure (-1/3 * trace of the stress tensor) every few timesteps, as specified by the *nevery* keyword. The size of the cells is calculated as multiple of the maximum cutoff, via the *cell_size_relative*. Note that at least a relative cell size of 3 is required.

Note that velocity is favre (mass) averaged, whereas radius is arithmetically averaged. To calculate the stress, this command internally uses a compute stress/atom . It includes the convective term correctly for granular particles with non-zero average velocity (which is not included in compute stress/atom). However, it does not include bond, angle, diahedral or kspace contributions so that the stress tensor finally reads

$$ S_{ab} = -\left[m(v_a - vave_a)(v_b - vave_b) + \frac{1}{2}\sum_{n=1}^{N_p}(r_{1_a}F_{1_b} + r_{2_a}F_{2_b}) \right] $$

where vave is the (cell-based) average velocity. The first term is a kinetic energy contribution for atom *I*. The second term is a pairwise energy contribution where *n* loops over the *Np* neighbors of atom *I*, *r1* and *r2* are the positions of the 2 atoms in the pairwise interaction, and *F1* and *F2* are the forces on the 2 atoms resulting from the pairwise interaction.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes the above-mentioned quantities for output via a dump euler/vtk command. The values can only be accessed on timesteps that are multiples of *nevery* since that is when calculations are performed.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Volume fractions and stresses are calculated based on the assumption of a structured (equidistant regular) grid, so volume fractions and stresses near walls that are not alligned with the grid will be incorrect.

Related commands:

[compute](#), [compute stress/atom](#), [fix ave/atom](#), [fix ave/histo](#), [fix ave/time](#), [fix ave/spatial](#),

Default: none

fix aveforce command

fix aveforce/cuda command

Syntax:

```
fix ID group-ID aveforce fx fy fz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- aveforce = style name of this fix command
- fx,fy,fz = force component values (force units)

```
    any of fx,fy,fz can be a variable (see below)
```
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix pressdown topwall aveforce 0.0 -1.0 0.0
fix 2 bottomwall aveforce NULL -1.0 0.0 region top
fix 2 bottomwall aveforce NULL -1.0 v_oscillate region top
```

Description:

Apply an additional external force to a group of atoms in such a way that every atom experiences the same force. This is useful for pushing on wall or boundary atoms so that the structure of the wall does not change over time.

The existing force is averaged for the group of atoms, component by component. The actual force on each atom is then set to the average value plus the component specified in this command. This means each atom in the group receives the same force.

Any of the fx,fy,fz values can be specified as NULL which means the force in that dimension is not changed. Note that this is not the same as specifying a 0.0 value, since that sets all forces to the same average value without adding in any additional force.

Any of the 3 quantities defining the force components can be specified as an equal-style [variable](#), namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the average force.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent average force.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for

round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various [output commands](#). This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

Restrictions: none

Related commands:

[fix setforce](#), [fix addforce](#)

Default: none

fix ave/histo command

Syntax:

```
fix ID group-ID ave/histo Nevery Nrepeat Nfreq lo hi Nbin value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/histo = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating histogram
- Nfreq = calculate histogram every this many timesteps
- lo,hi = lo/hi bounds within which to histogram
- Nbin = # of histogram bins
- one or more input values can be listed
- value = x, y, z, vx, vy, vz, fx, fy, fz, c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
x,y,z,vx,vy,vz,fx,fy,fz = atom attribute (position, velocity, force component)
c_ID = scalar or vector calculated by a compute with ID
c_ID[I] = Ith component of vector or Ith column of array calculated by a compute with ID
f_ID = scalar or vector calculated by a fix with ID
f_ID[I] = Ith component of vector or Ith column of array calculated by a fix with ID
v_name = value(s) calculated by an equal-style or atom-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *beyond* or *overwrite* or *title1* or *title2* or *title3*

```
mode arg = scalar or vector
   scalar = all input values are scalars
   vector = all input values are vectors
file arg = filename
   filename = name of file to output histogram(s) to
ave args = one or running or window
   one = output a new average value every Nfreq steps
   running = output cumulative average of all previous Nfreq steps
   window M = output average of M most recent Nfreq steps
start args = Nstart
   Nstart = start averaging on this timestep
beyond arg = ignore or end or extra
   ignore = ignore values outside histogram lo/hi bounds
   end = count values outside histogram lo/hi bounds in end bins
   extra = create 2 extra bins for value outside histogram lo/hi bounds
overwrite arg = none = overwrite output file with only latest output
title1 arg = string
   string = text to print as 1st line of output file
title2 arg = string
   string = text to print as 2nd line of output file
title3 arg = string
   string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/histo 100 5 1000 0.5 1.5 50 c_myTemp file temp.histo ave running
fix 1 all ave/histo 100 5 1000 -5 5 100 c_thermo_press[2] c_thermo_press[3] title1 "My output val
fix 1 all ave/histo 1 100 1000 -2.0 2.0 18 vx vy vz mode vector ave running beyond extra
```

Description:

Use one or more values as inputs every few timesteps, histogram them, and average the histogram over longer timescales. The resulting histogram can be used by other [output commands](#), and can also be written to a file.

The group specified with this command is ignored for global and local input values. For per-atom input values, only atoms in the group contribute to the histogram. Note that regardless of the specified group, specified values may represent calculations performed by computes and fixes which store their own "group" definition.

A histogram is simply a count of the number of values that fall within a histogram bin. *Nbins* are defined, with even spacing between *lo* and *hi*. Values that fall outside the lo/hi bounds can be treated in different ways; see the discussion of the *beyond* keyword below.

Each input value can be an atom attribute (position, velocity, force component) or can be the result of a compute or fix or the evaluation of an equal-style or atom-style variable. The set of input values can be either all global, all per-atom, or all local quantities. Inputs of different kinds (e.g. global and per-atom) cannot be mixed. Atom attributes are per-atom vector values. See the doc page for individual "compute" and "fix" commands to see what kinds of quantities they generate.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword.

If *mode* = vector, then the input values may either be vectors or arrays. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/histo commands are equivalent, since the compute com/molecule command creates a global array with 3 columns:

```
compute myCOM all com/molecule
fix 1 all ave/histo 100 1 100 c_myCOM file tmp1.com mode vector
fix 2 all ave/histo 100 1 100 c_myCOM[1] c_myCOM[2] c_myCOM[3] file tmp2.com mode vector
```

The output of this command is a single histogram for all input values combined together, not one histogram per input value. See below for details on the format of the output of this fix.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the histogram. The final histogram is generated on timesteps that are multiple of *Nfreq*. It is averaged over *Nrepeat* histograms, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the histogram cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then input values on timesteps 90,92,94,96,98,100 will be used to compute the final histogram on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging of the histogram is done; a histogram is simply generated on timesteps 100,200,etc.

The atom attribute values (x,y,z,vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global or per-atom or local vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per-atom or local array calculated by the compute is used.

Note that there is a compute reduce command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/histo. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these

commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global or per-atom or local vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global or per-atom or local array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. If *mode* = scalar, then only equal-style variables can be used, which produce a global value. If *mode* = vector, then only atom-style variables can be used, which produce a per-atom vector. See the variable command for details. Note that variables of style *equal* and *atom* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to histogram.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global or per-atom or local vectors, or columns of global or per-atom or local arrays.

The *beyond* keyword determines how input values that fall outside the *lo* to *hi* bounds are treated. Values such that *lo* <= value <= *hi* are assigned to one bin. Values on a bin boundary are assigned to the lower of the 2 bins. If *beyond* is set to *ignore* then values < *lo* and values > *hi* are ignored, i.e. they are not binned. If *beyond* is set to *end* then values < *lo* are counted in the first bin and values > *hi* are counted in the last bin. If *beyond* is set to *extend* then two extra bins are created, so that there are Nbins+2 total bins. Values < *lo* are counted in the first bin and values > *hi* are counted in the last bin (Nbins+1). Values between *lo* and *hi* (inclusive) are counted in bins 2 thru Nbins+1. The "coordinate" stored and printed for these two extra bins is *lo* and *hi*.

The *ave* keyword determines how the histogram produced every *Nfreq* steps are averaged with histograms produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the histograms produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each bin value in the histogram is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or by re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the histograms produced on timesteps that are multiples of *Nfreq* are summed within a moving "window" of time, so that the last M histograms are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the combined histogram of the individual histograms on steps 8000,9000,10000. Outputs on early steps will be sums over less than M histograms if they are not available.

The *start* keyword specifies what timestep histogramming will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed histogram.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one histogram is written to the file. This includes a leading line that contains the timestep, number of bins, the total count of values contributing to the histogram, the count of values that were not histogrammed (see the *beyond* keyword), the minimum value encountered, and the maximum value encountered. The min/max values include values that were not histogrammed. Following the leading line, one line per bin is written into the file. Each line contains the bin #, the coordinate for the center of the bin (between *lo* and *hi*), the count of values in the bin, and the normalized count. The normalized count is the bin count divided by the total count (not including values not histogrammed), so that the normalized values sum to 1.0 across all bins.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Histogram for fix ID
# TimeStep Number-of-bins Total-counts Missing-counts Min-value Max-value
# Bin Coord Count Count/Total
```

In the first line, ID is replaced with the fix-ID. The second line describes the six values that are printed at the first of each section of output. The third describes the 4 values printed for each bin in the histogram.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global vector and global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when a histogram is generated. The global vector has 4 values:

- 1 = total counts in the histogram
- 2 = values that were not histogrammed (see *beyond* keyword)
- 3 = min value of all input values, including ones not histogrammed
- 4 = max value of all input values, including ones not histogrammed

The global array has # of rows = Nbins and # of columns = 3. The first column has the bin coordinate, the 2nd column has the count of values in that histogram bin, and the 3rd column has the bin count divided by the total count (not including missing counts), so that the values in the 3rd column sum to 1.0.

The vector and array values calculated by this fix are all treated as "intensive". If this is not the case, e.g. due to histogramming per-atom input values, then you will need to account for that when interpreting the values produced by this fix.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

[compute](#), [fix ave/atom](#), [fix ave/spatial](#), [fix ave/time](#), [variable](#), [fix ave/correlate](#),

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, beyond = ignore, and title 1,2,3 = strings as described above.

fix ave/spatial command

Syntax:

```
fix ID group-ID ave/spatial Nevery Nrepeat Nfreq dim origin delta ... value1 value2 ... keyword a
```

- ID, group-ID are documented in [fix](#) command
- ave/spatial = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- dim, origin, delta can be repeated 1, 2, or 3 times for 1d, 2d, or 3d bins

```
dim = x or y or z
origin = lower or center or upper or coordinate value (distance units)
delta = thickness of spatial bins in dim (distance units)
```
- one or more input values can be listed
- value = vx, vy, vz, fx, fy, fz, density/mass, density/number, c_ID, c_ID[I], f_ID, f_ID[I], v_name

```
vx,vy,vz,fx,fy,fz = atom attribute (velocity, force component)
density/number, density/mass = number or mass density
c_ID = per-atom vector calculated by a compute with ID
c_ID[I] = Ith column of per-atom array calculated by a compute with ID
f_ID = per-atom vector calculated by a fix with ID
f_ID[I] = Ith column of per-atom array calculated by a fix with ID
v_name = per-atom vector calculated by an atom-style variable with name
```
- zero or more keyword/arg pairs may be appended
- keyword = *norm* or *units* or *file* or *ave* or *overwrite* or *title1* or *title2* or *title3* or *std*

```
  units arg = box or lattice or reduced
    norm arg = all or sample
    region arg = region-ID
      region-ID = ID of region atoms must be in to contribute to spatial averaging
    ave args = one or running or window M
      one = output new average value every Nfreq steps
      running = output cumulative average of all previous Nfreq steps
      window M = output average of M most recent Nfreq steps
    file arg = filename
      filename = file to write results to
    overwrite arg = none = overwrite output file with only latest output
    title1 arg = string
      string = text to print as 1st line of output file
    title2 arg = string
      string = text to print as 2nd line of output file
    title3 arg = string
      string = text to print as 3rd line of output file
    std arg = N1 N2 filename
          N1 = lower limit of particle number per bin for sampling
          N2 = upper limit of particle number per bin for sampling
          filename = file to write results into
```

Examples:

```
fix 1 all ave/spatial 10000 1 10000 z lower 0.02 c_myCentro units reduced &
                  title1 "My output values"
fix 1 flow ave/spatial 100 10 1000 y 0.0 1.0 vx vz norm sample file vel.profile
fix 1 flow ave/spatial 100 5 1000 z lower 1.0 y 0.0 2.5 density/mass ave running
fix 1 all ave/spatial 1000 1 1000 x 0 1e-3 y 0 1e-3 z 0 1e-3 f_tracer[0] file bin_data.dat std 12
```

Description:

Use one or more per-atom vectors as inputs every few timesteps, bin their values spatially into 1d, 2d, or 3d bins based on current atom coordinates, and average the bin values over longer timescales. The resulting bin averages can be used by other output commands such as thermo_style custom, and can also be written to a file.

The group specified with the command means only atoms within the group contribute to bin averages. If the *region* keyword is used, the atom must be in both the group and the specified geometric region in order to contribute to bin averages.

Each listed value can be an atom attribute (position, velocity, force component), a mass or number density, or the result of a compute or fix or the evaluation of an atom-style variable. In the latter cases, the compute, fix, or variable must produce a per-atom quantity, not a global quantity. If you wish to time-average global quantities from a compute, fix, or variable, then see the fix ave/time command.

Computes that produce per-atom quantities are those which have the word *atom* in their style name. See the doc pages for individual fixes to determine which ones produce per-atom quantities. Variables of style *atom* are the only ones that can be used with this fix since all other styles of variable produce global quantities.

The per-atom values of each input vector are binned and averaged independently of the per-atom values in other input vectors.

The size and dimensionality of the bins (1d = layers or slabs, 2d = pencils, 3d = boxes) are determined by the *dim*, *origin*, and *delta* settings and how many times they are specified (1, 2, or 3). See details below.

IMPORTANT NOTE: This fix works by creating an array of size Nbins by Nvalues on each processor. Nbins is the total number of bins; Nvalues is the number of input values specified. Each processor loops over its atoms, tallying its values to the appropriate bin. Then the entire array is summed across all processors. This means that using a large number of bins (easy to do for 2d or 3d bins) will incur an overhead in memory and computational cost (summing across processors), so be careful to use reasonable numbers of bins.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used to bin them and contribute to the average. The final averaged quantities are generated on timesteps that are a multiples of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

Each per-atom property is also averaged over atoms in each bin. Bins can be 1d layers or slabs, 2d pencils, or 3d boxes. This depends on how many times (1, 2, or 3) the *dim*, *origin*, and *delta* settings are specified in the fix ave/spatial command. For 2d or 3d bins, there is no restriction on specifying dim = x before dim = y, or dim = y before dim = z. Bins in a particular *dim* have a bin size in that dimension given by *delta*. Every Nfreq steps, when averaging is being performed and the per-atom property is calculated for the first time, the number of bins and the bin sizes and boundaries are computed. Thus if the simulation box changes size during a simulation, the number of bins and their boundaries may also change. In each dimension, bins are defined relative to a specified *origin*, which may be the lower/upper edge of the simulation box (in *dim*) or its center point, or a specified coordinate value. Starting at the origin, sufficient bins are created in both directions to completely cover the box. On subsequent timesteps every atom is mapped to one of the bins. Atoms beyond

the lowermost/uppermost bin in a dimension are counted in the first/last bin in that dimension.

For orthogonal simulation boxes, the bins are also layers, pencils, or boxes aligned with the xyz coordinate axes. For triclinic (non-orthogonal) simulation boxes, the bins are so that they are parallel to the tilted faces of the simulation box. See this section of the manual for a discussion of the geometry of triclinic boxes in LAMMPS. As described there, a tilted simulation box has edge vectors a,b,c. In that nomenclature, bins in the x dimension have faces with normals in the "b" cross "c" direction. Bins in y have faces normal to the "a" cross "c" direction. And bins in z have faces normal to the "a" cross "b" direction. Note that in order to define the size and position of these bins in an unambiguous fashion, the *units* option must be set to *reduced* when using a triclinic simulation box, as noted below.

The atom attribute values (vx,vy,vz,fx,fy,fz) are self-explanatory. Note that other atom attributes (including atom postitions x,y,z) can be used as inputs to this fix by using the compute property/atom command and then specifying an input value from that compute.

The *density/number* value means the number density is computed in each bin, i.e. a weighting of 1 for each atom. The *density/mass* value means the mass density is computed in each bind, i.e. each atom is weighted by its mass. The resulting density is normalized by the volume of the bin so that units of number/volume or density are output. See the units command doc page for the definition of density for each choice of units, e.g. gram/cm^3.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per-atom vector calculated by the compute is used. If a bracketed integer is appended, the Ith column of the per-atom array calculated by the compute is used. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If no bracketed integer is appended, the per-atom vector calculated by the fix is used. If a bracketed integer is appended, the Ith column of the per-atom array calculated by the fix is used. Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error results. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables of style *atom* can reference thermodynamic keywords and various per-atom attributes, or invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of generating per-atom quantities to spatially average.

Additional optional keywords also affect the operation of this fix.

The *units* keyword determines the meaning of the distance units used for the bin size *delta* and for *origin* if it is a coordinate value. For orthogonal simulation boxes, any of the 3 options may be used. For non-orthogonal (triclinic) simulation boxes, only the *reduced* option may be used.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. A *reduced* value means normalized unitless values between 0 and 1, which represent the lower and upper faces of the simulation box respectively. Thus an *origin* value of 0.5 means the center of the box in any dimension. A *delta* value of 0.1 means 10 bins span the box in that dimension.

Consider a non-orthogonal box, with bins that are 1d layers or slabs in the x dimension. No matter how the box is tilted, an *origin* of 0.0 means start layers at the lower "b" cross "c" plane of the simulation box and an *origin* of 1.0 means to start layers at the upper "b" cross "c" face of the box. A *delta* value of 0.1 means there will be 10 layers from 0.0 to 1.0, regardless of the current size or shape of the simulation box.

The *norm* keyword affects how averaging is done for the output produced every *Nfreq* timesteps. For an *all* setting, a bin quantity is summed over all atoms in all *Nrepeat* samples, as is the count of atoms in the bin. The printed value for the bin is Total-quantity / Total-count. In other words it is an average over the entire *Nfreq* timescale.

For a *sample* setting, the bin quantity is summed over atoms for only a single sample, as is the count, and a "average sample value" is computed, i.e. Sample-quantity / Sample-count. The printed value for the bin is the average of the *Nrepeat* "average sample values", In other words it is an average of an average.

The *ave* keyword determines how the bin values produced every *Nfreq* steps are averaged with bin values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the bin values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cumulative sense before being output. Each output bin value is thus the average of the bin value produced on that timestep with all preceding values for the same bin. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the [unfix](#) command, or re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the bin values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values for the same bin are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual bin values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *file* keyword allows a filename to be specified. Every *Nfreq* timesteps, a section of bin info will be written to a text file in the following format. A line with the timestep and number of bin is written. Then one line per bin is written, containing the bin ID (1-N), the coordinate of the center of the bin, the number of atoms in the bin, and one or more calculated values. The number of values in each line corresponds to the number of values specified in the fix ave/spatial command. The number of atoms and the value(s) are average quantities. If the value of the *units* keyword is *box* or *lattice*, the "coord" is printed in box units. If the value of the *units* keyword is *reduced*, the "coord" is printed in reduced units (0-1).

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows:

```
# Spatial-averaged data for fix ID and group name
# Timestep Number-of-bins
# Bin Coord1 Coord2 Coord3 Count value1 value2 ...
```

In the first line, ID and name are replaced with the fix-ID and group name. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/spatial command. The Coord2 and Coord3 entries in the third line only appear for 2d and 3d bins respectively. For 1d bins, the word Coord1 is replaced by just Coord.

If the *std* keyword is set, mean and standard deviation of the specified values (value1, value2, etc.) over samples of a defined size are calculated. The sample size has to be defined by a lower limit (N1) and an upper limit (N2>N1). All bins containing a particle count between N1 and N2 (including N1 and N2) are used as samples. Every *Nfreq* timestep a line is written to a file specified after N1 and N2, including the following numbers: timestep, total number of atoms, total number of bins, maximum number of atoms per bin, number of empty bins, number of bins including less atoms than N1, number of bins including more atoms than N2, number of samples, average number of atoms per sample, followed by three quantities for each defined value: true average (over all atoms), average over the chosen samples, standard deviation over the chosen samples. For the calculation of the standard deviation the (known) true average is used instead of the samples average (the latter is only an estimate for the true average!).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed. The global array has # of rows = Nbins and # of columns = Ndim+1+Nvalues, where Ndim = 1,2,3 for 1d,2d,3d bins. The first 1 or 2 or 3 columns have the bin coordinates (center of the bin) in the appropriate dimensions, the next column has the count of atoms in that bin, and the remaining columns are the Nvalue quantities. When the array is accessed with an I that exceeds the current number of bins, than a 0.0 is returned by the fix instead of an error, since the number of bins can vary as a simulation runs, depending on the simulation box size. 2d or 3d bins are ordered so that the last dimension(s) vary fastest. The array values calculated by this fix are "intensive", since they are already normalized by the count of atoms in each bin.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

When the *ave* keyword is set to *running* or *window* then the number of bins must remain the same during the simulation, so that the appropriate averaging can be done. This will be the case if the simulation box size doesn't change or if the *units* keyword is set to *reduced*.

Related commands:

compute, fix ave/atom, fix ave/histo, fix ave/time, variable, fix ave/correlate,

Default:

The option defaults are units = lattice, norm = all, no file output, and ave = one, title 1,2,3 = strings as described above.

fix ave/time command

Syntax:

```
fix ID group-ID ave/time Nevery Nrepeat Nfreq value1 value2 ... keyword args ...
```

- ID, group-ID are documented in [fix](#) command
- ave/time = style name of this fix command
- Nevery = use input values every this many timesteps
- Nrepeat = # of times to use input values for calculating averages
- Nfreq = calculate averages every this many timesteps
- one or more input values can be listed
- value = c_ID, c_ID[N], f_ID, f_ID[N], v_name

```
    c_ID = global scalar or vector calculated by a compute with ID
    c_ID[I] = Ith component of global vector or Ith column of global array calculated by a c
    f_ID = global scalar or vector calculated by a fix with ID
    f_ID[I] = Ith component of global vector or Ith column of global array calculated by a f
    v_name = global value calculated by an equal-style variable with name
```

- zero or more keyword/arg pairs may be appended
- keyword = *mode* or *file* or *ave* or *start* or *off* or *overwrite* or *title1* or *title2* or *title3*

```
  mode arg = scalar or vector
     scalar = all input values are global scalars
     vector = all input values are global vectors or global arrays
  ave args = one or running or window M
    one = output a new average value every Nfreq steps
    running = output cummulative average of all previous Nfreq steps
    window M = output average of M most recent Nfreq steps
  start args = Nstart
    Nstart = start averaging on this timestep
  off arg = M = do not average this value
    M = value # from 1 to Nvalues
  file arg = filename
    filename = name of file to output time averages to
  overwrite arg = none = overwrite output file with only latest output
  title1 arg = string
    string = text to print as 1st line of output file
  title2 arg = string
    string = text to print as 2nd line of output file
  title3 arg = string
    string = text to print as 3rd line of output file, only for vector mode
```

Examples:

```
fix 1 all ave/time 100 5 1000 c_myTemp c_thermo_temp file temp.profile
fix 1 all ave/time 100 5 1000 c_thermo_press[2] ave window 20 &
                             title1 "My output values"
fix 1 all ave/time 1 100 1000 f_indent f_indent[1] file temp.indent off 1
```

Description:

Use one or more global values as inputs every few timesteps, and average them over longer timescales. The resulting averages can be used by other [output commands](#) such as [thermo_style custom](#), and can also be written to a file. Note that if no time averaging is done, this command can be used as a convenient way to simply output one or more global values to a file.

The group specified with this command is ignored. However, note that specified values may represent calculations performed by computes and fixes which store their own "group" definitions.

Each listed value can be the result of a compute or fix or the evaluation of an equal-style variable. In each case, the compute, fix, or variable must produce a global quantity, not a per-atom or local quantity. If you wish to spatial- or time-average or histogram per-atom quantities from a compute, fix, or variable, then see the fix ave/spatial, fix ave/atom, or fix ave/histo commands. If you wish to sum a per-atom quantity into a single global quantity, see the compute reduce command.

Computes that produce global quantities are those which do not have the word *atom* in their style name. Only a few fixes produce global quantities. See the doc pages for individual fixes for info on which ones produce such values. Variables of style *equal* are the only ones that can be used with this fix. Variables of style *atom* cannot be used, since they produce per-atom values.

The input values must either be all scalars or all vectors (or arrays), depending on the setting of the *mode* keyword. In both cases, the averaging is performed independently on each input value. I.e. each input scalar is averaged independently and each element of each input vector (or array) is averaged independently.

If *mode* = vector, then the input values may either be vectors or arrays and all must be the same "length", which is the length of the vector or number of rows in the array. If a global array is listed, then it is the same as if the individual columns of the array had been listed one by one. E.g. these 2 fix ave/time commands are equivalent, since the compute rdf command creates, in this case, a global array with 3 columns, each of length 50:

```
compute myRDF all rdf 50 1 2
fix 1 all ave/time 100 1 100 c_myRDF file tmp1.rdf mode vector
fix 2 all ave/time 100 1 100 c_myRDF[1] c_myRDF[2] c_myRDF[3] file tmp2.rdf mode vector
```

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the input values will be used in order to contribute to the average. The final averaged quantities are generated on timesteps that are a mlutiple of *Nfreq*. The average is over *Nrepeat* quantities, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average value cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then values on timesteps 90,92,94,96,98,100 will be used to compute the final average on timestep 100. Similarly for timesteps 190,192,194,196,198,200 on timestep 200, etc. If Nrepeat=1 and Nfreq = 100, then no time averaging is done; values are simply generated on timesteps 100,200,etc.

If a value begins with "c_", a compute ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the compute is used. If a bracketed term is appended, the Ith element of the global vector calculated by the compute is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the compute is used. Or if the compute calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the compute is used.

Note that there is a compute reduce command which can sum per-atom quantities into a global scalar or vector which can thus be accessed by fix ave/time. Or it can be a compute defined not in your input script, but by thermodynamic output or other fixes such as fix nvt or fix temp/rescale. See the doc pages for these commands which give the IDs of these computes. Users can also write code for their own compute styles and add them to LAMMPS.

If a value begins with "f_", a fix ID must follow which has been previously defined in the input script. If *mode* = scalar, then if no bracketed term is appended, the global scalar calculated by the fix is used. If a bracketed

term is appended, the Ith element of the global vector calculated by the fix is used. If *mode* = vector, then if no bracketed term is appended, the global vector calculated by the fix is used. Or if the fix calculates an array, all of the columns of the global array are used as if they had been specified as individual vectors (see description above). If a bracketed term is appended, the Ith column of the global array calculated by the fix is used.

Note that some fixes only produce their values on certain timesteps, which must be compatible with *Nevery*, else an error will result. Users can also write code for their own fix styles and add them to LAMMPS.

If a value begins with "v_", a variable name must follow which has been previously defined in the input script. Variables can only be used as input for *mode* = scalar. Only equal-style variables can be referenced. See the variable command for details. Note that variables of style *equal* define a formula which can reference individual atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when they are evaluated, so this is a very general means of specifying quantities to time average.

Additional optional keywords also affect the operation of this fix.

If the *mode* keyword is set to *scalar*, then all input values must be global scalars, or elements of global vectors. If the *mode* keyword is set to *vector*, then all input values must be global vectors, or columns of global arrays. They can also be global arrays, which are converted into a series of global vectors (one per column), as explained above.

The *ave* keyword determines how the values produced every *Nfreq* steps are averaged with values produced on previous steps that were multiples of *Nfreq*, before they are accessed by another output command or written to a file.

If the *ave* setting is *one*, then the values produced on timesteps that are multiples of *Nfreq* are independent of each other; they are output as-is without further averaging.

If the *ave* setting is *running*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged in a cummulative sense before being output. Each output value is thus the average of the value produced on that timestep with all preceding values. This running average begins when the fix is defined; it can only be restarted by deleting the fix via the unfix command, or by re-defining the fix by re-specifying it.

If the *ave* setting is *window*, then the values produced on timesteps that are multiples of *Nfreq* are summed and averaged within a moving "window" of time, so that the last M values are used to produce the output. E.g. if M = 3 and Nfreq = 1000, then the output on step 10000 will be the average of the individual values on steps 8000,9000,10000. Outputs on early steps will average over less than M values if they are not available.

The *start* keyword specifies what timestep averaging will begin on. The default is step 0. Often input values can be 0.0 at time 0, so setting *start* to a larger value can avoid including a 0.0 in a running or windowed average.

The *off* keyword can be used to flag any of the input values. If a value is flagged, it will not be time averaged. Instead the most recent input value will always be stored and output. This is useful if one of more of the inputs produced by a compute or fix or variable are effectively constant or are simply current values. E.g. they are being written to a file with other time-averaged values for purposes of creating well-formatted output.

The *file* keyword allows a filename to be specified. Every *Nfreq* steps, one quantity or vector of quantities is written to the file for each input value specified in the fix ave/time command. For *mode* = scalar, this means a single line is written each time output is performed. Thus the file ends up to be a series of lines, i.e. one column of numbers for each input value. For *mode* = vector, an array of numbers is written each time output is performed. The number of rows is the length of the input vectors, and the number of columns is the number of values. Thus the file ends up to be a series of these array sections.

The *overwrite* keyword will continuously overwrite the output file with the latest output, so that it only contains one timestep worth of output. This option can only be used with the *ave running* setting.

The *title1* and *title2* and *title3* keywords allow specification of the strings that will be printed as the first 2 or 3 lines of the output file, assuming the *file* keyword was used. LAMMPS uses default values for each of these, so they do not need to be specified.

By default, these header lines are as follows for *mode* = scalar:

```
# Time-averaged data for fix ID
# TimeStep value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. In the second line the values are replaced with the appropriate fields from the fix ave/time command. There is no third line in the header of the file, so the *title3* setting is ignored when *mode* = scalar.

By default, these header lines are as follows for *mode* = vector:

```
# Time-averaged data for fix ID
# TimeStep Number-of-rows
# Row value1 value2 ...
```

In the first line, ID is replaced with the fix-ID. The second line describes the two values that are printed at the first of each section of output. In the third line the values are replaced with the appropriate fields from the fix ave/time command.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a global scalar or global vector or global array which can be accessed by various output commands. The values can only be accessed on timesteps that are multiples of *Nfreq* since that is when averaging is performed.

A scalar is produced if only a single input value is averaged and *mode* = scalar. A vector is produced if multiple input values are averaged for *mode* = scalar, or a single input value for *mode* = vector. In the first case, the length of the vector is the number of inputs. In the second case, the length of the vector is the same as the length of the input vector. An array is produced if multiple input values are averaged and *mode* = vector. The global array has # of rows = length of the input vectors and # of columns = number of inputs.

If the fix prouduces a scalar or vector, then the scalar and each element of the vector can be either "intensive" or "extensive". If the fix produces an array, then all elements in the array must be the same, either "intensive" or "extensive". If a compute or fix provides the value being time averaged, then the compute or fix determines whether the value is intensive or extensive; see the doc page for that compute or fix for further info. Values produced by a variable are treated as intensive.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute, fix ave/atom, fix ave/spatial, fix ave/histo, variable, fix ave/correlate,

Default: none

The option defaults are mode = scalar, ave = one, start = 0, no file output, title 1,2,3 = strings as described above, and no off settings for any input values.

fix bond/break command

Syntax:

```
fix ID group-ID bond/break Nevery bondtype Rmax keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- bond/break = style name of this fix command
- Nevery = attempt bond breaking every this many steps
- bondtype = type of bonds to break
- Rmax = bond longer than Rmax can break (distance units)
- zero or more keyword/value pairs may be appended to args
- keyword = *prob*

```
  prob values = fraction seed
      fraction = break a bond with this probability if otherwise eligible
      seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/break 10 2 1.2
fix 5 polymer bond/break 1 1 2.0 prob 0.5 49829
```

Description:

Break bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model the dissolution of a polymer network due to stretching of the simulation box or other deformations. In this context, a bond means an interaction between a pair of atoms computed by the [bond_style](#) command. Once the bond is broken it will be permanently deleted. This is different than a [pairwise](#) bond-order potential such as Tersoff or AIREBO which infers bonds and many-body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds from timestep to timestep as atoms move.

A check for possible bond breakage is performed every *Nevery* timesteps. If two bonded atoms I,J are further than a distance *Rmax* of each other, if the bond is of type *bondtype*, and if both I and J are in the specified fix group, then I,J is labeled as a "possible" bond to break.

If several bonds involving an atom are stretched, it may have multiple possible bonds to break. Every atom checks its list of possible bonds to break and labels the longest such bond as its "sole" bond to break. After this is done, if atom I is bonded to atom J in its sole bond, and atom J is bonded to atom I in its sole bond, then the I,J bond is "eligible" to be broken.

Note that these rules mean an atom will only be part of at most one broken bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk > Rij), then this means atom I will not be part of a broken bond on this timestep, even if it has other possible bond partners.

The *prob* keyword can effect whether an eligible bond is actually broken. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only broken if the random number < fraction.

When a bond is broken, data structures within LAMMPS that store bond topology are updated to reflect the breakage. This can also affect subsequent computation of pairwise interactions involving the atoms in the bond. See the Restriction section below for additional information.

Computationally, each timestep this fix operates, it loops over bond lists and computes distances between pairs of bonded atoms in the list. It also communicates between neighboring processors to coordinate which bonds are broken. Thus it will increase the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the dump local command.

IMPORTANT NOTE: Breaking a bond typically alters the energy of a system. You should be careful not to choose bond breaking criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and break it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will be dramatically released when the bond is broken. More generally, you may need to thermostat your system to compensate for energy changes resulting from broken bonds.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various output commands. The vector values calculated by this fix are "intensive".

These are the 2 quantities:

- (1) # of bonds broken on the most recent breakage timestep
- (2) cummulative # of bonds broken

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is broken, you might wish to turn off angle and dihedral interactions that include that bond. However, LAMMPS does not check for these angles and dihedrals, even if your simulation defines an angle_style or dihedral_style.

This fix requires that the pairwise weightings defined by the special_bonds command be 0,1,1 for 1-2, 1-3, and 1-4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J is turned off when a bond between them exists and will be turned on when the bond is broken. It also means that the pairwise interaction of I with J's other bond partners is unaffected by the existence of the bond.

Related commands:

fix bond/create, fix bond/swap, dump local, special_bonds

Default:

The option defaults are prob = 1.0.

fix bond/create command

Syntax:

```
fix ID group-ID bond/create Nevery itype jtype Rmin bondtype keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- bond/create = style name of this fix command
- Nevery = attempt bond creation every this many steps
- itype,jtype = atoms of itype can bond to atoms of jtype
- Rmin = 2 atoms separated by less than Rmin can bond (distance units)
- bondtype = type of created bonds
- zero or more keyword/value pairs may be appended to args
- keyword = *iparam* or *jparam* or *prob*

```
  iparam values = maxbond, newtype
     maxbond = max # of bonds of bondtype the itype atom can have
     newtype = change the itype atom to this type when maxbonds exist
   jparam values = maxbond, newtype
     maxbond = max # of bonds of bondtype the jtype atom can have
     newtype = change the jtype atom to this type when maxbonds exist
  prob values = fraction seed
     fraction = create a bond with this probability if otherwise eligible
     seed = random number seed (positive integer)
```

Examples:

```
fix 5 all bond/create 10 1 2 0.8 1
fix 5 all bond/create 1 3 3 0.8 1 prob 0.5 85784 iparam 2 3
```

Description:

Create bonds between pairs of atoms as a simulation runs according to specified criteria. This can be used to model cross-linking of polymers, the formation of a percolation network, etc. In this context, a bond means an interaction between a pair of atoms computed by the [bond_style](#) command. Once the bond is created it will be permanently in place. This is different than a [pairwise](#) bond-order potential such as Tersoff or AIREBO which infers bonds and many-body interactions based on the current geometry of a small cluster of atoms and effectively creates and destroys bonds from timestep to timestep as atoms move.

A check for possible new bonds is performed every *Nevery* timesteps. If two atoms I,J are within a distance *Rmin* of each other, if I is of atom type *itype*, if J is of atom type *jtype*, if both I and J are in the specified fix group, if a bond does not already exist between I and J, and if both I and J meet their respective *maxbond* requirement (explained below), then I,J is labeled as a "possible" bond pair.

If several atoms are close to an atom, it may have multiple possible bond partners. Every atom checks its list of possible bond partners and labels the closest such partner as its "sole" bond partner. After this is done, if atom I has atom J as its sole partner, and atom J has atom I as its sole partner, then the I,J bond is "eligible" to be formed.

Note that these rules mean an atom will only be part of at most one created bond on a given timestep. It also means that if atom I chooses atom J as its sole partner, but atom J chooses atom K is its sole partner (due to Rjk < Rij), then this means atom I will not form a bond on this timestep, even if it has other possible bond partners.

It is permissible to have *itype* = *jtype*. *Rmin* must be <= the pairwise cutoff distance between *itype* and *jtype* atoms, as defined by the pair_style command.

The *iparam* and *jparam* keywords can be used to limit the bonding functionality of the participating atoms. Each atom keeps track of how many bonds of *bondtype* it already has. If atom I of itype already has *maxbond* bonds (as set by the *iparam* keyword), then it will not form any more. Likewise for atom J. If *maxbond* is set to 0, then there is no limit on the number of bonds that can be formed with that atom.

The *newtype* value for *iparam* and *jparam* can be used to change the atom type of atom I or J when it reaches *maxbond* number of bonds of type *bondtype*. This means it can now interact in a pairwise fashion with other atoms in a different way by specifying different pair_coeff coefficients. If you do not wish the atom type to change, simply specify *newtype* as *itype* or *jtype*.

The *prob* keyword can also effect whether an eligible bond is actually created. The *fraction* setting must be a value between 0.0 and 1.0. A uniform random number between 0.0 and 1.0 is generated and the eligible bond is only created if the random number < fraction.

Any bond that is created is assigned a bond type of *bondtype*. Data structures within LAMMPS that store bond topology are updated to reflect the new bond. This can also affect subsequent computation of pairwise interactions involving the atoms in the bond. See the Restriction section below for additional information.

IMPORTANT NOTE: To create a new bond, the internal LAMMPS data structures that store this information must have space for it. When LAMMPS is initialized from a data file, the list of bonds is scanned and the maximum number of bonds per atom is tallied. If some atom will acquire more bonds than this limit as this fix operates, then the "extra bonds per atom" parameter in the data file header must be set to allow for it. See the read_data command for more details. Note that if this parameter needs to be set, it means a data file must be used to initialize the system, even if it initially has no bonds. A data file with no atoms can be used if you wish to add unbonded atoms via the create atoms command, e.g. for a percolation simulation.

IMPORTANT NOTE: LAMMPS also maintains a data structure that stores a list of 1st, 2nd, and 3rd neighbors of each atom (in the bond topology of the system) for use in weighting pairwise interactions for bonded atoms. Adding a bond adds a single entry to this list. The "extra" keyword of the special_bonds command should be used to leave space for new bonds if the maximum number of entries for any atom will be exceeded as this fix operates. See the special_bonds command for details.

Note that even if your simulation starts with no bonds, you must define a bond_style and use the bond_coeff command to specify coefficients for the *bondtype*. Similarly, if new atom types are specified by the *iparam* or *jparam* keywords, they must be within the range of atom types allowed by the simulation and pairwise coefficients must be specified for the new types.

Computationally, each timestep this fix operates, it loops over neighbor lists and computes distances between pairs of atoms in the list. It also communicates between neighboring processors to coordinate which bonds are created. Thus it roughly doubles the cost of a timestep. Thus you should be cautious about invoking this fix too frequently.

You can dump out snapshots of the current bond topology via the dump local command.

IMPORTANT NOTE: Creating a bond typically alters the energy of a system. You should be careful not to choose bond creation criteria that induce a dramatic change in energy. For example, if you define a very stiff harmonic bond and create it when 2 atoms are separated by a distance far from the equilibribum bond length, then the 2 atoms will oscillate dramatically when the bond is formed. More generally, you may need to thermostat your system to compensate for energy changes resulting from created bonds.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](). None of the [fix_modify]() options are relevant to this fix.

This fix computes two statistics which it stores in a global vector of length 2, which can be accessed by various [output commands](). The vector values calculated by this fix are "intensive".

These are the 2 quantities:

- (1) # of bonds created on the most recent creation timestep
- (2) cummulative # of bonds created

No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions:

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS]() section for more info.

Currently, there are 2 restrictions for using this fix. We may relax these in the future if there are new models that would be enabled by it.

When a bond is created, you might wish to induce new angle and dihedral interactions that include that bond. However, LAMMPS does not create these angles and dihedrals, even if your simulation defines an [angle_style]() or [dihedral_style]().

This fix requires that the pairwise weightings defined by the [special_bonds]() command be 0,1,1 for 1-2, 1-3, and 1-4 neighbors within the bond topology. This effectively means that the pairwise interaction between atoms I and J will be turned off when a bond between them is created. It also means that the pairwise interaction of I with J's other bond partners will be unaffected by the new bond.

Related commands:

[fix bond/break](), [fix bond/swap](), [dump local](), [special_bonds]()

Default:

The option defaults are iparam = (0,itype), jparam = (0,jtype), and prob = 1.0.

fix bond/swap command

Syntax:

```
fix ID group-ID bond/swap fraction cutoff seed
```

- ID, group-ID are documented in fix command
- bond/swap = style name of this fix command
- fraction = fraction of group atoms to consider for swapping
- cutoff = distance at which swapping will be considered (distance units)
- seed = random # seed (positive integer)

Examples:

```
fix 1 all bond/swap 0.5 1.3 598934
```

Description:

In a simulation of polymer chains, this command attempts to swap bonds between two different chains, effectively grafting the end of one chain onto another chain and vice versa. This is done via Monte Carlo rules using the Boltzmann acceptance criterion. The purpose is to equilibrate the polymer chain conformations more rapidly than dynamics alone would do it, by enabling instantaneous large conformational changes in a dense polymer melt. The polymer chains should thus more rapidly converge to the proper end-to-end distances and radii of gyration. It is designed for use with systems of FENE or harmonic bead-spring polymer chains where each polymer is a linear chain of monomers, but LAMMPS does not enforce this requirement, i.e. any bond_style can be used.

A schematic of the kinds of bond swaps that can occur is shown here:
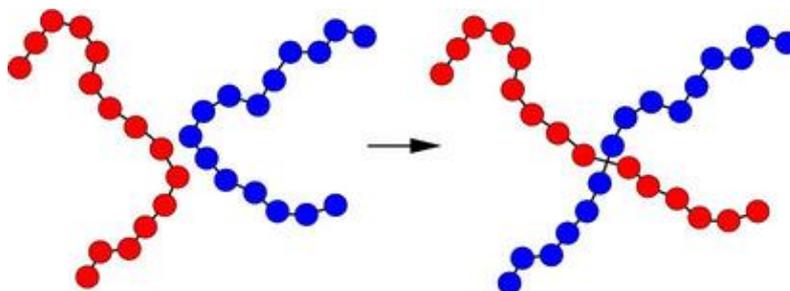
On the left, the red and blue chains have two monomers A1 and B1 close to each other, which are currently bonded to monomers A2 and B2 respectively within their own chains. The bond swap operation will attempt to delete the A1-A2 and B1-B2 bonds and replace them with A1-B2 and B1-A2 bonds. If the swap is energetically favorable, the two chains on the right are the result and each polymer chain has undergone a dramatic conformational change. This reference provides more details on how the algorithm works and its application: (Sides).

The bond swapping operation is invoked each time neighbor lists are built during a simulation, since it potentially alters the list of which neighbors are considered for pairwise interaction. At each reneighboring step, each processor considers a random specified *fraction* of its atoms as potential swapping monomers for this timestep. Choosing a small *fraction* value can reduce the likelihood of a reverse swap occurring soon after an initial swap.

For each monomer A1, its neighbors are examined to find a possible B1 monomer. Both A1 and B1 must be in the fix group, their separation must be less than the specified *cutoff*, and the molecule IDs of A1 and B1 must be the same (see below). If a suitable partner is found, the energy change due to swapping the 2 bonds is computed. This includes changes in pairwise, bond, and angle energies due to the altered connectivity of the 2 chains. Dihedral and improper interactions are not allowed to be defined when this fix is used.

If the energy decreases due to the swap operation, the bond swap is accepted. If the energy increases it is accepted with probability exp(-delta/kT) where delta is the increase in energy, k is the Boltzmann constant, and T is the current temperature of the system. Whether the swap is accepted or rejected, no other swaps are attempted by this processor on this timestep.

The criterion for matching molecule IDs is how bond swaps performed by this fix conserve chain length. To use this features you must setup the molecule IDs for your polymer chains in a certain way, typically in the data file, read by the read_data comand. Consider a system of 6-mer chains. You have 2 choices. If the molecule IDs for monomers on each chain are set to 1,2,3,4,5,6 then swaps will conserve chain length. For a particular momoner there will be only one other monomer on another chain which is a potential swap partner. If the molecule IDs for monomers on each chain are set to 1,2,3,3,2,1 then swaps will conserve chain length but swaps will be able to occur at either end of a chain. Thus for a particular monomer there will be 2 possible swap partners on another chain. In this scenario, swaps can also occur within a single chain, i.e. the two ends of a chain swap with each other.

IMPORTANT NOTE: If your simulation uses molecule IDs in the usual way, where all monomers on a single chain are assigned the same ID (different for each chain), then swaps will only occur within the same chain. If you assign the same molecule ID to all monomers in all chains then inter-chain swaps will occur, but they will not conserve chain length. Neither of these scenarios is probably not what you want for this fix.

IMPORTANT NOTE: When a bond swap occurs the image flags of monomers in the new polymer chains can become inconsistent. See the dump command for a discussion of image flags. This is not an issue for running dynamics, but can affect calculation of some diagnostic quantities or the printing of unwrapped coordinates to a dump file.

This fix computes a temperature each time it is invoked for use by the Boltzmann criterion. To do this, the fix creates its own compute of style *temp*, as if this command had been issued:

```
compute fix-ID_temp all temp
```

See the compute temp command for details. Note that the ID of the new compute is the fix-ID with underscore + "temp" appended and the group for the new compute is "all", so that the temperature of the entire system is used.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the compute_modify command or print this temperature during thermodyanmic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Restart, fix_modify, thermo output, run start/stop, minimize info:

No information about this fix is written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior. Also note that each processor generates possible swaps independently of other processors. Thus if you repeat the same simulation on a different number of processors, the specific swaps performed will be different.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used to compute the temperature for the Boltzmann criterion.

This fix computes two statistical quantities as a global 2-vector of output, which can be accessed by various output commands. The first component of the vector is the cummulative number of swaps performed by all processors. The second component of the vector is the cummulative number of swaps attempted (whether accepted or rejected). Note that a swap "attempt" only occurs when swap partners meeting the criteria described above are found on a particular timestep. The vector values calculated by this fix are "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The settings of the "special_bond" command must be 0,1,1 in order to use this fix, which is typical of bead-spring chains with FENE or harmonic bonds. This means that pairwise interactions between bonded atoms are turned off, but are turned on between atoms two or three hops away along the chain backbone.

Currently, energy changes in dihedral and improper interactions due to a bond swap are not considered. Thus a simulation that uses this fix cannot use a dihedral or improper potential.

Related commands: none

Default: none

(Sides) Sides, Grest, Stevens, Plimpton, J Polymer Science B, 42, 199-208 (2004).

fix box/relax command

Syntax:

```
fix ID group-ID box/relax keyword value ...
```

- ID, group-ID are documented in <u>fix</u> command
- box/relax = style name of this fix command

```
one or more keyword value pairs may be appended
keyword = iso or aniso or tri or x or y or z or xy or yz or xz or couple or nreset or vmax
  iso or aniso or tri value = Ptarget = desired pressure (pressure units)
  x or y or z or xy or yz or xz value = Ptarget = desired pressure (pressure units)
  couple = none or xyz or xy or yz or xz
  nreset value = reset reference cell every this many minimizer iterations
  vmax value = fraction = max allowed volume change in one iteration
  dilate value = all or partial
  scaleyz value = yes or no = scale yz with lz
  scalexz value = yes or no = scale xz with lz
  scalexy value = yes or no = scale xy with ly
  fixedpoint values = x y z
    x,y,z = perform relaxation dilation/contraction around this point (distance units)
```

Examples:

```
fix 1 all box/relax iso 0.0 vmax 0.001
fix 2 water box/relax aniso 0.0 dilate partial
fix 2 ice box/relax tri 0.0 couple xy nreset 100
```

Description:

Apply an external pressure or stress tensor to the simulation box during an <u>energy minimization</u>. This allows the box size and shape to vary during the iterations of the minimizer so that the final configuration will be both an energy minimum for the potential energy of the atoms, and the system pressure tensor will be close to the specified external tensor. Conceptually, specifying a positive pressure is like squeezing on the simulation box; a negative pressure typically allows the box to expand.

The external pressure tensor is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during the mimimization.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The <u>create_box</u>, <u>read data</u>, and <u>read_restart</u> commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures *Ptarget* for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For example, if the *y* keyword is used, the y-box length will change during the minimization. If the *xy* keyword is used, the xy tilt factor will change. A box dimension will not change if that component is not specified.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

When the size of the simulation box changes, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. This can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when relaxing triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LAMMPS, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

The *fixedpoint* keyword specifies the fixed point for cell relaxation. By default, it is the center of the box. Whatever point is chosen will not move during the simulation. For example, if the lower periodic boundaries pass through (0,0,0), and this point is provided to *fixedpoint*, then the lower periodic boundaries will remain at (0,0,0), while the upper periodic boundaries will move twice as far. In all cases, the particle positions at each iteration are unaffected by the chosen value, except that all particles are displaced by the same amount, different on each iteration.

IMPORTANT NOTE: Appling an external pressure to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt factors, i.e. a dramatically deformed simulation box. This typically indicates that there is something badly wrong with how the simulation was constructed. The two most common sources of this error are applying a shear stress to a liquid system or specifying an external shear stress tensor that exceeds the yield stress of the solid. In either case the minimization may converge to a bogus conformation or not converge at all. Also note that if the box shape tilts to an extreme shape, LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Ptarget* values for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
y Ptarget
z Ptarget
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Ptarget" is the same as specifying these 4 keywords:

```
x Ptarget
y Ptarget
z Ptarget
```

```
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using "tri Ptarget" is the same as specifying these 7 keywords:

```
x Ptarget
y Ptarget
z Ptarget
xy 0.0
yz 0.0
xz 0.0
couple none
```

The *vmax* keyword can be used to limit the fractional change in the volume of the simulation box that can occur in one iteration of the minimizer. If the pressure is not settling down during the minimization this can be because the volume is fluctuating too much. The specified fraction must be greater than 0.0 and should be << 1.0. A value of 0.001 means the volume cannot change by more than 1/10 of a percent in one iteration when *couple xyz* has been specified. For any other case it means no linear dimension of the simulation box can change by more than 1/10 of a percent.

With this fix, the potential energy used by the minimizer is augmented by an additional energy provided by the fix. The overall objective function then is:

$$E = U + P_t \left(V - V_0 \right) + E_{strain}$$

where *U* is the system potential energy, P_t is the desired hydrostatic pressure, *V* and *V*_0 are the system and reference volumes, respectively. *E_strain* is the strain energy expression proposed by Parrinello and Rahman (Parrinello1981). Taking derivatives of *E* w.r.t. the box dimensions, and setting these to zero, we find that at the minimum of the objective function, the global system stress tensor **P** will satisfy the relation:

$$\mathbf{P} = P_t \mathbf{I} + \mathbf{S}_t \left(\mathbf{h}_0^{-1} \right)^t \mathbf{h}_{0d}$$

where **I** is the identity matrix, **h**_0 is the box dimension tensor of the reference cell, and **h**_0*d* is the diagonal part of **h**_0. **S**_*t* is a symmetric stress tensor that is chosen by LAMMPS so that the upper-triangular components of **P** equal the stress tensor specified by the user.

This equation only applies when the box dimensions are equal to those of the reference dimensions. If this is not the case, then the converged stress tensor will not equal that specified by the user. We can resolve this problem by periodically resetting the reference dimensions. The keyword *nreset_ref* controls how often this is done. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. A value of *nstep* means that every *nstep* minimization steps, the reference dimensions are set to those of the current simulation domain. Note that resetting the reference dimensions changes the objective function and gradients, which sometimes causes the minimization to fail. This can be resolved by changing the value of *nreset*, or simply continuing the minimization from a restart file.

IMPORTANT NOTE: As normally computed, pressure includes a kinetic- energy or temperature-dependent component; see the compute pressure command. However, atom velocities are ignored during a minimization,

and the applied pressure(s) specified with this command are assumed to only be the virial component of the pressure (the non-kinetic portion). Thus if atoms have a non-zero temperature and you print the usual thermodynamic pressure, it may not appear the system is converging to your specified pressure. The solution for this is to either (a) zero the velocities of all atoms before performing the minimization, or (b) make sure you are monitoring the pressure without its kinetic component. The latter can be done by outputting the pressure from the fix this command creates (see below) or a pressure fix you define yourself.

IMPORTANT NOTE: Because pressure is often a very sensitive function of volume, it can be difficult for the minimizer to equilibrate the system the desired pressure with high precision, particularly for solids. Some techniques that seem to help are (a) use the "min_modify line quadratic" option when minimizing with box relaxations, and (b) minimize several times in succession if need be, to drive the pressure closer to the target pressure. Also note that some systems (e.g. liquids) will not sustain a non-hydrostatic applied pressure, which means the minimizer will not converge.

This fix computes a temperature and pressure each timestep. The temperature is used to compute the kinetic contribution to the pressure, even though this is subsequently ignored by default. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp virial
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is the same as the fix group. Also note that the pressure compute does not include a kinetic component.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its temperature and pressure calculation, as described above. Note that as described above, if you assign a pressure compute to this fix that includes a kinetic energy component it will affect the minimization, most likely in an undesirable way.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a pressure compute defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the pressure-volume energy, plus the strain energy, if it exists.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is invoked during energy minimization, but not for the purpose of adding a contribution to the energy or forces being minimized. Instead it alters the simulation box geometry as described above.

Restrictions:

Only dimensions that are available can be adjusted by this fix. Non-periodic dimensions are not available. *z*, *xz*, and *yz*, are not available for 2D simulations. *xy*, *xz*, and *yz* are only available if the simulation domain is non-orthogonal. The [create_box](), [read data](), and [read_restart]() commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The *scaleyz yes* and *scalexz yes* keyword/value pairs can not be used for 2D simulations. *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the 2nd dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, *yz*, *xz*, and *xy*.

Related commands:

[fix npt](), [minimize]()

Default:

The keyword defaults are dilate = all, vmax = 0.0001, nreset = 0.

(Parrinello1981) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

fix check/timestep/gran command

Syntax:

```
fix ID group-ID check/timestep/gran nevery fraction_r fraction_h
```

- ID, group-ID are documented in [fix](#) command
- check/timestep/gran = style name of this fix command
- nevery = evaluate time-step size accuracy every this many time-steps
- fraction_r = warn if time-step size exceeds this fraction of the Rayleigh time
- fraction_h = warn if time-step size exceeds this fraction of the Hertz time

Examples:

```
fix ts_check all check/timestep/gran 1000 0.1 0.1
```

Description:

Periodically calculate estimations of the Rayleigh- and Hertz time dt_r and dt_h for a granular system every 'nevery' time-steps. The user can specify two quantities *fraction_r* and *fraction_h*. A warning message is printed if the time-step size as specified via the [timestep](#) command exceeds either of $dt_r * fraction_r$ or $dt_h * fraction_h$.

The former quantity is

$dt_r = PI*r*sqrt(rho/G)/(0.1631*nu+0.8766)$,

where rho is particle density, G is the shear modulus and nu is Poisson's ratio. The latter quantity is expressed by

$dt_h = 2.87*(m\_eff^2/(r\_eff*Y\_eff^2*v\_max))^{0.2}$.

The effective mass m_eff, the effective radius r_eff and the effective Young's modulus Y_eff are as defined in [pair gran](#). v_max is the maximum relative velocity, taking mesh movement into account. Please note that the Hertz criterion will also be used if you use a different granular pair style (e.g. Hooke).

Additionally, this command checks the ratio of skin to the distance that particles can travel relative to each other in one time-step. This value should be >1, otherwise some interactions may be missed or overlap energy may be generated artificially. This command will warn you if this is the case.

These criteria are checked every 'nevery' time-steps. Rayleigh time dt_r is calculated for each particle in the simulation, and the minimum value is taken for further calculations. Hertz time dt_h is estimated by testing a collision of each particle with itself using v_max as the assumed collision velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. This fix computes a 3-vector, for access by various [output commands](#). The vector consists of the time-step size expressed as fraction of the Rayleigh and Hertz time-step sizes and the ratio of skin to the distance particles can travel relative to each other in one time-step, respectively. No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands: none

Default: none

fix colvars command

Syntax:

```
fix ID group-ID colvars configfile keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- colvars = style name of this fix command
- configfile = the configuration file for the colvars module
- keyword = *input* or *output* or *seed* or *tstat*

```
  input arg = colvars.state file name or prefix or NULL (default: NULL)
    output arg = output filename prefix (default: out)
    seed arg = seed for random number generator (default: 1966)
    unwrap arg = yes or no
      use unwrapped coordinates in collective variables (default: yes)
    tstat arg = fix id of a thermostat or NULL (default: NULL)
```

Examples:

```
fix mtd all colvars peptide.colvars.inp seed 2122 input peptide.colvars.state output peptide
fix abf all colvars colvars.inp tstat 1
```

Description:

This fix interfaces LAMMPS to a "collective variables" or "colvars" module library which allows to calculate potentials of mean force (PMFs) for any set of colvars, using different sampling methods: currently implemented are the Adaptive Biasing Force (ABF) method, metadynamics, Steered Molecular Dynamics (SMD) and Umbrella Sampling (US) via a flexible harmonic restraint bias. The colvars library is hosted at http://colvars.github.io/

This documentation describes only the fix colvars command itself and LAMMPS specific parts of the code. The full documentation of the colvars library is available as [this supplementary PDF document](#)

A detailed discussion of the implementation of the portable collective variable library is in [(Fiorin)](#). Additional information can be found in [(Henin)](#).

There are some example scripts for using this package with LAMMPS in the examples/USER/colvars directory.

The only mandatory argument to the fix is the filename to the colvars input file that contains the input that is independent from the MD program in which the colvars library has been integrated.

The *group-ID* entry is ignored. The collective variable module will always apply to the entire system and there can only be one instance of the colvars fix at a time. The colvars fix will only communicate the minimum information necessary and the colvars library supports multiple, completely independent collective variables, so there is no restriction to functionaliry by limiting the number of colvars fixes.

The *input* keyword allows to specify a state file that would contain the restart information required in order to continue a calculation from a prerecorded state. Fix colvars records it state in [binary restart](#) files, so when using the [read_restart](#) command, this is usually not needed.

The *output* keyword allows to specify the output prefix. All output files generated will use this prefix followed by the ".colvars." and a word like "state" or "traj".

The *seed* keyword contains the seed for the random number generator that will be used in the colvars module.

The *unwrap* keyword controls whether wrapped or unwrapped coordinates are passed to the colvars library for calculation of the collective variables and the resulting forces. The default is *yes*, i.e. to use the image flags to reconstruct the absolute atom positions. Setting this to *no* will use the current local coordinates that are wrapped back into the simulation cell at each re-neighboring instead.

The *tstat* keyword can be either NULL or the label of a thermostating fix that thermostats all atoms in the fix colvars group. This will be used to provide the colvars module with the current thermostat target temperature.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the current status of the colvars module into binary restart files. This is in addition to the text mode status file that is written by the colvars module itself and the kind of information in both files is identical.

The fix_modify *energy* option is supported by this fix to add the energy change from the biasing force added by the fix to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

Restrictions:

This fix is part of the USER-COLVARS package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

There can only be one colvars fix active at a time. Since the interface communicates only the minimum amount of information and colvars module itself can handle an arbitrary number of collective variables, this is not a limitation of functionality.

Related commands:

fix smd

Default:

The default options are input = NULL, output = out, seed = 1966, unwrap yes, and tstat = NULL.

(Fiorin) Fiorin , Klein, Henin, Mol. Phys., DOI:10.1080/00268976.2013.813594

(Henin) Henin, Fiorin, Chipot, Klein, J. Chem. Theory Comput., 6, 35-47 (2010)

fix deform command

Syntax:

```
fix ID group-ID deform N parameter args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- deform = style name of this fix command
- N = perform box deformation every this many timesteps
- one or more parameter/arg pairs may be appended

```
parameter = x or y or z or xy or xz or yz
  x, y, z args = style value(s)
    style = final or delta or scale or vel or erate or trate or volume or wiggle or variab
      final values = lo hi
        lo hi = box boundaries at end of run (distance units)
      delta values = dlo dhi
        dlo dhi = change in box boundaries at end of run (distance units)
      scale values = factor
        factor = multiplicative factor for change in box length at end of run
      vel value = V
        V = change box length at this velocity (distance/time units),
            effectively an engineering strain rate
      erate value = R
        R = engineering strain rate (1/time units)
      trate value = R
        R = true strain rate (1/time units)
      volume value = none = adjust this dim to preserve volume of system
      wiggle values = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
      variable values = v_name1 v_name2
        v_name1 = variable with name1 for box length change as function of time
        v_name2 = variable with name2 for change rate as function of time
  xy, xz, yz args = style value
    style = final or delta or vel or erate or trate or wiggle
      final value = tilt
        tilt = tilt factor at end of run (distance units)
      delta value = dtilt
        dtilt = change in tilt factor at end of run (distance units)
      vel value = V
        V = change tilt factor at this velocity (distance/time units),
            effectively an engineering shear strain rate
      erate value = R
        R = engineering shear strain rate (1/time units)
      trate value = R
        R = true shear strain rate (1/time units)
      wiggle values = A Tp
        A = amplitude of oscillation (distance units)
        Tp = period of oscillation (time units)
      variable values = v_name1 v_name2
        v_name1 = variable with name1 for tilt change as function of time
        v_name2 = variable with name2 for change rate as function of time
```

- zero or more keyword/value pairs may be appended
- keyword = *remap* or *flip* or *units*

```
remap value = x or v or none
    x = remap coords of atoms in group into deforming box
    v = remap velocities of all atoms when they cross periodic boundaries
    none = no remapping of x or v
  flip value = yes or no
```

```
        allow or disallow box flips when it becomes highly skewed
    units value = lattice or box
      lattice = distances are defined in lattice units
      box = distances are defined in simulation box units
```

Examples:

```
fix 1 all deform 1 x final 0.0 9.0 z final 0.0 5.0 units box
fix 1 all deform 1 x trate 0.1 y volume z volume
fix 1 all deform 1 xy erate 0.001 remap v
fix 1 all deform 10 y delta -0.5 0.5 xz vel 1.0
```

Description:

Change the volume and/or shape of the simulation box during a dynamics run. Orthogonal simulation boxes have 3 adjustable parameters (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable parameters (x,y,z,xy,xz,yz). Any or all of them can be adjusted independently and simultaneously by this command. This fix can be used to perform non-equilibrium MD (NEMD) simulations of a continuously strained system. See the fix nvt/sllod and compute temp/deform commands for more details.

For the *x*, *y*, *z* parameters, the associated dimension cannot be shrink-wrapped. For the *xy*, *yz*, *xz* parameters, the associated 2nd dimension cannot be shrink-wrapped. Dimensions not varied by this command can be periodic or non-periodic. Dimensions corresponding to unspecified parameters can also be controlled by a fix npt or fix nph command.

The size and shape of the simulation box at the beginning of the simulation run were either specified by the create_box or read_data or read_restart command used to setup the simulation initially if it is the first run, or they are the values from the end of the previous run. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors. If fix deform changes the xy,xz,yz tilt factors, then the simulation box must be triclinic, even if its initial tilt factors are 0.0.

As described below, the desired simulation box size and shape at the end of the run are determined by the parameters of the fix deform command. Every Nth timestep during the run, the simulation box is expanded, contracted, or tilted to ramped values between the initial and final values.

For the *x*, *y*, and *z* parameters, this is the meaning of their styles and values.

The *final*, *delta*, *scale*, *vel*, and *erate* styles all change the specified dimension of the box via "constant displacement" which is effectively a "constant engineering strain rate". This means the box dimension changes linearly with time from its initial to final value.

For style *final*, the final lo and hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, plus or minus changes in the lo/hi box boundaries of a dimension are specified. The values can be in lattice or box distance units. See the discussion of the units keyword below.

For style *scale*, a multiplicative factor to apply to the box length of a dimension is specified. For example, if the initial box length is 10, and the factor is 1.1, then the final box length will be 11. A factor less than 1.0 means compression.

For style *vel*, a velocity at which the box length changes is specified in units of distance/time. This is effectively a "constant engineering strain rate", where rate = V/L0 and L0 is the initial box length. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial box length is 100 Angstroms, and V is 10 Angstroms/psec, then after 10 psec, the box length will

fix deform command

have doubled. After 20 psec, it will have tripled.

The *erate* style changes a dimension of the the box at a "constant engineering strain rate". The units of the specified strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length. The box length L as a function of time will change as

```
L(t) = L0 (1 + erate*dt)
```

where dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the box length will increase by 10% of its original length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. R = -0.01 means the box length will shrink by 1% of its original length every picosecond. Note that for an "engineering" rate the change is based on the original box length, so running with R = 1 for 10 picoseconds expands the box length by a factor of 11 (strain of 10), which is different that what the *trate* style would induce.

The *trate* style changes a dimension of the box at a "constant true strain rate". Note that this is not an "engineering strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the box dimension changes non-linearly with time from its initial to final value. The units of the specified strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Tensile strain is unitless and is defined as delta/L0, where L0 is the original box length and delta is the change relative to the original length.

The box length L as a function of time will change as

```
L(t) = L0 exp(trate*dt)
```

where dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the box length will increase by 10% of its current (not original) length every picosecond. I.e. strain after 1 psec = 0.1, strain after 2 psec = 0.21, etc. R = ln(2) or ln(3) means the box length will double or triple every picosecond. R = ln(0.99) means the box length will shrink by 1% of its current length every picosecond. Note that for a "true" rate the change is continuous and based on the current length, so running with R = ln(2) for 10 picoseconds does not expand the box length by a factor of 11 as it would with *erate*, but by a factor of 1024 since the box length will double every picosecond.

Note that to change the volume (or cross-sectional area) of the simulation box at a constant rate, you can change multiple dimensions via *erate* or *trate*. E.g. to double the box volume in a picosecond picosecond, you could set "x erate M", "y erate M", "z erate M", with M = pow(2,1/3) - 1 = 0.26, since if each box dimension grows by 26%, the box volume doubles. Or you could set "x trate M", "y trate M", "z trate M", with M = ln(1.26) = 0.231, and the box volume would double every picosecond.

The *volume* style changes the specified dimension in such a way that the box volume remains constant while other box dimensions are changed explicitly via the styles discussed above. For example, "x scale 1.1 y scale 1.1 z volume" will shrink the z box length as the x,y box lengths increase, to keep the volume constant (product of x,y,z lengths). If "x scale 1.1 z volume" is specified and parameter *y* is unspecified, then the z box length will shrink as x increases to keep the product of x,z lengths constant. If "x scale 1.1 y volume z volume" is specified, then both the y,z box lengths will shrink as x increases to keep the volume constant (product of x,y,z lengths). In this case, the y,z box lengths shrink so as to keep their relative aspect ratio constant.

For solids or liquids, note that when one dimension of the box is expanded via fix deform (i.e. tensile strain), it may be physically undesirable to hold the other 2 box lengths constant (unspecified by fix deform) since that implies a density change. Using the *volume* style for those 2 dimensions to keep the box volume constant may make more physical sense, but may also not be correct for materials and potentials whose Poisson ratio is

not 0.5. An alternative is to use fix npt aniso with zero applied pressure on those 2 dimensions, so that they respond to the tensile strain dynamically.

The *wiggle* style oscillates the specified box length dimension sinusoidally with the specified amplitude and period. I.e. the box length L as a function of time is given by

```
L(t) = L0 + A sin(2*pi t/Tp)
```

where L0 is its initial length. If the amplitude A is a positive number the box initially expands, then contracts, etc. If A is negative then the box initially contracts, then expands, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

The *variable* style changes the specified box length dimension by evaluating a variable, which presumably is a function of time. The variable with *name1* must be an equal-style variable and should calculate a change in box length in units of distance. Note that this distance is in box units, not lattice units; see the discussion of the *units* keyword below. The formula associated with variable *name1* can reference the current timestep. Note that it should return the "change" in box length, not the absolute box length. This means it should evaluate to 0.0 when invoked on the initial timestep of the run following the definition of fix deform. It should evaluate to a value > 0.0 to dilate the box at future times, or a value < 0.0 to compress the box.

The variable *name2* must also be an equal-style variable and should calculate the rate of box length change, in units of distance/time, i.e. the time-derivative of the *name1* variable. This quantity is used internally by LAMMPS to reset atom velocities when they cross periodic boundaries. It is computed internally for the other styles, but you must provide it when using an arbitrary variable.

Here is an example of using the *variable* style to perform the same box deformation as the *wiggle* style formula listed above, where we assume that the current timestep = 0.

```
variable A equal 5.0
variable Tp equal 10.0
variable displace equal "v_A * sin(2*PI * step*dt/v_Tp)"
variable rate equal "2*PI*v_A/v_Tp * cos(2*PI * step*dt/v_Tp)"
fix 2 all deform 1 x variable v_displace v_rate remap v
```

For the *scale*, *vel*, *erate*, *trate*, *volume*, *wiggle*, and *variable* styles, the box length is expanded or compressed around its mid point.

For the *xy*, *xz*, and *yz* parameters, this is the meaning of their styles and values. Note that changing the tilt factors of a triclinic box does not change its volume.

The *final*, *delta*, *vel*, and *erate* styles all change the shear strain at a "constant engineering shear strain rate". This means the tilt factor changes linearly with time from its initial to final value.

For style *final*, the final tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *delta*, a plus or minus change in the tilt factor is specified. The value can be in lattice or box distance units. See the discussion of the units keyword below.

For style *vel*, a velocity at which the tilt factor changes is specified in units of distance/time. This is effectively an "engineering shear strain rate", where rate = V/L0 and L0 is the initial box length perpendicular to the direction of shear. The distance can be in lattice or box distance units. See the discussion of the units keyword below. For example, if the initial tilt factor is 5 Angstroms, and the V is 10 Angstroms/psec, then after 1 psec, the tilt factor will be 15 Angstroms. After 2 psec, it will be 25 Angstroms.

The *erate* style changes a tilt factor at a "constant engineering shear strain rate". The units of the specified shear strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 + L0*erate*dt
```

where T0 is the initial tilt factor, L0 is the original length of the box perpendicular to the shear direction (e.g. y box length for xy deformation), and dt is the elapsed time (in time units). Thus if *erate* R is specified as 0.1 and time units are picoseconds, this means the shear strain will increase by 0.1 every picosecond. I.e. if the xy shear strain was initially 0.0, then strain after 1 psec = 0.1, strain after 2 psec = 0.2, etc. Thus the tilt factor would be 0.0 at time 0, 0.1*ybox at 1 psec, 0.2*ybox at 2 psec, etc, where ybox is the original y box length. R = 1 or 2 means the tilt factor will increase by 1 or 2 every picosecond. R = -0.01 means a decrease in shear strain by 0.01 every picosecond.

The *trate* style changes a tilt factor at a "constant true shear strain rate". Note that this is not an "engineering shear strain rate", as the other styles are. Rather, for a "true" rate, the rate of change is constant, which means the tilt factor changes non-linearly with time from its initial to final value. The units of the specified shear strain rate are 1/time. See the [units](#) command for the time units associated with different choices of simulation units, e.g. picoseconds for "metal" units). Shear strain is unitless and is defined as offset/length, where length is the box length perpendicular to the shear direction (e.g. y box length for xy deformation) and offset is the displacement distance in the shear direction (e.g. x direction for xy deformation) from the unstrained orientation.

The tilt factor T as a function of time will change as

```
T(t) = T0 exp(trate*dt)
```

where T0 is the initial tilt factor and dt is the elapsed time (in time units). Thus if *trate* R is specified as ln(1.1) and time units are picoseconds, this means the shear strain or tilt factor will increase by 10% every picosecond. I.e. if the xy shear strain was initially 0.1, then strain after 1 psec = 0.11, strain after 2 psec = 0.121, etc. R = ln(2) or ln(3) means the tilt factor will double or triple every picosecond. R = ln(0.99) means the tilt factor will shrink by 1% every picosecond. Note that the change is continuous, so running with R = ln(2) for 10 picoseconds does not change the tilt factor by a factor of 10, but by a factor of 1024 since it doubles every picosecond. Note that the initial tilt factor must be non-zero to use the *trate* option.

Note that shear strain is defined as the tilt factor divided by the perpendicular box length. The *erate* and *trate* styles control the tilt factor, but assume the perpendicular box length remains constant. If this is not the case (e.g. it changes due to another fix deform parameter), then this effect on the shear strain is ignored.

The *wiggle* style oscillates the specified tilt factor sinusoidally with the specified amplitude and period. I.e. the tilt factor T as a function of time is given by

```
T(t) = T0 + A sin(2*pi t/Tp)
```

where T0 is its initial value. If the amplitude A is a positive number the tilt factor initially becomes more positive, then more negative, etc. If A is negative then the tilt factor initially becomes more negative, then more positive, etc. The amplitude can be in lattice or box distance units. See the discussion of the units keyword below.

The *variable* style changes the specified tilt factor by evaluating a variable, which presumably is a function of time. The variable with *name1* must be an [equal-style variable](#) and should calculate a change in tilt in units of distance. Note that this distance is in box units, not lattice units; see the discussion of the *units* keyword below. The formula associated with variable *name1* can reference the current timestep. Note that it should return the "change" in tilt factor, not the absolute tilt factor. This means it should evaluate to 0.0 when invoked on the initial timestep of the run following the definition of fix deform.

The variable *name2* must also be an [equal-style variable](#) and should calculate the rate of tilt change, in units of distance/time, i.e. the time-derivative of the *name1* variable. This quantity is used internally by LAMMPS to reset atom velocities when they cross periodic boundaries. It is computed internally for the other styles, but you must provide it when using an arbitrary variable.

Here is an example of using the *variable* style to perform the same box deformation as the *wiggle* style formula listed above, where we assume that the current timestep = 0.

```
variable A equal 5.0
variable Tp equal 10.0
variable displace equal "v_A * sin(2*PI * step*dt/v_Tp)"
variable rate equal "2*PI*v_A/v_Tp * cos(2*PI * step*dt/v_Tp)"
fix 2 all deform 1 xy variable v_displace v_rate remap v
```

All of the tilt styles change the xy, xz, yz tilt factors during a simulation. In LAMMPS, tilt factors (xy,xz,yz) for triclinic boxes are normally bounded by half the distance of the parallel box length. See the discussion of the *flip* keyword below, to allow this bound to be exceeded, if desired.

For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all equivalent.

To obey this constraint and allow for large shear deformations to be applied via the *xy*, *xz*, or *yz* parameters, the following algorithm is used. If *prd* is the associated parallel box length (10 in the example above), then if the tilt factor exceeds the accepted range of -5 to 5 during the simulation, then the box is flipped to the other limit (an equivalent box) and the simulation continues. Thus for this example, if the initial xy tilt factor was 0.0 and "xy final 100.0" was specified, then during the simulation the xy tilt factor would increase from 0.0 to 5.0, the box would be flipped so that the tilt factor becomes -5.0, the tilt factor would increase from -5.0 to 5.0, the box would be flipped again, etc. The flip occurs 10 times and the final tilt factor at the end of the simulation would be 0.0. During each flip event, atoms are remapped into the new box in the appropriate manner.

The one exception to this rule is if the 1st dimension in the tilt factor (x for xy) is non-periodic. In that case, the limits on the tilt factor are not enforced, since flipping the box in that dimension does not change the atom positions due to non-periodicity. In this mode, if you tilt the system to extreme angles, the simulation will simply become inefficient due to the highly skewed simulation box.

Each time the box size or shape is changed, the *remap* keyword determines whether atom positions are remapped to the new box. If *remap* is set to *x* (the default), atoms in the fix group are remapped; otherwise they are not. Note that their velocities are not changed, just their positions are altered. If *remap* is set to *v*, then any atom in the fix group that crosses a periodic boundary will have a delta added to its velocity equal to the difference in velocities between the lo and hi boundaries. Note that this velocity difference can include tilt components, e.g. a delta in the x velocity when an atom crosses the y periodic boundary. If *remap* is set to *none*, then neither of these remappings take place.

Conceptually, setting *remap* to *x* forces the atoms to deform via an affine transformation that exactly matches the box deformation. This setting is typically appropriate for solids. Note that though the atoms are effectively

"moving" with the box over time, it is not due to their having a velocity that tracks the box change, but only due to the remapping. By contrast, setting *remap* to *v* is typically appropriate for fluids, where you want the atoms to respond to the change in box size/shape on their own and acquire a velocity that matches the box change, so that their motion will naturally track the box without explicit remapping of their coordinates.

IMPORTANT NOTE: When non-equilibrium MD (NEMD) simulations are performed using this fix, the option "remap v" should normally be used. This is because fix nvt/sllod adjusts the atom positions and velocities to induce a velocity profile that matches the changing box size/shape. Thus atom coordinates should NOT be remapped by fix deform, but velocities SHOULD be when atoms cross periodic boundaries, since that is consistent with maintaining the velocity profile already created by fix nvt/sllod. LAMMPS will warn you if the *remap* setting is not consistent with fix nvt/sllod.

IMPORTANT NOTE: For non-equilibrium MD (NEMD) simulations using "remap v" it is usually desirable that the fluid (or flowing material, e.g. granular particles) stream with a velocity profile consistent with the deforming box. As mentioned above, using a thermostat such as fix nvt/sllod or fix lavgevin (with a bias provided by compute temp/deform), will typically accomplish that. If you do not use a thermostat, then there is no driving force pushing the atoms to flow in a manner consistent with the deforming box. E.g. for a shearing system the box deformation velocity may vary from 0 at the bottom to 10 at the top of the box. But the stream velocity profile of the atoms may vary from -5 at the bottom to +5 at the top. You can monitor these effects using the fix ave/spatial, compute temp/deform, and compute temp/profile commands. One way to induce atoms to stream consistent with the box deformation is to give them an initial velocity profile, via the velocity ramp command, that matches the box deformation rate. This also typically helps the system come to equilibrium more quickly, even if a thermostat is used.

IMPORTANT NOTE: If a fix rigid is defined for rigid bodies, and *remap* is set to *x*, then the center-of-mass coordinates of rigid bodies will be remapped to the changing simulation box. This will be done regardless of whether atoms in the rigid bodies are in the fix deform group or not. The velocity of the centers of mass are not remapped even if *remap* is set to *v*, since fix nvt/sllod does not currently do anything special for rigid particles. If you wish to perform a NEMD simulation of rigid particles, you can either thermostat them independently or include a background fluid and thermostat the fluid via fix nvt/sllod.

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed above. If the *flip* value is set to *yes*, the bound is enforced by flipping the box when it is exceeded. If the *flip* value is set to *no*, the tilt will continue to change without flipping. Note that if you apply large deformations, this means the box shape can tilt dramatically LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *units* keyword determines the meaning of the distance units used to define various arguments. A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Note that the units choice also affects the *vel* style parameters since it is defined in terms of distance/time. Also note that the units keyword does not affect the *variable* style. You should use the *xlat*, *ylat*, *zlat* keywords of the thermo_style command if you want to include lattice spacings in a variable formula.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands.

This fix can perform deformation over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

You cannot apply x, y, or z deformations to a dimension that is shrink-wrapped via the [boundary](#) comamnd.

You cannot apply xy, yz, or xz deformations to a 2nd dimension (y in xy) that is shrink-wrapped via the [boundary](#) comamnd.

Related commands:

[change_box](#)

Default:

The option defaults are remap = x, flip = yes, and units = lattice.

fix deposit command

Syntax:

```
fix ID group-ID deposit N type M seed keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- deposit = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- M = insert a single particle every M steps
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *id* or *global* or *local* or *near* or *attempt* or *rate* or *vx* or *vy* or *vz* or *units*

```
  region value = region-ID
      region-ID = ID of region to use as insertion volume
    id value = max or next
      max = atom ID for new atom is max ID of all current atoms plus one
      next = atom ID for new atom increments by one for every deposition
    global values = lo hi
      lo,hi = put new particle a distance lo-hi above all other particles (distance units)
    local values = lo hi delta
      lo,hi = put new particle a distance lo-hi above any nearby particle beneath it (distance
      delta = lateral distance within which a neighbor is considered "nearby" (distance unit
    near value = R
      R = only insert particle if further than R from existing particles (distance units)
    attempt value = Q
      Q = attempt a single insertion up to Q times
    rate value = V
      V = z velocity (y in 2d) at which insertion volume moves (velocity units)
    vx values = vxlo vxhi
      vxlo,vxhi = range of x velocities for inserted particle (velocity units)
    vy values = vylo vyhi
      vylo,vyhi = range of y velocities for inserted particle (velocity units)
    vz values = vzlo vzhi
      vzlo,vzhi = range of z velocities for inserted particle (velocity units)
    target values = tx ty tz
      tx,ty,tz = location of sputtering target (distance units)
    units value = lattice or box
      lattice = the geometry is defined in lattice units
      box = the geometry is defined in simulation box units
```

Examples:

```
fix 3 all deposit 1000 2 100 29494 region myblock local 1.0 1.0 1.0 units box
fix 2 newatoms deposit 10000 1 500 12345 region disk near 2.0 vz -1.0 -0.8

fix 4 sputter deposit 1000 2 500 12235 region sphere vz -1.0 -1.0 target 5.0 5.0 0.0 units lattic
```

Description:

Insert a single particle into the simulation domain every M timesteps until N particles have been inserted. This is useful for simulating the deposition of particles onto a surface.

Inserted particles have the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix deposit command (which can also be "all").

If you are computing temperature values which include inserted particles, you will want to use the [compute_modify](#) dynamic option, which insures the current number of atoms is used as a normalizing factor each time temperature is computed.

Care must be taken that inserted particles are not too near existing particles, using the options described below. When inserting particles above a surface in a non-periodic box (see the [boundary](#) command), the possibility of a particle escaping the surface and flying upward should be considered, since the particle may be lost or the box size may grow infinitely large. A [fix wall/reflect](#) command can be used to prevent this behavior. Note that if a shrink-wrap boundary is used, it is OK to insert the new particle outside the box, however the box will immediately be expanded to include the new particle. When simulating a sputtering experiment it is probably more realistic to ignore those atoms using the [thermo_modify](#) command with the *lost ignore* option and a fixed [boundary](#).

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a [region](#) command. It must be defined with side = *in*.

Each timestep a particle is to be inserted, its coordinates are chosen as follows. A random position within the insertion volume is generated. If neither the *global* or *local* keyword is used, that is the trial position. If the *global* keyword is used, the random x,y values are used, but the z position of the new particle is set above the highest current atom in the simulation by a distance randomly chosen between lo/hi. (For a 2d simulation, this is done for the y position.) If the *local* keyword is used, the z position is set a distance between lo/hi above the highest current atom in the simulation that is "nearby" the chosen x,y position. In this context, "nearby" means the lateral distance (in x,y) between the new and old particles is less than the delta parameter.

Once a trial x,y,z location has been computed, the insertion is only performed if no current particle in the simulation is within a distance R of the new particle. If this test fails, a new random position within the insertion volume is chosen and another trial is made. Up to Q attempts are made. If an atom is not successfully deposited, LAMMPS prints a warning message.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables particles to be inserted from a successively higher height over time. Note that this parameter is ignored if the *global* or *local* keywords are used, since those options choose a z-coordinate for insertion independently.

The vx, vy, and vz components of velocity for the inserted particle are set using the values specified for the *vx*, *vy*, and *vz* keywords. Note that normally, new particles should be a assigned a negative vertical velocity so that they move towards the surface.

In case the *target* option is used, the velocity vector of the inserted particle will be changed in a way so that it would pass through the specified coordinate. This allows convenient simulation of a sputtering process.

The *id* keyword determines how an atom ID is assigned to newly deposited atoms. For the *max* setting, the IDs of all current atoms are checked and the new ID is the current maximum value plus one. This means that if atoms have left the system, the new ID may reflect this fact. For the *next* setting, the maximum ID of all atoms is stored at the time the fix is defined. Each time a deposited atom is added, this value is incremented by one and assigned to the new atom. Thus atom IDs for deposited atoms will be consecutive even if atoms leave the system over time.

The *units* keyword determines the meaning of the distance units used for the other deposition parameters. A *box* value selects standard distance units as defined by the [units](#) command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing. Note that the units choice affects all the keyword values that have units of distance or velocity.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the deposition to [binary restart files](#). This includes information about how many atoms have been depositied, the random number generator seed, the next timestep for deposition, etc. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

The specified insertion region cannot be a "dynamic" region, as defined by the [region](#) command.

Related commands:

[fix_pour](#), [region](#)

Default:

The option defaults are id = max, delta = 0.0, near = 0.0, attempt = 10, rate = 0.0, vx = 0.0 0.0, vy = 0.0 0.0, vz = 0.0 0.0, and units = lattice.

fix drag command

Syntax:

```
fix ID group-ID drag x y z fmag delta
```

- ID, group-ID are documented in [fix](#) command
- drag = style name of this fix command
- x,y,z = coord to drag atoms towards
- fmag = magnitude of force to apply to each atom (force units)
- delta = cutoff distance inside of which force is not applied (distance units)

Examples:

```
fix center small-molecule drag 0.0 10.0 0.0 5.0 2.0
```

Description:

Apply a force to each atom in a group to drag it towards the point (x,y,z). The magnitude of the force is specified by fmag. If an atom is closer than a distance delta to the point, then the force is not applied.

Any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

This command can be used to steer one or more atoms to a new location in the simulation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various [output commands](#). This is the total force on the group of atoms by the drag force. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix spring](#), [fix spring/self](#), [fix spring/rg](#), [fix smd](#)

Default: none

fix dt/reset command

Syntax:

```
fix ID group-ID dt/reset N Tmin Tmax Xmax keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- dt/reset = style name of this fix command
- N = recompute dt every N timesteps
- Tmin = minimum dt allowed which can be NULL (time units)
- Tmax = maximum dt allowed which can be NULL (time units)
- Xmax = maximum distance for an atom to move in one timestep (distance units)
- zero or more keyword/value pairs may be appended
- keyword = *units*

```
units value = lattice or box
    lattice = Xmax is defined in lattice units
    box = Xmax is defined in simulation box units
```

Examples:

```
fix 5 all dt/reset 10 1.0e-5 0.01 0.1
fix 5 all dt/reset 10 0.01 2.0 0.2 units box
```

Description:

Reset the timestep size every N steps during a run, so that no atom moves further than Xmax, based on current atom velocities and forces. This can be useful when starting from a configuration with overlapping atoms, where forces will be large. Or it can be useful when running an impact simulation where one or more high-energy atoms collide with a solid, causing a damage cascade.

This fix overrides the timestep size setting made by the [timestep](#) command. The new timestep size *dt* is computed in the following manner.

For each atom, the timestep is computed that would cause it to displace *Xmax* on the next integration step, as a function of its current velocity and force. Since performing this calculation exactly would require the solution to a quartic equation, a cheaper estimate is generated. The estimate is conservative in that the atom's displacement is guaranteed not to exceed *Xmax*, though it may be smaller.

Given this putative timestep for each atom, the minimum timestep value across all atoms is computed. Then the *Tmin* and *Tmax* bounds are applied, if specified. If one (or both) is specified as NULL, it is not applied.

When the [run style](#) is *respa*, this fix resets the outer loop (largest) timestep, which is the same timestep that the [timestep](#) command sets.

Note that the cumulative simulation time (in time units), which accounts for changes in the timestep size as a simulation proceeds, can be accessed by the [thermo_style time](#) keyword.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar stores the last timestep on which the timestep was reset to a new value.

The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

timestep

Default:

The option defaults is units = lattice.

fix efield command

Syntax:

```
fix ID group-ID efield ex ey ez keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- efield = style name of this fix command
- ex,ey,ez = E-field component values (electric field units)
- any of ex,ey,ez can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region* or *energy*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
    energy value = v_name
      v_name = variable with name that calculates the potential energy of each atom in the ad
```

Examples:

```
fix kick external-field efield 1.0 0.0 0.0
fix kick external-field efield 0.0 0.0 v_oscillate
```

Description:

Add a force F = qE to each charged atom in the group due to an external electric field being applied to the system. If the system contains point-dipoles, also add a torque on the dipoles due to the external electric field.

For charges, any of the 3 quantities defining the E-field components can be specified as an equal-style or atom-style [variable](#), namely *ex*, *ey*, *ez*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the E-field component.

For point-dipoles, equal-style variables can be used, but atom-style variables are not currently supported, since they imply a spatial gradient in the electric field which means additional terms with gradients of the field are required for the force and torque on dipoles.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent E-field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent E-field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

Adding a force or torque to atoms implies a change in their potential energy as they move or rotate due to the applied E-field.

For dynamics via the "run" command, this energy can be optionally added to the system's potential energy for thermodynamic output (see below). For energy minimization via the "minimize" command, this energy must

be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

The *energy* keyword is not allowed if the added field is a constant vector (ex,ey,ez), with all components defined as numeric constants and not as variables. This is because LAMMPS can compute the energy for each charged particle directly as E = -x dot qE = -q (x*ex + y*ey + z*ez), so that -Grad(E) = F. Similarly for point-dipole particles the energy can be computed as E = -mu dot E = -(mux*ex + muy*ey + muz*ez).

The *energy* keyword is optional if the added force is defined with one or more variables, and if you are performing dynamics via the run command. If the keyword is not used, LAMMPS will set the energy to 0.0, which is typically fine for dynamics.

The *energy* keyword is required if the added force is defined with one or more variables, and you are performing energy minimization via the "minimize" command for charged particles. It is not required for point-dipoles, but a warning is issued since the minimizer in LAMMPS does not rotate dipoles, so you should not expect to be able to minimize the orientation of dipoles in an applied electric field.

The *energy* keyword specifies the name of an atom-style variable which is used to compute the energy of each atom as function of its position. Like variables used for *ex*, *ey*, *ez*, the energy variable is specified as v_name, where name is the variable name.

Note that when the *energy* keyword is used during an energy minimization, you must insure that the formula defined for the atom-style variable is consistent with the force variable formulas, i.e. that -Grad(E) = F. For example, if the force due to the electric field were a spring-like F = kx, then the energy formula should be E = -0.5kx^2. If you don't do this correctly, the minimization will not converge properly.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential "energy" inferred by the added force due to the electric field to the system's potential energy as part of thermodynamic output. This is a fictitious quantity but is needed so that the minimize command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force due to the electric field.

This fix computes a global scalar and a global 3-vector of forces, which can be accessed by various output commands. The scalar is the potential energy discussed above. The vector is the total force added to the group of atoms. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. You should not specify force components with a variable that has time-dependence for use with a minimizer, since the minimizer increments the timestep as the iteration count during the minimization.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix addforce

Default: none

fix enforce2d command

fix enforce2d/cuda command

Syntax:

```
fix ID group-ID enforce2d
```

- ID, group-ID are documented in fix command
- enforce2d = style name of this fix command

Examples:

```
fix 5 all enforce2d
```

Description:

Zero out the z-dimension velocity and force on each atom in the group. This is useful when running a 2d simulation to insure that atoms do not move from their initial z coordinate.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

Restrictions: none

Related commands: none

Default: none

fix evaporate command

Syntax:

```
fix ID group-ID evaporate N M region-ID seed
```

- ID, group-ID are documented in [fix](#) command
- evaporate = style name of this fix command
- N = delete atoms every this many timesteps
- M = number of atoms to delete each time
- region-ID = ID of region within which to perform deletions
- seed = random number seed to use for choosing atoms to delete
- zero or more keyword/value pairs may be appended

```
  keyword = molecule
    molecule value = no or yes
```

Examples:

```
fix 1 solvent evaporate 1000 10 surface 49892
fix 1 solvent evaporate 1000 10 surface 38277 molecule yes
```

Description:

Remove M atoms from the simulation every N steps. This can be used, for example, to model evaporation of solvent particles or moleclues (i.e. drying) of a system. Every N steps, the number of atoms in the fix group and within the specifed region are counted. M of these are chosen at random and deleted. If there are less than M eligible particles, then all of them are deleted.

If the setting for the *molecule* keyword is *no*, then only single atoms are deleted. In this case, you should insure you do not delete only a portion of a molecule (only some of its atoms), or LAMMPS will soon generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for deletion have a non-zero molecule ID, but does not check for this at the time of deletion.

If the setting for the *molecule* keyword is *yes*, then when an atom is chosen for deletion, the entire molecule it is part of is deleted. The count of deleted atoms is incremented by the number of atoms in the molecule, which may make it exceed *M*. If the molecule ID of the chosen atom is 0, then it is assumed to not be part of a molecule, and just the single atom is deleted.

As an example, if you wish to delete 10 water molecules every *N* steps, you should set *M* to 30. If only the water's oxygen atoms were in the fix group, then two hydrogen atoms would be deleted when an oxygen atom is selected for deletion, whether the hydrogens are inside the evaporation region or not.

Note that neighbor lists are re-built on timesteps that atoms are removed. Thus you should not remove atoms too frequently or you will incur overhead due to the cost of building neighbor lists.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global scalar, which can be accessed by various [output commands](#). The scalar is the cummulative number of deleted atoms. The scalar value calculated by this fix is "intensive".

fix enforce2d/cuda command   356

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix deposit

Default:

The option defaults are molecule = no.

fix external command

Syntax:

```
fix ID group-ID external mode args
```

- ID, group-ID are documented in [fix](#) command
- external = style name of this fix command
- mode = *pf/callback* or *pf/array*

```
  pf/callback args = Ncall Napply
      Ncall = make callback every Ncall steps
      Napply = apply callback forces every Napply steps
    pf/array args = Napply
      Napply = apply array forces every Napply steps
```

Examples:

```
fix 1 all external pf/callback 1 1
fix 1 all external pf/callback 100 1
fix 1 all external pf/array 10
```

Description:

This fix allows external programs that are running LAMMPS through its [library interface](#) to modify certain LAMMPS properties on specific timesteps, similar to the way other fixes do. The external driver can be a [C/C++ or Fortran program](#) or a [Python script](#).

If mode is *pf/callback* then the fix will make a callback every *Ncall* timesteps or minimization iterations to the external program. The external program computes forces on atoms by setting values in an array owned by the fix. The fix then adds these forces to each atom in the group, once every *Napply* steps, similar to the way the [fix addforce](#) command works. Note that if *Ncall* > *Napply*, the force values produced by one callback will persist, and be used multiple times to update atom forces.

The callback function "foo" is invoked by the fix as:

```
foo(void *ptr, bigint timestep, int nlocal, int *ids, double **x, double **fexternal);
```

The arguments are as follows:

- ptr = pointer provided by and simply passed back to external driver
- timestep = current LAMMPS timestep
- nlocal = # of atoms on this processor
- ids = list of atom IDs on this processor
- x = coordinates of atoms on this processor
- fexternal = forces to add to atoms on this processor

Note that timestep is a "bigint" which is defined in src/lmptype.h, typically as a 64-bit integer.

Fexternal are the forces returned by the driver program.

The fix has a set_callback() method which the external driver can call to pass a pointer to its foo() function. See the couple/lammps_quest/lmpqst.cpp file in the LAMMPS distribution for an example of how this is done. This sample application performs classical MD using quantum forces computed by a density functional

code [Quest](#).

If mode is *pf/array* then the fix simply stores force values in an array. The fix adds these forces to each atom in the group, once every *Napply* steps, similar to the way the [fix addforce](#) command works.

The name of the public force array provided by the FixExternal class is

```
double **fexternal;
```

It is allocated by the FixExternal class as an (N,3) array where N is the number of atoms owned by a processor. The 3 corresponds to the fx, fy, fz components of force.

It is up to the external program to set the values in this array to the desired quantities, as often as desired. For example, the driver program might perform an MD run in stages of 1000 timesteps each. In between calls to the LAMMPS [run](#) command, it could retrieve atom coordinates from LAMMPS, compute forces, set values in fexternal, etc.

To use this fix during energy minimization, the energy corresponding to the added forces must also be set so as to be consistent with the added forces. Otherwise the minimization will not converge correctly.

This can be done from the external driver by calling this public method of the FixExternal class:

```
void set_energy(double eng);
```

where eng is the potential energy. Eng is an extensive quantity, meaning it should be the sum over per-atom energies of all affected atoms. It should also be provided in [energy units](#) consistent with the simulation. See the details below for how to insure this energy setting is used appropriately in a minimization.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the potential "energy" set by the external driver to the system's potential energy as part of [thermodynamic output](#). This is a fictitious quantity but is needed so that the [minimize](#) command can include the forces added by this fix in a consistent manner. I.e. there is a decrease in potential energy when atoms move in the direction of the added force.

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the potential energy discussed above. The scalar stored by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the [fix_modify](#) *energy* option for this fix.

Restrictions: none

Related commands: none

Default: none

---

fix freeze command

fix freeze/cuda command

Syntax:

```
fix ID group-ID freeze
```

- ID, group-ID are documented in [fix](#) command
- freeze = style name of this fix command

Examples:

```
fix 2 bottom freeze
```

Description:

Zero out the force and torque on a granular particle. This is useful for preventing certain particles from moving in a simulation. The [granular pair styles](#) also detect if this fix has been defined and compute interactions between frozen and non-frozen particles appropriately, as if the frozen particle has infinite mass.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various [output commands](#). This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

There can only be a single freeze fix defined. This is because other the [granular pair styles](#) treat frozen particles differently and need to be able to reference a single group to which this fix is applied.

Related commands: none

[atom_style sphere](#)

Default: none

fix gcmc command

Syntax:

```
fix ID group-ID gcmc N X M type seed T mu displace keyword values ...
```

- ID, group-ID are documented in fix command
- gcmc = style name of this fix command
- N = invoke this fix every N steps
- X = number of exchanges to attempt every N steps
- M = number of MC displacements to attempt every N steps
- type = atom type or molecule ID of exchanged gas
- seed = random # seed (positive integer)
- T = temperature of the ideal gas reservoir (temperature units)
- mu = chemical potential of the ideal gas reservoir (energy units)
- displace = maximum Monte Carlo displacement distance (length units)
- zero or more keyword/value pairs may be appended to args
- keyword = *molecule*, *region*, *maxangle*, *pressure*, or *fugacity_coeff*

```
molecule value = no or yes
  region value = region-ID
    region-ID = ID of region to use as an exchange/move volume
  maxangle value = maximum molecular rotation angle (degrees)
  pressure value = pressue of the gas reservoir (pressure units)
   fugacity_coeff value = fugacity coefficient of the gas reservoir (unitless)
```

Examples:

```
fix 2 gas gcmc 10 1000 1000 2 29494 298.0 -0.5 0.01
fix 3 Kr gcmc 10 100 100 1 3456543 3.0 -2.5 0.1 molecule yes maxangle 180
fix 4 my_gas gcmc 1 10 10 1 123456543 300.0 -12.5 1.0 region disk
```

Description:

This fix performs grand canonical Monte Carlo (GCMC) exchanges of atoms or molecules of the given type with an imaginary ideal gas reservoir at the specified T and chemical potential (mu) as discussed in (Frenkel). If used with the fix nvt command, simulations in the grand canonical enemble (muVT, constant chemical potential, constant volume, and constant temperature) can be performed. Specific uses include computing isotherms in microporous materials, or computing vapor-liquid coexistence curves.

Perform up to X exchanges of gas atoms or molecules of the given type between the simulation domain and the imaginary reservoir every N timesteps. Also perform M Monte Carlo displacements or rotations (for molecules) of gas of the given type within the simulation domain. M should typically be chosen to be approximately equal to the expected number of gas atoms or molecules of the given type within the domain, which will result in roughly one MC translation per atom or molecule per MC cycle.

For MC moves of molecular gasses, rotations and translations are each attempted with 50% probability. For MC moves of atomic gasses, translations are attempted 100% of the time. For MC exchanges of either molecular or atomic gasses, deletions and insertions are each attempted with 50% probability.

This fix cannot be used to perform MC insertions of gas atoms or molecules other than the exchanged type, but MC deletions, translations, and rotations can be performed on any atom/molecule in the fix group. All atoms in the simulation domain can be moved using regular time integration displacements, e.g. via fix_nvt, resulting in a hybrid GCMC+MD simulation. A smaller-than-usual timestep size may be needed when

running such a hybrid simulation, especially if the inserted molecules are not well equilibrated.

This command may optionally use the *region* keyword to define an exchange and move volume. The specified region must have been previously defined with a region command. It must be defined with side = *in*. Insertion attempts occur only within the specified region. Move and deletion attempt candidates are selected from gas atoms or molecules within the region. If no candidate can be found within the specified region after randomly selecting candidates 1000 times, the move or deletion attempt is considered a failure. Moves must start within the specified region, but may move the atom or molecule slightly outside of the region.

If used with [fix_nvt](fix_nvt), the temperature of the imaginary reservoir, T, should be set to be equivalent to the target temperature used in [fix_nvt](fix_nvt). Otherwise, the imaginary reservoir will not be in thermal equilibrium with the simulation domain.

Note that neighbor lists are re-built every timestep that this fix is invoked, so you should not set N to be too small. However, periodic rebuilds are necessary in order to avoid dangerous rebuilds and missed interactions. Specifically, avoid performing so many MC displacements per timestep that atoms can move beyond the neighbor list skin distance. See the neighbor command for details.

When an atom or molecule is to be inserted, its center-of-mass coordinates are chosen as a random position within the current simulation domain, and new atom velocities are randomly chosen from the specified temperature distribution given by T. Relative coordinates for atoms in a molecule are taken from the template molecule provided by the user. A random initial rotation is used in the case of molecule insertions.

If the setting for the *molecule* keyword is *no*, then only single atoms are exchanged. In this case, you should ensure you do not delete only a portion of a molecule (only some of its atoms), or LAMMPS will soon generate an error when it tries to find those atoms. LAMMPS will warn you if any of the atoms eligible for deletion have a non-zero molecule ID, but does not check for this at the time of deletion.

If the setting for the *molecule* keyword is *yes*, entire molecules are exchanged. The user must supply a model molecule in the data file to use as a template for exchanges, and that molecule's number must be given in the fix GCMC command as the "type" of the exchanged gas. Note that the model molecule must be present whenever the fix is initialized. This is a limitation that will likely be remedied in the not-to-distant future.

Optionally, users may specify the maximum rotation angle for molecular rotations using the *maxangle* keyword and specifying the angle in degrees. The specified angle will apply to all three Euler angles used internally to define the rotation matrix for molecular rotations. The max angle can be set to zero, but rotations will be pointless. Note that the default is ten degrees for each Euler angle.

For atomic gasses, inserted atoms have the specified atom type, but deleted atoms are any atoms that have been inserted or that belong to the user-specified fix group. For molecular gasses, exchanged molecules use the same atom types as in the template molecule supplied by the user. In both cases, exchanged atoms/molecules are assigned to two groups: the default group "all" and the group specified in the fix gcmc command (which can also be "all").

The gas reservoir pressure can be specified using the *pressure* keyword, in which case the user-specified chemical potential is ignored. For non-ideal gas reservoirs, the user may also specify the fugacity coefficient using the *fugacity_coeff* keyword.

Use of this fix typically will cause the number of atoms to fluctuate, therefore, you will want to use the compute_modify command to insure that the current number of atoms is used as a normalizing factor each time temperature is computed. Here is the necessary command:

```
compute_modify thermo_temp dynamic yes
```

If LJ units are used, note that a value of 0.18292026 is used by this fix as the reduced value for Planck's constant. This value was derived from LJ paramters for argon, where h* = h/sqrt(sigma^2 * epsilon * mass), sigma = 3.429 angstroms, epsilon/k = 121.85 K, and mass = 39.948 amu.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the deposition to binary restart files. This includes information about the random number generator seed, the next timestep for MC exchanges, etc. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix computes a global vector of length 6, which can be accessed by various output commands. The vector values are the following global cummulative quantities:

- 1 = displacement attempts
- 2 = displacement successes
- 3 = insertion attempts
- 4 = insertion successes
- 5 = deletion attempts
- 6 = deletion successes
- 7 = rotation attempts
- 8 = rotation successes

The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Do not set "neigh_modify once yes" or else this fix will never be called. Reneighboring is required.

Only pairwise interactions, as defined by the pair_style command, are included in this calculation. Long-range interactions due to a kspace_style command are not included. Not all pair potentials can be evaluated in a pairwise mode as required by this fix. For example, 3-body potentials, such as Tersoff and Stillinger-Weber cannot be used. EAM potentials for metals only include the pair potential portion of the EAM interaction, not the embedding term.

Can be run in parallel, but aspects of the GCMC part will not scale well in parallel. Only usable for 3D simulations with orthogonal simulation cells.

Note that very lengthy simulations involving insertions/deletions of billions of gas molecules may run out of atom or molecule IDs and trigger an error, so it is better to run multiple shorter-duration simulations. Likewise, very large molecules have not been tested and may turn out to be problematic.

Use of multiple fix gcmc commands in the same input script can be problematic if using a template molecule. The issue is that the user-referenced template molecule in the second fix gcmc command may no longer exist since it might have been deleted by the first fix gcmc command. An existing template molecule will need to be referenced by the user for each subsequent fix gcmc command.

Related commands:

[fix_nvt](), [neighbor](), [fix_deposit](), [fix_evaporate](), [delete_atoms]()

Default:

The option defaults are molecule = no, maxangle = 10.

(Frenkel) Frenkel and Smit, Understanding Molecular Simulation, Academic Press, London, 2002.

fix gld command

Syntax:

```
fix ID group-ID gld Tstart Tstop N_k seed series c_1 tau_1 ... c_N_k tau_N_k keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- gld = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- N_k = number of terms in the Prony series representation of the memory kernel
- seed = random number seed to use for white noise (positive integer)
- series = *pprony* is presently the only available option
- c_k = the weight of the kth term in the Prony series (mass per time units)
- tau_k = the time constant of the kth term in the Prony series (time units)
- zero or more keyword/value pairs may be appended

```
  keyword = frozen or zero
    frozen value = no or yes
      no = initialize extended variables using values drawn from equilibrium distribution at
      yes = initialize extended variables to zero (i.e., from equilibrium distribution at ze:
    zero value = no or yes
      no = do not set total random force to zero
      yes = set total random force to zero
```

Examples:

```
fix 1 all gld 1.0 1.0 2 82885 pprony 0.5 1.0 1.0 2.0 frozen yes zero yes
fix 3 rouse gld 7.355 7.355 4 48823 pprony 107.1 0.02415 186.0 0.04294 428.6 0.09661 1714 0.38643
```

Description:

Applies Generalized Langevin Dynamics to a group of atoms, as described in [(Baczewski)](#). This is intended to model the effect of an implicit solvent with a temporally non-local dissipative force and a colored Gaussian random force, consistent with the Fluctuation-Dissipation Theorem. The functional form of the memory kernel associated with the temporally non-local force is constrained to be a Prony series.

IMPORTANT NOTE: While this fix bears many similarities to [fix langevin](#), it has one significant difference. Namely, [fix gld](#) performs time integration, whereas [fix langevin](#) does NOT. To this end, the specification of another fix to perform time integration, such as [fix nve](#), is NOT necessary.

With this fix active, the force on the *j*th atom is given as

$$\mathbf{F}_j(t) = \mathbf{F}_j^C(t) - \int_0^t \Gamma_j(t-s)\mathbf{v}_j(s)\,ds + \mathbf{F}_j^R(t)$$

$$\Gamma_j(t-s) = \sum_{k=1}^{N_k} \frac{C_k}{\tau_k} e^{-(t-s)/\tau_k}$$

$$\langle \mathbf{F}_j^R(t), \mathbf{F}_j^R(s) \rangle = k_B T\, \Gamma_j(t-s)$$

Here, the first term is representative of all conservative (pairwise, bonded, etc) forces external to this fix, the second is the temporally non-local dissipative force given as a Prony series, and the third is the colored Gaussian random force.

The Prony series form of the memory kernel is chosen to enable an extended variable formalism, with a number of exemplary mathematical features discussed in (Baczewski). In particular, 3N_k extended variables are added to each atom, which effect the action of the memory kernel without having to explicitly evaluate the integral over time in the second term of the force. This also has the benefit of requiring the generation of uncorrelated random forces, rather than correlated random forces as specified in the third term of the force.

Presently, the Prony series coefficients are limited to being greater than or equal to zero, and the time constants are limited to being greater than zero. To this end, the value of series MUST be set to *pprony*, for now. Future updates will allow for negative coefficients and other representations of the memory kernel. It is with these updates in mind that the series option was included.

The units of the Prony series coefficients are chosen to be mass per time to ensure that the numerical integration scheme stably approaches the Newtonian and Langevin limits. Details of these limits, and the associated numerical concerns are discussed in (Baczewski).

The desired temperature at each timestep is ramped from *Tstart* to *Tstop* over the course of the next run.

The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics of the system will not be identical on two runs on different numbers of processors.

The keyword/value option pairs are used in the following ways.

The keyword *frozen* can be used to specify how the extended variables associated with the GLD memory kernel are initialized. Specifying no (the default), the initial values are drawn at random from an equilibrium distribution at *Tstart*, consistent with the Fluctuation-Dissipation Theorem. Specifying yes, initializes the extended variables to zero.

The keyword *zero* can be used to eliminate drift due to the thermostat. Because the random forces on different atoms are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the center-of-mass of the system undergoes a slow random walk. If the keyword *zero* is set to *yes*, the total random force is set exactly to zero by subtracting off an equal part of it from each atom in the group. As a result, the center-of-mass of a system with zero initial momentum will not drift over time.

Restart, run start/stop, minimize info:

fix gld command   367

The instantaneous values of the extended variables are written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix langevin, fix viscous, pair_style dpd/tstat

Default:

The option defaults are frozen = no, zero = no.

(Baczewski) A.D. Baczewski and S.D. Bond, J. Chem. Phys. 139, 044107 (2013).

fix gravity command

fix gravity/cuda command

fix gravity/omp command

Syntax:

```
fix ID group gravity magnitude style args
```

- ID, group are documented in [fix](#) command
- gravity = style name of this fix command
- magnitude = size of acceleration (force/mass units)
- magnitude can be a variable (see below)
- style = *chute* or *spherical* or *gradient* or *vector*

```
  chute args = angle
      angle = angle in +x away from -z or -y axis in 3d/2d (in degrees)
      angle can be a variable (see below)
    spherical args = phi theta
      phi = azimuthal angle from +x axis (in degrees)
      theta = angle from +z or +y axis in 3d/2d (in degrees)
      phi or theta can be a variable (see below)
    vector args = x y z
      x y z = vector direction to apply the acceleration
      x or y or z can be a variable (see below)
```

Examples:

```
fix 1 all gravity 1.0 chute 24.0
fix 1 all gravity v_increase chute 24.0
fix 1 all gravity 1.0 spherical 0.0 -180.0
fix 1 all gravity 10.0 spherical v_phi v_theta
fix 1 all gravity 100.0 vector 1 1 0
```

Description:

Impose an additional acceleration on each particle in the group. This fix is typically used with granular systems to include a "gravity" term acting on the macroscopic particles. More generally, it can represent any kind of driving field, e.g. a pressure gradient inducing a Poiseuille flow in a fluid. Note that this fix operates differently than the [fix addforce](#) command. The addforce fix adds the same force to each atom, independent of its mass. This command imparts the same acceleration to each atom (force/mass).

The *magnitude* of the acceleration is specified in force/mass units. For granular systems (LJ units) this is typically 1.0. See the [units](#) command for details.

Style *chute* is typically used for simulations of chute flow where the specified *angle* is the chute angle, with flow occurring in the +x direction. For 3d systems, the tilt is away from the z axis; for 2d systems, the tilt is away from the y axis.

Style *spherical* allows an arbitrary 3d direction to be specified for the acceleration vector. *Phi* and *theta* are defined in the usual spherical coordinates. Thus for acceleration acting in the -z direction, *theta* would be 180.0 (or -180.0). *Theta* = 90.0 and *phi* = -90.0 would mean acceleration acts in the -y direction. For 2d systems, *phi* is ignored and *theta* is an angle in the xy plane where *theta* = 0.0 is the y-axis.

Style *vector* imposes an acceleration in the vector direction given by (x,y,z). Only the direction of the vector is important; it's length is ignored. For 2d systems, the *z* component is ignored.

Any of the quantities *magnitude*, *angle*, *phi*, *theta*, *x*, *y*, *z* which define the gravitational magnitude and direction, can be specified as an equal-style variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the quantity. You should insure that the variable calculates a result in the approriate units, e.g. force/mass or degrees.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent gravitational field.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the gravitational potential energy of the system to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. This scalar is the gravitational potential energy of the particles in the defined field, namely mass * (g dot x) for each particles, where x and mass are the particles position and mass, and g is the gravitational field. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

atom_style sphere, fix addforce

Default: none

fix heat/gran command

fix heat/gran/conduction command

Syntax:

```
fix ID group-ID heat/gran initial_temperature T0 keyword values
```

```
fix ID group-ID heat/gran/conduction initial_temperature T0 keyword values
```

- ID, group-ID are documented in fix command
- heat/gran/conduction or fix heat/gran = style name of this fix command
- initial_temperature = obligatory keyword
- T0 = initial (default) temperature for the particles
- zero or more keyword/value pairs may be appended
- keyword = *contact_area* or *area_correction*

  contact_area values = *overlap* or *constant* **value** or *projection*
    area_correction values = *yes* or *no*

Examples:

```
fix 3 hg heat/gran/conduction initial_temperature 273.15
```

LIGGGHTS vs. LAMMPS info:

This command is not available in LAMMPS.

Description:

Calculates heat conduction between particles in contact and temperature update according to

$$\dot{Q}_{pi-pj} = h_{c,i-j} \, \Delta T_{pi-pj}$$

$$h_{c,i-j} = \frac{4 k_{pi} k_{pj}}{k_{pi} + k_{pj}} \left(A_{contact,i-j} \right)^{1/2}$$

$$m_p c_p \frac{dT_{p,i}}{dt} = \underbrace{\sum_{contacts \, i-j} \dot{Q}_{pi-pj}}_{\substack{heat \; conduction \\ by \; contacts}} + \underbrace{\dot{Q}_{pi,source}}_{\substack{heat \; generation \\ due \; to \; sources, \\ e.g. \; reactions}}$$

h_c heat transfer coefficient $\left[\frac{J}{K \, s} \right]$

k_{pi} thermal conductivity of particle i $\left[\frac{J}{K \, s \, m} \right]$

c_p specific thermal capacity $\left[\frac{J}{kg \, K} \right]$

$A_{contact,i-j}$ particle contact area$[m^2]$

It is assumed that the temperature within the particles is uniform. To make particles adiabatic (so they do not change the temperature), do not include them in the fix group. However, heat transfer is calculated

fix heat/gran command   371

between particles in the group and particles not in the group (but temperature update is not performed for particles not in the group). Thermal conductivity and specific thermal capacity must be defined for each atom type used in the simulation by means of fix property/global commands:

```
fix id all property/global thermalConductivity peratomtype value_1 value_2 ...
(value_i=value for thermal conductivity of atom type i)

fix id all property/global thermalCapacity peratomtype value_1 value_2 ...
(value_i=value for thermal capacity of atom type i)
```

To set the temperature for a group of particles, you can use the set command with keyword *property/atom* and values *Temp T*. *T* is the temperature value you want the particles to have. To set heat sources (or sinks) for a group of particles, you can also use the set command with the set keyword: *property/atom* and the set values: *heatSource h* where *h* is the heat source value you want the particles to have (in Energy/time units). A negative value means it is a heat sink. Examples would be:

```
set region halfbed property/peratom Temp 800.
set region srcreg property/peratom heatSource 0.5
```

Contact area calculation:

Using keyword *contact_area*, you can choose from 3 modes of calulating the contact area for particle-particle heat transfer: If *overlap* is used, the contact area is calculated from the area of the sphere-sphere intersection If *constant* is used, a constant user-defined overlap area is assumed. If *projection* is used, the overlap area is assumed to be equal to rmin*rmin*Pi, where rmin is the radius of the smaller of the two particles in contact.

Area correction:

For *contact_area* = overlap, an area correction can additionally be performed using keyword *area_correction* to account for the fact that the Young's modulus might have been decreased in order to speed-up the simulation, thus artificially increasing the overlap. In this case, you have to specify the original Young's modulus of each material by means of a fix property/global command:

```
fix id all property/global youngsModulusOriginal peratomtype value_1 value_2 ...
(value_i=value for original Young's modulus of atom type i)
```

This area correction is performed by scaling the contact area with (Y*/Y*,orig)^a, where Y* and Y*,orig are calculated as defined in pair_style gran . The scaling factor is given as e.g. a=1 for a Hooke and a=2/3 for a Hertz interaction.

Output info:

You can visualize the heat sources by accessing f_heatSource[0], and the heatFluxes by f_heatFlux[0] . With f_directionalHeatFlux[0], f_directionalHeatFlux[1] and f_directionalHeatFlux[2] you can access the conductive heat fluxes in x,y,z directions. The conductive heat fluxes are calculated per-contact and half the value is stored in each atom participating in the contact. With f_Temp[0] you can access the per-particle temperature. You can also access the total thermal energy of the fix group (useful for the thermo command) with f_id .

Restart, fix_modify, run start/stop, minimize info:

The particle temperature and heat source is written is written to binary restart files so simulations can continue properly. None of the fix_modify options are relevant to this fix.

This fix computes a scalar which can be accessed by various [output commands](). This scalar is the total thermal energy of the fix group

No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions:

Related commands:

[compute temp](), [compute temp/region]()

Default: *contact_area* = overlap, *area_correction = off*

fix heat command

Syntax:

```
fix ID group-ID heat N eflux
```

- ID, group-ID are documented in [fix](#) command
- heat = style name of this fix command
- N = add/subtract heat every this many timesteps
- eflux = rate of heat addition or subtraction (energy/time units)
- eflux can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix 3 qin heat 1 1.0
fix 3 qin heat 10 v_flux
fix 4 qout heat 1 -1.0 region top
```

Description:

Add non-translational kinetic energy (heat) to a group of atoms in a manner that conserves their aggregate momentum. Two of these fixes can be used to establish a temperature gradient across a simulation domain by adding heat (energy) to one group of atoms (hot reservoir) and subtracting heat from another (cold reservoir). E.g. a simulation sampling from the McDLT ensemble.

If the *region* keyword is used, the atom must be in both the group and the specified geometric [region](#) in order to have energy added or subtracted to it. If not specified, then the atoms in the group are affected wherever they may move to.

Heat addition/subtraction is performed every N timesteps. The *eflux* parameter can be specified as a numeric constant or as a variable (see below). If it is a numeric constant or equal-style variable which evaluates to a scalar value, then the *eflux* determines the change in aggregate energy of the entire group of atoms per unit time, e.g. in eV/psec for [metal units](#). In this case it is an "extensive" quantity, meaning its magnitude should be scaled with the number of atoms in the group. Note that since *eflux* has per-time units (i.e. it is a flux), this means that a larger value of N will add/subtract a larger amount of energy each time the fix is invoked.

If *eflux* is specified as an atom-style variable (see below), then the variable computes one value per atom. In this case, each value is the energy flux for a single atom, again in units of energy per unit time. In this case, each value is an "intensive" quantity, which need not be scaled with the number of atoms in the group.

As mentioned above, the *eflux* parameter can be specified as an equal-style or atom_style [variable](#). If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value(s) used to determine the flux.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent flux.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent flux with optional time-dependence as well.

IMPORTANT NOTE: If heat is subtracted from the system too aggressively so that the group's kinetic energy would go to zero, or any individual atom's kinetic energy would go to zero for the case where *eflux* is an atom-style variable, then LAMMPS will halt with an error message.

Fix heat is different from a thermostat such as fix nvt or fix temp/rescale in that energy is added/subtracted continually. Thus if there isn't another mechanism in place to counterbalance this effect, the entire system will heat or cool continuously. You can use multiple heat fixes so that the net energy change is 0.0 or use fix viscous to drain energy from the system.

This fix does not change the coordinates of its atoms; it only scales their velocities. Thus you must still use an integration fix (e.g. fix nve) on the affected atoms. This fix should not normally be used on atoms that have their temperature controlled by another fix - e.g. fix nvt or fix langevin fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. This scalar is the most recent value by which velocites were scaled. The scalar value calculated by this fix is "intensive". If *eflux* is specified as an atom-style variable, this fix computes the average value by which the velocities were scaled for all of the atoms that had their velocities scaled.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

compute temp, compute temp/region

Default: none

fix command

Syntax:

```
fix ID group-ID style args
```

- ID = user-assigned name for the fix
- group-ID = ID of the group of atoms to apply the fix to
- style = one of a long list of possible style names (see below)
- args = arguments used by a particular style

Examples:

```
fix 1 all nve
fix 3 all nvt temp 300.0 300.0 0.01
fix mine top setforce 0.0 NULL 0.0
```

Description:

Set a fix that will be applied to a group of atoms. In LAMMPS, a "fix" is any operation that is applied to the system during timestepping or minimization. Examples include updating of atom positions and velocities due to time integration, controlling temperature, applying constraint forces to atoms, enforcing boundary conditions, computing diagnostics, etc. There are dozens of fixes defined in LAMMPS and new ones can be added; see [this section](#) for a discussion.

Fixes perform their operations at different stages of the timestep. If 2 or more fixes operate at the same stage of the timestep, they are invoked in the order they were specified in the input script.

The ID of a fix can only contain alphanumeric characters and underscores.

Fixes can be deleted with the [unfix](#) command.

IMPORTANT NOTE: The [unfix](#) command is the only way to turn off a fix; simply specifying a new fix with a similar style will not turn off the first one. This is especially important to realize for integration fixes. For example, using a [fix nve](#) command for a second run after using a [fix nvt](#) command for the first run, will not cancel out the NVT time integration invoked by the "fix nvt" command. Thus two time integrators would be in place!

If you specify a new fix with the same ID and style as an existing fix, the old fix is deleted and the new one is created (presumably with new settings). This is the same as if an "unfix" command were first performed on the old fix, except that the new fix is kept in the same order relative to the existing fixes as the old one originally was. Note that this operation also wipes out any additional changes made to the old fix via the [fix_modify](#) command.

The [fix modify](#) command allows settings for some fixes to be reset. See the doc page for individual fixes for details.

Some fixes store an internal "state" which is written to binary restart files via the [restart](#) or [write_restart](#) commands. This allows the fix to continue on with its calculations in a restarted simulation. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file. See the doc pages for individual fixes for info on which ones can be restarted.

Some fixes calculate one of three styles of quantities: global, per-atom, or local, which can be used by other commands or output as described below. A global quantity is one or more system-wide values, e.g. the energy of a wall interacting with particles. A per-atom quantity is one or more values per atom, e.g. the displacement vector for each atom since time 0. Per-atom values are set to 0.0 for atoms not in the specified fix group. Local quantities are calculated by each processor based on the atoms it owns, but there may be zero or more per atoms.

Note that a single fix may produces either global or per-atom or local quantities (or none at all), but never more than one of these.

Global, per-atom, and local quantities each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for each fix describes the style and kind of values it produces, e.g. a per-atom vector. Some fixes produce more than one kind of a single style, e.g. a global scalar and a global vector.

When a fix quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID is the ID of the fix:

f_ID	entire scalar, vector, or array
f_ID[I]	one element of vector, one column of array
f_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the quantity once (vector -> scalar, array -> vector). Using two brackets reduces the dimension twice (array -> scalar). Thus a command that uses scalar fix values as input can also process elements of a vector or array.

Note that commands and variables which use fix quantities typically do not allow for all kinds, e.g. a command may require a vector of values, not a scalar. This means there is no ambiguity about referring to a fix quantity as f_ID even if it produces, for example, both a scalar and vector. The doc pages for various commands explain the details.

In LAMMPS, the values generated by a fix can be used in several ways:

- Global values can be output via the thermo_style custom or fix ave/time command. Or the values can be referenced in a variable equal or variable atom command.
- Per-atom values can be output via the dump custom command or the fix ave/spatial command. Or they can be time-averaged via the fix ave/atom command or reduced by the compute reduce command. Or the per-atom values can be referenced in an atom-style variable.
- Local values can be reduced by the compute reduce command, or histogrammed by the fix ave/histo command.

See this howto section for a summary of various LAMMPS output options, many of which involve fixes.

The results of fixes that calculate global quantities can be either "intensive" or "extensive" values. Intensive means the value is independent of the number of atoms in the simulation, e.g. temperature. Extensive means the value scales with the number of atoms in the simulation, e.g. total rotational kinetic energy. Thermodynamic output will normalize extensive values by the number of atoms in the system, depending on the "thermo_modify norm" setting. It will not normalize intensive values. If a fix value is accessed in another way, e.g. by a variable, you may want to know whether it is an intensive or extensive value. See the doc page for individual fixes for further info.

Each fix style has its own documentation page which describes its arguments and what it does, as listed below. Here is an alphabetic list of fix styles available in LAMMPS:

- adapt - change a simulation parameter over time

- [addforce](#) - add a force to each atom
- [append/atoms](#) - append atoms to a running simulation
- [aveforce](#) - add an averaged force to each atom
- [ave/atom](#) - compute per-atom time-averaged quantities
- [ave/histo](#) - compute/output time-averaged histograms
- [ave/spatial](#) - compute/output time-averaged per-atom quantities by layer
- [ave/time](#) - compute/output global time-averaged quantities
- [bond/break](#) - break bonds on the fly
- [bond/create](#) - create bonds on the fly
- [bond/swap](#) - Monte Carlo bond swapping
- [box/relax](#) - relax box size during energy minimization
- [deform](#) - change the simulation box size/shape
- [deposit](#) - add new atoms above a surface
- [drag](#) - drag atoms towards a defined coordinate
- [dt/reset](#) - reset the timestep based on velocity, forces
- [efield](#) - impose electric field on system
- [enforce2d](#) - zero out z-dimension velocity and force
- [evaporate](#) - remove atoms from simulation periodically
- [external](#) - callback to an external driver program
- [freeze](#) - freeze atoms in a granular simulation
- [gravity](#) - add gravity to atoms in a granular simulation
- [gcmc](#) - grand canonical insertions/deletions
- [heat](#) - add/subtract momentum-conserving heat
- [indent](#) - impose force due to an indenter
- [langevin](#) - Langevin temperature control
- [lineforce](#) - constrain atoms to move in a line
- [momentum](#) - zero the linear and/or angular momentum of a group of atoms
- [move](#) - move atoms in a prescribed fashion
- [msst](#) - multi-scale shock technique (MSST) integration
- [neb](#) - nudged elastic band (NEB) spring forces
- [nph](#) - constant NPH time integration via Nose/Hoover
- [nph/asphere](#) - NPH for aspherical particles
- [nph/sphere](#) - NPH for spherical particles
- [nphug](#) - constant-stress Hugoniostat integration
- [npt](#) - constant NPT time integration via Nose/Hoover
- [npt/asphere](#) - NPT for aspherical particles
- [npt/sphere](#) - NPT for spherical particles
- [nve](#) - constant NVE time integration
- [nve/asphere](#) - NVE for aspherical particles
- [nve/asphere/noforce](#) - NVE for aspherical particles without forces"
- [nve/body](#) - NVE for body particles
- [nve/limit](#) - NVE with limited step length
- [nve/line](#) - NVE for line segments
- [nve/noforce](#) - NVE without forces (v only)
- [nve/sphere](#) - NVE for spherical particles
- [nve/tri](#) - NVE for triangles
- [nvt](#) - constant NVT time integration via Nose/Hoover
- [nvt/asphere](#) - NVT for aspherical particles
- [nvt/sllod](#) - NVT for NEMD with SLLOD equations
- [nvt/sphere](#) - NVT for spherical particles
- [orient/fcc](#) - add grain boundary migration force
- [planeforce](#) - constrain atoms to move in a plane
- [poems](#) - constrain clusters of atoms to move as coupled rigid bodies
- [pour](#) - pour new atoms into a granular simulation domain

fix command

- press/berendsen - pressure control by Berendsen barostat
- print - print text and variables during a simulation
- property/atom - add customized per-atom values
- reax/bonds - write out ReaxFF bond information recenter - constrain the center-of-mass position of a group of atoms
- restrain - constrain a bond, angle, dihedral
- rigid - constrain one or more clusters of atoms to move as a rigid body with NVE integration
- rigid/nph - constrain one or more clusters of atoms to move as a rigid body with NPH integration
- rigid/npt - constrain one or more clusters of atoms to move as a rigid body with NPT integration
- rigid/nve - constrain one or more clusters of atoms to move as a rigid body with alternate NVE integration
- rigid/nvt - constrain one or more clusters of atoms to move as a rigid body with NVT integration
- rigid - constrain many small clusters of atoms to move as a rigid body with NVE integration
- setforce - set the force on each atom
- shake - SHAKE constraints on bonds and/or angles
- spring - apply harmonic spring force to group of atoms
- spring/rg - spring on radius of gyration of group of atoms
- spring/self - spring from each atom to its origin
- [srd](srd) - stochastic rotation dynamics (SRD)
- store/force - store force on each atom
- store/state - store attributes for each atom
- temp/berendsen - temperature control by Berendsen thermostat
- temp/rescale - temperature control by velocity rescaling
- thermal/conductivity - Muller-Plathe kinetic energy exchange for thermal conductivity calculation
- [tmd](tmd) - guide a group of atoms to a new configuration
- [ttm](ttm) - two-temperature model for electronic/atomic coupling
- viscosity - Muller-Plathe momentum exchange for viscosity calculation
- viscous - viscous damping for granular simulations
- wall/colloid - Lennard-Jones wall interacting with finite-size particles
- wall/gran - frictional wall(s) for granular simulations
- wall/harmonic - harmonic spring wall
- wall/lj126 - Lennard-Jones 12-6 wall
- wall/lj93 - Lennard-Jones 9-3 wall
- wall/piston - moving reflective piston wall
- wall/reflect - reflecting wall(s)
- wall/region - use region surface as wall
- wall/srd - slip/no-slip wall for SRD particles

There are also additional fix styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the fix section of this page.

There are also additional accelerated fix styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

Restrictions:

Some fix styles are part of specific packages. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual fixes tell if it is part of a package.

Related commands:

unfix, fix_modify

Default: none

fix imd command

Syntax:

```
fix ID group-ID imd trate port keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- imd = style name of this fix command
- port = port number on which the fix listens for an IMD client
- keyword = *unwrap* or *fscale* or *trate*

```
  unwrap arg = on or off
      off = coordinates are wrapped back into the principal unit cell (default)
      on = "unwrapped" coordinates using the image flags used
    fscale arg = factor
      factor = floating point number to scale IMD forces (default: 1.0)
    trate arg = transmission rate of coordinate data sets (default: 1)
    nowait arg = on or off
      off = LAMMPS waits to be connected to an IMD client before continuing (default)
      on = LAMMPS listens for an IMD client, but continues with the run
```

Examples:

```
fix vmd all imd 5678
fix comm all imd 8888 trate 5 unwrap on fscale 10.0
```

Description:

This fix implements the "Interactive MD" (IMD) protocol which allows realtime visualization and manipulation of MD simulations through the IMD protocol, as initially implemented in VMD and NAMD. Specifically it allows LAMMPS to connect an IMD client, for example the [VMD visualization program](#), so that it can monitor the progress of the simulation and interactively apply forces to selected atoms.

If LAMMPS is compiled with the preprocessor flag -DLAMMPS_ASYNC_IMD then fix imd will use POSIX threads to spawn a IMD communication thread on MPI rank 0 in order to offload data reading and writing from the main execution thread and potentially lower the inferred latencies for slow communication links. This feature has only been tested under linux.

There are example scripts for using this package with LAMMPS in examples/USER/imd. Additional examples and a driver for use with the Novint Falcon game controller as haptic device can be found at: http://sites.google.com/site/akohlmey/software/vrpn-icms.

The source code for this fix includes code developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. We thank them for providing a software interface that allows codes like LAMMPS to hook to [VMD](#).

Upon initialization of the fix, it will open a communication port on the node with MPI task 0 and wait for an incoming connection. As soon as an IMD client is connected, the simulation will continue and the fix will send the current coordinates of the fix's group to the IMD client at every trate MD step. When using r-RESPA, trate applies to the steps of the outmost RESPA level. During a run with an active IMD connection also the IMD client can request to apply forces to selected atoms of the fix group.

The port number selected must be an available network port number. On many machines, port numbers < 1024 are reserved for accounts with system manager privilege and specific applications. If multiple imd fixes would be active at the same time, each needs to use a different port number.

The *nowait* keyword controls the behavior of the fix when no IMD client is connected. With the default setting of *off*, LAMMPS will wait until a connection is made before continuing with the execution. Setting *nowait* to *on* will have the LAMMPS code be ready to connect to a client, but continue with the simulation. This can for example be used to monitor the progress of an ongoing calculation without the need to be permanently connected or having to download a trajectory file.

The *trate* keyword allows to select how often the coordinate data is sent to the IMD client. It can also be changed on request of the IMD client through an IMD protocol message. The *unwrap* keyword allows to send "unwrapped" coordinates to the IMD client that undo the wrapping back of coordinates into the principle unit cell, as done by default in LAMMPS. The *fscale* keyword allows to apply a scaling factor to forces transmitted by the IMD client. The IMD protocols stipulates that forces are transferred in kcal/mol/angstrom under the assumption that coordinates are given in angstrom. For LAMMPS runs with different units or as a measure to tweak the forces generated by the manipulation of the IMD client, this option allows to make adjustments.

To connect VMD to a listening LAMMPS simulation on the same machine with fix imd enabled, one needs to start VMD and load a coordinate or topology file that matches the fix group. When the VMD command prompts appears, one types the command line:

```
imd connect localhost 5678
```

This assumes that *fix imd* was started with 5678 as a port number for the IMD protocol.

The steps to do interactive manipulation of a running simulation in VMD are the following:

In the Mouse menu of the VMD Main window, select "Mouse -> Force -> Atom". You may alternately select "Residue", or "Fragment" to apply forces to whole residues or fragments. Your mouse can now be used to apply forces to your simulation. Click on an atom, residue, or fragment and drag to apply a force. Click quickly without moving the mouse to turn the force off. You can also use a variety of 3D position trackers to apply forces to your simulation. Game controllers or haptic devices with force-feedback such as the Novint Falcon or Sensable PHANTOM allow you to feel the resistance due to inertia or interactions with neighbors that the atoms experience you are trying to move, as if they were real objects. See the VMD IMD Homepage and the VRPN-ICMS Homepage for more details.

If IMD control messages are received, a line of text describing the message and its effect will be printed to the LAMMPS output screen, if screen output is active.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

When used in combination with VMD, a topology or coordinate file has to be loaded, which matches (in number and ordering of atoms) the group the fix is applied to. The fix internally sorts atom IDs by ascending

integer value; in VMD (and thus the IMD protocol) those will be assigned 0-based consecutive index numbers.

When using multiple active IMD connections at the same time, each needs to use a different port number.

Related commands: none

Default: none

fix indent command

Syntax:

```
fix ID group-ID indent K keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- indent = style name of this fix command
- K = force constant for indenter surface (force/distance^2 units)
- one or more keyword/value pairs may be appended
- keyword = *sphere* or *cylinder* or *plane* or *side* or *units*

```
  sphere args = x y z R
     x,y,z = initial position of center of indenter (distance units)
     R = sphere radius of indenter (distance units)
     any of x,y,z,R can be a variable (see below)
  cylinder args = dim c1 c2 R
     dim = x or y or z = axis of cylinder
     c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
     R = cylinder radius of indenter (distance units)
     any of c1,c2,R can be a variable (see below)
  plane args = dim pos side
     dim = x or y or z = plane perpendicular to this dimension
     pos = position of plane in dimension x, y, or z (distance units)
     pos can be a variable (see below)
     side = lo or hi
  side value = in or out
     in = the indenter acts on particles inside the sphere or cylinder
     out = the indenter acts on particles outside the sphere or cylinder
  units value = lattice or box
     lattice = the geometry is defined in lattice units
     box = the geometry is defined in simulation box units
```

Examples:

```
fix 1 all indent 10.0 sphere 0.0 0.0 15.0 3.0
fix 1 all indent 10.0 sphere v_x v_y 0.0 v_radius side in
fix 2 flow indent 10.0 cylinder z 0.0 0.0 10.0 units box
```

Description:

Insert an indenter within a simulation box. The indenter repels all atoms that touch it, so it can be used to push into a material or as an obstacle in a flow. Or it can be used as a constraining wall around a simulation; see the discussion of the *side* keyword below.

The indenter can either be spherical or cylindrical or planar. You must set one of those 3 keywords.

A spherical indenter exerts a force of magnitude

```
F(r) = - K (r - R)^2
```

on each atom where K is the specified force constant, r is the distance from the atom to the center of the indenter, and R is the radius of the indenter. The force is repulsive and F(r) = 0 for $r > R$.

A cylindrical indenter exerts the same force, except that r is the distance from the atom to the center axis of the cylinder. The cylinder extends infinitely along its axis.

Spherical and cylindrical indenters account for periodic boundaries in two ways. First, the center point of a spherical indenter (x,y,z) or axis of a cylindrical indenter (c1,c2) is remapped back into the simulation box, if the box is periodic in a particular dimension. This occurs every timestep if the indenter geometry is specified with a variable (see below), e.g. it is moving over time. Second, the calculation of distance to the indenter center or axis accounts for periodic boundaries. Both of these mean that an indenter can effectively move through and straddle one or more periodic boundaries.

A planar indenter is really an axis-aligned infinite-extent wall exerting the same force on atoms in the system, where *R* is the position of the plane and *r-R* is the distance from the plane. If the *side* parameter of the plane is specified as *lo* then it will indent from the lo end of the simulation box, meaning that atoms with a coordinate less than the plane's current position will be pushed towards the hi end of the box and atoms with a coordinate higher than the plane's current position will feel no force. Vice versa if *side* is specified as *hi*.

Any of the 4 quantities defining a spherical indenter's geometry can be specified as an equal-style [variable](), namely *x*, *y*, *z*, or *R*. Similarly, for a cylindrical indenter, any of *c1*, *c2*, or *R*, can be a variable. For a planar indenter, *pos* can be a variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to define the indenter geometry.

Note that equal-style variables can specify formulas with various mathematical functions, and include [thermo_style]() command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify indenter properties that change as a function of time or span consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the [run]() command and the *elaplong* keyword of [thermo_style custom]() for details.

For example, if a spherical indenter's x-position is specfied as v_x, then this variable definition will keep it's center at a relative position in the simulation box, 1/4 of the way from the left edge to the right edge, even if the box size changes:

```
variable x equal "xlo + 0.25*lx"
```

Similarly, either of these variable definitions will move the indenter from an initial position at 2.5 at a constant velocity of 5:

```
variable x equal "2.5 + 5*elaplong*dt"
variable x equal vdisplace(2.5,5)
```

If a spherical indenter's radius is specified as v_r, then these variable definitions will grow the size of the indenter at a specfied rate.

```
variable r0 equal 0.0
variable rate equal 1.0
variable r equal "v_r0 + step*dt*v_rate"
```

If the *side* keyword is specified as *out*, which is the default, then particles outside the indenter are pushded away from its outer surface, as described above. This only applies to spherical or cylindrical indenters. If the *side* keyword is specified as *in*, the action of the indenter is reversed. Particles inside the indenter are pushed away from its inner surface. In other words, the indenter is now a containing wall that traps the particles inside it. If the radius shrinks over time, it will squeeze the particles.

The *units* keyword determines the meaning of the distance units used to define the indenter geometry. A *box* value selects standard distance units as defined by the [units]() command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The [lattice]() command must have been previously used to define the lattice spacing. The (x,y,z) coords of the indenter position are scaled by the x,y,z lattice spacings respectively. The radius of a spherical or cylindrical indenter is scaled by the x lattice spacing.

Note that the units keyword only affects indenter geometry parameters specified directly with numbers, not those specified as variables. In the latter case, you should use the *xlat*, *ylat*, *zlat* keywords of the [thermo_style](#) command if you want to include lattice spacings in a variable formula.

The force constant *K* is not affected by the *units* keyword. It is always in force/distance^2 units where force and distance are defined by the [units](#) command. If you wish K to be scaled by the lattice spacing, you can define K with a variable whose formula contains *xlat*, *ylat*, *zlat* keywords of the [thermo_style](#) command, e.g.

```
variable k equal 100.0/xlat/xlat
fix 1 all indent $k sphere ...
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the energy of interaction between atoms and the indenter to the system's potential energy as part of [thermodynamic output](#). The energy of each particle interacting with the indenter is K/3 (r - R)^3.

This fix computes a global scalar energy and a global 3-vector of forces (on the indenter), which can be accessed by various [output commands](#). The scalar and vector values calculated by this fix are "extensive".

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. Note that if you define the indenter geometry with a variable using a time-dependent formula, LAMMPS uses the iteration count in the minimizer as the timestep. But it is almost certainly a bad idea to have the indenter change its position or size during a minimization. LAMMPS does not check if you have done this.

IMPORTANT NOTE: If you want the atom/indenter interaction energy to be included in the total potential energy of the system (the quantity being minimized), you must enable the [fix_modify](#) *energy* option for this fix.

Restrictions: none

Related commands: none

Default:

The option defaults are side = out and units = lattice.

fix insert/pack command

Syntax:

```
fix ID group-ID insert/pack seed seed_value distributiontemplate dist-ID general_keywords general
```

- ID, group-ID are documented in fix command
- insert/pack = style names of this fix command
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a fix_particledistribution_discrete to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattampt* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  insert_every value = once or ie
    once = value to signalise that isertion takes place only once (the step after the fix
    ie = every how many time-steps particles are inserted - insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current t
  overlapcheck value = yes or no
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
  omega values = constant omegax omegay omegaz
    constant = obligatory word
    omegax = x-comonent of angular velocity (1/time units)
    omegay = y-comonent of angular velocity (1/time units)
    omegaz = z-comonent of angular velocity (1/time units)
```

- following the general keyword/value section, one or more pack keyword/value pairs can be appended for the fix insert/pack command
- pack_keywords = *region* or *volumefraction_region* or *particles_in_region* or *mass_in_region* or *ntry_mc*

```
pack_keywords = where exactly one out of volumefraction_region or particles_in_region or ma
```

```
        region value = region-ID
          region-ID = ID of the region where the particles will be generated (positive integer)
        volumefraction_region values = vol
          vol =  desired volume fraction for the region (positive float, 0 <vol  <1)
        particles_in_region values = np
          np =  desired number of particles in the region (positive integer)
        mass_in_region values = m
          m =  desired mass in the region (positive float, m > 0)
        ntry_mc values = n
           n = number of Monte-Carlo steps for calculating the region's volume  (positive integer
```

Examples:

```
fix ins all insert/pack seed 1001 distributiontemplate pdd1 insert_every once overlapcheck yes vo
```

Description:

Insert particles into a granular run either once or every few timesteps within the specified region, as defined via the *region* keyword.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

This command must use the distributiontemplate keyword to refer to a fix_particledistribution_discrete (defined by dist-fix-ID) that defines the properties of the inserted particles.

At each insertion step, fix insert/pack tries inserts as many particles as needed to reach a defined target, which can be either a region volume fraction (keyword *volumefraction_region*), the total number of particles in the region (keyword *particles_in_region*), or the total particle mass in the region (keyword *mass_in_region*). Exactly one out of the keywords *volumefraction_region*, *particles_in_region*, *mass_in_region* must be defined.

The frequency of the particle insertion can be controlled by the keyword *insert_every*, which defines the number of time-steps between two insertions. Alternatively, by specifying *insert_every once*, particles are inserted only once.

The *start* keyword can be used to set the time-step at which the insertion should start.

Inserted particles are assigned the atom type specified by the particledistribution defined via the fix_particledistribution_discrete and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the fix_particledistribution_discrete and fix_particletemplate_sphere command (all of which can also be "all").

The keyword *overlapcheck* controls if overlap is checked for at insertion, both within the inserted particle package and with other existig particles. If this option is turned off, insertion will scale very well in parallel, otherwise not. Be aware that in case of no overlap check, highly overlapping configurations will be produced, so you will have to relax these configurations.

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in yes*) or only the particle center (*all_in no*). Currently *all_in yes* is not yet supported for all types of insertion.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation.

Description for fix insert/pack:

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. Dynamic regions are not supported as insertion region. Each timestep particles are inserted, they are placed randomly inside the insertion volume.

The *volumefraction* option specifies what volume fraction of the insertion volume will be filled with particles. The higher the value, the more particles are inserted each timestep. Since inserted particles should not overlap, the maximum volume fraction should be no higher than about 0.6.

To determine the volume of the insertion region, a Monte Carlo approach might be used for some cases where the volume is difficult to calculate or where the volume calculation is simply not implemented by the region. The *ntry_mc* keyword is used to control the number of MC tries that are used for the volume calculation.

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to binary restart files. This means you can restart a simulation while inserting particles, when the restart file was written during the insertion operation.

None of the fix_modify options are relevant to this fix. A global vector is stored by this fix for access by various output commands. The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command. This fix is not invoked during energy minimization.

Restrictions:

The *overlapcheck* = 'yes' option performs an inherently serial operation and will thus not scale well in parallel. For this reason, if you want to generate large systems, you are advised to turn *overlapcheck* off and let the packing relax afterwards to generate a valid packing.

Keywords *duration* and *extrude_length* can not be used together.

Currently *all_in yes* is not yet supported for all types of insertion.

Dynamic regions are not supported as insertion region.

Related commands:

fix_insert_stream, fix_insert_rate_region, fix_deposit, region, fix_pour

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, ntry_mc = 100000, random_distribute = exact

fix insert/rate/region command

Syntax:

```
fix ID group-ID insert/rate/region  seed seed_value distributiontemplate dist-ID general_keyword
```

- ID, group-ID are documented in fix command
- insert/pack and insert/stream = style names of these fix commands
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a fix_particledistribution_discrete to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattampt* or *nparticles* or *mass* or *particlerate* or *massrate* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  nparticles values = np or INF
    np =  number of particles to insert (positive integer)
    INF =  insert as many particles as possible
  mass values = mp
    mp = mass of particles to be inserted (positive float)
    INF =  insert as many particles as possible
  particlerate values = pr
    pr = particle inseration rate (particles/time units)
  massrate values = mr
    mr = mass inseration rate (mass/time units)
  insert_every value = once or ie
    once = value to signalise that isertion takes place only once (the step after the fix
    ie = every how many time-steps particles are inserted - insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current t
  overlapcheck value = yes or no
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
  omega values = constant omegax omegay omegaz
```

```
            constant = obligatory word
            omegax = x-comonent of angular velocity (1/time units)
            omegay = y-comonent of angular velocity (1/time units)
            omegaz = z-comonent of angular velocity (1/time units)
```

- following the general keyword/value section, one or more rate_region keyword/value pairs can be appended for the fix insert/rate/region command
- rate_region keywords = *region* or *ntry_mc*

```
    region value = region-ID
        region-ID = ID of the region where the particles will be generated (positive integer)
      ntry_mc values = n
        n = number of Monte-Carlo steps for calculating the region's volume  (positive integer
```

Examples:

```
fix ins all insert/rate/region seed 1001 distributiontemplate pdd1 nparticles 1000 particlerate 5
```

General description:

Insert particles into a granular run every few timesteps within a specified region at a specified rate.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a region command. Dynamic regions are not supported as insertion region. Each timestep particles are inserted, they are placed randomly inside the insertion volume.

To specify the number of particles to be inserted, you must use either the *nparticles* or the *mass* keyword (but not both). In the latter case, the number of particles to be inserted is calculated from the mass expectancy given by the particle distribution.

Likewise, you can use the *particlerate* or the *massrate* keyword (but not both) to control the insertion rate.

The frequency of the particle insertion is controlled by the keyword *insert_every*, which defines the number of time-steps between two insertions. The number of particles to be inserted at each insertion event is calculated from the insertion rate and *insert_every*. The *start* keyword can be used to set the time-step at which the insertion should start.

This command must use the distributiontemplate keyword to refer to a fix_particledistribution_discrete (defined by dist-fix-ID) that defines the properties of the inserted particles.

Inserted particles are assigned the atom type specified by the particledistribution defined via the fix_particledistribution_discrete and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the fix_particledistribution_discrete and fix_particletemplate_sphere command (all of which can also be "all").

The keyword *overlapcheck* controls if overlap is checked for at insertion, both within the inserted particle package and with other existig particles. If this option is turned off, insertion will scale very well in parallel, otherwise not. Be aware that in case of no overlap check, highly overlapping configurations will be produced, so you will have to relax these configurations.

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

fix insert/rate/region command

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in yes*) or only the particle center (*all_in no*). Currently *all_in yes* is not yet supported for all types of insertion.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation.

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to binary restart files. This means you can restart a simulation simulation while inserting particles, when the restart file was written during the insertion operation.

None of the fix_modify options are relevant to this fix. A global vector is stored by this fix for access by various output commands. The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

The *overlapcheck* = 'yes' option performs an inherently serial operation and will thus not scale well in parallel. For this reason, if you want to generate large systems, you are advised to turn *overlapcheck* off and let the packing relax afterwards to generate a valid packing.

Keywords *duration* and *extrude_length* can not be used together.

Currently *all_in yes* is not yet supported for all types of insertion.

Dynamic regions are not supported as insertion region.

Related commands:

fix_insert_stream, fix_insert_pack, fix_deposit, fix_gravity, region, fix_pour_dev, fix_pour

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, ntry_mc = 100000, random_distribute = exact

fix insert/stream command

Syntax:

```
fix ID group-ID insert/stream seed seed_value distributiontemplate dist-ID general_keywords gener
```

- ID, group-ID are documented in [fix](fix) command
- insert/stream = style name of this fix command
- seed = obligatory keyword
- seed_value = random # seed (positive integer)
- distributiontemplate = obligatory keyword
- dist-ID = ID of a fix_particledistribution_discrete to be used for particle insertion
- one or more general keyword/value pairs can be appended
- general_keywords = *verbose* or *maxattampt* or *nparticles* or *mass* or *particlerate* or *massrate* or *insert_every* or *overlapcheck* or *all_in* or *random_distribute* or *vel constant* or *vel uniform* or *vel gaussian* or *orientation* or *omega*

```
verbose = yes or no
  maxattempt value = ma
    ma = max # of insertion attempts per atom (positive integer)
  nparticles values = np or INF
    np =  number of particles to insert (positive integer)
    INF =  insert as many particles as possible
  mass values = mp
    mp = mass of particles to be inserted (positive float)
    INF =  insert as many particles as possible
  particlerate values = pr
    pr = particle inseration rate (particles/time units)
  massrate values = mr
    mr = mass inseration rate (mass/time units)
  insert_every value = ie
    ie = every how many time-steps particles are inserted - insertion happens periodically
  start value = ts
    ts = time-step at which insertion should start (positive integer larger than current t.
  all_in value = yes or no
  random_distribute value = exact or uncorrelated
  vel constant values = vx vy vz
    vx = x-velocity at insertion (velocity units)
    vy = y-velocity at insertion (velocity units)
    vz = z-velocity at insertion (velocity units)
  vel uniform values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = amplitude of uniform x-velocity fluctuation at insertion (velocity units)
    vFlucty = amplitude of uniform y-velocity fluctuation at insertion (velocity units)
    vFluctz = amplitude of uniform z-velocity fluctuation at insertion (velocity units)
  vel gaussian values = vx vy vz vFluctx vFlucty vFluctz
    vx = mean x-velocity at insertion (velocity units)
    vy = mean y-velocity at insertion (velocity units)
    vz = mean z-velocity at insertion (velocity units)
    vFluctx = standard deviation of Gaussian x-velocity fluctuation at insertion (velocity
    vFlucty = standard deviation of Gaussian y-velocity fluctuation at insertion (velocity
    vFluctz = standard deviation of Gaussian z-velocity fluctuation at insertion (velocity
  orientation values = random or template
    random = randomize rotational orientation
    template = use orientation from particle template
  omega values = constant omegax omegay omegaz
    constant = obligatory word
    omegax = x-comonent of angular velocity (1/time units)
```

```
        omegay = y-comonent of angular velocity (1/time units)
        omegaz = z-comonent of angular velocity (1/time units)
```
- following the general keyword/value section, one or more stream keyword/value pairs can be appended for the fix insert/stream command
- stream_keywords = *duration* or *parallel* or *insertion_face* or *extrude_length*

```
  insertion_face value = mesh-ID
     mesh-ID = ID of the fix mesh/surface or fix mesh/surface/planar to use as starting face
  extrude_length values = L
     L =  length for extruding the insertion face in normal direction so to generate in inse
  parallel values = yes or no
     yes, no = pre-calculate location of overlap of processor subdomains and extrusion volum
  duration values = du
     du =  duration of insertion in time-steps
```

Examples:

fix ins all insert/stream seed 1001 distributiontemplate pdd1 nparticles 5000 vel constant 0. -0.5 -2. particlerate 1000 overlapcheck yes insertion_face ins_mesh extrude_length 0.6

Description:

Insert particles into a granular run either once or every few timesteps within a specified region until either np particles have been inserted or the desired particle mass (mp) has been reached.

The *verbose* keyword controlls whether statistics about particle insertion is output to the screen each time particles are inserted.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. The insertion volume is generated by extruding the insertion face as specified via *insertion_face* in the direction of the face normal. The sign of this face normal is automatically flipped so that it is opposite to the normal component of the insertion velocity.

To specify the number of particles to be inserted, you must use either the *nparticles* or the *mass* keyword (but not both). In the latter case, the number of particles to be inserted is calculated from the mass expectancy given by the particle distribution. The *start* keyword can be used to set the time-step at which the insertion should start.

Likewise, you can use the *particlerate* or the *massrate* keyword (but not both) to control the insertion rate. Particles are not inserted continuously, but in packets (for efficiency reasons). Particles are inserted again after enough time has elapsed that the previously inserted particles have left the insertion volume.

One of the two keywords *insert_every* and *extrude_length* must be provided by the user (but not both).

In case *insert_every* is defined, this sets the frequency of the particle insertion directly, i.e. the number of time-steps between two insertions. The number of particles to be inserted at each insertion event is calculated from the insertion rate and *insert_every*.

If *extrude_length* is specified, the amount of extrusion is fixed and the insertion frequency is calculated from *extrude_length* and the insertion velocity normal to the insertion face.

When defining *insert_every*, you have the possibility to define the duration of each insertion via the *duration* keyword. *duration* < *insert_every* will generate a "pulsed" stream as opposed to a continuous stream. Example: Setting *insert_every* = 1000 and *duration* = 600 will produce a stream that pours material for 600 time-steps, will pause for 400 time-steps, pour for another 600 time-steps etc.

As mentioned above, particles are inserted again after enough time has elapsed that the previously inserted particles have left the insertion volume. Until the time these particles reach the insertion face, no other forces affect the particles (pair forces, gravity etc.). Fix insert/stream internally issues a special integrator to take care of this. This procedure guarantees that the specified velocity, omega etc. values are perfectly met at the specified insertion face.

The larger the volume, the more particles that can be inserted at one insertion step. Insertions will continue until the desired # of particles has been inserted.

NOTE: The insertion face must be a planar face, and the insertion velocity projected on the face normal must be non-zero.

NOTE: Keywords *insert_every* and *extrude_length* may not be used together

NOTE: Keywords *duration* and *extrude_length* cannot be used together.

This command must use the distributiontemplate keyword to refer to a [fix_particledistribution_discrete](#) (defined by dist-fix-ID) that defines the properties of the inserted particles.

Inserted particles are assigned the atom type specified by the particledistribution defined via the [fix_particledistribution_discrete](#) and are assigned to 4 groups: the default group "all" and the group specified in the fix insert command, as well as the groups specified in the [fix_particledistribution_discrete](#) and [fix_particletemplate_sphere](#) command (all of which can also be "all").

If overlapcheck if performed, the number of insertion attempts per particle can be specified via the *maxattempt* keyword. Each timestep particles are inserted, the command will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * ma. If unsuccessful at completing all insertions, a warning will be printed.

The *all_in* flag determines if the particle is completely contained in the insertion region (*all_in* = *yes*) or only the particle center (*all_in* = *no*).Using *all_in* = yes requires you to use an insertion face of style [fix mesh/surface/planar](#)

NOTE: You also have to use [fix mesh/surface/planar](#) if there is a [run](#) command between the definition of the insertion face and the fix insert/stream command. Otherwise, a [fix mesh/surface/planar](#) will do.

Keyword *random_distribute* controls how the number of particles to be inserted is distributed among parallel processors and among the particle templates in the particle distribution. For style *exact*, the number of particles to be inserted each step is matched exactly. For style *uncorrelated*, the number of particles to be inserted for each particle template will be rounded in an uncorrelated way, so the total number of inserted particles may vary for each insertion step. However, statistically both ways should produce the same result. Style *uncorrelated* may be faster in parallel since it does not need global MPI operations. Please note that if the # of particles to be inserted is calculated e.g. from a particle mass to be inserted, the number of particles to be inserted each insertion step will vary by 1, irrespective of the *random_distribute* settings. This is because in this case the # of particles to insert in each step will be a floating point number, and applying a simple floor/ceil rounding operation would lead to a statistical bias.

If keyword *parallel* is set to 'yes', LIGGGHTS tries to pre-calculate more accurately the overlap of process subdomains and extrusion volume. For cases where the insertion volume is highly divided between different processes, this can lead to a speed-up of insertion as random number generation is more efficient. For cases where the extrusion volume is divided among few processes this will impose a small computation overhead.

The initial velocity and rotational velocity can be controlled via the *vel* and *omega* keywords. *vel constant* simply patches a constant velocity to the inserted particles, *vel uniform* sets uniformly distributed velocities

with mean and amplitude. *vel gaussian* sets Gaussian distributed particle velocities with mean and std. deviation. The insertion velocity must be non-zero.

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to binary restart files. This means you can restart a simulation simulation while inserting particles, when the restart file was written during the insertion operation.

None of the fix_modify options are relevant to this fix. A global vector is stored by this fix for access by various output commands. The first component of the vector is the number of particles already inserted, the second component is the mass of particles already inserted. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Keywords *duration* and *extrude_length* can not be used together. The insertion face cannot move.

Related commands:

fix_insert_pack, fix_insert_rate_region, fix_deposit, fix_pour_dev, fix_pour

Default:

The defaults are maxattempt = 50, all_in = no, overlapcheck = yes vel = 0.0 0.0 0.0, omega = 0.0 0.0 0.0, start = next time-step, duration = insert_every, random_distribute = exact, parallel = no

fix langevin/eff command

Syntax:

```
fix ID group-ID langevin/eff Tstart Tstop damp seed keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- langevin/eff = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended

```
  keyword = scale or tally or zero
    scale values = type ratio
      type = atom type (1-N)
      ratio = factor by which to scale the damping coefficient
    tally values = no or yes
      no = do not tally the energy added/subtracted to atoms
      yes = do tally the energy added/subtracted to atoms

  zero value = no or yes
      no = do not set total random force to zero
      yes = set total random force to zero
```

Examples:

```
fix 3 boundary langevin/eff 1.0 1.0 10.0 699483
fix 1 all langevin/eff 1.0 1.1 10.0 48279 scale 3 1.5
```

Description:

Apply a Langevin thermostat as described in [(Schneider)](#) to a group of nuclei and electrons in the [electron force field](#) model. Used with [fix nve/eff](#), this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

```
F = Fc + Ff + Fr
Ff = - (m / damp) v
Fr is proportional to sqrt(Kb T m / (dt damp))
```

Fc is the conservative force computed via the usual inter-particle interactions ([pair_style](#)).

The Ff and Fr terms are added by this fix on a per-particle basis.

The operation of this fix is exactly like that described by the [fix langevin](#) command, except that the thermostatting is also applied to the radial electron velocity for electron particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Langevin thermostatting to the system's potential energy as part of thermodynamic output. Note that use of this option requires setting the *tally* keyword to *yes*.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive". Note that calculation of this quantity requires setting the *tally* keyword to *yes*.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

This fix is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix langevin

Default:

The option defaults are scale = 1.0 for all types and tally = no.

(Dunweg) Dunweg and Paul, Int J of Modern Physics C, 2, 817-27 (1991).

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

fix langevin command

Syntax:

```
fix ID group-ID langevin Tstart Tstop damp seed keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- langevin = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- Tstart can be a variable (see below)
- damp = damping parameter (time units)
- seed = random number seed to use for white noise (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *angmom* or *omega* or *scale* or *tally* or *zero*

```
  angmom value = no or scale
      no = do not thermostat rotational degrees of freedom via the angular momentum
      factor = do thermostat rotational degrees of freedom via the angular momentum and apply
    gjf value = no or yes
      no = use standard formulation
      yes = use Gronbech-Jensen/Farago formulation
    omega value = no or yes
      no = do not thermostat rotational degrees of freedom via the angular velocity
      yes = do thermostat rotational degrees of freedom via the angular velocity
    scale values = type ratio
      type = atom type (1-N)
      ratio = factor by which to scale the damping coefficient
    tally value = no or yes
      no = do not tally the energy added/subtracted to atoms
      yes = do tally the energy added/subtracted to atoms
    zero value = no or yes
      no = do not set total random force to zero
      yes = set total random force to zero
```

Examples:

```
fix 3 boundary langevin 1.0 1.0 1000.0 699483
fix 1 all langevin 1.0 1.1 100.0 48279 scale 3 1.5
fix 1 all langevin 1.0 1.1 100.0 48279 angmom 3.333
```

Description:

Apply a Langevin thermostat as described in [(Schneider)](#) to a group of atoms which models an interaction with a background implicit solvent. Used with [fix nve](#), this command performs Brownian dynamics (BD), since the total force on each atom will have the form:

```
F = Fc + Ff + Fr
Ff = - (m / damp) v
Fr is proportional to sqrt(Kb T m / (dt damp))
```

Fc is the conservative force computed via the usual inter-particle interactions ([pair_style](#), [bond_style](#), etc).

The Ff and Fr terms are added by this fix on a per-particle basis. See the [pair_style dpd/tstat](#) command for a thermostatting option that adds similar terms on a pairwise basis to pairs of interacting particles.

Ff is a frictional drag or viscous damping term proportional to the particle's velocity. The proportionality constant for each atom is computed as m/damp, where m is the mass of the particle and damp is the damping

factor specified by the user.

Fr is a force due to solvent atoms at a temperature T randomly bumping into the particle. As derived from the fluctuation/dissipation theorem, its magnitude as shown above is proportional to sqrt(Kb T m / dt damp), where Kb is the Boltzmann constant, T is the desired temperature, m is the mass of the particle, dt is the timestep size, and damp is the damping factor. Random numbers are used to randomize the direction and magnitude of this force as described in (Dunweg), where a uniform random number is used (instead of a Gaussian random number) for speed.

Note that unless you use the *omega* or *angmom* keywords, the thermostat effect of this fix is applied to only the translational degrees of freedom for the particles, which is an important consideration for finite-size particles, which have rotational degrees of freedom, are being thermostatted. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

IMPORTANT NOTE: Unlike the fix nvt command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies forces to effect thermostatting. Thus you must use a separate time integration fix, like fix nve to actually update the velocities and positions of atoms using the modified forces. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by fix nvt or fix temp/rescale commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*.

Tstart can be specified as an equal-style or atom-style variable. In this case, the *Tstop* setting is ignored. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the target temperature.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent temperature.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent temperature with optional time-dependence as well.

Like other fixes that perform thermostatting, this fix can be used with compute commands that remove a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or removing the x-component of velocity from the calculation. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

The *damp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec - see the units command). The damp factor can be thought of as inversely related to the viscosity of the solvent. I.e. a small relaxation time implies a hi-viscosity solvent and vice versa. See the discussion about gamma and viscosity in the documentation for the fix viscous command for more details.

The random # *seed* must be a positive integer. A Marsaglia random number generator is used. Each processor uses the input seed to generate its own unique seed and its own stream of random numbers. Thus the dynamics

of the system will not be identical on two runs on different numbers of processors.

The keyword/value option pairs are used in the following ways.

The keyword *angmom* and *omega* keywords enable thermostatting of rotational degrees of freedom in addition to the usual translational degrees of freedom. This can only be done for finite-size particles.

A simulation using atom_style sphere defines an omega for finite-size spheres. A simulation using atom_style ellipsoid defines a finite size and shape for aspherical particles and an angular momentum. The Langevin formulas for thermostatting the rotational degrees of freedom are the same as those above, where force is replaced by torque, m is replaced by the moment of inertia I, and v is replaced by omega (which is derived from the angular momentum in the case of aspherical particles).

The rotational temperature of the particles can be monitored by the compute temp/sphere and compute temp/asphere commands with their rotate options.

For the *omega* keyword there is also a scale factor of 10.0/3.0 that is applied as a multiplier on the Ff (damping) term in the equation above and of sqrt(10.0/3.0) as a multiplier on the Fr term. This does not affect the thermostatting behaviour of the Langevin formalism but insures that the randomized rotational diffusivity of spherical particles is correct.

For the *angmom* keyword a similar scale factor is needed which is 10.0/3.0 for spherical particles, but is anisotropic for aspherical particles (e.g. ellipsoids). Currently LAMMPS only applies an isotropic scale factor, and you can choose its magnitude as the specified value of the *angmom* keyword. If your aspherical particles are (nearly) spherical than a value of 10.0/3.0 = 3.333 is a good choice. If they are highly aspherical, a value of 1.0 is as good a choice as any, since the effects on rotational diffusivity of the particles will be incorrect regardless. Note that for any reasonable scale factor, the thermostatting effect of the *angmom* keyword on the rotational temperature of the aspherical particles should still be valid.

The keyword *scale* allows the damp factor to be scaled up or down by the specified factor for atoms of that type. This can be useful when different atom types have different sizes or masses. It can be used multiple times to adjust damp for several atom types. Note that specifying a ratio of 2 increases the relaxation time which is equivalent to the solvent's viscosity acting on particles with 1/2 the diameter. This is the opposite effect of scale factors used by the fix viscous command, since the damp factor in fix *langevin* is inversely related to the gamma factor in fix *viscous*. Also note that the damping factor in fix *langevin* includes the particle mass in Ff, unlike fix *viscous*. Thus the mass and size of different atom types should be accounted for in the choice of ratio values.

The keyword *tally* enables the calculation of the cumulative energy added/subtracted to the atoms as they are thermostatted. Effectively it is the energy exchanged between the infinite thermal reservoir and the particles. As described below, this energy can then be printed out or added to the potential energy of the system to monitor energy conservation.

The keyword *zero* can be used to eliminate drift due to the thermostat. Because the random forces on different atoms are independent, they do not sum exactly to zero. As a result, this fix applies a small random force to the entire system, and the center-of-mass of the system undergoes a slow random walk. If the keyword *zero* is set to *yes*, the total random force is set exactly to zero by subtracting off an equal part of it from each atom in the group. As a result, the center-of-mass of a system with zero initial momentum will not drift over time.

The keyword *gjf* can be used to run the Gronbech-Jensen/Farago time-discretization of the Langevin model. The effective random force is composed of the average of two random forces representing half-contributions from the previous and current time intervals. This discretization has been shown to be consistent with the underlying physical model of Langevin dynamics and produces the correct Boltzmann distribution of positions for large timesteps, up to the numerical stability limit. Because the discretized momenta generated

by the time integration scheme are not exactly conjugate to the positions, the kinetic energy distribution is systematically lower than the Boltzmann distribution by an amount that grows with the timestep.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. Because the state of the random number generator is not saved in restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Langevin thermostatting to the system's potential energy as part of thermodynamic output. Note that use of this option requires setting the *tally* keyword to *yes*.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive". Note that calculation of this quantity requires setting the *tally* keyword to *yes*.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nvt, fix temp/rescale, fix viscous, fix nvt, pair_style dpd/tstat

Default:

The option defaults are angmom = no, omega = no, scale = 1.0 for all types, tally = no, zero = no, gjf = no.

(Dunweg) Dunweg and Paul, Int J of Modern Physics C, 2, 817-27 (1991).

(Schneider) Schneider and Stoll, Phys Rev B, 17, 1302 (1978).

(Gronbech-Jensen) Gronbech-Jensen and Farago, Mol Phys, 111, 983 (2013); Gronbech-Jensen, Hayre, and Farago, arXiv:1303.7011.v2 (2013)

fix lb/fluid command

Syntax:

```
fix ID group-ID lb/fluid nevery LBtype viscosity density keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- lb/fluid = style name of this fix command
- nevery = update the lattice-Boltzmann fluid every this many timesteps
- LBtype = 1 to use the standard finite difference LB integrator, 2 to use the LB integrator of [Ollila et al.](#)
- viscosity = the fluid viscosity (units of mass/(time*length)).
- density = the fluid density.
- zero or more keyword/value pairs may be appended
- keyword = *setArea* or *setGamma* or *scaleGamma* or *dx* or *dm* or *a0* or *noise* or *calcforce* or *trilinear* or *D3Q19* or *read_restart* or *write_restart* or *zwall_velocity* or *bodyforce* or *printfluid*

```
  setArea values = type node_area
        type = atom type (1-N)
        node_area = portion of the surface area of the composite object associated with the p
    setGamma values = gamma
        gamma = user set value for the force coupling constant.
    scaleGamma values = type gammaFactor
        type = atom type (1-N)
        gammaFactor = factor to scale the setGamma gamma value by, for the specified atom ty
    dx values = dx_LB = the lattice spacing.
    dm values = dm_LB = the lattice-Boltzmann mass unit.
    a0 values = a_0_real = the square of the speed of sound in the fluid.
    noise values = Temperature seed
        Temperature = fluid temperature.
        seed = random number generator seed (positive integer)
    calcforce values = N forcegroup-ID
        N = output the force and torque every N timesteps
        forcegroup-ID = ID of the particle group to calculate the force and torque of
    trilinear values = none (used to switch from the default Peskin interpolation stencil to
    D3Q19 values = none (used to switch from the default D3Q15, 15 velocity lattice, to the
    read_restart values = restart file = name of the restart file to use to restart a fluid
    write_restart values = N = write a restart file every N MD timesteps.
    zwall_velocity values = velocity_bottom velocity_top = velocities along the y-direction
    bodyforce values = bodyforcex bodyforcey bodyforcez = the x,y and z components of a const
    printfluid values = N = print the fluid density and velocity at each grid point every N
```

Examples:

```
fix 1 all lb/fluid 1 2 1.0 1.0 setGamma 13.0 dx 4.0 dm 10.0 calcforce sphere1
fix 1 all lb/fluid 1 1 1.0 0.0009982071 setArea 1 1.144592082 dx 2.0 dm 0.3 trilinear noise 300.0
```

Description:

Implement a lattice-Boltzmann fluid on a uniform mesh covering the LAMMPS simulation domain. The MD particles described by *group-ID* apply a velocity dependent force to the fluid.

The lattice-Boltzmann algorithm solves for the fluid motion governed by the Navier Stokes equations,

$$\partial_t \rho + \partial_\beta \left(\rho u_\beta \right) = 0$$

$$\partial_t \left(\rho u_\alpha \right) + \partial_\beta \left(\rho u_\alpha u_\beta \right) = \partial_\beta \sigma_{\alpha\beta} + F_\alpha + \partial_\beta \left(\eta_{\alpha\beta\gamma\nu} \partial_\gamma u_\nu \right)$$

with,

$$\eta_{\alpha\beta\gamma\nu} = \eta \left[\delta_{\alpha\gamma} \delta_{\beta\nu} + \delta_{\alpha\nu} \delta_{\beta\gamma} - \frac{2}{3} \delta_{\alpha\beta} \delta_{\gamma\nu} \right] + \Lambda \delta_{\alpha\beta} \delta_{\gamma\nu}$$

where rho is the fluid density, u is the local fluid velocity, sigma is the stress tensor, F is a local external force, and eta and Lambda are the shear and bulk viscosities respectively. Here, we have implemented

$$\sigma_{\alpha\beta} = -P_{\alpha\beta} = -\rho a_0 \delta_{\alpha\beta}$$

with a_0 set to 1/3 (dx/dt)^2 by default.

The algorithm involves tracking the time evolution of a set of partial distribution functions which evolve according to a velocity discretized version of the Boltzmann equation,

$$\left(\partial_t + e_{i\alpha} \partial_\alpha \right) f_i = -\frac{1}{\tau} \left(f_i - f_i^{eq} \right) + W_i$$

where the first term on the right hand side represents a single time relaxation towards the equilibrium distribution function, and tau is a parameter physically related to the viscosity. On a technical note, we have implemented a 15 velocity model (D3Q15) as default; however, the user can switch to a 19 velocity model (D3Q19) through the use of the *D3Q19* keyword. This fix provides the user with the choice of two algorithms to solve this equation, through the specification of the keyword *LBtype*. If *LBtype* is set equal to 1, the standard finite difference LB integrator is used. If *LBtype* is set equal to 2, the algorithm of Ollila et al. is used.

Physical variables are then defined in terms of moments of the distribution functions,

$$\rho = \sum_i f_i$$

$$\rho u_\alpha = \sum_i f_i e_{i\alpha}$$

Full details of the lattice-Boltzmann algorithm used can be found in Mackay et al..

fix lb/fluid command

The fluid is coupled to the MD particles described by *group-ID* through a velocity dependent force. The contribution to the fluid force on a given lattice mesh site j due to MD particle alpha is calculated as:

$$\mathbf{F}_{j\alpha} = \gamma \left(\mathbf{v}_n - \mathbf{u}_f \right) \zeta_{j\alpha}$$

where v_n is the velocity of the MD particle, u_f is the fluid velocity interpolated to the particle location, and gamma is the force coupling constant. Zeta is a weight assigned to the grid point, obtained by distributing the particle to the nearest lattice sites. For this, the user has the choice between a trilinear stencil, which provides a support of 8 lattice sites, or the immersed boundary method Peskin stencil, which provides a support of 64 lattice sites. While the Peskin stencil is seen to provide more stable results, the trilinear stencil may be better suited for simulation of objects close to walls, due to its smaller support. Therefore, by default, the Peskin stencil is used; however the user may switch to the trilinear stencil by specifying the keyword, *trilinear*.

By default, the force coupling constant, gamma, is calculated according to

$$\gamma = \frac{2m_u m_v}{m_u + m_v} \left(\frac{1}{\Delta t_{collision}} \right)$$

Here, m_v is the mass of the MD particle, m_u is a representative fluid mass at the particle location, and dt_collision is a collision time, chosen such that tau/dt_collision = 1 (see Mackay and Denniston for full details). In order to calculate m_u, the fluid density is interpolated to the MD particle location, and multiplied by a volume, node_area*dx_lb, where node_area represents the portion of the surface area of the composite object associated with a given MD particle. By default, node_area is set equal to dx_lb*dx_lb; however specific values for given atom types can be set using the *setArea* keyword.

The user also has the option of specifying their own value for the force coupling constant, for all the MD particles associated with the fix, through the use of the *setGamma* keyword. This may be useful when modelling porous particles. See Mackay et al. for a detailed description of the method by which the user can choose an appropriate gamma value.

IMPORTANT NOTE: while this fix applies the force of the particles on the fluid, it does not apply the force of the fluid to the particles. When the force coupling constant is set using the default method, there is only one option to include this hydrodynamic force on the particles, and that is through the use of the lb/viscous fix. This fix adds the hydrodynamic force to the total force acting on the particles, after which any of the built-in LAMMPS integrators can be used to integrate the particle motion. However, if the user specifies their own value for the force coupling constant, as mentioned in Mackay et al., the built-in LAMMPS integrators may prove to be unstable. Therefore, we have included our own integrators fix lb/rigid/pc/sphere, and fix lb/pc, to solve for the particle motion in these cases. These integrators should not be used with the lb/viscous fix, as they add hydrodynamic forces to the particles directly. In addition, they can not be used if the force coupling constant has been set the default way.

IMPORTANT NOTE: if the force coupling constant is set using the default method, and the lb/viscous fix is NOT used to add the hydrodynamic force to the total force acting on the particles, this physically corresponds to a situation in which an infinitely massive particle is moving through the fluid (since collisions between the particle and the fluid do not act to change the particle's velocity). Therefore, the user should set the mass of the particle to be significantly larger than the mass of the fluid at the particle location, in order to approximate an infinitely massive particle (see the dragforce test run for an example).

Inside the fix, parameters are scaled by the lattice-Boltzmann timestep, dt, grid spacing, dx, and mass unit, dm. dt is set equal to (nevery*dt_MD), where dt_MD is the MD timestep. By default, dm is set equal to 1.0, and dx is chosen so that tau/(dt) = (3*eta*dt)/(rho*dx^2) is approximately equal to 1. However, the user has the option of specifying their own values for dm, and dx, by using the optional keywords *dm*, and *dx* respectively.

IMPORTANT NOTE: Care must be taken when choosing both a value for dx, and a simulation domain size. This fix uses the same subdivision of the simulation domain among processors as the main LAMMPS program. In order to uniformly cover the simulation domain with lattice sites, the lengths of the individual LAMMPS subdomains must all be evenly divisible by dx. If the simulation domain size is cubic, with equal lengths in all dimensions, and the default value for dx is used, this will automatically be satisfied.

Physical parameters describing the fluid are specified through *viscosity*, *density*, and *a0*. If the force coupling constant is set the default way, the surface area associated with the MD particles is specified using the *setArea* keyword. If the user chooses to specify a value for the force coupling constant, this is set using the *setGamma* keyword. These parameters should all be given in terms of the mass, distance, and time units chosen for the main LAMMPS run, as they are scaled by the LB timestep, lattice spacing, and mass unit, inside the fix.

The *setArea* keyword allows the user to associate a surface area with a given atom type. For example if a spherical composite object of radius R is represented as a spherical shell of N evenly distributed MD particles, all of the same type, the surface area per particle associated with that atom type should be set equal to 4*pi*R^2/N. This keyword should only be used if the force coupling constant, gamma, is set the default way.

The *setGamma* keyword allows the user to specify their own value for the force coupling constant, gamma, instead of using the default value.

The *scaleGamma* keyword should be used in conjunction with the *setGamma* keyword, when the user wishes to specify different gamma values for different atom types. This keyword allows the user to scale the *setGamma* gamma value by a factor, gammaFactor, for a given atom type.

The *dx* keyword allows the user to specify a value for the LB grid spacing.

The *dm* keyword allows the user to specify the LB mass unit.

If the *a0* keyword is used, the value specified is used for the square of the speed of sound in the fluid. If this keyword is not present, the speed of sound squared is set equal to (1/3)*(dx/dt)^2. Setting a0 > (dx/dt)^2 is not allowed, as this may lead to instabilities.

If the *noise* keyword is used, followed by a a positive temperature value, and a positive integer random number seed, a thermal lattice-Boltzmann algorithm is used. If *LBtype* is set equal to 1 (i.e. the standard LB integrator is chosen), the thermal LB algorithm of Adhikari et al. is used; however if *LBtype* is set equal to 2 both the LB integrator, and thermal LB algorithm described in Ollila et al. are used.

If the *calcforce* keyword is used, both the fluid force and torque acting on the specified particle group are printed to the screen every N timesteps.

If the keyword *trilinear* is used, the trilinear stencil is used to interpolate the particle nodes onto the fluid mesh. By default, the immersed boundary method, Peskin stencil is used. Both of these interpolation methods are described in Mackay et al..

If the keyword *D3Q19* is used, the 19 velocity (D3Q19) lattice is used by the lattice-Boltzmann algorithm. By default, the 15 velocity (D3Q15) lattice is used.

If the keyword *write_restart* is used, followed by a positive integer, N, a binary restart file is printed every N

LB timesteps. This restart file only contains information about the fluid. Therefore, a LAMMPS restart file should also be written in order to print out full details of the simulation.

IMPORTANT NOTE: When a large number of lattice grid points are used, the restart files may become quite large.

In order to restart the fluid portion of the simulation, the keyword *read_restart* is specified, followed by the name of the binary lb_fluid restart file to be used.

If the *zwall_velocity* keyword is used y-velocities are assigned to the lower and upper walls. This keyword requires the presence of walls in the z-direction. This is set by assigning fixed boundary conditions in the z-direction. If fixed boundary conditions are present in the z-direction, and this keyword is not used, the walls are assumed to be stationary.

If the *bodyforce* keyword is used, a constant body force is added to the fluid, defined by it's x, y and z components.

If the *printfluid* keyword is used, followed by a positive integer, N, the fluid densities and velocities at each lattice site are printed to the screen every N timesteps.

For further details, as well as descriptions and results of several test runs, see [Mackay et al.](). Please include a citation to this paper if the lb_fluid fix is used in work contributing to published research.

Restart, fix_modify, output, run start/stop, minimize info:

Due to the large size of the fluid data, this fix writes it's own binary restart files, if requested, independent of the main LAMMPS [binary restart files](); no information about *lb_fluid* is written to the main LAMMPS [binary restart files]().

None of the [fix_modify]() options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](). No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions:

This fix is part of the USER-LB package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS]() section for more info.

This fix can only be used with an orthogonal simulation domain.

Walls have only been implemented in the z-direction. Therefore, the boundary conditions, as specified via the main LAMMPS boundary command must be periodic for x and y, and either fixed or periodic for z. Shrink-wrapped boundary conditions are not permitted with this fix.

This fix must be used before any of [fix lb/viscous](), [fix lb/momentum](), [fix lb/rigid/pc/sphere](), and/ or [fix lb/pc]() , as the fluid needs to be initialized before any of these routines try to access its properties. In addition, in order for the hydrodynamic forces to be added to the particles, this fix must be used in conjunction with the [lb/viscous]() fix if the force coupling constant is set by default, or either the [lb/viscous]() fix or one of the [lb/rigid/pc/sphere]() or [lb/pc]() integrators, if the user chooses to specifiy their own value for the force coupling constant.

Related commands:

[fix lb/viscous](), [fix lb/momentum](), [fix lb/rigid/pc/sphere](), [fix lb/pc]()

fix lb/fluid command

Default:

By default, the force coupling constant is set according to

$$\gamma = \frac{2m_u m_v}{m_u + m_v} \left(\frac{1}{\Delta t_{collision}} \right)$$

and an area of dx_lb^2 per node, used to calculate the fluid mass at the particle node location, is assumed.

dx is chosen such that tau/(delta t_LB) = (3 eta dt_LB)/(rho dx_lb^2) is approximately equal to 1. dm is set equal to 1.0. a0 is set equal to (1/3)*(dx_lb/dt_lb)^2. The Peskin stencil is used as the default interpolation method. The D3Q15 lattice is used for the lattice-Boltzmann algorithm. If walls are present, they are assumed to be stationary.

(Ollila et al.) Ollila, S.T.T., Denniston, C., Karttunen, M., and Ala-Nissila, T., Fluctuating lattice-Boltzmann model for complex fluids, J. Chem. Phys. 134 (2011) 064902.

(Mackay et al.) Mackay, F. E., Ollila, S.T.T., and Denniston, C., Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 184 (2013) 2021-2031.

(Mackay and Denniston) Mackay, F. E., and Denniston, C., Coupling MD particles to a lattice-Boltzmann fluid through the use of conservative forces, J. Comput. Phys. 237 (2013) 289-298.

(Adhikari et al.) Adhikari, R., Stratford, K., Cates, M. E., and Wagner, A. J., Fluctuating lattice Boltzmann, Europhys. Lett. 71 (2005) 473-479.

fix lb/momentum command

Syntax:

```
fix ID group-ID lb/momentum nevery keyword values ...
```

- ID, group-ID are documented in the fix command
- lb/momentum = style name of this fix command
- nevery = adjust the momentum every this many timesteps
- zero or more keyword/value pairs may be appended
- keyword = *linear*

```
  linear values = xflag yflag zflag
      xflag,yflag,zflag = 0/1 to exclude/include each dimension.
```

Examples:

```
fix 1 sphere lb/momentum
fix 1 all lb/momentum linear 1 1 0
```

Description:

This fix is based on the fix momentum command, and was created to be used in place of that command, when a lattice-Boltzmann fluid is present.

Zero the total linear momentum of the system, including both the atoms specified by group-ID and the lattice-Boltzmann fluid every nevery timesteps. This is accomplished by adjusting the particle velocities and the fluid velocities at each lattice site.

NOTE: This fix only considers the linear momentum of the system.

By default, the subtraction is performed for each dimension. This can be changed by specifying the keyword *linear*, along with a set of three flags set to 0/1 in order to exclude/ include the corresponding dimension.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Can only be used if a lattice-Boltzmann fluid has been created via the fix lb/fluid command, and must come after this command.

This fix is part of the USER-LB package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix momentum, fix lb/fluid

Default:

Zeros the total system linear momentum in each dimension.

fix lb/pc command

Syntax:

```
fix ID group-ID lb/pc
```

- ID, group-ID are documented in the [fix](fix) command
- lb/pc = style name of this fix command

Examples:

```
fix 1 all lb/pc
```

Description:

Update the positions and velocities of the individual particles described by *group-ID*, experiencing velocity-dependent hydrodynamic forces, using the integration algorithm described in [Mackay et al.](Mackay). This integration algorithm should only be used if a user-specified value for the force-coupling constant used in fix lb/fluid has been set; do not use this integration algorithm if the force coupling constant has been set by default.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-LB package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Can only be used if a lattice-Boltzmann fluid has been created via the fix lb/fluid command, and must come after this command.

Related commands:

fix lb/fluid fix lb/rigid/pc/sphere

Default: None.

(Mackay et al.) Mackay, F. E., Ollila, S.T.T., and Denniston, C., Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 184 (2013) 2021-2031.

fix lb/rigid/pc/sphere command

Syntax:

```
fix ID group-ID lb/rigid/pc/sphere bodystyle args keyword values ...
```

- ID, group-ID are documented in fix command
- lb/rigid/pc/sphere = style name of this fix command
- bodystyle = *single* or *molecule* or *group*
- *single* args = none *molecule* args = none *group* args = N groupID1 groupID2 ... N = # of groups zero or more keyword/value pairs may be appended
- keyword = *force* or *torque* or *innerNodes*

```
    force values = M xflag yflag zflag
      M = which rigid body from 1-Nbody (see asterisk form below)
      xflag,yflag,zflag = off/on if component of center-of-mass force is active
    torque values = M xflag yflag zflag
      M = which rigid body from 1-Nbody (see asterisk form below)
      xflag,yflag,zflag = off/on if component of center-of-mass torque is active
    innerNodes values = innergroup-ID
      innergroup-ID = ID of the atom group which does not experience a hydrodynamic force fr
```

Examples:

```
fix 1 spheres lb/rigid/pc/sphere
fix 1 all lb/rigid/pc/sphere force 1 0 0 innerNodes ForceAtoms
```

Description:

This fix is based on the fix rigid command, and was created to be used in place of that fix, to integrate the equations of motion of spherical rigid bodies when a lattice-Boltzmann fluid is present with a user-specified value of the force-coupling constant. The fix uses the integration algorithm described in Mackay et al. to update the positions, velocities, and orientations of a set of spherical rigid bodies experiencing velocity dependent hydrodynamic forces. The spherical bodies are assumed to rotate as solid, uniform density spheres, with moments of inertia calculated using the combined sum of the masses of all the constituent particles (which are assumed to be point particles).

By default, all of the atoms that this fix acts on experience a hydrodynamic force due to the presence of the lattice-Boltzmann fluid. However, the *innerNodes* keyword allows the user to specify atoms belonging to a rigid object which do not interact with the lattice-Boltzmann fluid (i.e. these atoms do not feel a hydrodynamic force from the lattice-Boltzmann fluid). This can be used to distinguish between atoms on the surface of a non-porous object, and those on the inside.

This feature can be used, for example, when implementing a hard sphere interaction between two spherical objects. Instead of interactions occurring between the particles on the surfaces of the two spheres, it is desirable simply to place an atom at the center of each sphere, which does not contribute to the hydrodynamic force, and have these central atoms interact with one another.

Apart from the features described above, this fix is very similar to the rigid fix (although it includes fewer optional arguments, and assumes the constituent atoms are point particles); see fix_rigid for a complete documentation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about the *rigid* and *rigid/nve* fixes are written to [binary restart files](#).

Similar to the [fix_rigid](#) command: " The rigid fix computes a global scalar which can be accessed by various [output commands](#). The scalar value calculated by these fixes is "intensive". The scalar is the current temperature of the collection of rigid bodies. This is averaged over all rigid bodies and their translational and rotational degrees of freedom. The translational energy of a rigid body is $1/2\ m\ v^2$, where m = total mass of the body and v = the velocity of its center of mass. The rotational energy of a rigid body is $1/2\ I\ w^2$, where I = the moment of inertia tensor of the body and w = its angular velocity. Degrees of freedom constrained by the *force* and *torque* keywords are removed from this calculation."

"All of these fixes compute a global array of values which can be accessed by various [output commands](#). The number of rows in the array is equal to the number of rigid bodies. The number of columns is 15. Thus for each rigid body, 15 values are stored: the xyz coords of the center of mass (COM), the xyz components of the COM velocity, the xyz components of the force acting on the COM, the xyz components of the torque acting on the COM, and the xyz image flags of the COM, which have the same meaning as image flags for atom positions (see the "dump" command). The force and torque values in the array are not affected by the *force* and *torque* keywords in the fix rigid command; they reflect values before any changes are made by those keywords.

The ordering of the rigid bodies (by row in the array) is as follows. For the *single* keyword there is just one rigid body. For the *molecule* keyword, the bodies are ordered by ascending molecule ID. For the *group* keyword, the list of group IDs determines the ordering of bodies.

The array values calculated by these fixes are "intensive", meaning they are independent of the number of atoms in the simulation.

No parameter of these fixes can be used with the *start/stop* keywords of the [run](#) command. These fixes are not invoked during [energy minimization](#). "

Restrictions:

This fix is part of the USER-LB package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Can only be used if a lattice-Boltzmann fluid has been created via the [fix lb/fluid](#) command, and must come after this command. Should only be used if the force coupling constant used in [fix lb/fluid](#) has been set by the user; this integration fix cannot be used if the force coupling constant is set by default.

Related commands:

[fix lb/fluid](#), [fix lb/pc](#)

Default:

The defaults are force * on on on, and torque * on on on.

(Mackay et al.) Mackay, F. E., Ollila, S.T.T., and Denniston, C., Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 184 (2013) 2021-2031.

fix lb/viscous command

Syntax:

```
fix ID group-ID lb/viscous
```

- ID, group-ID are documented in [fix](#) command
- lb/viscous = style name of this fix command

Examples:

fix 1 flow lb/viscous

Description:

This fix is similar to the [fix viscous](#) command, and is to be used in place of that command when a lattice-Boltzmann fluid is present, and the user wishes to integrate the particle motion using one of the built in LAMMPS integrators.

This fix adds a force, F = - Gamma*(velocity-fluid_velocity), to each atom, where Gamma is the force coupling constant described in the [fix lb/fluid](#) command (which applies an equal and opposite force to the fluid).

IMPORTANT NOTE: This fix should only be used in conjunction with one of the built in LAMMPS integrators; it should not be used with the [fix lb/pc](#) or [fix lb/rigid/pc/sphere](#) integrators, which already include the hydrodynamic forces. These latter fixes should only be used if the force coupling constant has been set by the user (instead of using the default value); if the default force coupling value is used, then this fix provides the only method for adding the hydrodynamic forces to the particles.

For further details, as well as descriptions and results of several test runs, see [Mackay et al.](#). Please include a citation to this paper if this fix is used in work contributing to published research.

Restart, fix_modify, output, run start/stop, minimize info:

As described in the [fix viscous](#) documentation:

"No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command. This fix should only be used with damped dynamics minimizers that allow for non-conservative forces. See the [min_style](#) command for details."

Restrictions:

This fix is part of the USER-LB package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Can only be used if a lattice-Boltzmann fluid has been created via the [fix lb/fluid](#) command, and must come after this command.

This fix should not be used if either the [fix lb/pc](#) or [fix lb/rigid/pc/sphere](#) integrator is used.

Related commands:

[fix lb/fluid](#), [fix lb/pc](#), [fix lb/rigid/pc/sphere](#)

Default: none

(Mackay et al.) Mackay, F. E., Ollila, S.T.T., and Denniston, C., Hydrodynamic Forces Implemented into LAMMPS through a lattice-Boltzmann fluid, Computer Physics Communications 184 (2013) 2021-2031.

fix lineforce command

Syntax:

```
fix ID group-ID lineforce x y z
```

- ID, group-ID are documented in [fix](#) command
- lineforce = style name of this fix command
- x y z = direction of line as a 3-vector

Examples:

```
fix hold boundary lineforce 0.0 1.0 1.0
```

Description:

Adjust the forces on each atom in the group so that only the component of force along the linear direction specified by the vector (x,y,z) remains. This is done by subtracting out components of force in the plane perpendicular to the line.

If the initial velocity of the atom is 0.0 (or along the line), then it should continue to move along the line thereafter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

Restrictions: none

Related commands:

[fix planeforce](#)

Default: none

fix massflow/mesh command

Syntax:

```
fix id group massflow/mesh mesh mesh-ID vec_side vx vy vz keyword value ...
```

- ID, group-ID are documented in fix command
- massflow/mesh = style name of this fix command
- mesh = obligatory keyword
- mesh-ID = ID of a fix mesh/surface command
- vec_side = obligatory keyword
- vx, vy, vz = vector components defining the "outside" of the mesh
- zero or more keyword/value pairs may be appended to args
- keywords = *count* or *point_at_outlet* or *append* or *file* or *screen* or *delete_atoms*

```
  count value = once or multiple
      once = count particles only once
      multiple = allow particles to be counted multiple times
  point_at_outlet  pointX pointY pointZ
     pointX pointY pointZ = coordinates of point on the outlet side of the surface
  inside_out
     use this in connection with point_at_outlet to flip direction particle counting
  file value = filename
  append value = filename
   filename = name of the file to print diameter, position and velocity values of the part
  screen value = yes or no
  delete_atoms value = yes or no
     yes = to remove the particles that pass through the mesh surface
```

Examples:

```
fix mass all massflow/mesh mesh inface vec_side 0. 0. -1.

fix mass all massflow/mesh mesh inface count once point_at_outlet 0. 0. 0.
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command is not available in LAMMPS.

Description:

Fix massflow/mesh tracks how many particles penetrate through a mesh surface, as defined by a fix mesh/surface command. It counts the total number of particles and the associated mass. Only particles part of *group* are eligible for counting.

Particles are counted if they cross from the inner side of the mesh to the outer side of the mesh. The outer side can be defined by using the keyword *vec_side*, by specifying a point at the outlet side of the mesh (keyword *point_at_outlet*). Note that the vector defined by *vec_side* does not necessarily have to be perpendicular to the mesh face.

The following restrictions apply in case *vec_side* is specified: (i) the fix mesh/surface has to be planar, and (ii) the vector defined by *vec_side* may not lie inside the mesh plane.

The following restriction applies in case *point_at_outlet* is used: the *count* value has to be set to once.

The keyword *point_at_outlet* is especially useful in case a cylindrically-shaped surface is used. The *point_at_outlet* value should be on the cylinder axis in this case. If you like to track particles moving away from the cylinder axis, specify the *point_at_outlet* on the axis, and use the keyword *inside_out* to flip the direction.

When *count* = once, then each particle is only counted once, for *count* = multiple a particle contributes to number and mass count each time it crosses the mesh face. This can happen e.g. in the case of periodic boundary conditions or in re-circulating flow conditions.

The diameter, position and velocity of the particles can be written into a file using the *file* keyword, by specifying a filename.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

If the *delete_atoms* keyword is used then the particles passing through the mesh surface are deleted at the next re-neighboring step.

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to binary restart files .

This fix computes a per-atom vector (the marker) which can be accessed by various output commands. The per-atom vector (i.e., the marker) can be accessed by dumps by using "f_massflow_ID", . This fix also computes a global vector of length 6. This vector can be accessed via "f_ID", where ID is the fix id. The first vector component is equal to the total mass which has crossed the mesh surface, the second vector component indicates the particle count. The third vector component is equal to the total mass which has crossed the mesh surface since the last output divived by the time since the last output (i.e., the mass flow rate), the fourth vector component indicates the particle count since the last output divived by the time since the last output (i.e., the number rate of particles). The fifth and sixth vector components are the deleted mass and the number of deleted particles. This vector can also be accessed by various output commands.

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

compute nparticles/tracer/region

Default:

count = multiple, *inside_out* =false, *delete_atoms* = false

fix mesh/surface command

fix mesh/surface/planar command

Syntax:

```
fix ID group-ID mesh/surface file filename premesh_keywords premesh_values mesh_keywords mesh_val
fix ID group-ID mesh/surface/planar file filename premesh_keywords premesh_values mesh_keywords m
```

- ID, is documented in fix command, the group-ID is ignored for this command.
- mesh/surface or mesh/surface/planar = style name of this fix command
- file = obligatory keyword
- filename = name of STL or VTK file containing the triangle mesh data
- zero or more premesh_keywords/premesh_value pairs may be appended
- premesh_keyword = *type* or *precision* or *heal* or *verbose*
- *type* value = atom type (material type) of the wall imported from the STL file *precision* value = length mesh nodes this far away at maximum will be recognized as identical (length units) *heal* value = auto_remove_duplicates or no *verbose* value = yes or no zero or more mesh_keywords/mesh_value pairs may be appended
- mesh_keyword = *scale* or *move* or *rotate* or *temperature*

```
   scale value = factor
      factor = factor to scale the mesh in x-, y-, and z-direction (double value)
   move values = mx my mz
      mx my mz = move the mesh by this extent in x-, y-, and z-direction (length units)
   rotate values = axis ax ay az angle ang
      axis = obligatory keyword
      ax, ay, az = axis vector for rotation (length units)
      angle = obligatory keyword
      ang = angle to rotate the geometry around the specified axis (in degrees)
   temperature value = T0
      T0 = Temperature of the wall (temperature units)
```

- zero or more surface_keywords/surface_value pairs may be appended
- surface_keyword = *surface_vel* or *surface_ang_vel* or *curvature*

```
   surface_vel values = vx vy vz
      vx vy vz = conveyor belt surface velocity  (velocity units)
   surface_ang_vel values = origin ox oy oz axis ax ay az omega om
      origin = mandatory keyword
      ox oy oz = origin of rotation (length units)
      axis = mandatory keyword
      ax ay az = axis vector for rotation (length units)
      omega = mandatory keyword
      om = rotaional velocity around specifyied axis (rad/time units)
   curvature value = c
      c = maximum angle between mesh faces belonging to the same surface (in degree)
```

Examples:

```
fix cad all mesh/surface file mesh.stl type 1
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

This fix allows the import of triangual surfeace mesh wall geometry for granular simulations from ASCII STL files or legacy ASCII VTK files. Style *mesh/surface* is a general surface mesh, and *mesh/surface/planar* represents a planar mesh. *mesh/surface/planar* requires the mesh to consist of only 1 planar face.

Generall, you can apply scaling, offset and rotation to the imported mesh via keywords *scale*, *move*, *rotate*. Operations are applied in the order as they are specified. The group-ID is ignored for this command. For periodic boundaries, the mesh is mapped.

One fix represents one wall with a specific material, where the material is identified via keyword *type*. If multiple meshes with different materials are desired, the respective meshes must be imported with different fix mesh/surface commands.

With the *temperature* keyword, you can define a constant temperature for a mesh in conjunction with heat conduction via fix heat/gran. Note that the actual calculation of the heat transfer happens only if you use the mesh in conjunction with a granular wall, see fix wall/gran.

With the optional *surface_vel* keyword, you can specify the imported mesh as conveyor belt. The velocity direction for each mesh face is given by the projection of the conveyor belt velocity parallel to the mesh face, the velocity magnitude for each mesh face is equal to the conveyor belt velocity. This ensures the model makes sense also in case the mesh is curved. Likewise, the optional rotation model activated via keyword *surface_ang_vel* mimics rotational motion of the mesh (e.g. for modeling a shear cell)

The *precision* keyword specifies how far away mesh nodes can be at maximum to be recognized as identical.

If LIGGGHTS stalls because of duplicate elements, you can try setting *heal* to auto_remove_duplicates. LIGGGHTS will then try to heal the geometry by removing duplicate elements.

IMPORTANT NOTE: You should check the changes to the geometry, e.g. by using a dump mesh/stl command.

The *curvature* keyword lets you specify up to which angle between two triangles the triangles should be treated as belonging to the same surface (e.g. useful for bends). This angle is used to decide if (a) contact history is copied from one triangle to the other as the contact point proceeds and (b) if edge and corner interaction is calculated.

Quality checks / error and warning messages:

LIGGGHTS checks a couple of quality criteria upon loading a mesh. LIGGGHTS tries to give you as much information about the issue as possible.

Warning messages:

- There should be no angle $< 0.5Â°$ in any element
- The number of neighbor elements should be $<= 5$ for any element
- All nodes should be within the simlation box

    If any of the obove rules is not fulfilled, a warning is generated. Keyword *verbose* controls if details about the warning are written to the screen.

    Error messages:
- The *curvature* must not be larger than any angle in any mesh element
- Mesh elements must not be duplicate
- Coplanar mesh elements that share an edge must not overlap
- Mesh elements must not leave the simulation domain

fix mesh/surface/planar command

If any of the obove rules is not fulfilled, an error is generated and LIGGGHTS halts. Error messages are always *verbose*. If LIGGGHTS halts due to the last error, you might think about (a) changing the mesh import parameters (*scale*, *move*, *rotate*), (b) changing the mesh dynamics if a [fix move/mesh](#) is applied or using [boundary m m m](#)

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the STL data to binary [restart files](#) to be able to correctly resume the simulation in case the mesh is moved. None of the [fix_modify](#) options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

To date, only ASCII STL and VTK files can be read (binary is not supported). In the current implementation, each processor allocates memory for the whole geometry, which may lead to memory issues for very large geometries . It is not supported to use both the moving mesh and the conveyor belt feature.

Related commands:

[fix wall/gran](#)

Default: curvature = 0.256235 degrees, precision = 1e-8, verbose = no, heal = no

fix mesh/surface/stress command

Syntax:

```
fix ID group-ID mesh/surface/stress file filename premesh_keywords premesh_values mesh_keywords m
```

- ID, is documented in fix command, the group-ID is ignored for this command.
- mesh/surface/stress = style name of this fix command
- file filename premesh_keywords premesh_values mesh_keywords mesh_values surface_keyword surface_values are documented in fix mesh/surface.
- zero or more stress_keyword/value pairs may be appended
- stress_keyword = *stress* or *wear*

```
  stress value = on or off
    reference_point values = rx ry rz
      rx, ry, rz = coordinates of reference point
  wear value = finnie or off
```

Examples:

```
fix cad all mesh/surface/stress file mesh.stl type 1 wear finnie
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

This fix is identical to fix mesh/surface except for the fact that the pressure and shear force that the particles in the fix group exert on each triangle of the mesh is evaluated (which costs a bit of performance). Also, the total force and torque on the particle is calculated (see output info). The per-element forces can be dumped into VTK format using dump mesh/vtk.

With the optional *stress* keyword, stress tracking can be turned off is desired. The reference point for calculating the body torque can be controlled via the *referece_point* keyword. The optional *wear* keyword can activates a simple qualitative wear model (*finnie*) - for details on the model, see the seperate /doc/finnie-wear.pdf. The finnie constant k in Eqn. (4.23) has to be specified as follows:

```
fix id all property/global k_finnie peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
```

```
   (value_ij=value for the finnie constant between atom type i and j; n_atomtypes is the number
```

Restart, fix_modify, output, run start/stop, minimize info:

This fix stores a global vector with 6 components for access by various output commands. The first 3 components are equal to the total force on the mesh, the last 3 components store the total torque on the body exerted by the particles. Other info see fix mesh.

Related commands:

fix mesh/surface fix wall/gran

Default:

fix mesh/surface/stress command 423

stress = on reference_point = 0. 0. 0. wear = off

fix mesh/surface/stress/servo command

Syntax:

```
fix ID group-ID mesh/surface/stress/servo file filename premesh_keywords premesh_values mesh_keyw
```

- ID, is documented in fix command, the group-ID is ignored for this command.
- mesh/surface/stress/servo = style name of this fix command
- file filename premesh_keywords premesh_values mesh_keywords mesh_values surface_keyword
  surface_values stress_keywords stress_values are documented in fix mesh/surface/stress.
- zero or more servo_keyword/value pairs may be appended servo keywords = *com* (obligatory) or *dim*
  (obligatory) or *ctrlPV* (obligatory) or *vel_max* (obligatory) or *kp* or *ki* or *kd*

```
com values = x, y, z
    x, y, z = coordinates of the center of mass of the body (distance units)
ctrlPV values = force or torque
    force = use force as controll process value, i.e. control force
    torque = use torque as controll process value, i.e. control torque
axis args = x y z
    x y z = vector direction to apply the controlled mesh motion
    x or y or z can be a variable (see below)
target_val values = val
    val = target value for the controller (force units or torque units, depending on ctrlP
vel_max values = v
    v = maximum velocity magnitude for servo wall (velocity units)
kp values = k
    k = proportional constant for PID controller
ki values = k
    k = integral constant for PID controller
kd values = k
    k = differential constant for PID controller
mode values = auto
    auto = use alternative controller algorithm
ratio values = dr
    dr = constant for the alternative controller approach (mode = auto)
```

Examples:

```
fix servo all mesh/surface/stress/servo file plate.stl type 1 com 0. 0. 0. ctrlPV force axis 0. 0
fix servo all mesh/surface/stress/servo file stirrer.stl type 1 com 0. 0. 0. ctrlPV torque axis 0
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

This fix implements the functionality of fix mesh/surface/stress but it additionally assumes the mesh being a servo wall that compacts a particle packing until either a total force (for *ctrlPV* = force) or a total torque (for *ctrlPV* = torque) is acting on the mesh. The target value is defined via keyword *target_val*. The servo can act in any dimension (as specified by the *axis* keyword). Only the direction of the axis is important; it's length is ignored. A negative value for *target_val* leads to a wall motion towards negative *axis*-direction and vice versa. The user has to specify the center of mass (keyword *com*) and the maximum velocity allowed for the servo wall by keyword *vel_max*. Note that *vel_max* < skin /(2* timestep) is required.

The controller itself is a proportional-integral-derivative (PID) controller which is controlled by 3 constants *kp*, *ki*, *kd*:

output = kp * error + ki * errorsum + kd * errorchange

where 'error' is the current deviation of the controll process value to the target value, 'errorsum' is the time integration (sum) of the error values and 'errorchange' its derivative. The controller also includes an "anti-wind-up scheme" which prohibits accumulation of erroneous controller output caused by the integral part due to unavoidable long-lasting deviations.

By using the keyword *mode* = auto an alternative controller approach is applied. It is a pure proportional controller with gain scheduling. In the absence of neighbour particles the servo wall may move with maximum velocity (defined by *vel_max*). Otherwise, the maximum wall velocity is defined by *ratio* * min(radius) / dt. Approaching *target_val* the maximum velocity decreases to 0.1 * *ratio* * min(radius) / dt.

Restart, fix_modify, output, run start/stop, minimize info:

This fix stores a global vector with 9 components for access by various output commands. The first 3 components are equal to the total force on the mesh, the next 3 components store the total torque on the mesh. The last 3 components output the wall position. Furthermore, this fix writes the state of the servo wall to binary restart files so that a simulation can continue correctly. This fix supports fix_modify with option *integrate* = 'start' or 'stop' to start or stop the servo wall integration inbetween two runs. This fix also supports fix_modify with option *target_val* = val to change the target value inbetween two runs. This fix also supports fix_modify with option *ctrlParam* = kp ki kd to change the controller params inbetween two runs.

Restrictions:

When using this fix, along with scaling or rotate the body, all the servo_keyword/value pairs have to represent the state after scaling/rotation. Mesh elements may not be deleted in case due to leaving the simulation box for a fixed boundary. In this case, an error is generated. See boundary command for details. This fix can not be used in conjunction with another fix that manipulates mesh geometry, such as a fix move/mesh

Related commands:

fix mesh/surface/stress, fix wall/gran

Default:

kp = 1e-2, ki = 0, kd = 0

fix meso command

Syntax:

```
fix ID group-ID meso
```

- ID, group-ID are documented in fix command
- meso = style name of this fix command

Examples:

```
fix 1 all meso
```

Description:

Perform time integration to update position, velocity, internal energy and local density for atoms in the group each timestep. This fix is needed to time-integrate mesoscopic systems where particles carry internal variables such as SPH or DPDE.

See this PDF guide to using SPH in LAMMPS.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

"fix meso/stationary"

Default: none

fix meso/stationary command

Syntax:

```
fix ID group-ID meso/stationary
```

> • ID, group-ID are documented in [fix](#) command
> • meso = style name of this fix command

Examples:

```
fix 1 boundary meso/stationary
```

Description:

Perform time integration to update internal energy and local density, but not position or velocity for atoms in the group each timestep. This fix is needed for SPH simulations to correctly time-integrate fixed boundary particles which constrain a fluid to a given region in space.

See [this PDF guide](#) to using SPH in LAMMPS.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

"fix meso"

Default: none

fix_modify command

Syntax:

```
fix_modify fix-ID keyword value ...
```

- fix-ID = ID of the fix to modify
- one or more keyword/value pairs may be appended
- keyword = *temp* or *press* or *energy*

```
  temp value = compute ID that calculates a temperature
    press value = compute ID that calculates a pressure
    energy value = yes or no
```

Examples:

```
fix_modify 3 temp myTemp press myPress
fix_modify 1 energy yes
```

Description:

Modify one or more parameters of a previously defined fix. Only specific fix styles support specific parameters. See the doc pages for individual fix commands for info on which ones support which fix_modify parameters.

The *temp* keyword is used to determine how a fix computes temperature. The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a temperature. All fixes that compute temperatures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing T.

The *press* keyword is used to determine how a fix computes pressure. The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a pressure. All fixes that compute pressures define their own compute by default, as described in their documentation. Thus this option allows the user to override the default method for computing P.

For fixes that calculate a contribution to the potential energy of the system, the *energy* keyword will include that contribution in thermodynamic output of potential energy. See the [thermo_style](#) command for info on how potential energy is output. The contribution by itself can be printed by using the keyword f_ID in the thermo_style custom command, where ID is the fix-ID of the appropriate fix. Note that you must use this setting for a fix if you are using it when performing an [energy minimization](#) and if you want the energy and forces it produces to be part of the optimization criteria.

Restrictions: none

Related commands:

[fix](#), [compute temp](#), [compute pressure](#), [thermo_style](#)

Default:

The option defaults are temp = ID defined by fix, press = ID defined by fix, energy = no.

fix momentum command

Syntax:

```
fix ID group-ID momentum N keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- momentum = style name of this fix command
- N = adjust the momentum every this many timesteps one or more keyword/value pairs may be appended
- keyword = *linear* or *angular*

```
  linear values = xflag yflag zflag
      xflag,yflag,zflag = 0/1 to exclude/include each dimension
    angular values = none
```

Examples:

```
fix 1 all momentum 1 linear 1 1 0
fix 1 all momentum 100 linear 1 1 1 angular
```

Description:

Zero the linear and/or angular momentum of the group of atoms every N timesteps by adjusting the velocities of the atoms. One (or both) of the *linear* or *angular* keywords must be specified.

If the *linear* keyword is used, the linear momentum is zeroed by subtracting the center-of-mass velocity of the group from each atom. This does not change the relative velocity of any pair of atoms. One or more dimensions can be excluded from this operation by setting the corresponding flag to 0.

If the *angular* keyword is used, the angular momentum is zeroed by subtracting a rotational component from each atom.

This command can be used to insure the entire collection of atoms (or a subset of them) does not drift or rotate during the simulation due to random perturbations (e.g. [fix langevin](#) thermostatting).

Note that the [velocity](#) command can be used to create initial velocities with zero aggregate linear and/or angular momentum.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix recenter](#), [velocity](#)

Default: none

fix move command

Syntax:

```
fix ID group-ID move style args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- move = style name of this fix command
- style = *linear* or *wiggle* or *rotate* or *variable*

```
  linear args = Vx Vy Vz
     Vx,Vy,Vz = components of velocity vector (velocity units), any component can be specif
   wiggle args = Ax Ay Az period
     Ax,Ay,Az = components of amplitude vector (distance units), any component can be speci
     period = period of oscillation (time units)
   rotate args = Px Py Pz Rx Ry Rz period
     Px,Py,Pz = origin point of axis of rotation (distance units)
     Rx,Ry,Rz = axis of rotation vector
     period = period of rotation (time units)
   variable args = v_dx v_dy v_dz v_vx v_vy v_vz
     v_dx,v_dy,v_dz = 3 variable names that calculate x,y,z displacement as function of tim
     v_vx,v_vy,v_vz = 3 variable names that calculate x,y,z velocity as function of time, a
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
  units value = box or lattice
```

Examples:

```
fix 1 boundary move wiggle 3.0 0.0 0.0 1.0 units box
fix 2 boundary move rotate 0.0 0.0 0.0 0.0 0.0 1.0 5.0
fix 2 boundary move variable v_myx v_myy NULL v_VX v_VY NULL
```

Description:

Perform updates of position and velocity for atoms in the group each timestep using the specified settings or formulas, without regard to forces on the atoms. This can be useful for boundary or other atoms, whose movement can influence nearby atoms.

IMPORTANT NOTE: The atoms affected by this fix should not normally be time integrated by other fixes (e.g. [fix nve](#), [fix nvt](#)), since that will change their positions and velocities twice.

IMPORTANT NOTE: As atoms move due to this fix, they will pass thru periodic boundaries and be remapped to the other side of the simulation box, just as they would during normal time integration (e.g. via the [fix nve](#) command). It is up to you to decide whether periodic boundaries are appropriate with the kind of atom motion you are prescribing with this fix.

IMPORTANT NOTE: As dicsussed below, atoms are moved relative to their initial position at the time the fix is specified. These initial coordinates are stored by the fix in "unwrapped" form, by using the image flags associated with each atom. See the [dump custom](#) command for a discussion of "unwrapped" coordinates. See the Atoms section of the [read_data](#) command for a discussion of image flags and how they are set for each atom. You can reset the image flags (e.g. to 0) before invoking this fix by using the [set image](#) command.

The *linear* style moves atoms at a constant velocity, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + V * delta
```

where *X0* = (x0,y0,z0) is their position at the time the fix is specified, *V* is the specified velocity vector with components (Vx,Vy,Vz), and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to V = (Vx,Vy,Vz). If any of the velocity components is specified as NULL, then the position and velocity of that component is time integrated the same as the [fix nve](#) command would perform, using the corresponding force component on the atom.

Note that the *linear* style is identical to using the *variable* style with an [equal-style variable](#) that uses the vdisplace() function. E.g.

```
variable V equal 10.0
variable x equal vdisplace(0.0,$V)
fix 1 boundary move variable v_x NULL NULL v_V NULL NULL
```

The *wiggle* style moves atoms in an oscillatory fashion, so that their position X = (x,y,z) as a function of time is given in vector notation as

```
X(t) = X0 + A sin(omega*delta)
```

where *X0* = (x0,y0,z0) is their position at the time the fix is specified, *A* is the specified amplitude vector with components (Ax,Ay,Az), *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to the time derivative of this expression. If any of the amplitude components is specified as NULL, then the position and velocity of that component is time integrated the same as the [fix nve](#) command would perform, using the corresponding force component on the atom.

Note that the *wiggle* style is identical to using the *variable* style with [equal-style variables](#) that use the swiggle() and cwiggle() functions. E.g.

```
variable A equal 10.0
variable T equal 5.0
variable omega equal 2.0*PI/$T
variable x equal swiggle(0.0,$A,$T)
variable v equal v_omega*($A-cwiggle(0.0,$A,$T))
fix 1 boundary move variable v_x NULL NULL v_v NULL NULL
```

The *rotate* style rotates atoms around a rotation axis *R* = (Rx,Ry,Rz) that goes thru a point *P* = (Px,Py,Pz). The *period* of the rotation is also specified. This style also sets the velocity of each atom to (omega cross Rperp) where omega is its angular velocity around the rotation axis and Rperp is a perpendicular vector from the rotation axis to the atom. If the defined [atom_style](#) assigns an angular velocity to each atom, then each atom's angular velocity is also set to omega. Note that the direction of rotation for the atoms around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *variable* style allows the position and velocity components of each atom to be set by formulas specified via the [variable](#) command. Each of the 6 variables is specified as an argument to the fix as v_name, where name is the variable name that is defined elsewhere in the input script.

Each variable must be of either the *equal* or *atom* style. *Equal*-style variables compute a single numeric quantity, that can be a function of the timestep as well as of other simulation values. *Atom*-style variables compute a numeric quantity for each atom, that can be a function per-atom quantities, such as the atom's position, as well as of the timestep and other simulation values. Note that this fix stores the original coordinates of each atom (see note below) so that per-atom quantity can be used in an atom-style variable formula. See the [variable](#) command for details.

The first 3 variables (v_dx,v_dy,v_dz) specified for the *variable* style are used to calculate a displacement

from the atom's original position at the time the fix was specified. The second 3 variables (v_vx,v_vy,v_vz) specified are used to compute a velocity for each atom.

Any of the 6 variables can be specified as NULL. If both the displacement and velocity variables for a particular x,y,z component are specified as NULL, then the position and velocity of that component is time integrated the same as the fix nve command would perform, using the corresponding force component on the atom. If only the velocity variable for a component is specified as NULL, then the displacement variable will be used to set the position of the atom, and its velocity component will not be changed. If only the displacement variable for a component is specified as NULL, then the velocity variable will be used to set the velocity of the atom, and the position of the atom will be time integrated using that velocity.

The *units* keyword determines the meaning of the distance units used to define the *linear* velocity and *wiggle* amplitude and *rotate* origin. This setting is ignored for the *variable* style. A *box* value selects standard units as defined by the units command, e.g. velocity in Angstroms/fmsec and amplitude and position in Angstroms for units = real. A *lattice* value means the velocity units are in lattice spacings per time and the amplitude and position are in lattice spacings. The lattice command must have been previously used to define the lattice spacing. Each of these 3 quantities may be dependent on the x,y,z dimension, since the lattice spacings can be different in x,y,z.

For rRESPA time integration, this fix adjusts the position and velocity of atoms on the outermost rRESPA level.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of moving atoms to binary restart files, as well as the initial timestep, so that the motion can be continuous in a restarted simulation. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

IMPORTANNT NOTE: Because the move positions are a function of the current timestep and the initial timestep, you cannot reset the timestep to a different value after reading a restart file, if you expect a fix move command to work in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix produces a per-atom array which can be accessed by various output commands. The number of columns for each atom is 3, and the columns store the original unwrapped x,y,z coords of each atom. The per-atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve, displace_atoms

Default: none

The option default is units = lattice.

fix move/mesh command

Syntax:

```
fix ID group-ID move/mesh mesh mesh-ID style args keyword values ...
```

- ID is documented in [fix](#) command, group-ID is ignored
- move/mesh = style name of this fix command
- mesh = obligatory keyword
- mesh-ID = ID for the fix mesh that the fix move/mesh is applied to
- style = *linear* or *linear/variable* or *wiggle* or *riggle* or *rotate* or *rotate/variable* or *viblin* or *vibrot*

```
  linear args = Vx Vy Vz
     Vx,Vy,Vz = components of velocity vector (velocity units)
   linear/variable args = var_Vx var_Vy var_Vz
     var_Vx,var_Vy,var_Vz = variables specifying components of velocity vector (velocity un
   wiggle args = amplitude Ax Ay Az period per
     amplitude = obligatory keyword
     Ax,Ay,Az = components of amplitude vector (distance units)
     period = obligatory keyword
     per = period of oscillation (time units)
   riggle args = origin Px Py Pz axis ax ay az period per amplitude ampl
     origin = obligatory keyword
     Px,Py,Pz = origin point of axis of rotation (distance units)
     axis = obligatory keyword
     ax,ay,az = axis of rotation vector (distance units)
     period = obligatory keyword
     per = period of rotation (time units)#
     amplitude = obligatory keyword
     ampl = amplitude of riggle movement (grad)
   rotate args = origin Px Py Pz axis ax ay az period per
     origin = obligatory keyword
     Px,Py,Pz = origin point of axis of rotation (distance units)
     ax,ay,az = axis of rotation vector (distance units)
     period = obligatory keyword
     per = period of rotation (time units)
   rotate/variable args = origin Px Py Pz axis ax ay az omega var_omega
     origin = obligatory keyword
     Px,Py,Pz = origin point of axis of rotation (distance units)
     ax,ay,az = axis of rotation vector (distance units)
     omega = obligatory keyword
     var_omega = variable specifying angular velocity (rad / time units)
   viblin args = axis ax ay az order n amplitude C1 ... Cn phase p1 ... pn period per
     axis = obligatory keyword
     ax,ay,az = components of moving direction vector (distance units)(origin 0 0 0)
     order= obligatory keyword
     n= order of trigonometric series n (from 1 to 10)
     amplitude = obligatory keyword
     C1, ..., Cn = amplitude (distance units)
     phase = obligatory keyword
     p1, ...,pn = phase of functionterm (rad) (number of terms is equivalent to order n)
     period = obligatory keyword
     per = period of rotation (time units)
   vibrot args = origin Px Py Pz axis ax ay az order n amplitude C1 ... Cn phase p1 ... pn 
     origin = obligatory keyword
     Px,Py,Pz = origin point of axis of rotation (distance units)
     axis = obligatory keyword
     ax,ay,az = axis of rotation vector
     order= obligatory keyword
     n= order of trigonometric series (from 1 to 10)
     amplitude = obligatory keyword
```

```
C1, ..., Cn = amplitude (rad)
phase = obligatory keyword
p1, ...,pn = phase of functionterm (rad) (number of terms is equivalent to order n)
period = obligatory keyword
per = period of rotation (time units)
```

Examples:

```
fix move all move/mesh mesh cad1 wiggle amplitude -0.1 0. 0. period 0.02
fix move all move/mesh mesh cad1 rotate origin 0. 0. 0. axis  0. 0. 1. period 0.05
fix move all move/mesh mesh cad1 linear 5. 5. 0.
fix move all move/mesh mesh cad1 viblin axis 0. 0. 1 order 5 amplitude 0.4 0.1 0.3 0.1 0.1 phase
fix move all move/mesh mesh cad1 vibrot origin 0. 0. 0 axis 0. 0. 1 order 2 amplitude 0.4 0.1  ph
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

Perform updates of position and velocity for mesh elements which are part of the fix_mesh_surface with ID *mesh-ID* using the specified settings or formulas. The fix group is ignored for this command.

The *linear* style moves mesh elements at a constant velocity, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + V * delta
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, *V* is the specified velocity vector with components (Vx,Vy,Vz), and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each atom to V = (Vx,Vy,Vz).

The *linear/variable* style does the same as the *linear* style, but uses three variables so that the velocity can be time-dependant.

The *wiggle* style moves atoms in an oscillatory fashion, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

```
X(t) = X0 + A sin(omega*delta)
```

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, *A* is the specified amplitude vector with components (Ax,Ay,Az), *omega* is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each element to the time derivative of this expression.

The *rotate* style rotates around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The *period* of the rotation is also specified. This style also sets the velocity of each element to (omega cross Rperp) where omega is its angular velocity around the rotation axis and Rperp is a perpendicular vector from the rotation axis to the atom. Note that the direction of rotation around the rotation axis is consistent with the right-hand rule: if your right-hand's thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *rotate/variable* style does the same as the *rotate* style, but uses a variable for the angular velocity so that the angular velocity can be time-dependant. IMPORTANT NOTE: style *rotate* takes the period of the rotation as input, *rotate/variable* takes angular velocity as input.

The *riggle* style imposes an oscillatory rotation around a rotation axis $R = (Rx,Ry,Rz)$ that goes thru a point $P = (Px,Py,Pz)$. The *period* of the oscillation is specified as well as the *amplitude* in grad (Â°). This style also

sets the velocity of each element accordingly.

The *viblin* style moves meshes in an oscillatory fashion with an vibration function of higher order, so that their position $X = (x,y,z)$ as a function of time is given in vector notation as

$$X(t) = X_0 + D \cdot \sum_{n=1}^{k} C_n \cdot cos(n \cdot omega \cdot delta + phase_n)$$

where $X0 = (x0,y0,z0)$ is their position at the time the fix is specified, n represents the order of the trigonometric series, Cn is the specified amplitude along the direction given by *axis* = (ax,ay,az). The vector D is the unit vector of *axis*. The angular velocity omega is 2 PI / *period*, and *delta* is the time elapsed since the fix was specified. This style also sets the velocity of each element to the time derivative of this expression.

The *vibrot* style generates an oscillatory rotation around a rotation *axis* = (ax,ay,az) that goes thru a point *origin* = (Px,Py,Pz). The *period* of the oscillation is used to calculate omega, the amplitudes Cn and the phase phase n are given in rad. The change of rotation angle per time *gamma(t)* is described by trigonometric series of order n. The formula for this change is

$$gamma(t) = \sum_{n=1}^{k} C_n \cdot cos(n \cdot omega \cdot delta + phase_n)$$

This style also sets the velocity of each element accordingly

NOTE: If a dangerous tri neighbor list build is detected, this may be due to the fact that the geometry is moved too close to a region where particle insertion is taking place so that initial interpenetration happens when the particles are inserted.

NOTE: When moving a mesh element, not only the node positions are moved but also a couple of other vectors. So moving one mesh element is more costly as one particle.

Superposition of multiple fix move/mesh commands:

It is possible to superpose multiple fix move/mesh commands. In this case, the reference frame for the second move command moves along as the mesh is moved by the first move command etc. E.g. for style *rotate*, the origin of the rotation axis would be in local reference frame.

Example: A mesh should rotate around a central axis and additionally revolve around its center of mass. The first move command should be the rotation around the central axis, the second move command the revolution around the center of mass of the mesh.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of moving elements to binary restart files, so that the motion can be continuous in a restarted simulation. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

If multiple fix move/mesh movements are superposed onto one mesh, they have to be deleted in reverse order of their creation Mesh elements may not be deleted in case due to leaving the simulation box for a fixed boundary. In this case, an error is generated. See boundary command for details.

Related commands:

[fix mesh/surface](fix_mesh_surface)

Default: none

fix msst command

Syntax:

```
fix ID group-ID msst dir shockvel keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- msst = style name of this fix
- dir = *x* or *y* or *z*
- shockvel = shock velocity (strictly positive, distance/time units)
- zero or more keyword value pairs may be appended
- keyword = *q* or *mu* or *p0* or *v0* or *e0* or *tscale*

```
  q value = cell mass-like parameter (mass^2/distance^4 units)
    mu value = artificial viscosity (mass/length/time units)
    p0 value = initial pressure in the shock equations (pressure units)
    v0 value = initial simulation cell volume in the shock equations (distance^3 units)
    e0 value = initial total energy (energy units)
    tscale value = reduction in initial temperature (unitless fraction between 0.0 and 1.0)
```

Examples:

```
fix 1 all msst y 100.0 q 1.0e5 mu 1.0e5
fix 2 all msst z 50.0 q 1.0e4 mu 1.0e4  v0 4.3419e+03 p0 3.7797e+03 e0 -9.72360e+02 tscale 0.01
```

Description:

This command performs the Multi-Scale Shock Technique (MSST) integration to update positions and velocities each timestep to mimic a compressive shock wave passing over the system. See [(Reed)](#) for a detailed description of this method. The MSST varies the cell volume and temperature in such a way as to restrain the system to the shock Hugoniot and the Rayleigh line. These restraints correspond to the macroscopic conservation laws dictated by a shock front. *shockvel* determines the steady shock velocity that will be simulated.

To perform a simulation, choose a value of *q* that provides volume compression on the timescale of 100 fs to 1 ps. If the volume is not compressing, either the shock speed is chosen to be below the material sound speed or *p0* has been chosen inaccurately. Volume compression at the start can be sped up by using a non-zero value of *tscale*. Use the smallest value of *tscale* that results in compression.

Under some special high-symmetry conditions, the pressure (volume) and/or temperature of the system may oscillate for many cycles even with an appropriate choice of mass-like parameter *q*. Such oscillations have physical significance in some cases. The optional *mu* keyword adds an artificial viscosity that helps break the system symmetry to equilibrate to the shock Hugoniot and Rayleigh line more rapidly in such cases.

tscale is a factor between 0 and 1 that determines what fraction of thermal kinetic energy is converted to compressive strain kinetic energy at the start of the simulation. Setting this parameter to a non-zero value may assist in compression at the start of simulations where it is slow to occur.

If keywords *e0*, *p0*,or *v0* are not supplied, these quantities will be calculated on the first step, after the energy specified by *tscale* is removed. The value of *e0* is not used in the dynamical equations, but is used in calculating the deviation from the Hugoniot.

Values of shockvel less than a critical value determined by the material response will not have compressive solutions. This will be reflected in lack of significant change of the volume in the MSST.

For all pressure styles, the simulation box stays orthogonal in shape. Parrinello-Rahman boundary conditions (tilted box) are supported by LAMMPS, but are not implemented for MSST.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

See the [compute temp](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press". The group for the new computes is "all".

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of all internal variables to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The progress of the MSST can be monitored by printing the global scalar and global vector quantities computed by the fix.

The scalar is the cumulative energy change due to the fix. This is also the energy added to the potential energy by the [fix_modify](#) *energy* command. With this command, the thermo keyword *etotal* prints the conserved quantity of the MSST dynamic equations. This can be used to test if the MD timestep is sufficiently small for accurate integration of the dynamic equations. See also [thermo_style](#) command.

The global vector contains four values in this order:

[*dhugoniot*, *drayleigh*, *lagrangian_speed*, *lagrangian_position*]

1. *dhugoniot* is the departure from the Hugoniot (temperature units).
2. *drayleigh* is the departure from the Rayleigh line (pressure units).
3. *lagrangian_speed* is the laboratory-frame Lagrangian speed (particle velocity) of the computational cell (velocity units).
4. *lagrangian_position* is the computational cell position in the reference frame moving at the shock speed. This is usually a good estimate of distance of the computational cell behind the shock front.

To print these quantities to the log file with descriptive column headers, the following LAMMPS commands are suggested:

```
fix             msst all msst z
fix_modify      msst energy yes
variable dhug   equal f_msst[1]
variable dray   equal f_msst[2]
variable lgr_vel equal f_msst[3]
variable lgr_pos equal f_msst[4]
thermo_style    custom step temp ke pe lz pzz etotal v_dhug v_dray v_lgr_vel v_lgr_pos f_msst
```

These fixes compute a global scalar and a global vector of 4 quantities, which can be accessed by various [output commands](#). The scalar values calculated by this fix are "extensive"; the vector values are "intensive".

Restrictions:

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

All cell dimensions must be periodic. This fix can not be used with a triclinic cell. The MSST fix has been tested only for the group-ID all.

Related commands:

fix nphug, fix deform

Default:

The keyword defaults are q = 10, mu = 0, tscale = 0.01. p0, v0, and e0 are calculated on the first step.

(Reed) Reed, Fried, and Joannopoulos, Phys. Rev. Lett., 90, 235503 (2003).

fix neb command

Syntax:

```
fix ID group-ID neb Kspring
```

- ID, group-ID are documented in [fix](#) command
- neb = style name of this fix command
- Kspring = inter-replica spring constant (force/distance units)

Examples:

```
fix 1 active neb 10.0
```

Description:

Add inter-replica forces to atoms in the group for a multi-replica simulation run via the [neb](#) command to perform a nudged elastic band (NEB) calculation for transition state finding. Hi-level explanations of NEB are given with the [neb](#) command and in [Section_howto 5](#) of the manual. The fix neb command must be used with the "neb" command to define how inter-replica forces are computed.

Only the N atoms in the fix group experience inter-replica forces. Atoms in the two end-point replicas do not experience these forces, but those in intermediate replicas do. During the initial stage of NEB, the 3N-length vector of interatomic forces Fi = -Grad(V) acting on the atoms of each intermediate replica I is altered, as described in the [(Henkelman1)](#) paper, to become:

```
Fi = -Grad(V) + (Grad(V) dot That) That + Kspring (|Ri+i - Ri| - |Ri - Ri-1|) That
```

Ri are the atomic coordinates of replica I; Ri-1 and Ri+1 are the coordinates of its neighbor replicas. That (t with a hat over it) is the unit "tangent" vector for replica I which is a function of Ri, Ri-1, Ri+1, and the potential energy of the 3 replicas; it points roughly in the direction of (Ri+i - Ri-1); see the [(Henkelman1)](#) paper for details.

The first two terms in the above equation are the component of the interatomic forces perpendicular to the tangent vector. The last term is a spring force between replica I and its neighbors, parallel to the tangent vector direction with the specified spring constant *Kspring*.

The effect of the first two terms is to push the atoms of each replica toward the minimum energy path (MEP) of conformational states that transition over the energy barrier. The MEP for an energy barrier is defined as a sequence of 3N-dimensional states which cross the barrier at its saddle point, each of which has a potential energy gradient parallel to the MEP itself.

The effect of the last term is to push each replica away from its two neighbors in a direction along the MEP, so that the final set of states are equidistant from each other.

During the second stage of NEB, the forces on the N atoms in the replica nearest the top of the energy barrier are altered so that it climbs to the top of the barrier and finds the saddle point. The forces on atoms in this replica are described in the [(Henkelman2)](#) paper, and become:

```
Fi = -Grad(V) + 2 (Grad(V) dot That) That
```

The inter-replica forces for the other replicas are unchanged from the first equation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, as invoked by the minimize command via the neb command.

Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the Making LAMMPS section for more info on packages.

Related commands:

neb

Default: none

(Henkelman1) Henkelman and Jonsson, J Chem Phys, 113, 9978-9985 (2000).

(Henkelman2) Henkelman, Uberuaga, Jonsson, J Chem Phys, 113, 9901-9904 (2000).

fix nvt/eff command

fix npt/eff command

fix nph/eff command

Syntax:

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style_name = *nvt/eff* or *npt/eff* or *nph/eff*

```
  one or more keyword value pairs may be appended
  keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain
    temp values = Tstart Tstop Tdamp
      Tstart,Tstop = external temperature at start/end of run
      Tdamp = temperature damping parameter (time units)
    iso or aniso or tri values = Pstart Pstop Pdamp
      Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
      Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
      Pdamp = stress damping parameter (time units)
    couple = none or xyz or xy or yz or xz
    tchain value = length of thermostat chain (1 = single thermostat)
    pchain values = length of thermostat chain on barostat (0 = no thermostat)
    mtk value = yes or no = add in MTK adjustment term or not
    tloop value = number of sub-cycles to perform on thermostat
    ploop value = number of sub-cycles to perform on barostat thermostat
    nreset value = reset reference cell every this many timesteps
    drag value = drag factor added to barostat/thermostat (0.0 = no drag)
    dilate value = all or partial
```

Examples:

```
fix 1 all nvt/eff temp 300.0 300.0 0.1
fix 1 part npt/eff temp 300.0 300.0 0.1 iso 0.0 0.0 1.0
fix 2 part npt/eff temp 300.0 300.0 0.1 tri 5.0 5.0 1.0
fix 2 ice nph/eff x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3
```

Description:

These commands perform time integration on Nose-Hoover style non-Hamiltonian equations of motion for nuclei and electrons in the group for the [electron force field](#) model. The fixes are designed to generate positions and velocities sampled from the canonical (nvt), isothermal-isobaric (npt), and isenthalpic (nph) ensembles. This is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostatting) and simulation domain dimensions (barostatting). In addition to basic thermostatting and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time-averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The operation of these fixes is exactly like that described by the fix nvt, npt, and nph commands, except that the radius and radial velocity of electrons are also updated. Likewise the temperature and pressure calculated by the fix, using the computes it creates (as discussed in the fix nvt, npt, and nph doc page), are performed with computes that include the eFF contribution to the temperature or kinetic energy from the electron radial velocity.

IMPORTANT NOTE: there are two different pressures that can be reported for eFF when defining the pair_style (see pair eff/cut to understand these settings), one (default) that considers electrons do not contribute radial virial components (i.e. electrons treated as incompressible 'rigid' spheres) and one that does. The radial electronic contributions to the virials are only tallied if the flexible pressure option is set, and this will affect both global and per-atom quantities. In principle, the true pressure of a system is somewhere in between the rigid and the flexible eFF pressures, but, for most cases, the difference between these two pressures will not be significant over long-term averaged runs (i.e. even though the energy partitioning changes, the total energy remains similar).

IMPORTANT NOTE: currently, there is no available option for the user to set or create temperature distributions that include the radial electronic degrees of freedom with the velocity command, so the the user must allow for these degrees of freedom to equilibrate (i.e. equi-partitioning of energy) through time integration.

Restart, fix_modify, output, run start/stop, minimize info:

See the doc page for the fix nvt, npt, and nph commands for details.

Restrictions:

This fix is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Other restriction discussed on the doc page for the fix nvt, npt, and nph commands also apply.

IMPORTANT NOTE: The temperature for systems (regions or groups) with only electrons and no nuclei is 0.0 (i.e. not defined) in the current temperature calculations, a practical example would be a uniform electron gas or a very hot plasma, where electrons remain delocalized from the nuclei. This is because, even though electron virials are included in the temperature calculation, these are averaged over the nuclear degrees of freedom only. In such cases a corrective term must be added to the pressure to get the correct kinetic contribution.

Related commands:

fix nvt, fix nph, fix npt, fix_modify, run_style

Default:

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = ploop = 1, nreset = 0, drag = 0.0, dilate = all, and couple = none.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez-Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

fix nvt command

fix nvt/cuda command

fix nvt/omp command

fix npt command

fix npt/cuda command

fix npt/omp command

fix nph command

fix nph/omp command

Syntax:

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style_name = *nvt* or *npt* or *nph*

```
  one or more keyword value pairs may be appended
  keyword = temp or iso or aniso or tri or x or y or z or xy or yz or xz or couple or tchain
    temp values = Tstart Tstop Tdamp
      Tstart,Tstop = external temperature at start/end of run
      Tdamp = temperature damping parameter (time units)
    iso or aniso or tri values = Pstart Pstop Pdamp
      Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
      Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
      Pdamp = stress damping parameter (time units)
    couple = none or xyz or xy or yz or xz
    tchain value = N
      N = length of thermostat chain (1 = single thermostat)
    pchain values = N
      N length of thermostat chain on barostat (0 = no thermostat)
    mtk value = yes or no = add in MTK adjustment term or not
    tloop value = M
      M = number of sub-cycles to perform on thermostat
    ploop value = M
      M = number of sub-cycles to perform on barostat thermostat
    nreset value = reset reference cell every this many timesteps
    drag value = Df
      Df = drag factor added to barostat/thermostat (0.0 = no drag)
    dilate value = dilate-group-ID
      dilate-group-ID = only dilate atoms in this group due to barostat volume changes
    scalexy value = yes or no = scale xy with ly
    scaleyz value = yes or no = scale yz with lz
    scalexz value = yes or no = scale xz with lz
    flip value = yes or no = allow or disallow box flips when it becomes highly skewed
    fixedpoint values = x y z
      x,y,z = perform barostat dilation/contraction around this point (distance units)
```

Examples:

```
fix 1 all nvt temp 300.0 300.0 100.0
fix 1 water npt temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 jello npt temp 300.0 300.0 100.0 tri 5.0 5.0 1000.0
fix 2 ice nph x 1.0 1.0 0.5 y 2.0 2.0 0.5 z 3.0 3.0 0.5 yz 0.1 0.1 0.5 xz 0.2 0.2 0.5 xy 0.3 0.3
```

Description:

These commands perform time integration on Nose-Hoover style non-Hamiltonian equations of motion which are designed to generate positions and velocities sampled from the canonical (nvt), isothermal-isobaric (npt), and isenthalpic (nph) ensembles. This updates the position and velocity for atoms in the group each timestep.

The thermostatting and barostatting is achieved by adding some dynamic variables which are coupled to the particle velocities (thermostatting) and simulation domain dimensions (barostatting). In addition to basic thermostatting and barostatting, these fixes can also create a chain of thermostats coupled to the particle thermostat, and another chain of thermostats coupled to the barostat variables. The barostat can be coupled to the overall box volume, or to individual dimensions, including the *xy*, *xz* and *yz* tilt dimensions. The external pressure of the barostat can be specified as either a scalar pressure (isobaric ensemble) or as components of a symmetric stress tensor (constant stress ensemble). When used correctly, the time-averaged temperature and stress tensor of the particles will match the target values specified by Tstart/Tstop and Pstart/Pstop.

The equations of motion used are those of Shinoda et al in (Shinoda), which combine the hydrostatic equations of Martyna, Tobias and Klein in (Martyna) with the strain energy proposed by Parrinello and Rahman in (Parrinello). The time integration schemes closely follow the time-reversible measure-preserving Verlet and rRESPA integrators derived by Tuckerman et al. in (Tuckerman).

The thermostat parameters for fix styles *nvt* and *npt* is specified using the *temp* keyword. Other thermostat-related keywords are *tchain*, *tloop* and *drag*, which are discussed below.

The thermostat is applied to only the translational degrees of freedom for the particles. The translational degrees of freedom can also have a bias velocity removed before thermostatting takes place; see the description below. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 10.0 means to relax the temperature in a timespan of (roughly) 10 time units (e.g. tau or fmsec or psec - see the units command). The atoms in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the integration.

IMPORTANT NOTE: A Nose-Hoover thermostat will not work well for arbitrary values of *Tdamp*. If *Tdamp* is too small, the temperature can fluctuate wildly; if it is too large, the temperature will take a very long time to equilibrate. A good choice for many models is a *Tdamp* of around 100 timesteps. Note that this is NOT the same as 100 time units for most units settings.

The barostat parameters for fix styles *npt* and *nph* is specified using one or more of the *iso*, *aniso*, *tri*, *x*, *y*, *z*, *xy*, *xz*, *yz*, and *couple* keywords. These keywords give you the ability to specify all 6 components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation.

Other barostat-related keywords are *pchain*, *mtk*, *ploop*, *nreset*, *drag*, and *dilate*, which are discussed below.

Orthogonal simulation boxes have 3 adjustable dimensions (x,y,z). Triclinic (non-orthogonal) simulation boxes have 6 adjustable dimensions (x,y,z,xy,xz,yz). The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z*, *xy*, *xz*, *yz* keywords, which correspond to the 6 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the y-box length will change. If the *xy* keyword is used, the xy tilt factor will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the fix deform command.

Note that in order to use the *xy*, *xz*, or *yz* keywords, the simulation box must be triclinic, even if its initial tilt factors are 0.0.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. tau or fmsec or psec - see the units command).

IMPORTANT NOTE: A Nose-Hoover barostat will not work well for arbitrary values of *Pdamp*. If *Pdamp* is too small, the pressure and volume can fluctuate wildly; if it is too large, the pressure will take a very long time to equilibrate. A good choice for many models is a *Pdamp* of around 1000 timesteps. Note that this is NOT the same as 1000 time units for most units settings.

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a *dilate-group-ID* for a group that represents a subset of the atoms. This can be useful, for example, to leave the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid. This option should be used with care, since it can be unphysical to dilate some atoms and not others, because it can introduce large, instantaneous displacements between a pair of atoms (one dilated, one not) that are far from the dilation origin. Also note that for atoms not in the fix group, a separate time integration fix like fix nve or fix nvt can be used on them, independent of whether they are dilated or not.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso*, *aniso*, and *tri* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword *tri* means *x*, *y*, *z*, *xy*, *xz*, and *yz* dimensions are controlled independently using their individual stress components as the driving forces, and the specified scalar pressure as the external normal stress. Using "tri Pstart Pstop Pdamp" is the same as specifying these 7 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
xy 0.0 0.0 Pdamp
yz 0.0 0.0 Pdamp
xz 0.0 0.0 Pdamp
couple none
```

In some cases (e.g. for solids) the pressure (volume) and/or temperature of the system can oscillate undesirably when a Nose/Hoover barostat and thermostat is applied. The optional *drag* keyword will damp these oscillations, although it alters the Nose/Hoover equations. A value of 0.0 (no drag) leaves the Nose/Hoover formalism unchanged. A non-zero value adds a drag term; the larger the value specified, the greater the damping effect. Performing a short run and monitoring the pressure and temperature is the best way to determine if the drag term is working. Typically a value between 0.2 to 2.0 is sufficient to damp oscillations after a few periods. Note that use of the drag keyword will interfere with energy conservation and will also change the distribution of positions and velocities so that they do not correspond to the nominal NVT, NPT, or NPH ensembles.

An alternative way to control initial oscillations is to use chain thermostats. The keyword *tchain* determines the number of thermostats in the particle thermostat. A value of 1 corresponds to the original Nose-Hoover thermostat. The keyword *pchain* specifies the number of thermostats in the chain thermostatting the barostat degrees of freedom. A value of 0 corresponds to no thermostatting of the barostat variables.

The *mtk* keyword controls whether or not the correction terms due to Martyna, Tuckerman, and Klein are included in the equations of motion (Martyna). Specifying *no* reproduces the original Hoover barostat, whose volume probability distribution function differs from the true NPT and NPH ensembles by a factor of 1/V. Hence using *yes* is more correct, but in many cases the difference is negligible.

The keyword *tloop* can be used to improve the accuracy of integration scheme at little extra cost. The initial and final updates of the thermostat variables are broken up into *tloop* substeps, each of length *dt/tloop*. This corresponds to using a first-order Suzuki-Yoshida scheme (Tuckerman2006). The keyword *ploop* does the same thing for the barostat thermostat.

The keyword *nreset* controls how often the reference dimensions used to define the strain energy are reset. If this keyword is not used, or is given a value of zero, then the reference dimensions are set to those of the initial simulation domain and are never changed. If the simulation domain changes significantly during the simulation, then the final average pressure tensor will differ significantly from the specified values of the external stress tensor. A value of *nstep* means that every *nstep* timesteps, the reference dimensions are set to those of the current simulation domain.

The *scaleyz*, *scalexz*, and *scalexy* keywords control whether or not the corresponding tilt factors are scaled with the associated box dimensions when barostatting triclinic periodic cells. The default values *yes* will turn on scaling, which corresponds to adjusting the linear dimensions of the cell while preserving its shape. Choosing *no* ensures that the tilt factors are not scaled with the box dimensions. See below for restrictions and default values in different situations. In older versions of LAMMPS, scaling of tilt factors was not performed. The old behavior can be recovered by setting all three scale keywords to *no*.

The *flip* keyword allows the tilt factors for a triclinic box to exceed half the distance of the parallel box length, as discussed below. If the *flip* value is set to *yes*, the bound is enforced by flipping the box when it is exceeded. If the *flip* value is set to *no*, the tilt will continue to change without flipping. Note that if applied stress induces large deformations (e.g. in a liquid), this means the box shape can tilt dramatically and LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

The *fixedpoint* keyword specifies the fixed point for barostat volume changes. By default, it is the center of the box. Whatever point is chosen will not move during the simulation. For example, if the lower periodic boundaries pass through (0,0,0), and this point is provided to *fixedpoint*, then the lower periodic boundaries will remain at (0,0,0), while the upper periodic boundaries will move twice as far. In all cases, the particle trajectories are unaffected by the chosen value, except for a time-dependent constant translation of positions.

IMPORTANT NOTE: Using a barostat coupled to tilt dimensions *xy*, *xz*, *yz* can sometimes result in arbitrarily large values of the tilt dimensions, i.e. a dramatically deformed simulation box. LAMMPS allows the tilt factors to grow a small amount beyond the normal limit of half the box length (0.6 times the box length), and then performs a box "flip" to an equivalent periodic cell. See the discussion of the *flip* keyword above, to allow this bound to be exceeded, if desired.

The flip operation is described in more detail in the doc page for fix deform. Both the barostat dynamics and the atom trajectories are unaffected by this operation. However, if a tilt factor is incremented by a large amount (1.5 times the box length) on a single timestep, LAMMPS can not accomodate this event and will terminate the simulation with an error. This error typically indicates that there is something badly wrong with how the simulation was constructed, such as specifying values of *Pstart* that are too far from the current stress value, or specifying a timestep that is too large. Triclinic barostatting should be used with care. This also is true for other barostat styles, although they tend to be more forgiving of insults. In particular, it is important to recognize that equilibrium liquids can not support a shear stress and that equilibrium solids can not support shear stresses that exceed the yield stress.

One exception to this rule is if the 1st dimension in the tilt factor (x for xy) is non-periodic. In that case, the limits on the tilt factor are not enforced, since flipping the box in that dimension does not change the atom positions due to non-periodicity. In this mode, if you tilt the system to extreme angles, the simulation will simply become inefficient due to the highly skewed simulation box.

IMPORTANT NOTE: Unlike the fix temp/berendsen command which performs thermostatting but NO time integration, these fixes perform thermostatting/barostatting AND time integration. Thus you should not use any other time integration fix, such as fix nve on atoms to which this fix is applied. Likewise, the *temp* options for fix nvt and fix npt should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by fix langevin or fix temp/rescale commands.

See this howto section of the manual for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

These fixes compute a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if one of these two sets of commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp

compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press". For fix nvt, the group for the new

computes is the same as the fix group. For fix nph and fix npt, the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, fix nvt and fix npt can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

These fixes can be used with either the *verlet* or *respa* integrators. When using one of the barostat fixes with *respa*, LAMMPS uses an integrator constructed according to the following factorization of the Liouville propagator (for two rRESPA levels):

$$
\begin{aligned}
\exp\left(\mathrm{i}L\Delta t\right) \;=\; & \exp\left(\mathrm{i}L_{\text{T-baro}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\text{T-part}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\epsilon,2}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_2^{(2)}\frac{\Delta t}{2}\right) \\
& \times\left[\exp\left(\mathrm{i}L_2^{(1)}\frac{\Delta t}{2n}\right)\exp\left(\mathrm{i}L_{\epsilon,1}\frac{\Delta t}{n}\right)\exp\left(\mathrm{i}L_1\frac{\Delta t}{n}\right)\exp\left(\mathrm{i}L_2^{(1)}\frac{\Delta t}{2n}\right)\right]^n \\
& \times\exp\left(\mathrm{i}L_2^{(2)}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\epsilon,2}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\text{T-part}}\frac{\Delta t}{2}\right)\exp\left(\mathrm{i}L_{\text{T-baro}}\frac{\Delta t}{2}\right) \\
& +\mathcal{O}\left(\Delta t^3\right)
\end{aligned}
$$

This factorization differs somewhat from that of Tuckerman et al., in that the barostat is only updated at the outermost rRESPA level, whereas Tuckerman's factorization requires splitting the pressure into pieces corresponding to the forces computed at each rRESPA level. In theory, the latter method will exhibit better numerical stability. In practice, because Pdamp is normally chosen to be a large multiple of the outermost rRESPA timestep, the barostat dynamics are not the limiting factor for numerical stability. Both factorizations are time-reversible and can be shown to preserve the phase space measure of the underlying non-Hamiltonian equations of motion.

The fix npt and fix nph commands can be used with rigid bodies or mixtures of rigid bodies and non-rigid particles (e.g. solvent). But there are also fix rigid/npt and fix rigid/nph commands, which are typically a more natural choice. See the doc page for those commands for more discussion of the various ways to do this.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

These fixes writes the state of all the thermostat and barostat variables to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by these fixes. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a pressure compute defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by fix npt), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

The fix_modify *energy* option is supported by these fixes to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output.

These fixes compute a global scalar and a global vector of quantities, which can be accessed by various output commands. The scalar value calculated by these fixes is "extensive"; the vector values are "intensive".

The scalar is the cumulative energy change due to the fix.

The vector stores internal Nose/Hoover thermostat and barostat variables. The number and meaning of the vector values depends on which fix is used and the settings for keywords *tchain* and *pchain*, which specify the number of Nose/Hoover chains for the thermostat and barostat. If no thermostatting is done, then *tchain* is 0. If no barostatting is done, then *pchain* is 0. In the following list, "ndof" is 0, 1, 3, or 6, and is the number of degrees of freedom in the barostat. Its value is 0 if no barostat is used, else its value is 6 if any off-diagonal stress tensor component is barostatted, else its value is 1 if *couple xyz* is used or *couple xy* for a 2d simulation, otherwise its value is 3.

The order of values in the global vector and their meaning is as follows. The notation means there are tchain values for eta, followed by tchain for eta_dot, followed by ndof for omega, etc:

- eta[tchain] = particle thermostat displacements (unitless)
- eta_dot[tchain] = particle thermostat velocities (1/time units)
- omega[ndof] = barostat displacements (unitless)
- omega_dot[ndof] = barostat velocities (1/time units)
- etap[pchain] = barostat thermostat displacements (unitless)
- etap_dot[pchain] = barostat thermostat velocities (1/time units)

fix nph/omp command   452

- PE_eta[tchain] = potential energy of each particle thermostat displacement (energy units)
- KE_eta_dot[tchain] = kinetic energy of each particle thermostat velocity (energy units)
- PE_omega[ndof] = potential energy of each barostat displacement (energy units)
- KE_omega_dot[ndof] = kinetic energy of each barostat velocity (energy units)
- PE_etap[pchain] = potential energy of each barostat thermostat displacement (energy units)
- KE_etap_dot[pchain] = kinetic energy of each barostat thermostat velocity (energy units)
- PE_strain[1] = scalar strain energy (energy units)

These fixes can ramp their external temperature and pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

These fixes are not invoked during energy minimization.

Restrictions:

X, *y*, *z* cannot be barostatted if the associated dimension is not periodic. *Xy*, *xz*, and *yz* can only be barostatted if the simulation domain is triclinic and the 2nd dimension in the keyword (*y* dimension in *xy*) is periodic. *Z*, *xz*, and *yz*, cannot be barostatted for 2D simulations. The create_box, read data, and read_restart commands specify whether the simulation box is orthogonal or non-orthogonal (triclinic) and explain the meaning of the xy,xz,yz tilt factors.

For the *temp* keyword, the final Tstop cannot be 0.0 since it would make the external T = 0.0 at some timestep during the simulation which is not allowed in the Nose/Hoover formulation.

The *scaleyz yes* and *scalexz yes* keyword/value pairs can not be used for 2D simulations. *scaleyz yes*, *scalexz yes*, and *scalexy yes* options can only be used if the 2nd dimension in the keyword is periodic, and if the tilt factor is not coupled to the barostat via keywords *tri*, *yz*, *xz*, and *xy*.

Related commands:

fix nve, fix_modify, run_style

Default:

The keyword defaults are tchain = 3, pchain = 3, mtk = yes, tloop = ploop = 1, nreset = 0, drag = 0.0, dilate = all, couple = none, scaleyz = scalexz = scalexy = yes if periodic in 2nd dimension and not coupled to barostat, otherwise no.

(Martyna) Martyna, Tobias and Klein, J Chem Phys, 101, 4177 (1994).

(Parrinello) Parrinello and Rahman, J Appl Phys, 52, 7182 (1981).

(Tuckerman) Tuckerman, Alejandre, Lopez-Rendon, Jochim, and Martyna, J Phys A: Math Gen, 39, 5629 (2006).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

fix nph/asphere command

fix nph/asphere/omp command

Syntax:

```
fix ID group-ID nph/asphere args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nph/asphere = style name of this fix command
- additional barostat related keyword/value pairs from the [fix nph](#) command can be appended

Examples:

```
fix 1 all nph/asphere iso 0.0 0.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0
fix 2 all nph/asphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/asphere aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPH integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the [fix nph](#) command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the [fix nph](#) command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/asphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/asphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic

output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover barostat to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nph command.

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nph, fix nve_asphere, fix nvt_asphere, fix npt_asphere, fix_modify

Default: none

fix nph/sphere command

fix nph/sphere/omp command

Syntax:

```
fix ID group-ID nph/sphere args keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nph/sphere = style name of this fix command
- additional barostat related keyword/value pairs from the [fix nph](#) command can be appended

Examples:

```
fix 1 all nph/sphere iso 0.0 0.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0
fix 2 all nph/sphere x 5.0 5.0 1000.0 drag 0.2
fix 2 water nph/sphere aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPH integration to update position, velocity, and angular velocity each timestep for finite-size spherical particles in the group using a Nose/Hoover pressure barostat. P is pressure; H is enthalpy. This creates a system trajectory consistent with the isenthalpic ensemble.

This fix differs from the [fix nph](#) command, which assumes point particles and only updates their position and velocity.

Additional parameters affecting the barostat are specified by keywords and values documented with the [fix nph](#) command. See, for example, discussion of the *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPH integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/sphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/sphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic

output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover barostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *temp* and *press* options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change induced by Nose/Hoover barostatting to the system's potential energy as part of [thermodynamic output](#).

This fix computes the same global scalar and global vector of quantities as does the [fix nph](#) command.

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the [atom_style sphere](#) command.

All particles in the group must be finite-size spheres. They cannot be point particles.

Related commands:

[fix nph](#), [fix nve_sphere](#), [fix nvt_sphere](#), [fix npt_sphere](#), [fix_modify](#)

Default: none

LIGGGHTS Users Manual

- -

fix nphug command

fix nphug/omp command

Syntax:

```
fix ID group-ID nphug keyword value ...
```

- ID, group-ID are documented in command

```
  one or more keyword value pairs may be appended
  keyword = temp or iso or aniso or tri or x or y or z or couple or tchain or pchain or mtk
    temp values = Value1 Value2 Tdamp
      Value1, Value2 = Nose-Hoover target temperatures, ignored by Hugoniostat
      Tdamp = temperature damping parameter (time units)
    iso or aniso or tri values = Pstart Pstop Pdamp
      Pstart,Pstop = scalar external pressures, must be equal (pressure units)
      Pdamp = pressure damping parameter (time units)
    x or y or z or xy or yz or xz values = Pstart Pstop Pdamp
      Pstart,Pstop = external stress tensor components, must be equal (pressure units)
      Pdamp = stress damping parameter (time units)
    couple = none or xyz or xy or yz or xz
    tchain value = length of thermostat chain (1 = single thermostat)
    pchain values = length of thermostat chain on barostat (0 = no thermostat)
    mtk value = yes or no = add in MTK adjustment term or not
    tloop value = number of sub-cycles to perform on thermostat
    ploop value = number of sub-cycles to perform on barostat thermostat
    nreset value = reset reference cell every this many timesteps
    drag value = drag factor added to barostat/thermostat (0.0 = no drag)
    dilate value = all or partial
    scaleyz value = yes or no = scale yz with lz
    scalexz value = yes or no = scale xz with lz
    scalexy value = yes or no = scale xy with ly
```

Examples:

```
fix myhug all nphug temp 1.0 1.0 10.0 z 40.0 40.0 70.0
fix myhug all nphug temp 1.0 1.0 10.0 iso 40.0 40.0 70.0 drag 200.0 tchain 1 pchain 0
```

Description:

This command is a variant of the Nose-Hoover fix style. It performs time integration of the Hugoniostat equations of motion developed by Ravelo et al. . These equations compress the system to a state with average axial stress or pressure equal to the specified target value and that satisfies the Rankine-Hugoniot (RH) jump conditions for steady shocks.

The compression can be performed either hydrostatically (using keyword *iso*, *aniso*, or *tri*) or uniaxially (using keywords *x*, *y*, or *z*). In the hydrostatic case, the cell dimensions change dynamically so that the average axial stress in all three directions converges towards the specified target value. In the uniaxial case, the chosen cell dimension changes dynamically so that the average axial stress in that direction converges towards the target value. The other two cell dimensions are kept fixed (zero lateral strain).

This leads to the following additional restrictions on the keywords:

- One and only one of the following keywords should be used: *iso*, *aniso*, *tri*, *x*, *y*, *z*
- The specified initial and final target pressures must be the same.
- The keywords *xy*, *xz*, *yz* may not be used.

- The only admissible value for the couple keyword is *xyz*, which has the same effect as keyword *iso*
- The *temp* keyword must be used to specify the time constant for kinetic energy relaxation, but initial and final target temperature values are ignored.

Essentially, a Hugoniostat simulation is an NPT simulation in which the user-specified target temperature is replaced with a time-dependent target temperature Tt obtained from the following equation:

$$T_t - T = \frac{\left(\frac{1}{2}\left(P + P_0\right)\left(V_0 - V\right) + E_0 - E\right)}{N_{dof}k_B} = Delta$$

where T and Tt are the instantaneous and target temperatures, P and P0 are the instantaneous and reference pressures or axial stresses, depending on whether hydrostatic or uniaxial compression is being performed, V and V0 are the instantaneous and reference volumes, E and E0 are the instantaneous and reference internal energy (potential plus kinetic), Ndof is the number of degrees of freedom used in the definition of temperature, and kB is the Boltzmann constant. Delta is the negative deviation of the instantaneous temperature from the target temperature. When the system reaches a stable equilibrium, the value of Delta should fluctuate about zero.

The values of E0, V0, and P0 are the instantaneous values at the start of the simulation. These can be overridden using the fix_modify keywords *e0*, *v0*, and *p0* described below.

IMPORTANT NOTE: Unlike the fix temp/berendsen command which performs thermostatting but NO time integration, this fix performs thermostatting/barostatting AND time integration. Thus you should not use any other time integration fix, such as fix nve on atoms to which this fix is applied. Likewise, this fix should not be used on atoms that have their temperature controlled by another fix - e.g. by fix langevin or fix temp/rescale commands.

This fix computes a temperature and pressure at each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if one of these two sets of commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp

compute fix-ID_temp all temp
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp and compute pressure commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press". The group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

fix nphug/omp command   460

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the values of E0, V0, and P0, as well as the state of all the thermostat and barostat variables to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *e0*, *v0* and *p0* keywords can be used to define the values of E0, V0, and P0. Note the the values for *e0* and *v0* are extensive, and so must correspond to the total energy and volume of the entire system, not energy and volume per atom. If any of these quantities are not specified, then the instantaneous value in the system at the start of the simulation is used.

The fix_modify *temp* and *press* options are supported by these fixes. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure, as described above. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by these fixes to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output. Either way, this energy is *not* included in the definition of internal energy E when calculating the value of Delta in the above equation.

These fixes compute a global scalar and a global vector of quantities, which can be accessed by various output commands. The scalar value calculated by these fixes is "extensive"; the vector values are "intensive".

The scalar is the cumulative energy change due to the fix.

The vector stores three quantities unique to this fix (Delta, Us, and up), followed by all the internal Nose/Hoover thermostat and barostat variables defined for fix_style npt. Delta is the deviation of the temperature from the target temperature, given by the above equation. Us and up are the shock and particle velocity corresponding to a steady shock calculated from the RH conditions. They have units of distance/time.

Restrictions:

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

All the usual restrictions for fix_style npt apply, plus the additional ones mentioned above.

Related commands:

fix msst, fix npt, fix_modify

Default:

The keyword defaults are the same as those for [fix npt](fix npt)

(Ravelo) Ravelo, Holian, Germann and Lomdahl, Phys Rev B, 70, 014103 (2004).

fix npt/asphere command

fix npt/asphere/omp command

Syntax:

```
fix ID group-ID npt/asphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- npt/asphere = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the [fix npt](#) command can be appended

Examples:

```
fix 1 all npt/asphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/asphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/asphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal-isobaric ensemble.

This fix differs from the [fix npt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the [fix npt](#) command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/asphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/asphere
compute fix-ID_press all pressure fix-ID_temp
```

See the [compute temp/asphere](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat and barostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *temp* and *press* options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of [thermodynamic output](#).

This fix computes the same global scalar and global vector of quantities as does the [fix npt](#) command.

This fix can ramp its target temperature and pressure over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

fix npt/asphere/omp command

This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the [atom_style ellipsoid](#) command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

[fix npt](#), [fix nve_asphere](#), [fix nvt_asphere](#), [fix_modify](#)

Default: none

fix npt/sphere command

fix npt/sphere/omp command

Syntax:

```
fix ID group-ID npt/sphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- npt/sphere = style name of this fix command
- additional thermostat and barostat related keyword/value pairs from the [fix npt](#) command can be appended

Examples:

```
fix 1 all npt/sphere temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0
fix 2 all npt/sphere temp 300.0 300.0 100.0 x 5.0 5.0 1000.0 drag 0.2
fix 2 water npt/sphere temp 300.0 300.0 100.0 aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Perform constant NPT integration to update position, velocity, and angular velocity each timestep for finite-sizex spherical particles in the group using a Nose/Hoover temperature thermostat and Nose/Hoover pressure barostat. P is pressure; T is temperature. This creates a system trajectory consistent with the isothermal-isobaric ensemble.

This fix differs from the [fix npt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat and barostat are specified by keywords and values documented with the [fix npt](#) command. See, for example, discussion of the *temp*, *iso*, *aniso*, and *dilate* keywords.

The particles in the fix group are the only ones whose velocities and positions are updated by the velocity/position update portion of the NPT integration.

Regardless of what particles are in the fix group, a global pressure is computed for all particles. Similarly, when the size of the simulation box is changed, all particles are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the particles in the fix group are re-scaled. The latter can be useful for leaving the coordinates of particles in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp/sphere" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp all temp/sphere
compute fix-ID_press all pressure fix-ID_temp
```

See the compute temp/sphere and compute pressure commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is "all" since pressure is computed for the entire system.

Note that these are NOT the computes used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the compute_modify command or print this temperature or pressure during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat and barostat to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* and *press* options are supported by this fix. You can use them to assign a compute you have defined to this fix which will be used in its thermostatting or barostatting procedure. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting and barostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix npt command.

This fix can ramp its target temperature and pressure over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command.

All particles in the group must be finite-size spheres. They cannot be point particles.

Related commands:

fix npt, fix nve_sphere, fix nvt_sphere, fix npt_asphere, fix_modify

Default: none

LIGGGHTS Users Manual

fix nve/asphere command

Syntax:

```
fix ID group-ID nve/asphere
```

- ID, group-ID are documented in fix command
- nve/asphere = style name of this fix command

Examples:

```
fix 1 all nve/asphere
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for aspherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the fix nve command, which assumes point particles and only updates their position and velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nve, fix nve/sphere

Default: none

fix nve/asphere/noforce command

Syntax:

```
fix ID group-ID nve/asphere/noforce
```

- ID, group-ID are documented in fix command
- nve/asphere/noforce = style name of this fix command

Examples:

fix 1 all nve/asphere/noforce

Description:

Perform updates of position and orientation, but not velocity or angular momentum for atoms in the group each timestep. In other words, the force and torque on the atoms is ignored and their velocity and angular momentum are not updated. The atom velocities and angularm momenta are used to update their positions and orientation.

This is useful as an implicit time integrator for Fast Lubrication Dynamics, since the velocity and angular momentum are updated by the pair_style lubricuteU command.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style ellipsoid command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

fix nve/noforce, fix nve/asphere

Default: none

fix nve/body command

Syntax:

```
fix ID group-ID nve/body
```

- ID, group-ID are documented in fix command
- nve/body = style name of this fix command

Examples:

```
fix 1 all nve/body
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for body particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See Section_howto 14 of the manual and the body doc page for more details on using body particles.

This fix differs from the fix nve command, which assumes point particles and only updates their position and velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the atom_style body command.

All particles in the group must be body particles. They cannot be point particles.

Related commands:

fix nve, fix nve/sphere, fix nve/asphere

Default: none

fix nve/eff command

Syntax:

```
fix ID group-ID nve/eff
```

- ID, group-ID are documented in [fix](#) command
- nve/eff = style name of this fix command

Examples:

```
fix 1 all nve/eff
```

Description:

Perform constant NVE integration to update position and velocity for nuclei and electrons in the group for the [electron force field](#) model. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

The operation of this fix is exactly like that described by the [fix nve](#) command, except that the radius and radial velocity of electrons are also updated.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[fix nve](#), [fix nvt/eff](#), [fix npt/eff](#)

Default: none

fix nve command

fix nve/cuda command

fix nve/omp command

Syntax:

```
fix ID group-ID nve
```

- ID, group-ID are documented in <u>fix</u> command
- nve = style name of this fix command

Examples:

```
fix 1 all nve
```

Description:

Perform constant NVE integration to update position and velocity for atoms in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in <u>Section_accelerate</u> of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the <u>Making LAMMPS</u> section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the <u>-suffix command-line switch</u> when you invoke LAMMPS, or you can use the <u>suffix</u> command in your input script.

See <u>Section_accelerate</u> of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to <u>binary restart files</u>. None of the <u>fix_modify</u> options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various <u>output commands</u>. No parameter of this fix can be used with the *start/stop* keywords of the <u>run</u> command. This fix is not invoked during <u>energy minimization</u>.

Restrictions: none

Related commands:

fix nvt, fix npt

Default: none

fix nve/limit command

Syntax:

```
fix ID group-ID nve/limit limitstyle xmax
```

- ID, group-ID are documented in <u>fix</u> command
- nve/limit = style name of this fix command
- limitstyle = absolute or radius_ratio
- xmax = maximum distance an atom can move in one timestep (distance units or relative to atom radius)

Examples:

```
fix 1 all nve/limit absolute 0.1
```

Description:

Perform constant NVE updates of position and velocity for atoms in the group each timestep. A limit is imposed on the maximum distance an atom can move in one timestep. This is useful when starting a simulation with a configuration containing highly overlapped atoms. Normally this would generate huge forces which would blow atoms out of the simulation box, causing LAMMPS to stop with an error.

Using this fix can overcome that problem. Forces on atoms must still be computable (which typically means 2 atoms must have a separation distance > 0.0). But large velocities generated by large forces are reset to a value that corresponds to a displacement of length *xmax* in a single timestep. *Xmax* is specified in distance units; see the <u>units</u> command for details. The value of *xmax* should be consistent with the neighbor skin distance and the frequency of neighbor list re-building, so that pairwise interactions are not missed on successive timesteps as atoms move. See the <u>neighbor</u> and <u>neigh_modify</u> commands for details.

If limitstyle *absolute* is used, *xmax* is applied directly. If limitstyle *radius_ratio* is used, a maxmimum distance per step of *xmax**radius is applied for each atom. This can be useful for the simulation of poly-disperse systems. Note that this option requires the atom radius to be stored by using an appropriate atom style.

Note that if a velocity reset occurs the integrator will not conserve energy. On steps where no velocity resets occur, this integrator is exactly like the <u>fix nve</u> command. Since forces are unaltered, pressures computed by thermodynamic output will still be very large for overlapped configurations.

IMPORTANT NOTE: You should not use <u>fix shake</u> in conjunction with this fix. That is because fix shake applies contraint forces based on the predicted postitions of atoms after the next timestep. It has no way of knowing the timestep may change due to this fix, which will cause the constraint forces to be invalid. A better strategy is to turn off fix shake when performing initial dynamics that need this fix, then turn fix shake on when doing normal dynamics with a fixed-size timestep.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to <u>binary restart files</u>. None of the <u>fix_modify</u> options are relevant to this fix.

This fix computes a global scalar which can be accessed by various <u>output commands</u>. The scalar is the count of how many updates of atom's velocity/position were limited by the maximum distance criterion. This should

fix nve/omp command   474

be roughly the number of atoms so affected, except that updates occur at both the beginning and end of a timestep in a velocity Verlet timestepping algorithm. This is a cumulative quantity for the current run, but is re-initialized to zero each time a run is performed. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix nve, fix nve/noforce, pair_style soft

Default: none

fix nve/line command

Syntax:

```
fix ID group-ID nve/line
```

- ID, group-ID are documented in [fix](#) command
- nve/line = style name of this fix command

Examples:

```
fix 1 all nve/line
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular velocity for line segment particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See [Section_howto 14](#) of the manual for an overview of using line segment particles.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix requires that particles be line segments as defined by the [atom_style line](#) command.

Related commands:

[fix nve](#), [fix nve/asphere](#)

Default: none

fix nve/noforce command

Syntax:

```
fix ID group-ID nve
```

- ID, group-ID are documented in [fix](#) command
- nve/noforce = style name of this fix command

Examples:

```
fix 3 wall nve/noforce
```

Description:

Perform updates of position, but not velocity for atoms in the group each timestep. In other words, the force on the atoms is ignored and their velocity is not updated. The atom velocities are used to update their positions.

This can be useful for wall atoms, when you set their velocities, and want the wall to move (or stay stationary) in a prescribed fashion.

This can also be accomplished via the [fix setforce](#) command, but with fix nve/noforce, the forces on the wall atoms are unchanged, and can thus be printed by the [dump](#) command or queried with an equal-style [variable](#) that uses the fcm() group function to compute the total force on the group of atoms.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix nve](#)

Default: none

fix nve/sphere command

fix nve/sphere/omp command

Syntax:

```
fix ID group-ID nve/sphere
```

- ID, group-ID are documented in [fix](#) command
- nve/sphere = style name of this fix command
- zero or more keyword/value pairs may be appended
- keyword = *update*

```
  update value = dipole
      dipole = update orientation of dipole moment during integration
```

Examples:

```
fix 1 all nve/sphere
fix 1 all nve/sphere update dipole
```

Description:

Perform constant NVE integration to update position, velocity, and angular velocity for finite-size spherical particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

If the *update* keyword is used with the *dipole* value, then the orientation of the dipole moment of each particle is also updated during the time integration. This option should be used for models where a dipole moment is assigned to particles via use of the [atom_style dipole](#) command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to

this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the atom_style sphere command. If the *dipole* keyword is used, then they must also store a dipole moment as defined by the atom_style dipole command.

All particles in the group must be finite-size spheres. They cannot be point particles.

Related commands:

fix nve, fix nve/asphere

Default: none

fix nve/tri command

Syntax:

```
fix ID group-ID nve/tri
```

- ID, group-ID are documented in [fix](#) command
- nve/tri = style name of this fix command

Examples:

```
fix 1 all nve/tri
```

Description:

Perform constant NVE integration to update position, velocity, orientation, and angular momentum for triangular particles in the group each timestep. V is volume; E is energy. This creates a system trajectory consistent with the microcanonical ensemble. See [Section_howto 14](#) of the manual for an overview of using triangular particles.

This fix differs from the [fix nve](#) command, which assumes point particles and only updates their position and velocity.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix requires that particles be triangles as defined by the [atom_style tri](#) command.

Related commands:

[fix nve](#), [fix nve/asphere](#)

Default: none

fix nvt/asphere command

fix nvt/asphere/omp command

Syntax:

```
fix ID group-ID nvt/asphere keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/asphere = style name of this fix command
- additional thermostat related keyword/value pairs from the [fix nvt](#) command can be appended

Examples:

```
fix 1 all nvt/asphere temp 300.0 300.0 100.0
fix 1 all nvt/asphere temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update position, velocity, orientation, and angular velocity each timestep for aspherical or ellipsoidal particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the [fix nvt](#) command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the aspherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/asphere", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/asphere
```

See the [compute temp/asphere](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for

individual [compute commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *temp* option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostatting procedure.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of [thermodynamic output](#).

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix requires that atoms store torque and angular momementum and a quaternion as defined by the [atom_style ellipsoid](#) command.

All particles in the group must be finite-size. They cannot be point particles, but they can be aspherical or spherical as defined by their shape attribute.

Related commands:

[fix nvt](#), [fix nve_asphere](#), [fix npt_asphere](#), [fix_modify](#)

Default: none

fix nvt/sllod/eff command

Syntax:

```
fix ID group-ID nvt/sllod/eff keyword value ...
```

- ID, group-ID are documented in fix command
- nvt/sllod/eff = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt/eff command can be appended

Examples:

```
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1
fix 1 all nvt/sllod/eff temp 300.0 300.0 0.1 drag 0.2
```

Description:

Perform constant NVT integration to update positions and velocities each timestep for nuclei and electrons in the group for the electron force field model, using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

The operation of this fix is exactly like that described by the fix nvt/sllod command, except that the radius and radial velocity of electrons are also updated and thermostatted. Likewise the temperature calculated by the fix, using the compute it creates (as discussed in the fix nvt, npt, and nph doc page), is performed with a compute temp/deform/eff commmand that includes the eFF contribution to the temperature from the electron radial velocity.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt/eff command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix works best without Nose-Hoover chain thermostats, i.e. using tchain = 1. Setting tchain to larger values can result in poor equilibration.

fix nvt/sllod/eff command

Related commands:

fix nve/eff, fix nvt/eff, fix langevin/eff, fix nvt/sllod, fix_modify, compute temp/deform/eff

Default:

Same as fix nvt/eff, except tchain = 1.

(**Tuckerman**) Tuckerman, Mundy, Balasubramanian, Klein, J Chem Phys, 106, 5615 (1997).

fix nvt/sllod command

fix nvt/sllod/omp command

Syntax:

```
fix ID group-ID nvt/sllod keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- nvt/sllod = style name of this fix command
- additional thermostat related keyword/value pairs from the [fix nvt](#) command can be appended

Examples:

```
fix 1 all nvt/sllod temp 300.0 300.0 100.0
fix 1 all nvt/sllod temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update positions and velocities each timestep for atoms in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This thermostat is used for a simulation box that is changing size and/or shape, for example in a non-equilibrium MD (NEMD) simulation. The size/shape change is induced by use of the [fix deform](#) command, so each point in the simulation box can be thought of as having a "streaming" velocity. This position-dependent streaming velocity is subtracted from each atom's actual velocity to yield a thermal velocity which is used for temperature computation and thermostatting. For example, if the box is being sheared in x, relative to y, then points at the bottom of the box (low y) have a small x velocity, while points at the top of the box (hi y) have a large x velocity. These velocities do not contribute to the thermal "temperature" of the atom.

IMPORTANT NOTE: [Fix deform](#) has an option for remapping either atom coordinates or velocities to the changing simulation box. To use fix nvt/sllod, fix deform should NOT remap atom positions, because fix nvt/sllod adjusts the atom positions and velocities to create a velocity profile that matches the changing box size/shape. Fix deform SHOULD remap atom velocities when atoms cross periodic boundaries since that is consistent with maintaining the velocity profile created by fix nvt/sllod. LAMMPS will give an error if this setting is not consistent.

The SLLOD equations of motion coupled to a Nose/Hoover thermostat are discussed in [(Tuckerman)](#) (eqs 4 and 5), which is what is implemented in LAMMPS in a velocity Verlet formulation.

Additional parameters affecting the thermostat are specified by keywords and values documented with the [fix nvt](#) command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/deform", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/deform
```

See the [compute temp/deform](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual compute commands to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix_modify *temp* option is supported by this fix. You can use it to assign a compute you have defined to this fix which will be used in its thermostatting procedure.

The fix_modify *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of thermodynamic output.

This fix computes the same global scalar and global vector of quantities as does the fix nvt command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix works best without Nose-Hoover chain thermostats, i.e. using tchain = 1. Setting tchain to larger values can result in poor equilibration.

fix nvt/sllod/omp command   487

Related commands:

fix nve, fix nvt, fix temp/rescale, fix langevin, fix_modify, compute temp/deform

Default:

Same as fix nvt, except tchain = 1.

(Tuckerman) Tuckerman, Mundy, Balasubramanian, Klein, J Chem Phys, 106, 5615 (1997).

fix nvt/sphere command

fix nvt/sphere/omp command

Syntax:

```
fix ID group-ID nvt/sphere keyword value ...
```

- ID, group-ID are documented in fix command
- nvt/sphere = style name of this fix command
- additional thermostat related keyword/value pairs from the fix nvt command can be appended

Examples:

```
fix 1 all nvt/sphere temp 300.0 300.0 100.0
fix 1 all nvt/sphere temp 300.0 300.0 100.0 drag 0.2
```

Description:

Perform constant NVT integration to update position, velocity, and angular velocity each timestep for finite-size spherical particles in the group using a Nose/Hoover temperature thermostat. V is volume; T is temperature. This creates a system trajectory consistent with the canonical ensemble.

This fix differs from the fix nvt command, which assumes point particles and only updates their position and velocity.

The thermostat is applied to both the translational and rotational degrees of freedom for the spherical particles, assuming a compute is used which calculates a temperature that includes the rotational degrees of freedom (see below). The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Additional parameters affecting the thermostat are specified by keywords and values documented with the fix nvt command. See, for example, discussion of the *temp* and *drag* keywords.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp/sphere", as if this command had been issued:

```
compute fix-ID_temp group-ID temp/sphere
```

See the compute temp/sphere command for details. Note that the ID of the new compute is the fix-ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the thermo_style command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the compute_modify command or print this temperature during thermodynamic output via the thermo_style custom command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with compute commands that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the fix_modify command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for

individual [compute commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the Nose/Hoover thermostat to [binary restart files](#). See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *temp* option is supported by this fix. You can use it to assign a [compute](#) you have defined to this fix which will be used in its thermostatting procedure.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change induced by Nose/Hoover thermostatting to the system's potential energy as part of [thermodynamic output](#).

This fix computes the same global scalar and global vector of quantities as does the [fix nvt](#) command.

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

This fix requires that atoms store torque and angular velocity (omega) and a radius as defined by the [atom_style sphere](#) command.

All particles in the group must be finite-size spheres. They cannot be point particles.

Related commands:

[fix nvt](#), [fix nve_sphere](#), [fix nvt_asphere](#), [fix npt_sphere](#), [fix_modify](#)

Default: none

fix orient/fcc command

```
fix ID group-ID orient/fcc nstats dir alat dE cutlo cuthi file0 file1
```

- ID, group-ID are documented in [fix](#) command
- nstats = print stats every this many steps, 0 = never
- dir = 0/1 for which crystal is used as reference
- alat = fcc cubic lattice constant (distance units)
- dE = energy added to each atom (energy units)
- cutlo,cuthi = values between 0.0 and 1.0, cutlo < cuthi
- file0,file1 = files that specify orientation of each grain

Examples:

```
fix gb all orient/fcc 0 1 4.032008 0.001 0.25 0.75 xi.vec chi.vec
```

Description:

The fix applies an orientation-dependent force to atoms near a planar grain boundary which can be used to induce grain boundary migration (in the direction perpendicular to the grain boundary plane). The motivation and explanation of this force and its application are described in [(Janssens)](#). The force is only applied to atoms in the fix group.

The basic idea is that atoms in one grain (on one side of the boundary) have a potential energy dE added to them. Atoms in the other grain have 0.0 potential energy added. Atoms near the boundary (whose neighbor environment is intermediate between the two grain orientations) have an energy between 0.0 and dE added. This creates an effective driving force to reduce the potential energy of atoms near the boundary by pushing them towards one of the grain orientations. For dir = 1 and dE > 0, the boundary will thus move so that the grain described by file0 grows and the grain described by file1 shrinks. Thus this fix is designed for simulations of two-grain systems, either with one grain boundary and free surfaces parallel to the boundary, or a system with periodic boundary conditions and two equal and opposite grain boundaries. In either case, the entire system can displace during the simulation, and such motion should be accounted for in measuring the grain boundary velocity.

The potential energy added to atom I is given by these formulas

$$\xi_i = \sum_{j=1}^{12} \left| \mathbf{r}_j - \mathbf{r}_j^{\mathrm{I}} \right| \tag{1}$$

$$\xi_{\mathrm{IJ}} = \sum_{j=1}^{12} \left| \mathbf{r}_j^{\mathrm{J}} - \mathbf{r}_j^{\mathrm{I}} \right| \tag{2}$$

$$\xi_{\mathrm{low}} = \mathrm{cutlo}\,\xi_{\mathrm{IJ}} \tag{3}$$
$$\xi_{\mathrm{high}} = \mathrm{cuthi}\,\xi_{\mathrm{IJ}} \tag{4}$$

$$\omega_i = \frac{\pi}{2}\frac{\xi_i - \xi_{\mathrm{low}}}{\xi_{\mathrm{high}} - \xi_{\mathrm{low}}} \tag{5}$$

$$
\begin{aligned}
u_i &= 0 & \text{for } \xi_i < \xi_{\mathrm{low}} \\
&= \mathrm{dE}\,\frac{1 - \cos(2\omega_i)}{2} & \text{for } \xi_{\mathrm{low}} < \xi_i < \xi_{\mathrm{high}} \\
&= \mathrm{dE} & \text{for } \xi_{\mathrm{high}} < \xi_i
\end{aligned}
\tag{6}
$$

which are fully explained in (Janssens). The order parameter Xi for atom I in equation (1) is a sum over the 12 nearest neighbors of atom I. Rj is the vector from atom I to its neighbor J, and RIj is a vector in the reference (perfect) crystal. That is, if dir = 0/1, then RIj is a vector to an atom coord from file 0/1. Equation (2) gives the expected value of the order parameter XiIJ in the other grain. Hi and lo cutoffs are defined in equations (3) and (4), using the input parameters *cutlo* and *cuthi* as thresholds to avoid adding grain boundary energy when the deviation in the order parameter from 0 or 1 is small (e.g. due to thermal fluctuations in a perfect crystal). The added potential energy Ui for atom I is given in equation (6) where it is interpolated between 0 and dE using the two threshold Xi values and the Wi value of equation (5).

The derivative of this energy expression gives the force on each atom which thus depends on the orientation of its neighbors relative to the 2 grain orientations. Only atoms near the grain boundary feel a net force which tends to drive them to one of the two grain orientations.

In equation (1), the reference vector used for each neighbor is the reference vector closest to the actual neighbor position. This means it is possible two different neighbors will use the same reference vector. In such cases, the atom in question is far from a perfect orientation and will likely receive the full dE addition, so the effect of duplicate reference vector usage is small.

The *dir* parameter determines which grain wants to grow at the expense of the other. A value of 0 means the first grain will shrink; a value of 1 means it will grow. This assumes that *dE* is positive. The reverse will be true if *dE* is negative.

fix orient/fcc command

The *alat* parameter is the cubic lattice constant for the fcc material and is only used to compute a cutoff distance of 1.57 * alat / sqrt(2) for finding the 12 nearest neighbors of each atom (which should be valid for an fcc crystal). A longer/shorter cutoff can be imposed by adjusting *alat*. If a particular atom has less than 12 neighbors within the cutoff, the order parameter of equation (1) is effectively multiplied by 12 divided by the actual number of neighbors within the cutoff.

The *dE* parameter is the maximum amount of additional energy added to each atom in the grain which wants to shrink.

The *cutlo* and *cuthi* parameters are used to reduce the force added to bulk atoms in each grain far away from the boundary. An atom in the bulk surrounded by neighbors at the ideal grain orientation would compute an order parameter of 0 or 1 and have no force added. However, thermal vibrations in the solid will cause the order parameters to be greater than 0 or less than 1. The cutoff parameters mask this effect, allowing forces to only be added to atoms with order-parameters between the cutoff values.

File0 and *file1* are filenames for the two grains which each contain 6 vectors (6 lines with 3 values per line) which specify the grain orientations. Each vector is a displacement from a central atom (0,0,0) to a nearest neighbor atom in an fcc lattice at the proper orientation. The vector lengths should all be identical since an fcc lattice has a coordination number of 12. Only 6 are listed due to symmetry, so the list must include one from each pair of equal-and-opposite neighbors. A pair of orientation files for a Sigma=5 tilt boundary are show below.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *energy* option is supported by this fix to add the potential energy of atom interactions with the grain boundary driving force to the system's potential energy as part of [thermodynamic output](#).

This fix calculates a global scalar which can be accessed by various [output commands](#). The scalar is the potential energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix also calculates a per-atom array which can be accessed by various [output commands](#). The array stores the order parameter Xi and normalized order parameter (0 to 1) for each atom. The per-atom values can be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix should only be used with fcc lattices.

Related commands:

[fix_modify](#)

Default: none

(Janssens) Janssens, Olmsted, Holm, Foiles, Plimpton, Derlet, Nature Materials, 5, 124-127 (2006).

For illustration purposes, here are example files that specify a Sigma=5 tilt boundary. This is for a lattice constant of 3.5706 Angs.

file0:

```
 0.798410432046075    1.785300000000000    1.596820864092150
-0.798410432046075    1.785300000000000   -1.596820864092150
 2.395231296138225    0.000000000000000    0.798410432046075
 0.798410432046075    0.000000000000000   -2.395231296138225
 1.596820864092150    1.785300000000000   -0.798410432046075
 1.596820864092150   -1.785300000000000   -0.798410432046075
```

file1:

```
-0.798410432046075    1.785300000000000    1.596820864092150
 0.798410432046075    1.785300000000000   -1.596820864092150
 0.798410432046075    0.000000000000000    2.395231296138225
 2.395231296138225    0.000000000000000   -0.798410432046075
 1.596820864092150    1.785300000000000    0.798410432046075
 1.596820864092150   -1.785300000000000    0.798410432046075
```

fix particledistribution/discrete command

Syntax:

```
fix ID group-ID particledistribution/discrete seed ntemp t_id t_m ...
```

- ID, group-ID are documented in [fix](#) command
- particledistribution/discrete = style name of this fix command
- seed = random number generator seed (integer value)
- ntemp = number of particle templates to be used in this command
- zero or more *t_id*/*t_m* pairs are appended, number of pairs must match ntemp

  ```
  t_id = id of a fix of type particletemplate/sphere
  t_m = mass % for this template in the distribution
  ```

Examples:

```
fix pdd1 all particledistribution/discrete 6778  1 pts1 1.0
fix pdd1 all particledistribution/discrete 1239  2 pts1 0.3 pts2 0.7
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

Define a discrete particle distribution that defines a discrete particle distribution to be inserted by the [fix_pour_dev](#) command. It takes several templates of type [fix_particletemplate_sphere](#) as inputs, which define the properties of the single particles that are part of the distribution.

At insertion, particles are chosen according to the mass-% distribution as defined by each pair (t_id, t_m). Note that the sum of all weights t_m must be equal to 1.0, if this is not the case the user is warned at the distribution is normalized. Note that large particles are inserted first, so that a higher volume fraction can be achieved. If not all desired insertions could be performed, it is likely that the distribution is not accurately reproduced.

IMPORTANT NOTE: As opposed to the number-based distributions used by [fix_pour](#) and [fix_pour_legacy](#), this fix uses the more common distribution based on mass-% as input (as does [fix_particletemplate_sphere](#)).

Restart, fix_modify, output, run start/stop, minimize info:

Information about the random state in this fix is written to [binary restart files](#) so you can restart a simulation with the same particles being chosen for insertion. None of the [fix_modify](#) options are relevant to this fix. No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start*/*stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix_pour_dev](#)

Default: none

fix particletemplate/sphere command

Syntax:

```
fix ID group-ID particletemplate/sphere seed keyword values ...
```

- ID, group-ID are documented in fix command
- particletemplate/sphere = style name of this fix command
- seed = random number generator seed (integer value)
- n_spheres = number of spheres in the template
- zero or more keyword/value pairs can be appended
- keyword = *atom_type* or *density* or *volume_limit* or *radius*

```
  atom_type value = atom type assigned to this particle template
  density values = random_style param1 (param2)
    random_style = 'constant' or 'uniform' or 'gaussian'
    param1 = density for 'constant', low value of density for 'uniform', expectancy value fo
      param2 = omitted for 'constant', high value of density for 'uniform', sigma value for
  volume_limit value = lowest particle volume allowed in simulation
  radius values = random_style param1 (param2)
    random_style = 'constant' or 'uniform number' or 'uniform mass' or 'gaussian number'
    param1 = radius for 'constant', low value of radius for 'uniform', mu value for 'gaussia
      param2 = omitted for 'constant', high value of radius for 'uniform', sigma value for 'ga
```

Examples:

```
fix pts1 all particletemplate/sphere 1 atom_type 1 density constant 2500 radius constant 0.0015
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command is not available in LAMMPS.

Description:

Define a particle that is used as input for a fix_particledistribution_discrete command. You can choose the atom type, density and radius of the particles. For density and radius, you can choose between 'constant', 'uniform' and 'gaussian' random styles. Note that for radius, you can additionally choose between a number-based and mass-based uniform distribution, where the latter is used more frequently typically. 'gaussian' for radius only supports a number-based distribution.

It is thus possible to define a uniform or gaussian distribution on top of the discrete distribution defined by fix_particledistribution_discrete.

IMPORTANT NOTE: As opposed to the number-based distributions used by fix_pour and fix_pour_legacy, this fix uses the more common distribution based on mass-% for the radius distribution (as does fix_particledistribution_discrete).

LIGGGHTS will throw an error if the particle volume is too small compared to machine precision. If you are sure you know what you are doing you can override the default limit of 1e-12.

Restart, fix_modify, output, run start/stop, minimize info:

Information about the random state in this fix is written to binary restart files so you can restart a simulation with the same particles being chosen for insertion. None of the fix_modify options are relevant to this fix. No

global scalar or vector or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start*/*stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

fix_particletemplate_sphere

Default: radius = 1.0, density = 1.0, atom_type = 1, volume_limit = 1e-12

fix phonon command

Syntax:

```
fix ID group-ID phonon N Noutput Nwait map_file prefix keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- phonon = style name of this fix command
- N = measure the Green's function every this many timesteps
- Noutput = output the dynamical matrix every this many measurements
- Nwait = wait this many timesteps before measuring
- map_file = *file* or *GAMMA*

  file is the file that contains the mapping info between atom ID and the lattice indices.

  GAMMA flags to treat the whole simulation box as a unit cell, so that the mapping
    info can be generated internally. In this case, dynamical matrix at only the gamma-point
    will/can be evaluated.
- prefix = prefix for output files
- one or none keyword/value pairs may be appended
- keyword = *sysdim* or *nasr*

```
sysdim value = d
    d = dimension of the system, usually the same as the MD model dimension
  nasr value = n
    n = number of iterations to enforce the acoustic sum rule
```

Examples:

```
fix 1 all phonon 20 5000 200000 map.in LJ1D sysdim 1
fix 1 all phonon 20 5000 200000 map.in EAM3D
fix 1 all phonon 10 5000 500000 GAMMA EAM0D nasr 100
```

Description:

Calculate the dynamical matrix from molecular dynamics simulations based on fluctuation-dissipation theory for a group of atoms.

Consider a crystal with N unit cells in three dimensions labelled $l = (l_1, l_2, l_3)$ where l_i are integers. Each unit cell is defined by three linearly independent vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$ forming a parallelipiped, containing K basis atoms labelled k.

Based on fluctuation-dissipation theory, the force constant coefficients of the system in reciprocal space are given by ([Campañá](#) , [Kong](#))

$$_{k,k'}(\mathbf{q}) = k_B T\, \mathbf{G}^{-1}_{k,k'}(\mathbf{q}),$$

where $\mathbf{G}$ is the Green's functions coefficients given by

$$\mathbf{G}_{k,k'}(\mathbf{q}) = <\mathbf{u}_k(\mathbf{q})\, \mathbf{u}_{k'}{}^{*}(\mathbf{q})>,$$

where denotes the ensemble average, and

$$\mathbf{u}_k(\mathbf{q}) = \sum_l \mathbf{u}_{lk}\, \exp(i\mathbf{q}\mathbf{r}_l)$$

fix phonon command 499

is the   component of the atomic displacement for the *k*th atom in the unit cell in reciprocal space at **q**. In practice, the Green's functions coefficients can also be measured according to the following formula,

$$\mathbf{G}_{k,k'}(\mathbf{q}) = <\mathbf{R}_k(\mathbf{q}) \ \mathbf{R}^*_{k'}(\mathbf{q})> - <\mathbf{R}>_k(\mathbf{q}) \ <\mathbf{R}>^*_{k'}(\mathbf{q}),$$

where **R** is the instantaneous positions of atoms, and **<R>** is the averaged atomic positions. It gives essentially the same results as the displacement method and is easier to implement in an MD code.

Once the force constant matrix is known, the dynamical matrix **D** can then be obtained by

$$\mathbf{D}_{k,k'}(\mathbf{q}) = (m_k m_{k'})^{-1/2} \ _{k,k'}(\mathbf{q})$$

whose eigenvalues are exactly the phonon frequencies at **q**.

This fix uses positions of atoms in the specified group and calculates two-point correlations. To achieve this. the positions of the atoms are examined every *Nevery* steps and are Fourier-transformed into reciprocal space, where the averaging process and correlation computation is then done. After every *Noutput* measurements, the matrix **G(q)** is calculated and inverted to obtain the elastic stiffness coefficients. The dynamical matrices are then constructed and written to *prefix*.bin.timestep files in binary format and to the file *prefix*.log for each wavevector **q**.

A detailed description of this method can be found in ([Kong2011](#)).

The *sysdim* keyword is optional. If specified with a value smaller than the dimensionality of the LAMMPS simulation, its value is used for the dynamical matrix calculation. For example, using LAMMPS ot model a 2D or 3D system, the phonon dispersion of a 1D atomic chain can be computed using *sysdim* = 1.

The *nasr* keyword is optional. An iterative procedure is employed to enforce the acoustic sum rule on   at   , and the number provided by keyword *nasr* gives the total number of iterations. For a system whose unit cell has only one atom, *nasr* = 1 is sufficient; for other systems, *nasr* = 10 is typically sufficient.

The *map_file* contains the mapping information between the lattice indices and the atom IDs, which tells the code which atom sits at which lattice point; the lattice indices start from 0. An auxiliary code, [latgen](#), can be employed to generate the compatible map file for various crystals.

In case one simulates an aperiodic system, where the whole simulation box is treated as a unit cell, one can set *map_file* as *GAMMA*, so that the mapping info will be generated internally and a file is not needed. In this case, the dynamical matrix at only the gamma-point will/can be evaluated. Please keep in mind that fix-phonon is designed for cyrstals, it will be inefficient and even degrade the performance of lammps in case the unit cell is too large.

The calculated dynamical matrix elements are written out in [energy/distance^2/mass](#) units. The coordinates for *q* points in the log file is in the units of the basis vectors of the corresponding reciprocal lattice.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *temp* option is supported by this fix. You can use it to change the temperature compute from thermo_temp to the one that reflects the true temperature of atoms in the group.

No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#).

Instead, this fix outputs its initialization information (including mapping information) and the calculated

dynamical matrices to the file *prefix*.log, with the specified *prefix*. The dynamical matrices are also written to files *prefix*.bin.timestep in binary format. These can be read by the post-processing tool in tools/phonon to compute the phonon density of states and/or phonon dispersion curves.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is not invoked during energy minimization.

Restrictions:

This fix assumes a crystalline system with periodical lattice. The temperature of the system should not exceed the melting temperature to keep the system in its solid state.

This fix is part of the USER-PHONON package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix requires LAMMPS be built with an FFT library. See the Making LAMMPS section for more info.

Related commands:

compute msd

Default:

The option defaults are sysdim = the same dimemsion as specified by the dimension command, and nasr = 20.

(Campañá) C. Campañá and M. H. Müser, *Practical Green's function approach to the simulation of elastic semi-infinite solids*, Phys. Rev. B **74**, 075420 (2006)

(Kong) L.T. Kong, G. Bartels, C. Campañá, C. Denniston, and Martin H. Müser, *Implementation of Green's function molecular dynamics: An extension to LAMMPS*, Computer Physics Communications **180**(6):1004-1010 (2009).

L.T. Kong, C. Denniston, and Martin H. Müser, *An improved version of the Green's function molecular dynamics method*, Computer Physics Communications **182**(2):540-541 (2011).

(Kong2011) L.T. Kong, *Phonon dispersion measured directly from molecular dynamics simulations*, Computer Physics Communications **182**(10):2201-2207, (2011).

fix planeforce command

Syntax:

```
fix ID group-ID planeforce x y z
```

- ID, group-ID are documented in <u>fix</u> command
- lineforce = style name of this fix command
- x y z = 3-vector that is normal to the plane

Examples:

```
fix hold boundary planeforce 1.0 0.0 0.0
```

Description:

Adjust the forces on each atom in the group so that only the components of force in the plane specified by the normal vector (x,y,z) remain. This is done by subtracting out the component of force perpendicular to the plane.

If the initial velocity of the atom is 0.0 (or in the plane), then it should continue to move in the plane thereafter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to <u>binary restart files</u>. None of the <u>fix_modify</u> options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various <u>output commands</u>. No parameter of this fix can be used with the *start/stop* keywords of the <u>run</u> command.

The forces due to this fix are imposed during an energy minimization, invoked by the <u>minimize</u> command.

Restrictions: none

Related commands:

<u>fix lineforce</u>

Default: none

fix poems

Syntax:

```
fix ID group-ID poems keyword values
```

- ID, group-ID are documented in [fix](#) command
- poems = style name of this fix command
- keyword = *group* or *file* or *molecule*

```
  group values = list of group IDs
    molecule values = none
     file values = filename
```

Examples:

```
fix 3 fluid poems group clump1 clump2 clump3
fix 3 fluid poems file cluster.list
```

Description:

Treats one or more sets of atoms as coupled rigid bodies. This means that each timestep the total force and torque on each rigid body is computed and the coordinates and velocities of the atoms are updated so that the collection of bodies move as a coupled set. This can be useful for treating a large biomolecule as a collection of connected, coarse-grained particles.

The coupling, associated motion constraints, and time integration is performed by the software package [Parallelizable Open source Efficient Multibody Software (POEMS)](#) which computes the constrained rigid-body motion of articulated (jointed) multibody systems [(Anderson)](#). POEMS was written and is distributed by Prof Kurt Anderson, his graduate student Rudranarayan Mukherjee, and other members of his group at Rensselaer Polytechnic Institute (RPI). Rudranarayan developed the LAMMPS/POEMS interface. For copyright information on POEMS and other details, please refer to the documents in the poems directory distributed with LAMMPS.

This fix updates the positions and velocities of the rigid atoms with a constant-energy time integration, so you should not update the same atoms via other fixes (e.g. nve, nvt, npt, temp/rescale, langevin).

Each body must have a non-degenerate inertia tensor, which means if must contain at least 3 non-collinear atoms. Which atoms are in which bodies can be defined via several options.

For option *group*, each of the listed groups is treated as a rigid body. Note that only atoms that are also in the fix group are included in each rigid body.

For option *molecule*, each set of atoms in the group with a different molecule ID is treated as a rigid body.

For option *file*, sets of atoms are read from the specified file and each set is treated as a rigid body. Each line of the file specifies a rigid body in the following format:

ID type atom1-ID atom2-ID atom3-ID ...

ID as an integer from 1 to M (the number of rigid bodies). Type is any integer; it is not used by the fix poems command. The remaining arguments are IDs of atoms in the rigid body, each typically from 1 to N (the number of atoms in the system). Only atoms that are also in the fix group are included in each rigid body.

Blank lines and lines that begin with '#' are skipped.

A connection between a pair of rigid bodies is inferred if one atom is common to both bodies. The POEMS solver treats that atom as a spherical joint with 3 degrees of freedom. Currently, a collection of bodies can only be connected by joints as a linear chain. The entire collection of rigid bodies can represent one or more chains. Other connection topologies (tree, ring) are not allowed, but will be added later. Note that if no joints exist, it is more efficient to use the fix rigid command to simulate the system.

When the poems fix is defined, it will print out statistics on the total # of clusters, bodies, joints, atoms involved. A cluster in this context means a set of rigid bodies connected by joints.

For computational efficiency, you should turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The "neigh_modify exclude" and "delete_bonds" commands can be used to do this if each rigid body is a group.

For computational efficiency, you should only define one fix poems which includes all the desired rigid bodies. LAMMPS will allow multiple poems fixes to be defined, but it is more expensive.

The degrees-of-freedom removed by coupled rigid bodies are accounted for in temperature and pressure computations. Similarly, the rigid body contribution to the pressure virial is also accounted for. The latter is only correct if forces within the bodies have been turned off, and there is only a single fix poems defined.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the POEMS package. It is only enabled if LAMMPS was built with that package, which also requires the POEMS library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

fix rigid, delete_bonds, neigh_modify exclude

Default: none

(Anderson) Anderson, Mukherjee, Critchley, Ziegler, and Lipton "POEMS: Parallelizable Open-source Efficient Multibody Software ", Engineering With Computers (2006). (link to paper)

LIGGGHTS Users Manual

fix pour/legacy command

Syntax:

```
fix ID group-ID pour/legacy N type seed keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- pour/legacy = style name of this fix command
- N = # of atoms to insert
- type = atom type to assign to inserted atoms
- seed = random # seed (positive integer)
- one or more keyword/value pairs may be appended to args
- keyword = *region* or *diam* or *dens* or *vol* or *rate* or *vel*

```
  region value = region-ID
     region-ID = ID of region to use as insertion volume
   diam values = lo hi
     lo,hi = range of diameters for inserted particles (distance units)
   dens values = lo hi
     lo,hi = range of densities for inserted particles
   vol values = fraction Nattempt
     fraction = desired volume fraction for filling insertion volume
     Nattempt = max # of insertion attempts per atom
   rate value = V
     V = z velocity (3d) or y velocity (2d) at which
         insertion volume moves (velocity units)
   vel values (3d) = vxlo vxhi vylo vyhi vz
   vel values (2d) = vxlo vxhi vy
     vxlo,vxhi = range of x velocities for inserted particles (velocity units)
     vylo,vyhi = range of y velocities for inserted particles (velocity units)
     vz = z velocity (3d) assigned to inserted particles (velocity units)
     vy = y velocity (2d) assigned to inserted particles (velocity units)
```

Examples:

```
fix 3 all pour/legacy 1000 2 29494 region myblock
fix 2 all pour/legacy 10000 1 19985583 region disk vol 0.33 100 rate 1.0 diam 0.9 1.1
```

Description:

Insert particles into a granular run every few timesteps within a specified region until N particles have been inserted. This is useful for simulating the pouring of particles into a container under the influence of gravity.

Inserted particles are assigned the specified atom type and are assigned to two groups: the default group "all" and the group specified in the fix pour/legacy command (which can also be "all").

This command must use the *region* keyword to define an insertion volume. The specified region must have been previously defined with a [region](#) command. It must be of type *block* or a z-axis *cylinder* and must be defined with side = *in*. The cylinder style of region can only be used with 3d simulations.

Each timestep particles are inserted, they are placed randomly inside the insertion volume so as to mimic a stream of poured particles. The larger the volume, the more particles that can be inserted at any one timestep. Particles are inserted again after enough time has elapsed that the previously inserted particles fall out of the insertion volume under the influence of gravity. Insertions continue every so many timesteps until the desired # of particles has been inserted.

All other keywords are optional with defaults as shown below. The *diam*, *dens*, and *vel* options enable inserted particles to have a range of diameters or densities or xy velocities. The specific values for a particular inserted particle will be chosen randomly and uniformly between the specified bounds. The *vz* or *vy* value for option *vel* assigns a z-velocity (3d) or y-velocity (2d) to each inserted particle.

The *vol* option specifies what volume fraction of the insertion volume will be filled with particles. The higher the value, the more particles are inserted each timestep. Since inserted particles cannot overlap, the maximum volume fraction should be no higher than about 0.6. Each timestep particles are inserted, LAMMPS will make up to a total of M tries to insert the new particles without overlaps, where M = # of inserted particles * Nattempt. If LAMMPS is unsuccessful at completing all insertions, it prints a warning.

The *rate* option moves the insertion volume in the z direction (3d) or y direction (2d). This enables pouring particles from a successively higher height over time.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](). This means you must be careful when restarting a pouring simulation, when the restart file was written in the middle of the pouring operation. Specifically, you should use a new fix pour command in the input script for the restarted simulation that continues the operation. You will need to adjust the arguments of the original fix pour command to do this.

Also note that because the state of the random number generator is not saved in restart files, you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior if you adjust the fix pour/legacy parameters appropriately.

None of the [fix_modify]() options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](). No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions:

This fix is part of the GRANULAR package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS]() section for more info.

For 3d simulations, a gravity fix in the -z direction must be defined for use in conjunction with this fix. For 2d simulations, gravity must be defined in the -y direction.

The specified insertion region cannot be a "dynamic" region, as defined by the [region]() command.

Related commands:

[fix_deposit](), [fix_gravity](), [region]()

Default:

The option defaults are diam = 1.0 1.0, dens = 1.0 1.0, vol = 0.25 50, rate = 0.0, vel = 0.0 0.0 0.0 0.0 0.0.

fix press/berendsen command

Syntax:

```
fix ID group-ID press/berendsen keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- press/berendsen = style name of this fix command

```
  one or more keyword value pairs may be appended
  keyword = iso or aniso or x or y or z or couple or dilate or modulus
    iso or aniso values = Pstart Pstop Pdamp
      Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
      Pdamp = pressure damping parameter (time units)
    x or y or z values = Pstart Pstop Pdamp
      Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
      Pdamp = stress damping parameter (time units)
    couple = none or xyz or xy or yz or xz
    modulus value = bulk modulus of system (pressure units)
    dilate value = all or partial
```

Examples:

```
fix 1 all press/berendsen iso 0.0 0.0 1000.0
fix 2 all press/berendsen aniso 0.0 0.0 1000.0 dilate partial
```

Description:

Reset the pressure of the system by using a Berendsen barostat [(Berendsen)](#), which rescales the system volume and (optionally) the atoms coordinates within the simulation box every timestep.

Regardless of what atoms are in the fix group, a global pressure is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a value of *partial*, in which case only the atoms in the fix group are re-scaled. The latter can be useful for leaving the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid.

IMPORTANT NOTE: Unlike the [fix npt](#) or [fix nph](#) commands which perform Nose/Hoover barostatting AND time integration, this fix does NOT perform time integration. It only modifies the box size and atom coordinates to effect barostatting. Thus you must use a separate time integration fix, like [fix nve](#) or [fix nvt](#) to actually update the positions and velocities of atoms. This fix can be used in conjunction with thermostatting fixes to control the temperature, such as [fix nvt](#) or [fix langevin](#) or [fix temp/berendsen](#).

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting and barostatting.

The barostat is specified using one or more of the *iso*, *aniso*, *x*, *y*, *z*, and *couple* keywords. These keywords give you the ability to specify the 3 diagonal components of an external stress tensor, and to couple various of these components together so that the dimensions they represent are varied together during a constant-pressure simulation. Unlike the [fix npt](#) and [fix nph](#) commands, this fix cannot be used with triclinic (non-orthogonal) simulation boxes to control all 6 components of the general pressure tensor.

The target pressures for each of the 3 diagonal components of the stress tensor can be specified independently via the *x*, *y*, *z*, keywords, which correspond to the 3 simulation box dimensions. For each component, the

external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the y-box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the fix deform command.

For all barostat keywords, the *Pdamp* parameter determines the time scale on which pressure is relaxed. For example, a value of 1000.0 means to relax the pressure in a timespan of (roughly) 1000 time units (tau or fmsec or psec - see the units command).

IMPORTANT NOTE: The relaxation time is actually also a function of the bulk modulus of the system (inverse of isothermal compressibility). The bulk modulus has units of pressure and is the amount of pressure that would need to be applied (isotropically) to reduce the volume of the system by a factor of 2 (assuming the bulk modulus was a constant, independent of density, which it's not). The bulk modulus can be set via the keyword *modulus*. The *Pdamp* parameter is effectively multiplied by the bulk modulus, so if the pressure is relaxing faster than expected or desired, increasing the bulk modulus has the same effect as increasing *Pdamp*. The converse is also true. LAMMPS does not attempt to guess a correct value of the bulk modulus; it just uses 10.0 as a default value which gives reasonable relaxation for a Lennard-Jones liquid, but will be way off for other materials and way too small for solids. Thus you should experiment to find appropriate values of *Pdamp* and/or the *modulus* when using this fix.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso* and *aniso* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

This fix computes a temperature and pressure each timestep. To do this, the fix creates its own computes of style "temp" and "pressure", as if these commands had been issued:

```
compute fix-ID_temp group-ID temp
compute fix-ID_press group-ID pressure fix-ID_temp
```

fix press/berendsen command

See the [compute temp](#) and [compute pressure](#) commands for details. Note that the IDs of the new computes are the fix-ID + underscore + "temp" or fix_ID + underscore + "press", and the group for the new computes is the same as the fix group.

Note that these are NOT the computes used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp* and *thermo_press*. This means you can change the attributes of this fix's temperature or pressure via the [compute_modify](#) command or print this temperature or pressure during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* or *thermo_press* will have no effect on this fix.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *temp* and *press* options are supported by this fix. You can use them to assign a [compute](#) you have defined to this fix which will be used in its temperature and pressure calculations. If you do this, note that the kinetic energy derived from the compute temperature should be consistent with the virial term computed using all atoms for the pressure. LAMMPS will warn you if you choose to compute temperature on a subset of atoms.

No global or per-atom quantities are stored by this fix for access by various [output commands](#).

This fix can ramp its target pressure over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions:

Any dimension being adjusted by this fix must be periodic.

Related commands:

[fix nve](#), [fix nph](#), [fix npt](#), [fix temp/berendsen](#), [fix_modify](#)

Default:

The keyword defaults are dilate = all, modulus = 10.0 in units of pressure for whatever [units](#) are defined.

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

fix print command

Syntax:

```
fix ID group-ID print N string keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- print = style name of this fix command
- N = print every N steps
- string = text string to print with optional variable names
- zero or more keyword/value pairs may be appended
- keyword = *file* or *append* or *screen* or *title*

```
  file value = filename
    append value = filename
    screen value = yes or no
    title value = string
      string =  text to print as 1st line of output file
```

Examples:

```
fix extra all print 100 "Coords of marker atom = $x $y $z"
fix extra all print 100 "Coords of marker atom = $x $y $z" file coord.txt
```

Description:

Print a text string every N steps during a simulation run. This can be used for diagnostic purposes or as a debugging tool to monitor some quantity during a run. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables it must be enclosed in double quotes to insure they are not evaluated when the input script line is read, but will instead be evaluated each time the string is printed.

See the [variable](#) command for a description of *equal* style variables which are the most useful ones to use with the fix print command, since they are evaluated afresh each timestep that the fix print line is output. Equal-style variables calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a [compute](#) or [fix](#), or references to other [variables](#).

If the *file* or *append* keyword is used, a filename is specified to which the output generated by this fix will be written. If *file* is used, then the filename is overwritten if it already exists. If *append* is used, then the filename is appended to if it already exists, or created if it does not exist.

If the *screen* keyword is used, output by this fix to the screen and logfile can be turned on or off as desired.

The *title* keyword allow specification of the string that will be printed as the first line of the output file, assuming the *file* keyword was used. By default, the title line is as follows:

```
# Fix print output for fix ID
```

where ID is replaced with the fix-ID.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No

parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[variable](#), [print](#)

Default:

The option defaults are no file output, screen = yes, and title string as described above.

fix property/atom/tracer command

Syntax:

```
fix id group property/atom/tracer region_mark region-ID mark_step s keyword value ...
```

- ID, group-ID are documented in fix command
- property/atom/tracer = style name of this fix command
- region_mark = obligatory keyword
- region-ID = ID of region atoms must be in to be marked
- mark_step = obligatory keyword
- s = step when atoms are marked (or started to be marked, depending on marker_style)
- zero or more keyword/value pairs may be appended to args
- keyword = *marker_style* or *check_mark_every*

```
  marker_style value = dirac or heaviside
      dirac = use a dirac impulse at time step s to mark the particles
      heaviside = use a dirac impulse staring at time step s to mark the particles
    check_mark_every value = n
      n = check every that many time-step if atom are in region to be marked
```

Examples:

```
fix tracer all property/atom/tracer region_mark mark mark_step 10000 marker_style dirac check_mar
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command is not available in LAMMPS.

Description:

Fix property/atom/tracer marks particles using either a Dirac delta impulse (default) or a Heaviside impulse, as specified by the *marker_style* keyword. Particles are marked if they are inside the region specified by the *region_mark* keyword. Using the Dirac impulse means that all the particles which are in the region at the time-step specified by the *mark_step* keyword are marked. Using the Heaviside impulse means that all the particles which pass by the specified region after the specified time-step are marked.

Keyword *check_mark_every* can be used to control how often the region is checked. Typically, this is useful when the *heaviside* option is used, because you may not want to check each particle each time-step. However, be careful not to choose this value too large, in this case you could skip particles passing through the region.

It is useful to combine this command with a compute nparticles/tracer/region command to compute residence time distributions.

IMPORTANT NOTE: Using compute nparticles/tracer/region can change the tracer value (keyword **reset_marker**.)

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to **binary restart files** .

This fix computes a per-atom vector (the marker) which can be accessed by various output commands. . This fix also computes a global scalar indicating how many particles were marked since the last time the global

scalar was computed. This scalar can also be accessed by various [output commands](). .

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

[compute nparticles/tracer/region]()

Default:

marker_style = dirac, check_mark_every = 10

fix property/atom/tracer/stream command

Syntax:

```
fix id group property/atom/tracer/stream region_mark region-ID mark_step s n_tracer n insert_stre
```

- ID, group-ID are documented in fix command
- property/atom/tracer/stream = style name of this fix command
- mark_step = obligatory keyword
- s = step when atoms are marked (or started to be marked, depending on marker_style)
- n_tracer = obligatory keyword
- n = number of tracer atoms to be marked
- insert_stream = obligatory keyword
- ins-ID = ID of a fix insert/stream
- every = obligatory keyword

```
  e = 'once' or integer > 0
```

Examples:

```
fix tracer all property/atom/tracer/stream mark_step 10000  insert_stream ins  n_tracer 20  every
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command is not available in LAMMPS.

Description:

Fix property/atom/tracer/stream marks a given number of particles (as defined by keyword *n_tracer*) which are generated by a fix insert/stream command (as defined by keyword *ins-ID*. The first *n_tracer* particles which pass the insertion face after time-step *mark_step* are being marked as tracers. In case of **every** = once, this procedure is performed once, otherwise the procedure is repeated for the first *n_tracer* particles which pass the insertion face after step *mark_step* + *every*.

Note that even for option *once*, particles are the marking procedure can extend over multiple packets of insertion by a fix insert/stream in case that the number of particles inserted in a packet is smaller than the number of particles to tag (as defined by keyword *n_tracer*).

An arbitrary number of fix property/atom/tracer/stream commands can be used for a given fix insert/stream.

It is useful to combine this command with a compute nparticles/tracer/region command to compute residence time distributions.

IMPORTANT NOTE: Due to some parallel operation which needed to tag the particles, you need an atom map to be allocated, see the atom_modify command for details.

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to binary restart files .

This fix computes a per-atom vector (the marker) which can be accessed by various output commands. . This fix also computes a global scalar indicating how many particles were marked since the last time the global scalar was computed. This scalar can also be accessed by various output commands. .

fix property/atom/tracer/stream command   514

Restrictions:

Currently, this feature does not support multi-sphere particles.

Related commands:

compute nparticles/tracer/region

Default: none

fix property/global command

fix property/atom command

Syntax:

```
fix id group property/atom variablename style restartvalue comm_ghost_value comm_reverse_ghost_va
fix id group property/global variablename style stylearg defaultvalue(s)...
```

- ID, group-ID are documented in [fix](#) command
- property/global or property/atom = style name of this fix command
- variablename = a valid C++ string
- restartvalues = 'yes' or 'no'
- comm_ghost_value = 'yes' or 'no'
- comm_reverse_ghost_value = 'yes' or 'no'

fix property/global:

- style = scalar or vector or atomtype or matrix or atomtypepair

```
        stylearg for scalar/vector: none
        stylearg for matrix/atomtypepair: nCols
```

fix property/atom:

- style = scalar or vector
- restartvalue = yes or no
- communicate_ghost_value = yes or no
- communicate_reverse_ghost_value = yes or no

Examples:

```
fix m3 all property/global coefficientRestitution peratomtypepair 1 0.3
fix m5 all property/global characteristicVelocity scalar 2.
fix uf all property/atom uf vector yes no no  0. 0. 0.
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command is not available in LAMMPS.

Description:

Fix property/atom reserves per-atom properties to be accessed by the user or other fixes. Style *scalar* reserves one value per atom, style *vector* multiple values per atoms, where the number of *defaultvalues* (that are assigned to the atoms at creation) determines the length of the vector. The group of atoms the fix is applied to is always "all", irrespective of which group is used for the fix command . If you want to assign different values for different groups, you can use the [set](#) command with keyword 'property/atom'. Keyword *restartvalues* determines whether information about the values stored by this fix is written to binary restart files. Keyword *communicate_ghost_value* determines whether information about the values stored by this fix can be communicated to ghost particles (forward communication). The exact location during a time-step when this happens depends on the model that uses this fix. Keyword *communicate_reverse_ghost_value* determines whether information about the values stored by this fix can be communicated from ghost particles to owned particles (reverse communication). The exact location during a time-step when this happens depends on the

model that uses this fix.

Fix property/global reserves global properties to be accessed by the user or other fixes or pair styles. The number of defaultvalues determines the length of the vector / the number of matrix components . For style *vector* or *atomtype*, the user provides the number of vector components . For style *matrix* or *atomtypepair*, the user provides the number of matrix columns (*nCols*) .

Example: *nCols*= 2 and *defaultvalues* = 1 2 3 4 5 6 would be mapped into a matrix like

[1 2]

[3 4]

[5 6]

Note that the number of default values must thus be a multiple of *nCols*. Note that *vector* and *atomtype* do the same thing, *atomtype* is just provided to make input scripts more readable . Note that *matrix* and *atomtypepair* both refer to a matrix of global values. However, a matrix defined via *atomtypepair* is required to be symmetric.

Note that the group of atoms the fix is applied to is ignored (as the fix is not applied to atoms, but defines values of global scope).

Restart, fix_modify, output, run start/stop, minimize info:

Information about this fix is written to **binary restart files** if you set *restartvalue* to 'yes'.

Restrictions: none

Related commands:

set, pair_gran

Default: none

fix qeq/comb command

fix qeq/comb/omp command

Syntax:

```
fix ID group-ID qeq/comb Nevery precision keyword value ...
```

- ID, group-ID are documented in fix command
- qeq/comb = style name of this fix command
- Nevery = perform charge equilibration every this many steps
- precision = convergence criterion for charge equilibration
- zero or more keyword/value pairs may be appended
- keyword = *file*

```
  file value = filename
        filename = name of file to write QEQ equilibration info to
```

Examples:

```
fix 1 surface qeq/comb 10 0.0001
```

Description:

Perform charge equilibration (QeQ) in conjunction with the COMB (Charge-Optimized Many-Body) potential as described in (COMB_1) and (COMB_2). It performs the charge equilibration portion of the calculation using the so-called QEq method, whereby the charge on each atom is adjusted to minimize the energy of the system. This fix can only be used with the COMB potential; see the fix qeq/reax command for a QeQ calculation that can be used with any potential.

Only charges on the atoms in the specified group are equilibrated. The fix relies on the pair style (COMB in this case) to calculate the per-atom electronegativity (effective force on the charges). An electronegativity equalization calculation (or QEq) is performed in an interative fashion, which in parallel requires communication at each iteration for processors to exchange charge information about nearby atoms with each other. See Rappe_and_Goddard and Rick_and_Stuart for details.

During a run, charge equilibration is peformed every *Nevery* time steps. Charge equilibration is also always enforced on the first step of each run. The *precision* argument controls the tolerance for the difference in electronegativity for all atoms during charge equilibration. *Precision* is a trade-off between the cost of performing charge equilibration (more iterations) and accuracy.

If the *file* keyword is used, then information about each equilibration calculation is written to the specifed file.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix produces a per-atom vector which can be accessed by various output commands. The vector stores the gradient of the charge on each atom. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix can be invoked during energy minimization.

Restrictions:

This fix command currently only supports pair style *comb*.

Related commands:

pair_style comb

Default:

No file output is performed.

(COMB_1) J. Yu, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 75, 085311 (2007),

(COMB_2) T.-R. Shan, B. D. Devine, T. W. Kemper, S. B. Sinnott, S. R. Phillpot, Phys Rev B, 81, 125328 (2010).

(Rappe_and_Goddard) A. K. Rappe, W. A. Goddard, J Phys Chem 95, 3358 (1991).

(Rick_and_Stuart) S. W. Rick, S. J. Stuart, B. J. Berne, J Chem Phys 101, 16141 (1994).

fix qeq/reax command

Syntax:

```
fix ID group-ID qeq/reax Nevery cutlo cuthi tolerance params
```

- ID, group-ID are documented in [fix](#) command
- qeq/reax = style name of this fix command
- Nevery = perform QEq every this many steps
- cutlo,cuthi = lo and hi cutoff for Taper radius
- tolerance = precision to which charges will be equilibrated
- params = reax/c or a filename

Examples:

```
fix 1 all qeq/reax 1 0.0 10.0 1.0e-6 reax/c
fix 1 all qeq/reax 1 0.0 10.0 1.0e-6 param.qeq
```

Description:

Perform the charge equilibration (QEq) method as described in [(Rappe and Goddard)](#) and formulated in [(Nakano)](#). It is typically used in conjunction with the ReaxFF force field model as implemented in the [pair_style reax/c](#) command, but it can be used with any potential in LAMMPS, so long as it defines and uses charges on each atom. The [fix qeq/comb](#) command should be used to perform charge equliibration with the [COMB potential](#). For more technical details about the charge equilibration performed by fix qeq/reax, see the [(Aktulga)](#) paper.

The QEq method minimizes the electrostatic energy of the system by adjusting the partial charge on individual atoms based on interactions with their neighbors. It reqires some parameters for each atom type. If the *params* setting above is the word "reax/c", then these are extracted from the [pair_style reax/c](#) command and the ReaxFF force field file it reads in. If a file name is specified for *params*, then the parameters are taken from the specified file and the file must contain one line for each atom type. The latter form must be used when performing QeQ with a non-ReaxFF potential. Each line should be formatted as follows:

```
itype chi eta gamma
```

where *itype* is the atom type from 1 to Ntypes, *chi* denotes the electronegativity in eV, *eta* denotes the self-Coulomb potential in eV, and *gamma* denotes the valence orbital exponent. Note that these 3 quantities are also in the ReaxFF potential file, except that eta is defined here as twice the eta value in the ReaxFF file. Note that unlike the rest of LAMMPS, the units of this fix are hard-coded to be A, eV, and electronic charge.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). No global scalar or vector or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is invoked during [energy minimization](#).

Restrictions:

This fix is part of the USER-REAXC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

This fix does not correctly handle interactions involving multiple periodic images of the same atom. Hence, it should not be used for periodic cell dimensions less than 10 angstroms.

Related commands:

pair_style reax/c

Default: none

(Rappe) Rappe and Goddard III, Journal of Physical Chemistry, 105, 3358-3363 (1991).

(Nakano) Nakano, Computer Physics Communications, 104, 59-69 (1997).

(Aktulga) Aktulga, Fogarty, Pandit, Grama, Parallel Computing, 38, 245-259 (2012).

fix reax/bonds command

fix reax/c/bonds command

Syntax:

```
fix ID group-ID reax/bonds Nevery filename
```

- ID, group-ID are documented in [fix](#) command
- reax/bonds = style name of this fix command
- Nevery = output interval in timesteps
- filename = name of output file

Examples:

```
fix 1 all reax/bonds 100 bonds.tatb
fix 1 all reax/c/bonds 100 bonds.reaxc
```

Description:

Write out the bond information computed by the ReaxFF potential specified by [pair_style reax](#) or [pair_style reax/c](#) in the exact same format as the original stand-alone ReaxFF code of Adri van Duin. The bond information is written to *filename* on timesteps that are multiples of *Nevery*, including timestep 0. For time-averaged chemical species analysis, please see the [fix species](#) command.

The format of the output file should be self-explantory.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](#). No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

The fix reax/bonds command requires that the [pair_style reax](#) be invoked. This fix is part of the REAX package. It is only enabled if LAMMPS was built with that package, which also requires the REAX library be built and linked with LAMMPS. The fix reax/c/bonds command requires that the [pair_style reax/c](#) be invoked. This fix is part of the USER-REAXC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_style reax](#), [pair_style reax/c](#), [fix reax/c/species](#)

Default: none

fix reax/c/species command

Syntax:

```
fix ID group-ID reax/c/species Nevery Nrepeat Nfreq filename keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- reax/c/species = style name of this command
- Nevery = sample bond-order every this many timesteps
- Nrepeat = # of bond-order samples used for calculating averages
- Nfreq = calculate average bond-order every this many timesteps
- filename = name of output file
- zero or more keyword/value pairs may be appended
- keyword = *cutoff* or *element* or *position*

```
  cutoff value = I J Cutoff
      I, J = atom types
      Cutoff = Bond-order cutoff value for this pair of atom types
    element value = Element1, Element2, ...
    position value = posfreq filepos
      posfreq = write position files every this many timestep
      filepos = name of position output file
```

Examples:

```
fix 1 all reax/c/species 10 10 100 species.out
fix 1 all reax/c/species 1 2 20 species.out cutoff 1 1 0.40 cutoff 1 2 0.55
fix 1 all reax/c/species 1 100 100 species.out element Au O H position 1000 AuOH.pos
```

Description:

Write out the chemical species information computed by the ReaxFF potential specified by [pair_style reax/c](#). Bond-order values (either averaged or instantaneous, depending on value of *Nrepeat*) are used to determine chemical bonds. Every *Nfreq* timesteps, chemical species information is written to *filename* as a two line output. The first line is a header containing labels. The second line consists of the following: timestep, total number of molecules, total number of distinct species, number of molecules of each species. The chemical formula of each species is given in the first line.

Optional keyword *cutoff* can be assigned to change the minimum bond-order values used in identifying chemical bonds between pairs of atoms. Bond-order cutoffs should be carefully chosen, as bond-order cutoffs that are too small may include too many bonds (which will result in an error), while too-large cutoffs will result in fragmented molecules. The default cutoff of 0.3 usually gives good estimate.

Optional keyword *element* can be used to specify the chemical symbol printed for each LAMMPS atom type. The number of symbols must match the number of LAMMPS atom types and each symbol must consist of 1 or 2 alphanumeric characters. Normally, these symbols should be chosen to match the chemical identity of each LAMMPS atom type, as specified using the [reax/c pair_coeff](#) command and the ReaxFF force field file.

Optional keyword *position* writes center-of-mass positions of each identified molecules to file *filepos* every *posfreq* timesteps. The first line contains information on timestep, total number of molecules, total number of distinct species, and box dimensions. The second line is a header containing labels. From the third line downward, each molecule writes a line of output containing the following information: molecule ID, number of atoms in this molecule, chemical formula, total charge, and center-of-mass xyz positions of this molecule. The xyz positions are in fractional coordinates relative to the box dimensions.

Keyword *position* output file *filepos* can contain the wildcard character "*". If the "*" character appears in *filepos*, then one file per snapshot is written at *posfreq* and the "*" character is replaced with the timestep value. For example, AuO.pos.* becomes AuO.pos.0, AuO.pos.1000, etc.

The *Nevery*, *Nrepeat*, and *Nfreq* arguments specify on what timesteps the bond-order values are sampled to get the average bond order. The species analysis is performed using the average bond-order on timesteps that are a multiple of *Nfreq*. The average is over *Nrepeat* bond-order samples, computed in the preceding portion of the simulation every *Nevery* timesteps. *Nfreq* must be a multiple of *Nevery* and *Nevery* must be non-zero even if *Nrepeat* is 1. Also, the timesteps contributing to the average bond-order cannot overlap, i.e. Nfreq > (Nrepeat-1)*Nevery is required.

For example, if Nevery=2, Nrepeat=6, and Nfreq=100, then bond-order values on timesteps 90,92,94,96,98,100 will be used to compute the average bond-order for the species analysis output on timestep 100.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

The fix species currently only works with pair_style reax/c and it requires that the pair_style reax/c be invoked. This fix is part of the USER-REAXC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

It should be possible to extend it to other reactive pair_styles (such as rebo, airebo, comb, and bop), but this has not yet been done.

Related commands:

pair_style reax/c, fix reax/bonds

Default:

The default values for bond-order cutoffs are 0.3 for all I-J pairs. The default element symbols are C, H, O, N. Position files are not written by default.

fix recenter command

Syntax:

```
fix ID group-ID recenter x y z keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- recenter = style name of this fix command
- x,y,z = constrain center-of-mass to these coords (distance units), any coord can also be NULL or INIT (see below)
- zero or more keyword/value pairs may be appended
- keyword = *shift* or *units*

```
  shift value = group-ID
      group-ID = group of atoms whose coords are shifted
    units value = box or lattice or fraction
```

Examples:

```
fix 1 all recenter 0.0 0.5 0.0
fix 1 all recenter INIT INIT NULL
fix 1 all recenter INIT 0.0 0.0 units box
```

Description:

Constrain the center-of-mass position of a group of atoms by adjusting the coordinates of the atoms every timestep. This is simply a small shift that does not alter the dynamics of the system or change the relative coordinates of any pair of atoms in the group. This can be used to insure the entire collection of atoms (or a portion of them) do not drift during the simulation due to random perturbations (e.g. [fix langevin](#) thermostatting).

Distance units for the x,y,z values are determined by the setting of the *units* keyword, as discussed below. One or more x,y,z values can also be specified as NULL, which means exclude that dimension from this operation. Or it can be specified as INIT which means to constrain the center-of-mass to its initial value at the beginning of the run.

The center-of-mass (COM) is computed for the group specified by the fix. If the current COM is different than the specified x,y,z, then a group of atoms has their coordinates shifted by the difference. By default the shifted group is also the group specified by the fix. A different group can be shifted by using the *shift* keyword. For example, the COM could be computed on a protein to keep it in the center of the simulation box. But the entire system (protein + water) could be shifted.

If the *units* keyword is set to *box*, then the distance units of x,y,z are defined by the [units](#) command - e.g. Angstroms for *real* units. A *lattice* value means the distance units are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing. A *fraction* value means a fractional distance between the lo/hi box boundaries, e.g. 0.5 = middle of the box. The default is to use lattice units.

Note that the [velocity](#) command can be used to create velocities with zero aggregate linear and/or angular momentum.

IMPORTANT NOTE: This fix performs its operations at the same point in the timestep as other time integration fixes, such as [fix nve](#), [fix nvt](#), or [fix npt](#). Thus fix recenter should normally be the last such fix specified in the input script, since the adjustments it makes to atom coordinates should come after the changes

made by time integration. LAMMPS will warn you if your fixes are not ordered this way.

IMPORTANT NOTE: If you use this fix on a small group of atoms (e.g. a molecule in solvent) without using the *shift* keyword to adjust the positions of all atoms in the system, then the results can be unpredictable. For example, if the molecule is pushed in one direction by the solvent, its velocity will increase. But its coordinates will be recentered, meaning it is pushed back towards the force. Thus over time, the velocity and temperature of the molecule could become very large (though it won't appear to be moving due to the recentering). If you are thermostatting the entire system, then the solvent would be cooled to compensate. A better solution for this simulation scenario is to use the fix spring command to tether the molecule in place.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix should not be used with an x,y,z setting that causes a large shift in the system on the 1st timestep, due to the requested COM being very different from the initial COM. This could cause atoms to be lost, especially in parallel. Instead, use the displace_atoms command, which can be used to move atoms a large distance.

Related commands:

fix momentum, velocity

Default:

The option defaults are shift = fix group-ID, and units = lattice.

fix restrain command

Syntax:

```
fix ID group-ID restrain keyword args ...
```

- ID, group-ID are documented in fix command
- restrain = style name of this fix command
- one or more keyword/arg pairs may be appended
- keyword = *bond* or *angle* or *dihedral*

```
  bond args = atom1 atom2 Kstart Kstop r0
      atom1,atom2 = IDs of 2 atoms in bond
      Kstart,Kstop = restraint coefficients at start/end of run (energy units)
      r0 = equilibrium bond distance (distance units)
    angle args = atom1 atom2 atom3 Kstart Kstop theta0
      atom1,atom2,atom3 = IDs of 3 atoms in angle, atom2 = middle atom
      Kstart,Kstop = restraint coefficients at start/end of run (energy units)
      theta0 = equilibrium angle theta (degrees)
    bond args = atom1 atom2 atom3 atom4 Kstart Kstop phi0
      atom1,atom2,atom3,atom4 = IDs of 4 atoms in dihedral in linear order
      Kstart,Kstop = restraint coefficients at start/end of run (energy units)
      phi0 = equilibrium dihedral angle phi (degrees)
```

Examples:

```
fix holdem all restrain bond 45 48 2000.0 2000.0 2.75
fix holdem all restrain dihedral 1 2 3 4 2000.0 2000.0 120.0
fix holdem all restrain bond 45 48 2000.0 2000.0 2.75 dihedral 1 2 3 4 2000.0 2000.0 120.0
fix texas_holdem all restrain dihedral 1 2 3 4 0.0 2000.0 120.0 dihedral 1 2 3 5 0.0 2000.0 -120.
```

Description:

Restrain the motion of the specified sets of atoms by making them part of a bond or angle or dihedral interaction whose strength can vary over time during a simulation. This is functionally equivalent to creating a bond or angle or dihedral for the same atoms in a data file, as specified by the read_data command, albeit with a time-varying pre-factor coefficient. For the purpose of forcefield parameter-fitting or mapping a molecular potential energy surface, this fix reduces the hassle and risk associated with modifying data files. In other words, use this fix to temporarily force a molecule to adopt a particular conformation. To create a permanent bond or angle or dihedral, you should modify the data file.

The group-ID specified by this fix is ignored.

The second example above applies a restraint to hold the dihedral angle formed by atoms 1, 2, 3, and 4 near 120 degrees using a constant restraint coefficient. The fourth example applies similar restraints to multiple dihedral angles using a restraint coefficient that increases from 0.0 to 2000.0 over the course of the run.

IMPORTANT NOTE: Adding a force to atoms implies a change in their potential energy as they move due to the applied force field. For dynamics via the run command, this energy can be added to the system's potential energy for thermodynamic output (see below). For energy minimization via the minimize command, this energy must be added to the system's potential energy to formulate a self-consistent minimization problem (see below).

In order for a restraint to be effective, the restraint force must typically be significantly larger than the forces associated with conventional forcefield terms. If the restraint is applied during a dynamics run (as opposed to

during an energy minimization), a large restraint coefficient can significantly reduce the stable timestep size, especially if the atoms are initially far from the preferred conformation. You may need to experiment to determine what value of K works best for a given application.

For the case of finding a minimum energy structure for a single molecule with particular restratins (e.g. for fitting forcefield parameters or constructing a potential energy surface), commands such as the following may be useful:

```
# minimize molecule energy with restraints
velocity all create 600.0 8675309 mom yes rot yes dist gaussian
fix NVE all nve
fix TFIX all langevin 600.0 0.0 100 24601
fix REST all restrain dihedral 2 1 3 8 0.0 5000.0 $angle1 dihedral 3 1 2 9 0.0 5000.0 $angle2
fix_modify REST energy yes
run 10000
fix TFIX all langevin 0.0 0.0 100 24601
fix REST all restrain dihedral 2 1 3 8 5000.0 5000.0 $angle1 dihedral 3 1 2 9 5000.0 5000.0 $angl
fix_modify REST energy yes
run 10000
# sanity check for convergence
minimize 1e-6 1e-9 1000 100000
# report unrestrained energies
unfix REST
run 0
```

The *bond* keyword applies a bond restraint to the specified atoms using the same functional form used by the bond_style harmonic command. The potential associated with the restraint is

$$E = K(r - r_0)^2$$

with the following coefficients:

- K (energy/distance^2)
- r0 (distance)

K and r0 are specified with the fix. Note that the usual 1/2 factor is included in K.

The *angle* keyword applies an angle restraint to the specified atoms using the same functional form used by the angle_style harmonic command. The potential associated with the restraint is

$$E = K(\theta - \theta_0)^2$$

with the following coefficients:

- K (energy/radian^2)
- theta0 (degrees)

K and theta0 are specified with the fix. Note that the usual 1/2 factor is included in K.

The *dihedral* keyword applies a dihedral restraint to the specified atoms using a simplified form of the function used by the dihedral_style charmm command. The potential associated with the restraint is

$$E = K[1 + \cos(n\phi - d)]$$

with the following coefficients:

- K (energy)
- n = 1
- d (degrees) = phi0 + 180

K and phi0 are specified with the fix. Note that the value of n is hard-wired to 1. Also note that the energy will be a minimum when the current dihedral angle phi is equal to phi0.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the potential energy associated with this fix to the system's potential energy as part of thermodynamic output.

IMPORTANT NOTE: If you want the fictitious potential energy associated with the added forces to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

This fix computes a global scalar, which can be accessed by various output commands. The scalar is the potential energy for all the restraints as discussed above. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

Restrictions: none

Related commands: none

Default: none

fix rigid command

fix rigid/nve command

fix rigid/nvt command

fix rigid/npt command

fix rigid/nph command

fix rigid/small command

Syntax:

```
fix ID group-ID style bodystyle args keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- style = *rigid* or *rigid/nve* or *rigid/nvt* or *rigid/npt* or *rigid/nph* or *rigid/small*
- bodystyle = *single* or *molecule* or *group*

```
  single args = none
    molecule args = none
    group args = N groupID1 groupID2 ...
      N = # of groups
      groupID1, groupID2, ... = list of N group IDs
```

- zero or more keyword/value pairs may be appended
- keyword = *langevin* or *temp* or *iso* or *aniso* or *x* or *y* or *z* or *couple* or *tparam* or *pchain* or *dilate* or *force* or *torque* or *infile*

```
  langevin values = Tstart Tstop Tperiod seed
     Tstart,Tstop = desired temperature at start/stop of run (temperature units)
     Tdamp = temperature damping parameter (time units)
     seed = random number seed to use for white noise (positive integer)
  temp values = Tstart Tstop Tdamp
     Tstart,Tstop = desired temperature at start/stop of run (temperature units)
     Tdamp = temperature damping parameter (time units)
  iso or aniso values = Pstart Pstop Pdamp
     Pstart,Pstop = scalar external pressure at start/end of run (pressure units)
     Pdamp = pressure damping parameter (time units)
  x or y or z values = Pstart Pstop Pdamp
     Pstart,Pstop = external stress tensor component at start/end of run (pressure units)
     Pdamp = stress damping parameter (time units)
  couple = none or xyz or xy or yz or xz
  tparam values = Tchain Titer Torder
     Tchain = length of Nose/Hoover thermostat chain
     Titer = number of thermostat iterations performed
     Torder = 3 or 5 = Yoshida-Suzuki integration parameters
  pchain values = Pchain
     Pchain = length of the Nose/Hoover thermostat chain coupled with the barostat
  dilate value = dilate-group-ID
     dilate-group-ID = only dilate atoms in this group due to barostat volume changes
  force values = M xflag yflag zflag
     M = which rigid body from 1-Nbody (see asterisk form below)
     xflag,yflag,zflag = off/on if component of center-of-mass force is active
  torque values = M xflag yflag zflag
     M = which rigid body from 1-Nbody (see asterisk form below)
     xflag,yflag,zflag = off/on if component of center-of-mass torque is active
```

```
infile filename
    filename = file with per-body values of mass, center-of-mass, moments of inertia
```

Examples:

```
fix 1 clump rigid single
fix 1 clump rigid/small molecule
fix 1 clump rigid single force 1 off off on langevin 1.0 1.0 1.0 428984
fix 1 polychains rigid/nvt molecule temp 1.0 1.0 5.0
fix 1 polychains rigid molecule force 1*5 off off off force 6*10 off off on
fix 1 polychains rigid/small molecule langevin 1.0 1.0 1.0 428984
fix 2 fluid rigid group 3 clump1 clump2 clump3 torque * off off off
fix 1 rods rigid/npt molecule temp 300.0 300.0 100.0 iso 0.5 0.5 10.0
fix 1 particles rigid/npt molecule temp 1.0 1.0 5.0 x 0.5 0.5 1.0 z 0.5 0.5 1.0 couple xz
fix 1 water rigid/nph molecule iso 0.5 0.5 1.0
```

Description:

Treat one or more sets of atoms as independent rigid bodies. This means that each timestep the total force and torque on each rigid body is computed as the sum of the forces and torques on its constituent particles and the coordinates, velocities, and orientations of the atoms in each body are updated so that the body moves and rotates as a single entity.

Examples of large rigid bodies are a large colloidal particle, or portions of a large biomolecule such as a protein.

Example of small rigid bodies are patchy nanoparticles, such as those modeled in [this paper](#) by Sharon Glotzer's group, clumps of granular particles, lipid molecules consiting of one or more point dipoles connected to other spheroids or ellipsoids, irregular particles built from line segments (2d) or triangles (3d), and coarse-grain models of nano or colloidal particles consisting of a small number of constituent particles. Note that the [fix shake](#) command can also be used to rigidify small molecules of 2, 3, or 4 atoms, e.g. water molecules. That fix treats the constituent atoms as point masses.

These fixes also update the positions and velocities of the atoms in each rigid body via time integration, in the NVE, NVT, NPT, or NPH ensemble, as described below.

There are two main variants of this fix, fix rigid and fix rigid/small. The NVE/NVT/NPT/NHT versions belong to one of the two variants, as their style names indicate.

IMPORTANT NOTE: Not all of the bodystyle options and keyword/value options are available for both the *rigid* and *rigid/small* variants. See details below.

The *rigid* variant is typically the best choice for a system with a small number of large rigid bodies, each of which can extend across the domain of many processors. It operates by creating a single global list of rigid bodies, which all processors contribute to. MPI_Allreduce operations are performed each timestep to sum the contributions from each processor to the force and torque on all the bodies. This operation will not scale well in parallel if large numbers of rigid bodies are simulated.

The *rigid/small* variant is typically best for a system with a large number of small rigid bodies. Each body is assigned to the atom closest to the geometrical center of the body. The fix operates using local lists of rigid bodies owned by each processor and information is exchanged and summed via local communication between neighboring processors when ghost atom info is accumulated.

IMPORTANT NOTE: To use *rigid/small* the ghost atom cutoff must be large enough to span the distance between the atom that owns the body and every other atom in the body. This distance value is printed out when the rigid bodies are defined. If the [pair_style](#) cutoff plus neighbor skin does not span this distance, then you should use the [communicate cutoff](#) command with a setting epsilon larger than the distance.

fix rigid/small command   531

Which of the two variants is faster for a particular problem is hard to predict. The best way to decide is to perform a short test run. Both variants should give identical numerical answers for short runs. Long runs should give statistically similar results, but round-off differences will accumulate to produce divergent trajectories.

IMPORTANT NOTE: You should not update the atoms in rigid bodies via other time-integration fixes (e.g. fix nve, fix nvt, fix npt), or you will be integrating their motion more than once each timestep. When performing a hybrid simulation with some atoms in rigid bodies, and some not, a separate time integration fix like fix nve or fix nvt should be used for the non-rigid particles.

IMPORTANT NOTE: These fixes are overkill if you simply want to hold a collection of atoms stationary or have them move with a constant velocity. A simpler way to hold atoms stationary is to not include those atoms in your time integration fix. E.g. use "fix 1 mobile nve" instead of "fix 1 all nve", where "mobile" is the group of atoms that you want to move. You can move atoms with a constant velocity by assigning them an initial velocity (via the velocity command), setting the force on them to 0.0 (via the fix setforce command), and integrating them as usual (e.g. via the fix nve command).

Each rigid body must have two or more atoms. An atom can belong to at most one rigid body. Which atoms are in which bodies can be defined via several options.

For bodystyle *single* the entire fix group of atoms is treated as one rigid body. This option is only allowed for fix rigid and its sub-styles.

For bodystyle *molecule*, each set of atoms in the fix group with a different molecule ID is treated as a rigid body. This option is allowed for fix rigid and fix rigid/small, and their sub-styles. Note that atoms with a molecule ID = 0 will be treated as a single rigid body. For a system with atomic solvent (typically this is atoms with molecule ID = 0) surrounding rigid bodies, this may not be what you want. Thus you should be careful to use a fix group that only includes atoms you want to be part of rigid bodies.

For bodystyle *group*, each of the listed groups is treated as a separate rigid body. Only atoms that are also in the fix group are included in each rigid body. This option is only allowed for fix rigid and its sub-styles.

IMPORTANT NOTE: To compute the initial center-of-mass position and other properties of each rigid body, the image flags for each atom in the body are used to "unwrap" the atom coordinates. Thus you must insure that these image flags are consistent so that the unwrapping creates a valid rigid body (one where the atoms are close together), particularly if the atoms in a single rigid body straddle a periodic boundary. This means the input data file or restart file must define the image flags for each atom consistently or that you have used the set command to specify them correctly. If a dimension is non-periodic then the image flag of each atom must be 0 in that dimension, else an error is generated.

The *force* and *torque* keywords discussed next are only allowed for fix rigid and its sub-styles.

By default, each rigid body is acted on by other atoms which induce an external force and torque on its center of mass, causing it to translate and rotate. Components of the external center-of-mass force and torque can be turned off by the *force* and *torque* keywords. This may be useful if you wish a body to rotate but not translate, or vice versa, or if you wish it to rotate or translate continuously unaffected by interactions with other particles. Note that if you expect a rigid body not to move or rotate by using these keywords, you must insure its initial center-of-mass translational or angular velocity is 0.0. Otherwise the initial translational or angular momentum the body has will persist.

An xflag, yflag, or zflag set to *off* means turn off the component of force of torque in that dimension. A setting of *on* means turn on the component, which is the default. Which rigid body(s) the settings apply to is determined by the first argument of the *force* and *torque* keywords. It can be an integer M from 1 to Nbody, where Nbody is the number of rigid bodies defined. A wild-card asterisk can be used in place of, or in

conjunction with, the M argument to set the flags for multiple rigid bodies. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of rigid bodies, then an asterisk with no numeric values means all bodies from 1 to N. A leading asterisk means all bodies from 1 to n (inclusive). A trailing asterisk means all bodies from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that you can use the *force* or *torque* keywords as many times as you like. If a particular rigid body has its component flags set multiple times, the settings from the final keyword are used.

For computational efficiency, you may wish to turn off pairwise and bond interactions within each rigid body, as they no longer contribute to the motion. The neigh_modify exclude and delete_bonds commands are used to do this.

For computational efficiency, you should typically define one fix rigid or fix rigid/small command which includes all the desired rigid bodies. LAMMPS will allow multiple rigid fixes to be defined, but it is more expensive.

The constituent particles within a rigid body can be point particles (the default in LAMMPS) or finite-size particles, such as spheres or ellipsoids or line segments or triangles. See the atom_style sphere and ellipsoid and line and tri commands for more details on these kinds of particles. Finite-size particles contribute differently to the moment of inertia of a rigid body than do point particles. Finite-size particles can also experience torque (e.g. due to frictional granular interactions) and have an orientation. These contributions are accounted for by these fixes.

Forces between particles within a body do not contribute to the external force or torque on the body. Thus for computational efficiency, you may wish to turn off pairwise and bond interactions between particles within each rigid body. The neigh_modify exclude and delete_bonds commands are used to do this. For finite-size particles this also means the particles can be highly overlapped when creating the rigid body.

The *rigid* and *rigid/small* and *rigid/nve* styles perform constant NVE time integration. The only difference is that the *rigid* and *rigid/small* styles use an integration technique based on Richardson iterations. The *rigid/nve* style uses the methods described in the paper by Miller, which are thought to provide better energy conservation than an iterative approach.

The *rigid/nvt* style performs constant NVT integration using a Nose/Hoover thermostat with chains as described originally in (Hoover) and (Martyna), which thermostats both the translational and rotational degrees of freedom of the rigid bodies. The rigid-body algorithm used by *rigid/nvt* is described in the paper by Kamberaj.

The *rigid/npt* and *rigid/nph* styles perform constant NPT or NPH integration using a Nose/Hoover barostat with chains. For the NPT case, the same Nose/Hoover thermostat is also used as with *rigid/nvt*.

The barostat parameters are specified using one or more of the *iso*, *aniso*, *x*, *y*, *z* and *couple* keywords. These keywords give you the ability to specify 3 diagonal components of the external stress tensor, and to couple these components together so that the dimensions they represent are varied together during a constant-pressure simulation. The effects of these keywords are similar to those defined in fix npt/nph

NOTE: Currently the *rigid/npt* and *rigid/nph* styles do not support triclinic (non-orthongonal) boxes.

The target pressures for each of the 6 components of the stress tensor can be specified independently via the *x*, *y*, *z* keywords, which correspond to the 3 simulation box dimensions. For each component, the external pressure or tensor component at each timestep is a ramped value during the run from *Pstart* to *Pstop*. If a target pressure is specified for a component, then the corresponding box dimension will change during a simulation. For example, if the *y* keyword is used, the y-box length will change. A box dimension will not change if that component is not specified, although you have the option to change that dimension via the fix deform command.

For all barostat keywords, the *Pdamp* parameter operates like the *Tdamp* parameter, determining the time scale on which pressure is relaxed. For example, a value of 10.0 means to relax the pressure in a timespan of (roughly) 10 time units (e.g. tau or fmsec or psec - see the units command).

Regardless of what atoms are in the fix group (the only atoms which are time integrated), a global pressure or stress tensor is computed for all atoms. Similarly, when the size of the simulation box is changed, all atoms are re-scaled to new positions, unless the keyword *dilate* is specified with a *dilate-group-ID* for a group that represents a subset of the atoms. This can be useful, for example, to leave the coordinates of atoms in a solid substrate unchanged and controlling the pressure of a surrounding fluid. Another example is a system consisting of rigid bodies and point particles where the barostat is only coupled with the rigid bodies. This option should be used with care, since it can be unphysical to dilate some atoms and not others, because it can introduce large, instantaneous displacements between a pair of atoms (one dilated, one not) that are far from the dilation origin.

The *couple* keyword allows two or three of the diagonal components of the pressure tensor to be "coupled" together. The value specified with the keyword determines which are coupled. For example, *xz* means the *Pxx* and *Pzz* components of the stress tensor are coupled. *Xyz* means all 3 diagonal components are coupled. Coupling means two things: the instantaneous stress will be computed as an average of the corresponding diagonal components, and the coupled box dimensions will be changed together in lockstep, meaning coupled dimensions will be dilated or contracted by the same percentage every timestep. The *Pstart*, *Pstop*, *Pdamp* parameters for any coupled dimensions must be identical. *Couple xyz* can be used for a 2d simulation; the *z* dimension is simply ignored.

The *iso* and *aniso* keywords are simply shortcuts that are equivalent to specifying several other keywords together.

The keyword *iso* means couple all 3 diagonal components together when pressure is computed (hydrostatic pressure), and dilate/contract the dimensions together. Using "iso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple xyz
```

The keyword *aniso* means *x*, *y*, and *z* dimensions are controlled independently using the *Pxx*, *Pyy*, and *Pzz* components of the stress tensor as the driving forces, and the specified scalar external pressure. Using "aniso Pstart Pstop Pdamp" is the same as specifying these 4 keywords:

```
x Pstart Pstop Pdamp
y Pstart Pstop Pdamp
z Pstart Pstop Pdamp
couple none
```

The keyword/value option pairs are used in the following ways.

The *langevin* and *temp* and *tparam* keywords perform thermostatting of the rigid bodies, altering both their translational and rotational degrees of freedom. What is meant by "temperature" of a collection of rigid bodies and how it can be monitored via the fix output is discussed below.

The *langevin* keyword applies a Langevin thermostat to the constant NVE time integration performed by either the *rigid* or *rigid/small* or *rigid/nve* styles. It cannot be used with the *rigid/nvt* style. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec - see the units command). The random # *seed* must be a positive integer.

The way that Langevin thermostatting operates is explained on the fix langevin doc page. If you wish to simply viscously damp the rotational motion without thermostatting, you can set *Tstart* and *Tstop* to 0.0, which means only the viscous drag term in the Langevin thermostat will be applied. See the discussion on the fix viscous doc page for details.

IMPORTANT NOTE: When the *langevin* keyword is used with fix rigid versus fix rigid/small, different dynamics will result for parallel runs. This is because of the way random numbers are used in the two cases. The dynamics for the two cases should be statistically similar, but will not be identical, even for a single timestep.

The *temp* and *tparam* keywords apply a Nose/Hoover thermostat to the NVT time integration performed by the *rigid/nvt* style. They cannot be used with the *rigid* or *rigid/small* or *rigid/nve* styles. The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec - see the units command).

Nose/Hoover chains are used in conjunction with this thermostat. The *tparam* keyword can optionally be used to change the chain settings used. *Tchain* is the number of thermostats in the Nose Hoover chain. This value, along with *Tdamp* can be varied to dampen undesirable oscillations in temperature that can occur in a simulation. As a rule of thumb, increasing the chain length should lead to smaller oscillations. The keyword *pchain* specifies the number of thermostats in the chain thermostatting the barostat degrees of freedom.

IMPORTANT NOTE: There are alternate ways to thermostat a system of rigid bodies. You can use fix langevin to treat the individual particles in the rigid bodies as effectively immersed in an implicit solvent, e.g. a Brownian dynamics model. For hybrid systems with both rigid bodies and solvent particles, you can thermostat only the solvent particles that surround one or more rigid bodies by appropriate choice of groups in the compute and fix commands for temperature and thermostatting. The solvent interactions with the rigid bodies should then effectively thermostat the rigid body temperature as well without use of the Langevin or Nose/Hoover options associated with the fix rigid commands.

The *infile* keyword allows a file of rigid body attributes to be read in from a file, rather then having LAMMPS compute them. There are 3 such attributes: the total mass of the rigid body, its center-of-mass position, and its 6 moments of inertia. For rigid bodies consisting of point particles or non-overlapping finite-size particles, LAMMPS can compute these values accurately. However, for rigid bodies consisting of finite-size particles which overlap each other, LAMMPS will ignore the overlaps when computing these 3 attributes. The amount of error this induces depends on the amount of overlap. To avoid this issue, the values can be pre-computed (e.g. using Monte Carlo integration).

The format of the file is as follows. Note that the file does not have to list attributes for every rigid body integrated by fix rigid. Only bodies which the file specifies will have their computed attributes overridden. The file can contain initial blank lines or comment lines starting with "#" which are ignored. The first non-blank, non-comment line should list N = the number of lines to follow. The N successive lines contain the following information:

```
ID1 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
ID2 masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
...
IDN masstotal xcm ycm zcm ixx iyy izz ixy ixz iyz
```

The rigid body IDs are all positive integers. For the *single* bodystyle, only an ID of 1 can be used. For the *group* bodystyle, IDs from 1 to Ng can be used where Ng is the number of specified groups. For the *molecule* bodystyle, use the molecule ID for the atoms in a specific rigid body as the rigid body ID.

fix rigid/small command   535

The masstotal and center-of-mass coordinates (xcm,ycm,zcm) are self-explanatory. The center-of-mass should be consistent with what is calculated for the position of the rigid body with all its atoms unwrapped by their respective image flags. If this produces a center-of-mass that is outside the simulation box, LAMMPS wraps it back into the box. The 6 moments of inertia (ixx,iyy,izz,ixy,ixz,iyz) should be the values consistent with the current orientation of the rigid body around its center of mass. The values are with respect to the simulation box XYZ axes, not with respect to the prinicpal axes of the rigid body itself. LAMMPS performs the latter calculation internally.

IMPORTANT NOTE: If you use the *infile* keyword and write restart files during a simulation, then each time a restart file is written, the fix also write an auxiliary restart file with the name rfile.rigid, where "rfile" is the name of the restart file, e.g. tmp.restart.10000 and tmp.restart.10000.rigid. This auxiliary file is in the same format described above and contains info on the current center-of-mass and 6 moments of inertia. Thus it can be used in a new input script that restarts the run and re-specifies a rigid fix using an *infile* keyword and the appropriate filename. Note that the auxiliary file will contain one line for every rigid body, even if the original file only listed a subset of the rigid bodies.

If you use a temperature compute with a group that includes particles in rigid bodies, the degrees-of-freedom removed by each rigid body are accounted for in the temperature (and pressure) computation, but only if the temperature group includes all the particles in a particular rigid body.

A 3d rigid body has 6 degrees of freedom (3 translational, 3 rotational), except for a collection of point particles lying on a straight line, which has only 5, e.g a dimer. A 2d rigid body has 3 degrees of freedom (2 translational, 1 rotational).

IMPORTANT NOTE: You may wish to explicitly subtract additional degrees-of-freedom if you use the *force* and *torque* keywords to eliminate certain motions of one or more rigid bodies. LAMMPS does not do this automatically.

The rigid body contribution to the pressure of the system (virial) is also accounted for by this fix.

IMPORTANT NOTE: The periodic image flags of atoms in rigid bodies are altered so that the rigid body can be reconstructed correctly when it straddles periodic boundaries. The atom image flags are not incremented/decremented as they would be for non-rigid atoms as the rigid body crosses periodic boundaries. Specifically, they are set so that the center-of-mass (COM) of the rigid body always remains inside the simulation box.

This means that if you output per-atom image flags you cannot interpret them as you normally would. I.e. the image flag values written to a dump file will be different than they would be if the atoms were not in a rigid body. Likewise the compute msd will not compute the expected mean-squared displacement for such atoms if the body moves across periodic boundaries. It also means that if you have bonds between a pair of rigid bodies and the bond straddles a periodic boundary, you cannot use the replicate command to increase the system size.

Here are details on how, you can post-process a dump file to calculate a diffusion coefficient for rigid bodies, using the altered per-atom image flags written to a dump file. The image flags for atoms in the same rigid body can be used to unwrap the body and calculate its center-of-mass (COM). As mentioned above, this COM will always be inside the simulation box. Thus it will "jump" from one side of the box to the other when the COM crosses a periodic boundary. If you keep track of the jumps, you can effectively "unwrap" the COM and use that value to track the displacement of each rigid body, and thus the mean-squared displacement (MSD) of an ensemble of bodies, and thus a diffusion coefficient.

Note that fix rigid does define image flags for each rigid body, which are incremented when the center-of-mass of the rigid body crosses a periodic boundary in the usual way. These image flags have the same meaning as atom images (see the "dump" command) and can be accessed and output as described below.

If your simlulation is a hybrid model with a mixture of rigid bodies and non-rigid particles (e.g. solvent) there are several ways these rigid fixes can be used in tandem with fix nve, fix nvt, fix npt, and fix nph.

If you wish to perform NVE dynamics (no thermostatting or barostatting), use fix rigid or fix rigid/nve to integrate the rigid bodies, and fix nve to integrate the non-rigid particles.

If you wish to perform NVT dynamics (thermostatting, but no barostatting), you can use fix rigid/nvt for the rigid bodies, and any thermostatting fix for the non-rigid particles (fix nvt, fix langevin, fix temp/berendsen). You can also use fix rigid or fix rigid/nve for the rigid bodies and thermostat them using fix langevin on the group that contains all the particles in the rigid bodies. The net force added by fix langevin to each rigid body effectively thermostats its translational center-of-mass motion. Not sure how well it does at thermostatting its rotational motion.

If you with to perform NPT or NPH dynamics (barostatting), you cannot use both fix npt and fix rigid/npt (or the nph variants). This is because there can only be one fix which monitors the global pressure and changes the simulation box dimensions. So you have 3 choices:

- Use fix rigid/npt for the rigid bodies. Use the *dilate* all option so that it will dilate the positions of the non-rigid particles as well. Use fix nvt (or any other thermostat) for the non-rigid particles.
- Use fix npt for the group of non-rigid particles. Use the *dilate* all option so that it will dilate the center-of-mass positions of the rigid bodies as well. Use fix rigid/nvt for the rigid bodies.
- Use fix press/berendsen to compute the pressure and change the box dimensions. Use fix rigid/nvt for the rigid bodies. Use fix nvt (or any other thermostat) for the non-rigid particles.

In all case, the rigid bodies and non-rigid particles both contribute to the global pressure and the box is scaled the same by any of the barostatting fixes.

You could even use the 2nd and 3rd options for a non-hybrid simulation consisting of only rigid bodies, assuming you give fix npt an empty group, though it's an odd thing to do. The barostatting fixes (fix npt and fix press/berensen) will monitor the pressure and change the box dimensions, but not time integrate any particles. The integration of the rigid bodies will be performed by fix rigid/nvt.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about the *rigid* and *rigid/small* and *rigid/nve* fixes are written to binary restart files. For style *rigid/nvt* the state of the Nose/Hoover thermostat is written to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

fix rigid/small command

The fix_modify *energy* option is supported by the rigid/nvt fix to add the energy change induced by the thermostatting to the system's potential energy as part of thermodynamic output.

The fix_modify *temp* and *press* options are supported by the rigid/npt and rigid/nph fixes to change the computes used to calculate the instantaneous pressure tensor. Note that the rigid/nvt fix does not use any external compute to compute instantaneous temperature.

The *rigid* and *rigid/small* and *rigid/nve* fixes compute a global scalar which can be accessed by various output commands. The scalar value calculated by these fixes is "intensive". The scalar is the current temperature of the collection of rigid bodies. This is averaged over all rigid bodies and their translational and rotational degrees of freedom. The translational energy of a rigid body is 1/2 m v^2, where m = total mass of the body and v = the velocity of its center of mass. The rotational energy of a rigid body is 1/2 I w^2, where I = the moment of inertia tensor of the body and w = its angular velocity. Degrees of freedom constrained by the *force* and *torque* keywords are removed from this calculation, but only for the *rigid* and *rigid/nve* fixes.

The *rigid/nvt*, *rigid/npt*, and *rigid/nph* fixes compute a global scalar which can be accessed by various output commands. The scalar value calculated by these fixes is "extensive". The scalar is the cumulative energy change due to the thermostatting and barostatting the fix performs.

All of the *rigid* fixes except *rigid/small* compute a global array of values which can be accessed by various output commands. The number of rows in the array is equal to the number of rigid bodies. The number of columns is 15. Thus for each rigid body, 15 values are stored: the xyz coords of the center of mass (COM), the xyz components of the COM velocity, the xyz components of the force acting on the COM, the xyz components of the torque acting on the COM, and the xyz image flags of the COM, which have the same meaning as image flags for atom positions (see the "dump" command). The force and torque values in the array are not affected by the *force* and *torque* keywords in the fix rigid command; they reflect values before any changes are made by those keywords.

The ordering of the rigid bodies (by row in the array) is as follows. For the *single* keyword there is just one rigid body. For the *molecule* keyword, the bodies are ordered by ascending molecule ID. For the *group* keyword, the list of group IDs determines the ordering of bodies.

The array values calculated by these fixes are "intensive", meaning they are independent of the number of atoms in the simulation.

No parameter of these fixes can be used with the *start/stop* keywords of the run command. These fixes are not invoked during energy minimization.

Restrictions:

These fixes are all part of the RIGID package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

delete_bonds, neigh_modify exclude

Default:

The option defaults are force * on on on and torque * on on on, meaning all rigid bodies are acted on by center-of-mass force and torque. Also Tchain = Pchain = 10, Titer = 1, Torder = 3.

(Hoover) Hoover, Phys Rev A, 31, 1695 (1985).

(Kamberaj) Kamberaj, Low, Neal, J Chem Phys, 122, 224114 (2005).

(Martyna) Martyna, Klein, Tuckerman, J Chem Phys, 97, 2635 (1992); Martyna, Tuckerman, Tobias, Klein, Mol Phys, 87, 1117.

(Miller) Miller, Eleftheriou, Pattnaik, Ndirango, and Newns, J Chem Phys, 116, 8649 (2002).

(Zhang) Zhang, Glotzer, Nanoletters, 4, 1407-1413 (2004).

fix setforce command

fix setforce/cuda command

Syntax:

```
fix ID group-ID setforce fx fy fz keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- setforce = style name of this fix command
- fx,fy,fz = force component values
- any of fx,fy,fz can be a variable (see below)
- zero or more keyword/value pairs may be appended to args
- keyword = *region*

```
  region value = region-ID
      region-ID = ID of region atoms must be in to have added force
```

Examples:

```
fix freeze indenter setforce 0.0 0.0 0.0
fix 2 edge setforce NULL 0.0 0.0
fix 2 edge setforce NULL 0.0 v_oscillate
```

Description:

Set each component of force on each atom in the group to the specified values fx,fy,fz. This erases all previously computed forces on the atom, though additional fixes could add new forces. This command can be used to freeze certain atoms in the simulation by zeroing their force, either for running dynamics or performing an energy minimization. For dynamics, this assumes their initial velocity is also zero.

Any of the fx,fy,fz values can be specified as NULL which means do not alter the force component in that dimension.

Any of the 3 quantities defining the force components can be specified as an equal-style or atom-style [variable](#), namely *fx*, *fy*, *fz*. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the force component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent force field.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent force field with optional time-dependence as well.

If the *region* keyword is used, the atom must also be in the specified geometric [region](#) in order to have force added to it.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for

round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global 3-vector of forces, which can be accessed by various output commands. This is the total force on the group of atoms before the forces on individual atoms are changed by the fix. The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command, but you cannot set forces to any value besides zero when performing a minimization. Use the fix addforce command if you want to apply a non-zero force to atoms during a minimization.

Restrictions: none

Related commands:

fix addforce, fix aveforce

Default: none

fix shake command

fix shake/cuda command

Syntax:

```
fix ID group-ID shake tol iter N keyword values ...
```

- ID, group-ID are documented in [fix](#) command
- shake = style name of this fix command
- tol = accuracy tolerance of SHAKE solution
- iter = max # of iterations in each SHAKE solution
- N = print SHAKE statistics every this many timesteps (0 = never)
- one or more keyword/value pairs are appended
- keyword = *b* or *a* or *t* or *m*

```
  b values = one or more bond types
    a values = one or more angle types
    t values = one or more atom types
    m value = one or more mass values
```

Examples:

```
fix 1 sub shake 0.0001 20 10 b 4 19 a 3 5 2
fix 1 sub shake 0.0001 20 10 t 5 6 m 1.0 a 31
```

Description:

Apply bond and angle constraints to specified bonds and angles in the simulation. This typically enables a longer timestep.

Each timestep the specified bonds and angles are reset to their equilibrium lengths and angular values via the well-known SHAKE algorithm. This is done by applying an additional constraint force so that the new positions preserve the desired atom separations. The equations for the additional force are solved via an iterative method that typically converges to an accurate solution in a few iterations. The desired tolerance (e.g. 1.0e-4 = 1 part in 10000) and maximum # of iterations are specified as arguments. Setting the N argument will print statistics to the screen and log file about regarding the lengths of bonds and angles that are being constrained. Small delta values mean SHAKE is doing a good job.

In LAMMPS, only small clusters of atoms can be constrained. This is so the constraint calculation for a cluster can be performed by a single processor, to enable good parallel performance. A cluster is defined as a central atom connected to others in the cluster by constrained bonds. LAMMPS allows for the following kinds of clusters to be constrained: one central atom bonded to 1 or 2 or 3 atoms, or one central atom bonded to 2 others and the angle between the 3 atoms also constrained. This means water molecules or CH2 or CH3 groups may be constrained, but not all the C-C backbone bonds of a long polymer chain.

The *b* keyword lists bond types that will be constrained. The *t* keyword lists atom types. All bonds connected to an atom of the specified type will be constrained. The *m* keyword lists atom masses. All bonds connected to atoms of the specified masses will be constrained (within a fudge factor of MASSDELTA specified in fix_shake.cpp). The *a* keyword lists angle types. If both bonds in the angle are constrained then the angle will also be constrained if its type is in the list.

For all keywords, a particular bond is only constrained if both atoms in the bond are in the group specified with the SHAKE fix.

The degrees-of-freedom removed by SHAKE bonds and angles are accounted for in temperature and pressure computations. Similarly, the SHAKE contribution to the pressure of the system (virial) is also accounted for.

IMPORTANT NOTE: This command works by using the current forces on atoms to caculate an additional constraint force which when added will leave the atoms in positions that satisfy the SHAKE constraints (e.g. bond length) after the next time integration step. If you define fixes (e.g. fix efield) that add additional force to the atoms after fix shake operates, then this fix will not take them into account and the time integration will typically not satisfy the SHAKE constraints. The solution for this is to make sure that fix shake is defined in your input script after any other fixes which add or change forces (to atoms that fix shake operates on).

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

For computational efficiency, there can only be one shake fix defined in a simulation.

If you use a tolerance that is too large or a max-iteration count that is too small, the constraints will not be enforced very strongly, which can lead to poor energy conservation. You can test for this in your system by running a constant NVE simulation with a particular set of SHAKE parameters and monitoring the energy versus time.

Related commands: none

Default: none

fix smd command

Syntax:

```
fix ID group-ID smd type values keyword values
```

- ID, group-ID are documented in <u>fix</u> command
- smd = style name of this fix command
- mode = *cvel* or *cfor* to select constant velocity or constant force SMD

```
  cvel values = K vel
      K = spring constant (force/distance units)
      vel = velocity of pulling (distance/time units)
    cfor values = force
      force = pulling force (force units)
```

- keyword = *tether* or *couple*

```
  tether values = x y z R0
      x,y,z = point to which spring is tethered
      R0 = distance of end of spring from tether point (distance units)
    couple values = group-ID2 x y z R0
      group-ID2 = 2nd group to couple to fix group with a spring
      x,y,z = direction of spring, automatically computed with 'auto'
      R0 = distance of end of spring (distance units)
```

Examples:

```
fix  pull    cterm smd cvel 20.0 -0.00005 tether NULL NULL 100.0 0.0
fix  pull    cterm smd cvel 20.0 -0.0001 tether 25.0 25 25.0 0.0
fix  stretch cterm smd cvel 20.0  0.0001 couple nterm auto auto auto 0.0
fix  pull    cterm smd cfor  5.0 tether 25.0 25.0 25.0 0.0
```

Description:

This fix implements several options of steered MD (SMD) as reviewed in <u>(Izrailev)</u>, which allows to induce conformational changes in systems and to compute the potential of mean force (PMF) along the assumed reaction coordinate <u>(Park)</u> based on Jarzynski's equality <u>(Jarzynski)</u>. This fix borrows a lot from <u>fix spring</u> and <u>fix setforce</u>.

You can apply a moving spring force to a group of atoms (*tether* style) or between two groups of atoms (*couple* style). The spring can then be used in either constant velocity (*cvel*) mode or in constant force (*cfor*) mode to induce transitions in your systems. When running in *tether* style, you may need some way to fix some other part of the system (e.g. via <u>fix spring/self</u>)

The *tether* style attaches a spring between a point at a distance of R0 away from a fixed point *x,y,z* and the center of mass of the fix group of atoms. A restoring force of magnitude K (R - R0) Mi / M is applied to each atom in the group where *K* is the spring constant, Mi is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per-atom force.

In *cvel* mode the distance R is incremented or decremented monotonously according to the pulling (or pushing) velocity. In *cfor* mode a constant force is added and the actual distance in direction of the spring is recorded.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group-ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are

fix smd command   544

displaced by a vector in direction *x,y,z* with respect to each other and at a distance R0 from that displacement. Note that *x,y,z* only provides a direction and will be internally normalized. But since it represents the *absolute* displacement of group-ID2 relative to the fix group, (1,1,0) is a different spring than (-1,-1,0). For each vector component, the displacement can be described with the *auto* parameter. In this case the direction is recomputed in every step, which can be useful for steering a local process where the whole object undergoes some other change. When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

For constant velocity pulling (*cvel* mode), the running integral over the pulling force in direction of the spring is recorded and can then later be used to compute the potential of mean force (PMF) by averaging over multiple independent trajectories along the same pulling path.

Restart, fix_modify, output, run start/stop, minimize info:

The fix stores the direction of the spring, current pulling target distance and the running PMF to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

This fix computes a vector list of 7 quantities, which can be accessed by various output commands. The quantities in the vector are in this order: the x-, y-, and z-component of the pulling force, the total force in direction of the pull, the equilibrium distance of the spring, the distance between the two reference points, and finally the accumulated PMF (the sum of pulling forces times displacement).

The force is the total force on the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group-ID) or the negative of the force on the 2nd group (group-ID2). The vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix drag, fix spring, fix spring/self, fix spring/rg

Default: none

(**Izrailev**) Izrailev, Stepaniants, Isralewitz, Kosztin, Lu, Molnar, Wriggers, Schulten. Computational Molecular Dynamics: Challenges, Methods, Ideas, volume 4 of Lecture Notes in Computational Science and Engineering, pp. 39-65. Springer-Verlag, Berlin, 1998.

(**Park**) Park, Schulten, J. Chem. Phys. 120 (13), 5946 (2004)

(**Jarzynski**) Jarzynski, Phys. Rev. Lett. 78, 2690 (1997)

fix smd command 545

fix sph/density/continuity command

Syntax:

```
fix ID group-ID sph/density/continuity
```

- ID, group-ID are documented in fix command
- sph/density/continuity = style name of this fix command

Examples:

```
fix density all sph/density/continuity
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

Based on the continuity equation in the form

$$\frac{d\rho}{dt} = -\nabla \cdot (\rho \vec{v}) + \vec{v} \cdot \nabla \rho$$

this fix calculates the density of each particle by the rule

$$\frac{d\rho_a}{dt} = \sum_b m_b (\vec{v}_a - \vec{v}_b) \cdot \nabla_a W_{ab}$$

where the summation is over all particles b other than particle a, m is the mass, v is the velocity, W_{ab} is the interpolating kernel (documented in pair_style sph/artVisc/tensCorr) and $_a$ is the gradient of W_{ab}.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

None of the fix_modify options are relevant to this fix.

No global scalar or vector or per_atom quantities are stored by this fix for access by various output commands.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is not invoked during energy minimization.

Restrictions:

There can be only one fix sph/density.

Related commands:

pair_style sph/artVisc/tensCorr, fix sph/pressure, fix sph/density/corr

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(Monaghan, 1992) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

fix sph/density/continuity command

fix sph/density/corr command

Syntax:

```
fix ID group-ID sph/density/corr style args
```

- ID, group-ID are documented in fix command
- sph/density/corr = style name of this fix command
- style = *shepard*
- args = list of arguments for a particular style

```
shepard args = every nSteps
    nSteps = determes number of timesteps
```

Examples:

```
fix corr all sph/density/corr shepard every 30
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

In general the pressure field in SPH exhibits large oscillations. One approach to overcome this problem is to perform a filter over the density.

The filterstyle *shepard* is one of the most simple and quick correction. Every *nSteps* timesteps the following rule is applied:

$$\rho_a^{new} = \frac{\sum_b m_b W_{ab}}{\sum_b \frac{m_b}{\rho_b} W_{ab}}$$

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

None of the fix_modify options are relevant to this fix.

No global scalar or vector or per_atom quantities are stored by this fix for access by various output commands.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

pair_style sph/artVisc/tensCorr, fix sph/pressure, fix sph/density/continuity

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

fix sph/density/summation command

Syntax:

```
fix ID group-ID sph/density/summation
```

- ID, group-ID are documented in [fix](#) command
- sph/density/summation = style name of this fix command

Examples:

```
fix density all sph/density/summation
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

Calculates the density field with the classic SPH-summation approach. The governing equation is given by:

$$\rho_a = \sum_b m_b W_{ab}$$

rho$_a$ is the density of particle a, m is the mass and W$_{ab}$ denotes the interpolating kernel for the particle-particle distance r$_a$ - r$_b$. The summation is over all particles b other than particle a.

NOTE: In the current version boundary or image particles are not implemented. Therefore, the density calculation in the vicinity to a wall will be wrong.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

None of the [fix_modify](#) options are relevant to this fix.

No global scalar or vector or per_atom quantities are stored by this fix for access by various [output commands](#).

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

This fix is not invoked during [energy minimization](#).

Restrictions:

There can be only one fix sph/density/... (except [fix sph/density/corr](#))

Related commands:

[pair_style sph/artVisc/tensCorr](#), [fix sph/pressure](#), [fix sph/density/continuity](#)

Default: none

(Liu and Liu, 2003) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(Monaghan, 1992) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

fix sph/pressure command

Syntax:

```
fix ID group-ID sph/pressure style args
```

- ID, group-ID are documented in fix command
- sph/pressure = style name of this fix command
- style = *absolut* or *Tait*
- args = list of arguments for a particular style

```
  absolut args = NULL
    Tait args = B density0 gamma
      B = coefficient
      density0 = reference density
      gamma = isentropic exponent
```

Examples:

```
fix pressure all sph/pressure absolut
fix pressure all sph/pressure Tait 2000000. 1000. 7.
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

The equation of state (EOS) for the SPH calculation is the link between the density field and the pressure field. A lot of different equations can be found in the literature.

The *absolut* style was the first implemented EOS. Based on "An initiation to SPH" from Lucas Braune and Thomas Lewiner this simple equation

$$P_a = 0.1 \rho_a^2$$

is implemented, where ρ_a is the density of particle a.

In case of *Tait* style the rule

$$P_a = B \left[\left(\frac{\rho_a}{\rho_0} \right)^\gamma - 1 \right]$$

is applied. B denotes the pressure prefactor which is calculated by

$$B = \frac{c_0^2 \rho_0}{\gamma}$$

where c_0 is the speed of sound of the material. ρ_0 is the reference density and γ is the isentropic exponent defined as c_p/c_v.

NOTE: Monaghan has found that the speed of sound could be artificially reduced. (Monaghan, 1994) Therefore, we can choose a greater time step. He argues that the minimum sound speed should be about ten times greater than the maximum expected flow speed. (Δ < 1%)

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

None of the fix_modify options are relevant to this fix.

No global scalar or vector or per-atom quantities are stored by this fix for access by various output commands.

No parameter of this fix can be used with the *start/stop* keywords of the run command.

This fix is not invoked during energy minimization.

Restrictions:

One fix sph/density/summation (only dev-version) or sph/denstiy/continuity has to exist.

Related commands:

pair_style sph, fix sph/density/continuity

Default: none

fix spring command

Syntax:

```
fix ID group-ID spring keyword values
```

- ID, group-ID are documented in [fix](#) command
- spring = style name of this fix command
- keyword = *tether* or *couple*

```
  tether values = K x y z R0
      K = spring constant (force/distance units)
      x,y,z = point to which spring is tethered
      R0 = equilibrium distance from tether point (distance units)
    couple values = group-ID2 K x y z R0
      group-ID2 = 2nd group to couple to fix group with a spring
      K = spring constant (force/distance units)
      x,y,z = direction of spring
      R0 = equilibrium distance of spring (distance units)
```

Examples:

```
fix pull ligand spring tether 50.0 0.0 0.0 0.0 0.0
fix pull ligand spring tether 50.0 0.0 0.0 0.0 5.0
fix pull ligand spring tether 50.0 NULL NULL 2.0 3.0
fix 5 bilayer1 spring couple bilayer2 100.0 NULL NULL 10.0 0.0
fix longitudinal pore spring couple ion 100.0 NULL NULL -20.0 0.0
fix radial pore spring couple ion 100.0 0.0 0.0 NULL 5.0
```

Description:

Apply a spring force to a group of atoms or between two groups of atoms. This is useful for applying an umbrella force to a small molecule or lightly tethering a large group of atoms (e.g. all the solvent or a large molecule) to the center of the simulation box so that it doesn't wander away over the course of a long simulation. It can also be used to hold the centers of mass of two groups of atoms at a given distance or orientation with respect to each other.

The *tether* style attaches a spring between a fixed point x,y,z and the center of mass of the fix group of atoms. The equilibrium position of the spring is R0. At each timestep the distance R from the center of mass of the group of atoms to the tethering point is computed, taking account of wrap-around in a periodic simulation box. A restoring force of magnitude K (R - R0) M_i / M is applied to each atom in the group where *K* is the spring constant, M_i is the mass of the atom, and M is the total mass of all atoms in the group. Note that *K* thus represents the total force on the group of atoms, not a per-atom force.

The *couple* style links two groups of atoms together. The first group is the fix group; the second is specified by group-ID2. The groups are coupled together by a spring that is at equilibrium when the two groups are displaced by a vector x,y,z with respect to each other and at a distance R0 from that displacement. Note that x,y,z is the equilibrium displacement of group-ID2 relative to the fix group. Thus (1,1,0) is a different spring than (-1,-1,0). When the relative positions and distance between the two groups are not in equilibrium, the same spring force described above is applied to atoms in each of the two groups.

For both the *tether* and *couple* styles, any of the x,y,z values can be specified as NULL which means do not include that dimension in the distance calculation or force application.

The first example above pulls the ligand towards the point (0,0,0). The second example holds the ligand near the surface of a sphere of radius 5 around the point (0,0,0). The third example holds the ligand a distance 3 away from the z=2 plane (on either side).

The fourth example holds 2 bilayers a distance 10 apart in z. For the last two examples, imagine a pore (a slab of atoms with a cylindrical hole cut out) oriented with the pore axis along z, and an ion moving within the pore. The fifth example holds the ion a distance of -20 below the z = 0 center plane of the pore (umbrella sampling). The last example holds the ion a distance 5 away from the pore axis (assuming the center-of-mass of the pore in x,y is the pore axis).

IMPORTANT NOTE: The center of mass of a group of atoms is calculated in "unwrapped" coordinates using atom image flags, which means that the group can straddle a periodic boundary. See the dump doc page for a discussion of unwrapped coordinates. It also means that a spring connecting two groups or a group and the tether point can cross a periodic boundary and its length be calculated correctly. One exception is for rigid bodies, which should not be used with the fix spring command, if the rigid body will cross a periodic boundary. This is because image flags for rigid bodies are used in a different way, as explained on the fix rigid doc page.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy stored in the spring to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the spring energy = 0.5 * K * r^2.

This fix also computes global 4-vector which can be accessed by various output commands. The first 3 quantities in the vector are xyz components of the total force added to the group of atoms by the spring. In the case of the *couple* style, it is the force on the fix group (group-ID) or the negative of the force on the 2nd group (group-ID2). The 4th quantity in the vector is the magnitude of the force added by the spring, as a positive value if (r-R0) > 0 and a negative value if (r-R0) < 0. This sign convention can be useful when using the spring force to compute a potential of mean force (PMF).

The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix drag, fix spring/self, fix spring/rg, fix smd

Default: none

fix spring/rg command

Syntax:

```
fix ID group-ID spring/rg K RG0
```

- ID, group-ID are documented in fix command
- spring/rg = style name of this fix command
- K = harmonic force constant (force/distance units)
- RG0 = target radius of gyration to constrain to (distance units)

```
if RG0 = NULL, use the current RG as the target value
```

Examples:

```
fix 1 protein spring/rg 5.0 10.0
fix 2 micelle spring/rg 5.0 NULL
```

Description:

Apply a harmonic restraining force to atoms in the group to affect their central moment about the center of mass (radius of gyration). This fix is useful to encourage a protein or polymer to fold/unfold and also when sampling along the radius of gyration as a reaction coordinate (i.e. for protein folding).

The radius of gyration is defined as RG in the first formula. The energy of the constraint and associated force on each atom is given by the second and third formulas, when the group is at a different RG than the target value RG0.

$$R_G{}^2 = \frac{1}{M} \sum_i^N m_i \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)^2$$

$$E = K \left(R_G - R_{G0} \right)^2$$

$$F_i = 2K \frac{m_i}{M} \left(1 - \frac{R_{G0}}{R_G} \right) \left(x_i - \frac{1}{M} \sum_j^N m_j x_j \right)$$

The (xi - center-of-mass) term is computed taking into account periodic boundary conditions, m_i is the mass of the atom, and M is the mass of the entire group. Note that K is thus a force constant for the aggregate force on the group of atoms, not a per-atom force.

If RG0 is specified as NULL, then the RG of the group is computed at the time the fix is specified, and that value is used as the target.

Restart, fix_modify, output, run start/stop, minimize info:

fix spring/rg command   556

No information about this fix is written to [binary restart files](). None of the [fix_modify]() options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various [output commands](). No parameter of this fix can be used with the *start/stop* keywords of the [run]() command. This fix is not invoked during [energy minimization]().

Restrictions: none

Related commands:

[fix spring](), [fix spring/self]() [fix drag](), [fix smd]()

Default: none

fix spring/self command

Syntax:

```
fix ID group-ID spring/self K dir
```

- ID, group-ID are documented in [fix](#) command
- spring/self = style name of this fix command
- K = spring constant (force/distance units)
- dir = xyz, xy, xz, yz, x, y, or z (optional, default: xyz)

Examples:

```
fix tether boundary-atoms spring/self 10.0
fix zrest  move spring/self 10.0 z
```

Description:

Apply a spring force independently to each atom in the group to tether it to its initial position. The initial position for each atom is its location at the time the fix command was issued. At each timestep, the magnitude of the force on each atom is -Kr, where r is the displacement of the atom from its current position to its initial position. The distance r correctly takes into account any crossings of periodic boundary by the atom since it was in its intitial position.

With the (optional) dir flag, one can select in which direction the spring force is applied. By default, the restraint is applied in all directions, but it can be limited to the xy-, xz-, yz-plane and the x-, y-, or z-direction, thus restraining the atoms to a line or a plane, respectively.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of tethered atoms to [binary restart files](#), so that the spring effect will be the same in a restarted simulation. See the [read_restart](#) command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The [fix_modify](#) *energy* option is supported by this fix to add the energy stored in the per-atom springs to the system's potential energy as part of [thermodynamic output](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is an energy which is the sum of the spring energy for each atom, where the per-atom energy is $0.5 * K * r^2$. The scalar value calculated by this fix is "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize](#) command.

IMPORTANT NOTE: If you want the per-atom spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the [fix_modify](#) *energy* option for this fix.

Restrictions: none

Related commands:

[fix drag](#), [fix spring](#), [fix smd](#), [fix spring/rg](#)

fix spring/self command 558

Default: none

fix srd command

Syntax:

```
fix ID group-ID srd N groupbig-ID Tsrd hgrid seed keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- srd = style name of this fix command
- N = reset SRD particle velocities every this many timesteps
- groupbig-ID = ID of group of large particles that SRDs interact with
- Tsrd = temperature of SRD particles (temperature units)
- hgrid = grid spacing for SRD grouping (distance units)
- seed = random # seed (positive integer)

- zero or more keyword/value pairs may be appended
- keyword = *lamda* or *collision* or *overlap* or *inside* or *exact* or *radius* or *bounce* or *search* or *cubic* or *shift* or *tstat* or *rescale*

```
  lamda value = mean free path of SRD particles (distance units)
    collision value = noslip or slip = collision model
    overlap value = yes or no = whether big particles may overlap
    inside value = error or warn or ignore = how SRD particles which end up inside a big par
    exact value = yes or no
    radius value = rfactor = scale collision radius by this factor
    bounce value = Nbounce = max # of collisions an SRD particle can undergo in one timestep
    search value = sgrid = grid spacing for collision partner searching (distance units)
    cubic values = style tolerance
      style = error or warn
      tolerance = fractional difference allowed (0 <= tol <= 1)
    shift values = flag shiftseed
      flag = yes or no or possible = SRD bin shifting for better statistics
        yes = perform bin shifting each time SRD velocities are rescaled
        no = no shifting
        possible = shift depending on mean free path and bin size
      shiftseed = random # seed (positive integer)
    tstat value = yes or no = thermostat SRD particles or not
    rescale value = yes or no or rotate or collide = rescaling of SRD velocities
      yes = rescale during velocity rotation and collisions
      no = no rescaling
      rotate = rescale during velocity rotation, but not collisions
      collide = rescale during collisions, but not velocity rotation
```

Examples:

```
fix 1 srd srd 10 big 1.0 0.25 482984
fix 1 srd srd 10 big 0.5 0.25 482984 collision slip search 0.5
```

Description:

Treat a group of partilces as stochastic rotation dynamics (SRD) particles that serve as a background solvent when interacting with big (colloidal) particles in groupbig-ID. The SRD formalism is described in [(Hecht)](#). The key idea behind using SRD particles as a cheap coarse-grained solvent is that SRD particles do not interact with each other, but only with the solute particles, which in LAMMPS can be spheroids, ellipsoids, or line segments, or triangles, or rigid bodies containing multiple spherioids or ellipsoids or line segments or triangles. The collision and rotation properties of the model imbue the SRD particles with fluid-like properties, including an effective viscosity. Thus simulations with large solute particles can be run more quickly, to measure solute propoerties like diffusivity and viscosity in a background fluid. The usual

fix srd command   560

LAMMPS fixes for such simulations, such as fix deform, fix viscosity, and fix nvt/sllod, can be used in conjunction with the SRD model.

For more details on how the SRD model is implemented in LAMMPS, this paper describes the implementation and usage of pure SRD fluids. This paper, which is nearly complete, describes the implementation and usage of mixture systems (solute particles in an SRD fluid). See the examples/srd directory for sample input scripts using SRD particles in both settings.

This fix does 2 things:

(1) It advects the SRD particles, performing collisions between SRD and big particles or walls every timestep, imparting force and torque to the big particles. Collisions also change the position and velocity of SRD particles.

(2) It resets the velocity distribution of SRD particles via random rotations every N timesteps.

SRD particles have a mass, temperature, characteristic timestep dt_SRD, and mean free path between collisions (lamda). The fundamental equation relating these 4 quantities is

```
lamda = dt_SRD * sqrt(Kboltz * Tsrd / mass)
```

The mass of SRD particles is set by the mass command elsewhere in the input script. The SRD timestep dt_SRD is N times the step dt defined by the timestep command. Big particles move in the normal way via a time integration fix with a short timestep dt. SRD particles advect with a large timestep dt_SRD >= dt.

If the *lamda* keyword is not specified, the the SRD temperature *Tsrd* is used in the above formula to compute lamda. If the *lamda* keyword is specified, then the *Tsrd* setting is ignored and the above equation is used to compute the SRD temperature.

The characteristic length scale for the SRD fluid is set by *hgrid* which is used to bin SRD particles for purposes of resetting their velocities. Normally hgrid is set to be 1/4 of the big particle diameter or smaller, to adequately resolve fluid properties around the big particles.

Lamda cannot be smaller than 0.6 * hgrid, else an error is generated (unless the *shift* keyword is used, see below). The velocities of SRD particles are bounded by Vmax, which is set so that an SRD particle will not advect further than Dmax = 4*lamda in dt_SRD. This means that roughly speaking, Dmax should not be larger than a big particle diameter, else SRDs may pass thru big particles without colliding. A warning is generated if this is the case.

Collisions between SRD particles and big particles or walls are modeled as a lightweight SRD point particle hitting a heavy big particle of given diameter or a wall at a point on its surface and bouncing off with a new velocity. The collision changes the momentum of the SRD particle. It imparts a force and torque to the big particle. It imparts a force to a wall. Static or moving SRD walls are setup via the fix wall/srd command. For the remainder of this doc page, a collision of an SRD particle with a wall can be viewed as a collision with a big particle of infinite radius and mass.

The *collision* keyword sets the style of collisions. The *slip* style means that the tangential component of the SRD particle momentum is preserved. Thus a force is imparted to a big particle, but no torque. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature *Tsrd*.

For the *noslip* style, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature *Tsrd*. Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts torque to a big particle. Thus a time integrator fix that rotates the big particles appropriately should be used.

The *overlap* keyword should be set to *yes* if two (or more) big particles can ever overlap. This depends on the pair potential interaction used for big-big interactions, or could be the case if multiple big particles are held together as rigid bodies via the [fix rigid](#) command. If the *overlap* keyword is *no* and big particles do in fact overlap, then SRD/big collisions can generate an error if an SRD ends up inside two (or more) big particles at once. How this error is treated is determined by the *inside* keyword. Running with *overlap* set to *no* allows for faster collision checking, so it should only be set to *yes* if needed.

The *inside* keyword determines how a collision is treated if the computation determines that the timestep started with the SRD particle already inside a big particle. If the setting is *error* then this generates an error message and LAMMPS stops. If the setting is *warn* then this generates a warning message and the code continues. If the setting is *ignore* then no message is generated. One of the output quantities logged by the fix (see below) tallies the number of such events, so it can be monitored. Note that once an SRD particle is inside a big particle, it may remain there for several steps until it drifts outside the big particle.

The *exact* keyword determines how accurately collisions are computed. A setting of *yes* computes the time and position of each collision as SRD and big particles move together. A setting of *no* estimates the position of each collision based on the end-of-timestep positions of the SRD and big particle. If *overlap* is set to yes, the setting of the *exact* keyword is ignored since time-accurate collisions are needed.

The *radius* keyword scales the effective size of big particles. If big particles will overlap as they undergo dynamics, then this keyword can be used to scale down their effective collision radius by an amount *rfactor*, so that SRD particle will only collide with one big particle at a time. For example, in a Lennard-Jones system at a temperature of 1.0 (in reduced LJ units), the minimum separation bewteen two big particles is as small as about 0.88 sigma. Thus an *rfactor* value of 0.85 should prevent dual collisions.

The *bounce* keyword can be used to limit the maximum number of collisions an SRD particle undergoes in a single timestep as it bounces between nearby big particles. Note that if the limit is reached, the SRD can be left inside a big particle. A setting of 0 is the same as no limit.

There are 2 kinds of bins created and maintained when running an SRD simulation. The first are "SRD bins" which are used to bin SRD particles and reset their velocities, as discussed above. The second are "search bins" which are used to identify SRD/big particle collisions.

The *search* keyword can be used to choose a search bin size for identifying SRD/big particle collisions. The default is to use the *hgrid* parameter for SRD bins as the search bin size. Choosing a smaller or large value may be more efficient, depending on the problem. But, in a statistical sense, it should not change the simulation results.

The *cubic* keyword can be used to generate an error or warning when the bin size chosen by LAMMPS creates SRD bins that are non-cubic or different than the requested value of *hgrid* by a specified *tolerance*. Note that using non-cubic SRD bins can lead to undetermined behavior when rotating the velocities of SRD particles, hence LAMMPS tries to protect you from this problem.

LAMMPS attempts to set the SRD bin size to exactly *hgrid*. However, there must be an integer number of bins in each dimension of the simulation box. Thus the actual bin size will depend on the size and shape of the overall simulation box. The actual bin size is printed as part of the SRD output when a simulation begins.

If the actual bin size in non-cubic by an amount exceeding the tolerance, an error or warning is printed, depending on the style of the *cubic* keyword. Likewise, if the actual bin size differs from the requested *hgrid* value by an amount exceeding the tolerance, then an error or warning is printed. The *tolerance* is a fractional difference. E.g. a tolerance setting of 0.01 on the shape means that if the ratio of any 2 bin dimensions exceeds (1 +/- tolerance) then an error or warning is generated. Similarly, if the ratio of any bin dimension with *hgrid* exceeds (1 +/- tolerance), then an error or warning is generated.

IMPORTANT NOTE: The fix srd command can be used with simluations the size and/or shape of the simulation box changes. This can be due to non-periodic boundary conditions or the use of fixes such as the [fix deform](#) or [fix wall/srd](#) commands to impose a shear on an SRD fluid or an interaction with an external wall. If the box size changes then the size of SRD bins must be recalculated every reneighboring. This is not necessary if only the box shape changes. This re-binning is always done so as to fit an integer number of bins in the current box dimension, whether it be a fixed, shrink-wrapped, or periodic boundary, as set by the [boundary](#) command. If the box size or shape changes, then the size of the search bins must be recalculated avery reneighboring. Note that changing the SRD bin size may alter the properties of the SRD fluid, such as its viscosity.

The *shift* keyword determines whether the coordinates of SRD particles are randomly shifted when binned for purposes of rotating their velocities. When no shifting is performed, SRD particles are binned and the velocity distribution of the set of SRD particles in each bin is adjusted via a rotation operator. This is a statistically valid operation if SRD particles move sufficiently far between successive rotations. This is determined by their mean-free path lamda. If lamda is less than 0.6 of the SRD bin size, then shifting is required. A shift means that all of the SRD particles are shifted by a vector whose coordinates are chosen randomly in the range [-1/2 bin size, 1/2 bin size]. Note that all particles are shifted by the same vector. The specified random number *shiftseed* is used to generate these vectors. This operation sufficiently randomizes which SRD particles are in the same bin, even if lamda is small.

If the *shift* flag is set to *no*, then no shifting is performed, but bin data will be communicated if bins overlap processor boundaries. An error will be generated if lamda < 0.6 of the SRD bin size. If the *shift* flag is set to *possible*, then shifting is performed only if lamda < 0.6 of the SRD bin size. A warning is generated to let you know this is occurring. If the *shift* flag is set to *yes* then shifting is performed regardless of the magnitude of lamda. Note that the *shiftseed* is not used if the *shift* flag is set to *no*, but must still be specified.

Note that shifting of SRD coordinates requires extra communication, hence it should not normally be enabled unless required.

The *tstat* keyword will thermostat the SRD particles to the specified *Tsrd*. This is done every N timesteps, during the velocity rotation operation, by rescaling the thermal velocity of particles in each SRD bin to the desired temperature. If there is a streaming velocity associated with the system, e.g. due to use of the [fix deform](#) command to perform a simulation undergoing shear, then that is also accounted for. The mean velocity of each bin of SRD particles is set to the position-dependent streaming velocity, based on the coordinates of the center of the SRD bin. Note that collisions of SRD particles with big particles or walls has a thermostatting effect on the colliding particles, so it may not be necessary to thermostat the SRD particles on a bin by bin basis in that case. Also note that for streaming simulations, if no thermostatting is performed (the default), then it may take a long time for the SRD fluid to come to equilibrium with a velocity profile that matches the simulation box deformation.

The *rescale* keyword enables rescaling of an SRD particle's velocity if it would travel more than 4 mean-free paths in an SRD timestep. If an SRD particle exceeds this velocity it is possible it will be lost when migrating to other processors or that collisions with big particles will be missed, either of which will generate errors. Thus the safest mode is to run with rescaling enabled. However rescaling removes kinetic energy from the system (the particle's velocity is reduced). The latter will not typically be a problem if thermostatting is enabled via the *tstat* keyword or if SRD collisions with big particles or walls effectively thermostat the system. If you wish to turn off rescaling (on is the default), e.g. for a pure SRD system with no thermostatting so that the temperature does not decline over time, the *rescale* keyword can be used. The *no* value turns rescaling off during collisions and the per-bin velocity rotation operation. The *collide* and *rotate* values turn it on for one of the operations and off for the other.

IMPORTANT NOTE: This fix is normally used for simulations with a huge number of SRD particles relative to the number of big particles, e.g. 100 to 1. In this scenario, computations that involve only big particles (neighbor list creation, communication, time integration) can slow down dramatically due to the large number

of background SRD particles.

Three other input script commands will largely overcome this effect, speeding up an SRD simulation by a significant amount. These are the atom_modify first, neigh_modify include, and communicate group commands. Each takes a group-ID as an argument, which in this case should be the group-ID of the big solute particles.

Additionally, when a pair_style for big/big particle interactions is specified, the pair_coeff command should be used to turn off big/SRD interactions, e.g. by setting their epsilon or cutoff length to 0.0.

The "delete_atoms overlap" command may be useful in setting up an SRD simulation to insure there are no initial overlaps between big and SRD particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix tabulates several SRD statistics which are stored in a vector of length 12, which can be accessed by various output commands. The vector values calculated by this fix are "intensive", meaning they do not scale with the size of the simulation. Technically, the first 8 do scale with the size of the simulation, but treating them as intensive means they are not scaled when printed as part of thermodyanmic output.

These are the 12 quantities. All are values for the current timestep, except for quantity 5 and the last three, each of which are cummulative quantities since the beginning of the run.

- (1) # of SRD/big collision checks performed
- (2) # of SRDs which had a collision
- (3) # of SRD/big colllisions (including multiple bounces)
- (4) # of SRD particles inside a big particle
- (5) # of SRD particles whose velocity was rescaled to be < Vmax
- (6) # of bins for collision searching
- (7) # of bins for SRD velocity rotation
- (8) # of bins in which SRD temperature was computed
- (9) SRD temperature
- (10) # of SRD particles which have undergone max # of bounces
- (11) max # of bounces any SRD particle has had in a single step
- (12) # of reneighborings due to SRD particles moving too far

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This command can only be used if LAMMPS was built with the SRD package. See the Making LAMMPS section for more info on packages.

Related commands:

fix wall/srd

Default:

The option defaults are lamda inferred from Tsrd, collision = noslip, overlap = no, inside = error, exact = yes,

radius = 1.0, bounce = 0, search = hgrid, cubic = error 0.01, shift = no, tstat = no, and rescale = yes.

(Hecht) Hecht, Harting, Ihle, Herrmann, Phys Rev E, 72, 011408 (2005).

(Petersen) Petersen, Lechman, Plimpton, Grest, in' t Veld, Schunk, J Chem Phys, 132, 174106 (2010).

(Lechman) Lechman, et al, in preparation (2010).

fix store/force command

Syntax:

```
fix ID group-ID store/force
```

- ID, group-ID are documented in [fix](#) command
- store/force = style name of this fix command

Examples:

```
fix 1 all store/force
```

Description:

Store the forces on atoms in the group at the point during each timestep when the fix is invoked, as described below. This is useful for storing forces before constraints or other boundary conditions are computed which modify the forces, so that unmodified forces can be [written to a dump file](#) or accessed by other [output commands](#) that use per-atom quantities.

This fix is invoked at the point in the velocity-Verlet timestepping immediately after [pair](#), [bond](#), [angle](#), [dihedral](#), [improper](#), and [long-range](#) forces have been calculated. It is the point in the timestep when various fixes that compute constraint forces are calculated and potentially modify the force on each atom. Examples of such fixes are [fix shake](#), [fix wall](#), and [fix indent](#).

IMPORTANT NOTE: The order in which various fixes are applied which operate at the same point during the timestep, is the same as the order they are specified in the input script. Thus normally, if you want to store per-atom forces due to force field interactions, before constraints are applied, you should list this fix first within that set of fixes, i.e. before other fixes that apply constraints. However, if you wish to include certain constraints (e.g. fix shake) in the stored force, then it could be specified after some fixes and before others.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#). None of the [fix_modify](#) options are relevant to this fix.

This fix produces a per-atom array which can be accessed by various [output commands](#). The number of columns for each atom is 3, and the columns store the x,y,z forces on each atom. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix store_state](#)

Default: none

fix store/state command

Syntax:

```
fix ID group-ID store/state N input1 input2 ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- store/state = style name of this fix command
- N = store atom attributes every N steps, N = 0 for initial store only
- input = one or more atom attributes

```
possible attributes = id, mol, type, mass,
                      x, y, z, xs, ys, zs, xu, yu, zu, ix, iy, iz,
                      vx, vy, vz, fx, fy, fz,
                      q, mux, muy, muz,
                      radius, omegax, omegay, omegaz,
                      angmomx, angmomy, angmomz, tqx, tqy, tqz
                      c_ID, c_ID[N], f_ID, f_ID[N], v_name

      id = atom ID
      mol = molecule ID
      type = atom type
      mass = atom mass
      x,y,z = unscaled atom coordinates
      xs,ys,zs = scaled atom coordinates
      xu,yu,zu = unwrapped atom coordinates
      ix,iy,iz = box image that the atom is in
      vx,vy,vz = atom velocities
      fx,fy,fz = forces on atoms
      q = atom charge
      mux,muy,muz = orientation of dipolar atom
      radius = radius of spherical particle
      omegax,omegay,omegaz = angular velocity of spherical particle
      angmomx,angmomy,angmomz = angular momentum of aspherical particle
      tqx,tqy,tqz = torque on finite-size particles
      c_ID = per-atom vector calculated by a compute with ID
      c_ID[I] = Ith column of per-atom array calculated by a compute with ID
      f_ID = per-atom vector calculated by a fix with ID
      f_ID[I] = Ith column of per-atom array calculated by a fix with ID
      v_name = per-atom vector calculated by an atom-style variable with name
```

- zero or more keyword/value pairs may be appended
- keyword = *com*

```
  com value = yes or no
```

Examples:

```
fix 1 all store/state 0 x y z
fix 1 all store/state 0 xu yu zu com yes
fix 2 all store/state 1000 vx vy vz
```

Description:

Define a fix that stores attributes for each atom in the group at the time the fix is defined. If *N* is 0, then the values are never updated, so this is a way of archiving an atom attribute at a given time for future use in a calculation or output. See the discussion of [output commands](#) that take fixes as inputs. And see for example, the [compute reduce](#), [fix ave/atom](#), [fix ave/histo](#), [fix ave/spatial](#), and [atom-style variable](#) commands.

If *N* is not zero, then the attributes will be updated every *N* steps.

IMPORTANT NOTE: Actually, only atom attributes specified by keywords like *xu* or *vy* are initially stored immediately at the point in your input script when the fix is defined. Attributes specified by a compute, fix, or variable are not initially stored until the first run following the fix definition begins. This is because calculating those attributes may require quantities that are not defined in between runs.

The list of possible attributes is the same as that used by the dump custom command, which describes their meaning.

If the *com* keyword is set to *yes* then the *xu*, *yu*, and *zu* inputs store the position of each atom relative to the center-of-mass of the group of atoms, instead of storing the absolute position. This option is used by the compute msd command.

The requested values are stored in a per-atom vector or array as discussed below. Zeroes are stored for atoms not in the specified group.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the per-atom values it stores to binary restart files, so that the values can be restored when a simulation is restarted. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

None of the fix_modify options are relevant to this fix.

If a single input is specified, this fix produces a per-atom vector. If multiple inputs are specified, a per-atom array is produced where the number of columns for each atom is the number of inputs. These can be accessed by various output commands. The per-atom values be accessed on any timestep.

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

dump custom, compute property/atom, variable

Default:

The option default is com = no.

fix temp/berendsen command

fix temp/berendsen/cuda command

Syntax:

```
fix ID group-ID temp/berendsen Tstart Tstop Tdamp
```

- ID, group-ID are documented in <u>fix</u> command
- temp/berendsen = style name of this fix command
- Tstart,Tstop = desired temperature at start/end of run

  ```
  Tstart can be a variable (see below)
  ```
- Tdamp = temperature damping parameter (time units)

Examples:

```
fix 1 all temp/berendsen 300.0 300.0 100.0
```

Description:

Reset the temperature of a group of atoms by using a Berendsen thermostat <u>(Berendsen)</u>, which rescales their velocities every timestep.

The thermostat is applied to only the translational degrees of freedom for the particles, which is an important consideration for finite-size particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

The desired temperature at each timestep is a ramped value during the run from *Tstart* to *Tstop*. The *Tdamp* parameter is specified in time units and determines how rapidly the temperature is relaxed. For example, a value of 100.0 means to relax the temperature in a timespan of (roughly) 100 time units (tau or fmsec or psec - see the <u>units</u> command).

Tstart can be specified as an equal-style <u>variable</u>. In this case, the *Tstop* setting is ignored. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the target temperature.

Equal-style variables can specify formulas with various mathematical functions, and include <u>thermo_style</u> command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent temperature.

IMPORTANT NOTE: Unlike the <u>fix nvt</u> command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like <u>fix nve</u> to actually update the positions of atoms using the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by <u>fix nvt</u> or <u>fix langevin</u> commands.

See <u>this howto section</u> of the manual for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp", as

if this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the [compute temp](#) command for details. Note that the ID of the new compute is the fix-ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *temp* option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of [thermodynamic output](#).

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions: none

Related commands:

[fix nve](fix nve), [fix nvt](fix nvt), fix temp/rescale, fix langevin, fix_modify, compute temp, fix press/berendsen

Default: none

(Berendsen) Berendsen, Postma, van Gunsteren, DiNola, Haak, J Chem Phys, 81, 3684 (1984).

LIGGGHTS Users Manual

LAMMPS WWW Site - LAMMPS Documentation - LAMMPS Commands

fix temp/rescale/eff command

Syntax:

```
fix ID group-ID temp/rescale/eff N Tstart Tstop window fraction
```

- ID, group-ID are documented in fix command
- temp/rescale/eff = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

Examples:

```
fix 3 flow temp/rescale/eff 10 1.0 100.0 0.02 1.0
```

Description:

Reset the temperature of a group of nuclei and electrons in the electron force field model by explicitly rescaling their velocities.

The operation of this fix is exactly like that described by the fix temp/rescale command, except that the rescaling is also applied to the radial electron velocity for electron particles.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *temp* option is supported by this fix. You can use it to assign a temperature compute you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The fix_modify *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the USER-EFF package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

fix langevin/eff, fix nvt/eff, fix_modify, fix_temp_rescale,

fix temp/rescale/eff command

572

Default: none

fix temp/rescale command

fix temp/rescale/cuda command

fix temp/rescale/limit/cuda command

Syntax:

```
fix ID group-ID temp/rescale N Tstart Tstop window fraction
```

- ID, group-ID are documented in [fix](#) command
- temp/rescale = style name of this fix command
- N = perform rescaling every N steps
- Tstart,Tstop = desired temperature at start/end of run (temperature units)

     Tstart can be a variable (see below)
- window = only rescale if temperature is outside this window (temperature units)
- fraction = rescale to target temperature by this fraction

Examples:

```
fix 3 flow temp/rescale 100 1.0 1.1 0.02 0.5
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
fix 3 boundary temp/rescale 1 1.0 1.5 0.05 1.0
```

Description:

Reset the temperature of a group of atoms by explicitly rescaling their velocities.

The rescaling is applied to only the translational degrees of freedom for the particles, which is an important consideration if finite-size particles which have rotational degrees of freedom are being thermostatted with this fix. The translational degrees of freedom can also have a bias velocity removed from them before thermostatting takes place; see the description below.

Rescaling is performed every N timesteps. The target temperature is a ramped value between the *Tstart* and *Tstop* temperatures at the beginning and end of the run.

Tstart can be specified as an equal-style [variable](#). In this case, the *Tstop* setting is ignored. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the target temperature.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent temperature.

Rescaling is only performed if the difference between the current and desired temperatures is greater than the *window* value. The amount of rescaling that is applied is a *fraction* (from 0.0 to 1.0) of the difference between the actual and desired temperature. E.g. if *fraction* = 1.0, the temperature is reset to exactly the desired value.

IMPORTANT NOTE: Unlike the [fix nvt](#) command which performs Nose/Hoover thermostatting AND time integration, this fix does NOT perform time integration. It only modifies velocities to effect thermostatting. Thus you must use a separate time integration fix, like [fix nve](#) to actually update the positions of atoms using

the modified velocities. Likewise, this fix should not normally be used on atoms that also have their temperature controlled by another fix - e.g. by [fix nvt](#) or [fix langevin](#) commands.

See [this howto section](#) of the manual for a discussion of different ways to compute temperature and perform thermostatting.

This fix computes a temperature each timestep. To do this, the fix creates its own compute of style "temp", as if one of this command had been issued:

```
compute fix-ID_temp group-ID temp
```

See the [compute temp](#) for details. Note that the ID of the new compute is the fix-ID + underscore + "temp", and the group for the new compute is the same as the fix group.

Note that this is NOT the compute used by thermodynamic output (see the [thermo_style](#) command) with ID = *thermo_temp*. This means you can change the attributes of this fix's temperature (e.g. its degrees-of-freedom) via the [compute_modify](#) command or print this temperature during thermodynamic output via the [thermo_style custom](#) command using the appropriate compute-ID. It also means that changing attributes of *thermo_temp* will have no effect on this fix.

Like other fixes that perform thermostatting, this fix can be used with [compute commands](#) that calculate a temperature after removing a "bias" from the atom velocities. E.g. removing the center-of-mass velocity from a group of atoms or only calculating temperature on the x-component of velocity or only calculating temperature for atoms in a geometric region. This is not done by default, but only if the [fix_modify](#) command is used to assign a temperature compute to this fix that includes such a bias term. See the doc pages for individual [compute commands](#) to determine which ones include a bias. In this case, the thermostat works in the following manner: the current temperature is calculated taking the bias into account, bias is removed from each atom, thermostatting is performed on the remaining thermal degrees of freedom, and the bias is added back in.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to [binary restart files](#).

The [fix_modify](#) *temp* option is supported by this fix. You can use it to assign a temperature [compute](#) you have defined to this fix which will be used in its thermostatting procedure, as described above. For consistency, the group used by this fix and by the compute should be the same.

The [fix_modify](#) *energy* option is supported by this fix to add the energy change implied by a velocity rescaling to the system's potential energy as part of [thermodynamic output](#).

fix temp/rescale/limit/cuda command 575

This fix computes a global scalar which can be accessed by various [output commands](#). The scalar is the cummulative energy change due to this fix. The scalar value calculated by this fix is "extensive".

This fix can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](#) command. See the [run](#) command for details of how to do this.

This fix is not invoked during [energy minimization](#).

Restrictions: none

Related commands:

[fix langevin](#), [fix nvt](#), [fix_modify](#)

Default: none

fix thermal/conductivity command

Syntax:

```
fix ID group-ID thermal/conductivity N edim Nbin keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- thermal/conductivity = style name of this fix command
- N = perform kinetic energy exchange every N steps
- edim = *x* or *y* or *z* = direction of kinetic energy transfer
- Nbin = # of layers in edim direction (must be even number)
- zero or more keyword/value pairs may be appended
- keyword = *swap*

    swap value = Nswap = number of swaps to perform every N steps

Examples:

```
fix 1 all thermal/conductivity 100 z 20
fix 1 all thermal/conductivity 50 z 20 swap 2
```

Description:

Use the Muller-Plathe algorithm described in [this paper](#) to exchange kinetic energy between two particles in different regions of the simulation box every N steps. This induces a temperature gradient in the system. As described below this enables a thermal conductivity of the fluid to be calculated. This algorithm is sometimes called a reverse non-equilibrium MD (reverse NEMD) approach to computing thermal conductivity. This is because the usual NEMD approach is to impose a temperature gradient on the system and measure the response as the resulting heat flux. In the Muller-Plathe method, the heat flux is imposed, and the temperature gradient is the system's response.

See the [compute heat/flux](#) command for details on how to compute thermal conductivity in an alternate way, via the Green-Kubo formalism.

The simulation box is divided into *Nbin* layers in the *edim* direction, where the layer 1 is at the low end of that dimension and the layer *Nbin* is at the high end. Every N steps, Nswap pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. The hottest Nswap atoms in layer 1 are selected. Similarly, the coldest Nswap atoms in the "middle" layer (see below) are selected. The two sets of Nswap atoms are paired up and their velocities are exchanged. This effectively swaps their kinetic energies, assuming their masses are the same. Over time, this induces a temperature gradient in the system which can be measured using commands such as the following, which writes the temperature profile (assuming z = edim) to the file tmp.profile:

```
compute    ke all ke/atom
variable   temp atom c_ke/1.5
fix        3 all ave/spatial 10 100 1000 z lower 0.05 v_temp &
              file tmp.profile units reduced
```

Note that by default, Nswap = 1, though this can be changed by the optional *swap* keyword. Setting this parameter appropriately, in conjunction with the swap rate N, allows the heat flux to be adjusted across a wide range of values, and the kinetic energy to be exchanged in large chunks or more smoothly.

The "middle" layer for velocity swapping is defined as the *Nbin*/2 + 1 layer. Thus if *Nbin* = 20, the two swapping layers are 1 and 11. This should lead to a symmetric temperature profile since the two layers are

separated by the same distance in both directions in a periodic sense. This is why *Nbin* is restricted to being an even number.

As described below, the total kinetic energy transferred by these swaps is computed by the fix and can be output. Dividing this quantity by time and the cross-sectional area of the simulation box yields a heat flux. The ratio of heat flux to the slope of the temperature profile is proportional to the thermal conductivity of the fluid, in appropriate units. See the Muller-Plathe paper for details.

IMPORTANT NOTE: If your system is periodic in the direction of the heat flux, then the flux is going in 2 directions. This means the effective heat flux in one direction is reduced by a factor of 2. You will see this in the equations for thermal conductivity (kappa) in the Muller-Plathe paper. LAMMPS is simply tallying kinetic energy which does not account for whether or not your system is periodic; you must use the value appropriately to yield a kappa for your system.

IMPORTANT NOTE: After equilibration, if the temperature gradient you observe is not linear, then you are likely swapping energy too frequently and are not in a regime of linear response. In this case you cannot accurately infer a thermal conductivity and should try increasing the Nevery parameter.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative kinetic energy transferred between the bottom and middle of the simulation box (in the *edim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumlates thereafter, once every N steps. The units of the quantity are energy; see the units command for details. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non-swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap the kinetic energy of atoms that are in constrained molecules, e.g. via fix shake or fix rigid. This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the Zhang paper for a discussion and results of this idea.

When running a simulation with large, massive particles or molecules in a background solvent, you may want to only exchange kinetic energy bewteen solvent particles.

Related commands:

fix ave/spatial, fix viscosity, compute heat/flux

Default:

fix thermal/conductivity command 578

The option defaults are swap = 1.

(Muller-Plathe) Muller-Plathe, J Chem Phys, 106, 6082 (1997).

(Zhang) Zhang, Lussetti, de Souza, Muller-Plathe, J Phys Chem B, 109, 15060-15067 (2005).

fix ti/rs command

Syntax:

```
fix ID group-ID ti/rs lambda_initial lambda_final t_switch t_equil keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- ti/rs = style name of this fix command
- lambda_initial/lambda_final = initial/final values of the coupling parameter
- t_switch/t_equil = number of steps of the switching/equilibration procedure
- keyword = *function*

```
  function value = function-ID
      function-ID = ID of the switching function (1, 2 or 3)
```

Example:

```
fix ref all ti/rs 50.0 2000 1000
fix vf vacancy ti/rs 10.0 70000 50000 function 2
```

Description:

This fix allows you to compute the free energy temperature dependence by performing a thermodynamic integration procedure known as Reversible Scaling [(de Koning99,](#) [de Koning00a)](#). The thermodynamic integration is performed using the nonequilibrium method of Adiabatic Switching [(Watanabe,](#) [de Koning96)](#).

The forces on the atoms are dynamically scaled during the simulation, the rescaling is done in the following manner:

$$F_{\text{total}} = \lambda F_{\text{int}}$$

where F_int is the total force on the atoms due to the interatomic potential and lambda is the coupling parameter of the thermodynamic integration.

The fix acts as follows: during the first *t_equil* steps after the fix is defined the value of lambda is *lambda_initial* , this is the period to equilibrate the system in the lambda = *lambda_initial* state. After this the value of lambda changes continuously from *lambda_initial* to *lambda_final* according to the function defined using the keyword *function* (described below), this is done in *t_switch* steps. Then comes the second equilibration period of *t_equil* to equilibrate the system in the lambda = *lambda_final* state. After that the switching back to the lambda = *lambda_initial* state is done using *t_switch* timesteps and following the same switching function. After this period the value of lambda is kept equal to *lambda_initial* indefinitely or until a [unfix](#) erase the fix.

The description of thermodynamic integration in both directions is done in [de Koning00b](#), the main reason is to try to eliminate the dissipated heat due to the nonequilibrium process.

The *function* keyword allows the use of three different switching rates. The option *1* results in a constant rescaling where the lambda parameter changes at a constant rate during the switching time according to the switching function

$$\lambda(\tau) = \lambda_i + \tau \left(\lambda_f - \lambda_i \right)$$

where tau is the scaled time variable t/t_switch. This switching function has the characteristic that the temperature scaling is faster at temperatures closer to the final temperature of the procedure. The option number *2* performs the switching at a rate defined by the following switching function

$$\lambda(\tau) = \frac{\lambda_i}{1 + \tau \left(\frac{\lambda_i}{\lambda_f} - 1 \right)}$$

This switching function has the characteristic that the temperature scaling occurs at a constant rate during all the procedure. The option number *3* performs the switching at a rate defined by the following switching function

$$\lambda(\tau) = \frac{\lambda_i}{1 + \log_2(1 + \tau) \left(\frac{\lambda_i}{\lambda_f} - 1 \right)}$$

This switching function has the characteristic that the temperature scaling is faster at temperatures closer to the initial temperature of the procedure.

An example script using this command is provided in the examples/USER/misc/ti directory.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

This fix computes a global vector quantitie which can be accessed by various output commands. The vector has 2 positions, the first one is the coupling parameter lambda and the second one is the time derivative of lambda. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

Related commands:

fix ti/spring

Restrictions:

This command is part of the USER-MISC package. It is only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

Default:

fix ti/rs command   581

The keyword default is function = 1.

(de Koning 99) M. de Koning, A. Antonelli and S. Yip, Phys Rev Lett, 83, 3973 (1999).

(Watanabe) M. Watanabe and W. P. Reinhardt, Phys Rev Lett, 65, 3301 (1990).

(de Koning 96) M. de Koning and A. Antonelli, Phys Rev E, 53, 465 (1996).

(de Koning 00a) M. de Koning, A. Antonelli and S. Yip, J Chem Phys, 115, 11025 (2000).

(de Koning 00b) M. de Koning et al., Computing in Science & Engineering, 2, 88 (2000).

fix ti/spring command

Syntax:

```
fix ID group-ID ti/spring K t_switch t_equil keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- ti/spring = style name of this fix command
- K = spring constant (force/distance units)
- t_switch/t_equil = number of steps of the switching/equilibration procedure
- zero or more keyword/value pairs may be appended to args
- keyword = *function*

```
  function value = function-ID
      function-ID = ID of the switching function (1 or 2)
```

Example:

```
fix ref all ti/spring 50.0 2000 1000 function 2
```

Description:

This fix allows you to compute the free energy of solids by performing a thermodynamic integration between the solid of interest and an Einstein crystal [(Frenkel)](#). The thermodynamic integration is performed using the nonequilibrium method of Adiabatic Switching [(Watanabe,](#) [de Koning96)](#).

A spring force is applied independently to each atom in the group to tether it to its initial position. The initial position for each atom is its location at the time the fix command was issued. More details about the springs are available in [fix spring/self](#). The forces on the atoms are dynamically scaled during the simulation, the rescaling is done in the following manner:

$$F_{\text{total}} = (1 - \lambda)\, F_{\text{solid}} + \lambda F_{\text{harm}}$$

where F_harm is the force due to the springs, F_solid is the total force on the atoms due to the interatomic potential and lambda is the coupling parameter of the thermodynamic integration.

The fix acts as follows: during the first *t_equil* steps after the fix is defined the value of lambda is zero, this is the period to equilibrate the system in the lambda = 0 state. After this the value of lambda changes continuously from 0 to 1 according to the function defined using the keyword *function* (described below), this is done in *t_switch* steps. Then comes the second equilibration period of *t_equil* to equilibrate the system in the lambda = 1 state. After that the switching back to the lambda = 0 state is made using *t_switch* timesteps and following the same switching function. After this period the value of lambda is kept equal to zero and the fix has no action in the dynamics of the system anymore.

The description of thermodynamic integration in both directions is done in [de Koning97](#), the main reason is to try to eliminate the dissipated heat due to the nonequilibrium process.

The *function* keyword allows the use of two different switching rates, the option *1* results in a constant rescaling where the lambda parameter changes at a constant rate during the switching time according to the

fix ti/spring command

switching function

$$\lambda(\tau) = \tau$$

where tau is the scaled time variable t/t_switch. The option number *2* performs the switching at a rate defined by the following switching function

$$\lambda(\tau) = \tau^5 \left(70\tau^4 - 315\tau^3 + 540\tau^2 - 420\tau + 126 \right)$$

This function has zero slope as lambda approaches its extreme values (0 and 1), according to (de Koning96) this results in smaller fluctuations on the integral to be computed on the thermodynamic integration.

IMPORTANT NOTE: It is importante to keep the center of mass fixed during the thermodynamic integration, a non-zero total velocity will result in divergencies during the integration due to the fact that the atoms are 'attatched' to its equilibrium positions by the Einstein crystal. Check the option *zero* of fix langevin and velocity. The use of the Nose-Hoover thermostat (fix nvt) is NOT recommended due to its well documented issues with the canonical sampling of harmonic degrees of freedom (notice that the *chain* option will NOT solve this problem). The Langevin thermostat (fix langevin) works fine.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the original coordinates of tethered atoms to binary restart files, so that the spring effect will be the same in a restarted simulation. See the read restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

The fix modify *energy* option is supported by this fix to add the energy stored in the per-atom springs to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar and a global vector quantities which can be accessed by various output commands. The scalar is an energy which is the sum of the spring energy for each atom, where the per-atom energy is 0.5 * K * r^2. The vector has 2 positions, the first one is the coupling parameter lambda and the second one is the time derivative of lambda. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the per-atom spring energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix modify *energy* option for this fix.

An example script using this command is provided in the examples/USER/misc/ti directory.

Related commands:

fix spring, fix ti/rs

Restrictions:

fix ti/spring command   584

This command is part of the USER-MISC package. It is only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

Default:

The keyword default is function = 1.

(Frenkel) Daan Frenkel and Anthony J. C. Ladd, J. Chem. Phys. 81, 3188 (1984).

(Watanabe) M. Watanabe and W. P. Reinhardt, Phys Rev Lett, 65, 3301 (1990).

(de Koning 96) M. de Koning and A. Antonelli, Phys Rev E, 53, 465 (1996).

(de Koning 97) M. de Koning and A. Antonelli, Phys Rev B, 55, 735 (1997).

fix tmd command

Syntax:

```
fix ID group-ID tmd rho_final file1 N file2
```

- ID, group-ID are documented in [fix](#) command
- tmd = style name of this fix command
- rho_final = desired value of rho at the end of the run (distance units)
- file1 = filename to read target structure from
- N = dump TMD statistics every this many timesteps, 0 = no dump
- file2 = filename to write TMD statistics to (only needed if N > 0)

Examples:

```
fix 1 all nve
fix 2 tmdatoms tmd 1.0 target_file 100 tmd_dump_file
```

Description:

Perform targeted molecular dynamics (TMD) on a group of atoms. A holonomic constraint is used to force the atoms to move towards (or away from) the target configuration. The parameter "rho" is monotonically decreased (or increased) from its initial value to rho_final at the end of the run.

Rho has distance units and is a measure of the root-mean-squared distance (RMSD) between the current configuration of the atoms in the group and the target coordinates listed in file1. Thus a value of rho_final = 0.0 means move the atoms all the way to the final structure during the course of the run.

The target file1 can be ASCII text or a gzipped text file (detected by a .gz suffix). The format of the target file1 is as follows:

```
0.0 25.0 xlo xhi
0.0 25.0 ylo yhi
0.0 25.0 zlo zhi
125     24.97311   1.69005     23.46956 0 0 -1
126     1.94691    2.79640     1.92799  1 0 0
127     0.15906    3.46099     0.79121  1 0 0
...
```

The first 3 lines may or may not be needed, depending on the format of the atoms to follow. If image flags are included with the atoms, the 1st 3 lo/hi lines must appear in the file. If image flags are not included, the 1st 3 lines should not appear. The 3 lines contain the simulation box dimensions for the atom coordinates, in the same format as in a LAMMPS data file (see the [read_data](#) command).

The remaining lines each contain an atom ID and its target x,y,z coordinates. The atom lines (all or none of them) can optionally be followed by 3 integer values: nx,ny,nz. For periodic dimensions, they specify which image of the box the atom is considered to be in, i.e. a value of N (positive or negative) means add N times the box length to the coordinate to get the true value.

The atom lines can be listed in any order, but every atom in the group must be listed in the file. Atoms not in the fix group may also be listed; they will be ignored.

TMD statistics are written to file2 every N timesteps, unless N is specified as 0, which means no statistics.

The atoms in the fix tmd group should be integrated (via a fix nve, nvt, npt) along with other atoms in the system.

Restarts can be used with a fix tmd command. For example, imagine a 10000 timestep run with a rho_initial = 11 and a rho_final = 1. If a restart file was written after 2000 time steps, then the configuration in the file would have a rho value of 9. A new 8000 time step run could be performed with the same rho_final = 1 to complete the conformational change at the same transition rate. Note that for restarted runs, the name of the TMD statistics file should be changed to prevent it being overwritten.

For more information about TMD, see (Schlitter1) and (Schlitter2).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands.

This fix can ramp its rho parameter over multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

This fix is not invoked during energy minimization.

Restrictions:

All TMD fixes must be listed in the input script after all integrator fixes (nve, nvt, npt) are applied. This ensures that atoms are moved before their positions are corrected to comply with the constraint.

Atoms that have a TMD fix applied should not be part of a group to which a SHAKE fix is applied. This is because LAMMPS assumes there are not multiple competing holonomic constraints applied to the same atoms.

To read gzipped target files, you must compile LAMMPS with the -DLAMMPS_GZIP option - see the Making LAMMPS section of the documentation.

Related commands: none

Default: none

(Schlitter1) Schlitter, Swegat, Mulders, "Distance-type reaction coordinates for modelling activated processes", J Molecular Modeling, 7, 171-177 (2001).

(Schlitter2) Schlitter and Klahn, "The free energy of a reaction coordinate at multiple constraints: a concise formulation", Molecular Physics, 101, 3439-3443 (2003).

fix ttm command

Syntax:

```
fix ID group-ID ttm seed C_e rho_e kappa_e gamma_p gamma_s v_0 Nx Ny Nz T_infile N T_outfile
```

- ID, group-ID are documented in [fix](#) command
- ttm = style name of this fix command
- seed = random number seed to use for white noise (positive integer)
- C_e = electronic specific heat (energy/(electron*temperature) units)
- rho_e = electronic density (electrons/volume units)
- kappa_e = electronic thermal conductivity (energy/(time*distance*temperature) units)
- gamma_p = friction coefficient due to electron-ion interactions (mass/time units)
- gamma_s = friction coefficient due to electronic stopping (mass/time units)
- v_0 = electronic stopping critical velocity (velocity units)
- Nx = number of thermal solve grid points in the x-direction (positive integer)
- Ny = number of thermal solve grid points in the y-direction (positive integer)
- Nz = number of thermal solve grid points in the z-direction (positive integer)
- T_infile = filename to read initial electronic temperature from
- N = dump TTM temperatures every this many timesteps, 0 = no dump
- T_outfile = filename to write TTM temperatures to (only needed if N > 0)

Examples:

```
fix 2 all ttm 699489 1.0 1.0 10 0.1 0.0 2.0 1 12 1 initialTs 1000 T.out
fix 2 all ttm 123456 1.0 1.0 1.0 1.0 1.0 5.0 5 5 5 Te.in 1 Te.out
```

Description:

Use a two-temperature model (TTM) to represent heat transfer through and between electronic and atomic subsystems. LAMMPS models the atomic subsystem as usual with a molecular dynamics model and the classical force field specified by the user, but the electronic subsystem is modeled as a continuum, or a background "gas", on a regular grid. Energy can be transferred spatially within the grid representing the electrons. Energy can also be transferred between the electronic and the atomic subsystems. The algorithm underlying this fix was derived by D. M. Duffy and A. M. Rutherford and is discussed in two J Physics: Condensed Matter papers: [(Duffy)](#) and [(Rutherford)](#). They used this algorithm in cascade simulations where a primary knock-on atom (PKA) was initialized with a high velocity to simulate a radiation event.

Heat transfer between the electronic and atomic subsystems is carried out via an inhomogeneous Langevin thermostat. This thermostat differs from the regular Langevin thermostat ([fix langevin](#)) in three important ways. First, the Langevin thermostat is applied uniformly to all atoms in the user-specified group for a single target temperature, whereas the TTM fix applies Langevin thermostatting locally to atoms within the volumes represented by the user-specified grid points with a target temperature specific to that grid point. Second, the Langevin thermostat couples the temperature of the atoms to an infinite heat reservoir, whereas the heat reservoir for fix TTM is finite and represents the local electrons. Third, the TTM fix allows users to specify not just one friction coefficient, but rather two independent friction coefficients: one for the electron-ion interactions (*gamma_p*), and one for electron stopping (*gamma_s*).

When the friction coefficient due to electron stopping, *gamma_s*, is non-zero, electron stopping effects are included for atoms moving faster than the electron stopping critical velocity, *v_0*. For further details about this algorithm, see [(Duffy)](#) and [(Rutherford)](#).

Energy transport within the electronic subsystem is solved according to the heat diffusion equation with added source terms for heat transfer between the subsystems:

$$C_e \rho_e \frac{\partial T_e}{\partial t} = \nabla(\kappa_e \nabla T_e) - g_p(T_e - T_a) + g_s T_a'$$

where C_e is the specific heat, rho_e is the density, kappa_e is the thermal conductivity, T is temperature, the "e" and "a" subscripts represent electronic and atomic subsystems respectively, g_p is the coupling constant for the electron-ion interaction, and g_s is the electron stopping coupling parameter. C_e, rho_e, and kappa_e are specified as parameters to the fix. The other quantities are derived. The form of the heat diffusion equation used here is almost the same as that in equation 6 of (Duffy), with the exception that the electronic density is explicitly reprensented, rather than being part of the the specific heat parameter.

Currently, this fix assumes that none of the user-supplied parameters will vary with temperature. This assumption can be relaxed by modifying the source code to include the desired temperature dependency and functional form for any of the parameters. Note that (Duffy) used a tanh() functional form for the temperature dependence of the electronic specific heat, but ignored temperature dependencies of any of the other parameters.

This fix requires use of periodic boundary conditions and a 3D simulation. Periodic boundary conditions are also used in the heat equation solve for the electronic subsystem. This varies from the approach of (Rutherford) where the atomic subsystem was embedded within a larger continuum representation of the electronic subsystem.

The initial electronic temperature input file, *T_infile*, is a text file LAMMPS reads in with no header and with four numeric columns (ix,iy,iz,Temp) and with a number of rows equal to the number of user-specified grid points (Nx by Ny by Nz). The ix,iy,iz are node indices from 0 to nxnodes-1, etc. For example, the initial electronic temperatures on a 1 by 2 by 3 grid could be specified in a *T_infile* as follows:

```
0 0 0 1.0
0 0 1 1.0
0 0 2 1.0
0 1 0 2.0
0 1 1 2.0
0 1 2 2.0
```

where the electronic temperatures along the y=0 plane have been set to 1.0, and the electronic temperatures along the y=1 plane have been set to 2.0. The order of lines in this file is no important. If all the nodal values are not specified, LAMMPS will generate an error.

The temperature output file, *T_oufile*, is created and written by this fix. Temperatures for both the electronic and atomic subsystems at every node and every N timesteps are output. If N is specified as zero, no output is generated, and no output filename is needed. The format of the output is as follows. One long line is written every output timestep. The timestep itself is given in the first column. The next Nx*Ny*Nz columns contain the temperatures for the atomic subsystem, and the final Nx*Ny*Nz columns contain the temperatures for the electronic subsystem. The ordering of the Nx*Ny*Nz columns is with the z index varing fastest, y the next fastest, and x the slowest.

This fix does not change the coordinates of its atoms; it only scales their velocities. Thus a time integration fix (e.g. fix nve) should still be used to time integrate the affected atoms. This fix should not normally be used on atoms that have their temperature controlled by another fix - e.g. fix nvt or fix langevin.

This fix computes 2 output quantities stored in a vector of length 2, which can be accessed by various output commands. The first quantity is the total energy of the electronic subsystem. The second quantity is the energy transferred from the electronic to the atomic subsystem on that timestep. Note that the velocity verlet integrator applies the fix ttm forces to the atomic subsystem as two half-step velocity updates: one on the current timestep and one on the subsequent timestep. Consequently, the change in the atomic subsystem energy is lagged by half a timestep relative to the change in the electronic subsystem energy. As a result of this, users may notice slight fluctuations in the sum of the atomic and electronic subsystem energies reported at the end of the timestep.

The vector values calculated by this fix are "extensive".

IMPORTANT NOTE: The current implementation creates a copy of the electron grid that overlays the entire simulation domain, for each processor. Values on the grid are summed across all processors. Thus you should insure that this grid is not too large, else your simulation could incur high memory and communication costs.

Restart, fix_modify, output, run start/stop, minimize info:

This fix writes the state of the electronic subsystem and the energy exchange between the subsystems to binary restart files. See the read_restart command for info on how to re-specify a fix in an input script that reads a restart file, so that the operation of the fix continues in an uninterrupted fashion.

Because the state of the random number generator is not saved in the restart files, this means you cannot do "exact" restarts with this fix, where the simulation continues on the same as if no restart had taken place. However, in a statistical sense, a restarted simulation should produce the same behavior.

None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This fix can only be used for 3d simulations and orthogonal simlulation boxes. You must use periodic boundary conditions with this fix.

Related commands:

fix langevin, fix dt/reset

Default: none

(Duffy) D M Duffy and A M Rutherford, J. Phys.: Condens. Matter, 19, 016207-016218 (2007).

(Rutherford) A M Rutherford and D M Duffy, J. Phys.: Condens. Matter, 19, 496201-496210 (2007).

fix tune/kspace command

Syntax:

```
fix ID group-ID tune/kspace N
```

- ID, group-ID are documented in [fix](#) command
- tune/kspace = style name of this fix command
- N = invoke this fix every N steps

Examples:

```
fix 2 all tune/kspace 100
```

Description:

This fix tests each kspace style (Ewald, PPPM, and MSM), and automatically selects the fastest style to use for the remainder of the run. If the fastest style is Ewald or PPPM, the fix also adjusts the coulomb cutoff towards optimal speed. Future versions of this fix will automatically select other kspace parameters to use for maximum simulation speed. The kspace parameters may include the style, cutoff, grid points in each direction, order, Ewald parameter, MSM parallelization cut-point, MPI tasks to use, etc.

The rationale for this fix is to provide the user with as-fast-as-possible simulations that include long-range electrostatics (kspace) while meeting the user-prescribed accuracy requirement. A simple heuristic could never capture the optimal combination of parameters for every possible run-time scenario. But by performing short tests of various kspace parameter sets, this fix allows parameters to be tailored specifically to the user's machine, MPI ranks, use of threading or accelerators, the simulated system, and the simulation details. In addition, it is possible that parameters could be evolved with the simulation on-the-fly, which is useful for systems that are dynamically evolving (e.g. changes in box size/shape or number of particles).

When this fix is invoked, LAMMPS will perform short timed tests of various parameter sets to determine the optimal parameters. Tests are performed on-the-fly, with a new test initialized every N steps. N should be chosen large enough so that adequate CPU time lapses between tests, thereby providing statistically significant timings. But N should not be chosen to be so large that an unfortunate parameter set test takes an inordinate amount of wall time to complete. An N of 100 for most problems seems reasonable. Once an optimal parameter set is found, that set is used for the remainder of the run.

This fix uses heristics to guide it's selection of parameter sets to test, but the actual timed results will be used to decide which set to use in the simulation.

It is not necessary to discard trajectories produced using sub-optimal parameter sets, or a mix of various parameter sets, since the user-prescribed accuracy will have been maintained throughout. However, some users may prefer to use this fix only to discover the optimal parameter set for a given setup that can then be used on subsequent production runs.

This fix starts with kspace parameters that are set by the user with the [kspace_style](#) and [kspace_modify](#) commands. The prescribed accuracy will be maintained by this fix throughout the simulation.

None of the [fix_modify](#) options are relevant to this fix.

No parameter of this fix can be used with the *start/stop* keywords of the [run](#) command. This fix is not invoked during [energy minimization](#).

Restrictions:

This fix is part of the KSPACE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Do not set "neigh_modify once yes" or else this fix will never be called. Reneighboring is required.

Related commands:

[kspace_style](#), [boundary](#) [kspace_modify](#), [pair_style lj/cut/coul/long](#), [pair_style lj/charmm/coul/long](#), [pair_style lj/long](#), [pair_style lj/long/coul/long](#), [pair_style buck/coul/long](#)

Default:

fix viscosity command

Syntax:

```
fix ID group-ID viscosity N vdim pdim Nbin keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- viscosity = style name of this fix command
- N = perform momentum exchange every N steps
- vdim = *x* or *y* or *z* = which momentum component to exchange
- pdim = *x* or *y* or *z* = direction of momentum transfer
- Nbin = # of layers in pdim direction (must be even number)
- zero or more keyword/value pairs may be appended
- keyword = *swap* or *target*

```
  swap value = Nswap = number of swaps to perform every N steps
    vtarget value = V or INF = target velocity of swap partners (velocity units)
```

Examples:

```
fix 1 all viscosity 100 x z 20
fix 1 all viscosity 50 x z 20 swap 2 vtarget 1.5
```

Description:

Use the Muller-Plathe algorithm described in [this paper](#) to exchange momenta between two particles in different regions of the simulation box every N steps. This induces a shear velocity profile in the system. As described below this enables a viscosity of the fluid to be calculated. This algorithm is sometimes called a reverse non-equilibrium MD (reverse NEMD) approach to computing viscosity. This is because the usual NEMD approach is to impose a shear velocity profile on the system and measure the response via an off-diagonal component of the stress tensor, which is proportional to the momentum flux. In the Muller-Plathe method, the momentum flux is imposed, and the shear velocity profile is the system's response.

The simulation box is divided into *Nbin* layers in the *pdim* direction, where the layer 1 is at the low end of that dimension and the layer *Nbin* is at the high end. Every N steps, Nswap pairs of atoms are chosen in the following manner. Only atoms in the fix group are considered. Nswap atoms in layer 1 with positive velocity components in the *vdim* direction closest to the target value *V* are selected. Similarly, Nswap atoms in the "middle" layer (see below) with negative velocity components in the *vdim* direction closest to the negative of the target value *V* are selected. The two sets of Nswap atoms are paired up and their *vdim* momenta components are swapped within each pair. This resets their velocities, typically in opposite directions. Over time, this induces a shear velocity profile in the system which can be measured using commands such as the following, which writes the profile to the file tmp.profile:

```
fix f1 all ave/spatial 100 10 1000 z lower 0.05 vx &
    file tmp.profile units reduced
```

Note that by default, Nswap = 1 and vtarget = INF, though this can be changed by the optional *swap* and *vtarget* keywords. When vtarget = INF, one or more atoms with the most positive and negative velocity components are selected. Setting these parameters appropriately, in conjunction with the swap rate N, allows the momentum flux rate to be adjusted across a wide range of values, and the momenta to be exchanged in large chunks or more smoothly.

The "middle" layer for momenta swapping is defined as the *Nbin*/2 + 1 layer. Thus if *Nbin* = 20, the two

swapping layers are 1 and 11. This should lead to a symmetric velocity profile since the two layers are separated by the same distance in both directions in a periodic sense. This is why *Nbin* is restricted to being an even number.

As described below, the total momentum transferred by these velocity swaps is computed by the fix and can be output. Dividing this quantity by time and the cross-sectional area of the simulation box yields a momentum flux. The ratio of momentum flux to the slope of the shear velocity profile is proportional to the viscosity of the fluid, in appropriate units. See the Muller-Plathe paper for details.

IMPORTANT NOTE: If your system is periodic in the direction of the momentum flux, then the flux is going in 2 directions. This means the effective momentum flux in one direction is reduced by a factor of 2. You will see this in the equations for viscosity in the Muller-Plathe paper. LAMMPS is simply tallying momentum which does not account for whether or not your system is periodic; you must use the value appropriately to yield a viscosity for your system.

IMPORTANT NOTE: After equilibration, if the velocity profile you observe is not linear, then you are likely swapping momentum too frequently and are not in a regime of linear response. In this case you cannot accurately infer a viscosity and should try increasing the Nevery parameter.

An alternative method for calculating a viscosity is to run a NEMD simulation, as described in Section howto 13 of the manual. NEMD simulations deform the simmulation box via the fix deform command. Thus they cannot be run on a charged system using a PPPM solver since PPPM does not currently support non-orthogonal boxes. Using fix viscosity keeps the box orthogonal; thus it does not suffer from this limitation.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global scalar which can be accessed by various output commands. The scalar is the cummulative momentum transferred between the bottom and middle of the simulation box (in the *pdim* direction) is stored as a scalar quantity by this fix. This quantity is zeroed when the fix is defined and accumlates thereafter, once every N steps. The units of the quantity are momentum = mass*velocity. The scalar value calculated by this fix is "intensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix is part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Swaps conserve both momentum and kinetic energy, even if the masses of the swapped atoms are not equal. Thus you should not need to thermostat the system. If you do use a thermostat, you may want to apply it only to the non-swapped dimensions (other than *vdim*).

LAMMPS does not check, but you should not use this fix to swap velocities of atoms that are in constrained molecules, e.g. via fix shake or fix rigid. This is because application of the constraints will alter the amount of transferred momentum. You should, however, be able to use flexible molecules. See the Maginn paper for an example of using this algorithm in a computation of alcohol molecule properties.

When running a simulation with large, massive particles or molecules in a background solvent, you may want

to only exchange momenta bewteen solvent particles.

Related commands:

[fix ave/spatial](), [fix thermal/conductivity]()

Default:

The option defaults are swap = 1 and vtarget = INF.

(**Muller-Plathe**) Muller-Plathe, Phys Rev E, 59, 4894-4898 (1999).

(**Maginn**) Kelkar, Rafferty, Maginn, Siepmann, Fluid Phase Equilibria, 260, 218-231 (2007).

fix viscous command

fix viscous/cuda command

Syntax:

```
fix ID group-ID viscous gamma keyword values ...
```

- ID, group-ID are documented in <u>fix</u> command
- viscous = style name of this fix command
- gamma = damping coefficient (force/velocity units)
- zero or more keyword/value pairs may be appended

```
  keyword = scale
    scale values = type ratio
      type = atom type (1-N)
      ratio = factor to scale the damping coefficient by
```

Examples:

```
fix 1 flow viscous 0.1
fix 1 damp viscous 0.5 scale 3 2.5
```

Description:

Add a viscous damping force to atoms in the group that is proportional to the velocity of the atom. The added force can be thought of as a frictional interaction with implicit solvent, i.e. the no-slip Stokes drag on a spherical particle. In granular simulations this can be useful for draining the kinetic energy from the system in a controlled fashion. If used without additional thermostatting (to add kinetic energy to the system), it has the effect of slowly (or rapidly) freezing the system; hence it can also be used as a simple energy minimization technique.

The damping force F is given by F = - gamma * velocity. The larger the coefficient, the faster the kinetic energy is reduced. If the optional keyword *scale* is used, gamma can scaled up or down by the specified factor for atoms of that type. It can be used multiple times to adjust gamma for several atom types.

IMPORTANT NOTE: You should specify gamma in force/velocity units. This is not the same as mass/time units, at least for some of the LAMMPS <u>units</u> options like "real" or "metal" that are not self-consistent.

In a Brownian dynamics context, gamma = Kb T / D, where Kb = Boltzmann's constant, T = temperature, and D = particle diffusion coefficient. D can be written as Kb T / (3 pi eta d), where eta = dynamic viscosity of the frictional fluid and d = diameter of particle. This means gamma = 3 pi eta d, and thus is proportional to the viscosity of the fluid and the particle diameter.

In the current implementation, rather than have the user specify a viscosity, gamma is specified directly in force/velocity units. If needed, gamma can be adjusted for atoms of different sizes (i.e. sigma) by using the *scale* keyword.

Note that Brownian dynamics models also typically include a randomized force term to thermostat the system at a chosen temperature. The <u>fix langevin</u> command does this. It has the same viscous damping term as fix viscous and adds a random force to each atom. The random force term is proportional to the sqrt of the chosen thermostatting temperature. Thus if you use fix langevin with a target T = 0, its random force term is zero, and you are essentially performing the same operation as fix viscous. Also note that the gamma of fix viscous is

related to the damping parameter of fix langevin, however the former is specified in units of force/velocity and the latter in units of time, so that it can more easily be used as a thermostat.

Styles with a *cuda* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command. This fix should only be used with damped dynamics minimizers that allow for non-conservative forces. See the min_style command for details.

Restrictions: none

Related commands:

fix langevin

Default: none

fix wall/gran command

Syntax:

```
fix ID group-ID style model_type model_name wallstyle wallstyleargs general_keywords general_valu
```

- ID, group-ID are documented in [fix](#) command
- style = *wall/gran*
- zero or more model_type/model_name pairs may be appended. They must be appended in the following order

  ```
  model values = described here
     tangential values = described here
     cohesion values = described here
     rolling_friction values = described here
     surface values = described here
  ```
- wallstyle = *mesh* or *primitive*
- wallstyle args for wallstyle *mesh* = *n_meshes* and *meshes*

  ```
  n_meshes value = nm
      nm = # of meshes (see fix mesh/surface) to use for the wall (positive integer)
     meshes values = meshlist
      meshlist =  id(s) of the mesh(es) (see fix mesh/surface) to be used. These must be def
  ```
- wallstyle args for wallstyle *primitive* = *type* or *xplane* or *yplane* or *zplane* or *xcylinder* or *ycylinder* or *zcylinder*

  ```
  type args = tp
     tp = atom_type (material type) of the wall
    xplane or yplane or zplane args = pos
     pos = position plane (distance units)
    xcylinder or ycylinder or zcylinder args = radius c1 c2
     radius = cylinder radius (distance units)
     c1,c2 = coordinates of center axis in other 2 dims (distance units)
  ```
- zero or more general_keyword/value pairs may be appended
- general_keyword = *shear* or *store_force* or *store_force_contact*

  ```
  shear values = dim vshear
     dim = x or y or z
     vshear = magnitude of shear velocity (velocity units)
    temperature value = T0
     T0 = Temperature of the wall (temperature units)
    contact_area values = 'overlap' or 'constant value' or 'projection'
    store_force value = 'yes' or 'no'
     yes, no = determines if the wall force exerted on the particles is stored in a fix pro
    store_force_contact value = 'yes' or 'no'
     yes, no = determines if the force for each particle-wall contact is stored in a fix pr
  ```
- following the general_keyword/value pairs, zero or more model_keyword/model_valuezero pairs may be appended in arbitrary order model_type/model_name pairs = described for each model separately [here](#)

Examples:

```
fix zwalls all wall/gran model hertz tangential history primitive type 1 zplane 0.15
fix meshwalls all wall/gran model hertz tangential history mesh n_meshes 2 meshes cad1 cad2
```

LIGGGHTS vs. LAMMPS Info:

This command has been improved in accordance with the overhauled [pair gran](#) styles. Furthermore, it offers the new *wallstyle mesh*, which enables fix wall/gran to handle complex wall geometries imported from CAD.

Description:

Bound the simulation domain of a granular system with a frictional wall. All particles in the group interact with the wall when they are close enough to touch it. The equation for the force between the wall and particles touching it is the same as the corresponding equation on the [pair_style granular](#) doc page, in the limit of one of the two particles going to infinite radius and mass (flat wall).

You must choose the models matching the pair style used, otherwise an error is created. As with [pair_style granular](#), you have to define the mechanical properties for each material you are used in the simulation with fix property commands. See [pair_style gran](#) for more details and [the model doc page](#) for details.

For wallstyle *mesh*, fix_id1, fix_id2 etc. must be IDs of valid fixes of type [fix mesh/surface](#), defining the granular mesh to be used for the wall. Triangle-particle neighbor lists are built to efficiently track particle-triangle contacts. Particle-tri neighbor list build is triggered if any particle moves more than half the skin distance or (in case of moving mesh) if the mesh itself moves more than half the skin distance since the last build. A warning is generated if a dangerous particle-tri neigh list build is detected (e.g. if particles are inserted too close to a wall, see section 'Restrictions'). For style *mesh*, the atom_type (material type) is inherited from the atom style provided in the [fix mesh/surface](#) command.

For *wallstyle primitive*, the atom_type (material type) has to be provided via keyword *type*. Primitve walls can be *xplane* or *yplane* or *zplane* or *cylindrical*. The 3 planar options specify a single wall in a dimension. Wall positions are given by values for lo and hi. For an *xcylinder*, *ycylinder* or *zcylinder*, the radius and the cylinder axis in the other two dims is specified.

Optionally, primitive walls can be moving, if the shear keyword is appended.

For the *shear* keyword, the wall moves continuously in the specified dimension with velocity vshear. The dimension must be tangential to walls with a planar wallstyle, e.g. in the y or z directions for an *xplane* wall. For *zcylinder* walls, a dimension of z means the cylinder is moving in the z-direction along it's axis. A dimension of x or y means the cylinder is spinning around the z-axis, either in the clockwise direction for vshear > 0 or counter-clockwise for vshear < 0. In this case, vshear is the tangential velocity of the wall at whatever radius has been defined. The same applies to *xcylinder* and *ycylinder* accordingly.

NOTE: The keywords wiggle or shear can NOT be used for *wallstyle mesh*. For a moving a granular wall with *wallstyle mesh*, use the more flexible command "fix move/mesh"fix_move_mesh.html, or use the keywords *velocity* or *angular_velocity* in [fix mesh/surface](#).

The keyword *temperature* is used to assign a constant temperature to the wall. This setting gets effective in conjunction with heat conduction via [fix heat/gran/conduction](#). For *wallstyle mesh*, the value for the temperature given in this command is ignored and the temperature value is specified per mesh via [fix mesh/surface](#). Contact area calculation can be controlled by keyword *contact_area* in the same manner as for [fix heat/gran/conduction](#).

By specifying *store_force* = 'yes', you can instruct the command to store the wall force exerted on the particles in a [fix property/atom](#) with id force_(ID), where (ID) is the id of the fix wall/gran command.

The effect of keyword *rolling_friction*, *cohesion*, *tangential_damping*, *viscous* and *absolute_damping* is explanted in [pair gran](#)

Restart, fix_modify, output, run start/stop, minimize info:

fix wall/gran command

If applicable, contact history is written to **binary restart files** so simulations can continue properly. None of the **fix_modify** options are relevant to this fix. No global scalar or vector or are stored by this fix. If *store_force* = 'yes' is specified, the per-atom wall force can be accessed by the various output commands via f_force(ID)**1**, f_force(ID)**2**, f_force(ID)**3**. (ID) is the id of the fix wall/gran command. No parameter of this fix can be used with the start/stop keywords of the **run** command. This fix is not invoked during **energy minimization** .

Restrictions:

There can be only one fix wall/gran command with style *mesh*. Note that this is not really a restriction because you can include multiple fixes of type **fix mesh/surface** in the fix wall/gran command.

When using style style *mesh*, you have to use the style *bin* for the **neighbor command**.

Style *mesh* can not be used in conjunction with triclinic simulation boxes.

When using style *mesh*' in combination with a particle insertion command, you always have to keep a minimum distance between the wall and the insertion region that is equal to maximum particle radius + half the skin defined in the **neighbor command**. Otherwise, newly inserted particles interpenetrate the walls before a triangle neighbor list is built the first time.

The keyword shear can NOT be used for style *mesh*. For moving granular wall with style mesh, use "fix move/mesh"fix_move_mesh.html.

Any dimension (xyz) that has a planar granular wall must be non-periodic.

Related commands:

fix mesh/surface, **fix_move_mesh**, **pair_style granular** Models for use with this command are described **here**

Default:

model = 'hertz' *tangential* = 'history' *rolling_friction* = 'off' *cohesion* = 'off' *surface* = 'default'

fix wall/lj93 command

fix wall/lj126 command

fix wall/lj1043 command

fix wall/colloid command

fix wall/harmonic command

Syntax:

```
fix ID group-ID style face args ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- style = *wall/lj93* or *wall/lj126* or *wall/lj1043* or *wall/colloid* or *wall/harmonic*
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
    args = coord epsilon sigma cutoff
      coord = position of wall = EDGE or constant or variable
        EDGE = current lo or hi edge of simulation box
        constant = number like 0.0 or -30.0 (distance units)
        variable = equal-style variable like v_x or v_wiggle
      epsilon = strength factor for wall-particle interaction (energy or energy/distance^2 u
        epsilon can be a variable (see below)
      sigma = size factor for wall-particle interaction (distance units)
        sigma can be a variable (see below)
      cutoff = distance from wall at which wall-particle interaction is cut off (distance un
```

- zero or more keyword/value pairs may be appended
- keyword = *units* or *fld*

```
  units value = lattice or box
      lattice = the wall position is defined in lattice units
      box = the wall position is defined in simulation box units
    fld value = yes or no
      yes = invoke the wall constraint to be compatible with implicit FLD
      no = invoke the wall constraint in the normal way
    pbc value = yes or no
      yes = allow periodic boundary in a wall dimension
      no = require non-perioidic boundaries in any wall dimension
```

Examples:

```
fix wallhi all wall/lj93 xlo -1.0 1.0 1.0 2.5 units box
fix wallhi all wall/lj93 xhi EDGE 1.0 1.0 2.5
fix wallhi all wall/lj126 v_wiggle 23.2 1.0 1.0 2.5
fix zwalls all wall/colloid zlo 0.0 1.0 1.0 0.858 zhi 40.0 1.0 1.0 0.858
```

Description:

Bound the simulation domain on one or more of its faces with a flat wall that interacts with the atoms in the group by generating a force on the atom in a direction perpendicular to the wall. The energy of wall-particle interactions depends on the style.

For style *wall/lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

For style *wall/lj126*, the energy E is given by the 12/6 potential:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

For style *wall/lj1043*, the energy E is given by the 10/4/3 potential:

$$E = 2\pi\epsilon \left[\frac{2}{5} \left(\frac{\sigma}{r} \right)^{10} - \left(\frac{\sigma}{r} \right)^4 - \frac{\sqrt{(2)}\sigma^3}{3 \left(r + \left(0.61/\sqrt{(2)} \right) \sigma \right)^3} \right] \qquad r < r_c$$

For style *wall/colloid*, the energy E is given by an integrated form of the <u>pair_style colloid</u> potential:

$$E = \epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R - D}{D^7} + \frac{D + 8R}{(D + 2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D + R) + D(D + 2R)\left[\ln D - \ln(D + 2R) \right]}{D(D + 2R)} \right) \right] \qquad r < r$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential:

$$E = \epsilon \left(r - r_c \right)^2 \qquad r < r_c$$

In all cases, *r* is the distance from the particle to the wall at position *coord*, and Rc is the *cutoff* distance at which the particle and wall no longer interact. The energy of the wall potential is shifted so that the wall-particle interaction energy is 0.0 at the cutoff distance.

Up to 6 walls or faces can be specified in a single command: *xlo, xhi, ylo, yhi, zlo, zhi*. A *lo* face interacts with particles near the lower side of the simulation box in that dimension. A *hi* face interacts with particles near the upper side of the simulation box in that dimension.

fix wall/harmonic command

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an equal-style variable name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position. See examples below.

For the *wall/lj93* and *wall/lj126* and *wall/lj1043* styles, *epsilon* and *sigma* are the usual Lennard-Jones parameters, which determine the strength and size of the particle as it interacts with the wall. Epsilon has energy units. Note that this *epsilon* and *sigma* may be different than any *epsilon* or *sigma* values defined for a pair style that computes particle-particle interactions.

The *wall/lj93* interaction is derived by integrating over a 3d half-lattice of Lennard-Jones 12/6 particles. The *wall/lj126* interaction is effectively a harder, more repulsive wall interaction. The *wall/lj1043* interaction is yet a different form of wall interaction, described in Magda et al in (Magda).

For the *wall/colloid* style, *R* is the radius of the colloid particle, *D* is the distance from the surface of the colloid particle to the wall (r-R), and *sigma* is the size of a constituent LJ particle inside the colloid particle and wall. Note that the cutoff distance Rc in this case is the distance from the colloid particle center to the wall. The prefactor *epsilon* can be thought of as an effective Hamaker constant with energy units for the strength of the colloid-wall interaction. More specifically, the *epsilon* pre-factor = 4 * pi^2 * rho_wall * rho_colloid * epsilon * sigma^6, where epsilon and sigma are the LJ parameters for the constituent LJ particles. Rho_wall and rho_colloid are the number density of the constituent particles, in the wall and colloid respectively, in units of 1/volume.

The *wall/colloid* interaction is derived by integrating over constituent LJ particles of size *sigma* within the colloid particle and a 3d half-lattice of Lennard-Jones 12/6 particles of size *sigma* in the wall. As mentioned in the preceeding paragraph, the density of particles in the wall and colloid can be different, as specified by the *epsilon* pre-factor.

For the *wall/harmonic* style, *epsilon* is effectively the spring constant K, and has units (energy/distance^2). The input parameter *sigma* is ignored. The minimum energy position of the harmonic spring is at the *cutoff*. This is a repulsive-only spring since the interaction is truncated at the *cutoff*

For any wall, the *epsilon* and/or *sigma* parameter can be specified as an equal-style variable, in which case it should be specified as v_name, where name is the variable name. As with a variable wall position, the variable is evaluated each timestep and the result becomes the current epsilon or sigma of the wall. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall interaction.

IMPORTANT NOTE: For all of the styles, you must insure that r is always > 0 for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles at the wall position *coord* (r = 0) or with particles on the wrong side of the wall (r < 0). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as r -> 0. The *wall/colloid* style is even more restrictive, since the energy blows up as D = r-R -> 0. This means the finite-size particles of radius R must be a distance larger than R from the wall position *coord*. The *harmonic* style is a softer potential and does not blow up as r -> 0, but you must use a large enough *epsilon* that particles always reamin on the correct side of the wall (r > 0).

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when

a numeric constant or variable is used. It is not relevant when EDGE is used to specify a face position. In the variable case, the variable is assumed to produce a value compatible with the *units* setting you specify.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

The *fld* keyword can be used with a *yes* setting to invoke the wall constraint before pairwise interactions are computed. This allows an implicit FLD model using pair_style lubricateU to include the wall force in its calculations. If the setting is *no*, wall forces are imposed after pairwise interactions, in the usual manner.

The *pbc* keyword can be used with a *yes* setting to allow walls to be specified in a periodic dimension. See the boundary command for options on simulation box boundaries. The default for *pbc* is *no*, which means the system must be non-periodic when using a wall. But you may wish to use a periodic box. E.g. to allow some particles to interact with the wall via the fix group-ID, and others to pass through it and wrap around a periodic box. In this case you should insure that the wall if sufficiently far enough away from the box boundary. If you do not, then particles may interact with both the wall and with periodic images on the other side of the box, which is probably not what you want.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style variables. The wall interaction parameters (epsilon, sigma) could be varied with additional variable definitions.

```
variable ramp equal ramp(0,10)
fix 1 all wall xlo v_ramp 1.0 1.0 2.5

variable linear equal vdisplace(0,20)
fix 1 all wall xlo v_linear 1.0 1.0 2.5

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall xlo v_wiggle 1.0 1.0 2.5
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The vdisplace(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 - cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and each wall to the system's potential energy as part of thermodynamic output.

fix wall/harmonic command 604

This fix computes a global scalar energy and a global vector of forces, which can be accessed by various [output commands](). Note that the scalar energy is the sum of interactions with all defined walls. If you want the energy on a per-wall basis, you need to use multiple fix wall commands. The length of the vector is equal to the number of walls defined by the fix. Each vector value is the normal force on a specific wall. Note that an outward force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run]() command.

The forces due to this fix are imposed during an energy minimization, invoked by the [minimize]() command.

IMPORTANT NOTE: If you want the atom/wall interaction energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the [fix_modify]() *energy* option for this fix.

Restrictions: none

Related commands:

[fix wall/reflect](), [fix wall/gran](), [fix wall/region]()

Default:

The option defaults units = lattice, fld = no, and pbc = no.

(Magda) Magda, Tirrell, Davis, J Chem Phys, 83, 1888-1901 (1985).

fix wall/piston command

Syntax:

```
fix ID group-ID wall/piston face ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- wall/piston = style name of this fix command
- face = *zlo*
- zero or more keyword/value pairs may be appended
- keyword = *pos* or *vel* or *ramp* or *units*

```
  pos args = z
     z = z coordinate at which the piston begins (distance units)
   vel args = vz
     vz = final velocity of the piston (velocity units)
   ramp = use a linear velocity ramp from 0 to vz
   temp args = target damp seed extent
     target = target velocity for region immediately ahead of the piston
     damp = damping paramter (time units)
     seed = random number seed for langevin kicks
     extent = extent of thermostated region (distance units)
   units value = lattice or box
     lattice = the wall position is defined in lattice units
     box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/piston zlo
fix walls all wall/piston zlo pos 1.0 0.0 0.0 vel 0.0 0.0 10.0 units box
fix top all wall/piston zlo vel 0.0 0.0 10.0 ramp
```

Description:

Bound the simulation with a moving wall which reflect particles in the specified group and drive the system with an effective infinite-mass piston capable of driving shock waves.

A momentum mirror technique is used, which means that if an atom (or the wall) moves such that an atom is outside the wall on a timestep by a distance delta (e.g. due to [fix nve](#)), then it is put back inside the face by the same delta, and the velocity relative to the moving wall is flipped in z. For instance, a stationary particle hit with a piston wall with velocity vz, will end the timestep with a velocity of 2*vz.

Currently the *face* keyword can only be *zlo*. This creates a piston moving in the positive z direction. Particles with z coordinate less than the wall position are reflected to a z coordinate greater than the wall position. If the piston velocity is vpz and the particle velocity before reflection is vzi, the particle velocity after reflection is -vzi + 2*vpz.

The initial position of the wall can be specified by the *pos* keyword.

The final velocity of the wall can be specified by the *vel* keyword

The *ramp* keyword will cause the wall/piston to adjust the velocity linearly from zero velocity to *vel* over the course of the run. If the *ramp* keyword is omitted then the wall/piston moves at a constant velocity defined by *vel*.

The *temp* keyword will cause the region immediately in front of the wall/piston to be thermostated with a Langevin thermostat. This region moves with the piston. The damping and kicking are measured in the reference frame of the piston. So, a temperature of zero would mean all particles were moving at exactly the speed of the wall/piston.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

This fix style is part of the SHOCK package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The face that has the wall/piston must be boundary type 's' (shrink-wrapped). The opposing face can be any boundary type other than periodic.

A wall/piston should not be used with rigid bodies such as those defined by a "fix rigid" command. This is because the wall/piston displaces atoms directly rather than exerting a force on them.

Related commands:

fix wall/reflect command, fix append/atoms command

Default:

The keyword defaults are pos = 0, vel = 0, units = lattice.

fix wall/reflect command

Syntax:

```
fix ID group-ID wall/reflect face arg ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- wall/reflect = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
  xlo,ylo,zlo arg = EDGE or constant or variable
      EDGE = current lo edge of simulation box
      constant = number like 0.0 or -30.0 (distance units)
      variable = equal-style variable like v_x or v_wiggle
    xhi,yhi,zhi arg = EDGE or constant or variable
      EDGE = current hi edge of simulation box
      constant = number like 50.0 or 100.3 (distance units)
      variable = equal-style variable like v_x or v_wiggle
```
- zero or more keyword/value pairs may be appended
- keyword = *units*

```
  units value = lattice or box
      lattice = the wall position is defined in lattice units
      box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/reflect xlo EDGE xhi EDGE
fix walls all wall/reflect xlo 0.0 ylo 10.0 units box
fix top all wall/reflect zhi v_pressdown
```

Description:

Bound the simulation with one or more walls which reflect particles in the specified group when they attempt to move thru them.

Reflection means that if an atom moves outside the wall on a timestep by a distance delta (e.g. due to [fix nve](#)), then it is put back inside the face by the same delta, and the sign of the corresponding component of its velocity is flipped.

When used in conjunction with [fix nve](#) and [run_style verlet](#), the resultant time-integration algorithm is equivalent to the primitive splitting algorithm (PSA) described by [Bond](#). Because each reflection event divides the corresponding timestep asymmetrically, energy conservation is only satisfied to O(dt), rather than to O(dt^2) as it would be for velocity-Verlet integration without reflective walls.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an [equal-style variable](#) name. In this case the variable is

fix wall/reflect command   608

evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant or variable is used. It is not relevant when EDGE is used to specify a face position. In the variable case, the variable is assumed to produce a value compatible with the *units* setting you specify.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style variables.

```
variable ramp equal ramp(0,10)
fix 1 all wall/reflect xlo v_ramp

variable linear equal vdisplace(0,20)
fix 1 all wall/reflect xlo v_linear

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall/reflect xlo v_wiggle

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall/reflect xlo v_wiggle
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The vdisplace(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 - cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix. No global or per-atom quantities are stored by this fix for access by various output commands. No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Any dimension (xyz) that has a reflecting wall must be non-periodic.

A reflecting wall should not be used with rigid bodies such as those defined by a "fix rigid" command. This is because the wall/reflect displaces atoms directly rather than exerts a force on them. For rigid bodies, use a soft

fix wall/reflect command

wall instead, such as fix wall/lj93. LAMMPS will flag the use of a rigid fix with fix wall/reflect with a warning, but will not generate an error.

Related commands:

fix wall/lj93 command

Default: none

(Bond) Bond and Leimkuhler, SIAM J Sci Comput, 30, p 134 (2007).

fix wall/region command

Syntax:

```
fix ID group-ID wall/region region-ID style epsilon sigma cutoff
```

- ID, group-ID are documented in [fix](#) command
- wall/region = style name of this fix command
- region-ID = region whose boundary will act as wall
- style = *lj93* or *lj126* or *colloid* or *harmonic*
- epsilon = strength factor for wall-particle interaction (energy or energy/distance^2 units)
- sigma = size factor for wall-particle interaction (distance units)
- cutoff = distance from wall at which wall-particle interaction is cut off (distance units)

Examples:

```
fix wall all wall/region mySphere lj93 1.0 1.0 2.5
```

Description:

Treat the surface of the geometric region defined by the *region-ID* as a bounding wall which interacts with nearby particles according to the specified style. The distance between a particle and the surface is the distance to the nearest point on the surface and the force the wall exerts on the particle is along the direction between that point and the particle, which is the direction normal to the surface at that point.

Regions are defined using the [region](#) command. Note that the region volume can be interior or exterior to the bounding surface, which will determine in which direction the surface interacts with particles, i.e. the direction of the surface normal. Regions can either be primitive shapes (block, sphere, cylinder, etc) or combinations of primitive shapes specified via the *union* or *intersect* region styles. These latter styles can be used to construct particle containers with complex shapes. Regions can also change over time via the [region](#) command keywords (move) and *rotate*. If such a region is used with this fix, then the of region surface will move over time in the corresponding manner.

IMPORTANT NOTE: As discussed on the [region](#) command doc page, regions in LAMMPS do not get wrapped across periodic boundaries. It is up to you to insure that periodic or non-periodic boundaries are specified appropriately via the [boundary](#) command when using a region as a wall that bounds particle motion. This also means that if you embed a region in your simulation box and want it to repulse particles from its surface (using the "side out" option in the [region](#) command), that its repulsive force will not be felt across a periodic boundary.

IMPORTANT NOTE: For primitive regions with sharp corners and/or edges (e.g. a block or cylinder), wall/particle forces are computed accurately for both interior and exterior regions. For *union* and *intersect* regions, additional sharp corners and edges may be present due to the intersection of the surfaces of 2 or more primitive volumes. These corners and edges can be of two types: concave or convex. Concave points/edges are like the corners of a cube as seen by particles in the interior of a cube. Wall/particle forces around these features are computed correctly. Convex points/edges are like the corners of a cube as seen by particles exterior to the cube, i.e. the points jut into the volume where particles are present. LAMMPS does NOT compute the location of these convex points directly, and hence wall/particle forces in the cutoff volume around these points suffer from inaccuracies. The basic problem is that the outward normal of the surface is not continuous at these points. This can cause particles to feel no force (they don't "see" the wall) when in one location, then move a distance epsilon, and suddenly feel a large force because they now "see" the wall. In the worst-case scenario, this can blow particles out of the simulation box. Thus, as a general rule you should not

use the fix wall/region command with *union* or *interesect* regions that have convex points or edges.

The energy of wall-particle interactions depends on the specified style.

For style *lj93*, the energy E is given by the 9/3 potential:

$$E = \epsilon \left[\frac{2}{15} \left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^3 \right] \qquad r < r_c$$

For style *lj126*, the energy E is given by the 12/6 potential:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

For style *colloid*, the energy E is given by an integrated form of the pair_style colloid potential:

$$E = \epsilon \left[\frac{\sigma^6}{7560} \left(\frac{6R - D}{D^7} + \frac{D + 8R}{(D + 2R)^7} \right) - \frac{1}{6} \left(\frac{2R(D + R) + D(D + 2R) \left[\ln D - \ln(D + 2R) \right]}{D(D + 2R)} \right) \right] \qquad r < r$$

For style *wall/harmonic*, the energy E is given by a harmonic spring potential:

$$E = \epsilon \left(r - r_c \right)^2 \qquad r < r_c$$

In all cases, *r* is the distance from the particle to the region surface, and Rc is the *cutoff* distance at which the particle and surface no longer interact. The energy of the wall potential is shifted so that the wall-particle interaction energy is 0.0 at the cutoff distance.

For the *lj93* and *lj126* styles, *epsilon* and *sigma* are the usual Lennard-Jones parameters, which determine the strength and size of the particle as it interacts with the wall. Epsilon has energy units. Note that this *epsilon* and *sigma* may be different than any *epsilon* or *sigma* values defined for a pair style that computes particle-particle interactions.

The *lj93* interaction is derived by integrating over a 3d half-lattice of Lennard-Jones 12/6 particles. The *lj126* interaction is effectively a harder, more repulsive wall interaction.

For the *colloid* style, *epsilon* is effectively a Hamaker constant with energy units for the colloid-wall interaction, *R* is the radius of the colloid particle, *D* is the distance from the surface of the colloid particle to

the wall (r-R), and *sigma* is the size of a constituent LJ particle inside the colloid particle. Note that the cutoff distance Rc in this case is the distance from the colloid particle center to the wall.

The *colloid* interaction is derived by integrating over constituent LJ particles of size *sigma* within the colloid particle and a 3d half-lattice of Lennard-Jones 12/6 particles of size *sigma* in the wall.

For the *wall/harmonic* style, *epsilon* is effectively the spring constant K, and has units (energy/distance^2). The input parameter *sigma* is ignored. The minimum energy position of the harmonic spring is at the *cutoff*. This is a repulsive-only spring since the interaction is truncated at the *cutoff*

IMPORTANT NOTE: For all of the styles, you must insure that r is always > 0 for all particles in the group, or LAMMPS will generate an error. This means you cannot start your simulation with particles on the region surface (r = 0) or with particles on the wrong side of the region surface (r < 0). For the *wall/lj93* and *wall/lj126* styles, the energy of the wall/particle interaction (and hence the force on the particle) blows up as r -> 0. The *wall/colloid* style is even more restrictive, since the energy blows up as D = r-R -> 0. This means the finite-size particles of radius R must be a distance larger than R from the region surface. The *harmonic* style is a softer potential and does not blow up as r -> 0, but you must use a large enough *epsilon* that particles always reamin on the correct side of the region surface (r > 0).

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files.

The fix_modify *energy* option is supported by this fix to add the energy of interaction between atoms and the wall to the system's potential energy as part of thermodynamic output.

This fix computes a global scalar energy and a global 3-length vector of forces, which can be accessed by various output commands. The scalar energy is the sum of energy interactions for all particles interacting with the wall represented by the region surface. The 3 vector quantities are the x,y,z components of the total force acting on the wall due to the particles. The scalar and vector values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the [run](run) command.

The forces due to this fix are imposed during an energy minimization, invoked by the minimize command.

IMPORTANT NOTE: If you want the atom/wall interaction energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for this fix.

Restrictions: none

Related commands:

fix wall/lj93, fix wall/lj126, fix wall/colloid, fix wall/gran

Default: none

fix wall/srd command

Syntax:

```
fix ID group-ID wall/srd face arg ... keyword value ...
```

- ID, group-ID are documented in [fix](#) command
- wall/srd = style name of this fix command
- one or more face/arg pairs may be appended
- face = *xlo* or *xhi* or *ylo* or *yhi* or *zlo* or *zhi*

```
  xlo,ylo,zlo arg = EDGE or constant or variable
      EDGE = current lo edge of simulation box
      constant = number like 0.0 or -30.0 (distance units)
      variable = equal-style variable like v_x or v_wiggle
    xhi,yhi,zhi arg = EDGE or constant or variable
      EDGE = current hi edge of simulation box
      constant = number like 50.0 or 100.3 (distance units)
      variable = equal-style variable like v_x or v_wiggle
```

- zero or more keyword/value pairs may be appended
- keyword = *units*

```
  units value = lattice or box
      lattice = the wall position is defined in lattice units
      box = the wall position is defined in simulation box units
```

Examples:

```
fix xwalls all wall/srd xlo EDGE xhi EDGE
fix walls all wall/srd xlo 0.0 ylo 10.0 units box
fix top all wall/srd zhi v_pressdown
```

Description:

Bound the simulation with one or more walls which interact with stochastic reaction dynamics (SRD) particles as slip (smooth) or no-slip (rough) flat surfaces. The wall interaction is actually invoked via the [fix srd](#) command, only on the group of SRD particles it defines, so the group setting for the fix wall/srd command is ignored.

A particle/wall collision occurs if an SRD particle moves outside the wall on a timestep. This alters the position and velocity of the SRD particle and imparts a force to the wall.

The *collision* and *Tsrd* settings specified via the [fix srd](#) command affect the SRD/wall collisions. A *slip* setting for the *collision* keyword means that the tangential component of the SRD particle momentum is preserved. Thus only a normal force is imparted to the wall. The normal component of the new SRD velocity is sampled from a Gaussian distribution at temperature *Tsrd*.

For a *noslip* setting of the *collision* keyword, both the normal and tangential components of the new SRD velocity are sampled from a Gaussian distribution at temperature *Tsrd*. Additionally, a new tangential direction for the SRD velocity is chosen randomly. This collision style imparts both a normal and tangential force to the wall.

Up to 6 walls or faces can be specified in a single command: *xlo*, *xhi*, *ylo*, *yhi*, *zlo*, *zhi*. A *lo* face reflects particles that move to a coordinate less than the wall position, back in the *hi* direction. A *hi* face reflects particles that move to a coordinate higher than the wall position, back in the *lo* direction.

The position of each wall can be specified in one of 3 ways: as the EDGE of the simulation box, as a constant value, or as a variable. If EDGE is used, then the corresponding boundary of the current simulation box is used. If a numeric constant is specified then the wall is placed at that position in the appropriate dimension (x, y, or z). In both the EDGE and constant cases, the wall will never move. If the wall position is a variable, it should be specified as v_name, where name is an equal-style variable name. In this case the variable is evaluated each timestep and the result becomes the current position of the reflecting wall. Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent wall position.

IMPORTANT NOTE: Because the trajectory of the SRD particle is tracked as it collides with the wall, you must insure that r = distance of the particle from the wall, is always > 0 for SRD particles, or LAMMPS will generate an error. This means you cannot start your simulation with SRD particles at the wall position *coord* (r = 0) or with particles on the wrong side of the wall (r < 0).

IMPORTANT NOTE: If you have 2 or more walls that come together at an edge or corner (e.g. walls in the x and y dimensions), then be sure to set the *overlap* keyword to *yes* in the fix srd command, since the walls effectively overlap when SRD particles collide with them. LAMMPS will issue a warning if you do not do this.

IMPORTANT NOTE: The walls of this fix only interact with SRD particles, as defined by the fix srd command. If you are simulating a mixture containing other kinds of particles, then you should typically use another wall command to act on the other particles. Since SRD particles will be colliding both with the walls and the other particles, it is important to insure that the other particle's finite extent does not overlap an SRD wall. If you do not do this, you may generate errors when SRD particles end up "inside" another particle or a wall at the beginning of a collision step.

The *units* keyword determines the meaning of the distance units used to define a wall position, but only when a numeric constant is used. It is not relevant when EDGE or a variable is used to specify a face position.

A *box* value selects standard distance units as defined by the units command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The lattice command must have been previously used to define the lattice spacings.

Here are examples of variable definitions that move the wall position in a time-dependent fashion using equal-style variables.

```
variable ramp equal ramp(0,10)
fix 1 all wall/srd xlo v_ramp

variable linear equal vdisplace(0,20)
fix 1 all wall/srd xlo v_linear

variable wiggle equal swiggle(0.0,5.0,3.0)
fix 1 all wall/srd xlo v_wiggle

variable wiggle equal cwiggle(0.0,5.0,3.0)
fix 1 all wall/srd xlo v_wiggle
```

The ramp(lo,hi) function adjusts the wall position linearly from lo to hi over the course of a run. The displace(c0,velocity) function does something similar using the equation position = c0 + velocity*delta, where delta is the elapsed time.

The swiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, where omega = 2 PI / period:

fix wall/srd command   615

```
position = c0 + A sin(omega*delta)
```

The cwiggle(c0,A,period) function causes the wall position to oscillate sinusoidally according to this equation, which will have an initial wall velocity of 0.0, and thus may impose a gentler perturbation on the particles:

```
position = c0 + A (1 - cos(omega*delta))
```

Restart, fix_modify, output, run start/stop, minimize info:

No information about this fix is written to binary restart files. None of the fix_modify options are relevant to this fix.

This fix computes a global array of values which can be accessed by various output commands. The number of rows in the array is equal to the number of walls defined by the fix. The number of columns is 3, for the x,y,z components of force on each wall.

Note that an outward normal force on a wall will be a negative value for *lo* walls and a positive value for *hi* walls. The array values calculated by this fix are "extensive".

No parameter of this fix can be used with the *start/stop* keywords of the run command. This fix is not invoked during energy minimization.

Restrictions:

Any dimension (xyz) that has an SRD wall must be non-periodic.

Related commands:

fix srd

Default: none

githubAccess_public

Description:

This routine describes how to setup a github account and pull repositories of the CFDEMproject.

Procedure:

- Basically the following steps have to be performed: *git clone* the desired repository
- update your repositories by *git pull*

git clone **the desired repository:**

To clone the public LIGGGHTS repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/LIGGGHTS-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://github.com/CFDEMproject/LIGGGHTS-PUBLIC.git
```

To clone the public CFDEMcoupling repository, open a terminal and execute:

```
git clone git@github.com:CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Note: the git protocol will not work if your computer is behind a firewall which blocks the relevant TCP port, you can use alternatively:

```
git clone https://github.com/CFDEMproject/CFDEMcoupling-PUBLIC.git
```

Update your repositories by *git pull*:

To get the latest version, open a terminal, go to the location of your local installation and type:

```
git pull
```

gran cohesion sjkr2

Syntax:

```
cohesion sjkr2
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This model can be used as part of pair gran and fix wall/gran

The modified simplified JKR - Johnson-Kendall-Roberts (SJKR2) model adds an additional normal force contribution. If two particle are in contact, it adds an additional normal force tending to maintain the contact, which writes

$F = k A$,

where A is the particle contact area and k is the cohesion energy density in J/m3. For *sjkr2*, the sphere-sphere contact area is calculated as

$A = 2*Pi * delta\_n * (2R*)$

If you are using the SJKR2 model, you must also define the cohesion energy density:

```
fix id all property/global cohesionEnergyDensity peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the cohesion energy density (in Energy/Length3 units) between atom type i
```

IMPORTANT NOTE: The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Restrictions:

The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

gran cohesion sjkr

Syntax:

```
cohesion sjkr
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This model can be used as part of pair gran and fix wall/gran

The simplified JKR - Johnson-Kendall-Roberts (SJKR) model adds an additional normal force contribution. If two particle are in contact, it adds an additional normal force tending to maintain the contact, which writes

$F = k\ A$,

where A is the particle contact area and k is the cohesion energy density in J/m3. For *sjkr*, the sphere-sphere contact area is calculated as (http://mathworld.wolfram.com/Sphere-SphereIntersection.html)

$A = Pi/4 * ((dist-Ri-Rj)*(dist+Ri-Rj)*(dist-Ri+Rj)*(dist+Ri+Rj)\ )/(dist*dist)$

where dist is the distance between the particle centers.

If you are using the SJKR model, you must also define the cohesion energy density:

```
fix id all property/global cohesionEnergyDensity peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the cohesion energy density (in Energy/Length3 units) between atom type i
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Restrictions:

The cohesion model has been derived for the Hertzian Style, it may note be appropriate for the Hookean styles.

gran model hertz

Syntax:

```
model hertz other model_type/model_name pairs as described here keyword values
```

• zero or more keyword/value pairs may be appended

```
limitForce values = 'on' or 'off'
    on = ensures that the normal force is never attractive (an artefact that can occur at the e
    off = standard implementation that might lead to attractive forces.
  tangential_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
```

LIGGGHTS vs. LAMMPS Info:

This part of [pair gran](#) and [fix wall/gran](#) is not availabe in LAMMPS.

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d:

$$F = \left(k_n\ \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n\ \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}}\right) + \left(k_t\ \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t\ \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}}\right)$$

$$\underbrace{\phantom{k_n\ \delta n_{ij} - \gamma_n\ vn_{ij}}}_{normal\ force} \qquad \underbrace{\phantom{k_t\ \delta t_{ij} - \gamma_t\ vt_{ij}}}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \leq X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the [tangential model](#) in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the [tangential model](#) chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles

- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

The Kn, Kt, gamma_n, and gamma_t coefficients are calculated as follows from the material properties:

$$k_n = \frac{4}{3} Y^* \sqrt{R^* \delta_n},$$

$$\gamma_n = -2\sqrt{\frac{5}{6}}\, \beta \sqrt{S_n m^*} \geq 0,$$

$$k_t = 8 G^* \sqrt{R^* \delta_n},$$

$$\gamma_t = -2\sqrt{\frac{5}{6}}\, \beta \sqrt{S_t m^*} \geq 0.$$

$$S_n = 2 Y^* \sqrt{R^* \delta_n}, \qquad S_t = 8 G^* \sqrt{R^* \delta_n}$$

$$\beta = \frac{ln(e)}{\sqrt{ln^2(e) + \pi^2}},$$

$$\frac{1}{Y^*} = \frac{\left(1 - v_1^2\right)}{Y_1} + \frac{\left(1 - v_2^2\right)}{Y_2},$$

$$\frac{1}{G^*} = \frac{2(2 + v_1)(1 - v_1)}{Y_1} + \frac{2(2 + v_2)(1 - v_2)}{Y_2}$$

$$\frac{1}{R^*} = \frac{1}{R_1} + \frac{1}{R_2}, \quad \frac{1}{m^*} = \frac{1}{m_1} + \frac{1}{m_2}$$

Y...Young's modulus   G...Shear modulus

v...Poisson ratio   e...coeff. of restitution

To define those material properties, it is mandatory to use multiple fix property/global commands:

gran model hertz   621

```
fix id all property/global youngsModulus peratomtype value_1 value_2 ...
    (value_i=value for Youngs Modulus of atom type i)
fix id all property/global poissonsRatio peratomtype value_1 value_2 ...
    (value_i=value for Poisson ratio of atom type i)
fix id all property/global coefficientRestitution peratomtypepair n_atomtypes value_11 value_12
    (value_ij=value for the coefficient of restitution between atom type i and j; n_atomtypes i
fix id all property/global coefficientFriction peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the (static) coefficient of friction between atom type i and j; n_atomt
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this. **Restrictions:**

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â   541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996). **(Schwager)** Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hertz/stiffness

Syntax:

```
model hertz other model_type/model_name pairs as described here keyword values
```

- zero or more keyword/value pairs may be appended

```
limitForce values = 'on' or 'off'
    on = ensures that the normal force is never attractive (an artefact that can occur at the e
    off = standard implementation that might lead to attractive forces.
  tangential_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d:

$$F = \left(k_n \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}} \right) + \left(k_t \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}} \right.$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad}_{normal\ force} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \leq X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the tangential model in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the tangential model chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles

- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

To define those k_n_specified, k_t_specified, gamma_n_specified, and gamma_t_specified coefficients (material properties), it is mandatory to use multiple [fix property/global](#) commands:

```
fix id all property/global kn peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_n between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global kt peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_t between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global gamman peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the total tangential and normal force components in the formulas above. Thus in the Hookean case, the tangential force between 2 particles grows according to a tangential spring and dash-pot model until Ft/Fn = cof and is then held at Ft = Fn*cof until the particles lose contact. In the Hertzian case, a similar analogy holds, though the spring is no longer linear.

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â   541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hooke

Syntax:

```
model hertz other model_type/model_name pairs as described here keyword values
```

• zero or more keyword/value pairs may be appended

```
tangential_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
  ktToKnUser values = 'on' or 'off'
    on = uses a different kt, namely kt = 2/7 * kn.
    off = standard implementation kt = kn.
  limitForce values = 'on' or 'off'
    on = ensures that the normal force is never attractive (an artefact that can occur at the e
    off = standard implementation that might lead to attractive forces.
  viscous = 'on' or 'off'
    on = restitution coefficient varies with a local Stokes number of the particle. Requires ad
    off = no modification to the restitution coefficient
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This granular model uses the following formula for the frictional force between two granular particles,
when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj.
There is no force between the particles when r > d:

$$F = \left(k_n \; \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n \; \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}} \right) + \left(k_t \; \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t \; \underbrace{vt_{ij}}_{\substack{tangenti \\ relative\ v}} \right)$$

The tangential overlap is truncated to fulfil $F_t \le X_\mu F_n$

In the first term is the normal force between the two particles and the second term is the tangential force.
The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a
shear force and a damping force. The shear force is a "history" effect that accounts for the tangential
displacement ("tangential overlap") between the particles for the duration of the time they are in contact.
This term is controlled by the tangential model in action

gran model hooke   626

Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

The Kn, Kt, gamma_n, and gamma_t coefficients are calculated as follows from the material properties:

$$k_n = \frac{16}{15}\sqrt{R^*}Y^*\left(\frac{15m^*V^2}{16\sqrt{R^*}Y^*}\right)^{1/5},$$

$$\gamma_n = \sqrt{\frac{4m^*k_n}{1+\left(\frac{\pi}{\ln(e)}\right)^2}} \geq 0,$$

$$k_t = k_n,$$

$$\gamma_t = \gamma_n$$

$V...characteristic\ impact\ velocity$

$$S_n = 2Y^* \sqrt{R^* \delta_n}, \qquad S_t = 8G^* \sqrt{R^* \delta_n}$$

$$\beta = \frac{ln(e)}{\sqrt{ln^2(e) + \pi^2}},$$

$$\frac{1}{Y^*} = \frac{\left(1 - v_1^2\right)}{Y_1} + \frac{\left(1 - v_2^2\right)}{Y_2},$$

$$\frac{1}{G^*} = \frac{2(2 + v_1)(1 - v_1)}{Y_1} + \frac{2(2 + v_2)(1 - v_2)}{Y_2}$$

$$\frac{1}{R^*} = \frac{1}{R_1} + \frac{1}{R_2}, \quad \frac{1}{m^*} = \frac{1}{m_1} + \frac{1}{m_2}$$

Y...Young's modulus   G...Shear modulus

v...Poisson ratio   e...coeff.of restitution

To define those material properties, it is mandatory to use multiple [fix property/global](#) commands:

```
fix id all property/global youngsModulus peratomtype value_1 value_2 ...
    (value_i=value for Youngs Modulus of atom type i)
fix id all property/global poissonsRatio peratomtype value_1 value_2 ...
    (value_i=value for Poisson ratio of atom type i)
fix id all property/global coefficientRestitution peratomtypepair n_atomtypes value_11 value_12
    (value_ij=value for the coefficient of restitution between atom type i and j; n_atomtypes i
fix id all property/global coefficientFriction peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for the (static) coefficient of friction between atom type i and j; n_atomt
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

The "characteristic impact velocity" is additionally used for hooke:

```
fix id all property/global characteristicVelocity scalar value
    (value=value for characteristic impact velocity)
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Viscous model:

Using option *viscous* = stokes adapts the coefficient of restitution as proposed by [(Legendre)](#), *viscous* = off performs no modification.

One has to provide the 3 peratomtypepair parameters via a [fix property/global](#) command needed for the viscous damping:

```
fix id all property/global FluidViscosity peratomtypepair n_atomtypes value_11 value_12 .. valu
    (value_ij=value for fluid viscosity between atom type i and j; n_atomtypes is the number of
fix id all property/global CriticalStokes peratomtypepair n_atomtypes value_11 value_12 .. valu
    (value_ij=value for critical Stokes number between atom type i and j; n_atomtypes is the nu
fix id all property/global MaximumRestitution peratomtypepair n_atomtypes value_11 value_12 ..
    (value_ij=value for maximum coefficient of restitution between atom type i and j; n_atomtyp
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the total tangential and normal force components in the formulas above. Thus in the Hookean case, the tangential force between 2 particles grows according to a tangential spring and dash-pot model until Ft/Fn = cof and is then held at Ft = Fn*cof until the particles lose contact. In the Hertzian case, a similar analogy holds, though the spring is no longer linear.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using *viscous*, FluidViscosity has to be > 0 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

viscous = 'off' *tangential_damping* = 'on' *ktToKnUser* = 'off' *limitForce* = 'off'

(Legendre) Legendre, Daniel and Guiraud. Phys. Fluids 17, 097105 (2005).

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran model hooke/stiffness

Syntax:

```
model hooke/stiffness other model_type/model_name pairs as described here keyword values
```

• zero or more keyword/value pairs may be appended

```
absolute_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
  limitForce values = 'on' or 'off'
    on = ensures that the normal force is never attractive (an artefact that can occur at the e
    off = standard implementation that might lead to attractive forces.
  tangential_damping values = 'on' or 'off'
    on = activates tangential damping
    off = no tangential damping
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This granular model uses the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. There is no force between the particles when r > d.

For the case of *absolute_damping* = 'off' (which is default), the specified damping coefficient is multiplied by the effective mass. The forces are implemented as

$$k_n = k_{n,specified},$$

$$\gamma_n = m_{eff}\,\gamma_{n,specified},$$

$$k_t = k_{t,specified},$$

$$\gamma_t = m_{eff}\,\gamma_{t,specified}.$$

For the case of *absolute_damping* = 'on', this multiplication is omitted and the forces become

$$k_n = k_{n,specified},$$

$$\gamma_n = \gamma_{n,specified},$$

$$k_t = k_{t,specified},$$

$$\gamma_t = \gamma_{t,specified}.$$

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a spring force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact. This term is controlled by the tangential model in action Keyword *tangential_damping* can be used to eliminate the second part of the force in tangential direction. The way how the Coulomb friction limit acts is also controlled by the tangential model chosen by the user.

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles which is truncated to satisfy a frictional yield criterion
- rmu = coefficient of rolling friction
- contactradius = contact radius, equal to particle radius - 0.5 * delta_n
- v_n = normal component of the relative velocity of the 2 particles
- v_t = tangential component of the relative velocity of the 2 particles
- w_r = relative rotational velocity of the 2 particles

To define those k_n_specified, k_t_specified, gamma_n_specified, and gamma_t_specified coefficients (material properties), it is mandatory to use multiple fix property/global commands:

```
fix id all property/global kn peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_n between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global kt peratomtypepair n_atomtypes value_11 value_12 .. value_21 value_2
    (value_ij=value for k_t between atom type i and j; n_atomtypes is the number of atom types
fix id all property/global gamman peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat peratomtypepair n_atomtypes value_11 value_12 .. value_21 val
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

If the absolute damping implementation is used (*absolute_damping* = 'on'), the damping coefficients must be named gamman_abs and gammat_abs instead of gamman, gammat as follows:

```
fix id all property/global gamman_abs peratomtypepair n_atomtypes value_11 value_12 .. value_21
```

```
    (value_ij=value for gamma_n between atom type i and j; n_atomtypes is the number of atom ty
fix id all property/global gammat_abs peratomtypepair n_atomtypes value_11 value_12 .. value_21
    (value_ij=value for gamma_t between atom type i and j; n_atomtypes is the number of atom ty
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the total tangential and normal force components in the formulas above. Thus in the Hookean case, the tangential force between 2 particles grows according to a tangential spring and dash-pot model until Ft/Fn = cof and is then held at Ft = Fn*cof until the particles lose contact. In the Hertzian case, a similar analogy holds, though the spring is no longer linear.

Force Limiting:

Note, that not using limitForce might lead to attractive forces between particles and walls, especially in case the coefficient of restitution is small. Be sure you include this key word for the pair style and the wall model if you like to avoid this.

Restrictions:

If using SI units, youngsModulus must be > 5e6 If using CGS units, youngsModulus must be > 5e5 When using the limitForce, the specified coefficient of restitution is only approximate. This might become problematic for low coefficients of resitution as showin in Schwager and Poschel.

Default:

tangential_damping = 'on' *limitForce* = 'off'

(Di Renzo) Alberto Di Renzo, Francesco Paolo Di Maio, CES, 59 (3), p 525â 541 (2004).

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

(Brilliantov) Brilliantov, Spahn, Hertzsch, Poschel, Phys Rev E, 53, p 5382-5392 (1996).

(Schwager) Schwager, Poschel, Gran Matt, 9, p 465-469 (2007).

(Silbert) Silbert, Ertas, Grest, Halsey, Levine, Plimpton, Phys Rev E, 64, p 051302 (2001).

(Zhang) Zhang and Makse, Phys Rev E, 72, p 011301 (2005).

gran rolling_friction cdt

Syntax:

```
rolling_friction cdt
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This model can be used as part of pair gran and fix wall/gran

The constant directional torque (CDT) model adds an additional torque contribution, equal to

torque_rf = rmu*k_n*delta_n*w_r_shear/mag(w_r_shear)*(R*).

w_r_shear is the projection of w_r into the shear plane, where w_r = w1-w2

If the rolling friction model is activated, the coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran rolling_friction epsd2

Syntax:

```
rolling_friction epsd2
```

LIGGGHTS vs. LAMMPS Info:

This part of [pair gran](#) and [fix wall/gran](#) is not availabe in LAMMPS.

Description:

This model can be used as part of [pair gran](#) and [fix wall/gran](#)

The alternative elastic-plastic spring-dashpot (EPSD2) model (see Iwashita and Oda) adds an additional torque contribution. It is similar to the [EPSD model](#), but in contrast to the original model the rolling stiffness k_r is defined as

$$k_r = k_t \cdot R^{*2}$$

where k_t is the abovementioned tangential stiffness. Furthermore, the viscous damping torque M_rd is disabled at all.

The coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

```
This coefficient rmu is equal to the rmu as defined in the CDT model.
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(Ai) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran rolling_friction epsd

Syntax:

```
rolling_friction epsd
```

LIGGGHTS vs. LAMMPS Info:

This part of [pair gran](#) and [fix wall/gran](#) is not availabe in LAMMPS.

Description:

This model can be used as part of [pair gran](#) and [fix wall/gran](#)

The elastic-plastic spring-dashpot (EPSD) model (see Ai et al.) adds an additional torque contribution, equal to

$$M_{\rm r} = M_{\rm r}^{\rm k} + M_{\rm r}^{\rm d}$$

where the torque due to the spring M_rk is calculated as

$$\Delta M_{\rm r}^{\rm k} = -k_{\rm r}\Delta\theta_{\rm r}$$
$$M_{\rm r,t+\Delta t}^{\rm k} = M_{\rm r,t}^{\rm k} + \Delta M_{\rm r}^{\rm k}$$
$$|M_{\rm r,t+\Delta t}^{\rm k}| \leq M_{\rm r}^{\rm m}$$
$$M_{\rm r}^{\rm m} = \mu_{\rm r}R^*F_{\rm n}$$

Here k_r denotes the rolling stiffness and dtheta_r is the incremental relative rotation between the particles. The spring torque is limited by the full mobilisation torque M_rm that is determined by the normal force F_n and the coefficient of rolling friction (rmu) (compare the [CDT model](#)).

The viscous damping torque M_rd is implemented as

$$M_{\rm r,t+\Delta t}^{\rm d} = \begin{cases} -C_{\rm r}\dot{\theta}_{\rm r} & {\rm if}\,|M_{\rm r,t+\Delta t}^{\rm k}| < M_{\rm r}^{\rm m} \\ -fC_{\rm r}\dot{\theta}_{\rm r} & {\rm if}\,|M_{\rm r,t+\Delta t}^{\rm k}| = M_{\rm r}^{\rm m} \end{cases}$$

where in the current implementation the damping is disabled in case of full mobilisation (f = 0). The damping coefficient C_r may be expressed as:

$$C_{\mathrm{r}} = \eta_{\mathrm{r}} C_{\mathrm{r}}^{\mathrm{crit}}$$

$$C_{\mathrm{r}}^{\mathrm{crit}} = 2\sqrt{I_{\mathrm{r}} k_{\mathrm{r}}}$$

$$I_{\mathrm{r}} = \left(\frac{1}{I_i + m_i r_i^2} + \frac{1}{I_j + m_j r_j^2} \right)^{-1}$$

Here I_i/j is the moment of inertia and m_i/j is the mass of the particles i and j, respectively.

The coefficient of rolling friction (rmu) must be defined as

```
fix id all property/global coefficientRollingFriction peratomtypepair n_atomtypes value_11 value_
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

This coefficient rmu is equal to the rmu as defined in the CDT model. In addition to rmu, eta_r is the required material property that must be defined as

```
fix id all property/global coefficientRollingViscousDamping peratomtypepair n_atomtypes value_11
    (value_ij=value for the coefficient of rolling friction between atom type i and j; n_atomtype
```

IMPORTANT NOTE: You have to use atom styles beginning from 1, e.g. 1,2,3,...

(**Ai**) Jun Ai, Jian-Fei Chen, J. Michael Rotter, Jin Y. Ooi, Powder Technology, 206 (3), p 269-282 (2011).

gran tangential history

Syntax:

```
tangential history
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This granular model is based on the general description of granular force interaction as described in pair gran.

The spring part of the tangential force (k_t) is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact.

If this model is chose, then this "tangential overlap" spring force is actually calculated / taken into account.

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the tangential spring and normal force components in the formulas above.

gran tangential no_history

Syntax:

```
tangential no_history
```

LIGGGHTS vs. LAMMPS Info:

This part of pair gran and fix wall/gran is not availabe in LAMMPS.

Description:

This granular model is based on the general description of granular force interaction as described in pair gran.

If this model is chose, then this "tangential overlap" spring force is NOT calculated / taken into account, i.e. $k_t = 0$.

The coefficient of friction cof is the upper limit of the tangential force through the Coulomb criterion Ft = cof*Fn, where Ft and Fn are the tangential spring and normal force components in the formulas above.

group2ndx command

Syntax:

```
group2ndx file group-ID ...
```

- file = name of index file to write out
- zero or more group IDs may be appended

Examples:

```
group2ndx allindex.ndx
group2ndx someindex.ndx upper lower mobile
```

Description:

Write a Gromacs style index file in text format that associates atom IDs with the corresponding group definitions. This index file can be used with in combination with Gromacs analysis tools or to import group definitions into the [fix colvars](#) input file.

Without specifying any group IDs, all groups will be written to the index file. When specifying group IDs, only those groups will be written to the index file. In order to follow the Gromacs conventions, the group *all* will be renamed to *System* in the index file.

Restrictions:

This command requires that atoms have atom IDs, since this is the information that is written to the index file.

This fix is part of the USER-COLVARS package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[group](#), [dump](#), [fix colvars](#)

Default: none

group command

Syntax:

```
group ID style args
```

- ID = user-defined name of the group
- style = *delete* or *region* or *type* or *id* or *molecule* or *variable* or *subtract* or *union* or *intersect*

```
  delete = no args
    region args = region-ID
    type or id or molecule
      args = list of one or more atom types, atom IDs, or molecule IDs
        any entry in list can be a sequence formatted as A:B or A:B:C where
        A = starting index, B = ending index,
        C = increment between indices, 1 if not specified
      args = logical value
        logical = "" or ">=" or "==" or "!="
        value = an atom type or atom ID or molecule ID (depending on style)
      args = logical value1 value2
        logical = ""
        value1,value2 = atom types or atom IDs or molecule IDs (depending on style)
    variable args  = variable-ID
    subtract args = two or more group IDs
    union args = one or more group IDs
    intersect args = two or more group IDs
```

Examples:

```
group edge region regstrip
group water type 3 4
group sub id 10 25 50
group sub id 10 25 50 500:1000
group sub id 100:10000:10
group sub id <= 150
group polyA molecule  50 250
group hienergy variable eng
group boundary subtract all a2 a3
group boundary union lower upper
group boundary intersect upper flow
group boundary delete
```

Description:

Identify a collection of atoms as belonging to a group. The group ID can then be used in other commands such as [fix](#), [compute](#), [dump](#), or [velocity](#) to act on those atoms together.

If the group ID already exists, the group command adds the specified atoms to the group.

The *delete* style removes the named group and un-assigns all atoms that were assigned to that group. Since there is a restriction (see below) that no more than 32 groups can be defined at any time, the *delete* style allows you to remove groups that are no longer needed, so that more can be specified. You cannot delete a group if it has been used to define a current [fix](#) or [compute](#) or [dump](#).

The *region* style puts all atoms in the region volume into the group. Note that this is a static one-time assignment. The atoms remain assigned (or not assigned) to the group even in they later move out of the region volume.

group command   640

The *type*, *id*, and *molecule* styles put all atoms with the specified atom types, atom IDs, or molecule IDs into the group. These 3 styles can use arguments specified in one of two formats.

The first format is a list of values (types or IDs). For example, the 2nd command in the examples above puts all atoms of type 3 or 4 into the group named *water*. Each entry in the list can be a colon-separated sequence A:B or A:B:C, as in two of the examples above. A "sequence" generates a sequence of values (types or IDs), with an optional increment. The first example with 500:1000 has the default increment of 1 and would add all atom IDs from 500 to 1000 (inclusive) to the group sub, along with 10,25,50 since they also appear in the list of values. The second example with 100:10000:10 uses an increment of 10 and would thus would add atoms IDs 100,110,120, ... 9990,10000 to the group sub.

The second format is a *logical* followed by one or two values (type or ID). The 7 valid logicals are listed above. All the logicals except take a single argument. The 3rd example above adds all atoms with IDs from 1 to 150 to the group named *sub*. The logical means "between" and takes 2 arguments. The 4th example above adds all atoms belonging to molecules with IDs from 50 to 250 (inclusive) to the group named polyA.

The *variable* style evaluates a variable to determine which atoms to add to the group. It must be an [atom-style variable](#) previously defined in the input script. If the variable evaluates to a non-zero value for a particular atom, then that atom is added to the specified group.

Atom-style variables can specify formulas that include thermodynamic quantities, per-atom values such as atom coordinates, or per-atom quantities calculated by computes, fixes, or other variables. They can also include Boolean logic where 2 numeric values are compared to yield a 1 or 0 (effectively a true or false). Thus using the *variable* style, is a general way to flag specific atoms to include or exclude from a group.

For example, these lines define a variable "eatom" that calculates the potential energy of each atom and includes it in the group if its potential energy is above the threshhold value -3.0.

```
compute          1 all pe/atom
compute          2 all reduce sum c_1
thermo_style     custom step temp pe c_2
run              0

variable         eatom atom "c_1 > -3.0"
group            hienergy variable eatom
```

Note that these lines

```
compute          2 all reduce sum c_1
thermo_style     custom step temp pe c_2
run              0
```

are necessary to insure that the "eatom" variable is current when the group command invokes it. Because the eatom variable computes the per-atom energy via the pe/atom compute, it will only be current if a run has been performed which evaluated pairwise energies, and the pe/atom compute was actually invoked during the run. Printing the thermodyanmic info for compute 2 insures that this is the case, since it sums the pe/atom compute values (in the reduce compute) to output them to the screen. See the "Variable Accuracy" section of the [variable](#) doc page for more details on insuring that variables are current when they are evaluated between runs.

The *subtract* style takes a list of two or more existing group names as arguments. All atoms that belong to the 1st group, but not to any of the other groups are added to the specified group.

The *union* style takes a list of one or more existing group names as arguments. All atoms that belong to any of the listed groups are added to the specified group.

The *intersect* style takes a list of two or more existing group names as arguments. Atoms that belong to every one of the listed groups are added to the specified group.

A group with the ID *all* is predefined. All atoms belong to this group. This group cannot be deleted.

Restrictions:

There can be no more than 32 groups defined at one time, including "all".

Related commands:

dump, fix, region, velocity

Default:

All atoms belong to the "all" group.

if command

Syntax:

```
if boolean then t1 t2 ... elif boolean f1 f2 ... elif boolean f1 f2 ... else e1 e2 ...
```

- boolean = a Boolean expression evaluated as TRUE or FALSE (see below)
- then = required word
- t1,t2,...,tN = one or more LAMMPS commands to execute if condition is met, each enclosed in quotes
- elif = optional word, can appear multiple times
- f1,f2,...,fN = one or more LAMMPS commands to execute if elif condition is met, each enclosed in quotes (optional arguments)
- else = optional argument
- e1,e2,...,eN = one or more LAMMPS commands to execute if no condition is met, each enclosed in quotes (optional arguments)

Examples:

```
if "${steps} > 1000" then quit
if "$x <= $y" then "print X is smaller = $x" else "print Y is smaller = $y"
if "(${eng} > 0.0) || ($n <1000)" then &
  "timestep 0.005" &
elif $n ${eng_previous}" then "jump file1" else "jump file2"
```

Description:

This command provides an if-then-else capability within an input script. A Boolean expression is evaluted and the result is TRUE or FALSE. Note that as in the examples above, the expression can contain variables, as defined by the [variable](#) command, which will be evaluated as part of the expression. Thus a user-defined formula that reflects the current state of the simulation can be used to issue one or more new commands.

If the result of the Boolean expression is TRUE, then one or more commands (t1, t2, ..., tN) are executed. If it is FALSE, then Boolean expressions associated with successive elif keywords are evaluated until one is found to be true, in which case its commands (f1, f2, ..., fN) are executed. If no Boolean expression is TRUE, then the commands associated with the else keyword, namely (e1, e2, ..., eN), are executed. The elif and else keywords and their associated commands are optional. If they aren't specified and the initial Boolean expression is FALSE, then no commands are executed.

The syntax for Boolean expressions is described below.

Each command (t1, f1, e1, etc) can be any valid LAMMPS input script command, except an [include](#) command, which is not allowed. If the command is more than one word, it must enclosed in quotes, so it will be treated as a single argument, as in the examples above.

IMPORTANT NOTE: If a command itself requires a quoted argument (e.g. a [print](#) command), then double and single quotes can be used and nested in the usual manner, as in the examples above and below. See [Section_commands 2](#) of the manual for more details on using quotes in arguments. Only one of level of nesting is allowed, but that should be sufficient for most use cases.

Note that by using the line continuation character "&", the if command can be spread across many lines, though it is still a single command:

```
if "$a <$b" then &
  "print 'Minimum value = $a'" &
```

```
  "run 1000" &
else &
  'print "Minimum value = $b"' &
  "minimize 0.001 0.001 1000 10000"
```

Note that if one of the commands to execute is [quit](link) (of an invalid LAMMPS command such as "blah"), as in the first example above, then executing the command will cause LAMMPS to halt.

Note that by jumping to a label in the same input script, the if command can be used to break out of a loop. See the [variable delete](link) command for info on how to delete the associated loop variable, so that it can be re-used later in the input script.

Here is an example of a double loop which uses the if and [jump](link) commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label       loopa
variable    a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        '$b > 2' then "print 'Jumping to another script'" "jump in.script break"
  next      b
  jump      in.script loopb
label       break
variable    b delete

next        a
jump        in.script loopa
```

The Boolean expressions for the if and elif keywords have a C-like syntax. Note that each expression is a single argument within the if command. Thus if you want to include spaces in the expression for clarity, you must enclose the entire expression in quotes.

An expression is built out of numbers:

```
0.2, 100, 1.0e20, -15.4, etc
```

and Boolean operators:

```
A == B, A != B, A <B, A  <= B, A > B, A >= B, A && B, A || B, !A
```

Each A and B is a number or a variable reference like $a or ${abc}, or another Boolean expression.

If a variable is used it must produce a number when evaluated and substituted for in the expression, else an error will be generated.

Expressions are evaluated left to right and have the usual C-style precedence: the unary logical NOT operator "!" has the highest precedence, the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator "&&"; and finally the logical OR operator "||" has the lowest precedence. Parenthesis can be used to group one or more portions of an expression and/or enforce a different order of evaluation than what would occur with the default precedence.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. The logical AND operator will return 1.0 if both its arguments are non-zero, else it returns 0.0. The logical OR operator will return 1.0 if either of its arguments is non-zero, else it returns 0.0. The logical NOT operator returns 1.0 if its argument is 0.0, else it returns 0.0.

if command   644

The overall Boolean expression produces a TRUE result if the result is non-zero. If the result is zero, the expression result is FALSE.

Restrictions: none

Related commands:

variable, print

Default: none

improper_style class2 command

improper_style class2/omp command

Syntax:

```
improper_style class2
```

Examples:

```
improper_style class2
improper_coeff 1 100.0 0
improper_coeff * aa 0.0 0.0 0.0 115.06 130.01 115.06
```

Description:

The *class2* improper style uses the potential

$$
\begin{aligned}
E &= E_i + E_{aa} \\
E_i &= K\left[\frac{\chi_{ijkl} + \chi_{kjli} + \chi_{ljik}}{3} - \chi_0\right]^2 \\
E_{aa} &= M_1(\theta_{ijk} - \theta_1)(\theta_{kjl} - \theta_3) + \\
&\quad M_2(\theta_{ijk} - \theta_1)(\theta_{ijl} - \theta_2) + \\
&\quad M_3(\theta_{ijl} - \theta_2)(\theta_{kjl} - \theta_3)
\end{aligned}
$$

where Ei is the improper term and Eaa is an angle-angle term. The 3 X terms in Ei are an average over 3 out-of-plane angles.

The 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L. X_IJKL refers to the angle between the plane of I,J,K and the plane of J,K,L, and the bond JK lies in both planes. Similarly for X_KJLI and X_LJIK. Note that atom J appears in the common bonds (JI, JK, JL) of all 3 X terms. Thus J (the 2nd atom in the quadruplet) is the atom of symmetry in the 3 X angles.

The subscripts on the various theta's refer to different combinations of 3 atoms (I,J,K,L) used to form a particular angle. E.g. Theta_IJL is the angle formed by atoms I,J,L with J in the middle. Theta1, theta2, theta3 are the equilibrium positions of those angles. Again, atom J (the 2nd atom in the quadruplet) is the atom of symmetry in the theta angles, since it is always the center atom.

Since atom J is the atom of symmetry, normally the bonds J-I, J-K, J-L would exist for an improper to be defined between the 4 atoms, but this is not required.

See (Sun) for a description of the COMPASS class2 force field.

Coefficients for the Ei and Eaa formulas must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands.

These are the 2 coefficients for the Ei formula:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

For the Eaa formula, each line in a improper_coeff command in the input script lists 7 coefficients, the first of which is "aa" to indicate they are AngleAngle coefficients. In a data file, these coefficients should be listed under a "AngleAngle Coeffs" heading and you must leave out the "aa", i.e. only list 6 coefficients after the improper type.

- aa
- M1 (energy/distance)
- M2 (energy/distance)
- M3 (energy/distance)
- theta1 (degrees)
- theta2 (degrees)
- theta3 (degrees)

The theta values are specified in degrees, but LAMMPS converts them to radians internally; hence the units of M are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the CLASS2 package. See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

(Sun) Sun, J Phys Chem B 102, 7338-7364 (1998).

improper_coeff command

Syntax:

```
improper_coeff N args
```

- N = improper type (see asterisk form below)
- args = coefficients for one or more improper types

Examples:

```
improper_coeff 1 300.0 0.0
improper_coeff * 80.2 -1 2
improper_coeff *4 80.2 -1 2
```

Description:

Specify the improper force field coefficients for one or more improper types. The number and meaning of the coefficients depends on the improper style. Improper coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the coefficients for multiple improper types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of improper types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

Note that using an improper_coeff command can override a previous setting for the same improper type. For example, these commands set the coeffs for all improper types, then overwrite the coeffs for just improper type 2:

```
improper_coeff * 300.0 0.0
improper_coeff 2 50.0 0.0
```

A line in a data file that specifies improper coefficients uses the exact same format as the arguments of the improper_coeff command in an input script, except that wild-card asterisks should not be used since coefficients for all N types must be listed in the file. For example, under the "Improper Coeffs" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 300.0 0.0
```

The [improper_style class2](#) is an exception to this rule, in that an additional argument is used in the input script to allow specification of the cross-term coefficients. See its doc page for details.

Here is an alphabetic list of improper styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated [improper_coeff](#) command.

Note that there are also additional improper styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the improper section of [this page](#).

- [improper_style none](#) - turn off improper interactions
- [improper_style hybrid](#) - define multiple styles of improper interactions

- [improper_style class2](#) - COMPASS (class 2) improper
- [improper_style cvff](#) - CVFF improper
- [improper_style harmonic](#) - harmonic improper
- [improper_style umbrella](#) - DREIDING improper

Restrictions:

This command must come after the simulation box is defined by a [read_data](#), [read_restart](#), or [create_box](#) command.

An improper style must be defined before any improper coefficients are set, either in the input script or in a data file.

Related commands:

[improper_style](#)

Default: none

improper_style cossq command

improper_style cossq/omp command

Syntax:

```
improper_style cossq
```

Examples:

```
improper_style cossq
improper_coeff 1 4.0 0.0
```

Description:

The *cossq* improper style uses the potential

$$E = \frac{1}{2} K \cos^2\left(\chi - \chi_0\right)$$

where x is the improper angle, x0 is its equilibrium value, and K is a prefactor.

If the 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L then X is the angle between the plane of I,J,K and the plane of J,K,L. Alternatively, you can think of atoms J,K,L as being in a plane, and atom I above the plane, and X as a measure of how far out-of-plane I is with respect to the other 3 atoms.

Note that defining 4 atoms to interact in this way, does not mean that bonds necessarily exist between I-J, J-K, or K-L, as they would in a linear dihedral. Normally, the bonds I-J, I-K, I-L would exist for an improper to be defined between the 4 atoms.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

improper_style cvff command

improper_style cvff/omp command

Syntax:

```
improper_style cvff
```

Examples:

```
improper_style cvff
improper_coeff 1 80.0 -1 4
```

Description:

The *cvff* improper style uses the potential

$$E = K[1 + d\cos(n\phi)]$$

where phi is the improper dihedral angle.

If the 4 atoms in an improper quadruplet (listed in the data file read by the [read_data](#) command) are ordered I,J,K,L then the improper dihedral angle is between the plane of I,J,K and the plane of J,K,L. Note that because this is effectively a dihedral angle, the formula for this improper style is the same as for [dihedral_style harmonic](#).

Note that defining 4 atoms to interact in this way, does not mean that bonds necessarily exist between I-J, J-K, or K-L, as they would in a linear dihedral. Normally, the bonds I-J, I-K, I-L would exist for an improper to be defined between the 4 atoms.

The following coefficients must be defined for each improper type via the [improper_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy)
- d (+1 or -1)
- n (0,1,2,3,4,6)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Related commands:

[improper_coeff](#)

Default: none

improper_style fourier command

improper_style fourier/omp command

Syntax:

```
improper_style fourier
```

Examples:

```
improper_style fourier
improper_coeff 1 100.0 180.0
```

Description:

The *fourier* improper style uses the following potential:

$$E = K[C_0 + C_1 \cos(\omega) + C_2 \cos(2\omega)]$$

where K is the force constant and omega is the angle between the IL axis and the IJK plane:
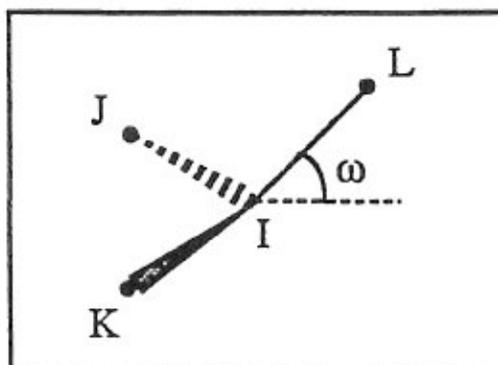
If all parameter (see bellow) is not zero, the all the three possible angles will taken in account.

The following coefficients must be defined for each improper type via the [improper_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy)
- C0 (real)
- C1 (real)
- C2 (real)
- all (integer >= 0)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

improper_style fourier command

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER_MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

improper_style harmonic command

improper_style harmonic/omp command

Syntax:

```
improper_style harmonic
```

Examples:

```
improper_style harmonic
improper_coeff 1 100.0 0
```

Description:

The *harmonic* improper style uses the potential

$$E = K(\chi - \chi_0)^2$$

where X is the improper angle, X0 is its equilibrium value, and K is a prefactor. Note that the usual 1/2 factor is included in K.

If the 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered I,J,K,L then X is the angle between the plane of I,J,K and the plane of J,K,L. Alternatively, you can think of atoms J,K,L as being in a plane, and atom I above the plane, and X as a measure of how far out-of-plane I is with respect to the other 3 atoms.

Note that defining 4 atoms to interact in this way, does not mean that bonds necessarily exist between I-J, J-K, or K-L, as they would in a linear dihedral. Normally, the bonds I-J, I-K, I-L would exist for an improper to be defined between the 4 atoms.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- X0 (degrees)

X0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

improper_style hybrid command

Syntax:

```
improper_style hybrid style1 style2 ...
```

- style1,style2 = list of one or more improper styles

Examples:

```
improper_style hybrid harmonic helix
improper_coeff 1 harmonic 120.0 30
improper_coeff 2 cvff 20.0 -1 2
```

Description:

The *hybrid* style enables the use of multiple improper styles in one simulation. An improper style is assigned to each improper type. For example, impropers in a polymer flow (of improper type 1) could be computed with a *harmonic* potential and impropers in the wall boundary (of improper type 2) could be computed with a *cvff* potential. The assignment of improper type to style is made via the [improper_coeff](#) command or in the data file.

In the improper_coeff command, the first coefficient sets the improper style and the remaining coefficients are those appropriate to that style. In the example above, the 2 improper_coeff commands would set impropers of improper type 1 to be computed with a *harmonic* potential with coefficients 120.0, 30 for K, X0. Improper type 2 would be computed with a *cvff* potential with coefficients 20.0, -1, 2 for K, d, n.

If the improper *class2* potential is one of the hybrid styles, it requires additional AngleAngle coefficients be specified in the data file. These lines must also have an additional "class2" argument added after the improper type. For improper types which are assigned to other hybrid styles, use the style name (e.g. "harmonic") appropriate to that style. The AngleAngle coeffs for that improper type will then be ignored.

An improper style of *none* can be specified as the 2nd argument to the improper_coeff command, if you desire to turn off certain improper types.

Restrictions:

This improper style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the [Making LAMMPS](#) section for more info on packages.

Unlike other improper styles, the hybrid improper style does not store improper coefficient info for individual sub-styles in a [binary restart files](#). Thus when retarting a simulation from a restart file, you need to re-specify improper_coeff commands.

Related commands:

[improper_coeff](#)

Default: none

improper_style none command

Syntax:

```
improper_style none
```

Examples:

```
improper_style none
```

Description:

Using an improper style of none means improper forces are not computed, even if quadruplets of improper atoms were listed in the data file read by the [read_data](#) command.

Restrictions: none

Related commands: none

Default: none

improper_style ring command

improper_style ring/omp command

Syntax:

```
improper_style ring
```

Examples:

```
improper_style ring
improper_coeff 1 8000 70.5
```

Description:

The *ring* improper style uses the potential

$$
\begin{aligned}
E &= \frac{1}{6}K\left(\Delta_{ijl} + \Delta_{ijk} + \Delta_{kjl}\right)^6 \\
\Delta_{ijl} &= \cos\theta_{ijl} - \cos\theta_0 \\
\Delta_{ijk} &= \cos\theta_{ijk} - \cos\theta_0 \\
\Delta_{kjl} &= \cos\theta_{kjl} - \cos\theta_0
\end{aligned}
$$

where K is a prefactor, theta is the angle formed by the atoms specified by (i,j,k,l) indices and theta0 its equilibrium value.

If the 4 atoms in an improper quadruplet (listed in the data file read by the read_data command) are ordered i,j,k,l then theta_*ijl* is the angle between atoms i,j and l, theta_*ijk* is the angle between atoms i,j and k, theta_*kjl* is the angle between atoms j,k, and l.

The "ring" improper style implements the improper potential introduced by Destree et al., in Equation (9) of (Destree). This potential does not affect small amplitude vibrations but is used in an ad-hoc way to prevent the onset of accidentially large amplitude fluctuations leading to the occurrence of a planar conformation of the three bonds i-j, j-k and j-l, an intermediate conformation toward the chiral inversion of a methine carbon. In the "Impropers" section of data file four atoms: i, j, k and l are specified with i,j and l lying on the backbone of the chain and k specifying the chirality of j.

The following coefficients must be defined for each improper type via the improper_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- K (energy/radian^2)
- theta0 (degrees)

theta0 is specified in degrees, but LAMMPS converts it to radians internally; hence the units of K are in energy/radian^2.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

(Destree) M. Destree, F. Laupretre, A. Lyulin, and J.-P. Ryckaert, J Chem Phys, 112, 9632 (2000).

improper_style command

Syntax:

```
improper_style style
```

- style = *none* or *hybrid* or *class2* or *cvff* or *harmonic*

Examples:

```
improper_style harmonic
improper_style cvff
improper_style hybrid cvff harmonic
```

Description:

Set the formula(s) LAMMPS uses to compute improper interactions between quadruplets of atoms, which remain in force for the duration of the simulation. The list of improper quadruplets is read in by a read_data or read_restart command from a data or restart file. Note that the ordering of the 4 atoms in an improper quadruplet determines the the definition of the improper angle used in the formula for each style. See the doc pages of individual styles for details.

Hybrid models where impropers are computed using different improper potentials can be setup using the *hybrid* improper style.

The coefficients associated with an improper style can be specified in a data or restart file or via the improper_coeff command.

All improper potentials store their coefficient data in binary restart files which means improper_style and improper_coeff commands do not need to be re-specified in an input script that restarts a simulation. See the read_restart command for details on how to do this. The one exception is that improper_style *hybrid* only stores the list of sub-styles in the restart file; improper coefficients need to be re-specified.

IMPORTANT NOTE: When both an improper and pair style is defined, the special_bonds command often needs to be used to turn off (or weight) the pairwise interaction that would otherwise exist between a group of 4 bonded atoms.

Here is an alphabetic list of improper styles defined in LAMMPS. Click on the style to display the formula it computes and coefficients specified by the associated improper_coeff command.

Note that there are also additional improper styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the improper section of this page.

- improper_style none - turn off improper interactions
- improper_style hybrid - define multiple styles of improper interactions

- improper_style class2 - COMPASS (class 2) improper
- improper_style cvff - CVFF improper
- improper_style harmonic - harmonic improper
- improper_style umbrella - DREIDING improper

Restrictions:

improper_style command   662

Improper styles can only be set for atom_style choices that allow impropers to be defined.

Most improper styles are part of the MOLECULAR package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info on packages. The doc pages for individual improper potentials tell if it is part of a package.

Related commands:

improper_coeff

Default:

```
improper_style none
```

improper_style umbrella command

improper_style umbrella/omp command

Syntax:

```
improper_style umbrella
```

Examples:

```
improper_style umbrella
improper_coeff 1 100.0 180.0
```

Description:

The *umbrella* improper style uses the following potential, which is commonly referred to as a classic inversion and used in the [DREIDING](#) force field:

$$E = \frac{1}{2}K \left(\frac{1 + cos\omega_0}{sin\omega_0} \right)^2 (cos\omega - cos\omega_0) \qquad \omega_0 \neq 0^o$$

$$E = K(1 - cos\omega) \qquad \omega_0 = 0^o$$

where K is the force constant and omega is the angle between the IL axis and the IJK plane:
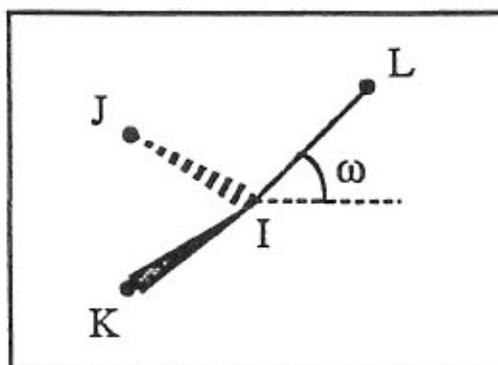
If omega0 = 0 the potential term has a minimum for the planar structure. Otherwise it has two minima at +/- omega0, with a barrier in between.

See [(Mayo)](#) for a description of the DREIDING force field.

The following coefficients must be defined for each improper type via the [improper_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- K (energy)
- omega0 (degrees)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This improper style can only be used if LAMMPS was built with the MOLECULAR package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

improper_coeff

Default: none

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897-8909 (1990),

include command

Syntax:

```
include file
```

- file = filename of new input script to switch to

Examples:

```
include newfile
include in.run2
```

Description:

This command opens a new input script file and begins reading LAMMPS commands from that file. When the new file is finished, the original file is returned to. Include files can be nested as deeply as desired. If input script A includes script B, and B includes A, then LAMMPS could run for a long time.

If the filename is a variable (see the [variable](#) command), different processor partitions can run different input scripts.

Restrictions: none

Related commands:

[variable](#), [jump](#)

Default: none

jump command

Syntax:

```
jump file label
```

- file = filename of new input script to switch to
- label = optional label within file to jump to

Examples:

```
jump newfile
jump in.run2 runloop
jump SELF runloop
```

Description:

This command closes the current input script file, opens the file with the specified name, and begins reading LAMMPS commands from that file. Unlike the [include](#) command, the original file is not returned to, although by using multiple jump commands it is possible to chain from file to file or back to the original file.

If the word "SELF" is used for the filename, then the current input script is re-opened and read again.

IMPORTANT NOTE: The SELF option is not guaranteed to work when the current input script is being read through stdin (standard input), e.g.

```
lmp_g++ <in.script
```

since the SELF option invokes the C-library rewind() call, which may not be supported for stdin on some systems. This can be worked around by using the [-in command-line argument](#) or the [-var command-line argument](#) to pass the script name as a variable to the input script In the latter case, the "fname" [variable](#) could be used in place of SELF. E.g.

```
lmp_g++ -in in.script
```

```
lmp_g++ -var fname n.script <in.script
```

The 2nd argument to the jump command is optional. If specified, it is treated as a label and the new file is scanned (without executing commands) until the label is found, and commands are executed from that point forward. This can be used to loop over a portion of the input script, as in this example. These commands perform 10 runs, each of 10000 steps, and create 10 dump files named file.1, file.2, etc. The [next](#) command is used to exit the loop after 10 iterations. When the "a" variable has been incremented for the tenth time, it will cause the next jump command to be skipped.

```
variable a loop 10
label loop
dump 1 all atom 100 file.$a
run 10000
undump 1
next a
jump in.lj loop
```

If the jump *file* argument is a variable, the jump command can be used to cause different processor partitions to run different input scripts. In this example, LAMMPS is run on 40 processors, with 4 partitions of 10 procs each. An in.file containing the example variable and jump command will cause each partition to run a

jump command
667

different simulation.

```
mpirun -np 40 lmp_ibm -partition 4x10 -in in.file
```

```
variable f world script.1 script.2 script.3 script.4
jump $f
```

Here is an example of a double loop which uses the [if](#) and jump commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label     loopa
variable  a loop 5
  label     loopb
  variable  b loop 5
  print     "A,B = $a,$b"
  run       10000
  if        $b > 2 then "jump in.script break"
  next      b
  jump      in.script loopb
label     break
variable  b delete

next      a
jump      in.script loopa
```

Restrictions:

If you jump to a file and it does not contain the specified label, LAMMPS will come to the end of the file and exit.

Related commands:

[variable](#), [include](#), [label](#), [next](#)

Default: none

kspace_modify command

Syntax:

```
kspace_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *mesh* or *order* or *order/disp* or *overlap* or *minorder* or *force* or *gewald* or *gewald/disp* or *slab* or (nozforce or *compute* or *cutoff/adjust* or *diff* or *kmax/ewald*

```
mesh value = x y z
    x,y,z = grid size in each dimension for long-range Coulombics
  mesh/disp value = x y z
    x,y,z = grid size in each dimension for 1/r^6 dispersion
  order value = N
    N = extent of Gaussian for PPPM or MSM mapping of charge to grid
  order/disp value = N
    N = extent of Gaussian for PPPM mapping of dispersion term to grid
  overlap = yes or no = whether the grid stencil for PPPM is allowed to overlap into more
  minorder value = M
    M = min allowed extent of Gaussian when auto-adjusting to minimize grid communication
  force value = accuracy (force units)
  gewald value = rinv (1/distance units)
    rinv = G-ewald parameter for Coulombics
  gewald/disp value = rinv (1/distance units)
    rinv = G-ewald parameter for dispersion
  slab value = volfactor or nozforce
    volfactor = ratio of the total extended volume used in the
      2d approximation compared with the volume of the simulation domain
  nozforce turns off kspace forces in the z direction
  compute value = yes or no
  cutoff/adjust value = yes or no
  fftbench value = yes or no
  diff value = ad or ik = 2 or 4 FFTs for PPPM in smoothed or non-smoothed mode
  kmax/ewald value = kx ky kz
    kx,ky,kz = number of Ewald sum kspace vectors in each dimension
```

Examples:

```
kspace_modify mesh 24 24 30 order 6
kspace_modify slab 3.0
```

Description:

Set parameters used by the kspace solvers defined by the [kspace_style](#) command. Not all parameters are relevant to all kspace styles.

The *mesh* keyword sets the grid size for kspace style *pppm* or *msm*. In the case of PPPM, this is the FFT mesh, and each dimension must be factorizable into powers of 2, 3, and 5. In the case of MSM, this is the finest scale real-space mesh, and each dimension must be factorizable into powers of 2. When this option is not set, the PPPM or MSM solver chooses its own grid size, consistent with the user-specified accuracy and pairwise cutoff. Values for x,y,z of 0,0,0 unset the option.

The *mesh/disp* keyword sets the grid size for kspace style *pppm/disp*. This is the FFT mesh for long-range dispersion and ach dimension must be factorizable into powers of 2, 3, and 5. When this option is not set, the PPPM solver chooses its own grid size, consistent with the user-specified accuracy and pairwise cutoff. Values for x,y,z of 0,0,0 unset the option.

The *order* keyword determines how many grid spacings an atom's charge extends when it is mapped to the grid in kspace style *pppm* or *msm*. The default for this parameter is 5 for PPPM and 8 for MSM, which means each charge spans 5 or 8 grid cells in each dimension, respectively. For the LAMMPS implementation of MSM, the order can range from 4 to 10 and must be even. For PPPM, the minimum allowed setting is 2 and the maximum allowed setting is 7. The larger the value of this parameter, the smaller that LAMMPS will set the grid size, to achieve the requested accuracy. Conversely, the smaller the order value, the larger the grid size will be. Note that there is an inherent trade-off involved: a small grid will lower the cost of FFTs or MSM direct sum, but a larger order parameter will increase the cost of interpolating charge/fields to/from the grid.

The *order/disp* keyword determines how many grid spacings an atom's dispersion term extends when it is mapped to the grid in kspace style *pppm/disp*. It has the same meaning as the *order* setting for Coulombics.

The *overlap* keyword can be used in conjunction with the *minorder* keyword with the PPPM styles to adjust the amount of communication that occurs when values on the FFT grid are exchangeed between processors. This communication is distinct from the communication inherent in the parallel FFTs themselves, and is required because processors interpolate charge and field values using grid point values owned by neighboring processors (i.e. ghost point communication). If the *overlap* keyword is set to *yes* then this communication is allowed to extend beyond nearest-neighbor processors, e.g. when using lots of processors on a small problem. If it is set to *no* then the communication will be limited to nearest-neighbor processors and the *order* setting will be reduced if necessary, as explained by the *minorder* keyword discussion. The *overlap* keyword is always set to *yes* in MSM.

The *minorder* keyword allows LAMMPS to reduce the *order* setting if necessary to keep the communication of ghost grid point limited to exchanges between nearest-neighbor processors. See the discussion of the *overlap* keyword for details. If the *overlap* keyword is set to *yes*, which is the default, this is never needed. If it set to *no* and overlap occurs, then LAMMPS will reduce the order setting, one step at a time, until the ghost grid overlap only extends to nearest neighbor processors. The *minorder* keyword limits how small the *order* setting can become. The minimum allowed value for PPPM is 2, which is the default. If *minorder* is set to the same value as *order* then no reduction is allowed, and LAMMPS will generate an error if the grid communcation is non-nearest-neighbor and *overlap* is set to *no*. The *minorder* keyword is not currently supported in MSM.

The PPPM order parameter may be reset by LAMMPS when it sets up the FFT grid if the implied grid stencil extends beyond the grid cells owned by neighboring processors. Typically this will only occur when small problems are run on large numbers of processors. A warning will be generated indicating the order parameter is being reduced to allow LAMMPS to run the problem. Automatic adjustment of the order parameter is not supported in MSM.

The *force* keyword overrides the relative accuracy parameter set by the [kspace_style](#) command with an absolute accuracy. The accuracy determines the RMS error in per-atom forces calculated by the long-range solver and is thus specified in force units. A negative value for the accuracy setting means to use the relative accuracy parameter. The accuracy setting is used in conjunction with the pairwise cutoff to determine the number of K-space vectors for style *ewald*, the FFT grid size for style *pppm*, or the real space grid size for style *msm*.

The *gewald* keyword sets the value of the Ewald or PPPM G-ewald parameter for charge as *rinv* in reciprocal distance units. Without this setting, LAMMPS chooses the parameter automatically as a function of cutoff, precision, grid spacing, etc. This means it can vary from one simulation to the next which may not be desirable for matching a KSpace solver to a pre-tabulated pairwise potential. This setting can also be useful if Ewald or PPPM fails to choose a good grid spacing and G-ewald parameter automatically. If the value is set to 0.0, LAMMPS will choose the G-ewald parameter automatically. MSM does not use the *gewald* parameter.

The *gewald/disp* keyword sets the value of the Ewald or PPPM G-ewald parameter for dispersion as *rinv* in reciprocal distance units. It has the same meaning as the *gewald* setting for Coulombics.

The *slab* keyword allows an Ewald or PPPM solver to be used for a systems that are periodic in x,y but non-periodic in z - a [boundary](#) setting of "boundary p p f". This is done by treating the system as if it were periodic in z, but inserting empty volume between atom slabs and removing dipole inter-slab interactions so that slab-slab interactions are effectively turned off. The volfactor value sets the ratio of the extended dimension in z divided by the actual dimension in z. The recommended value is 3.0. A larger value is inefficient; a smaller value introduces unwanted slab-slab interactions. The use of fixed boundaries in z means that the user must prevent particle migration beyond the initial z-bounds, typically by providing a wall-style fix. The methodology behind the *slab* option is explained in the paper by [(Yeh)](#). The *slab* option is also extended to non-neutral systems [(Ballenegger)](#). An alternative slab option can be invoked with the *nozforce* keyword in lieu of the volfactor. This turns off all kspace forces in the z direction. The *nozforce* option is not supported by MSM. For MSM, any combination of periodic, non-periodic, or shrink-wrapped boundaries can be set using [boundary](#) (the slab approximation in not needed). The *slab* keyword is not currently supported by Ewald or PPPM when using a triclinic simulation cell.

The *compute* keyword allows Kspace computations to be turned off, even though a [kspace_style](#) is defined. This is not useful for running a real simulation, but can be useful for debugging purposes or for computing only partial forces that do not include the Kspace contribution. You can also do this by simply not defining a [kspace_style](#), but a Kspace-compatible [pair_style](#) requires a kspace style to be defined. This keyword gives you that option.

The *cutoff/adjust* keyword applies only to MSM. If this option is turned on, the Coulombic cutoff will be automatically adjusted at the beginning of the run to give the desired estimated error. Other cutoffs such as LJ will not be affected. If the grid is not set using the *mesh* command, this command will also attempt to use the optimal grid that minimizes cost using an estimate given by [(Hardy)](#). Note that this cost estimate is not exact, somewhat experimental, and still may not yield the optimal parameters.

The *fftbench* keyword applies only to PPPM. It is on by default. If this option is turned off, LAMMPS will not take the time at the end of a run to give FFT benchmark timings, and will finish a few seconds faster than it would if this option were on.

The *diff* keyword specifies the differentiation scheme used by the PPPM method to compute forces on particles given electrostatic potentials on the PPPM mesh. The *ik* approach is the default for PPPM and is the original formulation used in [(Hockney)](#). It performs differentiation in Kspace, and uses 3 FFTs to transfer each component of the computed fields back to real space for total of 4 FFTs per timestep.

The analytic differentiation *ad* approach uses only 1 FFT to transfer information back to real space for a total of 2 FFTs per timestep. It then performs analytic differentiation on the single quantity to generate the 3 components of the electric field at each grid point. This is sometimes referred to as "smoothed" PPPM. This approach requires a somewhat larger PPPM mesh to achieve the same accuracy as the *ik* method. Currently, only the *ik* method (default) can be used for a triclinic simulation cell with PPPM. The *ad* method is always used for MSM.

IMPORTANT NOTE: Currently, not all PPPM styles support the *ad* option. Support for those PPPM variants will be added later.

The *kmax/ewald* keyword sets the number of kspace vectors in each dimension for kspace style *ewald*. The three values must be positive integers, or else (0,0,0), which unsets the option. When this option is not set, the Ewald sum scheme chooses its own kspace vectors, consistent with the user-specified accuracy and pairwise cutoff. In any case, if kspace style *ewald* is invoked, the values used are printed to the screen and the log file at the start of the run.

Restrictions: none

Related commands:

kspace_style, boundary

Default:

The option defaults are mesh = mesh/disp = 0 0 0, order = order/disp = 5 (PPPM), order = 10 (MSM), minorder = 2, overlap = yes, force = -1.0, gewald = gewald/disp = 0.0, slab = 1.0, compute = yes, cutoff/adjust = yes (MSM), fftbench = yes (PPPM), and diff = ik (PPPM).

(Hockney) Hockney and Eastwood, Computer Simulation Using Particles, Adam Hilger, NY (1989).

(Yeh) Yeh and Berkowitz, J Chem Phys, 111, 3155 (1999).

(Ballenegger) Ballenegger, Arnold, Cerda, J Chem Phys, 131, 094107 (2009).

(Hardy) David Hardy thesis: Multilevel Summation for the Fast Evaluation of Forces for the Simulation of Biomolecules, University of Illinois at Urbana-Champaign, (2006).

LIGGGHTS Users Manual

kspace_style command

Syntax:

```
kspace_style style value
```

- style = *none* or *ewald* or *ewald/disp* or *ewald/omp* or *pppm* or *pppm/cg* or *pppm/disp* or *pppm/tip4p* or *pppm/stagger* or *pppm/disp/tip4p* or *pppm/gpu* or *pppm/omp* or *pppm/cg/omp* or *pppm/tip4p/omp* or *msm* or *msm/cg* or *msm/omp* or *msm/cg/omp*

```
none value = none
  ewald value = accuracy
    accuracy = desired relative error in forces
  ewald/disp value = accuracy
    accuracy = desired relative error in forces
  ewald/omp value = accuracy
    accuracy = desired relative error in forces
  pppm value = accuracy
    accuracy = desired relative error in forces
  pppm/cg value = accuracy (smallq)
    accuracy = desired relative error in forces
    smallq = cutoff for charges to be considered (optional) (charge units)
  pppm/disp value = accuracy
    accuracy = desired relative error in forces
  pppm/tip4p value = accuracy
    accuracy = desired relative error in forces
  pppm/disp/tip4p value = accuracy
    accuracy = desired relative error in forces
  pppm/gpu value = accuracy
    accuracy = desired relative error in forces
  pppm/omp value = accuracy
    accuracy = desired relative error in forces
  pppm/cg/omp value = accuracy
    accuracy = desired relative error in forces
  pppm/tip4p/omp value = accuracy
    accuracy = desired relative error in forces
  pppm/stagger value = accuracy
    accuracy = desired relative error in forces
  msm value = accuracy
    accuracy = desired relative error in forces
  msm/cg value = accuracy (smallq)
    accuracy = desired relative error in forces
    smallq = cutoff for charges to be considered (optional) (charge units)
  msm/omp value = accuracy
    accuracy = desired relative error in forces
  msm/cg/omp value = accuracy (smallq)
    accuracy = desired relative error in forces
    smallq = cutoff for charges to be considered (optional) (charge units)
```

Examples:

```
kspace_style pppm 1.0e-4
kspace_style pppm/cg 1.0e-5 1.0e-6
kspace style msm 1.0e-4
kspace_style none
```

Description:

Define a long-range solver for LAMMPS to use each timestep to compute long-range Coulombic interactions or long-range 1/r^6 interactions. Most of the long-range solvers perform their computation in K-space, hence

the name of this command.

When such a solver is used in conjunction with an appropriate pair style, the cutoff for Coulombic or 1/r^N interactions is effectively infinite. If the Coulombic case, this means each charge in the system interacts with charges in an infinite array of periodic images of the simulation domain.

Note that using a long-range solver requires use of a matching pair style to perform consistent short-range pairwise calculations. This means that the name of the pair style contains a matching keyword to the name of the KSpace style, as in this table:

Pair style	KSpace style
coul/long	ewald or pppm
coul/msm	msm
lj/long or buck/long	disp (for dispersion)
tip4p/long	tip4p

The *ewald* style performs a standard Ewald summation as described in any solid-state physics text.

The *ewald/disp* style adds a long-range dispersion sum option for 1/r^6 potentials and is useful for simulation of interfaces (Veld). It also performs standard Coulombic Ewald summations, but in a more efficient manner than the *ewald* style. The 1/r^6 capability means that Lennard-Jones or Buckingham potentials can be used without a cutoff, i.e. they become full long-range potentials. The *ewald/disp* style can also be used with point-dipoles (Toukmaji) and is currently the only kspace solver in LAMMPS with this capability.

The *pppm* style invokes a particle-particle particle-mesh solver (Hockney) which maps atom charge to a 3d mesh, uses 3d FFTs to solve Poisson's equation on the mesh, then interpolates electric fields on the mesh points back to the atoms. It is closely related to the particle-mesh Ewald technique (PME) (Darden) used in AMBER and CHARMM. The cost of traditional Ewald summation scales as N^(3/2) where N is the number of atoms in the system. The PPPM solver scales as Nlog(N) due to the FFTs, so it is almost always a faster choice (Pollock).

The *pppm/cg* style is identical to the *pppm* style except that it has an optimization for systems where most particles are uncharged. Similarly the *msm/cg* style implements the same optimization for *msm*. The optional *smallq* argument defines the cutoff for the absolute charge value which determines whether a particle is considered charged or not. Its default value is 1.0e-5.

The *pppm/tip4p* style is identical to the *pppm* style except that it adds a charge at the massless 4th site in each TIP4P water molecule. It should be used with pair styles with a *tip4p/long* in their style name.

The *pppm/stagger* style performs calculations using two different meshes, one shifted slightly with respect to the other. This can reduce force aliasing errors and increase the accuracy of the method for a given mesh size. Or a coarser mesh can be used for the same target accuracy, which saves CPU time. However, there is a trade-off since FFTs on two meshes are now performed which increases the compuation required. See (Cerutti), (Neelov), and (Hockney) for details of the method.

For high relative accuracy, using staggered PPPM allows the mesh size to be reduced by a factor of 2 in each dimension as compared to regular PPPM (for the same target accuracy). This can give up to a 4x speedup in the KSpace time (8x less mesh points, 2x more expensive). However, for low relative accuracy, the staggered PPPM mesh size may be essentially the same as for regular PPPM, which means the method will be up to 2x slower in the KSpace time (simply 2x more expensive). For more details and timings, see Section_accelerate.

IMPORTANT NOTE: Using *pppm/stagger* may not give the same increase in the accuracy of energy and pressure as it does in forces, so some caution must be used if energy and/or pressure are quantities of interest,

such as when using a barostat.

The *pppm/disp* and *pppm/disp/tip4p* styles add a mesh-based long-range dispersion sum option for 1/r^6 potentials (Isele-Holder), similar to the *ewald/disp* style. The 1/r^6 capability means that Lennard-Jones or Buckingham potentials can be used without a cutoff, i.e. they become full long-range potentials.

For these styles, it is currently recommended that you set the dispersion mesh size and other parameters explicitly via the kspace_modify command, rather than let LAMMPS set them automatically. For example, a set of parameters that works well for surface systems when using real units is a LJ cutoff of 10 Angstrom, interpolation order = 5 (the default), grid spacing = 4.17 Angstroms, and Ewald parameter = 0.28. These parameters work well for the *ik* differentiation. For the *ad* setting, a smaller grid spacing is needed, e.g. 3 Angstroms. Further information on the influence of the parameters and how to choose them is described in (Isele-Holder).

IMPORTANT NOTE: All of the PPPM styles can be used with single-precision FFTs by using the compiler switch -DFFT_SINGLE for the FFT_INC setting in your lo-level Makefile. This setting also changes some of the PPPM operations (e.g. mapping charge to mesh and interpolating electric fields to particles) to be performed in single precision. This option can speed-up long-range calulations, particularly in parallel or on GPUs. The use of the -DFFT_SINGLE flag is discussed in this section of the manual. MSM does not currently support the -DFFT_SINGLE compiler switch.

The *msm* style invokes a multi-level summation method MSM solver, (Hardy) or (Hardy2), which maps atom charge to a 3d mesh, and uses a multi-level hierarchy of coarser and coarser meshes on which direct coulomb solves are done. This method does not use FFTs and scales as N. It may therefore be faster than the other K-space solvers for relatively large problems when running on large core counts. MSM can also be used for non-periodic boundary conditions and for mixed periodic and non-periodic boundaries.

MSM is most competitive versus Ewald and PPPM when only relatively low accuracy forces, about 1e-4 relative error or less accurate, are needed. Note that use of a larger coulomb cutoff (i.e. 15 angstroms instead of 10 angstroms) provides better MSM accuracy for both the real space and grid computed forces.

Currently the pressure calculation in MSM is expensive, so calculating the pressure at every timestep or using a fixed pressure simulation with MSM will cause the code to run slower.

The specified *accuracy* determines the relative RMS error in per-atom forces calculated by the long-range solver. It is set as a dimensionless number, relative to the force that two unit point charges (e.g. 2 monovalent ions) exert on each other at a distance of 1 Angstrom. This reference value was chosen as representative of the magnitude of electrostatic forces in atomic systems. Thus an accuracy value of 1.0e-4 means that the RMS error will be a factor of 10000 smaller than the reference force.

The accuracy setting is used in conjunction with the pairwise cutoff to determine the number of K-space vectors for style *ewald* or the grid size for style *pppm* or *msm*.

Note that style *pppm* only computes the grid size at the beginning of a simulation, so if the length or triclinic tilt of the simulation cell increases dramatically during the course of the simulation, the accuracy of the simulation may degrade. For example, for a triclinic system with all three tilt factors set to the maximum limit, the PPPM grid should be increased roughly by a factor of 1.5 in the y direction and 2.0 in the z direction as compared to the same system using a cubic orthogonal simulation cell. One way to ensure the accuracy requirement is being met is to run a short simulation at the maximum expected tilt or length, note the required grid size, and then use the kspace_modify *mesh* command to manually set the PPPM grid size to this value.

RMS force errors in real space for *ewald* and *pppm* are estimated using equation 18 of (Kolafa), which is also referenced as equation 9 of (Petersen). RMS force errors in K-space for *ewald* are estimated using equation 11 of (Petersen), which is similar to equation 32 of (Kolafa). RMS force errors in K-space for *pppm* are estimated

using equation 38 of [(Deserno)](). RMS force errors for *msm* are estimated using ideas from chapter 3 of [(Hardy)](), with equation 3.197 of particular note. When using *msm* with non-periodic boundary conditions, it is expected that the error estimation will be too pessimistic. RMS force errors for dipoles when using *ewald/disp* are estimated using equations 33 and 46 of [(Wang)]().

See the [kspace_modify]() command for additional options of the K-space solvers that can be set, including a *force* option for setting an absoulte RMS error in forces, as opposed to a relative RMS error.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate]() of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

More specifically, the *pppm/gpu* style performs charge assignment and force interpolation calculations on the GPU. These processes are performed either in single or double precision, depending on whether the -DFFT_SINGLE setting was specified in your lo-level Makefile, as discussed above. The FFTs themselves are still calculated on the CPU. If *pppm/gpu* is used with a GPU-enabled pair style, part of the PPPM calculation can be performed concurrently on the GPU while other calculations for non-bonded and bonded force calculation are performed on the CPU.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP, and OPT packages respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS]() section for more info.

See [Section_accelerate]() of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

Note that the long-range electrostatic solvers in LAMMPS assume conducting metal (tinfoil) boundary conditions for both charge and dipole interactions. Vacuum boundary conditions are not currently supported.

The *ewald/disp*, *ewald*, *pppm*, and *msm* styles support non-orthogonal (triclinic symmetry) simulation boxes. However, triclinic simulation cells may not yet be supported by suffix versions of these styles (such as *pppm/cuda*).

All of the kspace styles are part of the KSPACE package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS]() section for more info. Note that the KSPACE package is installed by default.

For MSM, a simulation must be 3d and one can use any combination of periodic, non-periodic, or shrink-wrapped boundaries (specified using the [boundary]() command).

For Ewald and PPPM, a simulation must be 3d and periodic in all dimensions. The only exception is if the slab option is set with [kspace_modify](), in which case the xy dimensions must be periodic and the z dimension must be non-periodic.

Related commands:

[kspace_modify](), [pair_style lj/cut/coul/long](), [pair_style lj/charmm/coul/long](), [pair_style lj/long/coul/long](), [pair_style buck/coul/long]()

Default:

```
kspace_style none
```

kspace_style command   676

(Darden) Darden, York, Pedersen, J Chem Phys, 98, 10089 (1993).

(Deserno) Deserno and Holm, J Chem Phys, 109, 7694 (1998).

(Hockney) Hockney and Eastwood, Computer Simulation Using Particles, Adam Hilger, NY (1989).

(Kolafa) Kolafa and Perram, Molecular Simualtion, 9, 351 (1992).

(Petersen) Petersen, J Chem Phys, 103, 3668 (1995).

(Wang) Wang and Holm, J Chem Phys, 115, 6277 (2001).

(Pollock) Pollock and Glosli, Comp Phys Comm, 95, 93 (1996).

(Cerutti) Cerutti, Duke, Darden, Lybrand, Journal of Chemical Theory and Computation 5, 2322 (2009)

(Neelov) Neelov, Holm, J Chem Phys 132, 234103 (2010)

(Veld) In 't Veld, Ismail, Grest, J Chem Phys, 127, 144711 (2007).

(Toukmaji) Toukmaji, Sagui, Board, and Darden, J Chem Phys, 113, 10913 (2000).

(Isele-Holder) Isele-Holder, Mitchell, Ismail, J Chem Phys, 137, 174107 (2012).

(Hardy) David Hardy thesis: Multilevel Summation for the Fast Evaluation of Forces for the Simulation of Biomolecules, University of Illinois at Urbana-Champaign, (2006).

(Hardy) Hardy, Stone, Schulten, Parallel Computing 35 (2009) 164-177.

label command

Syntax:

```
label ID
```

- ID = string used as label name

Examples:

```
label xyz
label loop
```

Description:

Label this line of the input script with the chosen ID. Unless a jump command was used previously, this does nothing. But if a [jump](#) command was used with a label argument to begin invoking this script file, then all command lines in the script prior to this line will be ignored. I.e. execution of the script will begin at this line. This is useful for looping over a section of the input script as discussed in the [jump](#) command.

Restrictions: none

Related commands: none

Default: none

lattice command

Syntax:

```
lattice style scale keyword values ...
```

- style = *none* or *sc* or *bcc* or *fcc* or *hcp* or *diamond* or *sq* or *sq2* or *hex* or *custom*
- scale = scale factor between lattice and simulation box

    ```
    scale = reduced density rho* (for LJ units)
    scale = lattice constant in distance units (for all other units)
    ```
- zero or more keyword/value pairs may be appended
- keyword = *origin* or *orient* or *spacing* or *a1* or *a2* or *a3* or *basis*

```
  origin values = x y z
     x,y,z = fractions of a unit cell (0 <= x,y,z <1)
  orient values = dim i j k
    dim = x or y or z
    i,j,k = integer lattice directions
  spacing values = dx dy dz
    dx,dy,dz = lattice spacings in the x,y,z box directions
  a1,a2,a3 values = x y z
     x,y,z = primitive vector components that define unit cell
  basis values = x y z
     x,y,z = fractional coords of a basis atom (0  <= x,y,z <1)
```

Examples:

```
lattice fcc 3.52
lattice hex 0.85
lattice sq 0.8 origin 0.0 0.5 0.0 orient x 1 1 0 orient y −1 1 0
lattice custom 3.52 a1 1.0 0.0 0.0 a2 0.5 1.0 0.0 a3 0.0 0.0 0.5 &
                basis 0.0 0.0 0.0 basis 0.5 0.5 0.5
lattice none 2.0
```

Description:

Define a lattice for use by other commands. In LAMMPS, a lattice is simply a set of points in space, determined by a unit cell with basis atoms, that is replicated infinitely in all dimensions. The arguments of the lattice command can be used to define a wide variety of crystallographic lattices.

A lattice is used by LAMMPS in two ways. First, the [create_atoms](#) command creates atoms on the lattice points inside the simulation box. Note that the [create_atoms](#) command allows different atom types to be assigned to different basis atoms of the lattice. Second, the lattice spacing in the x,y,z dimensions implied by the lattice, can be used by other commands as distance units (e.g. [create_box](#), [region](#) and [velocity](#)), which are often convenient to use when the underlying problem geometry is atoms on a lattice.

The lattice style must be consistent with the dimension of the simulation - see the [dimension](#) command. Styles *sc* or *bcc* or *fcc* or *hcp* or *diamond* are for 3d problems. Styles *sq* or *sq2* or *hex* are for 2d problems. Style *custom* can be used for either 2d or 3d problems.

A lattice consists of a unit cell, a set of basis atoms within that cell, and a set of transformation parameters (scale, origin, orient) that map the unit cell into the simulation box. The vectors a1,a2,a3 are the edge vectors of the unit cell. This is the nomenclature for "primitive" vectors in solid-state crystallography, but in LAMMPS the unit cell they determine does not have to be a "primitive cell" of minimum volume.

A lattice of style *none* does not define a unit cell and basis set, so it cannot be used with the <u>create_atoms</u> command. However it does define a lattice spacing via the specified scale parameter. As explained above the lattice spacings in x,y,z can be used by other commands as distance units. No additional keyword/value pairs can be specified for the *none* style. By default, a "lattice none 1.0" is defined, which means the lattice spacing is the same as one distance unit, as defined by the <u>units</u> command.

Lattices of style *sc*, *fcc*, *bcc*, and *diamond* are 3d lattices that define a cubic unit cell with edge length = 1.0. This means a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1. Style *hcp* has a1 = 1 0 0, a2 = 0 sqrt(3) 0, and a3 = 0 0 sqrt(8/3). The placement of the basis atoms within the unit cell are described in any solid-state physics text. A *sc* lattice has 1 basis atom at the lower-left-bottom corner of the cube. A *bcc* lattice has 2 basis atoms, one at the corner and one at the center of the cube. A *fcc* lattice has 4 basis atoms, one at the corner and 3 at the cube face centers. A *hcp* lattice has 4 basis atoms, two in the z = 0 plane and 2 in the z = 0.5 plane. A *diamond* lattice has 8 basis atoms.

Lattices of style *sq* and *sq2* are 2d lattices that define a square unit cell with edge length = 1.0. This means a1 = 1 0 0 and a2 = 0 1 0. A *sq* lattice has 1 basis atom at the lower-left corner of the square. A *sq2* lattice has 2 basis atoms, one at the corner and one at the center of the square. A *hex* style is also a 2d lattice, but the unit cell is rectangular, with a1 = 1 0 0 and a2 = 0 sqrt(3) 0. It has 2 basis atoms, one at the corner and one at the center of the rectangle.

A lattice of style *custom* allows you to specify a1, a2, a3, and a list of basis atoms to put in the unit cell. By default, a1 and a2 and a3 are 3 orthogonal unit vectors (edges of a unit cube). But you can specify them to be of any length and non-orthogonal to each other, so that they describe a tilted parallelepiped. Via the *basis* keyword you add atoms, one at a time, to the unit cell. Its arguments are fractional coordinates (0.0 <= x,y,z < 1.0), so that a value of 0.5 means a position half-way across the unit cell in that dimension.

This sub-section discusses the arguments that determine how the idealized unit cell is transformed into a lattice of points within the simulation box.

The *scale* argument determines how the size of the unit cell will be scaled when mapping it into the simulation box. I.e. it determines a multiplicative factor to apply to the unit cell, to convert it to a lattice of the desired size and distance units in the simulation box. The meaning of the *scale* argument depends on the <u>units</u> being used in your simulation.

For all unit styles except *lj*, the scale argument is specified in the distance units defined by the unit style. For example, in *real* or *metal* units, if the unit cell is a unit cube with edge length 1.0, specifying scale = 3.52 would create a cubic lattice with a spacing of 3.52 Angstroms. In *cgs* units, the spacing would be 3.52 cm.

For unit style *lj*, the scale argument is the Lennard-Jones reduced density, typically written as rho*. LAMMPS converts this value into the multiplicative factor via the formula "factor^dim = rho/rho*", where rho = N/V with V = the volume of the lattice unit cell and N = the number of basis atoms in the unit cell (described below), and dim = 2 or 3 for the dimensionality of the simulation. Effectively, this means that if LJ particles of size sigma = 1.0 are used in the simulation, the lattice of particles will be at the desired reduced density.

The *origin* option specifies how the unit cell will be shifted or translated when mapping it into the simulation box. The x,y,z values are fractional values (0.0 <= x,y,z < 1.0) meaning shift the lattice by a fraction of the lattice spacing in each dimension. The meaning of "lattice spacing" is discussed below.

The *orient* option specifies how the unit cell will be rotated when mapping it into the simulation box. The *dim* argument is one of the 3 coordinate axes in the simulation box. The other 3 arguments are the crystallographic direction in the lattice that you want to orient along that axis, specified as integers. E.g. "orient x 2 1 0" means the x-axis in the simulation box will be the [210] lattice direction. The 3 lattice directions you specify must be mutually orthogonal and obey the right-hand rule, i.e. (X cross Y) points in the Z direction. Note that this description is really only valid for orthogonal lattices. If you are using the more general lattice style *custom*

with non-orthogonal a1,a2,a3 vectors, then think of the 3 *orient* options as creating a 3x3 rotation matrix which is applied to a1,a2,a3 to rotate the original unit cell to a new orientation in the simulation box.

Several LAMMPS commands have the option to use distance units that are inferred from "lattice spacing" in the x,y,z box directions. E.g. the region command can create a block of size 10x20x20, where 10 means 10 lattice spacings in the x direction.

The *spacing* option sets the 3 lattice spacings directly. All must be non-zero (use 1.0 for dz in a 2d simulation). The specified values are multiplied by the multiplicative factor described above that is associated with the scale factor. Thus a spacing of 1.0 means one unit cell independent of the scale factor. This option can be useful if the spacings LAMMPS computes are inconvenient to use in subsequent commands, which can be the case for non-orthogonal or rotated lattices.

If the *spacing* option is not specified, the lattice spacings are computed by LAMMPS in the following way. A unit cell of the lattice is mapped into the simulation box (scaled, shifted, rotated), so that it now has (perhaps) a modified size and orientation. The lattice spacing in X is defined as the difference between the min/max extent of the x coordinates of the 8 corner points of the modified unit cell. Similarly, the Y and Z lattice spacings are defined as the difference in the min/max of the y and z coordinates.

Note that if the unit cell is orthogonal with axis-aligned edges (not rotated via the *orient* keyword), then the lattice spacings in each dimension are simply the scale factor (described above) multiplied by the length of a1,a2,a3. Thus a *hex* style lattice with a scale factor of 3.0 Angstroms, would have a lattice spacing of 3.0 in x and 3*sqrt(3.0) in y.

IMPORTANT NOTE: For non-orthogonal unit cells and/or when a rotation is applied via the *orient* keyword, then the lattice spacings may be less intuitive. In particular, in these cases, there is no guarantee that the lattice spacing is an integer multiple of the periodicity of the lattice in that direction. Thus, if you create an orthogonal periodic simulation box whose size in a dimension is a multiple of the lattice spacing, and then fill it with atoms via the create_atoms command, you will NOT necessarily create a periodic system. I.e. atoms may overlap incorrectly at the faces of the simulation box.

Regardless of these issues, the values of the lattice spacings LAMMPS calculates are printed out, so their effect in commands that use the spacings should be decipherable.

Restrictions:

The *a1,a2,a3,basis* keywords can only be used with style *custom*.

Related commands:

dimension, create_atoms, region

Default:

```
lattice none 1.0
```

For other lattice styles, the option defaults are origin = 0.0 0.0 0.0, orient = x 1 0 0, orient = y 0 1 0, orient = z 0 0 1, a1 = 1 0 0, a2 = 0 1 0, and a3 = 0 0 1.

LIGGGHTS 2.X Coding Manual

Introduction:

This is a short coding manual for LIGGGHTS 2.X

A short list of guidelines

- class data members should be denoted with a _ (like *x_*)
- inline access functions should be used to get access to *x_*, like *x()*
- class data members , both *x_particle_* and *xParticle_* are allowed
- template params should be written in LARGE_CAPS
- file names like fix_mesh_gran.cpp, mesh_mover.h
- fix style fix move/mesh/gran is in file fix_move_mesh_gran.cpp/h
- header guard in file fix_foo_model.h like LMP_FIX_FOO_MODEL_H
- coding format should be ANSI
- for fixes, use error->fix_error to throw error messages in constructors, this enhances object orientation

Files which are especially

LIGGGHTS 2.X vs. LIGGGHTS 1.5.3 - syntax changes

Introduction:

This is a short outline of the most important changes in LIGGGHTS 2.X compared to LIGGGHTS 1.5.3 regarding the syntax of major commands. The motivation for these changes was to make the script language more readable and to improve extendability (in the sense of object oriented programming) with respect to modelling approaches.

Commands covered by this tutorial:

- atom_style sphere (formerly atom_style granular)
- dump mesh/stl (formerly dump stl)
- dump mesh/vtk (formerly dump mesh/gran/VTK)
- fix heat/gran
- fix mesh/surface (formerly fix mesh/gran)
- fix mesh/surface/stress (formerly fix mesh/gran/stressanalysis)
- fix move/mesh (formerly fix move/mesh/gran)
- fix wall/gran/*
- pair_style gran/*

Changes in syntax for each command:

Changes are indicated as follows

```
OLD: old_syntax
NEW: new_syntax
```

where *old_syntax* refers to the syntax used in LIGGGHTS 1.5.3 and before, and *new_syntax* refers to the syntax used in LIGGGHTS 2.X

atom_style sphere (formerly atom_style granular):

```
OLD: atom_style granular
NEW: atom_style sphere
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS 2.0

For details, see atom_style sphere.

dump mesh/stl:

```
OLD: dmpstl all stl 300 post/dump*.stl
NEW: dmpstl all mesh/stl 300 post/dump*.stl
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS 2.0

dump mesh/vtk:

```
OLD: dmpstl all mesh/gran/VTK 300 post/dump*.stl id
NEW: dmpstl all mesh/vtk 300 post/dump*.stl id
```

NOTE: For compatibility reasons, the old syntax can still be used in LIGGGHTS 2.0

---

fix heat/gran:

```
OLD: fix ID group-ID heat/gran 273.15
NEW: fix ID group-ID heat/gran initial_temperature 273.15

OLD: fix ID group-ID heat/gran 273.15 + activate area correction via fix property/global
NEW: fix ID group-ID heat/gran initial_temperature 273.15 area_correction on
```

---

fix mesh/surface (formerly fix mesh/gran):

```
OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1

OLD: fix ID group-ID mesh/gran mesh.stl 1 0.001 0. 0. 0. -90. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 scale 0.001 rotate axis 1. 0. 0. angle -

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 1. 2. 3. 0. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 move 1. 2. 3

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. conveyor 5. 0. 0.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 surface_vel 5. 0. 0.

OLD: fix ID group-ID mesh/gran mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. rotate 0. 0. 0. 1. 0. 0. 5.
NEW: fix ID group-ID mesh/surface file mesh.stl type 1 surface_ang_vel origin 0. 0. 0. axis  1.
```

For details, see [fix mesh/surface](#).

---

fix mesh/surface/stress (formerly fix mesh/gran/stressanalysis):

```
OLD: fix ID group-ID mesh/gran/stressanalysis mesh.stl 1 1.0 0. 0. 0. 0. 0. 0. finnie yes
NEW: fix ID group-ID fix mesh/surface/stress file mesh.stl type 1 wear finnie
```

For details, see [fix mesh/surface/stress](#).

---

fix move/mesh (formerly fix move/mesh/gran):

```
OLD: fix ID group-ID move/mesh/gran wiggle -0.1 0. 0. 0.02 cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 wiggle amplitude -0.1 0. 0. period 0.02

OLD: fix ID group-ID move/mesh/gran rotate 0. 0. 0. 0. 0. 1. 0.05 cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 rotate origin 0. 0. 0. axis  0. 0. 1. period 0.05

OLD: fix ID group-ID move/mesh/gran linear 20. 20. 0. cad1 1
NEW: fix ID group-ID move/mesh mesh cad1 linear 20. 20. 0.
```

NOTES:
- The trailing "1" for the *old_syntax* was no longer used in LIGGGHTS 1.5.3
- For compatibility reasons, the command name *fix move/mesh/gran* can be used in LIGGGHTS 2.0. However, the syntax has to follow the LIGGGHTS 2.0 syntax.

For details, see [fix move/mesh](#).

---

fix wall/gran/*:

Changes in syntax for each command: 684

```
OLD: fix ID group-ID wall/gran/hertz/history 1 0 mesh/gran 2 cad1 cad2
NEW: fix ID group-ID wall/gran/hertz/history mesh n_meshes 2 meshes cad1 cad2


OLD: fix ID group-ID wall/gran/hertz/history 1 0 xplane -0.5 0.5 1
NEW: fix ID1 group-ID wall/gran/hertz/history type 1 xplane -0.5
     fix ID2 group-ID wall/gran/hertz/history type 1 xplane 0.5


OLD: fix ID group-ID wall/gran/hertz/history 3 0 xplane -0.5 0.5 1
NEW: fix ID1 group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 rolling_friction cdt
     fix ID2 group-ID wall/gran/hertz/history primitive type 1 xplane  0.5 rolling_friction cdt


OLD: fix ID group-ID wall/gran/hertz/history 1 1 xplane -0.5 0.5 1
NEW: fix ID group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 0.5 cohesion sjkr


OLD: fix ID group-ID wall/gran/hertz/history 0 0 zcylinder 0.05 1
NEW: fix ID group-ID wall/gran/hertz/history primitive type 1 zcylinder 0.05 0. 0. tangential_d
```

NOTES:
- Same applies for hooke/history, hooke, hooke/history/simple, hertz/history/simple
- *sjkr* stands for 'simplified JKR (Johnson-Kendall-Roberts)' model, and *cdt* for 'constant directional torque' model
- styles *xplane*, *yplane*, *zplane* take only one arg now (the wall position), if you want two walls you have to use two fix commands
- in addition to the existing style *zcylinder*, there is now *xcylinder* and *ycylinder* as well
- *xcylinder ycylinder* and *zcylinder* take 3 args: the cylinder radius, and the location of the axis in the other two dimensions

For details, see fix wall/gran.

---

pair_style gran/*:

```
OLD: pair_style gran/hertz/history 1 0
NEW: pair_style gran/hertz/history


OLD: pair_style gran/hertz/history 3 0
NEW: pair_style gran/hertz/history rolling_friction cdt


OLD: pair_style gran/hertz/history 1 1
NEW: pair_style gran/hertz/history cohesion sjkr


OLD: pair_style gran/hertz/history 0 0
NEW: pair_style gran/hertz/history tangential_damping off
```

NOTES:
- Same applies for hooke/history, hooke, hooke/history/simple, hertz/history/simple
- *sjkr* stands for "simplified JKR (Johnson-Kendall-Roberts)" model, and *cdt* for "constant directional torque" model

For details, see pair_style gran.

LIGGGHTS 3.X vs. LIGGGHTS 2.X - syntax changes

Introduction:

This is a short outline of the most important changes in LIGGGHTS 3.X compared to LIGGGHTS 2.X regarding the syntax of major commands. The motivation for these changes was to make the script language more readable and to improve extendability (in the sense of object oriented programming) with respect to modelling approaches.

Commands covered by this tutorial:

- fix wall/gran/*
- pair_style gran/*

Changes in syntax for each command:

Changes are indicated as follows

```
OLD: old_syntax
NEW: new_syntax
```

where *old_syntax* refers to the syntax used in LIGGGHTS 2.X and before, and *new_syntax* refers to the syntax used in LIGGGHTS 3.X.

fix wall/gran/*:

```
OLD: fix ID group-ID wall/gran/* WALL-OPTIONS
NEW: fix ID group-ID wall/gran MODEL-SELECTION WALL-OPTIONS MODEL-SETTINGS
```

MODEL-SELECTION

MODEL-SELECTION = model *M* [tangential *T*] [cohesion *C*] [rolling_friction *R*]

M = hooke | hooke/stiffness | hooke/hysteresis | hertz | hertz/stiffness

T = **no_history** | history

C = **off** | sjkr | sjkr2 | hamaker

R = **off** | cdt | epsd

MODEL-SETTINGS

MODEL-SELECTION = [tangential_damping (**on**|off)] [absolute_damping (on|**off**)] [viscous (on|**off**)]

Examples

```
OLD: fix ID group-ID wall/gran/hertz/history mesh n_meshes 2 meshes cad1 cad2
NEW: fix ID group-ID wall/gran model hertz tangential history mesh n_meshes 2 meshes cad1 cad2

OLD: fix ID1 group-ID wall/gran/hooke/history type 1 xplane -0.5
NEW: fix ID1 group-ID wall/gran model hooke tangential history type 1 xplane -0.5
```

```
OLD: fix ID1 group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 rolling_friction cdt
NEW: fix ID1 group-ID wall/gran model hertz tangential history rolling_friction cdt primitive t

OLD: fix ID group-ID wall/gran/hertz/history primitive type 1 xplane -0.5 0.5 cohesion sjkr
NEW: fix ID group-ID wall/gran model hertz tangential history cohesion sjkr primitive type 1 xp

OLD: fix ID group-ID wall/gran/hertz/history primitive type 1 zcylinder 0.05 0. 0. tangential_d
NEW: fix ID group-ID wall/gran model hertz tangential history primitive type 1 zcylinder 0.05 0
```

pair_style gran/*:

```
OLD: pair_style gran/* MODEL-SETTINGS
NEW: pair_style gran   MODEL-SELECTION MODEL-SETTINGS
```

MODEL-SELECTION

Same as in fix wall/gran/*

MODEL-SETTINGS

Same as in fix wall/gran/*

Examples

```
OLD: pair_style gran/hertz/history
NEW: pair_style gran model hertz tangential history

OLD: pair_style gran/hertz/history rolling_friction cdt
NEW: pair_style gran model hertz tangential history rolling_friction cdt

OLD: pair_style gran/hertz/history cohesion sjkr
NEW: pair_style gran model hertz tangential history cohesion sjkr

OLD: pair_style gran/hertz/history tangential_damping off
OLD: pair_style gran model hertz tangential history tangential_damping off
```

log command

Syntax:

```
log file keyword
```

- file = name of new logfile
- keyword = *append* if output should be appended to logfile (optional)

Examples:

```
log log.equil
log log.equil append
```

Description:

This command closes the current LAMMPS log file, opens a new file with the specified name, and begins logging information to it. If the specified file name is *none*, then no new log file is opened. If the optional keyword *append* is specified, then output will be appended to an existing log file, instead of overwriting it.

If multiple processor partitions are being used, the file name should be a variable, so that different processors do not attempt to write to the same log file.

The file "log.lammps" is the default log file for a LAMMPS run. The name of the initial log file can also be set by the command-line switch -log. See [Section_start 6](#) for details.

Restrictions: none

Related commands: none

Default:

The default LAMMPS log file is named log.lammps

LIGGGHTS Documentation, Version 3.X

LIGGGHTS stands for LAMMPS Improved for General Granular and Granular Heat Transfer Simulations. It is part of the CFDEMproject, www.cfdem.com

The core developers of LIGGGHTS are Christoph Kloss (DCS Computing GmbH, Linz and JKU Linz) and Richard Berger (JKU Linz), with major contributions from Philippe Seil, Andreas Aigner and Stefan Amberger (all JKU Linz) and Christoph Goniva (DCS Computing GmbH, Linz and JKU Linz)

CFDEMproject has more information about the code and its uses. For questions about the code, please use the forums at CFDEMproject.

LIGGGHTS is based on LAMMPS (see below), and so is its manual. So if the manual says 'LAMMPS', you could read 'LIGGGHTS' instead. However, we want to make clear which parts of the code and framework stem from the LAMMPS base.

LIGGGHTS Version info:

All LIGGGHTS versions are based on a specific version of LAMMPS, as printed in the file src/version.h LIGGGHTS version are identidied by a version number (e.g. '3.0'), a branch name (e.g. 'LIGGGHTS-PUBLIC' for the public release of LIGGGHTS), compilation info (date / time stamp and user name), and a LAMMPS version number (which is the LAMMPS version that the LIGGGHTS release is based on). For info on the LAMMPS version, see below.

LAMMPS Version info:

The LAMMPS "version" is the date when it was released, such as 1 May 2010. LAMMPS is updated continuously. Whenever we fix a bug or add a feature, we release it immediately, and post a notice on this page of the WWW site. Each dated copy of LAMMPS contains all the features and bug-fixes up to and

including that version date. The version date is printed to the screen and logfile every time you run LAMMPS. It is also in the file src/version.h and in the LAMMPS directory name created when you unpack a tarball, and at the top of the first page of the manual (this page).

- If you browse the HTML doc pages on the LAMMPS WWW site, they always describe the most current version of LAMMPS.
- If you browse the HTML doc pages included in your tarball, they describe the version you have.
- The PDF file on the WWW site or in the tarball is updated about once per month. This is because it is large, and we don't want it to be part of every patch.
- There is also a Developer.pdf file in the doc directory, which describes the internal structure and algorithms of LAMMPS.

LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator.

LAMMPS is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It was developed at Sandia National Laboratories, a US Department of Energy facility, with funding from the DOE. It is an open-source code, distributed freely under the terms of the GNU Public License (GPL).

The primary developers of LAMMPS are Steve Plimpton, Aidan Thompson, and Paul Crozier who can be contacted at sjplimp,athomps,pscrozi at sandia.gov. The LAMMPS WWW Site at http://lammps.sandia.gov has more information about the code and its uses.

The LAMMPS documentation is organized into the following sections. If you find errors or omissions in this manual or have suggestions for useful information to add, please send an email to the developers so we can improve the LAMMPS documentation.

Once you are familiar with LAMMPS, you may want to bookmark this page at Section_commands.html#comm since it gives quick access to documentation for all LAMMPS commands.

PDF file of the entire manual, generated by htmldoc

LAMMPS Version info: 690

mass command

Syntax:

```
mass I value
```

- I = atom type (see asterisk form below)
- value = mass

Examples:

```
mass 1 1.0
mass * 62.5
mass 2* 62.5
```

Description:

Set the mass for all atoms of one or more atom types. Per-type mass values can also be set in the read_data data file using the "Masses" keyword. See the units command for what mass units to use.

The I index can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to set the mass for multiple atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive).

A line in a data file that follows the "Masses" keyword specifies mass using the same format as the arguments of the mass command in an input script, except that no wild-card asterisk can be used. For example, under the "Masses" section of a data file, the line that corresponds to the 1st example above would be listed as

```
1 1.0
```

Note that the mass command can only be used if the atom style requires per-type atom mass to be set. Currently, all but the *sphere* and *ellipsoid* and *peri* styles do. They require mass to be set for individual particles, not types. Per-atom masses are defined in the data file read by the read_data command, or set to default values by the create_atoms command. Per-atom masses can also be set to new values by the set mass or set density commands.

Also note that pair_style eam defines the masses of atom types in the EAM potential file, in which case the mass command is normally not used.

If you define a hybrid atom style which includes one (or more) sub-styles which require per-type mass and one (or more) sub-styles which require per-atom mass, then you must define both. However, in this case the per-type mass will be ignored; only the per-atom mass will be used by LAMMPS.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

All masses must be defined before a simulation is run. They must also all be defined before a velocity or fix shake command is used.

The mass assigned to any type or atom must be > 0.0.

Related commands: none

Default: none

minimize command

Syntax:

```
minimize etol ftol maxiter maxeval
```

- etol = stopping tolerance for energy (unitless)
- ftol = stopping tolerance for force (force units)
- maxiter = max iterations of minimizer
- maxeval = max number of force/energy evaluations

Examples:

```
minimize 1.0e-4 1.0e-6 100 1000
minimize 0.0 1.0e-8 1000 100000
```

Description:

Perform an energy minimization of the system, by iteratively adjusting atom coordinates. Iterations are terminated when one of the stopping criteria is satisfied. At that point the configuration will hopefully be in local potential energy minimum. More precisely, the configuration should approximate a critical point for the objective function (see below), which may or may not be a local minimum.

The minimization algorithm used is set by the [min_style](#) command. Other options are set by the [min_modify](#) command. Minimize commands can be interspersed with [run](#) commands to alternate between relaxation and dynamics. The minimizers bound the distance atoms move in one iteration, so that you can relax systems with highly overlapped atoms (large energies and forces) by pushing the atoms off of each other.

Alternate means of relaxing a system are to run dynamics with a small or [limited timestep](#). Or dynamics can be run using [fix viscous](#) to impose a damping force that slowly drains all kinetic energy from the system. The [pair_style soft](#) potential can be used to un-overlap atoms while running dynamics.

The [minimization styles](#) *cg*, *sd*, and *hftn* involves an outer iteration loop which sets the search direction along which atom coordinates are changed. An inner iteration is then performed using a line search algorithm. The line search typically evaluates forces and energies several times to set new coordinates. Currently, a backtracking algorithm is used which may not be optimal in terms of the number of force evaulations performed, but appears to be more robust than previous line searches we've tried. The backtracking method is described in Nocedal and Wright's Numerical Optimization (Procedure 3.1 on p 41).

The [minimization styles](#) *quickmin* and *fire* perform damped dynamics using an Euler integration step. Thus they require a [timestep](#) be defined, typically the same value used for [running dynamics](#) with the system, though it may be more efficient to use a larger timestep.

The objective function being minimized is the total potential energy of the system as a function of the N atom coordinates:

$$E(r_1, r_2, \ldots, r_N) = \sum_{i,j} E_{pair}(r_i, r_j) + \sum_{ij} E_{bond}(r_i, r_j) + \sum_{ijk} E_{angle}(r_i, r_j, r_k) +$$

$$\sum_{ijkl} E_{dihedral}(r_i, r_j, r_k, r_l) + \sum_{ijkl} E_{improper}(r_i, r_j, r_k, r_l) + \sum_{i} E$$

where the first term is the sum of all non-bonded pairwise interactions including long-range Coulombic interactions, the 2nd thru 5th terms are bond, angle, dihedral, and improper interactions respectively, and the last term is energy due to fixes which can act as constraints or apply force to atoms, such as thru interaction with a wall. See the discussion below about how fix commands affect minimization.

The starting point for the minimization is the current configuration of the atoms.

The minimization procedure stops if any of several criteria are met:

- the change in energy between outer iterations is less than *etol*
- the 2-norm (length) of the global force vector is less than the *ftol*
- the line search fails because the step distance backtracks to 0.0
- the number of outer iterations or timesteps exceeds *maxiter*
- the number of total force evaluations exceeds *maxeval*

For the first criterion, the specified energy tolerance *etol* is unitless; it is met when the energy change between successive iterations divided by the energy magnitude is less than or equal to the tolerance. For example, a setting of 1.0e-4 for *etol* means an energy tolerance of one part in 10^4. For the damped dynamics minimizers this check is not performed for a few steps after velocities are reset to 0, otherwise the minimizer would prematurely converge.

For the second criterion, the specified force tolerance *ftol* is in force units, since it is the length of the global force vector for all atoms, e.g. a vector of size 3N for N atoms. Since many of the components will be near zero after minimization, you can think of *ftol* as an upper bound on the final force on any component of any atom. For example, a setting of 1.0e-4 for *ftol* means no x, y, or z component of force on any atom will be larger than 1.0e-4 (in force units) after minimization.

Either or both of the *etol* and *ftol* values can be set to 0.0, in which case some other criterion will terminate the minimization.

During a minimization, the outer iteration count is treated as a timestep. Output is triggered by this timestep, e.g. thermodynamic output or dump and restart files.

Using the thermo_style custom command with the *fmax* or *fnorm* keywords can be useful for monitoring the progress of the minimization. Note that these outputs will be calculated only from forces on the atoms, and will not include any extra degrees of freedom, such as from the fix box/relax command.

Following minimization, a statistical summary is printed that lists which convergence criterion caused the minimizer to stop, as well as information about the energy, force, final line search, and iteration counts. An example is as follows:

```
Minimization stats:
  Stopping criterion = max iterations
  Energy initial, next-to-last, final =
      -0.626828169302     -2.82642039062     -2.82643549739
  Force two-norm initial, final = 2052.1 91.9642
  Force max component initial, final = 346.048 9.78056
```

```
Final line search alpha, max atom move = 2.23899e-06 2.18986e-05
Iterations, force evaluations = 2000 12724
```

The 3 energy values are for before and after the minimization and on the next-to-last iteration. This is what the *etol* parameter checks.

The two-norm force values are the length of the global force vector before and after minimization. This is what the *ftol* parameter checks.

The max-component force values are the absolute value of the largest component (x,y,z) in the global force vector, i.e. the infinity-norm of the force vector.

The alpha parameter for the line-search, when multiplied by the max force component (on the last iteration), gives the max distance any atom moved during the last iteration. Alpha will be 0.0 if the line search could not reduce the energy. Even if alpha is non-zero, if the "max atom move" distance is tiny compared to typical atom coordinates, then it is possible the last iteration effectively caused no atom movement and thus the evaluated energy did not change and the minimizer terminated. Said another way, even with non-zero forces, it's possible the effect of those forces is to move atoms a distance less than machine precision, so that the energy cannot be further reduced.

The iterations and force evaluation values are what is checked by the *maxiter* and *maxeval* parameters.

IMPORTANT NOTE: There are several force fields in LAMMPS which have discontinuities or other approximations which may prevent you from performing an energy minimization to high tolerances. For example, you should use a pair style that goes to 0.0 at the cutoff distance when performing minimization (even if you later change it when running dynamics). If you do not do this, the total energy of the system will have discontinuities when the relative distance between any pair of atoms changes from cutoff+epsilon to cutoff-epsilon and the minimizer may behave poorly. Some of the manybody potentials use splines and other internal cutoffs that inherently have this problem. The long-range Coulombic styles (PPPM, Ewald) are approximate to within the user-specified tolerance, which means their energy and forces may not agree to a higher precision than the Kspace-specified tolerance. In all these cases, the minimizer may give up and stop before finding a minimum to the specified energy or force tolerance.

Note that a cutoff Lennard-Jones potential (and others) can be shifted so that its energy is 0.0 at the cutoff via the pair_modify command. See the doc pages for inidividual pair styles for details. Note that Coulombic potentials always have a cutoff, unless versions with a long-range component are used (e.g. pair_style lj/cut/coul/long). The CHARMM potentials go to 0.0 at the cutoff (e.g. pair_style lj/charmm/coul/charmm), as do the GROMACS potentials (e.g. pair_style lj/gromacs).

If a soft potential (pair_style soft) is used the Astop value is used for the prefactor (no time dependence).

The fix box/relax command can be used to apply an external pressure to the simulation box and allow it to shrink/expand during the minimization.

Only a few other fixes (typically those that apply force constraints) are invoked during minimization. See the doc pages for individual fix commands to see which ones are relevant.

IMPORTANT NOTE: Some fixes which are invoked during minimization have an associated potential energy. For that energy to be included in the total potential energy of the system (the quantity being minimized), you MUST enable the fix_modify *energy* option for that fix. The doc pages for individual fix commands specify if this should be done.

Restrictions:

Features that are not yet implemented are listed here, in case someone knows how they could be coded:

minimize command

It is an error to use [fix shake](#) with minimization because it turns off bonds that should be included in the potential energy of the system. The effect of a fix shake can be approximated during a minimization by using stiff spring constants for the bonds and/or angles that would normally be constrained by the SHAKE algorithm.

[Fix rigid](#) is also not supported by minimization. It is not an error to have it defined, but the energy minimization will not keep the defined body(s) rigid during the minimization. Note that if bonds, angles, etc internal to a rigid body have been turned off (e.g. via [neigh_modify exclude](#)), they will not contribute to the potential energy which is probably not what is desired.

Pair potentials that produce torque on a particle (e.g. [granular potentials](#) or the [GayBerne potential](#) for ellipsoidal particles) are not relaxed by a minimization. More specifically, radial relaxations are induced, but no rotations are induced by a minimization, so such a system will not fully relax.

Related commands:

[min_modify](#), [min_style](#), [run_style](#)

Default: none

min_modify command

Syntax:

```
min_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
keyword = dmax or line
  dmax value = max
    max = maximum distance for line search to move (distance units)
  line value = backtrack or quadraticor forcezero
    backtrack,quadratic,forcezero = style of linesearch to use
```

Examples:

```
min_modify dmax 0.2
```

Description:

This command sets parameters that affect the energy minimization algorithms selected by the [min_style](#) command. The various settings may affect the convergence rate and overall number of force evaluations required by a minimization, so users can experiment with these parameters to tune their minimizations.

The *cg* and *sd* minimization styles have an outer iteration and an inner iteration which is steps along a one-dimensional line search in a particular search direction. The *dmax* parameter is how far any atom can move in a single line search in any dimension (x, y, or z). For the *quickmin* and *fire* minimization styles, the *dmax* setting is how far any atom can move in a single iteration (timestep). Thus a value of 0.1 in real [units](#) means no atom will move further than 0.1 Angstroms in a single outer iteration. This prevents highly overlapped atoms from being moved long distances (e.g. through another atom) due to large forces.

The choice of line search algorithm for the *cg* and *sd* minimization styles can be selected via the *line* keyword. The default backtracking search is robust and should always find a local energy minimum. However, it will "converge" when it can no longer reduce the energy of the system. Individual atom forces may still be larger than desired at this point, because the energy change is measured as the difference of two large values (energy before and energy after) and that difference may be smaller than machine epsilon even if atoms could move in the gradient direction to reduce forces further.

By contrast, the *quadratic* line search algorithm tries to reduce the forces to zero, while guaranteeing that the energy changes is not positive (uphill). For some systems, it may also be more efficient than the backtracking algorithm by requiring fewer energy/force evaluations. The *forcezero* line search algorithm is similar to *quadratic*. It may be more efficient than *quadratic* on some systems.

Restrictions: none

Related commands:

[min_style](#), [minimize](#)

Default:

The option defaults are dmax = 0.1 and line = backtrack.

min_style command

Syntax:

```
min_style style
```

- style = *cg* or *hftn* or *sd* or *quickmin* or *fire*

Examples:

```
min_style cg
min_style fire
```

Description:

Choose a minimization algorithm to use when a [minimize](#) command is performed.

Style *cg* is the Polak-Ribiere version of the conjugate gradient (CG) algorithm. At each iteration the force gradient is combined with the previous iteration information to compute a new search direction perpendicular (conjugate) to the previous search direction. The PR variant affects how the direction is chosen and how the CG method is restarted when it ceases to make progress. The PR variant is thought to be the most effective CG choice for most problems.

Style *hftn* is a Hessian-free truncated Newton algorithm. At each iteration a quadratic model of the energy potential is solved by a conjugate gradient inner iteration. The Hessian (second derivatives) of the energy is not formed directly, but approximated in each conjugate search direction by a finite difference directional derivative. When close to an energy minimum, the algorithm behaves like a Newton method and exhibits a quadratic convergence rate to high accuracy. In most cases the behavior of *hftn* is similar to *cg*, but it offers an alternative if *cg* seems to perform poorly. This style is not affected by the [min_modify](#) command.

Style *sd* is a steepest descent algorithm. At each iteration, the search direction is set to the downhill direction corresponding to the force vector (negative gradient of energy). Typically, steepest descent will not converge as quickly as CG, but may be more robust in some situations.

Style *quickmin* is a damped dynamics method described in [(Sheppard)](#), where the damping parameter is related to the projection of the velocity vector along the current force vector for each atom. The velocity of each atom is initialized to 0.0 by this style, at the beginning of a minimization.

Style *fire* is a damped dynamics method described in [(Bitzek)](#), which is similar to *quickmin* but adds a variable timestep and alters the projection operation to maintain components of the velocity non-parallel to the current force vector. The velocity of each atom is initialized to 0.0 by this style, at the beginning of a minimization.

Either the *quickmin* and *fire* styles are useful in the context of nudged elastic band (NEB) calculations via the [neb](#) command.

IMPORTANT NOTE: The *quickmin* and *fire* styles do not yet support the use of the [fix box/relax](#) command or minimizations involving the electron radius in [eFF](#) models.

Restrictions: none

Related commands:

min_modify, minimize, neb

Default:

```
min_style cg
```

(Sheppard) Sheppard, Terrell, Henkelman, J Chem Phys, 128, 134106 (2008). See ref 1 in this paper for original reference to Qmin in Jonsson, Mills, Jacobsen.

(Bitzek) Bitzek, Koskinen, Gahler, Moseler, Gumbsch, Phys Rev Lett, 97, 170201 (2006).

neb command

Syntax:

```
neb etol ftol N1 N2 Nevery file-style arg
```

- etol = stopping tolerance for energy (energy units)
- ftol = stopping tolerance for force (force units)
- N1 = max # of iterations (timesteps) to run initial NEB
- N2 = max # of iterations (timesteps) to run barrier-climbing NEB
- Nevery = print replica energies and reaction coordinates every this many timesteps
- file-style= *final* or *each* or *none*

```
  final arg = filename
      filename = file with initial coords for final replica
        coords for intermediate replicas are linearly interpolated between first and last rep
    each arg = filename
      filename = unique filename for each replica (except first) with its initial coords
    none arg = no argument
      all replicas assumed to already have their initial coords
```

Examples:

```
neb 0.1 0.0 1000 500 50 final coords.final
neb 0.0 0.001 1000 500 50 each coords.initial.$i
neb 0.0 0.001 1000 500 50 none
```

Description:

Perform a nudged elastic band (NEB) calculation using multiple replicas of a system. Two or more replicas must be used; the first and last are the end points of the transition path.

NEB is a method for finding both the atomic configurations and height of the energy barrier associated with a transition state, e.g. for an atom to perform a diffusive hop from one energy basin to another in a coordinated fashion with its neighbors. The implementation in LAMMPS follows the discussion in these 3 papers: [(Henkelman1)](#), [(Henkelman2)](#), and [(Nakano)](#).

Each replica runs on a partition of one or more processors. Processor partitions are defined at run-time using the -partition command-line switch; see [Section_start 7](#) of the manual. Note that if you have MPI installed, you can run a multi-replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10-replica simulation on just one or two processors. You will simply not get the performance speed-up you would see with one or more physical processors per replica. See [this section](#) of the manual for further discussion.

IMPORTANT NOTE: The current NEB implementation in LAMMPS only allows there to be one processor per replica.

IMPORTANT NOTE: As explained below, a NEB calculation perfoms a damped-dynamics minimization across all the replicas. This will use whatever timestep you have defined in your input script, via the [timestep](#) command. You may get faster convergence for a NEB calculation if you use a larger timestep than you would normally use for dynamics with the same system.

When a NEB calculation is performed, it is assumed that each replica is running the same system, though LAMMPS does not check for this. I.e. the simulation domain, the number of atoms, the interaction potentials,

and the starting configuration when the neb command is issued should be the same for every replica.

In a NEB calculation each atom in a replica is connected to the same atom in adjacent replicas by springs, which induce inter-replica forces. These forces are imposed by the fix neb command, which must be used in conjunction with the neb command. The group used to define the fix neb command defines the NEB atoms which are the only ones that inter-replica springs are applied to. If the group does not include all atoms, then non-NEB atoms have no inter-replica springs and the forces they feel and their motion is computed in the usual way due only to other atoms within their replica. Conceptually, the non-NEB atoms provide a background force field for the NEB atoms. They can be allowed to move during the NEB minimiation procedure (which will typically induce different coordinates for non-NEB atoms in different replicas), or held fixed using other LAMMPS commands such as fix setforce. Note that the partition command can be used to invoke a command on a subset of the replicas, e.g. if you wish to hold NEB or non-NEB atoms fixed in only the end-point replicas.

The initial atomic configuration for each of the replicas can be specified in different manners via the *file-style* setting, as discussed below. Only atoms whose initial coordinates should differ from the current configuration need be specified.

Conceptually, the initial configuration for the first replica should be a state with all the atoms (NEB and non-NEB) having coordinates on one side of the energy barrier. A perfect energy minimum is not required, since atoms in the first replica experience no spring forces from the 2nd replica. Thus the damped dynamics minimizaiton will drive the first replica to an energy minimum if it is not already there. However, you will typically get better convergence if the initial state is already at a minimum. For example, for a system with a free surface, the surface should be fully relaxed before attempting a NEB calculation.

Likewise, the initial configuration of the final replica should be a state with all the atoms (NEB and non-NEB) on the other side of the energy barrier. Again, a perfect energy minimum is not required, since the atoms in the last replica also experience no spring forces from the next-to-last replica, and thus the damped dynamics minimization will drive it to an energy minimum.

As explained below, the initial configurations of intermediate replicas can be atomic coordinates interpolated in a linear fashion between the first and last replicas. This is often adequate state for simple transitions. For more complex transitions, it may lead to slow convergence or even bad results if the minimum energy path (MEP, see below) of states over the barrier cannot be correctly converged to from such an initial configuration. In this case, you will want to generate initial states for the intermediate replicas that are geometrically closer to the MEP and read them in.

For a *file-style* setting of *final*, a filename is specified which contains atomic coordinates for zero or more atoms, in the format described below. For each atom that appears in the file, the new coordinates are assigned to that atom in the final replica. Each intermediate replica also assigns a new position to that atom in an interpolated manner. This is done by using the current position of the atom as the starting point and the read-in position as the final point. The distance between them is calculated, and the new position is assigned to be a fraction of the distance. E.g. if there are 10 replicas, the 2nd replica will assign a position that is 10% of the distance along a line between the starting and final point, and the 9th replica will assign a position that is 90% of the distance along the line. Note that this procedure to produce consistent coordinates across all the replicas, the current coordinates need to be the same in all replicas. LAMMPS does not check for this, but invalid initial configurations will likely result if it is not the case.

NOTE: The "distance" between the starting and final point is calculated in a minimum-image sense for a periodic simulation box. This means that if the two positions are on opposite sides of a box (periodic in that dimension), the distance between them will be small, because the periodic image of one of the atoms is close to the other. Similarly, even if the assigned position resulting from the interpolation is outside the periodic box, the atom will be wrapped back into the box when the NEB calculation begins.

For a *file-style* setting of *each*, a filename is specified which is assumed to be unique to each replica. This can be done by using a variable in the filename, e.g.

```
variable i equal part
neb 0.0 0.001 1000 500 50 each coords.initial.$i
```

which in this case will substitute the partition ID (0 to N-1) for the variable I, which is also effectively the replica ID. See the variable command for other options, such as using world-, universe-, or uloop-style variables.

Each replica (except the first replica) will read its file, formatted as described below, and for any atom that appears in the file, assign the specified coordinates to its atom. The various files do not need to contain the same set of atoms.

For a *file-style* setting of *none*, no filename is specified. Each replica is assumed to already be in its initial configuration at the time the neb command is issued. This allows each replica to define its own configuration by reading a replica-specific data or restart or dump file, via the read_data, read_restart, or read_dump commands. The replica-specific names of these files can be specified as in the discussion above for the *each* file-style. Also see the section below for how a NEB calculation can produce restart files, so that a long calculation can be restarted if needed.

IMPORTANT NOTE: None of the *file-style* settings change the initial configuration of any atom in the first replica. The first replica must thus be in the correct initial configuration at the time the neb command is issued.

A NEB calculation proceeds in two stages, each of which is a minimization procedure, performed via damped dynamics. To enable this, you must first define a damped dynamics min_style, such as *quickmin* or *fire*. The *cg*, *sd*, and *hftn* styles cannot be used, since they perform iterative line searches in their inner loop, which cannot be easily synchronized across multiple replicas.

The minimizer tolerances for energy and force are set by *etol* and *ftol*, the same as for the minimize command.

A non-zero *etol* means that the NEB calculation will terminate if the energy criterion is met by every replica. The energies being compared to *etol* do not include any contribution from the inter-replica forces, since these are non-conservative. A non-zero *ftol* means that the NEB calculation will terminate if the force criterion is met by every replica. The forces being compared to *ftol* include the inter-replica forces between an atom and its images in adjacent replicas.

The maximum number of iterations in each stage is set by *N1* and *N2*. These are effectively timestep counts since each iteration of damped dynamics is like a single timestep in a dynamics run. During both stages, the potential energy of each replica and its normalized distance along the reaction path (reaction coordinate RD) will be printed to the screen and log file every *Nevery* timesteps. The RD is 0 and 1 for the first and last replica. For intermediate replicas, it is the cumulative distance (normalized by the total cumulative distance) between adjacent replicas, where "distance" is defined as the length of the 3N-vector of differences in atomic coordinates, where N is the number of NEB atoms involved in the transition. These outputs allow you to monitor NEB's progress in finding a good energy barrier. *N1* and *N2* must both be multiples of *Nevery*.

In the first stage of NEB, the set of replicas should converge toward the minimum energy path (MEP) of conformational states that transition over the barrier. The MEP for a barrier is defined as a sequence of 3N-dimensional states that cross the barrier at its saddle point, each of which has a potential energy gradient parallel to the MEP itself. The replica states will also be roughly equally spaced along the MEP due to the inter-replica spring force added by the fix neb command.

In the second stage of NEB, the replica with the highest energy is selected and the inter-replica forces on it are converted to a force that drives its atom coordinates to the top or saddle point of the barrier, via the

barrier-climbing calculation described in (Henkelman2). As before, the other replicas rearrange themselves along the MEP so as to be roughly equally spaced.

When both stages are complete, if the NEB calculation was successful, one of the replicas should be an atomic configuration at the top or saddle point of the barrier, the potential energies for the set of replicas should represent the energy profile of the barrier along the MEP, and the configurations of the replicas should be a sequence of configurations along the MEP.

A few other settings in your input script are required or advised to perform a NEB calculation. See the IMPORTANT NOTE about the choice of timestep at the beginning of this doc page.

An atom map must be defined which it is not by default for atom_style atomic problems. The atom_modify map command can be used to do this.

The "atom_modify sort 0 0.0" command should be used to turn off atom sorting.

NOTE: This sorting restriction will be removed in a future version of NEB in LAMMPS.

The minimizers in LAMMPS operate on all atoms in your system, even non-NEB atoms, as defined above. To prevent non-NEB atoms from moving during the minimization, you should use the fix setforce command to set the force on each of those atoms to 0.0. This is not required, and may not even be desired in some cases, but if those atoms move too far (e.g. because the initial state of your system was not well-minimized), it can cause problems for the NEB procedure.

The damped dynamics minimizers, such as *quickmin* and *fire*), adjust the position and velocity of the atoms via an Euler integration step. Thus you must define an appropriate timestep to use with NEB. Using the same timestep that would be used for a dynamics run of your system is advised.

Each file read by the neb command containing atomic coordinates used to initialize one or more replicas must be formatted as follows.

The file can be ASCII text or a gzipped text file (detected by a .gz suffix). The file can contain initial blank lines or comment lines starting with "#" which are ignored. The first non-blank, non-comment line should list N = the number of lines to follow. The N successive lines contain the following information:

```
ID1 x1 y1 z1
ID2 x2 y2 z2
...
IDN xN yN zN
```

The fields are the the atom ID, followed by the x,y,z coordinates. The lines can be listed in any order. Additional trailing information on the line is OK, such as a comment.

Note that for a typical NEB calculation you do not need to specify initial coordinates for very many atoms to produce differing starting and final replicas whose intermediate replicas will converge to the energy barrier. Typically only new coordinates for atoms geometrically near the barrier need be specified.

Also note there is no requirement that the atoms in the file correspond to the NEB atoms in the group defined by the fix neb command. Not every NEB atom need be in the file, and non-NEB atoms can be listed in the file.

Four kinds of output can be generated during a NEB calculation: energy barrier statistics, thermodynamic output by each replica, dump files, and restart files.

neb command   705

When running with multiple partitions (each of which is a replica in this case), the print-out to the screen and master log.lammps file contains a line of output, printed once every *Nevery* timesteps. It contains the timestep, the maximum force per replica, the maximum force per atom (in any replica), potential gradients in the initial, final, and climbing replicas, the forward and backward energy barriers, the total reaction coordinate (RDT), and the normalized reaction coordinate and potential energy of each replica.

The "maximum force per replica" is the two-norm of the 3N-length force vector for the atoms in each replica, maximized across replicas, which is what the *ftol* setting is checking against. In this case, N is all the atoms in each replica. The "maximum force per atom" is the maximum force component of any atom in any replica. The potential gradients are the two-norm of the 3N-length force vector solely due to the interaction potential i.e. without adding in inter-replica forces. Note that inter-replica forces are zero in the initial and final replicas, and only affect the direction in the climbing replica. For this reason, the "maximum force per replica" is often equal to the potential gradient in the climbing replica. In the first stage of NEB, there is no climbing replica, and so the potential gradient in the highest energy replica is reported, since this replica will become the climbing replica in the second stage of NEB.

The "reaction coordinate" (RD) for each replica is the two-norm of the 3N-length vector of distances between its atoms and the preceding replica's atoms, added to the RD of the preceding replica. The RD of the first replica RD1 = 0.0; the RD of the final replica RDN = RDT, the total reaction coordinate. The normalized RDs are divided by RDT, so that they form a monotonically increasing sequence from zero to one. When computing RD, N only includes the atoms being operated on by the fix neb command.

The forward (reverse) energy barrier is the potential energy of the highest replica minus the energy of the first (last) replica.

When running on multiple partitions, LAMMPS produces additional log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For a NEB calculation, these contain the thermodynamic output for each replica.

If [dump](#) commands in the input script define a filename that includes a *universe* or *uloop* style [variable](#), then one dump file (per dump command) will be created for each replica. At the end of the NEB calculation, the final snapshot in each file will contain the sequence of snapshots that transition the system over the energy barrier. Earlier snapshots will show the convergence of the replicas to the MEP.

Likewise, [restart](#) filenames can be specified with a *universe* or *uloop* style [variable](#), to generate restart files for each replica. These may be useful if the NEB calculation fails to converge properly to the MEP, and you wish to restart the calculation from an intermediate point with altered parameters.

There are 2 Python scripts provided in the tools/python directory, neb_combine.py and neb_final.py, which are useful in analyzing output from a NEB calculation. Assume a NEB simulation with M replicas, and the NEB atoms labelled with a specific atom type.

The neb_combine.py script extracts atom coords for the NEB atoms from all M dump files and creates a single dump file where each snapshot contains the NEB atoms from all the replicas and one copy of non-NEB atoms from the first replica (presumed to be identical in other replicas). This can be visualized/animated to see how the NEB atoms relax as the NEB calculation proceeds.

The neb_final.py script extracts the final snapshot from each of the M dump files to create a single dump file with M snapshots. This can be visualized to watch the system make its transition over the energy barrier.

To illustrate, here are images from the final snapshot produced by the neb_combine.py script run on the dump files produced by the two example input scripts in examples/neb. Click on them to see a larger image.
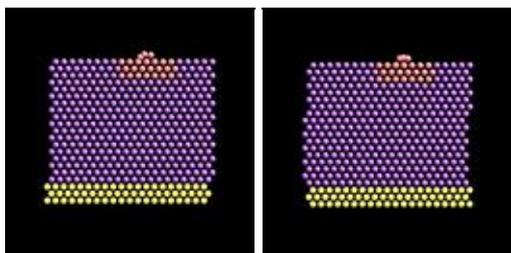
Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the Making LAMMPS section for more info on packages.

Related commands:

prd, temper, fix langevin, fix viscous

Default: none

(Henkelman1) Henkelman and Jonsson, J Chem Phys, 113, 9978-9985 (2000).

(Henkelman2) Henkelman, Uberuaga, Jonsson, J Chem Phys, 113, 9901-9904 (2000).

(Nakano) Nakano, Comp Phys Comm, 178, 280-289 (2008).

neighbor command

Syntax:

```
neighbor skin style
```

- skin = extra distance beyond force cutoff (distance units)
- style = *bin* or *nsq* or *multi*

Examples:

```
neighbor 0.3 bin
neighbor 2.0 nsq
```

Description:

This command sets parameters that affect the building of pairwise neighbor lists. All atom pairs within a neighbor cutoff distance equal to the their force cutoff plus the *skin* distance are stored in the list. Typically, the larger the skin distance, the less often neighbor lists need to be built, but more pairs must be checked for possible force interactions every timestep. The default value for *skin* depends on the choice of units for the simulation; see the default values below.

The *skin* distance is also used to determine how often atoms migrate to new processors if the *check* option of the neigh_modify command is set to *yes*. Atoms are migrated (communicated) to new processors on the same timestep that neighbor lists are re-built.

The *style* value selects what algorithm is used to build the list. The *bin* style creates the list by binning which is an operation that scales linearly with N/P, the number of atoms per processor where N = total number of atoms and P = number of processors. It is almost always faster than the *nsq* style which scales as (N/P)^2. For unsolvated small molecules in a non-periodic box, the *nsq* choice can sometimes be faster. Either style should give the same answers.

The *multi* style is a modified binning algorithm that is useful for systems with a wide range of cutoff distances, e.g. due to different size particles. For the *bin* style, the bin size is set to 1/2 of the largest cutoff distance between any pair of atom types and a single set of bins is defined to search over for all atom types. This can be inefficient if one pair of types has a very long cutoff, but other type pairs have a much shorter cutoff. For style *multi* the bin size is set to 1/2 of the shortest cutoff distance and multiple sets of bins are defined to search over for different atom types. This imposes some extra setup overhead, but the searches themselves may be much faster for the short-cutoff cases. See the communicate multi command for a communication option option that may also be beneficial for simulations of this kind.

The neigh_modify command has additional options that control how often neighbor lists are built and which pairs are stored in the list.

When a run is finished, counts of the number of neighbors stored in the pairwise list and the number of times neighbor lists were built are printed to the screen and log file. See this section for details.

Restrictions: none

Related commands:

neigh_modify, units, communicate

Default:

0.3 bin for units = lj, skin = 0.3 sigma
2.0 bin for units = real or metal, skin = 2.0 Angstroms
0.001 bin for units = si, skin = 0.001 meters = 1.0 mm
0.1 bin for units = cgs, skin = 0.1 cm = 1.0 mm

neigh_modify command

Syntax:

```
neigh_modify keyword values ...
```

- one or more keyword/value pairs may be listed

```
  keyword = delay or every or check or once or cluster or include or exclude or page or one
    delay value = N
      N = delay building until this many steps since last build
    every value = M
      M = build neighbor list every this many steps
    check value = yes or no
      yes = only build if some atom has moved half the skin distance or more
      no = always build on 1st step that every and delay are satisfied
    once
      yes = only build neighbor list once at start of run and never rebuild
      no = rebuild neighbor list according to other settings
    cluster
      yes = check bond,angle,etc neighbor list for nearby clusters
      no = do not check bond,angle,etc neighbor list for nearby clusters
    include value = group-ID
      group-ID = only build pair neighbor lists for atoms in this group
    exclude values:
      type M N
        M,N = exclude if one atom in pair is type M, other is type N
      group group1-ID group2-ID
        group1-ID,group2-ID = exclude if one atom is in 1st group, other in 2nd
      molecule group-ID
        groupname = exclude if both atoms are in the same molecule and in the same group
      none
        delete all exclude settings
    page value = N
      N = number of pairs stored in a single neighbor page
    one value = N
      N = max number of neighbors of one atom
    contact_distance_factor value = N
      N = contact distance factor used to extend the range of granular neighbor lists (must
    binsize value = size
      size = bin size for neighbor list construction (distance units)
```

Examples:

```
neigh_modify every 2 delay 10 check yes page 100000
neigh_modify exclude type 2 3
neigh_modify exclude group frozen frozen check no
neigh_modify exclude group residue1 chain3
neigh_modify exclude molecule rigid
neigh_modify delay 0 contact_distance_factor 1.5
```

Description:

This command sets parameters that affect the building and use of pairwise neighbor lists.

The *every*, *delay*, *check*, and *once* options affect how often lists are built as a simulation runs. The *delay* setting means never build a new list until at least N steps after the previous build. The *every* setting means build the list every M steps (after the delay has passed). If the *check* setting is *no*, the list is built on the 1st step that satisfies the *delay* and *every* settings. If the *check* setting is *yes*, then the list is only built on a

particular step if some atom has moved more than half the skin distance (specified in the [neighbor](#) command) since the last build. If the *once* setting is yes, then the neighbor list is only built once at the beginning of each run, and never rebuilt. This should only be done if you are certain atoms will not move far enough that the list should be rebuilt. E.g. running a simulation of a cold crystal. Note that it is not that expensive to check if neighbor lists should be rebuilt.

When the rRESPA integrator is used (see the [run_style](#) command), the *every* and *delay* parameters refer to the longest (outermost) timestep.

The *contact_distance_factor* setting can be used to increase the range of granular neighbor lists. When *contact_distance_factor* > 1.0, instead of the standard criterion ri+rj+skin < distance, LIGGGHTS is checking for *contact_distance_factor* *(ri+rj)+skin < distance to decided if a pair of granular particles goes into a neighbor list.

The *cluster* option does a sanity test every time neighbor lists are built for bond, angle, dihedral, and improper interactions, to check that each set of 2, 3, or 4 atoms is a cluster of nearby atoms. It does this by computing the distance between pairs of atoms in the interaction and insuring they are not further apart than half the periodic box length. If they are, an error is generated, since the interaction would be computed between far-away atoms instead of their nearby periodic images. The only way this should happen is if the pairwise cutoff is so short that atoms that are part of the same interaction are not communicated as ghost atoms. This is an unusual model (e.g. no pair interactions at all) and the problem can be fixed by use of the [communicate cutoff](#) command. Note that to save time, the default *cluster* setting is *no*, so that this check is not performed.

The *include* option limits the building of pairwise neighbor lists to atoms in the specified group. This can be useful for models where a large portion of the simulation is particles that do not interact with other particles or with each other via pairwise interactions. The group specified with this option must also be specified via the [atom_modify first](#) command.

The *exclude* option turns off pairwise interactions between certain pairs of atoms, by not including them in the neighbor list. These are sample scenarios where this is useful:

- In crack simulations, pairwise interactions can be shut off between 2 slabs of atoms to effectively create a crack.
- When a large collection of atoms is treated as frozen, interactions between those atoms can be turned off to save needless computation. E.g. Using the [fix setforce](#) command to freeze a wall or portion of a bio-molecule.
- When one or more rigid bodies are specified, interactions within each body can be turned off to save needless computation. See the [fix rigid](#) command for more details.

The *exclude type* option turns off the pairwise interaction if one atom is of type M and the other of type N. M can equal N. The *exclude group* option turns off the interaction if one atom is in the first group and the other is the second. Group1-ID can equal group2-ID. The *exclude molecule* option turns off the interaction if both atoms are in the specified group and in the same molecule, as determined by their molecule ID.

Each of the exclude options can be specified multiple times. The *exclude type* option is the most efficient option to use; it requires only a single check, no matter how many times it has been specified. The other exclude options are more expensive if specified multiple times; they require one check for each time they have been specified.

Note that the exclude options only affect pairwise interactions; see the [delete_bonds](#) command for information on turning off bond interactions.

IMPORTANT NOTE: Excluding pairwise interactions will not work correctly when also using a long-range solver via the [kspace_style](#) command. LAMMPS will give a warning to this effect. This is because the

short-range pairwise interaction needs to subtract off a term from the total energy for pairs whose short-range interaction is excluded, to compensate for how the long-range solver treats the interaction. This is done correctly for pairwise interactions that are excluded (or weighted) via the special_bonds command. But it is not done for interactions that are excluded via these neigh_modify exclude options.

The *page* and *one* options affect how memory is allocated for the neighbor lists. For most simulations the default settings for these options are fine, but if a very large problem is being run or a very long cutoff is being used, these parameters can be tuned. The indices of neighboring atoms are stored in "pages", which are allocated one after another as they fill up. The size of each page is set by the *page* value. A new page is allocated when the next atom's neighbors could potentially overflow the list. This threshold is set by the *one* value which tells LAMMPS the maximum number of neighbor's one atom can have.

IMPORTANT NOTE: LAMMPS can crash without an error message if the number of neighbors for a single particle is larger than the *page* setting, which means it is much, much larger than the *one* setting. This is because LAMMPS doesn't error check these limits for every pairwise interaction (too costly), but only after all the particle's neighbors have been found. This problem usually means something is very wrong with the way you've setup your problem (particle spacing, cutoff length, neighbor skin distance, etc). If you really expect that many neighbors per particle, then boost the *one* and *page* settings accordingly.

The *binsize* option allows you to specify what size of bins will be used in neighbor list construction to sort and find neighboring atoms. By default, for neighbor style bin, LAMMPS uses bins that are 1/2 the size of the maximum pair cutoff. For neighbor style multi, the bins are 1/2 the size of the minimum pair cutoff. Typically these are good values values for minimizing the time for neighbor list construction. This setting overrides the default. If you make it too big, there is little overhead due to looping over bins, but more atoms are checked. If you make it too small, the optimal number of atoms is checked, but bin overhead goes up. If you set the binsize to 0.0, LAMMPS will use the default binsize of 1/2 the cutoff.

Restrictions:

If the "delay" setting is non-zero, then it must be a multiple of the "every" setting.

The exclude molecule option can only be used with atom styles that define molecule IDs.

The value of the *page* setting must be at least 10x larger than the *one* setting. This insures neighbor pages are not mostly empty space.

Related commands:

neighbor, delete_bonds

Default:

The option defaults are delay = 10, every = 1, check = yes, once = no, cluster = no, include = all, exclude = none, page = 100000, one = 2000, and binsize = 0.0.

newton command

Syntax:

```
newton flag
newton flag1 flag2
```

- flag = *on* or *off* for both pairwise and bonded interactions
- flag1 = *on* or *off* for pairwise interactions
- flag2 = *on* or *off* for bonded interactions

Examples:

```
newton off
newton on off
```

Description:

This command turns Newton's 3rd law *on* or *off* for pairwise and bonded interactions. For most problems, setting Newton's 3rd law to *on* means a modest savings in computation at the cost of two times more communication. Whether this is faster depends on problem size, force cutoff lengths, a machine's compute/communication ratio, and how many processors are being used.

Setting the pairwise newton flag to *off* means that if two interacting atoms are on different processors, both processors compute their interaction and the resulting force information is not communicated. Similarly, for bonded interactions, newton *off* means that if a bond, angle, dihedral, or improper interaction contains atoms on 2 or more processors, the interaction is computed by each processor.

LAMMPS should produce the same answers for any newton flag settings, except for round-off issues.

With run_style *respa* and only bonded interactions (bond, angle, etc) computed in the innermost timestep, it may be faster to turn newton *off* for bonded interactions, to avoid extra communication in the innermost loop.

Restrictions:

The newton bond setting cannot be changed after the simulation box is defined by a read_data or create_box command.

Related commands:

run_style respa

Default:

```
newton on
```

next command

Syntax:

```
next variables
```

> • variables = one or more variable names

Examples:

```
next x
next a t x myTemp
```

Description:

This command is used with variables defined by the [variable](#) command. It assigns the next value to the variable from the list of values defined for that variable by the [variable](#) command. Thus when that variable is subsequently substituted for in an input script command, the new value is used.

See the [variable](#) command for info on how to define and use different kinds of variables in LAMMPS input scripts. If a variable name is a single lower-case character from "a" to "z", it can be used in an input script command as $a or $z. If it is multiple letters, it can be used as ${myTemp}.

If multiple variables are used as arguments to the *next* command, then all must be of the same variable style: *index*, *loop*, *file*, *universe*, or *uloop*. An exception is that *universe-* and *uloop*-style variables can be mixed in the same *next* command.

All the variables specified with the next command are incremented by one value from their respective list of values. A *file*-style variable reads the next line from its associated file. An *atomfile*-style variable reads the next set of lines (one per atom) from its associated file. *String-* or *atom-* or *equal-* or *world*-style variables cannot be used with the the next command, since they only store a single value.

When any of the variables in the next command has no more values, a flag is set that causes the input script to skip the next [jump](#) command encountered. This enables a loop containing a next command to exit. As explained in the [variable](#) command, the variable that has exhausted its values is also deleted. This allows it to be used and re-defined later in the input script. *File*-style and *atomfile*-style variables are exhausted when the end-of-file is reached.

When the next command is used with *index-* or *loop*-style variables, the next value is assigned to the variable for all processors. When the next command is used with *file*-style variables, the next line is read from its file and the string assigned to the variable. When the next command is used with *atomfile*-style variables, the next set of per-atom values is read from its file and assigned to the variable. When the next command is used with *universe-* or *uloop*-style variables, the next value is assigned to whichever processor partition executes the command first. All processors in the partition are assigned the same value. Running LAMMPS on multiple partitions of processors via the "-partition" command-line switch is described in [this section](#) of the manual. *Universe-* and *uloop*-style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

Here is an example of running a series of simulations using the next command with an *index*-style variable. If this input script is named in.polymer, 8 simulations would be run using data files from directories run1 thru run8.

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
```

```
shell cd $d
read_data data.polymer
run 10000
shell cd ..
clear
next d
jump in.polymer
```

If the variable "d" were of style *universe*, and the same in.polymer input script were run on 3 partitions of processors, then the first 3 simulations would begin, one on each set of processors. Whichever partition finished first, it would assign variable "d" the 4th value and run another simulation, and so forth until all 8 simulations were finished.

Jump and next commands can also be nested to enable multi-level loops. For example, this script will run 15 simulations in a double loop.

```
variable i loop 3
  variable j loop 5
  clear
  ...
  read_data data.polymer.$i$j
  print Running simulation $i.$j
  run 10000
  next j
  jump in.script
next i
jump in.script
```

Here is an example of a double loop which uses the [if](#) and [jump](#) commands to break out of the inner loop when a condition is met, then continues iterating thru the outer loop.

```
label      loopa
variable   a loop 5
  label      loopb
  variable   b loop 5
  print      "A,B = $a,$b"
  run        10000
  if         $b > 2 then "jump in.script break"
  next       b
  jump       in.script loopb
label      break
variable   b delete

next       a
jump       in.script loopa
```

Restrictions: none

Related commands:

[jump](#), [include](#), [shell](#), [variable](#),

Default: none

next command   715

package command

Syntax:

```
package style args
```

- style = *gpu* or *cuda* or *omp*
- args = arguments specific to the style

```
  gpu args = mode first last split keyword value ...
      mode = force or force/neigh
      first = ID of first GPU to be used on each node
      last = ID of last GPU to be used on each node
      split = fraction of particles assigned to the GPU
      zero or more keyword/value pairs may be appended
      keywords = threads_per_atom or cellsize or device
        threads_per_atom value = Nthreads
          Nthreads = # of GPU threads used per atom
        cellsize value = dist
          dist = length (distance units) in each dimension for neighbor bins
        device value = device_type
          device_type = kepler or fermi or cypress or generic
  cuda args = keyword value ...
      one or more keyword/value pairs may be appended
      keywords = gpu/node or gpu/node/special or timing or test or override/bpa
        gpu/node value = N
          N = number of GPUs to be used per node
        gpu/node/special values = N gpu1 .. gpuN
          N = number of GPUs to be used per node
          gpu1 .. gpuN = N IDs of the GPUs to use
        timing values = none
        test values = id
          id = atom-ID of a test particle
        override/bpa values = flag
          flag = 0 for TpA algorithm, 1 for BpA algorithm
  omp args = Nthreads mode
      Nthreads = # of OpenMP threads to associate with each MPI process
      mode = force or force/neigh (optional)
```

Examples:

```
package gpu force 0 0 1.0
package gpu force 0 0 0.75
package gpu force/neigh 0 0 1.0
package gpu force/neigh 0 1 -1.0
package cuda gpu/node/special 2 0 2
package cuda test 3948
package omp * force/neigh
package omp 4 force
```

Description:

This command invokes package-specific settings. Currently the following packages use it: GPU, USER-CUDA, and USER-OMP.

To use the accelerated GPU and USER-OMP styles, the use of the package command is required. However, as described in the "Defaults" section below, if you use the "-sf gpu" or "-sf omp" [command-line options](#) to enable use of these styles, then default package settings are enabled. In that case you only need to use the package command if you want to change the defaults.

To use the accelerate USER-CUDA styles, the package command is not required as defaults are assigned internally. You only need to use the package command if you want to change the defaults.

See Section_accelerate of the manual for more details about using these various packages for accelerating LAMMPS calculations.

The *gpu* style invokes options associated with the use of the GPU package.

The *mode* setting specifies where neighbor list calculations will be performed. If *mode* is force, neighbor list calculation is performed on the CPU. If *mode* is force/neigh, neighbor list calculation is performed on the GPU. GPU neighbor list calculation currently cannot be used with a triclinic box. GPU neighbor list calculation currently cannot be used with hybrid pair styles. GPU neighbor lists are not compatible with styles that are not GPU-enabled. When a non-GPU enabled style requires a neighbor list, it will also be built using CPU routines. In these cases, it will typically be more efficient to only use CPU neighbor list builds.

The *first* and *last* settings specify the GPUs that will be used for simulation. On each node, the GPU IDs in the inclusive range from *first* to *last* will be used.

The *split* setting can be used for load balancing force calculation work between CPU and GPU cores in GPU-enabled pair styles. If $0 < split < 1.0$, a fixed fraction of particles is offloaded to the GPU while force calculation for the other particles occurs simultaneously on the CPU. If *split*split = 1.0, all force calculations for GPU accelerated pair styles are performed on the GPU. In this case, hybrid, bond, angle, dihedral, improper, and long-range calculations can be performed on the CPU while the GPU is performing force calculations for the GPU-enabled pair style. If all CPU force computations complete before the GPU, LAMMPS will block until the GPU has finished before continuing the timestep.

As an example, if you have two GPUs per node and 8 CPU cores per node, and would like to run on 4 nodes (32 cores) with dynamic balancing of force calculation across CPU and GPU cores, you could specify

```
package gpu force/neigh 0 1 -1
```

In this case, all CPU cores and GPU devices on the nodes would be utilized. Each GPU device would be shared by 4 CPU cores. The CPU cores would perform force calculations for some fraction of the particles at the same time the GPUs performed force calculation for the other particles.

The *threads_per_atom* keyword allows control of the number of GPU threads used per-atom to perform the short range force calculation. By default, the value will be chosen based on the pair style, however, the value can be set with this keyword to fine-tune performance. For large cutoffs or with a small number of particles per GPU, increasing the value can improve performance. The number of threads per atom must be a power of 2 and currently cannot be greater than 32.

The *cellsize* keyword can be used to control the size of the cells used for binning atoms in neighbor list calculations. Setting this value is normally not needed; the optimal value is close to the default (equal to the cutoff distance for the short range interactions plus the neighbor skin). GPUs can perform efficiently with much larger cutoffs than CPUs and this can be used to reduce the time required for long-range calculations or in some cases to eliminate them with models such as coul/wolf or coul/dsf. For very large cutoffs, it can be more efficient to use smaller values for cellsize in parallel simulations. For example, with a cutoff of 20*sigma and a neighbor skin of sigma, a cellsize of 5.25*sigma can be efficient for parallel simulations.

The *device* keyword can be used to tune parameters to optimize for a specific accelerator when using OpenCL. For CUDA, the *device* keyword is ignored. Currently, the device type is limited to NVIDIA Kepler, NVIDIA Fermi, AMD Cypress, or a generic device. More devices will be added soon. The default device type can be specified when building LAMMPS with the GPU library.

---

package command   717

The *cuda* style invokes options associated with the use of the USER-CUDA package.

The *gpu/node* keyword specifies the number *N* of GPUs to be used on each node. An MPI process with rank *K* will use the GPU (K mod N). This implies that processes should be assigned with successive ranks on each node, which is the default with most (or even all) MPI implementations. The default value for *N* is 2.

The *gpu/node/special* keyword also specifies the number (N) of GPUs to be used on each node, but allows more control over their specification. An MPI process with rank *K* will use the GPU *gpuI* with l = (K mod N) + 1. This implies that processes should be assigned with successive ranks on each node, which is the default with most (or even all) MPI implementations. For example if you have three GPUs on a machine, one of which is used for the X-Server (the GPU with the ID 1) while the others (with IDs 0 and 2) are used for computations you would specify:

```
package cuda gpu/node/special 2 0 2
```

A main purpose of the *gpu/node/special* optoin is to allow two (or more) simulations to be run on one workstation. In that case one would set the first simulation to use GPU 0 and the second to use GPU 1. This is not necessary though, if the GPUs are in what is called *compute exclusive* mode. Using that setting, every process will get its own GPU automatically. This *compute exclusive* mode can be set as root using the *nvidia-smi* tool which is part of the CUDA installation.

Note that if the *gpu/node/special* keyword is not used, the USER-CUDA package sorts existing GPUs on each node according to their number of multiprocessors. This way, compute GPUs will be priorized over X-Server GPUs.

Use of the *timing* keyword will output detailed timing information for various subroutines.

The *test* keyword will output info for the the specified atom at several points during each time step. This is mainly usefull for debugging purposes. Note that the simulation will be severly slowed down if this option is used.

The *override/bpa* keyword can be used to specify which mode is used for pair-force evaluation. TpA = one thread per atom; BpA = one block per atom. If this keyword is not used, a short test at the begin of each run will determine which method is more effective (the result of this test is part of the LAMMPS output). Therefore it is usually not necessary to use this keyword.

The *omp* style invokes options associated with the use of the USER-OMP package.

The first argument allows to explicitly set the number of OpenMP threads to be allocated for each MPI process. For example, if your system has nodes with dual quad-core processors, it has a total of 8 cores per node. You could run MPI on 2 cores on each node (e.g. using options for the mpirun command), and set the *Nthreads* setting to 4. This would effectively use all 8 cores on each node. Since each MPI process would spawn 4 threads (one of which runs as part of the MPI process itself).

For performance reasons, you should not set *Nthreads* to more threads than there are physical cores (per MPI task), but LAMMPS cannot check for this.

An *Nthreads* value of '*' instructs LAMMPS to use whatever is the default for the given OpenMP environment. This is usually determined via the *OMP_NUM_THREADS* environment variable or the compiler runtime. Please note that in most cases the default for OpenMP capable compilers is to use one thread for each available CPU core when *OMP_NUM_THREADS* is not set, which can lead to extremely bad performance.

Which combination of threads and MPI tasks gives the best performance is difficult to predict and can depend on many components of your input. Not all features of LAMMPS support OpenMP and the parallel efficiency can be very different, too.

The *mode* setting specifies where neighbor list calculations will be multi-threaded as well. If *mode* is force, neighbor list calculation is performed in serial. If *mode* is force/neigh, a multi-threaded neighbor list build is used. Using the force/neigh setting is almost always faster and should produce idential neighbor lists at the expense of using some more memory (neighbor list pages are always allocated for all threads at the same time and each thread works on its own pages).

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

The cuda style of this command can only be invoked if LAMMPS was built with the USER-CUDA package. See the Making LAMMPS section for more info.

The gpu style of this command can only be invoked if LAMMPS was built with the GPU package. See the Making LAMMPS section for more info.

The omp style of this command can only be invoked if LAMMPS was built with the USER-OMP package. See the Making LAMMPS section for more info.

Related commands:

suffix

Default:

If the "-sf gpu" command-line switch is used then it is as if the command "package gpu force/neigh 0 0 1" were invoked, to specify default settings for the GPU package. If the command-line switch is not used, then no defaults are set, and you must specify the appropriate package command in your input script.

The default settings for the USER CUDA package are "package cuda gpu 2". This is the case whether the "-sf cuda" command-line switch is used or not.

If the "-sf omp" command-line switch is used then it is as if the command "package omp *" were invoked, to specify default settings for the USER-OMP package. If the command-line switch is not used, then no defaults are set, and you must specify the appropriate package command in your input script.

pair_style adp command

pair_style adp/omp command

Syntax:

```
pair_style adp
```

Examples:

```
pair_style adp
pair_coeff * * Ta.adp Ta
pair_coeff * * ../potentials/AlCu.adp Al Al Cu
```

Description:

Style *adp* computes pairwise interactions for metals and metal alloys using the angular dependent potential (ADP) of [(Mishin)](#), which is a generalization of the [embedded atom method (EAM) potential](#). The LAMMPS implementation is discussed in [(Singh)](#). The total energy Ei of an atom I is given by

$$E_i = F_\alpha\left(\sum_{j \neq i} \rho_\beta(r_{ij})\right) + \frac{1}{2}\sum_{j \neq i}\phi_{\alpha\beta}(r_{ij}) + \frac{1}{2}\sum_s (\mu_i^s)^2 + \frac{1}{2}\sum_{s,t}(\lambda_i^{st})^2 - \frac{1}{6}\nu_i^2$$

$$\mu_i^s = \sum_{j \neq i} u_{\alpha\beta}(r_{ij})r_{ij}^s$$

$$\lambda_i^{st} = \sum_{j \neq i} w_{\alpha\beta}(r_{ij})r_{ij}^s r_{ij}^t$$

$$\nu_i = \sum_s \lambda_i^{ss}$$

where F is the embedding energy which is a function of the atomic electron density rho, phi is a pair potential interaction, alpha and beta are the element types of atoms I and J, and s and t = 1,2,3 and refer to the cartesian coordinates. The mu and lambda terms represent the dipole and quadruple distortions of the local atomic environment which extend the original EAM framework by introducing angular forces.

Note that unlike for other potentials, cutoffs for ADP potentials are not set in the pair_style or pair_coeff command; they are specified in the ADP potential files themselves. Likewise, the ADP potential files list atomic masses; thus you do not need to use the [mass](#) command to specify them.

The NIST WWW site distributes and documents ADP potentials:

```
http://www.ctcms.nist.gov/potentials
```

Note that these must be converted into the extended DYNAMO *setfl* format discussed below.

The NIST site is maintained by Chandler Becker (cbecker at nist.gov) who is good resource for info on interatomic potentials and file formats.

Only a single pair_coeff command is used with the *adp* style which specifies an extended DYNAMO *setfl* file, which contains information for M elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of extended *setfl* elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, the potentials/AlCu.adp file, included in the potentials directory of the LAMMPS distrbution, is an extended *setfl* file which has tabulated ADP values for w elements and their alloy interactions: Cu and Al. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Al, and the 4th to be Cu, you would use the following pair_coeff command:

```
pair_coeff * * AlCu.adp Al Al Al Cu
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Al arguments map LAMMPS atom types 1,2,3 to the Al element in the extended *setfl* file. The final Cu argument maps LAMMPS atom type 4 to the Al element in the extended *setfl* file. Note that there is no requirement that your simulation use all the elements specified by the extended *setfl* file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *adp* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Adp files in the *potentials* directory of the LAMMPS distribution have an ".adp" suffix. A DYNAMO *setfl* file extended for ADP is formatted as follows. Basically it is the standard *setfl* format with additional tabulated functions u and w added to the file after the tabulated pair potentials. See the pair_eam command for further details on the *setfl* format.

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

Following the 5 header lines are Nelements sections, one for each element, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) (Nr values)

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed for all i,j element pairs in the same format as other arrays. Since these interactions are symmetric (i,j = j,i) only phi arrays with i >= j are listed, in the following order: i,j = (1,1), (2,1), (2,2), (3,1), (3,2), (3,3), (4,1), ..., (Nelements, Nelements). The tabulated values for each phi function are listed as r*phi (in units of eV-Angstroms), since they are for atom pairs, the same as for other EAM files.

After the phi(r) arrays, each of the u(r) arrays are listed in the same order with the same assumptions of symmetry. Directly following the u(r), the w(r) arrays are listed. Note that phi(r) is the only array tabulated with a scaling by r.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in

pair_style adp/omp command

Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, no special mixing rules are needed, since the ADP potential files specify alloy interactions explicitly.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in tabulated potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default).

Related commands:

pair_coeff, pair_eam

Default: none

(**Mishin**) Mishin, Mehl, and Papaconstantopoulos, Acta Mater, 53, 4029 (2005).

(**Singh**) Singh and Warner, Acta Mater, 58, 5797-5805 (2010),

pair_style airebo command

pair_style airebo/omp command

pair_style rebo command

pair_style rebo/omp command

Syntax:

```
pair_style style cutoff LJ_flag TORSION_flag
```

- style = *airebo* or *rebo*
- cutoff = LJ cutoff (sigma scale factor) (AIREBO only)
- LJ_flag = 0/1 to turn off/on the LJ term (AIREBO only, optional)
- TORSION_flag = 0/1 to turn off/on the torsion term (AIREBO only, optional)

Examples:

```
pair_style airebo 3.0
pair_style airebo 2.5 1 0
pair_coeff * * ../potentials/CH.airebo H C

pair_style rebo
pair_coeff * * ../potentials/CH.airebo H C
```

Description:

The *airebo* pair style computes the Adaptive Intermolecular Reactive Empirical Bond Order (AIREBO) Potential of [(Stuart)](#) for a system of carbon and/or hydrogen atoms. Note that this is the initial formulation of AIREBO from 2000, not the later formulation. The *rebo* pair style computes the Reactive Empirical Bond Order (REBO) Potential of [(Brenner)](#). Note that this is the so-called 2nd generation REBO from 2002, not the original REBO from 1990. As discussed below, 2nd generation REBO is closely related to the intial AIREBO; it is just a subset of the potential energy terms.

The AIREBO potential consists of three terms:

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} \left[E_{ij}^{REBO} + E_{ij}^{LJ} + \sum_{k \neq i,j} \sum_{l \neq i,j,k} E_{kijl}^{TORSION} \right]$$

By default, all three terms are included. For the *airebo* style, if the two optional flag arguments to the pair_style command are included, the LJ and torsional terms can be turned off. Note that both or neither of the flags must be included. If both of the LJ an torsional terms are turned off, it becomes the 2nd-generation REBO potential, with a small caveat on the spline fitting procedure mentioned below. This can be specified directly as pair_style rebo with no additional arguments.

The detailed formulas for this potential are given in [(Stuart)](#); here we provide only a brief description.

The E_REBO term has the same functional form as the hydrocarbon REBO potential developed in (Brenner). The coefficients for E_REBO in AIREBO are essentially the same as Brenner's potential, but a few fitted spline values are slightly different. For most cases the E_REBO term in AIREBO will produce the same energies, forces and statistical averages as the original REBO potential from which it was derived. The E_REBO term in the AIREBO potential gives the model its reactive capabilities and only describes short-ranged C-C, C-H and H-H interactions (r < 2 Angstroms). These interactions have strong coordination-dependence through a bond order parameter, which adjusts the attraction between the I,J atoms based on the position of other nearby atoms and thus has 3- and 4-body dependence.

The E_LJ term adds longer-ranged interactions (2 < r < cutoff) using a form similar to the standard Lennard Jones potential. The E_LJ term in AIREBO contains a series of switching functions so that the short-ranged LJ repulsion (1/r^12) does not interfere with the energetics captured by the E_REBO term. The extent of the E_LJ interactions is determined by the *cutoff* argument to the pair_style command which is a scale factor. For each type pair (C-C, C-H, H-H) the cutoff is obtained by multiplying the scale factor by the sigma value defined in the potential file for that type pair. In the standard AIREBO potential, sigma_CC = 3.4 Angstroms, so with a scale factor of 3.0 (the argument in pair_style), the resulting E_LJ cutoff would be 10.2 Angstroms.

The E_TORSION term is an explicit 4-body potential that describes various dihedral angle preferences in hydrocarbon configurations.

Only a single pair_coeff command is used with the *airebo* or *rebo* style which specifies an AIREBO potential file with parameters for C and H. Note that the *rebo* style in LAMMPS uses the same AIREBO-formatted potential file. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of AIREBO elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, if your LAMMPS simulation has 4 atom types and you want the 1st 3 to be C, and the 4th to be H, you would use the following pair_coeff command:

```
pair_coeff * * CH.airebo C C C H
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three C arguments map LAMMPS atom types 1,2,3 to the C element in the AIREBO file. The final H argument maps LAMMPS atom type 4 to the H element in the SW file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *airebo* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The parameters/coefficients for the AIREBO potentials are listed in the CH.airebo file to agree with the original (Stuart) paper. Thus the parameters are specific to this potential and the way it was fit, so modifying the file should be done cautiously.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support the pair_modify mix, shift, table, and tail options.

These pair styles do not write their information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles are part of the MANYBODY package. They are only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

These pair potentials require the newton setting to be "on" for pair interactions.

The CH.airebo potential file provided with LAMMPS (see the potentials directory) is parameterized for metal units. You can use the AIREBO or REBO potential with any LAMMPS units, but you would need to create your own AIREBO potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Stuart) Stuart, Tutein, Harrison, J Chem Phys, 112, 6472-6486 (2000).

(Brenner) Brenner, Shenderova, Harrison, Stuart, Ni, Sinnott, J Physics: Condensed Matter, 14, 783-802 (2002).

pair_style awpmd/cut command

Syntax:

```
pair_style awpmd/cut Rc keyword value ...
```

- Rc = global cutoff, -1 means cutoff of half the shortest box length
- zero or more keyword/value pairs may be appended
- keyword = *hartree* or *dproduct* or *uhf* or *free* or *pbc* or *fix* or *harm* or *ermscale* or *flex_press*

```
hartree value = none
  dproduct value = none
  uhf value = none
  free value = none
  pbc value = Plen
    Plen = periodic width of electron = -1 or positive value (distance units)
  fix value = Flen
    Flen = fixed width of electron = -1 or positive value (distance units)
  harm value = width
    width = harmonic width constraint
  ermscale value = factor
    factor = scaling between electron mass and width variable mass
  flex_press value = none
```

Examples:

```
pair_style awpmd/cut -1
pair_style awpmd/cut 40.0 uhf free
pair_coeff * *
pair_coeff 2 2 20.0
```

Description:

This pair style contains an implementation of the Antisymmetrized Wave Packet Molecular Dynamics (AWPMD) method. Need citation here. Need basic formulas here. Could be links to other documents.

Rc is the cutoff.

The pair_style command allows for several optional keywords to be specified.

The *hartree*, *dproduct*, and *uhf* keywords specify the form of the initial trial wave function for the system. If the *hartree* keyword is used, then a Hartree multielectron trial wave function is used. If the *dproduct* keyword is used, then a trial function which is a product of two determinants for each spin type is used. If the *uhf* keyword is used, then an unrestricted Hartree-Fock trial wave function is used.

The *free*, *pbc*, and *fix* keywords specify a width constraint on the electron wavepackets. If the *free* keyword is specified, then there is no constraint. If the *pbc* keyword is used and *Plen* is specified as -1, then the maximum width is half the shortest box length. If *Plen* is a positive value, then the value is the maximum width. If the *fix* keyword is used and *Flen* is specified as -1, then electrons have a constant width that is read from the data file. If *Flen* is a positive value, then the constant width for all electrons is set to *Flen*.

The *harm* keyword allow oscillations in the width of the electron wavepackets. More details are needed.

The *ermscale* keyword specifies a unitless scaling factor between the electron masses and the width variable mass. More details needed.

If the *flex_press* keyword is used, then a contribution from the electrons is added to the total virial and pressure of the system.

This potential is designed to be used with atom_style wavepacket definitions, in order to handle the description of systems with interacting nuclei and explicit electrons.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *awpmd/cut*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for this pair style.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default:

These are the defaults for the pair_style keywords: *hartree* for the initial wavefunction, *free* for the wavepacket width.

pair_style beck command

pair_style beck/gpu command

pair_style beck/omp command

Syntax:

```
pair_style beck Rc
```

- Rc = cutoff for interactions (distance units)

Examples:

```
pair_style beck 8.0
pair_coeff * * 399.671876712 0.0000867636112694 0.675 4.390 0.0003746
pair_coeff 1 1 399.671876712 0.0000867636112694 0.675 4.390 0.0003746 6.0
```

Description:

Style *beck* computes interactions based on the potential by (Beck), originally designed for simulation of Helium. It includes truncation at a cutoff distance Rc.

$$E\left(r\right) = A\exp\left[-\alpha r - \beta r^6\right] - \frac{B}{\left(r^2 + a^2\right)^3}\left(1 + \frac{2.709 + 3a^2}{r^2 + a^2}\right) \qquad r < R_c$$

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands.

- A (energy units)
- B (energy-distance^6 units)
- a (distance units)
- alpha (1/distance units)
- beta (1/distance^6 units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff Rc is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your

input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, coeffiecients must be specified. No default miture rules are used.

This pair style does not support the [pair_modify](#) shift option for the energy of the pair interaction.

The [pair_modify](#) table option is not relevant for this pair style.

This pair style does not support the [pair_modify](#) tail option for adding long-range tail corrections.

This pair style writes its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

(Beck) Beck, Molecular Physics, 14, 311 (1968).

pair_style body command

Syntax:

```
pair_style body cutoff
```

cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style body 3.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.5 2.5
```

Description:

Style *body* is for use with body particles and calculates pairwise body/body interactions as well as interactions between body and point-particles. See Section_howto 14 of the manual and the body doc page for more details on using body particles.

This pair style is designed for use with the "nparticle" body style, which is specified as an argument to the "atom-style body" command. See the body doc page for more details about the body styles LAMMPS supports. The "nparticle" style treats a body particle as a rigid body composed of N sub-particles.

The coordinates of a body particle are its center-of-mass (COM). If the COMs of a pair of body particles are within the cutoff (global or type-specific, as specified above), then all interactions between pairs of sub-particles in the two body particles are computed. E.g. if the first body particle has 3 sub-particles, and the second has 10, then 30 interactions are computed and summed to yield the total force and torque on each body particle.

IMPORTANT NOTE: In the example just described, all 30 interactions are computed even if the distance between a particular pair of sub-particles is greater than the cutoff. Likewise, no interaction between two body particles is computed if the two COMs are further apart than the cutoff, even if the distance between some pairs of their sub-particles is within the cutoff. Thus care should be used in defining the cutoff distances for body particles, depending on their shape and size.

Similar rules apply for a body particle interacting with a point particle. The distance between the two particles is calculated using the COM of the body particle and the position of the point particle. If the distance is within the cutoff and the body particle has N sub-particles, then N interactions with the point particle are computed and summed. If the distance is not within the cutoff, no interactions between the body and point particle are computed.

The interaction between two sub-particles, or a sub-particle and point particle, or betwee two point particles is computed as a Lennard-Jones interaction, using the standard formula

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

where Rc is the cutoff. As explained above, an interaction involving one or two body sub-particles may be computed even for r > Rc.

For style *body*, the following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the BODY package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Defining particles to be bodies so they participate in body/body or body/particle interactions requires the use of the atom_style body command.

Related commands:

pair_coeff, fix_rigid

Default: none

pair_style bop command

Syntax:

```
pair_style bop keyword ...
```

- zero or more keywords may be appended
- keyword = *table* or *save* or *sigmaoff*

```
  table = BOP potential file has tabulated form
    save = pre-compute and save some values
    sigmaoff = assume a_sigma = 0
```

Examples:

```
pair_style bop
pair_coeff * * ../potentials/CdTe_bop Cd Te
pair_style bop table save
pair_coeff * * ../potentials/CdTe.bop.table Cd Te Te
communicate single cutoff 14.70
```

Description:

The *bop* pair style computes Bond-Order Potentials (BOP) based on quantum mechanical theory incorporating both sigma and pi bondings. By analytically deriving the BOP from quantum mechanical theory its transferability to different phases can approach that of quantum mechanical methods. This particular BOP is extremely effective at modeling III-V and II-VI compounds such as GaAs and CdTe. This potential is similar to the original BOP developed by Pettifor ([Pettifor_1](#), [Pettifor_2](#), [Pettifor_3](#)) and later updated by Murdick, Zhou, and Ward ([Murdick](#), [Ward](#)).

The BOP potential consists of three terms:

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=i_1}^{i_N} \phi_{ij}\left(r_{ij}\right) - \sum_{i=1}^{N} \sum_{j=i_1}^{i_N} \beta_{\sigma,ij}\left(r_{ij}\right) \cdot \Theta_{\sigma,ij} - \sum_{i=1}^{N} \sum_{j=i_1}^{i_N} \beta_{\pi,ij}\left(r_{ij}\right) \cdot \Theta_{\pi,ij} + U_{prom}$$

where phi_ij(r_ij) is a short-range two-body function representing the repulsion between a pair of ion cores, beta_(sigma,ij)(r_ij) and beta_(sigma,ij)(r_ij) are respectively sigma and pi bond ingtegrals, THETA_(sigma,ij) and THETA_(pi,ij) are sigma and pi bond-orders, and U_prom is the promotion energy for sp-valent systems.

The detailed formulas for this potential are given in Ward ([Ward](#)); here we provide only a brief description.

The repulsive energy phi_ij(r_ij) and the bond integrals beta_(sigma,ij)(r_ij) and beta_(phi,ij)(r_ij) are functions of the interatomic distance r_ij between atom i and j. Each of these potentials has a smooth cutoff at a radius of r_(cut,ij). These smooth cutoffs ensure stable behavior at situations with high sampling near the cutoff such as melts and surfaces.

The bond-orders can be viewed as environment-dependent local variables that are ij bond specific. The maximum value of the sigma bond-order (THETA_sigma) is 1, while that of the pi bond-order (THETA_pi) is 2, attributing to a maximum value of the total bond-order (THETA_sigma+THETA_pi) of 3. The sigma and pi bond-orders reflect the ubiquitous single-, double-, and triple- bond behavior of chemistry. Their analytical

expressions can be derived from tight- binding theory by recursively expanding an inter-site Green's function as a continued fraction. To accurately represent the bonding with a computationally efficient potential formulation suitable for MD simulations, the derived BOP only takes (and retains) the first two levels of the recursive representations for both the sigma and the pi bond-orders. Bond-order terms can be understood in terms of molecular orbital hopping paths based upon the Cyrot-Lackmann theorem (Pettifor_1). The sigma bond-order with a half-full valence band filling. This pi bond-order expression also contains also contains a three-member ring term that allows implementation of an asymmetric density of states, which helps to either stabilize or destabilize close-packed structures. The pi bond-order includes hopping paths of length 4. This enables the incorporation of dihedral angles effects.

The cutoffs for the various interactions are defined in the BOP potential file.

IMPORTANT NOTE: You must use the communicate cutoff command to insure ghost atoms are acquired at a distance 3x further than the largest BOP cutoff (for a particular pair of elements). E.g. if the BOP cutoff is 4.9 Angstroms, then the ghost atom communication needs to be 14.7 Angstroms or greater as in the example above. This is because the BOP formulation uses neighbors of neighbors of neighbors to enumerate all the required many-body interactions. LAMMPS will generate an error if you do not use an appropriate setting for the communicate cutoff command.

Several options can be specified as keywords with the pair_style command.

The *table* keyword tells LAMMPS what format the BOP potential file is in. The default is a non-tabulated form. If the *table* keyword is used, the file is in a tabulated form containing pre-tabulated pair functions for phi_ij(r_ij), beta_(sigma,ij)(r_ij), and beta_(pi,ij)(r_ij). This allows you to use your own functional form for various interactions.

The *save* keyword gives you the option to calculate and store in advance a set of distances, angles, and derivatives of angles. The default is to not do this, but to calculate the various quantities on-the-fly each time they are needed. The former may be faster, but takes more memory. The latter requires less memory, but may be slower. It is best to test this option to see if it makes a difference on your machine for the specific problem you are modeling.

The *sigmaoff* keyword optimizes the BOP equations for the case of a_sigma = 0. For some published BOP potentials, a_sigma = 0 and several terms in the BOP equationas drop out. If this is the case, specifying *sigmaoff* will typically speed up the BOP pair style.

Only a single pair_coeff command is used with the *bop* style which specifies a BOP potential file, with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of BOP elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, imagine the CdTe.bop file has BOP values for Cd and Te. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Cd, and the 4th to be Te, you would use the following pair_coeff command:

```
pair_coeff * * CdTe Cd Cd Cd Te
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Cd arguments map LAMMPS atom types 1,2,3 to the Cd element in the BOP file. The final Te argument maps LAMMPS atom type 4 to the Te element in the BOP file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *bop* potential is used as part of the *hybrid* pair style. The NULL values

pair_style bop command   733

are placeholders for atom types that will be used with other potentials.

BOP files in the *potentials* directory of the LAMMPS distribution have a ".bop" or ".bop.table" suffix, depending on whether they are of the non-tabulated or tabulated form, as described above.

The parameters/coefficients format for the both kinds of BOP files are given below with variables matching the formulation of Ward ([Ward](#)). Each header line containing a ":" is preceded by a blank line.

- Line 1: elements: (header)
- Line 2: #elements N

The first two lines are followed by N lines containing the atomic number and mass of each element.

Non-tabulated potential file format:

Following the definition of the elements is the block of global variables for spline and quadratic fits of THETA_(S,ij) and its components THETA_0, THETA_1, and S.

- Line 1: global: (header)
- Line 2: delta_1-delta_7 (if all are not used in the particular formulation, set unused values to 0.0)
- Line 3: ncutoff, r_big, r_small (r_big and r_small are parameters for pairwise paramters gamma typically set to 0.99 and 0.01, respectively)
- Line 4: which, alpha, nfunc (these are options for the spline which=1.0 (means using a smooth function); which=2.0 (spline), alpha is a parameter in the spline, nfunc is the type of GSP function (f_ij) (nfunc=1 is the published equation from Ward ([Ward](#)); nfunc=2 $f\_ij(r\_ij)=\exp(n\_ij*r\_ij)$; nfunc=3 $f\_ij(r\_ij)=1/(r\_ij)^{(n\_ij)}$).
- Line 5: alpha_1, beta_1, gamma_1 (alpha_1=first coefficient for THETA_0; beta_1=first exponent for THETA_0; gamma_1=second exponent for THETA_0)
- Line 6: alpha_2, beta_2 (alpha_2=second coefficient for S; beta_2=first exponent for S)
- Line 7: alpha_3, beta_3 (alpha_3=first coefficient for THETA_1; beta_3=second coefficient for THETA_1)

The next block contains constants for the environment depend promotional energy for sp-valent systems, each of which are species dependent. Refer to Pettifor ([Pettifor_3](#)) for constant definitions. As well as one species dependent parameter p_pi.

- Line 1: ptrs: (header)

Following the ptrs header there are N lines for e_1-e_N containing $(A\_ij)^{(mu*nu)}$, $delta^{mu}$, p_pi

- Line 2: $(A\_ij)^{(mu*nu)}$, $delta^{mu}$, p_pi (for e_1)
- Line 3: $(A\_ij)^{(mu*nu)}$, $delta^{mu}$, p_pi (for e_2 and continues to e_N)

The next block contains constants for the pair interactions.

- Line 1: pairs: (header)

Following the header the block contains a series of constants for the number of pair interaction types, the block will be broken up into parameters for e_i-e_j with i=0->N, j=i->N. Each single interaction section for this block will contain (see [Ward](#) for parameter definitions):

- Line 2: r_0, r_c, r_1, r_cut (for e_1-e_2 interactions)
- Line 3: m, n, n_c
- Line 4: phi_0, beta_(sigma,0), beta_(pi,0)

- Line 5: a_sigma, c_sigma, delta_sigma (From complete formulation of 1/2 full valance shell for this particular formulation delta_sigma=0)
- Line 6: a_pi, c_pi, delta_pi
- Line 7: f_sigma, k_sigma, delta_3 (This delta_3 is similar to that of the previous section but is interaction type dependent)
- Line 8: r_0, r_c, r_1, r_cut (for e_1-e_2 interactions and repeats as above)

The next block contains tris.

- Line 1: tris: (header)

Following the header there is a line for each three body interaction types as e_j-e_i-e_k with i->N, j->N, k=j->N

- Line 2: g_sigma0, g_sigma1, g_sigma2 (these are coefficients for g_(sigma,ijk)(theta_ijk) for e_1-e_1-e_1 interaction. Ward contains the full expressions for the constants as functions of b_(sigma,ijk), p_(sigma,ijk), u(sigma,ijk)
- Line 3: g_sigma0, g_sigma1, g_sigma2 (for e_1-e_1-e_2)

This would be the end of the potential parameter file without pre- tabulated data.

Tabulated potential file format:

The parameters/coefficients format for the BOP potentials input file containing pre-tabulated functions of is given below with variables matching the formulation of Ward (Ward).

- Line 1: # elements N

The first two lines are followed by N lines containing the atomic number and mass of each element THETA_0 and THETA_1 (see Ward).

Following the definition of the elements several global variables for the tabulated functions are given.

- Line 1: nr, nBOt (nr is the number of divisions the radius is broken into for function tables and MUST be a factor of 5; nBOt is the number of divisions for the tabulated values of THETA_(S,ij)
- Line 2: delta_1-delta_7 (if all are not used in the particular
- formulation, set unused values to 0.0)

Following this N lines for e_1-e_N containing p_pi.

- Line 3: p_pi (for e_1)
- Line 4: p_pi (for e_2 and continues to e_N)

The next section contains several pair constants for the number of interaction types e_i-e_j, with i=1->N, j=i->N

- Line 1: r_cut (for e_1-e_1 interactions)
- Line 2: c_sigma, a_sigma, c_pi, a_pi
- Line 3: delta_sigma, delta_pi
- Line 4: f_sigma, k_sigma, delta_3 (This delta_3 is similar to that of the previous section but is interaction type dependent)

The next section contains a line for each three body interaction type e_j-e_i-e_k with i=0->N, j=0->N, k=j->N

- Line 1: g_(sigma0), g_(sigma1), g_(sigma2) (These are coefficients for g_(sigma,jik)(THETA_ijk) for e_1-e_1-e_1 interaction. [Ward] contains the full expressions for the constants as functions of b_(sigma,ijk), p_(sigma,ijk), u_(sigma,ijk))
- Line 2: g_(sigma0), g_(sigma1), g_(sigma2) (for e_1-e_1-e_2)

The next section contains a block for each interaction type for the phi_ij(r_ij). Each block has nr entries with 5 entries per line.

- Line 1: phi(r1), phi(r2), phi(r3), phi(r4), phi(r5) (for the e_1-e_1 interaction type)
- Line 2: phi(r6), phi(r7), phi(r8), phi(r9), phi(r10) (this continues until nr)
- ...
- Line nr/5_1: phi(r1), phi(r2), phi(r3), phi(r4), phi(r5), (for the e_1-e_1 interaction type)

The next section contains a block for each interaction type for the beta_(sigma,ij)(r_ij). Each block has nr entries with 5 entries per line.

- Line 1: beta_sigma(r1), beta_sigma(r2), beta_sigma(r3), beta_sigma(r4), beta_sigma(r5) (for the e_1-e_1 interaction type)
- Line 2: beta_sigma(r6), beta_sigma(r7), beta_sigma(r8), beta_sigma(r9), beta_sigma(r10) (this continues until nr)
- ...
- Line nr/5+1: beta_sigma(r1), beta_sigma(r2), beta_sigma(r3), beta_sigma(r4), beta_sigma(r5) (for the e_1-e_2 interaction type)

The next section contains a block for each interaction type for beta_(pi,ij)(r_ij). Each block has nr entries with 5 entries per line.

- Line 1: beta_pi(r1), beta_pi(r2), beta_pi(r3), beta_pi(r4), beta_pi(r5) (for the e_1-e_1 interaction type)
- Line 2: beta_pi(r6), beta_pi(r7), beta_pi(r8), beta_pi(r9), beta_pi(r10) (this continues until nr)
- ...
- Line nr/5+1: beta_pi(r1), beta_pi(r2), beta_pi(r3), beta_pi(r4), beta_pi(r5) (for the e_1-e_2 interaction type)

The next section contains a block for each interaction type for the THETA_(S,ij)((THETA_(sigma,ij))^(1/2), f_(sigma,ij)). Each block has nBOt entries with 5 entries per line.

- Line 1: THETA_(S,ij)(r1), THETA_(S,ij)(r2), THETA_(S,ij)(r3), THETA_(S,ij)(r4), THETA_(S,ij)(r5) (for the e_1-e_2 interaction type)
- Line 2: THETA_(S,ij)(r6), THETA_(S,ij)(r7), THETA_(S,ij)(r8), THETA_(S,ij)(r9), THETA_(S,ij)(r10) (this continues until nBOt)
- ...
- Line nBOt/5+1: THETA_(S,ij)(r1), THETA_(S,ij)(r2), THETA_(S,ij)(r3), THETA_(S,ij)(r4), THETA_(S,ij)(r5) (for the e_1-e_2 interaction type)

The next section contains a block of N lines for e_1-e_N

- Line 1: delta^mu (for e_1)
- Line 2: delta^mu (for e_2 and repeats to e_N)

The last section contains more constants for e_i-e_j interactions with i=0->N, j=i->N

- Line 1: (A_ij)^(mu*nu) (for e1-e1)
- Line 2: (A_ij)^(mu*nu) (for e1-e2 and repeats as above)

Mixing, shift, table tail correction, restart:

This pair style does not support the [pair_modify](#) mix, shift, table, and tail options.

This pair style does not write its information to [binary restart files](#), since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles are part of the MANYBODY package. They are only enabled if LAMMPS was built with that package (which it is by default). See the [Making LAMMPS](#) section for more info.

These pair potentials require the [newtion](#) setting to be "on" for pair interactions.

The CdTe.bop and GaAs.bop potential files provided with LAMMPS (see the potentials directory) are parameterized for metal [units](#). You can use the BOP potential with any LAMMPS units, but you would need to create your own BOP potential file with coefficients listed in the appropriate units if your simulation does not use "metal" units.

Related commands:

[pair_coeff](#)

Default:

non-tabulated potential file, a_0 is non-zero.

(Pettifor_1) D.G. Pettifor and I.I. Oleinik, Phys. Rev. B, 59, 8487 (1999).

(Pettifor_2) D.G. Pettifor and I.I. Oleinik, Phys. Rev. Lett., 84, 4124 (2000).

(Pettifor_3) D.G. Pettifor and I.I. Oleinik, Phys. Rev. B, 65, 172103 (2002).

(Murdick) D.A. Murdick, X.W. Zhou, H.N.G. Wadley, D. Nguyen-Manh, R. Drautz, and D.G. Pettifor, Phys. Rev. B, 73, 45206 (2006).

(Ward) D.K. Ward, X.W. Zhou, B.M. Wong, F.P. Doty, and J.A. Zimmerman, Phys. Rev. B, 85,115206 (2012).

pair_style born command

pair_style born/omp command

pair_style born/gpu command

pair_style born/coul/long command

pair_style born/coul/long/cuda command

pair_style born/coul/long/gpu command

pair_style born/coul/long/omp command

pair_style born/coul/msm command

pair_style born/coul/msm/omp command

pair_style born/coul/wolf command

pair_style born/coul/wolf/gpu command

pair_style born/coul/wolf/omp command

Syntax:

```
pair_style style args
```

- style = *born* or *born/coul/long* or *born/coul/msm* or *born/coul/wolf*
- args = list of arguments for a particular style

```
born args = cutoff
    cutoff = global cutoff for non-Coulombic interactions (distance units)
  born/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for non-Coulombic (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  born/coul/msm args = cutoff (cutoff2)
    cutoff = global cutoff for non-Coulombic (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  born/coul/wolf args = alpha cutoff (cutoff2)
    alpha = damping parameter (inverse distance units)
    cutoff = global cutoff for non-Coulombic (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style born 10.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51

pair_style born/coul/long 10.0
pair_style born/coul/long 10.0 8.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
```

pair_style born command

```
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51

pair_style born/coul/msm 10.0
pair_style born/coul/msm 10.0 8.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51

pair_style born/coul/wolf 0.25 10.0
pair_style born/coul/wolf 0.25 10.0 9.0
pair_coeff * * 6.08 0.317 2.340 24.18 11.51
pair_coeff 1 1 6.08 0.317 2.340 24.18 11.51
```

Description:

The *born* style computes the Born-Mayer-Huggins or Tosi/Fumi potential described in (Fumi and Tosi), given by

$$E = A \exp\left(\frac{\sigma - r}{\rho}\right) - \frac{C}{r^6} + \frac{D}{r^8} \qquad r < r_c$$

where sigma is an interaction-dependent length parameter, rho is an ionic-pair dependent length parameter, and Rc is the cutoff.

The styles with *coul/long* or *coul/msm* add a Coulombic term as described for the lj/cut pair styles. An additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* of *msm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

If one cutoff is specified for the *born/coul/long* and *born/coul/msm* style, it is used for both the A,C,D and Coulombic terms. If two cutoffs are specified, the first is used as the cutoff for the A,C,D terms, and the second is the cutoff for the Coulombic term.

The *born/coul/wolf* style adds a Coulombic term as described for the Wolf potential in the coul/wolf pair style.

Note that these potentials are related to the Buckingham potential.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- rho (distance units)
- sigma (distance units)
- C (energy units * distance units^6)
- D (energy units * distance units^8)
- cutoff (distance units)

The second coefficient, rho, must be greater than zero.

The last coefficient is optional. If not specified, the global A,C,D cutoff specified in the pair_style command is used.

pair_style born/coul/wolf/omp command   739

For *born/coul/long* and *born/coul/wolf* no Coulombic cutoff can be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These styles support the pair_modify shift option for the energy of the exp(), 1/r^6, and 1/r^8 portion of the pair interaction.

The *born/coul/long* pair style supports the pair_modify table option ti tabulate the short-range portion of the long-range Coulombic interaction.

These styles support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

Thess styles writes thei information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *born/coul/long* style is part of the KSPACE package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style buck

Default: none

Fumi and Tosi, J Phys Chem Solids, 25, 31 (1964), Fumi and Tosi, J Phys Chem Solids, 25, 45 (1964).

pair_style brownian command

pair_style brownian/omp command

pair_style brownian/poly command

pair_style brownian/poly/omp command

Syntax:

```
pair_style style mu flaglog flagfld cutinner cutoff t_target seed flagHI flagVF
```

- style = *brownian* or *brownian/poly*
- mu = dynamic viscosity (dynamic viscosity units)
- flaglog = 0/1 log terms in the lubrication approximation on/off
- flagfld = 0/1 to include/exclude Fast Lubrication Dynamics effects
- cutinner = inner cutoff distance (distance units)
- cutoff = outer cutoff for interactions (distance units)
- t_target = target temp of the system (temperature units)
- seed = seed for the random number generator (positive integer)
- flagHI (optional) = 0/1 to include/exclude 1/r hydrodynamic interactions
- flagVF (optional) = 0/1 to include/exclude volume fraction corrections in the long-range isotropic terms

Examples:

```
pair_style brownian 1.5 1 1 2.01 2.5 2.0 5878567 (assuming radius = 1)
pair_coeff 1 1 2.05 2.8
pair_coeff * *
```

Description:

Styles *brownian* and *brownain/poly* compute Brownian forces and torques on finite-size particles. The former requires monodisperse spherical particles; the latter allows for polydisperse spherical particles.

These pair styles are designed to be used with either the [pair_style lubricate](#) or [pair_style lubricateU](#) commands to provide thermostatting when dissipative lubrication forces are acting. Thus the parameters *mu*, *flaglog*, *flagfld*, *cutinner*, and *cutoff* should be specified consistent with the settings in the lubrication pair styles. For details, refer to either of the lubrication pair styles.

The *t_target* setting is used to specify the target temperature of the system. The random number *seed* is used to generate random numbers for the thermostatting procedure.

The *flagHI* and *flagVF* settings are optional. Neither should be used, or both must be defined.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- cutinner (distance units)
- cutoff (distance units)

The two coefficients are optional. If neither is specified, the two cutoffs specified in the pair_style command are used. Otherwise both must be specified.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the two cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These styles are part of the FLD package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Only spherical monodisperse particles are allowed for pair_style brownian.

Only spherical particles are allowed for pair_style brownian/poly.

Related commands:

pair_coeff, pair_style lubricate, pair_style lubricateU

Default:

The default settings for the optional args are flagHI = 1 and flagVF = 1.

pair_style buck command

pair_style buck/cuda command

pair_style buck/gpu command

pair_style buck/omp command

pair_style buck/coul/cut command

pair_style buck/coul/cut/cuda command

pair_style buck/coul/cut/gpu command

pair_style buck/coul/cut/omp command

pair_style buck/coul/long command

pair_style buck/coul/long/cuda command

pair_style buck/coul/long/gpu command

pair_style buck/coul/long/omp command

pair_style buck/coul/msm command

pair_style buck/coul/msm/omp command

Syntax:

```
pair_style style args
```

- style = *buck* or *buck/coul/cut* or *buck/coul/long* or *buck/coul/msm*
- args = list of arguments for a particular style

```
buck args = cutoff
   cutoff = global cutoff for Buckingham interactions (distance units)
  buck/coul/cut args = cutoff (cutoff2)
   cutoff = global cutoff for Buckingham (and Coulombic if only 1 arg) (distance units)
   cutoff2 = global cutoff for Coulombic (optional) (distance units)
  buck/coul/long args = cutoff (cutoff2)
   cutoff = global cutoff for Buckingham (and Coulombic if only 1 arg) (distance units)
   cutoff2 = global cutoff for Coulombic (optional) (distance units)
  buck/coul/msm args = cutoff (cutoff2)
   cutoff = global cutoff for Buckingham (and Coulombic if only 1 arg) (distance units)
   cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style buck 2.5
pair_coeff * * 100.0 1.5 200.0
```

pair_style buck command   743

```
pair_coeff * * 100.0 1.5 200.0 3.0


pair_style buck/coul/cut 10.0
pair_style buck/coul/cut 10.0 8.0
pair_coeff * * 100.0 1.5 200.0
pair_coeff 1 1 100.0 1.5 200.0 9.0
pair_coeff 1 1 100.0 1.5 200.0 9.0 8.0


pair_style buck/coul/long 10.0
pair_style buck/coul/long 10.0 8.0
pair_coeff * * 100.0 1.5 200.0
pair_coeff 1 1 100.0 1.5 200.0 9.0


pair_style buck/coul/msm 10.0
pair_style buck/coul/msm 10.0 8.0
pair_coeff * * 100.0 1.5 200.0
pair_coeff 1 1 100.0 1.5 200.0 9.0
```

Description:

The *buck* style computes a Buckingham potential (exp/6 instead of Lennard-Jones 12/6) given by

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

where rho is an ionic-pair dependent length parameter, and Rc is the cutoff on both terms.

The styles with *coul/cut* or *coul/long* or *coul/msm* add a Coulombic term as described for the lj/cut pair styles. For *buck/coul/long* and *buc/coul/msm*, an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* or *msm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

If one cutoff is specified for the *born/coul/cut* and *born/coul/long* and *born/coul/msm* styles, it is used for both the A,C and Coulombic terms. If two cutoffs are specified, the first is used as the cutoff for the A,C terms, and the second is the cutoff for the Coulombic term.

Note that these potentials are related to the Born-Mayer-Huggins potential.

IMPORTANT NOTE: For all these pair styles, the terms with A and C are always cutoff. The additional Coulombic term can be cutoff or long-range (no cutoff) depending on whether the style name includes coul/cut or coul/long or coul/msm. If you wish the C/r^6 term to be long-range (no cutoff), then see the pair_style buck/long/coul/long command.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy units)
- rho (distance units)
- C (energy-distance^6 units)
- cutoff (distance units)
- cutoff2 (distance units)

The second coefficient, rho, must be greater than zero.

pair_style buck/coul/msm/omp command   744

The latter 2 coefficients are optional. If not specified, the global A,C and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both A,C and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the A,C and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *buck*, since it has no Coulombic terms.

For *buck/coul/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These styles support the pair_modify shift option for the energy of the exp() and $1/r^6$ portion of the pair interaction.

The *buck/coul/long* pair style supports the pair_modify table option ti tabulate the short-range portion of the long-range Coulombic interaction.

These styles support the pair_modify tail option for adding long-range tail corrections to energy and pressure for the A,C terms in the pair interaction.

These styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *buck/coul/long* style is part of the KSPACE package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style born

Default: none

pair_style buck/long/coul/long command

pair_style buck/long/coul/long/omp command

Syntax:

```
pair_style buck/long/coul/long flag_buck flag_coul cutoff (cutoff2)
```

- flag_buck = *long* or *cut*

  ```
  long = use Kspace long-range summation for the dispersion term 1/r^6
    cut = use a cutoff
  ```
- flag_coul = *long* or *off*

  ```
  long = use Kspace long-range summation for the Coulombic term 1/r
    off = omit the Coulombic term
  ```
- cutoff = global cutoff for Buckingham (and Coulombic if only 1 cutoff) (distance units)
- cutoff2 = global cutoff for Coulombic (optional) (distance units)

Examples:

```
pair_style buck/long/coul/long cut off 2.5
pair_style buck/long/coul/long cut long 2.5 4.0
pair_style buck/long/coul/long long long 2.5 4.0
pair_coeff * * 1 1
pair_coeff 1 1 1 3 4
```

Description:

The *buck/long/coul/long* style computes a Buckingham potential (exp/6 instead of Lennard-Jones 12/6) and Coulombic potential, given by

$$E = Ae^{-r/\rho} - \frac{C}{r^6} \qquad r < r_c$$

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

Rc is the cutoff. If one cutoff is specified in the pair_style command, it is used for both the Buckingham and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the Buckingham and Coulombic terms respectively.

The purpose of this pair style is to capture long-range interactions resulting from both attractive 1/r^6 Buckingham and Coulombic 1/r interactions. This is done by use of the *flag_buck* and *flag_coul* settings. The "Ismail paper has more details on when it is appropriate to include long-range 1/r^6 interactions, using this potential.

If *flag_buck* is set to *long*, no cutoff is used on the Buckingham 1/r^6 dispersion term. The long-range portion is calculated by using the kspace_style ewald/n command. The specified Buckingham cutoff then determines which portion of the Buckingham interactions are computed directly by the pair potential versus which part is computed in reciprocal space via the Kspace style. If *flag_buck* is set to *cut*, the Buckingham interactions are simply cutoff, as with pair_style buck.

If *flag_coul* is set to *long*, no cutoff is used on the Coulombic interactions. The long-range portion is calculated by using any style, including *ewald/n* of the kspace_style command. Note that if *flag_buck* is also set to long, then only the *ewald/n* Kspace style can perform the long-range calculations for both the Buckingham and Coulombic interactions. If *flag_coul* is set to *off*, Coulombic interactions are not computed.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (energy units)
- rho (distance units)
- C (energy-distance^6 units)
- cutoff (distance units)
- cutoff2 (distance units)

The second coefficient, rho, must be greater than zero.

The latter 2 coefficients are optional. If not specified, the global Buckingham and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both Buckingham and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the Buckingham and Coulombic cutoffs for this type pair. Note that if you are using *flag_buck* set to *long*, you cannot specify a Buckingham cutoff for an atom type pair, since only one global Buckingham cutoff is allowed. Similarly, if you are using *flag_coul* set to *long*, you cannot specify a Coulombic cutoff for an atom type pair, since only one global Coulombic cutoff is allowed.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair styles does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This pair style supports the pair_modify shift option for the energy of the exp() and 1/r^6 portion of the pair interaction, assuming *flag_buck* is *cut*.

This pair style does not support the pair_modify shift option for the energy of the Buckingham portion of the pair interaction.

pair_style buck/long/coul/long/omp command 747

This pair style supports the [pair_modify](#) table and table/disp options since they can tabulate the short-range portion of the long-range Coulombic and dispersion interactions.

This pair style write its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the [run_style respa](#) command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the [run_style](#) command for details.

Restrictions:

This style is part of the KSPACE package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info. Note that the KSPACE package is installed by default.

Related commands:

[pair_coeff](#)

Default: none

(Ismail) Ismail, Tsige, In 't Veld, Grest, Molecular Physics (accepted) (2007).

LIGGGHTS Users Manual

pair_style lj/charmm/coul/charmm command

pair_style lj/charmm/coul/charmm/cuda command

pair_style lj/charmm/coul/charmm/omp command

pair_style lj/charmm/coul/charmm/implicit command

pair_style lj/charmm/coul/charmm/implicit/cuda command

pair_style lj/charmm/coul/charmm/implicit/omp command

pair_style lj/charmm/coul/long command

pair_style lj/charmm/coul/long/cuda command

pair_style lj/charmm/coul/long/gpu command

pair_style lj/charmm/coul/long/opt command

pair_style lj/charmm/coul/long/omp command

pair_style lj/charmm/coul/msm command

pair_style lj/charmm/coul/msm/omp command

Syntax:

```
pair_style style args
```

- style = *lj/charmm/coul/charmm* or *lj/charmm/coul/charmm/implicit* or *lj/charmm/coul/long* or *lj/charmm/coul/msm*
- args = list of arguments for a particular style

```
lj/charmm/coul/charmm args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for Lennard Jones (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
  lj/charmm/coul/charmm/implicit args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for LJ (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
  lj/charmm/coul/long args = inner outer (cutoff)
    inner, outer = global switching cutoffs for LJ (and Coulombic if only 2 args)
    cutoff = global cutoff for Coulombic (optional, outer is Coulombic cutoff if only 2 args)
  lj/charmm/coul/msm args = inner outer (cutoff)
    inner, outer = global switching cutoffs for LJ (and Coulombic if only 2 args)
    cutoff = global cutoff for Coulombic (optional, outer is Coulombic cutoff if only 2 args)
```

Examples:

```
pair_style lj/charmm/coul/charmm 8.0 10.0
pair_style lj/charmm/coul/charmm 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
```

pair_style lj/charmm/coul/charmm command   749

```
pair_coeff 1 1 100.0 2.0 150.0 3.5

pair_style lj/charmm/coul/charmm/implicit 8.0 10.0
pair_style lj/charmm/coul/charmm/implicit 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5


pair_style lj/charmm/coul/long 8.0 10.0
pair_style lj/charmm/coul/long 8.0 10.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5


pair_style lj/charmm/coul/msm 8.0 10.0
pair_style lj/charmm/coul/msm 8.0 10.0 9.0
pair_coeff * * 100.0 2.0
pair_coeff 1 1 100.0 2.0 150.0 3.5
```

Description:

The *lj/charmm* styles compute LJ and Coulombic interactions with an additional switching function S(r) that ramps the energy and force smoothly to zero between an inner and outer cutoff. It is a widely used potential in the [CHARMM](#) MD code. See [(MacKerell)](#) for a description of the CHARMM force field.

$$
\begin{aligned}
E &= LJ(r) & r &< r_{\text{in}} \\
&= S(r) * LJ(r) & r_{\text{in}} &< r < r_{\text{out}} \\
&= 0 & r &> r_{\text{out}} \\
E &= C(r) & r &< r_{\text{in}} \\
&= S(r) * C(r) & r_{\text{in}} &< r < r_{\text{out}} \\
&= 0 & r &> r_{\text{out}} \\
LJ(r) &= 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \\
C(r) &= \frac{C q_i q_j}{\epsilon r} \\
S(r) &= \frac{\left[r_{\text{out}}^2 - r^2 \right]^2 \left[r_{\text{out}}^2 + 2r^2 - 3r_{\text{in}}^2 \right]}{\left[r_{\text{out}}^2 - r_{\text{in}}^2 \right]^3}
\end{aligned}
$$

Both the LJ and Coulombic terms require an inner and outer cutoff. They can be the same for both formulas or different depending on whether 2 or 4 arguments are used in the pair_style command. In each case, the inner cutoff distance must be less than the outer cutoff. It it typical to make the difference between the 2 cutoffs about 1.0 Angstrom.

Style *lj/charmm/coul/charmm/implicit* computes the same formulas as style *lj/charmm/coul/charmm* except that an additional 1/r term is included in the Coulombic formula. The Coulombic energy thus varies as 1/r^2. This is effectively a distance-dependent dielectric term which is a simple model for an implicit solvent with additional screening. It is designed for use in a simulation of an unsolvated biomolecule (no explicit water

pair_style lj/charmm/coul/msm/omp command   750

molecules).

Styles *lj/charmm/coul/long* and *lj/charmm/coul/msm* compute the same formulas as style *lj/charmm/coul/charmm* except that an additional damping factor is applied to the Coulombic term, as described for the lj/cut pair styles. Only one Coulombic cutoff is specified for *lj/charmm/coul/long* and *lj/charmm/coul/msm*; if only 2 arguments are used in the pair_style command, then the outer LJ cutoff is used as the single Coulombic cutoff.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- epsilon_14 (energy units)
- sigma_14 (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum at $2^{(1/6)}$ sigma.

The latter 2 coefficients are optional. If they are specified, they are used in the LJ formula between 2 atoms of these types which are also first and fourth atoms in any dihedral. No cutoffs are specified because this CHARMM force field does not allow varying cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, epsilon_14, and sigma_14 coefficients for all of the lj/charmm pair styles can be mixed. The default mix value is *arithmetic* to coincide with the usual settings for the CHARMM force field. See the "pair_modify" command for details.

None of the lj/charmm pair styles support the pair_modify shift option, since the Lennard-Jones portion of the pair interaction is smoothed to 0.0 at the cutoff.

The *lj/charmm/coul/long* style supports the pair_modify table option since it can tabulate the short-range portion of the long-range Coulombic interaction.

None of the lj/charmm pair styles support the pair_modify tail option for adding long-range tail corrections to energy and pressure, since the Lennard-Jones portion of the pair interaction is smoothed to 0.0 at the cutoff.

pair_style lj/charmm/coul/msm/omp command

All of the lj/charmm pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The lj/charmm/coul/long pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. The other styles only support the *pair* keyword of run_style respa. See the run_style command for details.

Restrictions:

The *lj/charmm/coul/charmm* and *lj/charmm/coul/charmm/implicit* styles are part of the MOLECULE package. The *lj/charmm/coul/long* style is part of the KSPACE package. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info. Note that the MOLECULE and KSPACE packages are installed by default.

Related commands:

pair_coeff

Default: none

(**MacKerell**) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

pair_style lj/class2 command

pair_style lj/class2/cuda command

pair_style lj/class2/gpu command

pair_style lj/class2/omp command

pair_style lj/class2/coul/cut command

pair_style lj/class2/coul/cut/cuda command

pair_style lj/class2/coul/cut/omp command

pair_style lj/class2/coul/long command

pair_style lj/class2/coul/long/cuda command

pair_style lj/class2/coul/long/gpu command

pair_style lj/class2/coul/long/omp command

Syntax:

```
pair_style style args
```

- style = *lj/class2* or *lj/class2/coul/cut* or *lj/class2/coul/long*
- args = list of arguments for a particular style

```
lj/class2 args = cutoff
    cutoff = global cutoff for class 2 interactions (distance units)
  lj/class2/coul/cut args = cutoff (cutoff2)
    cutoff = global cutoff for class 2 (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/class2/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for class 2 (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/class2 10.0
pair_coeff * * 100.0 2.5
pair_coeff 1 2* 100.0 2.5 9.0

pair_style lj/class2/coul/cut 10.0
pair_style lj/class2/coul/cut 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
pair_coeff 1 1 100.0 3.5 9.0 9.0

pair_style lj/class2/coul/long 10.0
pair_style lj/class2/coul/long 10.0 8.0
pair_coeff * * 100.0 3.0
```

```
pair_coeff 1 1 100.0 3.5 9.0
```

Description:

The *lj/class2* styles compute a 6/9 Lennard-Jones potential given by

$$E = \epsilon \left[2 \left(\frac{\sigma}{r} \right)^9 - 3 \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

Rc is the cutoff.

The *lj/class2/coul/cut* and *lj/class2/coul/long* styles add a Coulombic term as described for the lj/cut pair styles.

See (Sun) for a description of the COMPASS class2 force field.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global class 2 and Coulombic cutoffs are used. If only one cutoff is specified, it is used as the cutoff for both class 2 and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the class 2 and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *lj/class2*, since it has no Coulombic terms.

For *lj/class2/coul/long* only the class 2 cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

If the pair_coeff command is not used to define coefficients for a particular I != J type pair, the mixing rule for epsilon and sigma for all class2 potentials is to use the *sixthpower* formulas documented by the pair_modify command. The pair_modify mix setting is thus ignored for class2 potentials for epsilon and sigma. However it is still followed for mixing the cutoff distance.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

pair_style lj/class2/coul/long/omp command   754

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/class2 pair styles can be mixed. Epsilon and sigma are always mixed with the value *sixthpower*. The cutoff distance is mixed by whatever option is set by the pair_modify command (default = geometric). See the "pair_modify" command for details.

All of the lj/class2 pair styles support the [pair_modify](#) shift option for the energy of the Lennard-Jones portion of the pair interaction.

The *lj/class2/coul/long* pair style does not support the [pair_modify](#) table option since a tabulation capability has not yet been added to this potential.

All of the lj/class2 pair styles support the [pair_modify](#) tail option for adding a long-range tail correction to the energy and pressure of the Lennard-Jones portion of the pair interaction.

All of the lj/class2 pair styles write their information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

All of the lj/class2 pair styles can only be used via the *pair* keyword of the [run_style respa](#) command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These styles are part of the CLASS2 package. They are only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#)

Default: none

(Sun) Sun, J Phys Chem B 102, 7338-7364 (1998).

pair_coeff command

Syntax:

```
pair_coeff I J args
```

- I,J = atom types (see asterisk form below)
- args = coefficients for one or more pairs of atom types

Examples:

```
pair_coeff 1 2 1.0 1.0 2.5
pair_coeff 2 * 1.0 1.0
pair_coeff 3* 1*2 1.0 1.0 2.5
pair_coeff * * 1.0 1.0
pair_coeff * * nialhjea 1 1 2
pair_coeff * 3 morse.table ENTRY1
pair_coeff 1 2 lj/cut 1.0 1.0 2.5 (for pair_style hybrid)
```

Description:

Specify the pairwise force field coefficients for one or more pairs of atom types. The number and meaning of the coefficients depends on the pair style. Pair coefficients can also be set in the data file read by the [read_data](#) command or in a restart file.

I and J can be specified in one of two ways. Explicit numeric values can be used for each, as in the 1st example above. I <= J is required. LAMMPS sets the coefficients for the symmetric J,I interaction to the same values.

A wildcard asterisk can be used in place of or in conjunction with the I,J arguments to set the coefficients for multiple pairs of atom types. This takes the form "*" or "*n" or "n*" or "m*n". If N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). Note that only type pairs with I <= J are considered; if asterisks imply type pairs where J < I, they are ignored.

Note that a pair_coeff command can override a previous setting for the same I,J pair. For example, these commands set the coeffs for all I,J pairs, then overwrite the coeffs for just the I,J = 2,3 pair:

```
pair_coeff * * 1.0 1.0 2.5
pair_coeff 2 3 2.0 1.0 1.12
```

A line in a data file that specifies pair coefficients uses the exact same format as the arguments of the pair_coeff command in an input script, with the exception of the I,J type arguments. In each line of the "Pair Coeffs" section of a data file, only a single type I is specified, which sets the coefficients for type I interacting with type I. This is because the section has exactly N lines, where N = the number of atom types. For this reason, the wild-card asterisk should also not be used as part of the I argument. Thus in a data file, the line corresponding to the 1st example above would be listed as

```
2 1.0 1.0 2.5
```

For many potentials, if coefficients for type pairs with I != J are not set explicitly by a pair_coeff command, the values are inferred from the I,I and J,J settings by mixing rules; see the [pair_modify](#) command for a discussion. Details on this option as it pertains to individual potentials are described on the doc page for the

potential.

Many pair styles, typically for many-body potentials, use tabulated potential files as input, when specifying the pair_coeff command. Potential files provided with LAMMPS are in the potentials directory of the distribution. For some potentials, such as EAM, other archives of suitable files can be found on the Web. They can be used with LAMMPS so long as they are in the format LAMMPS expects, as discussed on the individual doc pages.

When a pair_coeff command using a potential file is specified, LAMMPS looks for the potential file in 2 places. First it looks in the location specified. E.g. if the file is specified as "niu3.eam", it is looked for in the current working directory. If it is specified as "../potentials/niu3.eam", then it is looked for in the potentials directory, assuming it is a sister directory of the current working directory. If the file is not found, it is then looked for in the directory specified by the LAMMPS_POTENTIALS environment variable. Thus if this is set to the potentials directory in the LAMMPS distro, then you can use those files from anywhere on your system, without copying them into your working directory. Environment variables are set in different ways for different shells. Here are example settings for

```
csh, tcsh:
% setenv LAMMPS_POTENTIALS /path/to/lammps/potentials

bash:
% export LAMMPS_POTENTIALS=/path/to/lammps/potentials

Windows:
% set LAMMPS_POTENTIALS="C:\Path to LAMMPS\Potentials
```

Here is an alphabetic list of pair styles defined in LAMMPS. Click on the style to display the formula it computes, arguments specified in the pair_style command, and coefficients specified by the associated pair_coeff command.

Note that there are also additional pair styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the pair section of this page.

There are also additional accelerated pair styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

- pair_style hybrid - multiple styles of pairwise interactions
- pair_style hybrid/overlay - multiple styles of superposed pairwise interactions

- pair_style adp - angular dependent potential (ADP) of Mishin
- pair_style airebo - AIREBO potential of Stuart
- pair_style beck - Beck potential
- pair_style body - interactions between body particles
- pair_style bop - BOP potential of Pettifor
- pair_style born - Born-Mayer-Huggins potential
- pair_style born/coul/long - Born-Mayer-Huggins with long-range Coulombics
- pair_style born/coul/msm - Born-Mayer-Huggins with long-range MSM Coulombics
- pair_style born/coul/wolf - Born-Mayer-Huggins with Coulombics via Wolf potential
- pair_style brownian - Brownian potential for Fast Lubrication Dynamics
- pair_style brownian/poly - Brownian potential for Fast Lubrication Dynamics with polydispersity
- pair_style buck - Buckingham potential
- pair_style buck/coul/cut - Buckingham with cutoff Coulomb
- pair_style buck/coul/long - Buckingham with long-range Coulombics
- pair_style buck/coul/msm - Buckingham long-range MSM Coulombics

- pair_style buck/long/coul/long - long-range Buckingham with long-range Coulombics
- pair_style colloid - integrated colloidal potential
- pair_style comb - charge-optimized many-body (COMB) potential
- pair_style coul/cut - cutoff Coulombic potential
- pair_style coul/debye - cutoff Coulombic potential with Debye screening
- pair_style coul/dsf - Coulombics via damped shifted forces
- pair_style coul/long - long-range Coulombic potential
- pair_style coul/msm - long-range MSM Coulombics
- pair_style coul/wolf - Coulombics via Wolf potential
- pair_style dipole/cut - point dipoles with cutoff
- pair_style dpd - dissipative particle dynamics (DPD)
- pair_style dpd/tstat - DPD thermostatting
- pair_style dsmc - Direct Simulation Monte Carlo (DSMC)
- pair_style eam - embedded atom method (EAM)
- pair_style eam/alloy - alloy EAM
- pair_style eam/fs - Finnis-Sinclair EAM
- pair_style eim - embedded ion method (EIM)
- pair_style gauss - Gaussian potential
- pair_style gayberne - Gay-Berne ellipsoidal potential
- pair_style gran/hertz/history - granular potential with Hertzian interactions
- pair_style gran/hooke - granular potential with history effects
- pair_style gran/hooke/history - granular potential without history effects
- pair_style hbond/dreiding/lj - DREIDING hydrogen bonding LJ potential
- pair_style hbond/dreiding/morse - DREIDING hydrogen bonding Morse potential
- pair_style kim - interface to potentials provided by KIM project
- pair_style lcbop - long-range bond-order potential (LCBOP)
- pair_style line/lj - LJ potential between line segments
- pair_style lj/charmm/coul/charmm - CHARMM potential with cutoff Coulomb
- pair_style lj/charmm/coul/charmm/implicit - CHARMM for implicit solvent
- pair_style lj/charmm/coul/long - CHARMM with long-range Coulomb
- pair_style lj/charmm/coul/msm - CHARMM with long-range MSM Coulombics
- pair_style lj/class2 - COMPASS (class 2) force field with no Coulomb
- pair_style lj/class2/coul/cut - COMPASS with cutoff Coulomb
- pair_style lj/class2/coul/long - COMPASS with long-range Coulomb
- pair_style lj/cut - cutoff Lennard-Jones potential with no Coulomb
- pair_style lj/cut/coul/cut - LJ with cutoff Coulomb
- pair_style lj/cut/coul/debye - LJ with Debye screening added to Coulomb
- pair_style lj/cut/coul/dsf - LJ with Coulombics via damped shifted forces
- pair_style lj/cut/coul/long - LJ with long-range Coulombics
- pair_style lj/cut/coul/msm - LJ with long-range MSM Coulombics
- pair_style lj/cut/tip4p/cut - LJ with cutoff Coulomb for TIP4P water
- pair_style lj/cut/tip4p/long - LJ with long-range Coulomb for TIP4P water
- pair_style lj/expand - Lennard-Jones for variable size particles
- pair_style lj/gromacs - GROMACS-style Lennard-Jones potential
- pair_style lj/gromacs/coul/gromacs - GROMACS-style LJ and Coulombic potential
- pair_style lj/long/coul/long - long-range LJ and long-range Coulombics
- pair_style lj/long/tip4p/long - long-range LJ and long-range Coulomb for TIP4P water
- pair_style lj/smooth - smoothed Lennard-Jones potential
- pair_style lj/smooth/linear - linear smoothed Lennard-Jones potential
- pair_style lj96/cut - Lennard-Jones 9/6 potential
- pair_style lubricate - hydrodynamic lubrication forces
- pair_style lubricate/poly - hydrodynamic lubrication forces with polydispersity
- pair_style lubricateU - hydrodynamic lubrication forces for Fast Lubrication Dynamics
- pair_style lubricateU/poly - hydrodynamic lubrication forces for Fast Lubrication with polydispersity

pair_coeff command

- [pair_style meam](#) - modified embedded atom method (MEAM)
- [pair_style mie/cut](#) - Mie potential
- [pair_style morse](#) - Morse potential
- [pair_style peri/lps](#) - peridynamic LPS potential
- [pair_style peri/pmb](#) - peridynamic PMB potential
- [pair_style reax](#) - ReaxFF potential
- [pair_style rebo](#) - 2nd generation REBO potential of Brenner
- [pair_style resquared](#) - Everaers RE-Squared ellipsoidal potential
- [pair_style soft](#) - Soft (cosine) potential
- [pair_style sw](#) - Stillinger-Weber 3-body potential
- [pair_style table](#) - tabulated pair potential
- [pair_style tersoff](#) - Tersoff 3-body potential
- [pair_style tersoff/zbl](#) - Tersoff/ZBL 3-body potential
- [pair_style tri/lj](#) - LJ potential between triangles
- [pair_style yukawa](#) - Yukawa potential
- [pair_style yukawa/colloid](#) - screened Yukawa potential for finite-size particles
- [pair_style zbl](#) - Ziegler-Biersack-Littmark potential

Restrictions:

This command must come after the simulation box is defined by a [read_data](#), [read_restart](#), or [create_box](#) command.

Related commands:

[pair_style](#), [pair_modify](#), [read_data](#), [read_restart](#), [pair_write](#)

Default: none

pair_style colloid command

pair_style colloid/gpu command

pair_style colloid/omp command

Syntax:

```
pair_style colloid cutoff
```

- cutoff = global cutoff for colloidal interactions (distance units)

Examples:

```
pair_style colloid 10.0
pair_coeff * *  25 1.0 10.0 10.0
pair_coeff 1 1 144 1.0 0.0 0.0 3.0
pair_coeff 1 2  75.398 1.0 0.0 10.0 9.0
pair_coeff 2 2  39.478 1.0 10.0 10.0 25.0
```

Description:

Style *colloid* computes pairwise interactions between large colloidal particles and small solvent particles using 3 formulas. A colloidal particle has a size > sigma; a solvent particle is the usual Lennard-Jones particle of size sigma.

The colloid-colloid interaction energy is given by

$$U_A = -\frac{A_{cc}}{6}\left[\frac{2a_1a_2}{r^2-(a_1+a_2)^2}+\frac{2a_1a_2}{r^2-(a_1-a_2)^2}+\ln\left(\frac{r^2-(a_1+a_2)^2}{r^2-(a_1-a_2)^2}\right)\right]$$

$$U_R = \frac{A_{cc}}{37800}\frac{\sigma^6}{r}\left[\frac{r^2-7r(a_1+a_2)+6(a_1^2+7a_1a_2+a_2^2)}{(r-a_1-a_2)^7}\right.$$

$$+\frac{r^2+7r(a_1+a_2)+6(a_1^2+7a_1a_2+a_2^2)}{(r+a_1+a_2)^7}$$

$$-\frac{r^2+7r(a_1-a_2)+6(a_1^2-7a_1a_2+a_2^2)}{(r+a_1-a_2)^7}$$

$$\left.-\frac{r^2-7r(a_1-a_2)+6(a_1^2-7a_1a_2+a_2^2)}{(r-a_1+a_2)^7}\right]$$

$$U = U_A+U_R, \qquad r < r_c$$

where A_cc is the Hamaker constant, a1 and a2 are the radii of the two colloidal particles, and Rc is the cutoff. This equation results from describing each colloidal particle as an integrated collection of Lennard-Jones particles of size sigma and is derived in (Everaers).

The colloid-solvent interaction energy is given by

$$U = \frac{2\,a^3\,\sigma^3\,A_{cs}}{9\,(a^2-r^2)^3}\left[1-\frac{(5\,a^6+45\,a^4\,r^2+63\,a^2\,r^4+15\,r^6)\,\sigma^6}{15\,(a-r)^6\,(a+r)^6}\right], \quad r < r_c$$

where A_cs is the Hamaker constant, a is the radius of the colloidal particle, and Rc is the cutoff. This formula is derived from the colloid-colloid interaction, letting one of the particle sizes go to zero.

The solvent-solvent interaction energy is given by the usual Lennard-Jones formula

$$U = \frac{A_{ss}}{36}\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^6\right], \quad r < r_c$$

with A_ss set appropriately, which results from letting both particle sizes go to zero.

pair_style colloid/omp command   761

When used in combination with pair_style yukawa/colloid, the two terms become the so-called DLVO potential, which combines electrostatic repulsion and van der Waals attraction.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- A (energy units)
- sigma (distance units)
- d1 (distance units)
- d2 (distance units)
- cutoff (distance units)

A is the Hamaker energy prefactor and should typically be set as follows:

- A_cc = colloid/colloid = 4 pi^2 = 39.5
- A_cs = colloid/solvent = sqrt(A_cc*A_ss)
- A_ss = solvent/solvent = 144 (assuming epsilon = 1, so that 144/36 = 4)

Sigma is the size of the solvent particle or the constituent particles integrated over in the colloidal particle and should typically be set as follows:

- Sigma_cc = colloid/colloid = 1.0
- Sigma_cs = colloid/solvent = arithmetic mixing between colloid sigma and solvent sigma
- Sigma_ss = solvent/solvent = 1.0 or whatever size the solvent particle is

Thus typically Sigma_cs = 1.0, unless the solvent particle's size != 1.0.

D1 and d2 are particle diameters, so that $d1 = 2*a1$ and $d2 = 2*a2$ in the formulas above. Both d1 and d2 must be values >= 0. If $d1 > 0$ and $d2 > 0$, then the pair interacts via the colloid-colloid formula above. If $d1 = 0$ and $d2 = 0$, then the pair interacts via the solvent-solvent formula. I.e. a d value of 0 is a Lennard-Jones particle of size sigma. If either $d1 = 0$ or $d2 = 0$ and the other is larger, then the pair interacts via the colloid-solvent formula.

Note that the diameter of a particular particle type may appear in multiple pair_coeff commands, as it interacts with other particle types. You should insure the particle diameter is specified consistently each time it appears.

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used. However, you typically want different cutoffs for interactions between different particle sizes. E.g. if colloidal particles of diameter 10 are used with solvent particles of diameter 1, then a solvent-solvent cutoff of 2.5 would correspond to a colloid-colloid cutoff of 25. A good rule-of-thumb is to use a colloid-solvent cutoff that is half the big diameter + 4 times the small diameter. I.e. $9 = 5 + 4$ for the colloid-solvent cutoff in this case.

IMPORTANT NOTE: When using pair_style colloid for a mixture with 2 (or more) widely different particles sizes (e.g. sigma=10 colloids in a background sigam=1 LJ fluid), you will likely want to use these commands for efficiency: neighbor multi and communicate multi.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for

pair_style colloid/omp command

more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A, sigma, d1, and d2 coefficients and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. D1 and d2 are distance values and are mixed like sigma. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the COLLOID package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Normally, this pair style should be used with finite-size particles which have a diameter, e.g. see the atom_style sphere command. However, this is not a requirement, since the only definition of particle size is via the pair_coeff parameters for each type. In other words, the physical radius of the particle is ignored. Thus you should insure that the d1,d2 parameters you specify are consistent with the physical size of the particles of that type.

Per-particle polydispersity is not yet supported by this pair style; only per-type polydispersity is enabled via the pair_coeff parameters.

Related commands:

pair_coeff

Default: none

(Everaers) Everaers, Ejtehadi, Phys Rev E, 67, 041710 (2003).

pair_style comb command

pair_style comb/omp command

pair_style comb3 command

Syntax:

```
pair_style comb
pair_style comb3 keyword

keyword = polar
  polar value = polar_on or polar_off = whether or not to include atomic polarization
```

Examples:

```
pair_style comb
pair_coeff * * ../potentials/ffield.comb Si
pair_coeff * * ../potentials/ffield.comb Hf Si O

pair_style comb3 polar_off
pair_coeff * * ../potentials/ffield.comb3 O Cu N C O
```

Description:

Style *comb* computes the second-generation variable charge COMB (Charge-Optimized Many-Body) potential. Style *comb3* computes the third-generation COMB potential. These COMB potentials are described in (COMB) and (COMB3). Briefly, the total energy E_T of a system of atoms is given by

$$
\begin{aligned}
E_T &= \sum_i [E_i^{self}(q_i) + \sum_{j>i}[E_{ij}^{short}(r_{ij}, q_i, q_j) + E_{ij}^{Coul}(r_{ij}, q_i, q_j)] + \\
&\quad E^{polar}(q_i, r_{ij}) + E^{vdW}(r_{ij}) + E^{barr}(q_i) + E^{corr}(r_{ij}, \theta_{jik})]
\end{aligned}
$$

where E_i^{self} is the self-energy of atom *i* (including atomic ionization energies and electron affinities), E_{ij}^{short} is the bond-order potential between atoms *i* and *j*, E_{ij}^{Coul} is the Coulomb interactions, E^{polar} is the polarization term for organic systems (style *comb3* only), E^{vdW} is the van der Waals energy (style *comb3* only), E^{barr} is a charge barrier function, and E^{corr} are angular correction terms.

The COMB potentials (styles *comb* and *comb3*) are variable charge potentials. The equilibrium charge on each atom is calculated by the electronegativity equalization (QEq) method. See Rick for further details. This is implemented by the fix qeq/comb command, which should normally be specified in the input script when running a model with the COMB potential. The fix qeq/comb command has options that determine how often charge equilibration is performed, its convergence criterion, and which atoms are included in the calculation.

Only a single pair_coeff command is used with the *comb* and *comb3* styles which specifies the COMB potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the potential file in the pair_coeff command, where N is the number of LAMMPS atom types.

For example, if your LAMMPS simulation of a Si/SiO$_2$/ HfO$_2$ interface has 4 atom types, and you want the 1st and last to be Si, the 2nd to be Hf, and the 3rd to be O, and you would use the following pair_coeff command:

```
pair_coeff * * ../potentials/ffield.comb Si Hf O Si
```

The first two arguments must be * * so as to span all LAMMPS atom types. The first and last Si arguments map LAMMPS atom types 1 and 4 to the Si element in the *ffield.comb* file. The second Hf argument maps LAMMPS atom type 2 to the Hf element, and the third O argument maps LAMMPS atom type 3 to the O element in the potential file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *comb* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

For style *comb*, the provided potential file *ffield.comb* contains all currently-available 2nd generation COMB parameterizations: for Si, Cu, Hf, Ti, O, their oxides and Zr, Zn and U metals. For style *comb3*, the potential file *ffield.comb3* contains all currently-available 3rd generation COMB paramterizations: O, Cu, N, C, H, Ti and Zn. The status of the optimization of the compounds, for example Cu$_2$O, TiN and hydrocarbons, are given in the following table:

	O	Cu	N	C	H	Ti	Zn
O	F	F	F	F	F	F	F
Cu	F	F	P	F	F	P	F
N	F	P	F	M	F	P	P
C	F	F	M	F	F	M	M
H	F	F	F	F	F	M	M
Ti	F	P	P	M	M	F	P
Zn	F	F	P	M	M	P	F

F: Fully optimized

M: Only optimized for dimer molecule

P: in Progress but have it from mixing rule

For style *comb3*, in addition to ffield.comb3, a special parameter file, *lib.comb3*, that is exclusively used for C/O/H systems, will be automatically loaded if carbon atom is detected in LAMMPS input structure. Keyword *polar* indicates whether the force field includes the atomic polarization. Since the equilibration of the polarization has not yet been implemented, it can only set polar_off at present.

IMPORTANT NOTE: You can not use potential file *ffield.comb* with style *comb3*, nor file *ffield.comb3* with style *comb*.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for

pair_style comb3 command   765

more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

These pair styles does not support the pair_modify shift, table, and tail options.

These pair styles do not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style, pair_coeff, and fix qeq/comb commands in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles are part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

These pair styles requires the newton setting to be "on" for pair interactions.

The COMB potentials in the *ffield.comb* and *ffield.comb3* files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the COMB potential with any LAMMPS units, but you would need to create your own COMB potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_style, pair_coeff, fix_qeq/comb

Default: none

(**COMB**) S. R. Phillpot and S. B. Sinnott, Science 325, 1634-1635 (2009)

(**COMB3**) T. Liang, T.-R. Shan, Y.-T. Cheng, B. D. Devine, M. Noordhoek, Y. Li, Z. Lu, S. R. Phillpot, and S. B. Sinnott, Mat. Sci. & Eng: R, in press (DOI: 10.1016/j.mser.2013.07.001)

(**Rick**) S. W. Rick, S. J. Stuart, B. J. Berne, J Chem Phys 101, 6141 (1994).

pair_style comb3 command

pair_style coul/diel command

Syntax:

```
pair_style coul/diel cutoff
```

cutoff = global cutoff (distance units)

Examples:

```
pair_style coul/diel 3.5
pair_coeff 1 4 78. 1.375 0.112
```

Description:

Style *coul/diel* computes a Coulomb correction for implict solvent ion interactions in which the dielectric perimittivity is distance dependent. The dielectric permittivity epsilon_D(r) connects to limiting regimes: One limit is defined by a small dielectric permittivity (close to vacuum) at or close to contact seperation between the ions. At larger separations the dielectric permittivity reaches a bulk value used in the regular Coulomb interaction coul/long or coul/cut. The transition is modeled by a hyperbolic function which is incorporated in the Coulomb correction term for small ion separations as follows

$$
\begin{aligned}
E &= \frac{C q_i q_j}{\epsilon r}\left(\frac{\epsilon}{\epsilon_D(r)} - 1\right) \qquad r < r_c \\
\epsilon_D(r) &= \frac{5.2 + \epsilon}{2} + \frac{\epsilon - 5.2}{2}\tanh\left(\frac{r - r_{me}}{\sigma_e}\right)
\end{aligned}
$$

where r_me is the inflection point of epsilon_D(r) and sigma_e is a slope defining length scale. C is the same Coulomb conversion factor as in the pair_styles coul/cut, coul/long, and coul/debye. In this way the Coulomb interaction between ions is corrected at small distances r. The lower limit of epsilon_D(r->0)=5.2 due to dielectric saturation (Stiles) while the Coulomb interaction reaches its bulk limit by setting epsilon_D(r->\infty)=epsilon, the bulk value of the solvent which is 78 for water at 298K.

Examples of the use of this type of Coulomb interaction include implicit solvent simulations of salt ions (Lenart) and of ionic surfactants (Jusufi). Note that this potential is only reasonable for implicit solvent simulations and in combiantion with coul/cut or coul/long. It is also usually combined with gauss/cut, see (Lenart) or (Jusufi).

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- epsilon (no units)
- r_me (distance units)
- sigma_e (distance units)

The global cutoff (r_c) specified in the pair_style command is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support parameter mixing. Coefficients must be given explicitly for each type of particle pairs.

This pair style supports the pair_modify shift option for the energy of the Gauss-potential portion of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the "user-misc" package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff pair_style gauss/cut

Default: none

(Stiles) Stiles , Hubbard, and Kayser, J Chem Phys, 77, 6189 (1982).

(Lenart) Lenart , Jusufi, and Panagiotopoulos, J Chem Phys, 126, 044509 (2007).

(Jusufi) Jusufi, Hynninen, and Panagiotopoulos, J Phys Chem B, 112, 13783 (2008).

pair_style coul/cut command

pair_style coul/cut/omp command

pair_style coul/debye command

pair_style coul/debye/omp command

pair_style coul/dsf command

pair_style coul/dsf/gpu command

pair_style coul/dsf/omp command

pair_style coul/long command

pair_style coul/long/omp command

pair_style coul/long/gpu command

pair_style coul/msm command

pair_style coul/msm/omp command

pair_style coul/wolf command

pair_style coul/wolf/omp command

pair_style tip4p/cut command

pair_style tip4p/long command

pair_style tip4p/cut/omp command

pair_style tip4p/long/omp command

Syntax:

```
pair_style coul/cut cutoff
pair_style coul/debye kappa cutoff
pair_style coul/dsf alpha cutoff
pair_style coul/long cutoff
pair_style coul/long/gpu cutoff
pair_style coul/wolf alpha cutoff
pair_style tip4p/cut otype htype btype atype qdist cutoff
pair_style tip4p/long otype htype btype atype qdist cutoff
```

- cutoff = global cutoff for Coulombic interactions
- kappa = Debye length (inverse distance units)

pair_style coul/cut command 769

• alpha = damping parameter (inverse distance units)

Examples:

```
pair_style coul/cut 2.5
pair_coeff * *
pair_coeff 2 2 3.5

pair_style coul/debye 1.4 3.0
pair_coeff * *
pair_coeff 2 2 3.5

pair_style coul/dsf 0.05 10.0
pair_coeff * *

pair_style coul/long 10.0
pair_coeff * *

pair_style coul/msm 10.0
pair_coeff * *

pair_style coul/wolf 0.2 9.0
pair_coeff * *

pair_style tip4p/cut 1 2 7 8 0.15 12.0
pair_coeff * *

pair_style tip4p/long 1 2 7 8 0.15 10.0
pair_coeff * *
```

Description:

The *coul/cut* style computes the standard Coulombic interaction potential given by

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy-conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the [dielectric](#) command. The cutoff Rc truncates the interaction distance.

Style *coul/debye* adds an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{C q_i q_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

where kappa is the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

Style *coul/dsf* computes Coulombic interactions via the damped shifted force model described in [Fennell](#), given by:

pair_style tip4p/long/omp command   770

$$E = q_i q_j \left[\frac{\text{erfc}(\alpha r)}{r} - \frac{\text{erfc}(\alpha r_c)}{r_c} + \left(\frac{\text{erfc}(\alpha r_c)}{r_c^2} + \frac{2\alpha}{\sqrt{\pi}} \frac{\exp(-\alpha^2 r_c^2)}{r_c} \right) (r - r_c) \right] \qquad r$$

where *alpha* is the damping parameter and erfc() is the complementary error-function. The potential corrects issues in the Wolf model (described below) to provide consistent forces and energies (the Wolf potential is not differentiable at the cutoff) and smooth decay to zero.

Style *coul/wolf* computes Coulombic interactions via the Wolf summation method, described in Wolf, given by:

$$E_i = \frac{1}{2} \sum_{j \neq i} \frac{q_i q_j \text{erfc}(\alpha r_{ij})}{r_{ij}} + \frac{1}{2} \sum_{j \neq i} \frac{q_i q_j \text{erf}(\alpha r_{ij})}{r_{ij}} \qquad r < r_c$$

where *alpha* is the damping parameter, and erc() and erfc() are error-fuction and complementary error-function terms. This potential is essentially a short-range, spherically-truncated, charge-neutralized, shifted, pairwise *1/r* summation. With a manipulation of adding and substracting a self term (for i = j) to the first and second term on the right-hand-side, respectively, and a small enough *alpha* damping parameter, the second term shrinks and the potential becomes a rapidly-converging real-space summation. With a long enough cutoff and small enough alpha parameter, the energy and forces calcluated by the Wolf summation method approach those of the Ewald sum. So it is a means of getting effective long-range interactions with a short-range potential.

Styles *coul/long* and *coul/msm* compute the same Coulombic interactions as style *coul/cut* except that an additional damping factor is applied so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

Styles *tip4p/cut* and *tip4p/long* implement the coulomb part of the TIP4P water model of (Jorgensen), which introduces a massless site located a short distance away from the oxygen atom along the bisector of the HOH angle. The atomic types of the oxygen and hydrogen atoms, the bond and angle types for OH and HOH interactions, and the distance to the massless charge site are specified as pair_style arguments. Style *tip4p/cut* uses a global cutoff for Coulomb interactions; style *tip4p/long* is for use with a long-range Coulombic solver (Ewald or PPPM).

IMPORTANT NOTE: For each TIP4P water molecule in your system, the atom IDs for the O and 2 H atoms must be consecutive, with the O atom first. This is to enable LAMMPS to "find" the 2 H atoms associated with each O atom. For example, if the atom ID of an O atom in a TIP4P water molecule is 500, then its 2 H atoms must have IDs 501 and 502.

See the howto section for more information on how to use the TIP4P pair styles and lists of parameters to set. Note that the neighobr list cutoff for Coulomb interactions is effectively extended by a distance 2*qdist when using the TIP4P pair style, to account for the offset distance of the fictitious charges on O atoms in water molecules. Thus it is typically best in an efficiency sense to use a LJ cutoff >= Coulomb cutoff + 2*qdist, to shrink the size of the neighbor list. This leads to slightly larger cost for the long-range calculation, so you can test the trade-off for your model.

These potentials are designed to be combined with other pair potentials via the pair_style hybrid/overlay command. This is because they have no repulsive core. Hence if they are used by themselves, there will be no

pair_style tip4p/long/omp command   771

repulsion to keep two oppositely charged particles from overlapping each other.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *coul/cut* and *coul/debye*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

For *coul/long* and *coul/msm* no cutoff can be specified for an individual I,J type pair via the pair_coeff command. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the cutoff distance for the *coul/cut* style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

The pair_modify shift option is not relevant for these pair styles.

The *coul/long* style supports the pair_modify table option for tabulation of the short-range portion of the long-range Coulombic interaction.

These pair styles do not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *coul/long*, *coul/msm* and *tip4p/long* styles are part of the KSPACE package. They are only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style hybrid/overlay kspace_style

Default: none

(Wolf) D. Wolf, P. Keblinski, S. R. Phillpot, J. Eggebrecht, J Chem Phys, 110, 8254 (1999).

(Fennell) C. J. Fennell, J. D. Gezelter, J Chem Phys, 124, 234104 (2006).

LIGGGHTS Users Manual

pair_style lj/cut/dipole/cut command

pair_style lj/cut/dipole/cut/gpu command

pair_style lj/cut/dipole/cut/omp command

pair_style lj/sf/dipole/sf command

pair_style lj/sf/dipole/sf/gpu command

pair_style lj/sf/dipole/sf/omp command

pair_style lj/cut/dipole/long command

pair_style lj/long/dipole/long command

Syntax:

```
pair_style lj/cut/dipole/cut cutoff (cutoff2)
pair_style lj/sf/dipole/sf cutoff (cutoff2)
pair_style lj/cut/dipole/long cutoff (cutoff2)
pair_style lj/long/dipole/long flag_lj flag_coul cutoff (cutoff2)
```

- cutoff = global cutoff LJ (and Coulombic if only 1 arg) (distance units)
- cutoff2 = global cutoff for Coulombic and dipole (optional) (distance units)
- flag_lj = *long* or *cut* or *off*

  long = use long-range damping on dispersion 1/r^6 term
    cut = use a cutoff on dispersion 1/r^6 term
    off = omit disperion 1/r^6 term entirely
- flag_coul = *long* or *off*

  long = use long-range damping on Coulombic 1/r and point-dipole terms
    off = omit Coulombic and point-dipole terms entirely

Examples:

```
pair_style lj/cut/dipole/cut 10.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0


pair_style lj/sf/dipole/sf 9.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0


pair_style lj/cut/dipole/long 10.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0


pair_style lj/long/dipole/long long long 3.5 10.0
pair_coeff * * 1.0 1.0
pair_coeff 2 3 1.0 1.0 2.5 4.0
```

Description:

pair_style lj/cut/dipole/cut command   774

Style *lj/cut/dipole/cut* computes interactions between pairs of particles that each have a charge and/or a point dipole moment. In addition to the usual Lennard-Jones interaction between the particles (Elj) the charge-charge (Eqq), charge-dipole (Eqp), and dipole-dipole (Epp) interactions are computed by these formulas for the energy (E), force (F), and torque (T) between particles I and J.

$$E_{LJ} = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

$$E_{qq} = \frac{q_i q_j}{r}$$

$$E_{qp} = \frac{q}{r^3}(p \bullet \vec{r})$$

$$E_{pp} = \frac{1}{r^3}(\vec{p_i} \bullet \vec{p_j}) - \frac{3}{r^5}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})$$

$$F_{qq} = \frac{q_i q_j}{r^3}\vec{r}$$

$$F_{qp} = -\frac{q}{r^3}\vec{p} + \frac{3q}{r^5}(\vec{p} \bullet \vec{r})\vec{r}$$

$$F_{pp} = \frac{3}{r^5}(\vec{p_i} \bullet \vec{p_j})\vec{r} - \frac{15}{r^7}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})\vec{r} + \frac{3}{r^5}\left[(\vec{p_j} \bullet \vec{r})\vec{p_i} + (\vec{p_i} \bullet \vec{r})\vec{p_j}\right]$$

$$T_{pq} = T_{ij} = \frac{q_j}{r^3}(\vec{p_i} \times \vec{r})$$

$$T_{qp} = T_{ji} = -\frac{q_i}{r^3}(\vec{p_j} \times \vec{r})$$

$$T_{pp} = T_{ij} = -\frac{1}{r^3}(\vec{p_i} \times \vec{p_j}) + \frac{3}{r^5}(\vec{p_j} \bullet \vec{r})(\vec{p_i} \times \vec{r})$$

$$T_{pp} = T_{ji} = -\frac{1}{r^3}(\vec{p_j} \times \vec{p_i}) + \frac{3}{r^5}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \times \vec{r})$$

where qi and qj are the charges on the two particles, pi and pj are the dipole moment vectors of the two particles, r is their separation distance, and the vector r = Ri - Rj is the separation vector between the two particles. Note that Eqq and Fqq are simply Coulombic energy and force, Fij = -Fji as symmetric forces, and Tij != -Tji since the torques do not act symmetrically. These formulas are discussed in (Allen) and in (Toukmaji).

pair_style lj/long/dipole/long command   775

Style *lj/sf/dipole/sf* computes "shifted-force" interactions between pairs of particles that each have a charge and/or a point dipole moment. In general, a shifted-force potential is a (sligthly) modified potential containing extra terms that make both the energy and its derivative go to zero at the cutoff distance; this removes (cutoff-related) problems in energy conservation and any numerical instability in the equations of motion [(Allen)](). Shifted-force interactions for the Lennard-Jones (E_LJ), charge-charge (Eqq), charge-dipole (Eqp), dipole-charge (Epq) and dipole-dipole (Epp) potentials are computed by these formulas for the energy (E), force (F), and torque (T) between particles I and J:

$$E_{LJ} = 4\epsilon \left\{ \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] + \left[6\left(\frac{\sigma}{r_c}\right)^{12} - 3\left(\frac{\sigma}{r_c}\right)^{6} \right] \left(\frac{r}{r_c}\right)^{2} - 7\left(\frac{\sigma}{r_c}\right)^{12} + 4\left(\frac{\sigma}{r_c}\right)^{?}$$

$$E_{qq} = \frac{q_i q_j}{r} \left(1 - \frac{r}{r_c}\right)^{2}$$

$$E_{pq} = E_{ji} = -\frac{q}{r^3}\left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3\right](\vec{p} \bullet \vec{r})$$

$$E_{qp} = E_{ij} = \frac{q}{r^3}\left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3\right](\vec{p} \bullet \vec{r})$$

$$E_{pp} = \left[1 - 4\left(\frac{r}{r_c}\right)^3 + 3\left(\frac{r}{r_c}\right)^4\right]\left[\frac{1}{r^3}(\vec{p_i} \bullet \vec{p_j}) - \frac{3}{r^5}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})\right]$$

$$F_{LJ} = \left\{ \left[48\epsilon\left(\frac{\sigma}{r}\right)^{12} - 24\epsilon\left(\frac{\sigma}{r}\right)^6\right]\frac{1}{r^2} - \left[48\epsilon\left(\frac{\sigma}{r_c}\right)^{12} - 24\epsilon\left(\frac{\sigma}{r_c}\right)^6\right]\frac{1}{r_c^2}\right\}\vec{r}$$

$$F_{qq} = \frac{q_i q_j}{r}\left(\frac{1}{r^2} - \frac{1}{r_c^2}\right)\vec{r}$$

$$F_{pq} = F_{ij} = -\frac{3q}{r^5}\left[1 - \left(\frac{r}{r_c}\right)^2\right](\vec{p} \bullet \vec{r})\vec{r} + \frac{q}{r^3}\left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3\right]\vec{p}$$

$$F_{qp} = F_{ij} = \frac{3q}{r^5}\left[1 - \left(\frac{r}{r_c}\right)^2\right](\vec{p} \bullet \vec{r})\vec{r} - \frac{q}{r^3}\left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3\right]\vec{p}$$

$$F_{pp} = \frac{3}{r^5}\left\{\left[1 - \left(\frac{r}{r_c}\right)^4\right]\left[(\vec{p_i} \bullet \vec{p_j}) - \frac{3}{r^2}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})\right]\vec{r} + \right.$$
$$\left. \left[1 - 4\left(\frac{r}{r_c}\right)^3 + 3\left(\frac{r}{r_c}\right)^4\right]\left[(\vec{p_j} \bullet \vec{r})\vec{p_i} + (\vec{p_i} \bullet \vec{r})\vec{p_j} - \frac{2}{r^2}(\vec{p_i} \bullet \vec{r})(\vec{p_j} \bullet \vec{r})\vec{r}\right]\right\}$$

$$T_{pq} = T_{ij} = \frac{q_j}{r^3} \left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3 \right] (\vec{p_i} \times \vec{r})$$

$$T_{qp} = T_{ji} = -\frac{q_i}{r^3} \left[1 - 3\left(\frac{r}{r_c}\right)^2 + 2\left(\frac{r}{r_c}\right)^3 \right] (\vec{p_j} \times \vec{r})$$

$$T_{pp} = T_{ij} = -\frac{1}{r^3} \left[1 - 4\left(\frac{r}{r_c}\right)^3 + e3\left(\frac{r}{r_c}\right)^4 \right] (\vec{p_i} \times \vec{p_j}) +$$

$$\frac{3}{r^5} \left[1 - 4\left(\frac{r}{r_c}\right)^3 + 3\left(\frac{r}{r_c}\right)^4 \right] (\vec{p_j} \bullet \vec{r})(\vec{p_i} \times \vec{r})$$

$$T_{pp} = T_{ji} = -\frac{1}{r^3} \left[1 - 4\left(\frac{r}{r_c}\right)^3 + 3\left(\frac{r}{r_c}\right)^4 \right] (\vec{p_j} \times \vec{p_i}) +$$

$$\frac{3}{r^5} \left[1 - 4\left(\frac{r}{r_c}\right)^3 + 3\left(\frac{r}{r_c}\right)^4 \right] (\vec{p_i} \bullet \vec{r})(\vec{p_j} \times \vec{r})$$

where epsilon and sigma are the standard LJ parameters, r_c is the cutoff, qi and qj are the charges on the two particles, pi and pj are the dipole moment vectors of the two particles, r is their separation distance, and the vector r = Ri - Rj is the separation vector between the two particles. Note that Eqq and Fqq are simply Coulombic energy and force, Fij = -Fji as symmetric forces, and Tij != -Tji since the torques do not act symmetrically. The shifted-force formula for the Lennard-Jones potential is reported in (Stoddard). The original (unshifted) formulas for the electrostatic potentials, forces and torques can be found in (Price). The shifted-force electrostatic potentials have been obtained by applying equation 5.13 of (Allen). The formulas for the corresponding forces and torques have been obtained by applying the 'chain rule' as in appendix C.3 of (Allen).

If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic (q,p) terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic (q,p) terms respectively.

Style *lj/cut/dipole/long* computes long-range point-dipole interactions as discussed in (Toukmaji). Dipole-dipole, dipole-charge, and charge-charge interactions are all supported, along with the standard 12/6 Lennard-Jones interactions, which are computed with a cutoff. A kspace_style must be defined to use this pair style. Currently, only kspace_style ewald/disp support long-range point-dipole interactions.

Style *lj/long/dipole/long* also computes point-dipole interactions as discussed in (Toukmaji). Long-range dipole-dipole, dipole-charge, and charge-charge interactions are all supported, along with the standard 12/6 Lennard-Jones interactions. LJ interactions can be cutoff or long-ranged.

For style *lj/long/dipole/long*, if *flag_lj* is set to *long*, no cutoff is used on the LJ 1/r^6 dispersion term. The long-range portion is calculated by using the kspace_style ewald_disp command. The specified LJ cutoff then determines which portion of the LJ interactions are computed directly by the pair potential versus which part is computed in reciprocal space via the Kspace style. If *flag_lj* is set to *cut*, the LJ interactions are simply cutoff, as with pair_style lj/cut. If *flag_lj* is set to *off*, LJ interactions are not computed at all.

If *flag_coul* is set to *long*, no cutoff is used on the Coulombic or dipole interactions. The long-range portion is calculated by using *ewald_disp* of the kspace_style command. If *flag_coul* is set to *off*, Coulombic and dipole interactions are not computed at all.

Atoms with dipole moments should be integrated using the fix nve/sphere update dipole command to rotate the dipole moments. The *omega* option on the fix langevin command can be used to thermostat the rotational motion. The compute temp/sphere command can be used to monitor the temperature, since it includes rotational degrees of freedom. The atom_style dipole command should be used since it defines the point dipoles and their rotational state. The magnitude of the dipole moment for each type of particle can be defined by the dipole command or in the "Dipoles" section of the data file read in by the read_data command. Their initial orientation can be defined by the set dipole command or in the "Atoms" section of the data file.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

For atom type pairs I,J and I != J, the A, sigma, d1, and d2 coefficients and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. D1 and d2 are distance values and are mixed like sigma. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction; such energy goes to zero at the cutoff by construction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *lj/cut/dipole/cut*, *lj/cut/dipole/long*, and *lj/long/dipole/long* styles are part of the DIPOLE package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The *lj/sf/dipole/sf* style is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Using dipole pair styles with *electron* units is not currently supported.

Related commands:

pair_coeff

Default: none

(Allen) Allen and Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.

(Toukmaji) Toukmaji, Sagui, Board, and Darden, J Chem Phys, 113, 10913 (2000).

(Stoddard) Stoddard and Ford, Phys Rev A, 8, 1504 (1973).

(Price) Price, Stone and Alderton, Mol Phys, 52, 987 (1984).

pair_style dpd command

pair_style dpd/omp command

pair_style dpd/tstat command

pair_style dpd/tstat/omp command

Syntax:

```
pair_style dpd T cutoff seed
pair_style dpd/tstat Tstart Tstop cutoff seed
```

- T = temperature (temperature units)
- Tstart,Tstop = desired temperature at start/end of run (temperature units)
- cutoff = global cutoff for DPD interactions (distance units)
- seed = random # seed (positive integer)

Examples:

```
pair_style dpd 1.0 2.5 34387
pair_coeff * * 3.0 1.0
pair_coeff 1 1 3.0 1.0 1.0

pair_style dpd/tstat 1.0 1.0 2.5 34387
pair_coeff * * 1.0
pair_coeff 1 1 1.0 1.0
```

Description:

Style *dpd* computes a force field for dissipative particle dynamics (DPD) following the exposition in (Groot).

Style *dpd/tstat* invokes a DPD thermostat on pairwise interactions, which is equivalent to the non-conservative portion of the DPD force field. This pair-wise thermostat can be used in conjunction with any pair style, and in leiu of per-particle thermostats like fix langevin or ensemble thermostats like Nose Hoover as implemented by fix nvt. To use *dpd/stat* as a thermostat for another pair style, use the pair_style hybrid/overlay command to compute both the desired pair interaction and the thermostat for each pair of particles.

For style *dpd*, the force on atom I due to atom J is given as a sum of 3 terms

$$
\begin{aligned}
\vec{f} &= (F^C + F^D + F^R)\hat{r_{ij}} & r < r_c \\
F^C &= A w(r) \\
F^D &= -\gamma w^2(r)(\hat{r_{ij}} \bullet \vec{v_{ij}}) \\
F^R &= \sigma w(r) \alpha (\Delta t)^{-1/2} \\
w(r) &= 1 - r/r_c
\end{aligned}
$$

where Fc is a conservative force, Fd is a dissipative force, and Fr is a random force. Rij is a unit vector in the direction Ri - Rj, Vij is the vector difference in velocities of the two atoms = Vi - Vj, alpha is a Gaussian random number with zero mean and unit variance, dt is the timestep size, and w(r) is a weighting factor that varies between 0 and 1. Rc is the cutoff. Sigma is set equal to sqrt(2 Kb T gamma), where Kb is the Boltzmann constant and T is the temperature parameter in the pair_style command.

For style *dpd/tstat*, the force on atom I due to atom J is the same as the above equation, except that the conservative Fc term is dropped. Also, during the run, T is set each timestep to a ramped value from Tstart to Tstop.

For style *dpd*, the pairwise energy associated with style *dpd* is only due to the conservative force term Fc, and is shifted to be zero at the cutoff distance Rc. The pairwise virial is calculated using all 3 terms. For style *dpd/tstat* there is no pairwise energy, but the last two terms of the formula make a contribution to the virial.

For style *dpd*, the following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands:

- A (force units)
- gamma (force/velocity units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global DPD cutoff is used. Note that sigma is set equal to sqrt(2 T gamma), where T is the temperature set by the pair_style command so it does not need to be specified.

For style *dpd/tstat*, the coefficiencts defined for each pair of atoms types via the pair_coeff command is the same, except that A is not included.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These pair styles do not support the pair_modify shift option for the energy of the pair interaction. Note that as discussed above, the energy due to the conservative Fc term is already shifted to be 0.0 at the cutoff distance Rc.

The pair_modify table option is not relevant for these pair styles.

These pair style do not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

These pair styles writes their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file. Note that the user-specified random number seed is stored in the restart file, so when a simulation is restarted, each processor will re-initialize its random number generator the same way it did initially. This means the random forces will be random, but will not be the same as they would have been if the original simulation had continued past the restart time.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

The *dpd/tstat* style can ramp its target temperature over multiple runs, using the *start* and *stop* keywords of the [run](run) command. See the [run](run) command for details of how to do this.

Restrictions:

The default frequency for rebuilding neighbor lists is every 10 steps (see the neigh_modify command). This may be too infrequent for style *dpd* simulations since particles move rapidly and can overlap by large amounts. If this setting yields a non-zero number of "dangerous" reneighborings (printed at the end of a simulation), you should experiment with forcing reneighboring more often and see if system energies/trajectories change.

These pair styles requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

These pair styles will not restart exactly when using the read_restart command, though they should provide statistically similar results. This is because the forces they compute depend on atom velocities. See the read_restart command for more details.

Related commands:

pair_coeff, [fix nvt](fix nvt), fix langevin

Default: none

(Groot) Groot and Warren, J Chem Phys, 107, 4423-35 (1997).

pair_style dsmc command

Syntax:

```
pair_style dsmc max_cell_size seed weighting Tref Nrecompute Nsample
```

- max_cell_size = global maximum cell size for DSMC interactions (distance units)
- seed = random # seed (positive integer)
- weighting = macroparticle weighting
- Tref = reference temperature (temperature units)
- Nrecompute = recompute v*sigma_max every this many timesteps (timesteps)
- Nsample = sample this many times in recomputing v*sigma_max

Examples:

```
pair_style dsmc 2.5 34387 10 1.0 100 20
pair_coeff * * 1.0
pair_coeff 1 1 1.0
```

Description:

Style *dsmc* computes collisions between pairs of particles for a direct simulation Monte Carlo (DSMC) model following the exposition in [(Bird)](#). Each collision resets the velocities of the two particles involved. The number of pairwise collisions for each pair or particle types and the length scale within which they occur are determined by the parameters of the pair_style and pair_coeff commands.

Stochastic collisions are performed using the variable hard sphere (VHS) approach, with the user-defined *max_cell_size* value used as the maximum DSMC cell size, and reference cross-sections for collisions given using the pair_coeff command.

There is no pairwise energy or virial contributions associated with this pair style.

The following coefficient must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- sigma (area units, i.e. distance-squared)

The global DSMC *max_cell_size* determines the maximum cell length used in the DSMC calculation. A structured mesh is overlayed on the simulation box such that an integer number of cells are created in each direction for each processor's sub-domain. Cell lengths are adjusted up to the user-specified maximum cell size.

To perform a DSMC simulation with LAMMPS, several additional options should be set in your input script, though LAMMPS does not check for these settings.

Since this pair style does not compute particle forces, you should use the "fix nve/noforce" time integration fix for the DSMC particles, e.g.

```
fix 1 all nve/noforce
```

This pair style assumes that all particles will communicated to neighboring processors every timestep as they move. This makes it possible to perform all collisions between pairs of particles that are on the same processor. To ensure this occurs, you should use these commands:

```
neighbor 0.0 bin
neigh_modify every 1 delay 0 check no
atom_modify sort 0 0.0
communicate single cutoff 0.0
```

These commands ensure that LAMMPS communicates particles to neighboring processors every timestep and that no ghost atoms are created. The output statistics for a simulation run should indicate there are no ghost particles or neighbors.

In order to get correct DSMC collision statistics, users should specify a Gaussian velocity distribution when populating the simulation domain. Note that the default velocity distribution is uniform, which will not give good DSMC collision rates. Specify "dist gaussian" when using the velocity command as in the following:

```
velocity all create 594.6 87287 loop geom dist gaussian
```

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file. Note that the user-specified random number seed is stored in the restart file, so when a simulation is restarted, each processor will re-initialize its random number generator the same way it did initially. This means the random forces will be random, but will not be the same as they would have been if the original simulation had continued past the restart time.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the MC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, fix nve/noforce, neigh_modify, neighbor, communicate

Default: none

(**Bird**) G. A. Bird, "Molecular Gas Dynamics and the Direct Simulation of Gas Flows" (1994).

pair_style eam command

pair_style eam/cuda command

pair_style eam/gpu command

pair_style eam/omp command

pair_style eam/opt command

pair_style eam/alloy command

pair_style eam/alloy/cuda command

pair_style eam/alloy/gpu command

pair_style eam/alloy/omp command

pair_style eam/alloy/opt command

pair_style eam/cd command

pair_style eam/cd/omp command

pair_style eam/fs command

pair_style eam/fs/cuda command

pair_style eam/fs/gpu command

pair_style eam/fs/omp command

pair_style eam/fs/opt command

Syntax:

```
pair_style style
```

- style = *eam* or *eam/alloy* or *eam/cd* or *eam/fs*

Examples:

```
pair_style eam
pair_coeff * * cuu3
pair_coeff 1*3 1*3 niu3.eam

pair_style eam/alloy
pair_coeff * * ../potentials/NiAlH_jea.eam.alloy Ni Al Ni Ni
```

pair_style eam command   786

```
pair_style eam/cd
pair_coeff * * ../potentials/FeCr.cdeam Fe Cr

pair_style eam/fs
pair_coeff * * NiAlH_jea.eam.fs Ni Al Ni Ni
```

Description:

Style *eam* computes pairwise interactions for metals and metal alloys using embedded-atom method (EAM) potentials [(Daw)](). The total energy Ei of an atom I is given by

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_\beta(r_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

where F is the embedding energy which is a function of the atomic electron density rho, phi is a pair potential interaction, and alpha and beta are the element types of atoms I and J. The multi-body nature of the EAM potential is a result of the embedding energy term. Both summations in the formula are over all neighbors J of atom I within the cutoff distance.

The cutoff distance and the tabulated values of the functionals F, rho, and phi are listed in one or more files which are specified by the [pair_coeff]() command. These are ASCII text files in a DYNAMO-style format which is described below. DYNAMO was the original serial EAM MD code, written by the EAM originators. Several DYNAMO potential files for different metals are included in the "potentials" directory of the LAMMPS distribution. All of these files are parameterized in terms of LAMMPS [metal units]().

IMPORTANT NOTE: The *eam* style reads single-element EAM potentials in the DYNAMO *funcfl* format. Either single element or alloy systems can be modeled using multiple *funcfl* files and style *eam*. For the alloy case LAMMPS mixes the single-element potentials to produce alloy potentials, the same way that DYNAMO does. Alternatively, a single DYNAMO *setfl* file or Finnis/Sinclair EAM file can be used by LAMMPS to model alloy systems by invoking the *eam/alloy* or *eam/cd* or *eam/fs* styles as described below. These files require no mixing since they specify alloy interactions explicitly.

Note that unlike for other potentials, cutoffs for EAM potentials are not set in the pair_style or pair_coeff command; they are specified in the EAM potential files themselves. Likewise, the EAM potential files list atomic masses; thus you do not need to use the [mass]() command to specify them.

There are several WWW sites that distribute and document EAM potentials stored in DYNAMO or other formats:

```
http://www.ctcms.nist.gov/potentials
http://cst-www.nrl.navy.mil/ccm6/ap
http://enpub.fulton.asu.edu/cms/potentials/main/main.htm
```

These potentials should be usable with LAMMPS, though the alternate formats would need to be converted to the DYNAMO format used by LAMMPS and described on this page. The NIST site is maintained by Chandler Becker (cbecker at nist.gov) who is good resource for info on interatomic potentials and file formats.

For style *eam*, potential values are read from a file that is in the DYNAMO single-element *funcfl* format. If the DYNAMO file was created by a Fortran program, it cannot have "D" values in it for exponents. C only recognizes "e" or "E" for scientific notation.

pair_style eam/fs/opt command 787

Note that unlike for other potentials, cutoffs for EAM potentials are not set in the pair_style or pair_coeff command; they are specified in the EAM potential files themselves.

For style *eam* a potential file must be assigned to each I,I pair of atom types by using one or more pair_coeff commands, each with a single argument:

- filename

Thus the following command

```
pair_coeff *2 1*2 cuu3.eam
```

will read the cuu3 potential file and use the tabulated Cu values for F, phi, rho that it contains for type pairs 1,1 and 2,2 (type pairs 1,2 and 2,1 are ignored). See the pair_coeff doc page for alternate ways to specify the path for the potential file. In effect, this makes atom types 1 and 2 in LAMMPS be Cu atoms. Different single-element files can be assigned to different atom types to model an alloy system. The mixing to create alloy potentials for type pairs with I != J is done automatically the same way that the serial DYNAMO code originally did it; you do not need to specify coefficients for these type pairs.

Funcfl files in the *potentials* directory of the LAMMPS distribution have an ".eam" suffix. A DYNAMO single-element *funcfl* file is formatted as follows:

- line 1: comment (ignored)
- line 2: atomic number, mass, lattice constant, lattice type (e.g. FCC)
- line 3: Nrho, drho, Nr, dr, cutoff

On line 2, all values but the mass are ignored by LAMMPS. The mass is in mass units, e.g. mass number or grams/mole for metal units. The cubic lattice constant is in Angstroms. On line 3, Nrho and Nr are the number of tabulated values in the subsequent arrays, drho and dr are the spacing in density and distance space for the values in those arrays, and the specified cutoff becomes the pairwise cutoff used by LAMMPS for the potential. The units of dr are Angstroms; I'm not sure of the units for drho - some measure of electron density.

Following the three header lines are three arrays of tabulated values:

- embedding function F(rho) (Nrho values)
- effective charge function Z(r) (Nr values)
- density function rho(r) (Nr values)

The values for each array can be listed as multiple values per line, so long as each array starts on a new line. For example, the individual Z(r) values are for $r = 0, dr, 2*dr, ... (Nr-1)*dr$.

The units for the embedding function F are eV. The units for the density function rho are the same as for drho (see above, electron density). The units for the effective charge Z are "atomic charge" or sqrt(Hartree * Bohr-radii). For two interacting atoms i,j this is used by LAMMPS to compute the pair potential term in the EAM energy expression as r*phi, in units of eV-Angstroms, via the formula

```
r*phi = 27.2 * 0.529 * Zi * Zj
```

where 1 Hartree = 27.2 eV and 1 Bohr = 0.529 Angstroms.

Style *eam/alloy* computes pairwise interactions using the same formula as style *eam*. However the associated pair_coeff command reads a DYNAMO *setfl* file instead of a *funcfl* file. *Setfl* files can be used to model a single-element or alloy system. In the alloy case, as explained above, *setfl* files contain explicit tabulated values for alloy interactions. Thus they allow more generality than *funcfl* files for modeling alloys.

For style *eam/alloy*, potential values are read from a file that is in the DYNAMO multi-element *setfl* format, except that element names (Ni, Cu, etc) are added to one of the lines in the file. If the DYNAMO file was created by a Fortran program, it cannot have "D" values in it for exponents. C only recognizes "e" or "E" for scientific notation.

Only a single pair_coeff command is used with the *eam/alloy* style which specifies a DYNAMO *setfl* file, which contains information for M elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of *setfl* elements to atom types

As an example, the potentials/NiAlH_jea.eam.alloy file is a *setfl* file which has tabulated EAM values for 3 elements and their alloy interactions: Ni, Al, and H. See the [pair_coeff](#) doc page for alternate ways to specify the path for the potential file. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Ni, and the 4th to be Al, you would use the following pair_coeff command:

```
pair_coeff * * NiAlH_jea.eam.alloy Ni Ni Ni Al
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Ni arguments map LAMMPS atom types 1,2,3 to the Ni element in the *setfl* file. The final Al argument maps LAMMPS atom type 4 to the Al element in the *setfl* file. Note that there is no requirement that your simulation use all the elements specified by the *setfl* file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *eam/alloy* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Setfl files in the *potentials* directory of the LAMMPS distribution have an ".eam.alloy" suffix. A DYNAMO multi-element *setfl* file is formatted as follows:

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

In a DYNAMO *setfl* file, line 4 only lists Nelements = the # of elements in the *setfl* file. For LAMMPS, the element name (Ni, Cu, etc) of each element must be added to the line, in the order the elements appear in the file.

The meaning and units of the values in line 5 is the same as for the *funcfl* file described above. Note that the cutoff (in Angstroms) is a global value, valid for all pairwise interactions for all element pairings.

Following the 5 header lines are Nelements sections, one for each element, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) (Nr values)

As with the *funcfl* files, only the mass (in mass [units](#), e.g. mass number or grams/mole for metal units) is used by LAMMPS from the 1st line. The cubic lattice constant is in Angstroms. The F and rho arrays are unique to a single element and have the same format and units as in a *funcfl* file.

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed for all i,j element pairs in the same format as other arrays. Since these interactions are symmetric (i,j = j,i) only phi arrays with i

>= j are listed, in the following order: i,j = (1,1), (2,1), (2,2), (3,1), (3,2), (3,3), (4,1), ..., (Nelements, Nelements). Unlike the effective charge array Z(r) in *funcfl* files, the tabulated values for each phi function are listed in *setfl* files directly as r*phi (in units of eV-Angstroms), since they are for atom pairs.

Style *eam/cd* is similar to the *eam/alloy* style, except that it computes alloy pairwise interactions using the concentration-dependent embedded-atom method (CD-EAM). This model can reproduce the enthalpy of mixing of alloys over the full composition range, as described in (Stukowski).

The pair_coeff command is specified the same as for the *eam/alloy* style. However the DYNAMO *setfl* file must has two lines added to it, at the end of the file:

- line 1: Comment line (ignored)
- line 2: N Coefficient0 Coefficient1 ... CoeffincientN

The last line begins with the degree *N* of the polynomial function *h(x)* that modifies the cross interaction between A and B elements. Then *N+1* coefficients for the terms of the polynomial are then listed.

Modified EAM *setfl* files used with the *eam/cd* style must contain exactly two elements, i.e. in the current implementation the *eam/cd* style only supports binary alloys. The first and second elements in the input EAM file are always taken as the *A* and *B* species.

CD-EAM files in the *potentials* directory of the LAMMPS distribution have a ".cdeam" suffix.

Style *eam/fs* computes pairwise interactions for metals and metal alloys using a generalized form of EAM potentials due to Finnis and Sinclair (Finnis). The total energy Ei of an atom I is given by

$$E_i = F_\alpha \left(\sum_{j \neq i} \rho_{\alpha\beta}(r_{ij}) \right) + \frac{1}{2} \sum_{j \neq i} \phi_{\alpha\beta}(r_{ij})$$

This has the same form as the EAM formula above, except that rho is now a functional specific to the atomic types of both atoms I and J, so that different elements can contribute differently to the total electron density at an atomic site depending on the identity of the element at that atomic site.

The associated pair_coeff command for style *eam/fs* reads a DYNAMO *setfl* file that has been extended to include additional rho_alpha_beta arrays of tabulated values. A discussion of how FS EAM differs from conventional EAM alloy potentials is given in (Ackland1). An example of such a potential is the same author's Fe-P FS potential (Ackland2). Note that while FS potentials always specify the embedding energy with a square root dependence on the total density, the implementation in LAMMPS does not require that; the user can tabulate any functional form desired in the FS potential files.

For style *eam/fs*, the form of the pair_coeff command is exactly the same as for style *eam/alloy*, e.g.

```
pair_coeff * * NiAlH_jea.eam.fs Ni Ni Ni Al
```

where there are N additional arguments after the filename, where N is the number of LAMMPS atom types. See the pair_coeff doc page for alternate ways to specify the path for the potential file. The N values determine the mapping of LAMMPS atom types to EAM elements in the file, as described above for style *eam/alloy*. As with *eam/alloy*, if a mapping value is NULL, the mapping is not performed. This can be used when an *eam/fs* potential is used as part of the *hybrid* pair style. The NULL values are used as placeholders for atom types that will be used with other potentials.

pair_style eam/fs/opt command   790

FS EAM files include more information than the DYNAMO *setfl* format files read by *eam/alloy*, in that i,j density functionals for all pairs of elements are included as needed by the Finnis/Sinclair formulation of the EAM.

FS EAM files in the *potentials* directory of the LAMMPS distribution have an ".eam.fs" suffix. They are formatted as follows:

- lines 1,2,3 = comments (ignored)
- line 4: Nelements Element1 Element2 ... ElementN
- line 5: Nrho, drho, Nr, dr, cutoff

The 5-line header section is identical to an EAM *setfl* file.

Following the header are Nelements sections, one for each element I, each with the following format:

- line 1 = atomic number, mass, lattice constant, lattice type (e.g. FCC)
- embedding function F(rho) (Nrho values)
- density function rho(r) for element I at element 1 (Nr values)
- density function rho(r) for element I at element 2
- ...
- density function rho(r) for element I at element Nelement

The units of these quantities in line 1 are the same as for *setfl* files. Note that the rho(r) arrays in Finnis/Sinclair can be asymmetric (i,j != j,i) so there are Nelements^2 of them listed in the file.

Following the Nelements sections, Nr values for each pair potential phi(r) array are listed in the same manner (r*phi, units of eV-Angstroms) as in EAM *setfl* files. Note that in Finnis/Sinclair, the phi(r) arrays are still symmetric, so only phi arrays for i >= j are listed.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accerlate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above with the individual styles. You never need to specify a pair_coeff command with I != J arguments for the eam styles.

This pair style does not support the pair_modify shift, table, and tail options.

The eam pair styles do not write their information to binary restart files, since it is stored in tabulated potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a

restart file.

The eam pair styles can only be used via the *pair* keyword of the [run_style respa](#) command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

All of these styles except the *eam/cd* style are part of the MANYBODY package. They are only enabled if LAMMPS was built with that package (which it is by default). See the [Making LAMMPS](#) section for more info.

The *eam/cd* style is part of the USER-MISC package and also requires the MANYBODY package. It is only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#)

Default: none

(Ackland1) Ackland, Condensed Matter (2005).

(Ackland2) Ackland, Mendelev, Srolovitz, Han and Barashev, Journal of Physics: Condensed Matter, 16, S2629 (2004).

(Daw) Daw, Baskes, Phys Rev Lett, 50, 1285 (1983). Daw, Baskes, Phys Rev B, 29, 6443 (1984).

(Finnis) Finnis, Sinclair, Philosophical Magazine A, 50, 45 (1984).

(Stukowski) Stukowski, Sadigh, Erhart, Caro; Modeling Simulation Materials Science & Engineering, 7, 075005 (2009).

pair_style edip command

Syntax:

```
pair_style edip
```

```
pair_style edip/omp
```

Examples:

pair_style edip pair_coeff * * Si.edip Si

Description:

The *edip* style computes a 3-body EDIP potential which is popular for modeling silicon materials where it can have advantages over other models such as the Stillinger-Weber or Tersoff potentials. In EDIP, the energy E of a system of atoms is

$$E = \sum_{j \neq i} \phi_2(R_{ij}, Z_i) + \sum_{j \neq i} \sum_{k \neq i, k > j} \phi_3(R_{ij}, R_{ik}, Z_i)$$

$$\phi_2(r, Z) = A \left[\left(\frac{B}{r} \right)^\rho - e^{-\beta Z^2} \right] exp \left(\frac{\sigma}{r - a} \right)$$

$$\phi_3(R_{ij}, R_{ik}, Z_i) = exp \left(\frac{\gamma}{R_{ij} - a} \right) exp \left(\frac{\gamma}{R_{ik} - a} \right) h(cos\theta_{ijk}, Z_i)$$

$$Z_i = \sum_{m \neq i} f(R_{im}) \qquad f(r) = \begin{cases} 1 & r < c \\ exp \left(\frac{\alpha}{1 - x^{-3}} \right) & c < r < a \\ 0 & r > a \end{cases}$$

$$h(l, Z) = \lambda [(1 - e^{-Q(Z)(l + \tau(Z))^2}) + \eta Q(Z)(l + \tau(Z))^2]$$

$$Q(Z) = Q_0 e^{-\mu Z} \qquad \tau(Z) = u_1 + u_2(u_3 e^{-u_4 Z} - e^{-2u_4 Z})$$

where phi2 is a two-body term and phi3 is a three-body term. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = a. Both terms depend on the local environment of atom I through its effective coordination number defined by Z, which is unity for a cutoff distance < c and gently goes to 0 at distance = a.

Only a single pair_coeff command is used with the *edip* style which specifies a EDIP potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename

- N element names = mapping of EDIP elements to atom types

See the <u>pair_coeff</u> doc page for alternate ways to specify the path for the potential file.

As an example, imagine a file Si.edip has EDIP values for Si.

EDIP files in the *potentials* directory of the LAMMPS distribution have a ".edip" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to the two-body and three-body coefficients in the formula above:

- element 1 (the center atom in a 3-body interaction)
- element 2
- element 3
- A (energy units)
- B (distance units)
- cutoffA (distance units)
- cutoffC (distance units)
- alpha
- beta
- eta
- gamma (distance units)
- lambda (energy units)
- mu
- tho
- sigma (distance units)
- Q0
- u1
- u2
- u3
- u4

The A, B, beta, sigma parameters are used only for two-body interactions. The eta, gamma, lambda, mu, Q0 and all u1 to u4 parameters are used only for three-body interactions. The alpha and cutoffC parameters are used for the coordination environment function only.

The EDIP potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single-element simulation, only a single entry is required (e.g. SiSiSi). For a two-element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify EDIP parameters for all permutations of the two elements interacting in three-body configurations. Thus for 3 elements, 27 entries would be required, etc.

At the moment, only a single element parametrization is implemented. However, the author is not aware of other multi-element EDIP parametrizations. If you know any and you are interest in that, please contact the author of the EDIP package.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in <u>Section_accelerate</u> of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively.

pair_style edip command

They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the [pair_modify](#) shift, table, and tail options.

This pair style does not write its information to [binary restart files](#), since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This angle style can only be used if LAMMPS was built with the USER-MISC package. See the [Making LAMMPS](#) section for more info on packages.

This pair style requires the [newton](#) setting to be "on" for pair interactions.

The EDIP potential files provided with LAMMPS (see the potentials directory) are parameterized for metal [units](#). You can use the SW potential with any LAMMPS units, but you would need to create your own EDIP potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

[pair_coeff](#)

Default: none

(EDIP) J. F. Justo et al., Phys. Rev. B 58, 2539 (1998).

pair_style eff/cut command

Syntax:

```
pair_style eff/cut cutoff keyword args ...
```

- cutoff = global cutoff for Coulombic interactions
- zero or more keyword/value pairs may be appended

```
keyword = limit/eradius or pressure/evirials or ecp
  limit/eradius args = none
  pressure/evirials args = none
  ecp args = type element type element ...
    type = LAMMPS atom type (1 to Ntypes)
    element = element symbol (e.g. H, Si)
```

Examples:

```
pair_style eff/cut 39.7
pair_style eff/cut 40.0 limit/eradius
pair_style eff/cut 40.0 limit/eradius pressure/evirials
pair_style eff/cut 40.0 ecp 1 Si 3 C
pair_coeff * *
pair_coeff 2 2 20.0
pair_coeff 1 s 0.320852 2.283269 0.814857
pair_coeff 3 p 22.721015 0.728733 1.103199 17.695345 6.693621
```

Description:

This pair style contains a LAMMPS implementation of the electron Force Field (eFF) potential currently under development at Caltech, as described in (Jaramillo-Botero). The eFF for Z(Su) in 2007. It has been extended to higher Zs by using effective core potentials (ECPs) that now cover up to 2nd and 3rd row p-block elements of the periodic table.

eFF can be viewed as an approximation to QM wave packet dynamics and Fermionic molecular dynamics, combining the ability of electronic structure methods to describe atomic structure, bonding, and chemistry in materials, and of plasma methods to describe nonequilibrium dynamics of large systems with a large number of highly excited electrons. Yet, eFF relies on a simplification of the electronic wavefunction in which electrons are described as floating Gaussian wave packets whose position and size respond to the various dynamic forces between interacting classical nuclear particles and spherical Gaussian electron wavepackets. The wavefunction is taken to be a Hartree product of the wave packets. To compensate for the lack of explicit antisymmetry in the resulting wavefunction, a spin-dependent Pauli potential is included in the Hamiltonian. Substituting this wavefunction into the time-dependent Schrodinger equation produces equations of motion that correspond - to second order - to classical Hamiltonian relations between electron position and size, and their conjugate momenta. The N-electron wavefunction is described as a product of one-electron Gaussian functions, whose size is a dynamical variable and whose position is not constrained to a nuclear center. This form allows for straightforward propagation of the wavefunction, with time, using a simple formulation from which the equations of motion are then integrated with conventional MD algorithms. In addition to this spin-dependent Pauli repulsion potential term between Gaussians, eFF includes the electron kinetic energy from the Gaussians. These two terms are based on first-principles quantum mechanics. On the other hand, nuclei are described as point charges, which interact with other nuclei and electrons through standard electrostatic potential forms.

The full Hamiltonian (shown below), contains then a standard description for electrostatic interactions between a set of delocalized point and Gaussian charges which include, nuclei-nuclei (NN), electron-electron

(ee), and nuclei-electron (Ne). Thus, eFF is a mixed QM-classical mechanics method rather than a conventional force field method (in which electron motions are averaged out into ground state nuclear motions, i.e a single electronic state, and particle interactions are described via empirically parameterized interatomic potential functions). This makes eFF uniquely suited to simulate materials over a wide range of temperatures and pressures where electronically excited and ionized states of matter can occur and coexist. Furthermore, the interactions between particles -nuclei and electrons- reduce to the sum of a set of effective pairwise potentials in the eFF formulation. The *eff/cut* style computes the pairwise Coulomb interactions between nuclei and electrons (E_NN,E_Ne,E_ee), and the quantum-derived Pauli (E_PR) and Kinetic energy interactions potentials between electrons (E_KE) for a total energy expression given as,

$$U(R, r, s) = E_{NN}(R) + E_{Ne}(R, r, s) + E_{ee}(r, s) + E_{KE}(r, s) + E_{PR}(\uparrow\downarrow, S$$

The individual terms are defined as follows:

$$E_{KE} = \frac{\hbar^2}{m_e} \sum_i \frac{3}{2s_i^2}$$

$$E_{NN} = \frac{1}{4\pi\varepsilon_0} \sum_{i<j} \frac{Z_i Z_j}{R_{ij}}$$

$$E_{Ne} = -\frac{1}{4\pi\varepsilon_0} \sum_{i,j} \frac{Z_i}{R_{ij}} Erf\left(\frac{\sqrt{2}R_{ij}}{s_j}\right)$$

$$E_{ee} = \frac{1}{4\pi\varepsilon_0} \sum_{i<j} \frac{1}{r_{ij}} Erf\left(\frac{\sqrt{2}r_{ij}}{\sqrt{s_i^2 + s_j^2}}\right)$$

$$E_{Pauli} = \sum_{\sigma_i=\sigma_j} E(\uparrow\uparrow)_{ij} + \sum_{\sigma_i\neq\sigma_j} E(\uparrow\downarrow)_{ij}$$

where, s_i correspond to the electron sizes, the sigmas i's to the fixed spins of the electrons, Z_i to the charges on the nuclei, R_ij to the distances between the nuclei or the nuclei and electrons, and r_ij to the distances between electrons. For additional details see (Jaramillo-Botero).

The overall electrostatics energy is given in Hartree units of energy by default and can be modified by an

pair_style eff/cut command

energy-conversion constant, according to the units chosen (see electron_units). The cutoff Rc, given in Bohrs (by default), truncates the interaction distance. The recommended cutoff for this pair style should follow the minimum image criterion, i.e. half of the minimum unit cell length.

Style *eff/long* (not yet available) computes the same interactions as style *eff/cut* except that an additional damping factor is applied so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

This potential is designed to be used with atom_style electron definitions, in order to handle the description of systems with interacting nuclei and explicit electrons.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutoff (distance units)

For *eff/cut*, the cutoff coefficient is optional. If it is not used (as in some of the examples above), the default global value specified in the pair_style command is used.

For *eff/long* (not yet available) no cutoff will be specified for an individual I,J type pair via the pair_coeff command. All type pairs use the same global cutoff specified in the pair_style command.

The *limit/eradius* and *pressure/evirials* keywrods are optional. Neither or both must be specified. If not specified they are unset.

The *limit/eradius* keyword is used to restrain electron size from becoming excessively diffuse at very high temperatures were the Gaussian wave packet representation breaks down, and from expanding as free particles to infinite size. If unset, electron radius is free to increase without bounds. If set, a restraining harmonic potential of the form $E = 1/2k\_ss^2$ for $s > L\_box/2$, where $k\_s = 1$ Hartrees/Bohr^2, is applied on the electron radius.

The *pressure/evirials* keyword is used to control between two types of pressure computation: if unset, the computed pressure does not include the electronic radial virials contributions to the total pressure (scalar or tensor). If set, the computed pressure will include the electronic radial virial contributions to the total pressure (scalar and tensor).

The *ecp* keyword is used to associate an ECP representation for a particular atom type. The ECP captures the orbital overlap between a core pseudo particle and valence electrons within the Pauli repulsion. A list of type:element-symbol pairs may be provided for all ECP representations, after the "ecp" keyword.

IMPORTANT NOTE: Default ECP parameters are provided for C, N, O, Al, and Si. Users can modify these using the pair_coeff command as exemplified above. For this, the User must distinguish between two different functional forms supported, one that captures the orbital overlap assuming the s-type core interacts with an s-like valence electron (s-s) and another that assumes the interaction is s-p. For systems that exhibit significant p-character (e.g. C, N, O) the s-p form is recommended. The "s" ECP form requires 3 parameters and the "p" 5 parameters.

IMPORTANT NOTE: there are two different pressures that can be reported for eFF when defining this pair_style, one (default) that considers electrons do not contribute radial virial components (i.e. electrons treated as incompressible 'rigid' spheres) and one that does. The radial electronic contributions to the virials are only tallied if the flexible pressure option is set, and this will affect both global and per-atom quantities. In principle, the true pressure of a system is somewhere in between the rigid and the flexible eFF pressures, but,

for most cases, the difference between these two pressures will not be significant over long-term averaged runs (i.e. even though the energy partitioning changes, the total energy remains similar).

IMPORTANT NOTE: This implemention of eFF gives a reasonably accurate description for systems containing nuclei from Z = 1-6 in "all electron" representations. For systems with increasingly non-spherical electrons, Users should use the ECP representations. ECPs are now supported and validated for most of the 2nd and 3rd row elements of the p-block. Predefined parameters are provided for C, N, O, Al, and Si. The ECP captures the orbital overlap between the core and valence electrons (i.e. Pauli repulsion) with one of the functional forms:

$$E_{Pauli(ECP_s)} = p_1 \exp\left(-\frac{p_2 r^2}{p_3 + s^2}\right)$$

$$E_{Pauli(ECP_p)} = p_1 \left(\frac{2}{p_2/s + s/p_2}\right)(r - p_3 s)^2 \exp\left[-\frac{p_4(r - p_3 s)^2}{p_5 + s^2}\right]$$

Where the 1st form correspond to core interactions with s-type valence electrons and the 2nd to core interactions with p-type valence electrons.

The current version adds full support for models with fixed-core and ECP definitions. to enable larger timesteps (i.e. by avoiding the high frequency vibrational modes -translational and radial- of the 2 s electrons), and in the ECP case to reduce the increased orbital complexity in higher Z elements (up to Z

In general, eFF excels at computing the properties of materials in extreme conditions and tracing the system dynamics over multi-picosend timescales; this is particularly relevant where electron excitations can change significantly the nature of bonding in the system. It can capture with surprising accuracy the behavior of such systems because it describes consistently and in an unbiased manner many different kinds of bonds, including covalent, ionic, multicenter, ionic, and plasma, and how they interconvert and/or change when they become excited. eFF also excels in computing the relative thermochemistry of isodemic reactions and conformational changes, where the bonds of the reactants are of the same type as the bonds of the products. eFF assumes that kinetic energy differences dominate the overall exchange energy, which is true when the electrons present are nearly spherical and nodeless and valid for covalent compounds such as dense hydrogen, hydrocarbons, and diamond; alkali metals (e.g. lithium), alkali earth metals (e.g. beryllium) and semimetals such as boron; and various compounds containing ionic and/or multicenter bonds, such as boron dihydride.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the cutoff distance for the *eff/cut* style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

The pair_modify shift option is not relevant for these pair styles.

The *eff/long* (not yet available) style supports the pair_modify table option for tabulation of the short-range portion of the long-range Coulombic interaction.

These pair styles do not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

pair_style eff/cut command   799

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These pair styles will only be enabled if LAMMPS is built with the USER-EFF package. It will only be enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

These pair styles require that particles store electron attributes such as radius, radial velocity, and radital force, as defined by the atom_style. The *electron* atom style does all of this.

Thes pair styles require you to use the communicate vel yes option so that velocites are stored by ghost atoms.

Related commands:

pair_coeff

Default:

If not specified, limit_eradius = 0 and pressure_with_evirials = 0.

(Su) Su and Goddard, Excited Electron Dynamics Modeling of Warm Dense Matter, Phys Rev Lett, 99:185003 (2007).

(Jaramillo-Botero) Jaramillo-Botero, Su, Qi, Goddard, Large-scale, Long-term Non-adiabatic Electron Molecular Dynamics for Describing Material Properties and Phenomena in Extreme Environments, J Comp Chem, 32, 497-512 (2011).

pair_style eim command

pair_style eim/omp command

Syntax:

```
pair_style style
```

- style = *eim*

Examples:

```
pair_style eim
pair_coeff * * Na Cl ../potentials/ffield.eim Na Cl
pair_coeff * * Na Cl ffield.eim  Na Na Na Cl
pair_coeff * * Na Cl ../potentials/ffield.eim Cl NULL Na
```

Description:

Style *eim* computes pairwise interactions for ionic compounds using embedded-ion method (EIM) potentials (Zhou). The energy of the system E is given by

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=i_1}^{i_N} \phi_{ij}\left(r_{ij}\right) + \sum_{i=1}^{N} E_i\left(q_i, \sigma_i\right)$$

The first term is a double pairwise sum over the J neighbors of all I atoms, where phi_ij is a pair potential. The second term sums over the embedding energy E_i of atom I, which is a function of its charge q_i and the electrical potential sigma_i at its location. E_i, q_i, and sigma_i are calculated as

$$
\begin{aligned}
q_i &= \sum_{j=i_1}^{i_N} \eta_{ji}\left(r_{ij}\right) \\
\sigma_i &= \sum_{j=i_1}^{i_N} q_j \cdot \psi_{ij}\left(r_{ij}\right) \\
E_i\left(q_i, \sigma_i\right) &= \frac{1}{2} \cdot q_i \cdot \sigma_i
\end{aligned}
$$

where eta_ji is a pairwise function describing electron flow from atom I to atom J, and psi_ij is another pairwise function. The multi-body nature of the EIM potential is a result of the embedding energy term. A complete list of all the pair functions used in EIM is summarized below

$$(r) = \begin{cases} \left[\frac{E_{b,ij}\beta_{ij}}{\beta_{ij}-\alpha_{ij}} \exp\left(-\alpha_{ij}\frac{r-r_{e,ij}}{r_{e,ij}}\right) - \frac{E_{b,ij}\alpha_{ij}}{\beta_{ij}-\alpha_{ij}} \exp\left(-\beta_{ij}\frac{r-r_{e,ij}}{r_{e,ij}}\right) \right] f_c\left(r, r_{e,ij}, r_{c,\phi,ij}\right), \\ \left[\frac{E_{b,ij}\beta_{ij}}{\beta_{ij}-\alpha_{ij}} \left(\frac{r_{e,ij}}{r}\right)^{\alpha_{ij}} - \frac{E_{b,ij}\alpha_{ij}}{\beta_{ij}-\alpha_{ij}} \left(\frac{r_{e,ij}}{r}\right)^{\beta_{ij}} \right] f_c\left(r, r_{e,ij}, r_{c,\phi,ij}\right), \end{cases}$$

$$\eta_{ji} = A_{\eta,ij}\left(\chi_j - \chi_i\right) f_c\left(r, r_{s,\eta,ij}, r_{c,\eta,ij}\right)$$

$$\psi_{ij}\left(r\right) = A_{\psi,ij} \exp\left(-\zeta_{ij}r\right) f_c\left(r, r_{s,\psi,ij}, r_{c,\psi,ij}\right)$$

$$f_c\left(r, r_p, r_c\right) = 0.510204 \, erfc\left[\frac{1.64498\left(2r - r_p - r_c\right)}{r_c - r_p}\right] - 0.010204$$

Here E_b, r_e, r_(c,phi), alpha, beta, A_(psi), zeta, r_(s,psi), r_(c,psi), A_(eta), r_(s,eta), r_(c,eta), chi, and pair function type p are parameters, with subscripts ij indicating the two species of atoms in the atomic pair.

IMPORTANT NOTE: Even though the EIM potential is treating atoms as charged ions, you should not use a LAMMPS atom_style that stores a charge on each atom and thus requires you to assign a charge to each atom, e.g. the *charge* or *full* atom styles. This is because the EIM potential infers the charge on an atom from the equation above for q_i; you do not assign charges explicitly.

All the EIM parameters are listed in a potential file which is specified by the pair_coeff command. This is an ASCII text file in a format described below. The "ffield.eim" file included in the "potentials" directory of the LAMMPS distribution currently includes nine elements Li, Na, K, Rb, Cs, F, Cl, Br, and I. A system with any combination of these elements can be modeled. This file is parameterized in terms of LAMMPS metal units.

Note that unlike other potentials, cutoffs for EIM potentials are not set in the pair_style or pair_coeff command; they are specified in the EIM potential file itself. Likewise, the EIM potential file lists atomic masses; thus you do not need to use the mass command to specify them.

Only a single pair_coeff command is used with the *eim* style which specifies an EIM potential file and the element(s) to extract information for. The EIM elements are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- Elem1, Elem2, ...
- EIM potential file
- N element names = mapping of EIM elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example like one of those above, suppose you want to model a system with Na and Cl atoms. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Na, and the 4th to be Cl, you would use the following pair_coeff command:

pair_style eim/omp command   802

```
pair_coeff * * Na Cl ffield.eim Na Na Na Cl
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The filename is the EIM potential file. The Na and Cl arguments (before the file name) are the two elements for which info will be extracted from the potentail file. The first three trailing Na arguments map LAMMPS atom types 1,2,3 to the EIM Na element. The final Cl argument maps LAMMPS atom type 4 to the EIM Cl element.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when an *eim* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The ffield.eim file in the *potentials* directory of the LAMMPS distribution is formated as follows:

Lines starting with # are comments and are ignored by LAMMPS. Lines starting with "global:" include three global values. The first value divides the cations from anions, i.e., any elements with electronegativity above this value are viewed as anions, and any elements with electronegativity below this value are viewed as cations. The second and third values are related to the cutoff function - i.e. the 0.510204, 1.64498, and 0.010204 shown in the above equation can be derived from these values.

Lines starting with "element:" are formatted as follows: name of element, atomic number, atomic mass, electronic negativity, atomic radius (LAMMPS ignores it), ionic radius (LAMMPS ignores it), cohesive energy (LAMMPS ignores it), and q0 (must be 0).

Lines starting with "pair:" are entered as: element 1, element 2, r_(c,phi), r_(c,phi) (redundant for historical reasons), E_b, r_e, alpha, beta, r_(c,eta), A_(eta), r_(s,eta), r_(c,psi), A_(psi), zeta, r_(s,psi), and p.

The lines in the file can be in any order; LAMMPS extracts the info it needs.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default).

Related commands:

pair_coeff

Default: none

(Zhou) Zhou, submitted for publication (2010). Please contact Xiaowang Zhou (Sandia) for details via email at xzhou at sandia.gov.

pair_style gauss command

pair_style gauss/gpu command

pair_style gauss/omp command

pair_style gauss/cut command

pair_style gauss/cut/omp command

Syntax:

```
pair_style gauss cutoff
pair_style gauss/cut cutoff
```

- cutoff = global cutoff for Gauss interactions (distance units)

Examples:

```
pair_style gauss 12.0
pair_coeff * * 1.0 0.9
pair_coeff 1 4 1.0 0.9 10.0

pair_style gauss/cut 3.5
pair_coeff 1 4 0.2805 1.45 0.112
```

Description:

Style *gauss* computes a tethering potential of the form

$$E = -A \exp(-Br^2) \qquad r < r_c$$

between an atom and its corresponding tether site which will typically be a frozen atom in the simulation. Rc is the cutoff.

The following coefficients must be defined for each pair of atom types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- A (energy units)
- B (1/distance^2 units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff is used.

Style *gauss/cut* computes a generalized Gaussian interaction potential between pairs of particles:

$$E = \quad \frac{H}{\sigma_h \sqrt{2\pi}} \exp\left[-\frac{(r - r_{mh})^2}{2\sigma_h^2} \right]$$

where H determines together with the standard deviation sigma_h the peak height of the Gaussian function, and r_mh the peak position. Examples of the use of the Gaussian potentials include implicit solvent simulations of salt ions (Lenart) and of surfactants (Jusufi). In these instances the Gaussian potential mimics the hydration barrier between a pair of particles. The hydration barrier is located at r_mh and has a width of sigma_h. The prefactor determines the hight of the potential barrier.

The following coefficients must be defined for each pair of atom types via the pair_coeff command as in the example above, or in the data file or restart files read by the read_data or read_restart commands:

- H (energy * distance units)
- r_mh (distance units)
- sigma_h (distance units)

The global cutoff (r_c) specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the "-suffix command-line switch7_Section_start.html#start_6 when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

The *gauss* style does not support the pair_modify shift option. There is no effect due to the Gaussian well beyond the cutoff; hence reasonable cutoffs need to be specified.

The *gauss/cut* style supports the pair_modify shift option for the energy of the Gauss-potential portion of the pair interaction.

The pair_modify table and tail options are not relevant for these pair styles.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

The *gauss* pair style tallies an "occupancy" count of how many Gaussian-well sites have an atom within the distance at which the force is a maximum = sqrt(0.5/b). This quantity can be accessed via the compute pair command as a vector of values of length 1.

To print this quantity to the log file (with a descriptive column heading) the following commands could be included in an input script:

```
compute gauss all pair gauss
variable occ equal c_gauss[1]
thermo_style custom step temp epair v_occ
```

Restrictions:

The *gauss/cut* style is part of the "user-misc" package. It is only enabled if LAMMPS is build with that package. See the Making of LAMMPS section for more info.

Related commands:

pair_coeff, pair_style coul/diel

Default: none

(Lenart) Lenart , Jusufi, and Panagiotopoulos, J Chem Phys, 126, 044509 (2007).

(Jusufi) Jusufi, Hynninen, and Panagiotopoulos, J Phys Chem B, 112, 13783 (2008).

pair_style gayberne command

pair_style gayberne/gpu command

pair_style gayberne/omp command

Syntax:

```
pair_style gayberne gamma upsilon mu cutoff
```

- gamma = shift for potential minimum (typically 1)
- upsilon = exponent for eta orientation-dependent energy function
- mu = exponent for chi orientation-dependent energy function
- cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style gayberne 1.0 1.0 1.0 10.0
pair_coeff * * 1.0 1.7 1.7 3.4 3.4 1.0 1.0 1.0
```

Description:

The *gayberne* styles compute a Gay-Berne anisotropic LJ interaction (Berardi) between pairs of ellipsoidal particles or an ellipsoidal and spherical particle via the formulas

$$U(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}) = \mathbf{U_r}(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}, \gamma) \cdot \eta_{12}(\mathbf{A_1}, \mathbf{A_2}, \upsilon) \cdot \chi_{12}(\mathbf{A_1}, \mathbf{A_2}, \mathbf{r_{12}}, \mu)$$

$$U_r = 4\epsilon(\varrho^{12} - \varrho^6)$$

$$\varrho = \frac{\sigma}{h_{12} + \gamma\sigma}$$

where A1 and A2 are the transformation matrices from the simulation box frame to the body frame and r12 is the center to center vector between the particles. Ur controls the shifted distance dependent interaction based on the distance of closest approach of the two particles (h12) and the user-specified shift parameter gamma. When both particles are spherical, the formula reduces to the usual Lennard-Jones interaction (see details below for when Gay-Berne treats a particle as "spherical").

For large uniform molecules it has been shown that the energy parameters are approximately representable in terms of local contact curvatures (Everaers):

$$\epsilon_a = \sigma \cdot \frac{a}{b \cdot c}; \epsilon_b = \sigma \cdot \frac{b}{a \cdot c}; \epsilon_c = \sigma \cdot \frac{c}{a \cdot b}$$

The variable names utilized as potential parameters are for the most part taken from (Everaers) in order to be consistent with the RE-squared pair potential. Details on the upsilon and mu parameters are given here.

More details of the Gay-Berne formulation are given in the references listed below and in this supplementary document.

Use of this pair style requires the NVE, NVT, or NPT fixes with the *asphere* extension (e.g. fix nve/asphere) in order to integrate particle rotation. Additionally, atom_style ellipsoid should be used since it defines the rotational state and the size and shape of each ellipsoidal particle.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon = well depth (energy units)
- sigma = minimum effective particle radii (distance units)
- epsilon_i_a = relative well depth of type I for side-to-side interactions
- epsilon_i_b = relative well depth of type I for face-to-face interactions
- epsilon_i_c = relative well depth of type I for end-to-end interactions
- epsilon_j_a = relative well depth of type J for side-to-side interactions
- epsilon_j_b = relative well depth of type J for face-to-face interactions
- epsilon_j_c = relative well depth of type J for end-to-end interactions
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used.

It is typical with the Gay-Berne potential to define *sigma* as the minimum of the 3 shape diameters of the particles involved in an I,I interaction, though this is not required. Note that this is a different meaning for *sigma* than the pair_style resquared potential uses.

The epsilon_i and epsilon_j coefficients are actually defined for atom types, not for pairs of atom types. Thus, in a series of pair_coeff commands, they only need to be specified once for each atom type.

Specifically, if any of epsilon_i_a, epsilon_i_b, epsilon_i_c are non-zero, the three values are assigned to atom type I. If all the epsilon_i values are zero, they are ignored. If any of epsilon_j_a, epsilon_j_b, epsilon_j_c are non-zero, the three values are assigned to atom type J. If all three epsilon_j values are zero, they are ignored. Thus the typical way to define the epsilon_i and epsilon_j coefficients is to list their values in "pair_coeff I J" commands when I = J, but set them to 0.0 when I != J. If you do list them when I != J, you should insure they are consistent with their values in other pair_coeff commands, since only the last setting will be in effect.

Note that if this potential is being used as a sub-style of pair_style hybrid, and there is no "pair_coeff I I" setting made for Gay-Berne for a particular type I (because I-I interactions are computed by another hybrid pair potential), then you still need to insure the epsilon a,b,c coefficients are assigned to that type. e.g. in a "pair_coeff I J" command.

IMPORTANT NOTE: If the epsilon a = b = c for an atom type, and if the shape of the particle itself is spherical, meaning its 3 shape parameters are all the same, then the particle is treated as an LJ sphere by the

Gay-Berne potential. This is significant because if two LJ spheres interact, then the simple Lennard-Jones formula is used to compute their interaction energy/force using the specified epsilon and sigma as the standard LJ parameters. This is much cheaper to compute than the full Gay-Berne formula. To treat the particle as a LJ sphere with sigma = D, you should normally set epsilon a = b = c = 1.0, set the pair_coeff sigma = D, and also set the 3 shape parameters for the particle to D. The one exception is that if the 3 shape parameters are set to 0.0, which is a valid way in LAMMPS to specify a point particle, then the Gay-Berne potential will treat that as shape parameters of 1.0 (i.e. a LJ particle with sigma = 1), since it requires finite-size particles. In this case you should still set the pair_coeff sigma to 1.0 as well.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair styles supports the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction, but only for sphere-sphere interactions. There is no shifting performed for ellipsoidal interactions due to the anisotropic dependence of the interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *gayberne* style is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

These pair style require that atoms store torque and a quaternion to represent their orientation, as defined by the atom_style. It also require they store a per-type shape. The particles cannot store a per-particle diameter.

This pair style requires that atoms be ellipsoids as defined by the atom_style ellipsoid command.

Particles acted on by the potential can be finite-size aspherical or spherical particles, or point particles. Spherical particles have all 3 of their shape parameters equal to each other. Point particles have all 3 of their shape parameters equal to 0.0.

The Gay-Berne potential does not become isotropic as r increases (Everaers). The distance-of-closest-approach approximation used by LAMMPS becomes less accurate when high-aspect ratio ellipsoids are used.

Related commands:

pair_coeff, fix nve/asphere, compute temp/asphere, pair_style resquared

Default: none

(Everaers) Everaers and Ejtehadi, Phys Rev E, 67, 041710 (2003).

(Berardi) Berardi, Fava, Zannoni, Chem Phys Lett, 297, 8-14 (1998). Berardi, Muccioli, Zannoni, J Chem Phys, 128, 024905 (2008).

(Perram) Perram and Rasmussen, Phys Rev E, 54, 6565-6572 (1996).

(Allen) Allen and Germano, Mol Phys 104, 3225-3235 (2006).

pair_style gran command

Syntax:

```
pair_style model_type model_name model_keyword model_value
```

- zero or more model_type/model_name pairs may be appended. They must be appended in the following order (!)

  ```
  model values = described here
    tangential values = described here
    cohesion values = described here
    rolling_friction values = described here
    surface values = described here
  ```

- following the model_type/model_name pairs, zero or more model_keyword/model_valuezero pairs may be appended in arbitrary order

  ```
  model_type/model_name pairs = described for each model separately here
  ```

Examples:

```
pair_style gran model hooke tangential history
pair_style gran model hertz tangential history rolling_friction cdt
pair_style gran model hertz tangential no_history cohesion sjkr
```

LIGGGHTS vs. LAMMPS Info:

This LIGGGHTS command offers the following improvements vs. LAMMPS: The stiffness and damping coefficients k_n, k_t, gamma_n, gamma_t can now be derived from the material properties. Also, a new model for macroscopic cohesion is introduced.

General description:

The *gran* styles use the following formula for the frictional force between two granular particles, when the distance r between two particles of radii Ri and Rj is less than their contact distance d = Ri + Rj. Typically, there is no force between the particles when r > d.

$$F = \left(k_n \underbrace{\delta n_{ij}}_{\substack{normal \\ overlap}} - \gamma_n \underbrace{vn_{ij}}_{\substack{normal \\ relative\ vel.}} \right) + \left(k_t \underbrace{\delta t_{ij}}_{\substack{tangential \\ overlap}} - \gamma_t \underbrace{vt_{ij}}_{\substack{tangential \\ relative\ v}} \right)$$

$$\underbrace{\hspace{4cm}}_{normal\ force} \qquad \underbrace{\hspace{4cm}}_{tangential\ force}$$

The tangential overlap is truncated to fulfil $F_t \le X_\mu F_n$

The quantities in the equations are as follows:

- delta_n = d - r = overlap distance of 2 particles
- k_n = elastic constant for normal contact
- k_t = elastic constant for tangential contact
- gamma_n = viscoelastic damping constant for normal contact
- gamma_t = viscoelastic damping constant for tangential contact
- delta_t = tangential displacement vector between 2 spherical particles

In the first term is the normal force between the two particles and the second term is the tangential force. The normal force has 2 terms, a contact force and a damping force. The tangential force also has 2 terms: a shear force and a damping force. The shear force is a "history" effect that accounts for the tangential displacement ("tangential overlap") between the particles for the duration of the time they are in contact.

The concrete implementation for k_n, k_t, gamma_n, gamma_t and the shear history depend on the concrete models as chosen by the user. They are described on separate doc pages here

Also, other models may add additional forces or torques on the particles, such as cohesive or rolling friction forces. These are also described on separate doc pages here

IMPORTANT NOTE: The order of model keywords is important, you have to stick to the order as outlined in the "Syntax" section of this doc page.

General comments:

For granular styles there are no additional coefficients to set for each pair of atom types via the pair_coeff command. All settings are global and are made via the pair_style command. However you must still use the pair_coeff for all pairs of granular atom types. For example the command

```
pair_coeff * *
```

should be used if all atoms in the simulation interact via a granular potential (i.e. one of the pair styles above is used). If a granular potential is used as a sub-style of pair_style hybrid, then specific atom types can be used in the pair_coeff command to determine which atoms interact via a granular potential.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for granular pair styles.

These pair styles write their information to binary restart files, so a pair_style command does not need to be specified in an input script that reads a restart file.

IMPORTANT NOTE: The material properties are not written to restart files! Thus, if you restart a simulation, you have to re-define them (by using the fixes mentioned above).

These pair styles can only be used via the pair keyword of the run_style respa command. They do not support the inner, middle, outer keywords.

Restrictions:

These pair styles require that atoms store torque and angular velocity (omega) as defined by the atom_style. They also require a per-particle radius is stored. The *sphere* or *granular* atom style does all of this.

This pair style requires you to use the communicate vel yes option so that velocites are stored by ghost atoms.

pair_style gran command   813

Only unit system that are self-consistent (si, cgs, lj) can be used with this pair style.

Related commands:

pair_coeff Models for use with this command are described here

Default:

model = 'hertz' *tangential* = 'history' *rolling_friction* = 'off' *cohesion* = 'off' *surface* = 'default'

pair_style lj/gromacs command

pair_style lj/gromacs/cuda command

pair_style lj/gromacs/omp command

pair_style lj/gromacs/coul/gromacs command

pair_style lj/gromacs/coul/gromacs/cuda command

pair_style lj/gromacs/coul/gromacs/omp command

Syntax:

```
pair_style style args
```

- style = *lj/gromacs* or *lj/gromacs/coul/gromacs*
- args = list of arguments for a particular style

```
lj/gromacs args = inner outer
    inner, outer = global switching cutoffs for Lennard Jones
  lj/gromacs/coul/gromacs args = inner outer (inner2) (outer2)
    inner, outer = global switching cutoffs for Lennard Jones (and Coulombic if only 2 args)
    inner2, outer2 = global switching cutoffs for Coulombic (optional)
```

Examples:

```
pair_style lj/gromacs 9.0 12.0
pair_coeff * * 100.0 2.0
pair_coeff 2 2 100.0 2.0 8.0 10.0

pair_style lj/gromacs/coul/gromacs 9.0 12.0
pair_style lj/gromacs/coul/gromacs 8.0 10.0 7.0 9.0
pair_coeff * * 100.0 2.0
```

Description:

The *lj/gromacs* styles compute shifted LJ and Coulombic interactions with an additional switching function S(r) that ramps the energy and force smoothly to zero between an inner and outer cutoff. It is a commonly used potential in the [GROMACS](#) MD code and for the coarse-grained models of [(Marrink)](#).

$$E_{LJ} = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] + S_{LJ}(r) \qquad r < r_c$$

$$E_C = \frac{Cq_iq_j}{\epsilon r} + S_C(r) \qquad r < r_c$$

$$S(r) = C \qquad r < r_1$$

$$S(r) = \frac{A}{3}(r - r_1)^3 + \frac{B}{4}(r - r_1)^4 + C \qquad r_1 < r < r_c$$

$$A = (-3E'(r_c) + (r_c - r_1)E''(r_c))/(r_c - r_1)^2$$

$$B = (2E'(r_c) - (r_c - r_1)E''(r_c))/(r_c - r_1)^3$$

$$C = -E(r_c) + \frac{1}{2}(r_c - r_1)E'(r_c) - \frac{1}{12}(r_c - r_1)^2 E''(r_c))$$

r1 is the inner cutoff; rc is the outer cutoff. The coefficients A, B, and C are computed by LAMMPS to perform the shifting and smoothing. The function S(r) is actually applied once to each term of the LJ formula and once to the Coulombic formula, so there are 2 or 3 sets of A,B,C coefficients depending on which pair_style is used. The boundary conditions applied to the smoothing function are as follows: S'(r1) = S''(r1) = 0, S(rc) = -E(rc), S'(rc) = -E'(rc), and S''(rc) = -E''(rc), where E(r) is the corresponding term in the LJ or Coulombic potential energy function. Single and double primes denote first and second derivatives with respect to r, respectively.

The inner and outer cutoff for the LJ and Coulombic terms can be the same or different depending on whether 2 or 4 arguments are used in the pair_style command. The inner LJ cutoff must be > 0, but the inner Coulombic cutoff can be >= 0.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- inner (distance units)
- outer (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The last 2 coefficients are optional inner and outer cutoffs for style *lj/gromacs*. If not specified, the global *inner* and *outer* values are used.

The last 2 coefficients cannot be used with style *lj/gromacs/coul/gromacs* because this force field does not allow varying cutoffs for individual atom pairs; all pairs use the global cutoff(s) specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in

pair_style lj/gromacs/coul/gromacs/omp command

[Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

None of the GROMACS pair styles support the [pair_modify](#) shift option, since the Lennard-Jones portion of the pair interaction is already smoothed to 0.0 at the cutoff.

The [pair_modify](#) table option is not relevant for this pair style.

None of the GROMACS pair styles support the [pair_modify](#) tail option for adding long-range tail corrections to energy and pressure, since there are no corrections for a potential that goes to 0.0 at the cutoff.

All of the GROMACS pair styles write their information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

All of the GROMACS pair styles can only be used via the *pair* keyword of the [run_style respa](#) command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

(Marrink) Marrink, de Vries, Mark, J Phys Chem B, 108, 750-760 (2004).

pair_style hbond/dreiding/lj command

pair_style hbond/dreiding/lj/omp command

pair_style hbond/dreiding/morse command

pair_style hbond/dreiding/morse/omp command

Syntax:

```
pair_style style N inner_distance_cutoff outer_distance_cutoff angle_cutof
```

- style = *hbond/dreiding/lj* or *hbond/dreiding/morse*
- n = cosine angle periodicity
- inner_distance_cutoff = global inner cutoff for Donor-Acceptor interactions (distance units)
- outer_distance_cutoff = global cutoff for Donor-Acceptor interactions (distance units)
- angle_cutoff = global angle cutoff for Acceptor-Hydrogen-Donor
- interactions (degrees)

Examples:

```
pair_style hybrid/overlay lj/cut 10.0 hbond/dreiding/lj 4 9.0 11.0 90
pair_coeff 1 2 hbond/dreiding/lj 3 i 9.5 2.75 4 9.0 11.0 90.0

pair_style hybrid/overlay lj/cut 10.0 hbond/dreiding/morse 2 9.0 11.0 90
pair_coeff 1 2 hbond/dreiding/morse 3 i 3.88 1.7241379 2.9 2 9 11 90
```

Description:

The *hbond/dreiding* styles compute the Acceptor-Hydrogen-Donor (AHD) 3-body hydrogen bond interaction for the [DREIDING](#) force field, given by:

$$
\begin{aligned}
E &= [LJ(r)|Morse(r)] & r < r_{\text{in}} \\
&= S(r) * [LJ(r)|Morse(r)] & r_{\text{in}} < r < r_{\text{out}} \\
&= 0 & r > r_{\text{out}}
\end{aligned}
$$

$$
LJ(r) = AR^{-12} - BR^{-10}cos^n\theta = \epsilon \left\{ 5 \left[\frac{\sigma}{r}\right]^{12} - 6\left[\frac{\sigma}{r}\right]^{10} \right\} cos^n\theta
$$

$$
Morse(r) = D_0 \left\{ \chi^2 - 2\chi \right\} cos^n\theta = D_0 \left\{ e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right\} cos^n\theta
$$

$$
S(r) = \frac{[r_{\text{out}}^2 - r^2]^2 [r_{\text{out}}^2 + 2r^2 - 3r_{\text{in}}^2]}{[r_{\text{out}}^2 - r_{\text{in}}^2]^3}
$$

where Rin is the inner spline distance cutoff, Rout is the outer distance cutoff, theta_c is the angle cutoff, and n is the cosine periodicity.

Here, *r* is the radial distance between the donor (D) and acceptor (A) atoms and *theta* is the bond angle between the acceptor, the hydrogen (H) and the donor atoms:
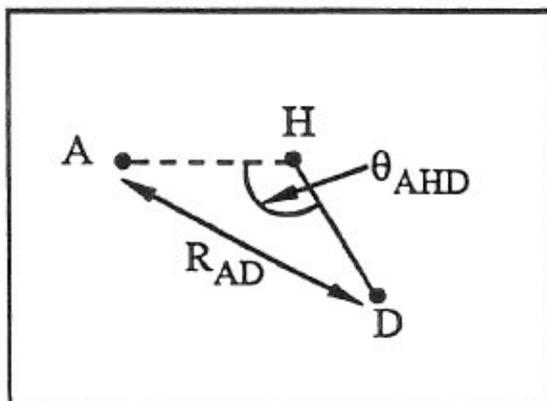
These 3-body interactions can be defined for pairs of acceptor and donor atoms, based on atom types. For each donor/acceptor atom pair, the 3rd atom in the interaction is a hydrogen permanently bonded to the donor atom, e.g. in a bond list read in from a data file via the read_data command. The atom types of possible hydrogen atoms for each donor/acceptor type pair are specified by the pair_coeff command (see below).

Style *hbond/dreiding/lj* is the original DREIDING potential of (Mayo). It uses a LJ 12/10 functional for the Donor-Acceptor interactions. To match the results in the original paper, use n = 4.

Style *hbond/dreiding/morse* is an improved version using a Morse potential for the Donor-Acceptor interactions. (Liu) showed that the Morse form gives improved results for Dendrimer simulations, when n = 2.

See this howto section of the manual for more information on the DREIDING forcefield.

IMPORTANT NOTE: Because the Dreiding hydrogen bond potential is only one portion of an overall force field which typically includes other pairwise interactions, it is common to use it as a sub-style in a pair_style hybrid/overlay command, where another pair style provides the repulsive core interaction between pairs of atoms, e.g. a 1/r^12 Lennard-Jones repulsion.

IMPORTANT NOTE: When using the hbond/dreiding pair styles with pair_style hybrid/overlay, you should explicitly define pair interactions between the donor atom and acceptor atoms, (as well as between these atoms and ALL other atoms in your system). Whenever pair_style hybrid/overlay is used, ordinary mixing rules are not applied to atoms like the donor and acceptor atoms because they are typically referenced in multiple pair styles. Neglecting to do this can cause difficult-to-detect physics problems.

IMPORTANT NOTE: In the original Dreiding force field paper 1-4 non-bonded interactions ARE allowed. If this is desired for your model, use the special_bonds command (e.g. "special_bonds lj 0.0 0.0 1.0") to turn these interactions on.

The following coefficients must be defined for pairs of eligible donor/acceptor types via the pair_coeff command as in the examples above.

IMPORTANT NOTE: Unlike other pair styles and their associated pair_coeff commands, you do not need to specify pair_coeff settings for all possible I,J type pairs. Only I,J type pairs for atoms which act as joint donors/acceptors need to be specified; all other type pairs are assumed to be inactive.

pair_style hbond/dreiding/morse/omp command   819

IMPORTANT NOTE: A pair_coeff command can be specied multiple times for the same donor/acceptor type pair. This enables multiple hydrogen types to be assigned to the same donor/acceptor type pair. For other pair_styles, if the pair_coeff command is re-used for the same I.J type pair, the settings for that type pair are overwritten. For the hydrogen bond potentials this is not the case; the settings are cummulative. This means the only way to turn off a previous setting, is to re-use the pair_style command and start over.

For the *hbond/dreiding/lj* style the list of coefficients is as follows:

- K = hydrogen atom type = 1 to Ntypes
- donor flag = *i* or *j*
- epsilon (energy units)
- sigma (distance units)
- n = exponent in formula above
- distance cutoff Rin (distance units)
- distance cutoff Rout (distance units)
- angle cutoff (degrees)

For the *hbond/dreiding/morse* style the list of coefficients is as follows:

- K = hydrogen atom type = 1 to Ntypes
- donor flag = *i* or *j*
- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)
- n = exponent in formula above
- distance cutoff Rin (distance units)
- distance cutoff Rout (distance units)
- angle cutoff (degrees)

A single hydrogen atom type K can be specified, or a wild-card asterisk can be used in place of or in conjunction with the K arguments to select multiple types as hydrogens. This takes the form "*" or "*n" or "n*" or "m*n". See the pair_coeff command doc page for details.

If the donor flag is *i*, then the atom of type I in the pair_coeff command is treated as the donor, and J is the acceptor. If the donor flag is *j*, then the atom of type J in the pair_coeff command is treated as the donor and I is the donor. This option is required because the pair_coeff command requires that I <= J.

Epsilon and sigma are settings for the hydrogen bond potential based on a Lennard-Jones functional form. Note that sigma is defined as the zero-crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

D0 and alpha and r0 are settings for the hydrogen bond potential based on a Morse functional form.

The last 3 coefficients for both styles are optional. If not specified, the global n, distance cutoff, and angle cutoff specified in the pair_style command are used. If you wish to only override the 2nd or 3rd optional parameter, you must also specify the preceding optional parameters.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for

more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. You must explicitly identify each donor/acceptor type pair.

These styles do not support the pair_modify shift option for the energy of the interactions.

The pair_modify table option is not relevant for these pair styles.

These pair styles do not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

These pair styles do not write their information to binary restart files, so pair_style and pair_coeff commands need to be re-specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

These pair styles tally a count of how many hydrogen bonding interactions they calculate each timestep and the hbond energy. These quantities can be accessed via the compute pair command as a vector of values of length 2.

To print these quantities to the log file (with a descriptive column heading) the following commands could be included in an input script:

```
compute hb all pair hbond/dreiding/lj
variable n_hbond equal c_hb[1] #number hbonds
variable E_hbond equal c_hb[2] #hbond energy
thermo_style custom step temp epair v_E_hbond
```

Restrictions: none

Related commands:

pair_coeff

Default: none

(**Mayo**) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897-8909 (1990).

(**Liu**) Liu, Bryantsev, Diallo, Goddard III, J. Am. Chem. Soc 131 (8) 2798 (2009)

pair_style hybrid command

pair_style hybrid/omp command

pair_style hybrid/overlay command

pair_style hybrid/overlay/omp command

Syntax:

```
pair_style hybrid style1 args style2 args ...
pair_style hybrid/overlay style1 args style2 args ...
```

- style1,style2 = list of one or more pair styles and their arguments

Examples:

```
pair_style hybrid lj/cut/coul/cut 10.0 eam lj/cut 5.0
pair_coeff 1*2 1*2 eam niu3
pair_coeff 3 3 lj/cut/coul/cut 1.0 1.0
pair_coeff 1*2 3 lj/cut 0.5 1.2

pair_style hybrid/overlay lj/cut 2.5 coul/long 2.0
pair_coeff * * lj/cut 1.0 1.0
pair_coeff * * coul/long
```

Description:

The *hybrid* and *hybrid/overlay* styles enable the use of multiple pair styles in one simulation. With the *hybrid* style, exactly one pair style is assigned to each pair of atom types. With the *hybrid/overlay* style, one or more pair styles can be assigned to each pair of atom types. The assignment of pair styles to type pairs is made via the [pair_coeff](#) command.

Here are two examples of hybrid simulations. The *hybrid* style could be used for a simulation of a metal droplet on a LJ surface. The metal atoms interact with each other via an *eam* potential, the surface atoms interact with each other via a *lj/cut* potential, and the metal/surface interaction is also computed via a *lj/cut* potential. The *hybrid/overlay* style could be used as in the 2nd example above, where multiple potentials are superposed in an additive fashion to compute the interaction between atoms. In this example, using *lj/cut* and *coul/long* together gives the same result as if the *lj/cut/coul/long* potential were used by itself. In this case, it would be more efficient to use the single combined potential, but in general any combination of pair potentials can be used together in to produce an interaction that is not encoded in any single pair_style file, e.g. adding Coulombic forces between granular particles.

All pair styles that will be used are listed as "sub-styles" following the *hybrid* or *hybrid/overlay* keyword, in any order. Each sub-style's name is followed by its usual arguments, as illustrated in the example above. See the doc pages of individual pair styles for a listing and explanation of the appropriate arguments.

Note that an individual pair style can be used multiple times as a sub-style. For efficiency this should only be done if your model requires it. E.g. if you have different regions of Si and C atoms and wish to use a Tersoff potential for pure Si for one set of atoms, and a Tersoff potnial for pure C for the other set (presumably with some 3rd potential for Si-C interactions), then the sub-style *tersoff* could be listed twice. But if you just want to use a Lennard-Jones or other pairwise potential for several different atom type pairs in your model, then you should just list the sub-style once and use the pair_coeff command to assign parameters for the different

type pairs.

IMPORTANT NOTE: An exception to this option to list an individual pair style multiple times are the pair styles implemented as Fortran libraries: pair_style meam and pair_style reax (pair_style reax/c is OK). This is because unlike a C++ class, they can not be instantiated multiple times, due to the manner in which they were coded in Fortran.

In the pair_coeff commands, the name of a pair style must be added after the I,J type specification, with the remaining coefficients being those appropriate to that style. If the pair style is used multiple times in the pair_style command with, then an additional numeric argument must also be included which is the number from 1 to M where M is the number of times the sub-style was listed in the pair style command. The extra number indicates which instance of the sub-style these coefficients apply to.

For example, consider a simulation with 3 atom types: types 1 and 2 are Ni atoms, type 3 are LJ atoms with charges. The following commands would set up a hybrid simulation:

```
pair_style hybrid eam/alloy lj/cut/coul/cut 10.0 lj/cut 8.0
pair_coeff * * eam/alloy nialhjea Ni Ni NULL
pair_coeff 3 3 lj/cut/coul/cut 1.0 1.0
pair_coeff 1*2 3 lj/cut 0.8 1.3
```

As an example of using the same pair style multiple times, consider a simulation with 2 atom types. Type 1 is Si, type 2 is C. The following commands would model the Si atoms with Tersoff, the C atoms with Tersoff, and the cross-interactions with Lennard-Jones:

```
pair_style hybrid lj/cut tersoff tersoff
pair_coeff * * tersoff 1 Si.tersoff Si NULL
pair_coeff * * tersoff 2 C.tersoff NULL C
pair_coeff 1 2 lj/cut 1.0 1.5
```

If pair coefficients are specified in the data file read via the read_data command, then the same rule applies. E.g. "eam/alloy" or "lj/cut" must be added after the atom type, for each line in the "Pair Coeffs" section, e.g.

```
Pair Coeffs

1 lj/cut/coul/cut 1.0 1.0
...
```

Note that the pair_coeff command for some potentials such as pair_style eam/alloy includes a mapping specification of elements to all atom types, which in the hybrid case, can include atom types not assigned to the *eam/alloy* potential. The NULL keyword is used by many such potentials (eam/alloy, Tersoff, AIREBO, etc), to denote an atom type that will be assigned to a different sub-style.

For the *hybrid* style, each atom type pair I,J is assigned to exactly one sub-style. Just as with a simulation using a single pair style, if you specify the same atom type pair in a second pair_coeff command, the previous assignment will be overwritten.

For the *hybrid/overlay* style, each atom type pair I,J can be assigned to one or more sub-styles. If you specify the same atom type pair in a second pair_coeff command with a new sub-style, then the second sub-style is added to the list of potentials that will be calculated for two interacting atoms of those types. If you specify the same atom type pair in a second pair_coeff command with a sub-style that has already been defined for that pair of atoms, then the new pair coefficients simply override the previous ones, as in the normal usage of the pair_coeff command. E.g. these two sets of commands are the same:

```
pair_style lj/cut 2.5
pair_coeff * * 1.0 1.0
pair_coeff 2 2 1.5 0.8
```

```
pair_style hybrid/overlay lj/cut 2.5
pair_coeff * * lj/cut 1.0 1.0
pair_coeff 2 2 lj/cut 1.5 0.8
```

Coefficients must be defined for each pair of atoms types via the pair_coeff command as described above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below.

For both the *hybrid* and *hybrid/overlay* styles, every atom type pair I,J (where I <= J) must be assigned to at least one sub-style via the pair_coeff command as in the examples above, or in the data file read by the read_data, or by mixing as described below.

If you want there to be no interactions between a particular pair of atom types, you have 3 choices. You can assign the type pair to some sub-style and use the neigh_modify exclude type command. You can assign it to some sub-style and set the coefficients so that there is effectively no interaction (e.g. epsilon = 0.0 in a LJ potential). Or, for *hybrid* and *hybrid/overlay* simulations, you can use this form of the pair_coeff command in your input script:

```
pair_coeff      2 3 none
```

or this form in the "Pair Coeffs" section of the data file:

```
3 none
```

If an assignment to *none* is made in a simulation with the *hybrid/overlay* pair style, it wipes out all previous assignments of that atom type pair to sub-styles.

Note that you may need to use an atom_style hybrid command in your input script, if atoms in the simulation will need attributes from several atom styles, due to using multiple pair potentials.

The potential energy contribution to the overall system due to an individual sub-style can be accessed and output via the compute pair command.

IMPORTANT: Several of the potentials defined via the pair_style command in LAMMPS are really many-body potentials, such as Tersoff, AIREBO, MEAM, ReaxFF, etc. The way to think about using these potentials in a hybrid setting is as follows.

A subset of atom types is assigned to the many-body potential with a single pair_coeff command, using "* *" to include all types and the NULL keywords described above to exclude specific types not assigned to that potential. If types 1,3,4 were assigned in that way (but not type 2), this means that all many-body interactions between all atoms of types 1,3,4 will be computed by that potential. Pair_style hybrid allows interactions between type pairs 2-2, 1-2, 2-3, 2-4 to be specified for computation by other pair styles. You could even add a second interaction for 1-1 to be computed by another pair style, assuming pair_style hybrid/overlay is used.

But you should not, as a general rule, attempt to exclude the many-body interactions for some subset of the type pairs within the set of 1,3,4 interactions, e.g. exclude 1-1 or 1-3 interactions. That is not conceptually well-defined for many-body interactions, since the potential will typically calculate energies and foces for small groups of atoms, e.g. 3 or 4 atoms, using the neighbor lists of the atoms to find the additional atoms in the group. It is typically non-physical to think of excluding an interaction between a particular pair of atoms when the potential computes 3-body or 4-body interactions.

However, you can still use the pair_coeff none setting or the neigh_modify exclude command to exclude certain type pairs from the neighbor list that will be passed to a manybody sub-style. This will alter the calculations made by a many-body potential, since it builds its list of 3-body, 4-body, etc interactions from the pair list. You will need to think carefully as to whether it produces a physically meaningful result for your

model.

For example, imagine you have two atom types in your model, type 1 for atoms in one surface, and type 2 for atoms in the other, and you wish to use a Tersoff potential to compute interactions within each surface, but not between surfaces. Then either of these two command sequences would implement that model:

```
pair_style hybrid tersoff
pair_coeff * * tersoff SiC.tersoff C C
pair_coeff 1 2 none

pair_style tersoff
pair_coeff * * SiC.tersoff C C
neigh_modify exclude type 1 2
```

Either way, only neighbor lists with 1-1 or 2-2 interactions would be passed to the Tersoff potential, which means it would compute no 3-body interactions containing both type 1 and 2 atoms.

Here is another example, using hybrid/overlay, to use 2 many-body potentials together, in an overlapping manner. Imagine you have CNT (C atoms) on a Si surface. You want to use Tersoff for Si/Si and Si/C interactions, and AIREBO for C/C interactions. Si atoms are type 1; C atoms are type 2. Something like this will work:

```
pair_style hybrid/overlay tersoff airebo 3.0
pair_coeff * * tersoff SiC.tersoff.custom Si C
pair_coeff * * airebo CH.airebo NULL C
```

Note that to prevent the Tersoff potential from computing C/C interactions, you would need to modify the SiC.tersoff file to turn off C/C interaction, i.e. by setting the appropriate coefficients to 0.0.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual.

Since the *hybrid* and *hybrid/overlay* styles delegate computation to the individual sub-styles, the suffix versions of the *hybrid* and *hybrid/overlay* styles are used to propagate the corresponding suffix to all sub-styles, if those versions exist. Otherwise the non-accelerated version will be used.

The individual accelerated sub-styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

Any pair potential settings made via the pair_modify command are passed along to all sub-styles of the hybrid potential.

For atom type pairs I,J and I != J, if the sub-style assigned to I,I and J,J is the same, and if the sub-style allows for mixing, then the coefficients for I,J can be mixed. This means you do not have to specify a pair_coeff command for I,J since the I,J type pair will be assigned automatically to the I,I sub-style and its coefficients generated by the mixing rule used by that sub-style. For the *hybrid/overlay* style, there is an additional

requirement that both the I,I and J,J pairs are assigned to a single sub-style. See the "pair_modify" command for details of mixing rules. See the See the doc page for the sub-style to see if allows for mixing.

The hybrid pair styles supports the [pair_modify](#) shift, table, and tail options for an I,J pair interaction, if the associated sub-style supports it.

For the hybrid pair styles, the list of sub-styles and their respective settings are written to [binary restart files](#), so a [pair_style](#) command does not need to specified in an input script that reads a restart file. However, the coefficient information is not stored in the restart file. Thus, pair_coeff commands need to be re-specified in the restart input script.

These pair styles support the use of the *inner*, *middle*, and *outer* keywords of the [run_style respa](#) command, if their sub-styles do.

Restrictions:

When using a long-range Coulombic solver (via the [kspace_style](#) command) with a hybrid pair_style, one or more sub-styles will be of the "long" variety, e.g. *lj/cut/coul/long* or *buck/coul/long*. You must insure that the short-range Coulombic cutoff used by each of these long pair styles is the same or else LAMMPS will generate an error.

Related commands:

[pair_coeff](#)

Default: none

pair_style kim command

Syntax:

```
pair_style kim virialmode model
```

- virialmode = KIMvirial or LAMMPSvirial
- model = name of KIM model (potential)

Examples:

```
pair_style kim KIMvirial model_Ar_P_Morse
pair_coeff * * Ar Ar
```

Description:

This pair style is a wrapper on the [Knowledge Base for Interatomic Models (KIM)](#) repository of interatomic potentials, so that they can be used by LAMMPS scripts.

In KIM lingo, a potential is a "model" and a model contains both the analytic formulas that define the potential as well as the parameters needed to run it for one or more materials, including coefficients and cutoffs.

The argument *virialmode* determines how the global virial is calculated. If *KIMvirial* is specified, the KIM model performs the global virial calculation. If *LAMMPSvirial* is specified, LAMMPS computes the global virial using its fdotr mechanism.

The argument *model* is the name of the KIM model for a specific potential as KIM defines it. In principle, LAMMPS can invoke any KIM model. You should get an error or warning message from either LAMMPS or KIM if there is an incompatibility.

Only a single pair_coeff command is used with the *kim* style which specifies the mapping of LAMMPS atom types to KIM elements. This is done by specifying N additional arguments after the * * in the pair_coeff command, where N is the number of LAMMPS atom types:

- N element names = mapping of KIM elements to atom types

As an example, imagine the KIM model supports Si and C atoms. If your LAMMPS simulation has 4 atom types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to Si as defined within KIM. The final C argument maps LAMMPS atom type 4 to C as defined within KIM. If a mapping value is specified as NULL, the mapping is not performed. This can only be used when a *kim* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

In addition to the usual LAMMPS error messages, the KIM library itself may generate errors, which should be printed to the screen. In this case it is also useful to check the kim.log file for additional error information. This file kim.log should be generated in the same directory where LAMMPS is running.

Here is information on how to build KIM for use with LAMMPS. There is a directory src/KIM/ with an important file in it: Makefile.lammps. When you do 'make yes-kim' LAMMPS will use the settings in src/KIM/Makefile.lammps to find KIM header files and the KIM library itself for linking purposes. Thus, you should ensure Makefile.lammps has the correct settings for your system and your build of KIM.

Consult the KIM documentation for further details on KIM specifics.

OpenKIM is available for download from [this site](#), namely http://openkim.org. The tarball you download is "openkim-api-vX.X.X.tgz", which can be unpacked via

```
tar xvfz openkim*tgz
```

The openkim-api-vX.X.X/DOCS directory has further documentation. For more information on installing KIM and troubleshooting refer to openkim/INSTALL.

Here is a brief summary of how to build KIM:

1. If you have previously used the openkim-api package (versions 1.1.1 or below), it is recommended that you remove all associated environment variables from your environment. (These include, KIM_DIR, KIM_INTEL, KIM_SYSTEM32, KIM_DYNAMIC, KIM_API_DIR, KIM_TESTS_DIR, KIM_MODEL_DRIVERS_DIR, and KIM_MODELS_DIR.) All setting are now specified in the Makefile.KIM_Config file.
2. Set up the Makefile.KIM_Config file

   (a) Copy the file Makefile.KIM_Config.example to Makefile.KIM_Config
       For example, if you untarred the `openkim-api-vX.X.X.tgz' tarball in
       your home directory, you would do:

       % cd $HOME/openkim-api-vX.X.X
       % cp Makefile.KIM_Config.example Makefile.KIM_Config

       The `%' symbol represents the bash sell prompt and should not be typed.

       (Above `vX.X.X' represents the current release number.)

   (b) Edit the file Makefile.KIM_Config and set the appropriate value for the
       KIM_DIR variable. This must expand to an absolute path. Using the same
       assumptions as above, this would be

       KIM_DIR = $(HOME)/openkim-api-vX.X.X

   (c) If appropriate, set explicit values for the remaining three
       directories

       KIM_MODEL_DRIVERS_DIR =
       KIM_MODELS_DIR =
       KIM_TESTS_DIR =

       If these lines are commented out, defaults will be provided by the
       openkim-api make system.

   (d) Set the value of KIM_COMPILERSUITE. Possible values are `GCC' and
       `INTEL'.

       KIM_COMPILERSUITE = GCC

   (e) Set the value of KIM_SYSTEMLINKER. Possible values are `linux',
       `freebsd', and `darwin'.

```
    KIM_SYSTEMLINKER = linux


(f) Set the value of KIM_SYSTEMARCH.  Possible values are `32bit' and
    `64bit'.

    KIM_SYSTEMARCH = 64bit


(g) Set the value of KIM_LINK.  Possible values are `dynamic-load',
    `dynamic-link', and `static-link'.

    KIM_LINK = dynamic-load

    `dynamic-load' is the preferred option. (Unless performance or
    other issues require it, you should use dynamic-load.)
    `dynamic-link' is like `dynamic-load' but does not use the dl.h
    library.  This option requires a more complicated Makefile
    process, but may improve computation time in some instances.
    `static-link' only works with ONE Model and AT MOST ONE Model
    Driver (in order to avoid the possibility of symbol
    clashes).


(h) Set override values for other variables, if necessary, by
    adding the desired variable name (after the "# overwrite
    default variable values here" line) and filling in appropriate
    values.

    # overwrite default variable values here

    # overwrite default compiler options

    CC =
    CXX =
    FC =

    # overwrite default compiler option flag lists
    FFLAGS   =
    CCFLAGS  =
    CXXFLAGS =
    LDFLAGS  =

    # overwrite default linker options
    LDSHAREDFLAG =
    LINKSONAME =
    LDWHOLEARCHIVESTARTFLAG =
    LDWHOLEARCHIVEENDFLAG =

    # overwrite default install directories
    package_name =
    prefix =
    libdir =
```

3. In the remainder of the documentation we will use the string $KD to represent the location of the openkim-api source package (the value of the KIM_DIR variable discussed above). If you wish to, type the below commands exactly as written to set the shell variable KD to the appropriate value:

```
bash:
% export KD=$HOME/openkim-api-vX.X.X
```

4. To compile the package, including the provided examples, change to the $KD directory and execute `make examples' and then `make':

```
% cd $KD
% make examples
% make
```

```
This builds all Model Drivers, Models, Tests, and the openkim-api service
routine library.  The targets defined by the Makefile in this directory
include:

  `make'               -- compiles the API and all Models and Tests
  `make all'           -- same as `make'
  `make clean'         -- will remove appropriate .o, .mod, .a, .so and
                          executable files
  `make install'       -- install files to `/usr/local/lib' by default
  `make uninstall'     -- delete files installed by `make install'
  `make openkim-api'   -- compiles only the API
  `make examples'      -- copies examples into the appropriate
                          directories (no overwrite)
  `make examples-force' -- copies examples into the appropriate
                          directories (overwrite)
  `make examples-clean' -- remove all examples from the MODEL_DRIVERS,
                          MODELS, and TESTS directories.
```
5. Verify that the compilation was successful by running a Test.

```
The provided example Tests read in the name of a Model (or Models)
which they use to perform their calculations.  For most Tests the
name of the Model can be piped in using an `echo' command.  For
example, the following Fortran 90 Test reads in one Model:

% cd $KD/TESTs/test_Ar_free_cluster_CLUSTER_F90
% echo "model_Ar_P_MLJ_CLUSTER_C" | ./test_Ar_free_cluster_CLUSTER_F90

 (See the README files in the Test directories for an explanation of what
  the Tests do.)
```
6. Each Test (and Model) has its own make file for compiling and linking. If changes are made to the
   code, re-compile (from the $KD directory).

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the [pair_modify](#) mix, shift, table, and tail options.

This pair style does not write its information to [binary restart files](#), since KIM stores the potential parameters.
Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the
inner, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the KIM package. It is only enabled if LAMMPS was built with that package. See the
[Making LAMMPS](#) section for more info.

This current version of pair_style kim is compatible with the openkim-api package version 1.2.0 and higher.

Related commands:

[pair_coeff](#)

Default: none

LIGGGHTS Users Manual

pair_style lcbop command

Syntax:

```
pair_style lcbop
```

Examples:

```
pair_style lcbop
pair_coeff * * ../potentials/C.lcbop C
```

Description:

The *lcbop* pair style computes the long-range bond-order potential for carbon (LCBOP) of (Los and Fasolino). See section II in that paper for the analytic equations associated with the potential.

Only a single pair_coeff command is used with the *lcbop* style which specifies an LCBOP potential file with parameters for specific elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of LCBOP elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, if your LAMMPS simulation has 4 atom types and you want the 1st 3 to be C you would use the following pair_coeff command:

```
pair_coeff * * C.lcbop C C C NULL
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first C argument maps LAMMPS atom type 1 to the C element in the LCBOP file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *lcbop* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The parameters/coefficients for the LCBOP potential as applied to C are listed in the C.lcbop file to agree with the original (Los and Fasolino) paper. Thus the parameters are specific to this potential and the way it was fit, so modifying the file should be done carefully.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the pair_modify mix, shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair styles is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair potential requires the newton setting to be "on" for pair interactions.

The C.lcbop potential file provided with LAMMPS (see the potentials directory) is parameterized for metal units. You can use the LCBOP potential with any LAMMPS units, but you would need to create your own LCBOP potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_airebo, pair_coeff

Default: none

(Los and Fasolino) J. H. Los and A. Fasolino, Phys. Rev. B 68, 024107 (2003).

pair_style line/lj command

pair_style line/lj/omp command

Syntax:

```
pair_style line/lj cutoff
```

cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style line/lj 3.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.5 2.5
```

Description:

Style *line/lj* treats particles which are line segments as a set of small spherical particles that tile the line segment length as explained below. Interactions between two line segments, each with N1 and N2 spherical particles, are calculated as the pairwise sum of N1*N2 Lennard-Jones interactions. Interactions between a line segment with N spherical particles and a point particle are treated as the pairwise sum of N Lennard-Jones interactions. See the [pair_style lj/cut](#) doc page for the definition of Lennard-Jones interactions.

The cutoff distance for an interaction between 2 line segments, or between a line segment and a point particle, is calculated from the position of the line segment (its center), not between pairs of individual spheres comprising the line segment. Thus an interaction is either calculated in its entirety or not at all.

The set of non-overlapping spherical particles that represent a line segment, for purposes of this pair style, are generated in the following manner. Their size is a function of the line segment length and the specified sigma for that particle type. If a line segment has a length L and is of type I, then the number of spheres N that represent the segment is calculated as N = L/sigma_II, rounded up to an integer value. Thus if L is not evenly divisibly by sigam_II, N is incremented to include one extra sphere. In this case, the spheres must be slightly smaller than sigma_II so as not to overlap, so a new sigma-prime is chosen as the sphere diameter, such that L/N = sigma-prime. Thus the line segment interacts with other segments or point particles as a collection of N spheres of diameter sigma-prime, evenly spaced along the line segment, so as to exactly cover its length.

The LJ interaction between 2 spheres on different line segments of types I,J is computed with an arithmetic mixing of the sigma values of the 2 spheres and using the specified epsilon value for I,J atom types. Note that because the sigma values for line segment spheres is computed using only sigma_II values, specific to the line segment's type, this means that any specified sigma_IJ values (for I != J) are effectively ignored.

For style *line/lj*, the following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Defining particles to be line segments so they participate in line/line or line/particle interactions requires the use the atom_style line command.

Related commands:

pair_coeff, pair_style tri/lj

Default: none

pair_style list command

Syntax:

```
pair_style list listfile cutoff keyword
```

- listfile = name of file with list of pairwise interactions
- cutoff = global cutoff (distance units)
- keyword = optional flag *nocheck* or *check* (default is *check*)

Examples:

```
pair_style list restraints.txt 200.0
pair_coeff * *

pair_style hybrid/overlay lj/cut 1.1225 list pair_list.txt 300.0
pair_coeff * * lj/cut 1.0 1.0
pair_coeff 3* 3* list
```

Description:

Style *list* computes interactions between explicitly listed pairs of atoms with the option to select functional form and parameters for each individual pair. Because the parameters are set in the list file, the pair_coeff command has no parameters (but still needs to be provided). The *check* and *nocheck* keywords enable/disable a test that checks whether all listed bonds were present and computed.

This pair style can be thought of as a hybrid between bonded, non-bonded, and restraint interactions. It will typically be used as an additional interaction within the *hybrid/overlay* pair style. It currently supports three interaction styles: a 12-6 Lennard-Jones, a Morse and a harmonic potential.

The format of the list file is as follows:

- one line per pair of atoms
- empty lines will be ignored
- comment text starts with a '#' character
- line syntax: *ID1 ID2 style coeffs cutoff*

```
ID1 = atom ID of first atom
ID2 = atom ID of second atom
style = style of interaction
coeffs = list of coeffs
cutoff = cutoff for interaction (optional)
```

The cutoff parameter is optional. If not specified, the global cutoff is used.

Here is an example file:

```
# this is a comment

15 259 lj126     1.0 1.0      50.0
15 603 morse    10.0 1.2 2.0  10.0 # and another comment
18 470 harmonic 50.0 1.2       5.0
```

The style *lj126* computes pairwise interactions with the formula

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \qquad r < r_c$$

and the coefficients:

- epsilon (energy units)
- sigma (distance units)

The style *morse* computes pairwise interactions with the formula

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \qquad r < r_c$$

and the coefficients:

- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)

The style *harmonic* computes pairwise interactions with the formula

$$E = K(r - r_0)^2$$

and the coefficients:

- K (energy units)
- r0 (distance units)

Note that the usual 1/2 factor is included in K.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing since all parameters are explicit for each pair.

The pair_modify shift option is supported by this pair style.

The pair_modify table and tail options are not relevant for this pair style.

This pair style does not write its information to binary restart files, so pair_style and pair_coeff commands need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

pair_style list command 836

This pair style does not use a neighbor list and instead identifies atoms by their IDs. This has two consequences: 1) The cutoff has to be chosen sufficiently large, so that the second atom of a pair has to be a ghost atom on the same node on which the first atom is local; otherwise the interaction will be skipped. You can use the *check* option to detect, if interactions are missing. 2) Unlike other pair styles in LAMMPS, an atom I will not interact with multiple images of atom J (assuming the images are within the cutoff distance), but only with the nearest image.

This style is part of the USER-MISC package. It is only enabled if LAMMPS is build with that package. See the Making of LAMMPS section for more info.

Related commands:

pair_coeff, pair_style hybrid/overlay, pair_style lj/cut, pair_style morse, bond_style harmonic

Default: none

pair_style lj96/cut command

pair_style lj96/cut/cuda command

pair_style lj96/cut/gpu command

pair_style lj96/cut/omp command

Syntax:

```
pair_style lj96/cut cutoff
```

- cutoff = global cutoff for lj96/cut interactions (distance units)

Examples:

```
pair_style lj96/cut 2.5
pair_coeff * * 1.0 1.0 4.0
pair_coeff 1 1 1.0 1.0
```

Description:

The *lj96/cut* style compute a 9/6 Lennard-Jones potential, instead of the standard 12/6 potential, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^9 - \left(\frac{\sigma}{r} \right)^6 \right] \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global LJ cutoff specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style supports the pair_modify tail option for adding a long-range tail correction to the energy and pressure of the pair interaction.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style lj/cubic command

pair_style lj/cubic/omp command

Syntax:

```
pair_style lj/cubic
```

Examples:

```
pair_style lj/cubic
pair_coeff * * 1.0 0.8908987
```

Description:

The *lj/cubic* style computes a truncated LJ interaction potential whose energy and force are continuous everywhere. Inside the inflection point the interaction is identical to the standard 12/6 [Lennard-Jones](#) potential. The LJ function outside the inflection point is replaced with a cubic function of distance. The energy, force, and second derivative are continuous at the inflection point. The cubic coefficient A3 is chosen so that both energy and force go to zero at the cutoff distance. Outside the cutoff distance the energy and force are zero.

$$
\begin{aligned}
E &= u_{LJ}(r) & r \leq r_s \\
&= u_{LJ}(r_s) + (r - r_s)u'_{LJ}(r_s) - \frac{1}{6}A_3(r - r_s)^3 & r_s < r \leq r_c \\
&= 0 & r > r_c
\end{aligned}
$$

The location of the inflection point rs is defined by the LJ diameter, rs/sigma = (26/7)^1/6. The cutoff distance is defined by rc/rs = 67/48 or rc/sigma = 1.737.... The analytic expression for the the cubic coefficient A3*rmin^3/epsilon = 27.93... is given in the paper by Holian and Ravelo [(Holian)](#).

This potential is commonly used to study the shock mechanics of FCC solids, as in Ravelo et al. [(Ravelo)](#).

The following coefficients must be defined for each pair of atom types via the [pair_coeff](#) command as in the example above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum, which is located at rmin = 2^(1/6)*sigma. In the above example, sigma = 0.8908987, so rmin = 1.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

The lj/cubic pair style does not support the pair_modify shift option, since pair interaction is already smoothed to 0.0 at the cutoff.

The pair_modify table option is not relevant for this pair style.

The lj/cubic pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure, since there are no corrections for a potential that goes to 0.0 at the cutoff.

The lj/cubic pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The lj/cubic pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

(Holian) Holian and Ravelo, Phys Rev B, 51, 11275 (1995).

(Ravelo) Ravelo, Holian, Germann and Lomdahl, Phys Rev B, 70, 014103 (2004).

pair_style lj/expand command

pair_style lj/expand/cuda command

pair_style lj/expand/gpu command

pair_style lj/expand/omp command

Syntax:

```
pair_style lj/expand cutoff
```

- cutoff = global cutoff for lj/expand interactions (distance units)

Examples:

```
pair_style lj/expand 2.5
pair_coeff * * 1.0 1.0 0.5
pair_coeff 1 1 1.0 1.0 -0.2 2.0
```

Description:

Style *lj/expand* computes a LJ interaction with a distance shifted by delta which can be useful when particles are of different sizes, since it is different that using different sigma values in a standard LJ formula:

$$E = 4\epsilon \left[\left(\frac{\sigma}{r - \Delta} \right)^{12} - \left(\frac{\sigma}{r - \Delta} \right)^{6} \right] \qquad r < r_c + \Delta$$

Rc is the cutoff which does not include the delta distance. I.e. the actual force cutoff is the sum of cutoff + delta.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- delta (distance units)
- cutoff (distance units)

The delta values can be positive or negative. The last coefficient is optional. If not specified, the global LJ cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, and shift coefficients and cutoff distance for this pair style can be mixed. Shift is always mixed via an *arithmetic* rule. The other coefficients are mixed according to the pair_modify mix value. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style supports the pair_modify tail option for adding a long-range tail correction to the energy and pressure of the pair interaction.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style lj/cut command

pair_style lj/cut/cuda command

pair_style lj/cut/experimental/cuda command

pair_style lj/cut/gpu command

pair_style lj/cut/opt command

pair_style lj/cut/omp command

pair_style lj/cut/coul/cut command

pair_style lj/cut/coul/cut/cuda command

pair_style lj/cut/coul/cut/gpu command

pair_style lj/cut/coul/cut/omp command

pair_style lj/cut/coul/debye command

pair_style lj/cut/coul/debye/cuda command

pair_style lj/cut/coul/debye/gpu command

pair_style lj/cut/coul/debye/omp command

pair_style lj/cut/coul/dsf command

pair_style lj/cut/coul/dsf/gpu command

pair_style lj/cut/coul/dsf/omp command

pair_style lj/cut/coul/long command

pair_style lj/cut/coul/long/cuda command

pair_style lj/cut/coul/long/gpu command

pair_style lj/cut/coul/long/opt command

pair_style lj/cut/coul/long/omp command

pair_style lj/cut/coul/msm command

pair_style lj/cut/coul/msm/gpu command

pair_style lj/cut/coul/msm/omp command

pair_style lj/cut/tip4p/cut command

pair_style lj/cut/tip4p/cut/omp command

pair_style lj/cut/tip4p/long command

pair_style lj/cut/tip4p/long/omp command

pair_style lj/cut/tip4p/long/opt command

Syntax:

```
pair_style style args
```

- style = *lj/cut* or *lj/cut/coul/cut* or *lj/cut/coul/debye* or *lj/cut/coul/dsf* or *lj/cut/coul/long* or *lj/cut/coul/msm* or *lj/cut/tip4p/long*
- args = list of arguments for a particular style

```
lj/cut args = cutoff
    cutoff = global cutoff for Lennard Jones interactions (distance units)
  lj/cut/coul/cut args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/debye args = kappa cutoff (cutoff2)
    kappa = inverse of the Debye length (inverse distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/dsf args = alpha cutoff (cutoff2)
    alpha = damping parameter (inverse distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (distance units)
  lj/cut/coul/long args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/coul/msm args = cutoff (cutoff2)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/tip4p/cut args = otype htype btype atype qdist cutoff (cutoff2)
    otype,htype = atom types for TIP4P O and H
    btype,atype = bond and angle types for TIP4P waters
    qdist = distance from O atom to massless charge (distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/tip4p/long args = otype htype btype atype qdist cutoff (cutoff2)
    otype,htype = atom types for TIP4P O and H
    btype,atype = bond and angle types for TIP4P waters
    qdist = distance from O atom to massless charge (distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/cut 2.5
pair_coeff * * 1 1
pair_coeff 1 1 1 1.1 2.8

pair_style lj/cut/coul/cut 10.0
pair_style lj/cut/coul/cut 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
pair_coeff 1 1 100.0 3.5 9.0 9.0

pair_style lj/cut/coul/debye 1.5 3.0
pair_style lj/cut/coul/debye 1.5 2.5 5.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.5 2.5
pair_coeff 1 1 1.0 1.5 2.5 5.0

pair_style lj/cut/coul/dsf 0.05 2.5 10.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.0 2.5

pair_style lj/cut/coul/long 10.0
pair_style lj/cut/coul/long 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0

pair_style lj/cut/coul/msm 10.0
pair_style lj/cut/coul/msm 10.0 8.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0

pair_style lj/cut/tip4p/cut 1 2 7 8 0.15 12.0
pair_style lj/cut/tip4p/cut 1 2 7 8 0.15 12.0 10.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0

pair_style lj/cut/tip4p/long 1 2 7 8 0.15 12.0
pair_style lj/cut/tip4p/long 1 2 7 8 0.15 12.0 10.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
```

Description:

The *lj/cut* styles compute the standard 12/6 Lennard-Jones potential, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \qquad r < r_c$$

Rc is the cutoff.

Style *lj/cut/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

pair_style lj/cut/tip4p/long/opt command

where C is an energy-conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the [dielectric](#) command. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

Style *lj/cut/coul/debye* adds an additional exp() damping factor to the Coulombic term, given by

$$E = \frac{Cq_iq_j}{\epsilon r} \exp(-\kappa r) \qquad r < r_c$$

where kappa is the inverse of the Debye length. This potential is another way to mimic the screening effect of a polar solvent.

Style *lj/cut/coul/dsf* computes the Coulombic term via the damped shifted force model described in [Fennell](#), given by:

$$E = q_iq_j \left[\frac{\text{erfc}(\alpha r)}{r} - \frac{\text{erfc}(\alpha r_c)}{r_c} + \left(\frac{\text{erfc}(\alpha r_c)}{r_c^2} + \frac{2\alpha}{\sqrt{\pi}} \frac{\exp(-\alpha^2 r_c^2)}{r_c} \right) (r - r_c) \right] \qquad r$$

where *alpha* is the damping parameter and erfc() is the complementary error-function. This potential is essentially a short-range, spherically-truncated, charge-neutralized, shifted, pairwise *1/r* summation. The potential is based on Wolf summation, proposed as an alternative to Ewald summation for condensed phase systems where charge screening causes electrostatic interactions to become effectively short-ranged. In order for the electrostatic sum to be absolutely convergent, charge neutralization within the cutoff radius is enforced by shifting the potential through placement of image charges on the cutoff sphere. Convergence can often be improved by setting *alpha* to a small non-zero value.

Styles *lj/cut/coul/long* and *lj/cut/coul/msm* compute the same Coulombic interactions as style *lj/cut/coul/cut* except that an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the [kspace_style](#) command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

Styles *lj/cut/tip4p/cut* and *lj/cut/tip4p/long* implement the TIP4P water model of [(Jorgensen)](#), which introduces a massless site located a short distance away from the oxygen atom along the bisector of the HOH angle. The atomic types of the oxygen and hydrogen atoms, the bond and angle types for OH and HOH interactions, and the distance to the massless charge site are specified as pair_style arguments. Style *lj/cut/tip4p/cut* uses a cutoff for Coulomb interactions; style *lj/cut/tip4p/long* is for use with a long-range Coulombic solver (Ewald or PPPM).

IMPORTANT NOTE: For each TIP4P water molecule in your system, the atom IDs for the O and 2 H atoms must be consecutive, with the O atom first. This is to enable LAMMPS to "find" the 2 H atoms associated with each O atom. For example, if the atom ID of an O atom in a TIP4P water molecule is 500, then its 2 H atoms must have IDs 501 and 502.

See the [howto section](#) for more information on how to use the TIP4P pair styles and lists of parameters to set. Note that the neighobr list cutoff for Coulomb interactions is effectively extended by a distance 2*qdist when using the TIP4P pair style, to account for the offset distance of the fictitious charges on O atoms in water

molecules. Thus it is typically best in an efficiency sense to use a LJ cutoff >= Coulomb cutoff + 2*qdist, to shrink the size of the neighbor list. This leads to slightly larger cost for the long-range calculation, so you can test the trade-off for your model.

For all of the *lj/cut* pair styles, the following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *lj/cut*, since it has no Coulombic terms.

For *lj/cut/coul/long* and *lj/cut/coul/msm* and *lj/cut/tip4p/cut* and *lj/cut/tip4p/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/cut pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

All of the *lj/cut* pair styles support the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction.

The *lj/cut/coul/long* and *lj/cut/tip4p/long* pair styles support the pair_modify table option since they can tabulate the short-range portion of the long-range Coulombic interaction.

All of the *lj/cut* pair styles support the pair_modify tail option for adding a long-range tail correction to the energy and pressure for the Lennard-Jones portion of the pair interaction.

pair_style lj/cut/tip4p/long/opt command   848

All of the *lj/cut* pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The *lj/cut* and *lj/cut/coul/long* pair styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. The other styles only support the *pair* keyword of run_style respa. See the run_style command for details.

Restrictions:

The *lj/cut/coul/long* and *lj/cut/tip4p/long* styles are part of the KSPACE package. The *lj/cut/tip4p/cut* style is part of the MOLECULE package. These styles are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info. Note that the KSPACE and MOLECULE packages are installed by default.

Related commands:

pair_coeff

Default: none

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

(Fennell) C. J. Fennell, J. D. Gezelter, J Chem Phys, 124, 234104 (2006).

LIGGGHTS Users Manual

pair_style lj/long/coul/long command

pair_style lj/long/coul/long/omp command

pair_style lj/long/coul/long/opt command

pair_style lj/long/tip4p/long command

Syntax:

```
pair_style style args
```

> - style = *lj/long/coul/long* or *lj/long/tip4p/long*
> - args = list of arguments for a particular style

```
lj/long/coul/long args = flag_lj flag_coul cutoff (cutoff2)
    flag_lj = long or cut or off
      long = use Kspace long-range summation for dispersion 1/r^6 term
      cut = use a cutoff on dispersion 1/r^6 term
      off = omit disperion 1/r^6 term entirely
    flag_coul = long or off
      long = use Kspace long-range summation for Coulombic 1/r term
      off = omit Coulombic term
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
  lj/cut/tip4p/long args = flag_lj flag_coul otype htype btype atype qdist cutoff (cutoff2)
    flag_lj = long or cut
      long = use Kspace long-range summation for dispersion 1/r^6 term
      cut = use a cutoff
    flag_coul = long or off
      long = use Kspace long-range summation for Coulombic 1/r term
      off = omit Coulombic term
    otype,htype = atom types for TIP4P O and H
    btype,atype = bond and angle types for TIP4P waters
    qdist = distance from O atom to massless charge (distance units)
    cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
    cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/long/coul/long cut off 2.5
pair_style lj/long/coul/long cut long 2.5 4.0
pair_style lj/long/coul/long long long 2.5 4.0
pair_coeff * * 1 1
pair_coeff 1 1 1 3 4

pair_style lj/long/tip4p/long long long 1 2 7 8 0.15 12.0
pair_style lj/long/tip4p/long long long 1 2 7 8 0.15 12.0 10.0
pair_coeff * * 100.0 3.0
pair_coeff 1 1 100.0 3.5 9.0
```

Description:

Style *lj/long/coul/long* computes the standard 12/6 Lennard-Jones and Coulombic potentials, given by

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \qquad r < r_c$$

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy-conversion constant, Qi and Qj are the charges on the 2 atoms, epsilon is the dielectric constant which can be set by the dielectric command, and Rc is the cutoff. If one cutoff is specified in the pair_style command, it is used for both the LJ and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the LJ and Coulombic terms respectively.

The purpose of this pair style is to capture long-range interactions resulting from both attractive 1/r^6 Lennard-Jones and Coulombic 1/r interactions. This is done by use of the *flag_lj* and *flag_coul* settings. The In 't Veld paper has more details on when it is appropriate to include long-range 1/r^6 interactions, using this potential.

Style *lj/cut/tip4p/long* implements the TIP4P water model of (Jorgensen), which introduces a massless site located a short distance away from the oxygen atom along the bisector of the HOH angle. The atomic types of the oxygen and hydrogen atoms, the bond and angle types for OH and HOH interactions, and the distance to the massless charge site are specified as pair_style arguments.

IMPORTANT NOTE: For each TIP4P water molecule in your system, the atom IDs for the O and 2 H atoms must be consecutive, with the O atom first. This is to enable LAMMPS to "find" the 2 H atoms associated with each O atom. For example, if the atom ID of an O atom in a TIP4P water molecule is 500, then its 2 H atoms must have IDs 501 and 502.

See the howto section for more information on how to use the TIP4P pair style. Note that the neighobr list cutoff for Coulomb interactions is effectively extended by a distance 2*qdist when using the TIP4P pair style, to account for the offset distance of the fictitious charges on O atoms in water molecules. Thus it is typically best in an efficiency sense to use a LJ cutoff >= Coulomb cutoff + 2*qdist, to shrink the size of the neighbor list. This leads to slightly larger cost for the long-range calculation, so you can test the trade-off for your model.

If *flag_lj* is set to *long*, no cutoff is used on the LJ 1/r^6 dispersion term. The long-range portion is calculated by using the kspace_style ewald/n command. The specified LJ cutoff then determines which portion of the LJ interactions are computed directly by the pair potential versus which part is computed in reciprocal space via the Kspace style. If *flag_lj* is set to *cut*, the LJ interactions are simply cutoff, as with pair_style lj/cut. If *flag_lj* is set to *off*, LJ interactions are not computed at all.

If *flag_coul* is set to *long*, no cutoff is used on the Coulombic interactions. The long-range portion is calculated by using any style, including *ewald/n* of the kspace_style command. Note that if *flag_lj* is also set to long, then only the *ewald/n* KSpace style can perform the long-range calculations for both the LJ and Coulombic interactions. If *flag_coul* is set to *off*, Coulombic interactions are not computed at all.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)
- cutoff2 (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum at 2^(1/6) sigma.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

Note that if you are using *flag_lj* set to *long*, you cannot specify a LJ cutoff for an atom type pair, since only one global LJ cutoff is allowed. Similarly, if you are using *flag_coul* set to *long*, you cannot specify a Coulombic cutoff for an atom type pair, since only one global Coulombic cutoff is allowed.

For *lj/long/tip4p/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/long pair styles can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

These pair styles support the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction, assuming *flag_lj* is *cut*.

These pair styles support the pair_modify table and table/disp options since they can tabulate the short-range portion of the long-range Coulombic and dispersion interactions.

Thes pair styles do not support the pair_modify tail option for adding a long-range tail correction to the Lennard-Jones portion of the energy and pressure.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The pair lj/long/coul/long styles support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the run_style command for details.

Restrictions:

These styles are part of the KSPACE package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info. Note that the KSPACE package is installed by default.

Related commands:

pair_coeff

Default: none

(In 't Veld) In 't Veld, Ismail, Grest, J Chem Phys (accepted) (2007).

pair_style lj/sf command

pair_style lj/sf/omp command

Syntax:

```
pair_style lj/sf cutoff
```

- cutoff = global cutoff for Lennard-Jones interactions (distance units)

Examples:

```
pair_style lj/sf 2.5
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.0 3.0
```

Description:

Style *lj/sf* computes a truncated and force-shifted LJ interaction (Shifted Force Lennard-Jones), so that both the potential and the force go continuously to zero at the cutoff [(Toxvaerd)](#):

$$
\begin{aligned}
F &= F_{\mathrm{LJ}}(r) - F_{\mathrm{LJ}}(r_c) \qquad r < r_c \\
E &= E_{\mathrm{LJ}}(r) - E_{\mathrm{LJ}}(r_c) + (r - r_c)F_{\mathrm{LJ}}(r_c) \qquad r < r_c
\end{aligned}
$$

$$
\text{with} \qquad E_{\mathrm{LJ}}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \qquad \text{and} \qquad F_{\mathrm{LJ}}(r) = -E'_{\mathrm{LJ}}(r)
$$

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global LJ cutoff specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your

input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for this pair style can be mixed. Rin is a cutoff value and is mixed like the cutoff. The default mix value is *geometric*. See the "pair_modify" command for details.

The [pair_modify](#) shift option is not relevant for this pair style, since the pair interaction goes to 0.0 at the cutoff.

The [pair_modify](#) table option is not relevant for this pair style.

This pair style does not support the [pair_modify](#) tail option for adding long-range tail corrections to energy and pressure, since the energy of the pair interaction is smoothed to 0.0 at the cutoff.

This pair style writes its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-MISC package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#)

Default: none

(Toxvaerd) Toxvaerd, Dyre, J Chem Phys, 134, 081102 (2011).

pair_style lj/smooth command

pair_style lj/smooth/cuda command

pair_style lj/smooth/omp command

Syntax:

```
pair_style lj/smooth Rin Rc
```

- Rin = inner cutoff beyond which force smoothing will be applied (distance units)
- Rc = outer cutoff for lj/smooth interactions (distance units)

Examples:

```
pair_style lj/smooth 8.0 10.0
pair_coeff * * 10.0 1.5
pair_coeff 1 1 20.0 1.3 7.0 9.0
```

Description:

Style *lj/smooth* computes a LJ interaction with a force smoothing applied between the inner and outer cutoff.

$$E = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right] \qquad r < r_{in}$$

$$F = C_1 + C_2(r - r_{in}) + C_3(r - r_{in})^2 + C_4(r - r_{in})^3 \qquad r_{in} < r < r_c$$

The polynomial coefficients C1, C2, C3, C4 are computed by LAMMPS to cause the force to vary smoothly from the inner cutoff Rin to the outer cutoff Rc.

At the inner cutoff the force and its 1st derivative will match the unsmoothed LJ formula. At the outer cutoff the force and its 1st derivative will be 0.0. The inner cutoff cannot be 0.0.

IMPORTANT NOTE: this force smoothing causes the energy to be discontinuous both in its values and 1st derivative. This can lead to poor energy conservation and may require the use of a thermostat. Plot the energy and force resulting from this formula via the pair_write command to see the effect.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- innner (distance units)
- outer (distance units)

The last 2 coefficients are optional inner and outer cutoffs. If not specified, the global values for Rin and Rc are used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon, sigma, Rin coefficients and the cutoff distance for this pair style can be mixed. Rin is a cutoff value and is mixed like the cutoff. The other coefficients are mixed according to the pair_modify mix option. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure, since the energy of the pair interaction is smoothed to 0.0 at the cutoff.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff, pair lj/smooth/linear

Default: none

pair_style lj/smooth/linear command

pair_style lj/smooth/linear/omp command

Syntax:

```
pair_style lj/smooth/linear Rc
```

- Rc = cutoff for lj/smooth/linear interactions (distance units)

Examples:

```
pair_style lj/smooth/linear 5.456108274435118
pair_coeff * * 0.7242785984051078 2.598146797350056
pair_coeff 1 1 20.0 1.3 9.0
```

Description:

Style *lj/smooth/linear* computes a LJ interaction that combines the standard 12/6 Lennard-Jones function and subtracts a linear term that includes the cutoff distance Rc, as in this formula:

$$
\begin{aligned}
\phi(r) &= 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \\
E(r) &= \phi(r) - \phi(R_c) - (r - R_c)\left.\frac{d\phi}{dr}\right|_{r=R_c} \qquad r < R_c
\end{aligned}
$$

At the cutoff Rc, the energy and force (its 1st derivative) will be 0.0.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global value for Rc is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance can be mixed. The default mix value is geometric. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure, since the energy of the pair interaction is smoothed to 0.0 at the cutoff.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff, pair lj/smooth

Default: none

pair_style lubricate command

pair_style lubricate/omp command

pair_style lubricate/poly command

pair_style lubricate/poly/omp command

Syntax:

```
pair_style style mu flaglog flagfld cutinner cutoff flagHI flagVF
```

- style = *lubricate* or *lubricate/poly*
- mu = dynamic viscosity (dynamic viscosity units)
- flaglog = 0/1 to exclude/include log terms in the lubrication approximation
- flagfld = 0/1 to exclude/include Fast Lubrication Dynamics (FLD) effects
- cutinner = inner cutoff distance (distance units)
- cutoff = outer cutoff for interactions (distance units)
- flagHI (optional) = 0/1 to exclude/include 1/r hydrodynamic interactions
- flagVF (optional) = 0/1 to exclude/include volume fraction corrections in the long-range isotropic terms

Examples: (all assume radius = 1)

```
pair_style lubricate 1.5 1 1 2.01 2.5
pair_coeff 1 1 2.05 2.8
pair_coeff * *

pair_style lubricate 1.5 1 1 2.01 2.5
pair_coeff * *
variable mu equal ramp(1,2)
fix 1 all adapt 1 pair lubricate mu * * v_mu
```

Description:

Styles *lubricate* and *lubricate/poly* compute hydrodynamic interactions between mono-disperse spherical particles in a pairwise fashion. The interactions have 2 components. The first is Ball-Melrose lubrication terms via the formulas in [(Ball and Melrose)](#)

$$
\begin{aligned}
W \;=\; &-a_{sq}|(v_1 - v_2)\bullet\mathbf{nn}|^2 - a_{sh}|(\omega_1 + \omega_2)\bullet(\mathbf{I} - \mathbf{nn}) - 2\Omega_N|^2 - \\
&a_{pu}|(\omega_1 - \omega_2)\bullet(\mathbf{I} - \mathbf{nn})|^2 - a_{tw}|(\omega_1 - \omega_2)\bullet\mathbf{nn}|^2 \qquad r < r_c
\end{aligned}
$$

$$
\Omega_N = \mathbf{n} \times (v_1 - v_2)/r
$$

which represents the dissipation W between two nearby particles due to their relative velocities in the presence of a background solvent with viscosity *mu*. Note that this is dynamic viscosity which has units of mass/distance/time, not kinematic viscosity.

The Asq (squeeze) term is the strongest and is included if *flagHI* is set to 1 (default). It scales as 1/gap where gap is the separation between the surfaces of the 2 particles. The Ash (shear) and Apu (pump) terms are only included if *flaglog* is set to 1. They are the next strongest interactions, and the only other singular interaction, and scale as log(gap). Note that *flaglog* = 1 and *flagHI* = 0 is invalid, and will result in a warning message, after which *flagHI* will be set to 1. The Atw (twist) term is currently not included. It is typically a very small contribution to the lubrication forces.

The *flagHI* and *flagVF* settings are optional. Neither should be used, or both must be defined.

Cutinner sets the minimum center-to-center separation that will be used in calculations irrespective of the actual separation. *Cutoff* is the maximum center-to-center separation at which an interaction is computed. Using a *cutoff* less than 3 radii is recommended if *flaglog* is set to 1.

The other component is due to the Fast Lubrication Dynamics (FLD) approximation, described in (Kumar), which can be represented by the following equation

$$F^H = -R_{FU}(U - U^\infty) + R_{FE}E^\infty$$

where U represents the velocities and angular velocities of the particles, U^*infty* represents the velocity and the angular velocity of the undisturbed fluid, and E^*infty* represents the rate of strain tensor of the undisturbed fluid with viscosity *mu*. Again, note that this is dynamic viscosity which has units of mass/distance/time, not kinematic viscosity. Volume fraction corrections to R_FU are included as long as *flagVF* is set to 1 (default).

IMPORTANT NOTE: When using the FLD terms, these pair styles are designed to be used with explicit time integration and a correspondingly small timestep. Thus either fix nve/sphere or fix nve/asphere should be used for time integration. To perform implicit FLD, see the pair_style lubricateU command.

Style *lubricate* requires monodisperse spherical particles; style *lubricate/poly* allows for polydisperse spherical particles.

The viscosity *mu* can be varied in a time-dependent manner over the course of a simluation, in which case in which case the pair_style setting for *mu* will be overridden. See the fix adapt command for details.

If the suspension is sheared via the fix deform command then the pair style uses the shear rate to adjust the hydrodynamic interactions accordingly. Volume changes due to fix deform are accounted for when computing the volume fraction corrections to R_FU.

When computing the volume fraction corrections to R_FU, the presence of walls (whether moving or stationary) will affect the volume fraction available to colloidal particles. This is currently accounted for with the following types of walls: wall/lj93, wall/lj126, wall/colloid, and wall/harmonic. For these wall styles, the correct volume fraction will be used when walls do not coincide with the box boundary, as well as when walls move and thereby cause a change in the volume fraction. Other wall styles will still work, but they will result in the volume fraction being computed based on the box boundaries.

Since lubrication forces are dissipative, it is usually desirable to thermostat the system at a constant temperature. If Brownian motion (at a constant temperature) is desired, it can be set using the pair_style brownian command. These pair styles and the brownian style should use consistent parameters for *mu*, *flaglog*, *flagfld*, *cutinner*, *cutoff*, *flagHI* and *flagVF*.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutinner (distance units)
- cutoff (distance units)

The two coefficients are optional. If neither is specified, the two cutoffs specified in the pair_style command are used. Otherwise both must be specified.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in this section of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See this section of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the two cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These styles are part of the FLD package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Only spherical monodisperse particles are allowed for pair_style lubricate.

Only spherical particles are allowed for pair_style lubricate/poly.

These pair styles will not restart exactly when using the read_restart command, though they should provide statistically similar results. This is because the forces they compute depend on atom velocities. See the read_restart command for more details.

Related commands:

pair_style lubricate/poly/omp command 862

pair_coeff, pair_style lubricateU

Default:

The default settings for the optional args are flagHI = 1 and flagVF = 1.

(Ball) Ball and Melrose, Physica A, 247, 444-472 (1997).

(Kumar) Kumar and Higdon, Phys Rev E, 82, 051401 (2010). See also his thesis for more details: A. Kumar, "Microscale Dynamics in Suspensions of Non-spherical Particles", Thesis, University of Illinois Urbana-Champaign, (2010). (https://www.ideals.illinois.edu/handle/2142/16032)

pair_style lubricateU command

pair_style lubricateU/poly command

Syntax:

```
pair_style style mu flaglog cutinner cutoff gdot flagHI flagVF
```

- style = *lubricateU* or *lubricateU/poly*
- mu = dynamic viscosity (dynamic viscosity units)
- flaglog = 0/1 to exclude/include log terms in the lubrication approximation
- cutinner = inner cut off distance (distance units)
- cutoff = outer cutoff for interactions (distance units)
- gdot = shear rate (1/time units)
- flagHI (optional) = 0/1 to exclude/include 1/r hydrodynamic interactions
- flagVF (optional) = 0/1 to exclude/include volume fraction corrections in the long-range isotropic terms

Examples: (all assume radius = 1)

```
pair_style lubricateU 1.5 1 2.01 2.5 0.01 1 1
pair_coeff 1 1 2.05 2.8
pair_coeff * *
```

Description:

Styles *lubricateU* and *lubricateU/poly* compute velocities and angular velocities such that the hydrodynamic interaction balances the force and torque due to all other types of interactions.

The interactions have 2 components. The first is Ball-Melrose lubrication terms via the formulas in (Ball and Melrose)

$$
\begin{aligned}
W \;=\;\; & -a_{sq}|(v_1 - v_2) \bullet \mathbf{nn}|^2 - a_{sh}|(\omega_1 + \omega_2) \bullet (\mathbf{I} - \mathbf{nn}) - 2\Omega_N|^2 - \\
& a_{pu}|(\omega_1 - \omega_2) \bullet (\mathbf{I} - \mathbf{nn})|^2 - a_{tw}|(\omega_1 - \omega_2) \bullet \mathbf{nn}|^2 \qquad r < r_c
\end{aligned}
$$

$$
\Omega_N = \mathbf{n} \times (v_1 - v_2)/r
$$

which represents the dissipation W between two nearby particles due to their relative velocities in the presence of a background solvent with viscosity *mu*. Note that this is dynamic viscosity which has units of mass/distance/time, not kinematic viscosity.

The Asq (squeeze) term is the strongest and is included as long as *flagHI* is set to 1 (default). It scales as 1/gap where gap is the separation between the surfaces of the 2 particles. The Ash (shear) and Apu (pump) terms are only included if *flaglog* is set to 1. They are the next strongest interactions, and the only other singular interaction, and scale as log(gap). Note that *flaglog* = 1 and *flagHI* = 0 is invalid, and will result in a warning message, after which *flagHI* will be set to 1. The Atw (twist) term is currently not included. It is typically a very small contribution to the lubrication forces.

The *flagHI* and *flagVF* settings are optional. Neither should be used, or both must be defined.

Cutinner sets the minimum center-to-center separation that will be used in calculations irrespective of the actual separation. *Cutoff* is the maximum center-to-center separation at which an interaction is computed. Using a *cutoff* less than 3 radii is recommended if *flaglog* is set to 1.

The other component is due to the Fast Lubrication Dynamics (FLD) approximation, described in [(Kumar)](). The equation being solved to balance the forces and torques is

$$-R_{FU}(U - U^\infty) = -R_{FE}E^\infty - F^{rest}$$

where U represents the velocities and angular velocities of the particles, U^*infty* represents the velocities and the angular velocities of the undisturbed fluid, and E^*infty* represents the rate of strain tensor of the undisturbed fluid flow with viscosity *mu*. Again, note that this is dynamic viscosity which has units of mass/distance/time, not kinematic viscosity. Volume fraction corrections to R_FU are included if *flagVF* is set to 1 (default).

F*rest* represents the forces and torques due to all other types of interactions, e.g. Brownian, electrostatic etc. Note that this algorithm neglects the inertial terms, thereby removing the restriction of resolving the small interial time scale, which may not be of interest for colloidal particles. This pair style solves for the velocity such that the hydrodynamic force balances all other types of forces, thereby resulting in a net zero force (zero inertia limit). When defining this pair style, it must be defined last so that when this style is invoked all other types of forces have already been computed. For the same reason, it won't work if additional non-pair styles are defined (such as bond or Kspace forces) as they are calculated in LAMMPS after the pairwise interactions have been computed.

IMPORTANT NOTE: When using these styles, the these pair styles are designed to be used with implicit time integration and a correspondingly larger timestep. Thus either [fix nve/noforce]() should be used for spherical particles defined via [atom_style sphere]() or [fix nve/asphere/noforce]() should be used for spherical particles defined via [atom_style ellipsoid](). This is because the velocity and angular momentum of each particle is set by the pair style, and should not be reset by the time integration fix.

Style *lubricateU* requires monodisperse spherical particles; style *lubricateU/poly* allows for polydisperse spherical particles.

If the suspension is sheared via the [fix deform]() command then the pair style uses the shear rate to adjust the hydrodynamic interactions accordingly. Volume changes due to fix deform are accounted for when computing the volume fraction corrections to R_FU.

When computing the volume fraction corrections to R_FU, the presence of walls (whether moving or stationary) will affect the volume fraction available to colloidal particles. This is currently accounted for with the following types of walls: [wall/lj93](), [wall/lj126](), [wall/colloid](), and "wall/harmonic_fix_wall.html". For these wall styles, the correct volume fraction will be used when walls do not coincide with the box boundary, as well as when walls move and thereby cause a change in the volume fraction. To use these wall styles with pair_style *lubricateU* or *lubricateU/poly*, the *fld yes* option must be specified in the fix wall command.

Since lubrication forces are dissipative, it is usually desirable to thermostat the system at a constant temperature. If Brownian motion (at a constant temperature) is desired, it can be set using the [pair_style brownian]() command. These pair styles and the brownian style should use consistent parameters for *mu*, *flaglog*, *flagfld*, *cutinner*, *cutoff*, *flagHI* and *flagVF*.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cutinner (distance units)
- cutoff (distance units)

The two coefficients are optional. If neither is specified, the two cutoffs specified in the pair_style command are used. Otherwise both must be specified.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the two cutoff distances for this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

These styles are part of the FLD package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Currently, these pair styles assume that all other types of forces/torques on the particles have been already been computed when it is invoked. This requires this style to be defined as the last of the pair styles, and that no fixes apply additional constraint forces. One exception is the fix wall/colloid commands, which has an "fld" option to apply their wall forces correctly.

Only spherical monodisperse particles are allowed for pair_style lubricateU.

Only spherical particles are allowed for pair_style lubricateU/poly.

For sheared suspensions, it is assumed that the shearing is done in the xy plane, with x being the velocity direction and y being the velocity-gradient direction. In this case, one must use fix deform with the same rate of shear (erate).

Related commands:

pair_coeff, pair_style lubricate

Default:

The default settings for the optional args are flagHI = 1 and flagVF = 1.

---

pair_style lubricateU/poly command   866

(Ball) Ball and Melrose, Physica A, 247, 444-472 (1997).

(Kumar) Kumar and Higdon, Phys Rev E, 82, 051401 (2010).

pair_style meam command

Syntax:

```
pair_style meam
```

Examples:

```
pair_style meam
pair_coeff * * ../potentials/library.meam Si ../potentials/si.meam Si
pair_coeff * * ../potentials/library.meam Ni Al NULL Ni Al Ni Ni
```

Description:

NOTE: The behavior of the MEAM potential for alloy systems has changed as of November 2010; see description below of the mixture_ref_t parameter

Style *meam* computes pairwise interactions for a variety of materials using modified embedded-atom method (MEAM) potentials (Baskes). Conceptually, it is an extension to the original EAM potentials which adds angular forces. It is thus suitable for modeling metals and alloys with fcc, bcc, hcp and diamond cubic structures, as well as covalently bonded materials like silicon and carbon.

In the MEAM formulation, the total energy E of a system of atoms is given by:

$$E = \sum_i \left\{ F_i(\bar{\rho}_i) + \frac{1}{2} \sum_{i \neq j} \phi_{ij}(r_{ij}) \right\}$$

where F is the embedding energy which is a function of the atomic electron density rho, and phi is a pair potential interaction. The pair interaction is summed over all neighbors J of atom I within the cutoff distance. As with EAM, the multi-body nature of the MEAM potential is a result of the embedding energy term. Details of the computation of the embedding and pair energies, as implemented in LAMMPS, are given in (Gullet) and references therein.

The various parameters in the MEAM formulas are listed in two files which are specified by the pair_coeff command. These are ASCII text files in a format consistent with other MD codes that implement MEAM potentials, such as the serial DYNAMO code and Warp. Several MEAM potential files with parameters for different materials are included in the "potentials" directory of the LAMMPS distribution with a ".meam" suffix. All of these are parameterized in terms of LAMMPS metal units.

Note that unlike for other potentials, cutoffs for MEAM potentials are not set in the pair_style or pair_coeff command; they are specified in the MEAM potential files themselves.

Only a single pair_coeff command is used with the *meam* style which specifies two MEAM files and the element(s) to extract information for. The MEAM elements are mapped to LAMMPS atom types by specifying N additional arguments after the 2nd filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- MEAM library file

pair_style meam command   868

- Elem1, Elem2, ...
- MEAM parameter file
- N element names = mapping of MEAM elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential files.

As an example, the potentials/library.meam file has generic MEAM settings for a variety of elements. The potentials/sic.meam file has specific parameter settings for a Si and C alloy system. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * library.meam Si C sic.meam Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The two filenames are for the library and parameter file respectively. The Si and C arguments (between the file names) are the two elements for which info will be extracted from the library file. The first three trailing Si arguments map LAMMPS atom types 1,2,3 to the MEAM Si element. The final C argument maps LAMMPS atom type 4 to the MEAM C element.

If the 2nd filename is specified as NULL, no parameter file is read, which simply means the generic parameters in the library file are used. Use of the NULL specification for the parameter file is discouraged for systems with more than a single element type (e.g. alloys), since the parameter file is expected to set element interaction terms that are not captured by the information in the library file.

If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *meam* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

The MEAM library file provided with LAMMPS has the name potentials/library.meam. It is the "meamf" file used by other MD codes. Aside from blank and comment lines (start with #) which can appear anywhere, it is formatted as a series of entries, each of which has 19 parameters and can span multiple lines:

elt, lat, z, ielement, atwt, alpha, b0, b1, b2, b3, alat, esub, asub, t0, t1, t2, t3, rozero, ibar

The "elt" and "lat" parameters are text strings, such as elt = Si or Cu and lat = dia or fcc. Because the library file is used by Fortran MD codes, these strings may be enclosed in single quotes, but this is not required. The other numeric parameters match values in the formulas above. The value of the "elt" string is what is used in the pair_coeff command to identify which settings from the library file you wish to read in. There can be multiple entries in the library file with the same "elt" value; LAMMPS reads the 1st matching entry it finds and ignores the rest.

Other parameters in the MEAM library file correspond to single-element potential parameters:

```
lat      = lattice structure of reference configuration
z        = number of nearest neighbors in the reference structure
ielement = atomic number
atwt     = atomic weight
alat     = lattice constant of reference structure
esub     = energy per atom (eV) in the reference structure at equilibrium
asub     = "A" parameter for MEAM (see e.g. (Baskes))
```

The alpha, b0, b1, b2, b3, t0, t1, t2, t3 parameters correspond to the standard MEAM parameters in the literature (Baskes) (the b parameters are the standard beta parameters). The rozero parameter is an element-dependent density scaling that weights the reference background density (see e.g. equation 4.5 in (Gullet)) and is typically 1.0 for single-element systems. The ibar parameter selects the form of the function G(Gamma) used to compute the electron density; options are

```
    0 => G = sqrt(1+Gamma)
    1 => G = exp(Gamma/2)
    2 => not implemented
    3 => G = 2/(1+exp(-Gamma))
    4 => G = sqrt(1+Gamma)
   -5 => G = +-sqrt(abs(1+Gamma))
```

If used, the MEAM parameter file contains settings that override or complement the library file settings. Examples of such parameter files are in the potentials directory with a ".meam" suffix. Their format is the same as is read by other Fortran MD codes. Aside from blank and comment lines (start with #) which can appear anywhere, each line has one of the following forms. Each line can also have a trailing comment (starting with #) which is ignored.

```
keyword = value
keyword(I) = value
keyword(I,J) = value
keyword(I,J,K) = value
```

The recognized keywords are as follows:

Ec, alpha, rho0, delta, lattce, attrac, repuls, nn2, Cmin, Cmax, rc, delr, augt1, gsmooth_factor, re

where

```
rc          = cutoff radius for cutoff function; default = 4.0
delr        = length of smoothing distance for cutoff function; default = 0.1
rho0(I)     = relative density for element I (overwrites value
                read from meamf file)
Ec(I,J)     = cohesive energy of reference structure for I-J mixture
delta(I,J)  = heat of formation for I-J alloy; if Ec_IJ is input as
                zero, then LAMMPS sets Ec_IJ = (Ec_II + Ec_JJ)/2 - delta_IJ
alpha(I,J)  = alpha parameter for pair potential between I and J (can
                be computed from bulk modulus of reference structure
re(I,J)     = equilibrium distance between I and J in the reference
                structure
Cmax(I,J,K) = Cmax screening parameter when I-J pair is screened
                by K (I<=J); default = 2.8
Cmin(I,J,K) = Cmin screening parameter when I-J pair is screened
                by K (I<=J); default = 2.0
lattce(I,J) = lattice structure of I-J reference structure:
                  dia = diamond (interlaced fcc for alloy)
                  fcc = face centered cubic
                  bcc = body centered cubic
                  dim = dimer
                  b1  = rock salt (NaCl structure)
                  hcp = hexagonal close-packed
                  c11 = MoSi2 structure
                  l12 = Cu3Au structure (lower case L, followed by 12)
                  b2  = CsCl structure (interpenetrating simple cubic)
nn2(I,J)    = turn on second-nearest neighbor MEAM formulation for
                I-J pair (see for example (Lee)).
                  0 = second-nearest neighbor formulation off
                  1 = second-nearest neighbor formulation on
                  default = 0
attrac(I,J) = additional cubic attraction term in Rose energy I-J pair potential
                  default = 0
repuls(I,J) = additional cubic repulsive term in Rose energy I-J pair potential
                  default = 0
zbl(I,J)    = blend the MEAM I-J pair potential with the ZBL potential for small
                atom separations (ZBL)
                  default = 1
gsmooth_factor = factor determining the length of the G-function smoothing
                    region; only significant for ibar=0 or ibar=4.
                      99.0 = short smoothing region, sharp step
```

```
                         0.5  = long smoothing region, smooth step
                         default = 99.0
augt1             = integer flag for whether to augment t1 parameter by
                    3/5*t3 to account for old vs. new meam formulations;
                       0 = don't augment t1
                       1 = augment t1
                       default = 1
ialloy            = integer flag to use alternative averaging rule for t parameters,
                    for comparison with the DYNAMO MEAM code
                       0 = standard averaging (matches ialloy=0 in DYNAMO)
                       1 = alternative averaging (matches ialloy=1 in DYNAMO)
                       2 = no averaging of t (use single-element values)
                       default = 0
mixture_ref_t     = integer flag to use mixture average of t to compute the background
                    reference density for alloys, instead of the single-element values
                    (see description and warning elsewhere in this doc page)
                       0 = do not use mixture averaging for t in the reference density
                       1 = use mixture averaging for t in the reference density
                       default = 0
erose_form        = integer value to select the form of the Rose energy function
                    (see description below).
                       default = 0
emb_lin_neg       = integer value to select embedding function for negative densities
                       0 = F(rho)=0
                       1 = F(rho) = -asub*esub*rho (linear in rho, matches DYNAMO)
                       default = 0
bkgd_dyn          = integer value to select background density formula
                       0 = rho_bkgd = rho_ref_meam(a) (as in the reference structure)
                       1 = rho_bkgd = rho0_meam(a)*Z_meam(a) (matches DYNAMO)
                       default = 0
```

Rc, delr, re are in distance units (Angstroms in the case of metal units). Ec and delta are in energy units (eV in the case of metal units).

Each keyword represents a quantity which is either a scalar, vector, 2d array, or 3d array and must be specified with the correct corresponding array syntax. The indices I,J,K each run from 1 to N where N is the number of MEAM elements being used.

Thus these lines

```
rho0(2) = 2.25
alpha(1,2) = 4.37
```

set rho0 for the 2nd element to the value 2.25 and set alpha for the alloy interaction between elements 1 and 2 to 4.37.

The augt1 parameter is related to modifications in the MEAM formulation of the partial electron density function. In recent literature, an extra term is included in the expression for the third-order density in order to make the densities orthogonal (see for example [(Wang)](#), equation 3d); this term is included in the MEAM implementation in lammps. However, in earlier published work this term was not included when deriving parameters, including most of those provided in the library.meam file included with lammps, and to account for this difference the parameter t1 must be augmented by 3/5*t3. If augt1=1, the default, this augmentation is done automatically. When parameter values are fit using the modified density function, as in more recent literature, augt1 should be set to 0.

The mixture_ref_t parameter is available to match results with those of previous versions of lammps (before January 2011). Newer versions of lammps, by default, use the single-element values of the t parameters to compute the background reference density. This is the proper way to compute these parameters. Earlier versions of lammps used an alloy mixture averaged value of t to compute the background reference density. Setting mixture_ref_t=1 gives the old behavior. WARNING: using mixture_ref_t=1 will give results that are

pair_style meam command   871

demonstrably incorrect for second-neighbor MEAM, and non-standard for first-neighbor MEAM; this option is included only for matching with previous versions of lammps and should be avoided if possible.

The parameters attrac and repuls, along with the integer selection parameter erose_form, can be used to modify the Rose energy function used to compute the pair potential. This function gives the energy of the reference state as a function of interatomic spacing. The form of this function is:

```
astar = alpha * (r/re - 1.d0)
if erose_form = 0: erose = -Ec*(1+astar+a3*(astar**3)/(r/re))*exp(-astar)
if erose_form = 1: erose = -Ec*(1+astar+(-attrac+repuls/r)*(astar**3))*exp(-astar)
if erose_form = 2: erose = -Ec*(1 +astar + a3*(astar**3))*exp(-astar)
a3 = repuls, astar <0
a3 = attrac, astar >= 0
```

Most published MEAM parameter sets use the default values attrac=repulse=0. Setting repuls=attrac=delta corresponds to the form used in several recent published MEAM parameter sets, such as (Vallone)

NOTE: The default form of the erose expression in LAMMPS was corrected in March 2009. The current version is correct, but may show different behavior compared with earlier versions of lammps with the attrac and/or repuls parameters are non-zero. To obtain the previous default form, use erose_form = 1 (this form does not seem to appear in the literature). An alternative form (see e.g. (Lee2)) is available using erose_form = 2.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS with user-specifiable parameters as described above. You never need to specify a pair_coeff command with I != J arguments for this style.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the MEAM package. It is only enabled if LAMMPS was built with that package, which also requires the MEAM library be built and linked with LAMMPS. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style eam, pair_style meam/spline

Default: none

(Baskes) Baskes, Phys Rev B, 46, 2727-2742 (1992).

(Gullet) Gullet, Wagner, Slepoy, SANDIA Report 2003-8782 (2003). This report may be accessed on-line via this link.

(Lee) Lee, Baskes, Phys. Rev. B, 62, 8564-8567 (2000).

(Lee2) Lee, Baskes, Kim, Cho. Phys. Rev. B, 64, 184102 (2001).

(Valone) Valone, Baskes, Martin, Phys. Rev. B, 73, 214209 (2006).

(Wang) Wang, Van Hove, Ross, Baskes, J. Chem. Phys., 121, 5410 (2004).

(ZBL) J.F. Ziegler, J.P. Biersack, U. Littmark, "Stopping and Ranges of Ions in Matter", Vol 1, 1985, Pergamon Press.

pair_style meam/spline

pair_style meam/spline/omp

Syntax:

```
pair_style meam/spline
```

Examples:

```
pair_style meam/spline
pair_coeff * * Ti.meam.spline Ti
pair_coeff * * Ti.meam.spline Ti Ti Ti
```

Description:

The *meam/spline* style computes pairwise interactions for metals using a variant of modified embedded-atom method (MEAM) potentials (Lenosky). The total energy E is given by

$$E = \sum_{ij} \phi(r_{ij}) + \sum_{i} U(\rho_i),$$

$$\rho_i = \sum_{j} \rho(r_{ij}) + \sum_{jk} f(r_{ij}) f(r_{ik}) g[\cos(\theta_{jik})]$$

where rho_i is the density at atom I, theta_jik is the angle between atoms J, I, and K centered on atom I. The five functions Phi, U, rho, f, and g are represented by cubic splines.

The cutoffs and the coefficients for these spline functions are listed in a parameter file which is specified by the pair_coeff command. Parameter files for different elements are included in the "potentials" directory of the LAMMPS distribution and have a ".meam.spline" file suffix. All of these files are parameterized in terms of LAMMPS metal units.

Note that unlike for other potentials, cutoffs for spline-based MEAM potentials are not set in the pair_style or pair_coeff command; they are specified in the potential files themselves.

Unlike the EAM pair style, which retrieves the atomic mass from the potential file, the spline-based MEAM potentials do not include mass information; thus you need to use the mass command to specify it.

Only a single pair_coeff command is used with the *meam/spline* style which specifies a potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of spline-based MEAM elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, imagine the Ti.meam.spline file has values for Ti. If your LAMMPS simulation has 3 atoms types and they are all to be treated with this potentials, you would use the following pair_coeff command:

```
pair_coeff * * Ti.meam.spline Ti Ti Ti
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The three Ti arguments map LAMMPS atom types 1,2,3 to the Ti element in the potential file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *meam/spline* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: The *meam/spline* style currently supports only single-element MEAM potentials. It may be extended for alloy systems in the future.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

The current version of this pair style does not support multiple element types or mixing. It has been designed for pure elements only.

This pair style does not support the pair_modify shift, table, and tail options.

The *meam/spline* pair style does not write its information to binary restart files, since it is stored in an external potential parameter file. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

The *meam/spline* pair style can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style requires the newton setting to be "on" for pair interactions.

This pair style is only enabled if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style meam

Default: none

pair_style meam/spline/omp

(Lenosky) Lenosky, Sadigh, Alonso, Bulatov, de la Rubia, Kim, Voter, Kress, Modelling Simulation Materials Science Enginerring, 8, 825 (2000).

pair_style meam/sw/spline

pair_style meam/sw/spline/omp

Syntax:

```
pair_style meam/sw/spline
```

Examples:

```
pair_style meam/sw/spline
pair_coeff * * Ti.meam.sw.spline Ti
pair_coeff * * Ti.meam.sw.spline Ti Ti Ti
```

Description:

The *meam/sw/spline* style computes pairwise interactions for metals using a variant of modified embedded-atom method (MEAM) potentials (Lenosky) with an additional Stillinger-Weber (SW) term (Stillinger) in the energy. This form of the potential was first proposed by Nicklas, Fellinger, and Park (Nicklas). We refer to it as MEAM+SW. The total energy E is given by

$$
\begin{aligned}
E &= E_{MEAM} + E_{SW} \\
E_{MEAM} &= \sum_{IJ} \phi(r_{IJ}) + \sum_{I} U(\rho_I) \\
E_{SW} &= \sum_{I} \sum_{JK} F(r_{IJ}) \, F(r_{IK}) \, G(\cos(\theta_{JIK})) \\
\rho_I &= \sum_{J} \rho(r_{IJ}) + \sum_{JK} f(r_{IJ}) \, f(r_{IK}) \, g(\cos(\theta_{JIK}))
\end{aligned}
$$

where rho_I is the density at atom I, theta_JIK is the angle between atoms J, I, and K centered on atom I. The seven functions Phi, F, G, U, rho, f, and g are represented by cubic splines.

The cutoffs and the coefficients for these spline functions are listed in a parameter file which is specified by the pair_coeff command. Parameter files for different elements are included in the "potentials" directory of the LAMMPS distribution and have a ".meam.sw.spline" file suffix. All of these files are parameterized in terms of LAMMPS metal units.

Note that unlike for other potentials, cutoffs for spline-based MEAM+SW potentials are not set in the pair_style or pair_coeff command; they are specified in the potential files themselves.

Unlike the EAM pair style, which retrieves the atomic mass from the potential file, the spline-based MEAM+SW potentials do not include mass information; thus you need to use the mass command to specify it.

Only a single pair_coeff command is used with the meam/sw/spline style which specifies a potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of spline-based MEAM+SW elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, imagine the Ti.meam.sw.spline file has values for Ti. If your LAMMPS simulation has 3 atoms types and they are all to be treated with this potential, you would use the following pair_coeff command:

pair_coeff * * Ti.meam.sw.spline Ti Ti Ti

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The three Ti arguments map LAMMPS atom types 1,2,3 to the Ti element in the potential file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *meam/sw/spline* potential is used as part of the hybrid pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: The *meam/sw/spline* style currently supports only single-element MEAM+SW potentials. It may be extended for alloy systems in the future.

Example input scripts that use this pair style are provided in the examples/USER/misc/meam_sw_spline directory.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair style does not support multiple element types or mixing. It has been designed for pure elements only.

This pair style does not support the pair_modify shift, table, and tail options.

The *meam/sw/spline* pair style does not write its information to binary restart files, since it is stored in an external potential parameter file. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

The *meam/sw/spline* pair style can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style requires the newton setting to be "on" for pair interactions.

This pair style is only enabled if LAMMPS was built with the USER-MISC package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, pair_style meam, pair_style meam/spline

Default: none

(Lenosky) Lenosky, Sadigh, Alonso, Bulatov, de la Rubia, Kim, Voter, Kress, Modell. Simul. Mater. Sci. Eng. 8, 825 (2000).

(Stillinger) Stillinger, Weber, Phys. Rev. B 31, 5262 (1985).

(Nicklas) The spline-based MEAM+SW format was first devised and used to develop potentials for bcc transition metals by Jeremy Nicklas, Michael Fellinger, and Hyoungki Park at The Ohio State University.

pair_style mie/cut command

pair_style mie/cut/gpu command

Syntax:

```
pair_style mie/cut cutoff
```

- cutoff = global cutoff for mie/cut interactions (distance units)

Examples:

```
pair_style mie/cut 10.0
pair_coeff 1 1 0.72 3.40 23.00 6.66
pair_coeff 2 2 0.30 3.55 12.65 6.00
pair_coeff 1 2 0.46 3.32 16.90 6.31
```

Description:

The *mie/cut* style computes the Mie potential, given by

$$E = C\epsilon \left[\left(\frac{\sigma}{r} \right)^{\gamma_{rep}} - \left(\frac{\sigma}{r} \right)^{\gamma_{att}} \right] \qquad r < r_c$$

Rc is the cutoff and C is a function that depends on the repulsive and attractive exponents, given by:

$$C = \left(\frac{\gamma_{rep}}{\gamma_{rep} - \gamma_{att}} \right) \left(\frac{\gamma_{rep}}{\gamma_{att}} \right)^{\left(\frac{\gamma_{att}}{\gamma_{rep} - \gamma_{att}} \right)}$$

Note that for 12/6 exponents, C is equal to 4 and the formula is the same as the standard Lennard-Jones potential.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- epsilon (energy units)
- sigma (distance units)
- gammaR
- gammaA
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the mie/cut pair styles can be mixed. If not explicity defined, both the repulsive and attractive gamma exponents for different atoms will be calculated following the same mixing rule defined for distances. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the [pair_modify](#) shift option for the energy of the pair interaction.

This pair style supports the [pair_modify](#) tail option for adding a long-range tail correction to the energy and pressure of the pair interaction.

This pair style writes its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style supports the use of the *inner*, *middle*, and *outer* keywords of the [run_style respa](#) command, meaning the pairwise forces can be partitioned by distance at different levels of the rRESPA hierarchy. See the [run_style](#) command for details.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

(Mie) G. Mie, Ann Phys, 316, 657 (1903).

(Avendano) C. Avendano, T. Lafitte, A. Galindo, C. S. Adjiman, G. Jackson, E. Muller, J Phys Chem B, 115, 11154 (2011).

pair_modify command

Syntax:

```
pair_modify keyword value ...
```

- one or more keyword/value pairs may be listed
- keyword = *shift* or *mix* or *table* or *table/disp* or *tabinner* or *tabinner/disp* or *tail* or *compute*

```
  mix value = geometric or arithmetic or sixthpower
    shift value = yes or no
    table value = N
      2^N = # of values in table
    table/disp value = N
      2^N = # of values in table
    tabinner value = cutoff
      cutoff = inner cutoff at which to begin table (distance units)
    tabinner/disp value = cutoff
      cutoff = inner cutoff at which to begin table (distance units)
    tail value = yes or no
    compute value = yes or no
```

Examples:

```
pair_modify shift yes mix geometric
pair_modify tail yes
pair_modify table 12
```

Description:

Modify the parameters of the currently defined pair style. Not all parameters are relevant to all pair styles.

The *mix* keyword affects pair coefficients for interactions between atoms of type I and J, when I != J and the coefficients are not explicitly set in the input script. Note that coefficients for I = J must be set explicitly, either in the input script via the "pair_coeff" command or in the "Pair Coeffs" section of the [data file](#). For some pair styles it is not necessary to specify coefficients when I != J, since a "mixing" rule will create them from the I,I and J,J settings. The pair_modify *mix* value determines what formulas are used to compute the mixed coefficients. In each case, the cutoff distance is mixed the same way as sigma.

Note that not all pair styles support mixing. Also, some mix options are not available for certain pair styles. See the doc page for individual pair styles for those restrictions. Note also that the [pair_coeff](#) command also can be to directly set coefficients for a specific I != J pairing, in which case no mixing is performed.

mix *geometric*

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = sqrt(sigma_i * sigma_j)
```

mix *arithmetic*

```
epsilon_ij = sqrt(epsilon_i * epsilon_j)
sigma_ij = (sigma_i + sigma_j) / 2
```

mix *sixthpower*

```
epsilon_ij = (2 * sqrt(epsilon_i*epsilon_j) * sigma_i^3 * sigma_j^3) /
```

pair_modify command   882

```
                 (sigma_i^6 + sigma_j^6)
sigma_ij = ((sigma_i**6 + sigma_j**6) / 2) ^ (1/6)
```

The *shift* keyword determines whether a Lennard-Jones potential is shifted at its cutoff to 0.0. If so, this adds an energy term to each pairwise interaction which will be included in the thermodynamic output, but does not affect pair forces or atom trajectories. See the doc page for individual pair styles to see which ones support this option.

The *table* and *table/disp* keywords apply to pair styles with a long-range Coulombic term or long-range dispersion term respectively; see the doc page for individual styles to see which potentials support these options. If N is non-zero, a table of length 2^N is pre-computed for forces and energies, which can shrink their computational cost by up to a factor of 2. The table is indexed via a bit-mapping technique [(Wolff)](#) and a linear interpolation is performed between adjacent table values. In our experiments with different table styles (lookup, linear, spline), this method typically gave the best performance in terms of speed and accuracy.

The choice of table length is a tradeoff in accuracy versus speed. A larger N yields more accurate force computations, but requires more memory which can slow down the computation due to cache misses. A reasonable value of N is between 8 and 16. The default value of 12 (table of length 4096) gives approximately the same accuracy as the no-table (N = 0) option. For N = 0, forces and energies are computed directly, using a polynomial fit for the needed erfc() function evaluation, which is what earlier versions of LAMMPS did. Values greater than 16 typically slow down the simulation and will not improve accuracy; values from 1 to 8 give unreliable results.

The *tabinner* and *tabinner/disp* keywords set an inner cutoff above which the pairwise computation is done by table lookup (if tables are invoked), for the corresponding Coulombic and dispersion tables discussed with the *table* and *table/disp* keywords. The smaller the cutoff is set, the less accurate the table becomes (for a given number of table values), which can require use of larger tables. The default cutoff value is sqrt(2.0) distance units which means nearly all pairwise interactions are computed via table lookup for simulations with "real" units, but some close pairs may be computed directly (non-table) for simulations with "lj" units.

When the *tail* keyword is set to *yes*, certain pair styles will add a long-range VanderWaals tail "correction" to the energy and pressure. See the doc page for individual styles to see which support this option. These corrections are included in the calculation and printing of thermodynamic quantities (see the [thermo_style](#) command). Their effect will also be included in constant NPT or NPH simulations where the pressure influences the simulation box dimensions (e.g. the [fix npt](#) and [fix nph](#) commands). The formulas used for the long-range corrections come from equation 5 of [(Sun)](#).

Several assumptions are inherent in using tail corrections, including the following:

- The simulated system is a 3d bulk homogeneous liquid. This option should not be used for systems that are non-liquid, 2d, have a slab geometry (only 2d periodic), or inhomogeneous.
- G(r), the radial distribution function (rdf), is unity beyond the cutoff, so a fairly large cutoff should be used (i.e. 2.5 sigma for an LJ fluid), and it is probably a good idea to verify this assumption by checking the rdf. The rdf is not exactly unity beyond the cutoff for each pair of interaction types, so the tail correction is necessarily an approximation.
- Thermophysical properties obtained from calculations with this option enabled will not be thermodynamically consistent with the truncated force-field that was used. In other words, atoms do not feel any LJ pair interactions beyond the cutoff, but the energy and pressure reported by the simulation include an estimated contribution from those interactions.

The *compute* keyword allows pairwise computations to be turned off, even though a [pair_style](#) is defined. This is not useful for running a real simulation, but can be useful for debugging purposes or for computing only partial forces that do not include the pairwise contribution. You can also do this by simply not defining a [pair_style](#), but a Kspace-compatible pair_style is required if you also want to define a [kspace_style](#). This keyword gives you that option.

pair_modify command   883

Restrictions: none

You cannot use *shift* yes with *tail* yes, since those are conflicting options. You cannot use *tail* yes with 2d simulations.

Related commands:

pair_style, pair_coeff, thermo_style

Default:

The option defaults are mix = geometric, shift = no, table = 12, tabinner = sqrt(2.0), tail = no, and compute = yes.

Note that some pair styles perform mixing, but only a certain style of mixing. See the doc pages for individual pair styles for details.

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200-32 (1999).

(Sun) Sun, J Phys Chem B, 102, 7338-7364 (1998).

pair_style morse command

pair_style morse/cuda command

pair_style morse/gpu command

pair_style morse/omp command

pair_style morse/opt command

Syntax:

```
pair_style morse cutoff
```

- cutoff = global cutoff for Morse interactions (distance units)

Examples:

```
pair_style morse 2.5
pair_coeff * * 100.0 2.0 1.5
pair_coeff 1 1 100.0 2.0 1.5 3.0
```

Description:

Style *morse* computes pairwise interactions with the formula

$$E = D_0 \left[e^{-2\alpha(r-r_0)} - 2e^{-\alpha(r-r_0)} \right] \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- D0 (energy units)
- alpha (1/distance units)
- r0 (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global morse cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

None of these pair styles support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

All of these pair styles support the pair_modify shift option for the energy of the pair interaction.

The pair_modify table options is not relevant for the Morse pair styles.

None of these pair styles support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

All of these pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff

Default: none

pair_style nb3b/harmonic command

pair_style nb3b/harmonic/omp command

Syntax:

```
pair_style nb3b/harmonic
```

Examples:

```
pair_style nb3b/harmonic
pair_coeff * * MgOH.nb3bharmonic Mg O H
```

Description:

This pair style computes a nonbonded 3-body harmonic potential for the energy E of a system of atoms as

$$E = K(\theta - \theta_0)^2$$

where *theta_0* is the equilibrium value of the angle and *K* is a prefactor. Note that the usual 1/2 factor is included in *K*. The form of the potential is identical to that used in angle_style *harmonic*, but in this case, the atoms do not need to be explicitly bonded.

Only a single pair_coeff command is used with this style which specifies a potential file with parameters for specified elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of elements to atom types

See the pair_coeff doc page for alternate ways to specify the path for the potential file.

As an example, imagine a file SiC.nb3b.harmonic has potential values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.nb3b.harmonic Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the potential file. The final C argument maps LAMMPS atom type 4 to the C element in the potential file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when the potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials. An example of a pair_coeff command for use with the *hybrid* pair style is:

pair_coeff * * nb3b/harmonic MgOH.nb3b.harmonic Mg O H

Three-body nonbonded harmonic files in the *potentials* directory of the LAMMPS distribution have a ".nb3b.harmonic" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements.

Each entry has six arguments. The first three are atom types as referenced in the LAMMPS input file. The first argument specifies the central atom. The fourth argument indicates the *K* parameter. The fifth argument indicates *theta_0*. The sixth argument indicates a separation cutoff in Angstroms.

For a given entry, if the second and third arguments are identical, then the entry is for a cutoff for the distance between types 1 and 2 (values for *K* and *theta_0* are irrelevant in this case).

For a given entry, if the first three arguments are all different, then the entry is for the *K* and *theta_0* parameters (the cutoff in this case is irrelevant).

It is *not* required that the potential file contain entries for all of the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

This pair style can only be used if LAMMPS was built with the MANYBODY package (which it is by default). See the Making LAMMPS section for more info on packages.

Related commands:

pair_coeff

Default: none

pair_style nm/cut command

pair_style nm/cut/coul/cut command

pair_style nm/cut/coul/long command

pair_style nm/cut/omp command

pair_style nm/cut/coul/cut/omp command

pair_style nm/cut/coul/long/omp command

Syntax:

```
pair_style style args
```

- style = *nm/cut* or *nm/cut/coul/cut* or *nm/cut/coul/long*
- args = list of arguments for a particular style

```
  nm/cut args = cutoff
     cutoff = global cutoff for Pair interactions (distance units)
    nm/cut/coul/cut args = cutoff (cutoff2)
      cutoff = global cutoff for Pair (and Coulombic if only 1 arg) (distance units)
      cutoff2 = global cutoff for Coulombic (optional) (distance units)
    nm/cut/coul/long args = cutoff (cutoff2)
      cutoff = global cutoff for Pair (and Coulombic if only 1 arg) (distance units)
      cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style nm/cut 12.0
pair_coeff * * 0.01 5.4 8.0 7.0
pair_coeff 1 1 0.01 4.4 7.0 6.0


pair_style nm/cut/coul/cut 12.0 15.0
pair_coeff * * 0.01 5.4 8.0 7.0
pair_coeff 1 1 0.01 4.4 7.0 6.0


pair_style nm/cut/coul/long 12.0 15.0
pair_coeff * * 0.01 5.4 8.0 7.0
pair_coeff 1 1 0.01 4.4 7.0 6.0
```

Description:

Style *nm* computes site-site interactions based on the N-M potential by [Clarke](#), mainly used for ionic liquids. A site can represent a single atom or a united-atom site. The energy of an interaction has the following form:

$$E = \frac{E_0}{(n-m)} \left[m \left(\frac{r_0}{r} \right)^n - n \left(\frac{r_0}{r} \right)^m \right] \qquad r < r_c$$

Rc is the cutoff.

Style *nm/cut/coul/cut* adds a Coulombic pairwise interaction given by

$$E = \frac{C q_i q_j}{\epsilon r} \qquad r < r_c$$

where C is an energy-conversion constant, Qi and Qj are the charges on the 2 atoms, and epsilon is the dielectric constant which can be set by the [dielectric](#) command. If one cutoff is specified in the pair_style command, it is used for both the NM and Coulombic terms. If two cutoffs are specified, they are used as cutoffs for the NM and Coulombic terms respectively.

Styles *nm/cut/coul/long* compute the same Coulombic interactions as style *nm/cut/coul/cut* except that an additional damping factor is applied to the Coulombic term so it can be used in conjunction with the [kspace_style](#) command and its *ewald* or *pppm* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

For all of the *nm* pair styles, the following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands.

- E0 (energy units)
- r0 (distance units)
- n (unitless)
- m (unitless)
- cutoff1 (distance units)
- cutoff2 (distance units)

The latter 2 coefficients are optional. If not specified, the global NM and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both NM and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the NM and Coulombic cutoffs for this type pair. You cannot specify 2 cutoffs for style *nm*, since it has no Coulombic terms.

For *nm/cut/coul/long* only the NM cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

All of the *nm* pair styles supports the [pair_modify](#) shift option for the energy of the pair interaction.

The *nm/cut/coul/long* pair styles support the [pair_modify](#) table option since they can tabulate the short-range portion of the long-range Coulombic interaction.

All of the *nm* pair styles support the [pair_modify](#) tail option for adding a long-range tail correction to the energy and pressure for the NM portion of the pair interaction.

All of the *nm* pair styles write their information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

pair_style nm/cut/coul/long/omp command

All of the *nm* pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

These pair styles are part of the MISC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Clarke) Clarke and Smith, J Chem Phys, 84, 2290 (1986).

pair_style none command

Syntax:

```
pair_style none
```

Examples:

```
pair_style none
```

Description:

Using a pair style of none means pair forces are not computed.

With this choice, the force cutoff is 0.0, which means that only atoms within the neighbor skin distance (see the [neighbor](#) command) are communicated between processors. You must insure the skin distance is large enough to acquire atoms needed for computing bonds, angles, etc.

A pair style of *none* will also prevent pairwise neighbor lists from being built. However if the [neighbor](#) style is *bin*, data structures for binning are still allocated. If the neighbor skin distance is small, then these data structures can consume a large amount of memory. So you should either set the neighbor style to *nsq* or set the skin distance to a larger value.

Restrictions: none

Related commands: none

Default: none

pair_style peri/pmb command

pair_style peri/pmb/omp command

pair_style peri/lps command

pair_style peri/lps/omp command

pair_style peri/ves command

Syntax:

```
pair_style style
```

- style = *peri/pmb* or *peri/lps* or *peri/ves*

Examples:

```
pair_style peri/pmb
pair_coeff * * 1.6863e22 0.0015001 0.0005 0.25

pair_style peri/lps
pair_coeff * * 14.9e9 14.9e9 0.0015001 0.0005 0.25

pair_style peri/ves
pair_coeff * * 14.9e9 14.9e9 0.0015001 0.0005 0.25 0.5 0.001
```

Description:

The peridynamic pair styles implement material models that can be used at the mescscopic and macroscopic scales.

Style *peri/pmb* implements the Peridynamic bond-based prototype microelastic brittle (PMB) model.

Style *peri/lps* implements the Peridynamic state-based linear peridynamic solid (LPS) model.

Style *peri/ves* implements the Peridynamic state-based linear peridynamic viscoelastic solid (VES) model.

The canonical papers on Peridynamics are [(Silling 2000)](#) and [(Silling 2007)](#). The implementation of Peridynamics in LAMMPS is described in [(Parks)](#). Also see the [PDLAMMPS user guide](#) for more details about the implementation of peridynamics in LAMMPS.

The peridynamic VES model in PDLAMMPS is implemented by R. Rahman and J.T Foster at University of Texas at San Antonio. The VES formulation is described in [(Mitchell)](#). An additional PDF doc with details is in [doc/PDF/PDLammps_VES.pdf](#). For questions regarding VES implementation in LAMMPS please contact Rezwan Rahman: rezwanur.rahman at utsa.edu.

The following coefficients must be defined for each pair of atom types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below.

For the *peri/pmb* style:

pair_style peri/pmb command 893

- c (energy/distance/volume^2 units)
- horizon (distance units)
- s00 (unitless)
- alpha (unitless)

C is the effectively a spring constant for Peridynamic bonds, the horizon is a cutoff distance for truncating interactions, and s00 and alpha are used as a bond breaking criteria. The units of c are such that c/distance = stiffness/volume^2, where stiffness is energy/distance^2 and volume is distance^3. See the users guide for more details.

For the *peri/lps* style:

- K (force/area units)
- G (force/area units)
- horizon (distance units)
- s00 (unitless)
- alpha (unitless)

K is the bulk modulus and G is the shear modulus. The horizon is a cutoff distance for truncating interactions, and s00 and alpha are used as a bond breaking criteria. See the users guide for more details.

For the *peri/ves* style:

- K (force/area units)
- G (force/area units)
- horizon (distance units)
- s00 (unitless)
- alpha (unitless)
- m_lambdai (unitless)
- m_taubi (unitless)

The same as for *peri/lps*, K is the bulk modulus and G is the shear modulus. The horizon is a cutoff distance for truncating interactions, and s00 and alpha are used as a bond breaking criteria. m_lambdai and m_taubi are the viscoelastic relaxation parameter and time constant, respectively. m_lambdai varies within zero to one. For very small values of m_lambdai the viscoelsatic model responds very similar to a linear elastic model. For details see the description in (Mitchell).

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

These pair styles do not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

These pair styles do not support the pair_modify shift option.

The pair_modify table and tail options are not relevant for these pair styles.

These pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The *peri/pmb*, *peri/lps* and *peri/ves* styles are part of the PERI package. They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff

Default: none

(Parks) Parks, Lehoucq, Plimpton, Silling, Comp Phys Comm, 179(11), 777-783 (2008).

(Silling 2000) Silling, J Mech Phys Solids, 48, 175-209 (2000).

(Silling 2007) Silling, Epton, Weckner, Xu, Askari, J Elasticity, 88, 151-184 (2007).

(Mitchell) Mitchell, "A non-local, ordinary-state-based viscoelasticity model for peridynamics", Sandia National Lab Report, 8064:1-28 (2011).

pair_style reax/c command

Syntax:

```
pair_style reax/c cfile keyword value
```

- cfile = NULL or name of a control file
- zero or more keyword/value pairs may be appended

```
  keyword = checkqeq or lgvdw or safezone or mincap
    checkqeq value = yes or no = whether or not to require qeq/reax fix
    lgvdw value = yes or no = whether or not to use a low gradient vdW correction
    safezone = factor used for array allocation
    mincap = minimum size for array allocation
```

Examples:

```
pair_style reax/c NULL
pair_style reax/c controlfile checkqeq no
pair_style reax/c NULL lgvdw yes
pair_style reax/c NULL safezone 1.6 mincap 100
pair_coeff * * ffield.reax C H O N
```

Description:

Style *reax/c* computes the ReaxFF potential of van Duin, Goddard and co-workers. ReaxFF uses distance-dependent bond-order functions to represent the contributions of chemical bonding to the potential energy. There is more than one version of ReaxFF. The version implemented in LAMMPS uses the functional forms documented in the supplemental information of the following paper: [(Chenoweth et al., 2008)](#). The version integrated into LAMMPS matches the most up-to-date version of ReaxFF as of summer 2010. For more technical details about the pair reax/c implementation of ReaxFF, see the [(Aktulga)](#) paper.

The *reax/c* style differs from the [pair_style reax](#) command in the lo-level implementation details. The *reax* style is a Fortran library, linked to LAMMPS. The *reax/c* style was initially implemented as stand-alone C code and is now integrated into LAMMPS as a package.

LAMMPS provides several different versions of ffield.reax in its potentials dir, each called potentials/ffield.reax.label. These are documented in potentials/README.reax. The default ffield.reax contains parameterizations for the following elements: C, H, O, N, S.

The format of these files is identical to that used originally by van Duin. We have tested the accuracy of *pair_style reax/c* potential against the original ReaxFF code for the systems mentioned above. You can use other ffield files for specific chemical systems that may be available elsewhere (but note that their accuracy may not have been tested).

IMPORTANT NOTE: We do not distribute a wide variety of ReaxFF force field files with LAMMPS. Adri van Duin's group at PSU is the central repository for this kind of data as they are continuously deriving and updating parameterizations for different classes of materials. You can visit their WWW site at [http://www.engr.psu.edu/adri](#), register as a "new user", and then submit a request to their group describing material(s) you are interested in modeling with ReaxFF. They can tell you what is currently available or what it would take to create a suitable ReaxFF parameterization.

The *cfile* setting can be specified as NULL, in which case default settings are used. A control file can be specified which defines values of control variables. Some control variables are global parameters for the

ReaxFF potential. Others define certain performance and output settings. Each line in the control file specifies the value for a control variable. The format of the control file is described below.

IMPORTANT NOTE: The LAMMPS default values for the ReaxFF global parameters correspond to those used by Adri van Duin's stand-alone serial code. If these are changed by setting control variables in the control file, the results from LAMMPS and the serial code will not agree.

Two examples using *pair_style reax/c* are provided in the examples/reax sub-directory, along with corresponding examples for pair_style reax.

Use of this pair style requires that a charge be defined for every atom. See the atom_style and read_data commands for details on how to specify charges.

The ReaxFF parameter files provided were created using a charge equilibration (QEq) model for handling the electrostatic interactions. Therefore, by default, LAMMPS requires that the fix qeq/reax command be used with *pair_style reax/c* when simulating a ReaxFF model, to equilibrate charge each timestep. Using the keyword *checkqeq* with the value *no* turns off the check for *fix qeq/reax*, allowing a simulation to be run without charge equilibration. In this case, the static charges you assign to each atom will be used for computing the electrostatic interactions in the system. See the fix qeq/reax command for details.

Using the optional keyword *lgvdw* with the value *yes* turns on the low-gradient correction of the ReaxFF/C for long-range London Dispersion, as described in the (Liu) paper. Force field file *ffield.reax.lg* is designed for this correction, and is trained for several energetic materials (see "Liu"). When using lg-correction, recommended value for parameter *thb* is 0.01, which can be set in the control file. Note: Force field files are different for the original or lg corrected pair styles, using wrong ffield file generates an error message.

Optional keywords *safezone* and *mincap* are used for allocating reax/c arrays. Increase these values can avoid memory problems, such as segmentation faults and bondchk failed errors, that could occur under certain conditions.

The thermo variable *evdwl* stores the sum of all the ReaxFF potential energy contributions, with the exception of the Coulombic and charge equilibration contributions which are stored in the thermo variable *ecoul*. The output of these quantities is controlled by the thermo command.

This pair style tallies a breakdown of the total ReaxFF potential energy into sub-categories, which can be accessed via the compute pair command as a vector of values of length 14. The 14 values correspond to the following sub-categories (the variable names in italics match those used in the original FORTRAN ReaxFF code):

1. *eb* = bond energy
2. *ea* = atom energy
3. *elp* = lone-pair energy
4. *emol* = molecule energy (always 0.0)
5. *ev* = valence angle energy
6. *epen* = double-bond valence angle penalty
7. *ecoa* = valence angle conjugation energy
8. *ehb* = hydrogen bond energy
9. *et* = torsion energy
10. *eco* = conjugation energy
11. *ew* = van der Waals energy
12. *ep* = Coulomb energy
13. *efi* = electric field energy (always 0.0)
14. *eqeq* = charge equilibration energy

To print these quantities to the log file (with descriptive column headings) the following commands could be included in an input script:

```
compute reax all pair reax/c
variable eb      equal c_reax[1]
variable ea      equal c_reax[2]
...
variable eqeq    equal c_reax[14]
thermo_style custom step temp epair v_eb v_ea ... v_eqeq
```

Only a single pair_coeff command is used with the *reax/c* style which specifies a ReaxFF potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N indices = ReaxFF elements

The filename is the ReaxFF potential file. Unlike for the *reax* pair style, any filename can be used.

In the ReaxFF potential file, near the top, after the general parameters, is the atomic parameters section that contains element names, each with a couple dozen numeric parameters. If there are M elements specified in the *ffield* file, think of these as numbered 1 to M. Each of the N indices you specify for the N atom types of LAMMPS atoms must be an integer from 1 to M. Atoms with LAMMPS type 1 will be mapped to whatever element you specify as the first index value, etc. If a mapping value is specified as NULL, the mapping is not performed. This can be used when the *reax/c* style is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: Currently the reax/c pair style cannot be used as part of the *hybrid* pair style. Some additional work still need to be done to enable this.

As an example, say your LAMMPS simulation has 4 atom types and the elements are ordered as C, H, O, N in the *ffield* file. If you want the LAMMPS atom type 1 and 2 to be C, type 3 to be N, and type 4 to be H, you would use the following pair_coeff command:

```
pair_coeff * * ffield.reax C C N H
```

The format of a line in the control file is as follows:

```
variable_name value
```

and it may be followed by an "!" character and a trailing comment.

If the value of a control variable is not specified, then default values are used. What follows is the list of variables along with a brief description of their use and default values.

simulation_name: Output files produced by *pair_style reax/c* carry this name + extensions specific to their contents. Partial energies are reported with a ".pot" extension, while the trajectory file has ".trj" extension.

tabulate_long_range: To improve performance, long range interactions can optionally be tabulated (0 means no tabulation). Value of this variable denotes the size of the long range interaction table. The range from 0 to long range cutoff (defined in the *ffield* file) is divided into *tabulate_long_range* points. Then at the start of simulation, we fill in the entries of the long range interaction table by computing the energies and forces resulting from van der Waals and Coulomb interactions between every possible atom type pairs present in the input system. During the simulation we consult to the long range interaction table to estimate the energy and forces between a pair of atoms. Linear interpolation is used for estimation. (default value = 0)

energy_update_freq: Denotes the frequency (in number of steps) of writes into the partial energies file. (default value = 0)

nbrhood_cutoff: Denotes the near neighbors cutoff (in Angstroms) regarding the bonded interactions. (default value = 5.0)

hbond_cutoff: Denotes the cutoff distance (in Angstroms) for hydrogen bond interactions.(default value = 7.5. Value of 0.0 turns off hydrogen bonds)

bond_graph_cutoff: is the threshold used in determining what is a physical bond, what is not. Bonds and angles reported in the trajectory file rely on this cutoff. (default value = 0.3)

thb_cutoff: cutoff value for the strength of bonds to be considered in three body interactions. (default value = 0.001)

thb_cutoff_sq: cutoff value for the strength of bond order products to be considered in three body interactions. (default value = 0.00001)

write_freq: Frequency of writes into the trajectory file. (default value = 0)

traj_title: Title of the trajectory - not the name of the trajectory file.

atom_info: 1 means print only atomic positions + charge (default = 0)

atom_forces: 1 adds net forces to atom lines in the trajectory file (default = 0)

atom_velocities: 1 adds atomic velocities to atoms line (default = 0)

bond_info: 1 prints bonds in the trajectory file (default = 0)

angle_info: 1 prints angles in the trajectory file (default = 0)

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the pair_modify mix, shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-REAXC package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

The ReaxFF potential files provided with LAMMPS in the potentials directory are parameterized for real units. You can use the ReaxFF potential with any LAMMPS units, but you would need to create your own potential file with coefficients listed in the appropriate units if your simulation doesn't use "real" units.

Related commands:

pair_coeff, fix qeq/reax, fix reax/c/bonds, fix reax/c/species, pair_style reax

pair_style reax/c command   899

Default:

The keyword defaults are checkqeq = yes, lgvdw = no, safezone = 1.2, mincap = 50.

(Chenoweth_2008) Chenoweth, van Duin and Goddard, Journal of Physical Chemistry A, 112, 1040-1053 (2008).

(Aktulga) Aktulga, Fogarty, Pandit, Grama, Parallel Computing, 38, 245-259 (2012).

(Liu) L. Liu, Y. Liu, S. V. Zybin, H. Sun and W. A. Goddard, Journal of Physical Chemistry A, 115, 11016-11022 (2011).

pair_style reax command

Syntax:

```
pair_style reax hbcut hbnewflag tripflag precision
```

- hbcut = hydrogen-bond cutoff (optional) (distance units)
- hbnewflag = use old or new hbond function style (0 or 1) (optional)
- tripflag = apply stabilization to all triple bonds (0 or 1) (optional)
- precision = precision for charge equilibration (optional)

Examples:

```
pair_style reax
pair_style reax 10.0 0 1 1.0e-5
pair_coeff * * ffield.reax 3 1 2 2
pair_coeff * * ffield.reax 3 NULL NULL 3
```

Description:

Style *reax* computes the ReaxFF potential of van Duin, Goddard and co-workers. ReaxFF uses distance-dependent bond-order functions to represent the contributions of chemical bonding to the potential energy. There is more than one version of ReaxFF. The version implemented in LAMMPS uses the functional forms documented in the supplemental information of the following paper: (Chenoweth). The version integrated into LAMMPS matches the most up-to-date version of ReaxFF as of summer 2010.

The *reax* style differs from the pair_style reax/c command in the lo-level implementation details. The *reax* style is a Fortran library, linked to LAMMPS. The *reax/c* style was initially implemented as stand-alone C code and is now integrated into LAMMPS as a package.

LAMMPS requires that a file called ffield.reax be provided, containing the ReaxFF parameters for each atom type, bond type, etc. The format is identical to the ffield file used by van Duin and co-workers. The filename is required as an argument in the pair_coeff command. Any value other than "ffield.reax" will be rejected (see below).

LAMMPS provides several different versions of ffield.reax in its potentials dir, each called potentials/ffield.reax.label. These are documented in potentials/README.reax. The default ffield.reax contains parameterizations for the following elements: C, H, O, N, S.

IMPORTANT NOTE: We do not distribute a wide variety of ReaxFF force field files with LAMMPS. Adri van Duin's group at PSU is the central repository for this kind of data as they are continuously deriving and updating parameterizations for different classes of materials. You can visit their WWW site at http://www.engr.psu.edu/adri, register as a "new user", and then submit a request to their group describing material(s) you are interested in modeling with ReaxFF. They can tell you what is currently available or what it would take to create a suitable ReaxFF parameterization.

The format of these files is identical to that used originally by van Duin. We have tested the accuracy of *pair_style reax* potential against the original ReaxFF code for the systems mentioned above. You can use other ffield files for specific chemical systems that may be available elsewhere (but note that their accuracy may not have been tested).

The *hbcut*, *hbnewflag*, *tripflag*, and *precision* settings are optional arguments. If none are provided, default settings are used: *hbcut* = 6 (which is Angstroms in real units), *hbnewflag* = 1 (use new hbond function style),

tripflag = 1 (apply stabilization to all triple bonds), and *precision* = 1.0e-6 (one part in 10^6). If you wish to override any of these defaults, then all of the settings must be specified.

Two examples using *pair_style reax* are provided in the examples/reax sub-directory, along with corresponding examples for [pair_style reax/c](#).

Use of this pair style requires that a charge be defined for every atom since the *reax* pair style performs a charge equilibration (QEq) calculation. See the [atom_style](#) and [read_data](#) commands for details on how to specify charges.

The thermo variable *evdwl* stores the sum of all the ReaxFF potential energy contributions, with the exception of the Coulombic and charge equilibration contributions which are stored in the thermo variable *ecoul*. The output of these quantities is controlled by the [thermo](#) command.

This pair style tallies a breakdown of the total ReaxFF potential energy into sub-categories, which can be accessed via the [compute pair](#) command as a vector of values of length 14. The 14 values correspond to the following sub-categories (the variable names in italics match those used in the ReaxFF FORTRAN library):

1. *eb* = bond energy
2. *ea* = atom energy
3. *elp* = lone-pair energy
4. *emol* = molecule energy (always 0.0)
5. *ev* = valence angle energy
6. *epen* = double-bond valence angle penalty
7. *ecoa* = valence angle conjugation energy
8. *ehb* = hydrogen bond energy
9. *et* = torsion energy
10. *eco* = conjugation energy
11. *ew* = van der Waals energy
12. *ep* = Coulomb energy
13. *efi* = electric field energy (always 0.0)
14. *eqeq* = charge equilibration energy

To print these quantities to the log file (with descriptive column headings) the following commands could be included in an input script:

```
compute reax all pair reax
variable eb      equal c_reax[1]
variable ea      equal c_reax[2]
...
variable eqeq    equal c_reax[14]
thermo_style custom step temp epair v_eb v_ea ... v_eqeq
```

Only a single pair_coeff command is used with the *reax* style which specifies a ReaxFF potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N indices = mapping of ReaxFF elements to atom types

The specification of the filename and the mapping of LAMMPS atom types recognized by the ReaxFF is done differently than for other LAMMPS potentials, due to the non-portable difficulty of passing character strings (e.g. filename, element names) between C++ and Fortran.

The filename has to be "ffield.reax" and it has to exist in the directory you are running LAMMPS in. This means you cannot prepend a path to the file in the potentials dir. Rather, you should copy that file into the

directory you are running from. If you wish to use another ReaxFF potential file, then name it "ffield.reax" and put it in the directory you run from.

In the ReaxFF potential file, near the top, after the general parameters, is the atomic parameters section that contains element names, each with a couple dozen numeric parameters. If there are M elements specified in the *ffield* file, think of these as numbered 1 to M. Each of the N indices you specify for the N atom types of LAMMPS atoms must be an integer from 1 to M. Atoms with LAMMPS type 1 will be mapped to whatever element you specify as the first index value, etc. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a ReaxFF potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

IMPORTANT NOTE: Currently the reax pair style cannot be used as part of the *hybrid* pair style. Some additional changes still need to be made to enable this.

As an example, say your LAMMPS simulation has 4 atom types and the elements are ordered as C, H, O, N in the *ffield* file. If you want the LAMMPS atom type 1 and 2 to be C, type 3 to be N, and type 4 to be H, you would use the following pair_coeff command:

```
pair_coeff * * ffield.reax 1 1 4 2
```

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the [pair_modify](#) mix, shift, table, and tail options.

This pair style does not write its information to [binary restart files](#), since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

The ReaxFF potential files provided with LAMMPS in the potentials directory are parameterized for real [units](#). You can use the ReaxFF potential with any LAMMPS units, but you would need to create your own potential file with coefficients listed in the appropriate units if your simulation doesn't use "real" units.

Related commands:

[pair_coeff](#), [pair_style reax/c](#), [fix_reax_bonds](#)

Default:

The keyword defaults are *hbcut* = 6, *hbnewflag* = 1, *tripflag* = 1, *precision* = 1.0e-6.

(Chenoweth_2008) Chenoweth, van Duin and Goddard, Journal of Physical Chemistry A, 112, 1040-1053 (2008).

pair_style resquared command

pair_style resquared/gpu command

pair_style resquared/omp command

Syntax:

```
pair_style resquared cutoff
```

- cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style resquared 10.0
pair_coeff * * 1.0 1.0 1.7 3.4 3.4 1.0 1.0 1.0
```

Description:

Style *resquared* computes the RE-squared anisotropic interaction [(Everaers)](#), [(Babadi)](#) between pairs of ellipsoidal and/or spherical Lennard-Jones particles. For ellipsoidal interactions, the potential considers the ellipsoid as being comprised of small spheres of size sigma. LJ particles are a single sphere of size sigma. The distinction is made to allow the pair style to make efficient calculations of ellipsoid/solvent interactions.

Details for the equations used are given in the references below and in [this supplementary document](#).

Use of this pair style requires the NVE, NVT, or NPT fixes with the *asphere* extension (e.g. [fix nve/asphere](#)) in order to integrate particle rotation. Additionally, [atom_style ellipsoid](#) should be used since it defines the rotational state and the size and shape of each ellipsoidal particle.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- A12 = Energy Prefactor/Hamaker constant (energy units)
- sigma = atomic interaction diameter (distance units)
- epsilon_i_a = relative well depth of type I for side-to-side interactions
- epsilon_i_b = relative well depth of type I for face-to-face interactions
- epsilon_i_c = relative well depth of type I for end-to-end interactions
- epsilon_j_a = relative well depth of type J for side-to-side interactions
- epsilon_j_b = relative well depth of type J for face-to-face interactions
- epsilon_j_c = relative well depth of type J for end-to-end interactions
- cutoff (distance units)

The parameters used depend on the type of the interacting particles, i.e. ellipsoids or LJ spheres. The type of a particle is determined by the diameters specified for its 3 shape paramters. If all 3 shape parameters = 0.0, then the particle is treated as an LJ sphere. The epsilon_i_* or epsilon_j_* parameters are ignored for LJ spheres. If the 3 shape paraemters are > 0.0, then the particle is treated as an ellipsoid (even if the 3 parameters are equal to each other).

A12 specifies the energy prefactor which depends on the types of the two interacting particles.

For ellipsoid/ellipsoid interactions, the interaction is computed by the formulas in the supplementary docuement referenced above. A12 is the Hamaker constant as described in (Everaers). In LJ units:

$$A_{12} = 4\pi^2 \epsilon_{LJ}(\rho\sigma^3)^2$$

where rho gives the number density of the spherical particles composing the ellipsoids and epsilon_LJ determines the interaction strength of the spherical particles.

For ellipsoid/LJ sphere interactions, the interaction is also computed by the formulas in the supplementary docuement referenced above. A12 has a modifed form (see here for details):

$$A_{12} = 4\pi^2 \epsilon_{LJ}(\rho\sigma^3)$$

For ellipsoid/LJ sphere interactions, a correction to the distance- of-closest approach equation has been implemented to reduce the error from two particles of disparate sizes; see this supplementary document.

For LJ sphere/LJ sphere interactions, the interaction is computed using the standard Lennard-Jones formula, which is much cheaper to compute than the ellipsoidal formulas. A12 is used as epsilon in the standard LJ formula:

$$A_{12} = \epsilon_{LJ}$$

and the specified *sigma* is used as the sigma in the standard LJ formula.

When one of both of the interacting particles are ellipsoids, then *sigma* specifies the diameter of the continuous distribution of constituent particles within each ellipsoid used to model the RE-squared potential. Note that this is a different meaning for *sigma* than the pair_style gayberne potential uses.

The epsilon_i and epsilon_j coefficients are defined for atom types, not for pairs of atom types. Thus, in a series of pair_coeff commands, they only need to be specified once for each atom type.

Specifically, if any of epsilon_i_a, epsilon_i_b, epsilon_i_c are non-zero, the three values are assigned to atom type I. If all the epsilon_i values are zero, they are ignored. If any of epsilon_j_a, epsilon_j_b, epsilon_j_c are non-zero, the three values are assigned to atom type J. If all three epsilon_i values are zero, they are ignored. Thus the typical way to define the epsilon_i and epsilon_j coefficients is to list their values in "pair_coeff I J" commands when I = J, but set them to 0.0 when I != J. If you do list them when I != J, you should insure they are consistent with their values in other pair_coeff commands.

Note that if this potential is being used as a sub-style of pair_style hybrid, and there is no "pair_coeff I I" setting made for RE-squared for a particular type I (because I-I interactions are computed by another hybrid pair potential), then you still need to insure the epsilon a,b,c coefficients are assigned to that type in a "pair_coeff I J" command.

For large uniform molecules it has been shown that the epsilon_*_* energy parameters are approximately representable in terms of local contact curvatures (Everaers):

$$\epsilon_a = \sigma \cdot \frac{a}{b \cdot c}; \epsilon_b = \sigma \cdot \frac{b}{a \cdot c}; \epsilon_c = \sigma \cdot \frac{c}{a \cdot b}$$

where a, b, and c give the particle diameters.

The last coefficient is optional. If not specified, the global cutoff specified in the pair_style command is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance can be mixed, but only for sphere pairs. The default mix value is *geometric*. See the "pair_modify" command for details. Other type pairs cannot be mixed, due to the different meanings of the energy prefactors used to calculate the interactions and the implicit dependence of the ellipsoid-sphere interaction on the equation for the Hamaker constant presented here. Mixing of sigma and epsilon followed by calculation of the energy prefactors using the equations above is recommended.

This pair styles supports the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction, but only for sphere-sphere interactions. There is no shifting performed for ellipsoidal interactions due to the anisotropic dependence of the interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords of the run_style command.

Restrictions:

This style is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This pair style requires that atoms be ellipsoids as defined by the [atom_style ellipsoid](#) command.

Particles acted on by the potential can be finite-size aspherical or spherical particles, or point particles. Spherical particles have all 3 of their shape parameters equal to each other. Point particles have all 3 of their shape parameters equal to 0.0.

The distance-of-closest-approach approximation used by LAMMPS becomes less accurate when high-aspect ratio ellipsoids are used.

Related commands:

[pair_coeff](#), [fix nve/asphere](#), [compute temp/asphere](#), [pair_style gayberne](#)

Default: none

(Everaers) Everaers and Ejtehadi, Phys Rev E, 67, 041710 (2003).

(Berardi) Babadi, Ejtehadi, Everaers, J Comp Phys, 219, 770-779 (2006).

LIGGGHTS Users Manual

pair_style lj/sdk command

pair_style lj/sdk/gpu command

pair_style lj/sdk/omp command

pair_style lj/sdk/coul/long command

pair_style lj/sdk/coul/long/gpu command

pair_style lj/sdk/coul/long/omp command

Syntax:

```
pair_style style args
```

- style = *lj/sdk* or *lj/sdk/coul/long*
- args = list of arguments for a particular style

```
lj/sdk args = cutoff
   cutoff = global cutoff for Lennard Jones interactions (distance units)
 lj/sdk/coul/long args = cutoff (cutoff2)
   cutoff = global cutoff for LJ (and Coulombic if only 1 arg) (distance units)
   cutoff2 = global cutoff for Coulombic (optional) (distance units)
```

Examples:

```
pair_style lj/sdk 2.5
pair_coeff 1 1 lj12_6 1 1.1 2.8

pair_style lj/sdk/coul/long 10.0
pair_style lj/sdk/coul/long 10.0 12.0
pair_coeff 1 1 lj9_6 100.0 3.5 12.0
```

Description:

The *lj/sdk* styles compute a 9/6, 12/4, or 12/6 Lennard-Jones potential, given by

$$E = \frac{27}{4}\epsilon\left[\left(\frac{\sigma}{r}\right)^9 - \left(\frac{\sigma}{r}\right)^6\right] \qquad r < r_c$$

$$E = \frac{3\sqrt{3}}{2}\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^4\right] \qquad r < r_c$$

$$E = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right] \qquad r < r_c$$

as required for the SDK Coarse-grained MD parametrization discussed in [(Shinoda)](#) and [(DeVane)](#). Rc is the cutoff.

Style *lj/sdk/coul/long* computes the adds Coulombic interactions with an additional damping factor applied so it can be used in conjunction with the kspace_style command and its *ewald* or *pppm* or *pppm/cg* option. The Coulombic cutoff specified for this style means that pairwise interactions within this distance are computed directly; interactions outside that distance are computed in reciprocal space.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above, or in the data file or restart files read by the read_data or read_restart commands, or by mixing as described below:

- cg_type (lj9_6, lj12_4, or lj12_6)
- epsilon (energy units)
- sigma (distance units)
- cutoff1 (distance units)

Note that sigma is defined in the LJ formula as the zero-crossing distance for the potential, not as the energy minimum. The prefactors are chosen so that the potential minimum is at -epsilon.

The latter 2 coefficients are optional. If not specified, the global LJ and Coulombic cutoffs specified in the pair_style command are used. If only one cutoff is specified, it is used as the cutoff for both LJ and Coulombic interactions for this type pair. If both coefficients are specified, they are used as the LJ and Coulombic cutoffs for this type pair.

For *lj/sdk/coul/long* only the LJ cutoff can be specified since a Coulombic cutoff cannot be specified for an individual I,J type pair. All type pairs use the same global Coulombic cutoff specified in the pair_style command.

Styles with a *cuda*, *gpu*, *omp* or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP, and OPT packages respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, and rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of the lj/sdk pair styles *cannot* be mixed, since different pairs may have different exponents. So all parameters for all pairs have to be specified explicitly through the "pair_coeff" command. Defining then in a data file is also not supported, due to limitations of that file format.

All of the lj/sdk pair styles support the pair_modify shift option for the energy of the Lennard-Jones portion of the pair interaction.

The *lj/sdk/coul/long* pair styles support the pair_modify table option since they can tabulate the short-range portion of the long-range Coulombic interaction.

pair_style lj/sdk/coul/long/omp command

All of the lj/sdk pair styles write their information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

The lj/sdk and lj/cut/coul/long pair styles do not support the use of the *inner*, *middle*, and *outer* keywords of the run_style respa command.

Restrictions:

All of the lj/sdk pair styles are part of the USER-CG-CMM package. The *lj/sdk/coul/long* style also requires the KSPACE package to be built (which is enabled by default). They are only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Related commands:

pair_coeff, angle_style sdk

Default: none

(Shinoda) Shinoda, DeVane, Klein, Mol Sim, 33, 27 (2007).

(DeVane) Shinoda, DeVane, Klein, Soft Matter, 4, 2453-2462 (2008).

pair_style soft command

pair_style soft/gpu command

pair_style soft/omp command

Syntax:

```
pair_style soft cutoff
```

- cutoff = global cutoff for soft interactions (distance units)

Examples:

```
pair_style soft 2.5
pair_coeff * * 10.0
pair_coeff 1 1 10.0 3.0

pair_style soft 2.5
pair_coeff * * 0.0
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Description:

Style *soft* computes pairwise interactions with the formula

$$E = A \left[1 + \cos \left(\frac{\pi r}{r_c} \right) \right] \qquad r < r_c$$

It is useful for pushing apart overlapping atoms, since it does not blow up as r goes to 0. A is a pre-factor that can be made to vary in time from the start to the end of the run (see discussion below), e.g. to start with a very soft potential and slowly harden the interactions over time. Rc is the cutoff. See the [fix nve/limit](#) command for another way to push apart overlapping atoms.

The following coefficients must be defined for each pair of atom types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- A (energy units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global soft cutoff is used.

IMPORTANT NOTE: The syntax for [pair_coeff](#) with a single A coeff is different in the current version of LAMMPS than in older versions which took two values, Astart and Astop, to ramp between them. This functionality is now available in a more general form through the [fix adapt](#) command, as explained below. Note that if you use an old input script and specify Astart and Astop without a cutoff, then LAMMPS will interpret that as A and a cutoff, which is probabably not what you want.

The fix adapt command can be used to vary A for one or more pair types over the course of a simulation, in which case pair_coeff settings for A must still be specified, but will be overridden. For example these commands will vary the prefactor A for all pairwise interactions from 0.0 at the beginning to 30.0 at the end of a run:

```
variable prefactor equal ramp(0,30)
fix 1 all adapt 1 pair soft a * * v_prefactor
```

Note that a formula defined by an equal-style variable can use the current timestep, elapsed time in the current run, elapsed time since the beginning of a series of runs, as well as access other variables.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is always mixed via a *geometric* rule. The cutoff is mixed according to the pair_modify mix value. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift option, since the pair interaction goes to 0.0 at the cutoff.

The pair_modify table and tail options are not relevant for this pair style.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

pair_coeff, fix nve/limit, fix adapt

Default: none

pair_style sph/artVisc/tensCorr command

Syntax:

```
pair_style sph/artVisc/tensCorr kernelstyle args keyword values ...
```

- sph/artVisc/tensCorr = name of this pair_style command
- kernelstyle = *cubicspline* or *wendland*
- args = list of arguments for a particular style

```
cubicspline or wendland args = h
   h = smoothing length
```

- zero or more keyword/value pairs may be appended to args
- keyword = *artVisc* or *tensCorr*

```
artVisc values = alpha beta eta
   alpha = free parameter to control shear viscosity
   beta = free parameter to control bulk viscosity
   eta = coefficient to avoid singularities
 tensCorr values = epsilon deltap
   epsilon = free parameter
   deltap = initial particle distribution
```

Examples:

```
pair_style sph/artVisc/tensCorr wendland 0.001 artVisc 1e-4 0 1e-8
pair_style sph/artVisc/tensCorr cubicspline 0.001 artVisc 1e-4 0 1e-8 tensCorr 0.2 1e-2
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

The *sph/artVisc/tensCorr* style uses the smoothed particle hydrodynamics (SPH) method according to Monaghan (1992). The acting force is calculated from the acceleration as stated in the equation:

$$\frac{d\vec{v}_a}{dt} = -\sum_b m_b \left(\frac{P_a}{\rho_a^2} + \frac{P_b}{\rho_b^2} \right) \vec{\nabla}_a W_{ab}$$

Whereas the indices a and b stand for particles, P_j stands for pressure and ρ_j for the density. W_{ab} represents the kernel, which is defined by the kernelstyle.

For kernelstyle cubicspline a piecewise defined, 3-order kernel is used:

$$W(r,h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leq q < 1 \\ \frac{1}{4}(2-q)^3 & 1 \leq q < 2 \\ 0 & 2 \leq q \end{cases}$$

The wendland kernel (Wendland,1995) is defined as

$$W(r, h) = \frac{1}{\pi h^3} \left(1 - \frac{q}{2}\right)^4 (2q + 1) \qquad 0 \le q \le 2$$

The smoothing length h is the most important parameter for SPH-calculations. It depends on initial particle spacing, initial density $_0$ and mass per particle m_j. In case that the smoothing length is about 1.2 times the initial particle spacing and it is a 3-dimensional cubic lattice (therefore the summation is over 57 particles), you can use the following equation [(Liu and Liu, 2003, p. 211-213)](#):

$$h_i^0 = \frac{3}{32\pi}^{1/3} \left(\frac{\sum_{j=1}^{57} m_j}{\rho_i^0}\right)$$

The atom style *sph/var* uses the input argument h as initial smoothing length for all particles. In case the atom style *sph* (per-type smoothing length) is used an additional per-type property *sl* must be defined, e.g.,

```
fix         m2 all property/global sl peratomtype 0.0012
```

For further details on the basics of the SPH-method we recommend the papers from Monaghan [(1992)](#), [(1994)](#), etc.

Optionally, this pairstyle can take into account the artificial viscosity proposed my Monaghan (1985), if the artVisc keyword is appended. In this case, $_{ab}$ is added to the bracket term in the above acceleration equation, where $_{ab}$ is given by

$$\Pi_{ab} = \begin{cases} \frac{-\alpha \bar{c}_{ab}\mu_{ab} + \beta\mu_{ab}^2}{\bar{\rho}_{ab}} & \vec{v}_{ab} \cdot \vec{r}_{ab} < 0 \\ 0 & \vec{v}_{ab} \cdot \vec{r}_{ab} \ge 0 \end{cases}$$

and

$$\mu_{ab} = \frac{h\vec{v}_{ab} \cdot \vec{r}_{ab}}{\vec{r}_{ab}^2 + \eta^2}$$

This expression produces a shear and bulk viscosity. The quadratic term enables simulation of high Mach number shocks. The parameter ^{2} prevents singularities. A good choice is normally $^2 = 0.01h^2$.

The choice of   and   should not be critical, although there are some aspects which you should take into account:

"In the present case, with negligible changes in the density [weakly compressible SPH], the viscosity is almost entirely shear viscosity with a viscosity coefficient approximately   hc." [(Monaghan, 1994)](#)

Bar-parameters like c_{ab} and $_{ab}$ are mean values of particle a and b. NOTE: $_{ab}$ is calculated, and for the calculation of c_{ab} the per-type property *speedOfSound* has to be defined, e.g.,

```
fix             m1 all property/global speedOfSound peratomtype 20.
```

By appending the keyword *tensCorr* you enable the tensile correction algorithm [(Monaghan, 2000)](#) which improves results in combination with negative pressures (e.g. EOS like Tait's equation). This method adds $R*(f_{ab})^n$ to the bracket term, where the factor R is related to the pressure and can be calculated by $R = R_a + R_b$. In case of negative pressures ($P_a < 0$) we use the rule

$$R_a = \frac{\epsilon \, |P_a|}{\rho_a^2}$$

otherwise R_a is zero. Typical values of *epsilon* are about 0.2. fab is calculated by

$$f_{ab} = \frac{W\,(r_{ab})}{W\,(\Delta p)}$$

where Δp denotes the initial particle spacing. NOTE: In a next version this calculation should be improved too.

Mixing, shift, table, tail correction, restart, rRESPA info:

The pair_modify mix, shift, table, and tail options are not relevant for sph pair styles.

These pair styles write their information to binary restart files, so a pair_style command does not need to be specified in an input script that reads a restart file.

These pair styles can only be used via the *pair* keyword of the run_style respa command. They do not support the *inner*, *middle*, *outer* keywords.

Restrictions:

...

Related commands:

pair_coeff

Default: none

(**Liu and Liu, 2003**) "Smoothed Particle Hydrodynamics: A Meshfree Particle Method", G. R. Liu and M. B. Liu, World Scientific, p. 449 (2003).

(**Monaghan, 1992**) "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

(**Monaghan, 1994**) J. J. Monaghan, Journal of Computational Physics, 110, p. 399-406 (1994).

(**Monaghan, 2000**) J. J. Monaghan, Journal of Computational Physics, 159, p. 290-311 (2000).

pair_style sph/heatconduction command

Syntax:

```
pair_style sph/heatconduction
```

Examples:

```
pair_style sph/heatconduction
pair_coeff * * 1.0 2.4
```

Description:

The sph/heatconduction style computes heat transport between SPH particles. The transport model is the diffusion euqation for the internal energy.

See [this PDF guide](#) to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- D diffusion coefficient (length^2/time units)
- h kernel function cutoff (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the [pair_modify](#) shift, table, and tail options.

This style does not write information to [binary restart files](#). Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#), pair_sph/rhosum

Default: none

pair_style sph command

Syntax:

pair_style sph kernelstyle args keyword values ...

- sph = name of this pair_style command
- kernelstyle = *cubicspline* or *wendland*
- args = list of arguments for a particular style

```
cubicspline or wendland args = h
    h = smoothing length
```
- zero or more keyword/value pairs may be appended to args
- keyword = *artVisc* or *tensCorr*

```
artVisc values = alpha beta cAB eta
    alpha = free parameter to control shear viscosity
    beta = free parameter to control bulk viscosity
    cAB = average speed of sound (velocity units)
    eta = coefficient to avoid singularities
  tensCorr values = epsilon
    epsilon = free parameter
```

Examples:

```
pair_style sph wendland 0.001 artVisc 1e-4 0 1484. 1e-8
pair_style sph cubicspline 0.001 artVisc 1e-4 0 480 1e-8 tensCorr 0.2
```

LIGGGHTS vs. LAMMPS Info:

This command is not available in LAMMPS.

Description:

The *sph* style use the smoothed particle hydrodynamics (SPH) method according to Monaghan: "Smoothed Particle Hydrodynamics", J. J. Monaghan, Annual Review of Astronomy and Astrophysics (1992), Volume 30, Pages 543-574. The acting force is calculated from the acceleration as stated in the equation:

$$\frac{d\vec{v}_a}{dt} = - \sum_b m_b \left(\frac{P_b}{\rho_b^2} + \frac{P_a}{\rho_a^2} \right) \vec{\nabla}_a W_{ab}$$

Whereas the indices a and b stand for particles, P_j stands for pressure and ρ_j for the density. W_{ab} represents the kernel, which is defined by the kernelstyle.

For kernelstyle *cubicspline* a piecewise defined, 3-order kernel is used:

$$W(r, h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3 & 0 \leq q \leq 1 \\ \frac{1}{4}(2 - q)^3 & 1 \leq q \leq 2 \\ 0 & q \geq 2 \end{cases}$$

The *wendland* kernel (Wendland,1995) is defined as

$$W(r,h) = \frac{21}{16\pi h^3}\left(1 - \frac{q}{2}\right)^4 (2q + 1) \qquad 0 \le q \le 2$$

The smoothing length h is the most important parameter for SPH-calculations. It depends on initial particle spacing, initial density ρ_0 and mass per particle m_j. In case that the smoothing length is about 1.2 times the initial particle spacing and it is a 3-dimensional cubic lattice (therefore the summation is over 57 particles), you can use the following equation (Liu and Liu, 2003, p. 211-213):

$$h_i^0 = \frac{3}{32\pi}^{1/3}\left(\frac{\sum_{j=1}^{57} m_j}{\rho_i^0}\right)$$

For further details on the basics of the SPH-method we recommend the papers from Monaghan 1992,1994, etc.

Optionally, this pairstyle can take into account the artificial viscosity proposed my Monaghan (1985), if the *artVisc* keyword is appended. In this case, Π_{ab} is added to the bracket term in the above acceleration equation, where Π_{ab} is given by

$$\Pi_{ab} = \begin{cases} \dfrac{-\alpha \bar{c}_{ab}\mu_{ab} + \beta \mu_{ab}^2}{\bar{\rho}_{ab}} & \vec{v}_{ab} \cdot \vec{r}_{ab} < 0 \\ 0 & \vec{v}_{ab} \cdot \vec{r}_{ab} > 0 \end{cases}$$

and

$$\mu_{ab} = \frac{h\vec{v}_{ab} \cdot \vec{r}_{ab}}{\vec{r}_{ab}^2 + \eta^2}.$$

This expression produces a shear and bulk viscosity. The quadratic term enables simulation of high Mach number shocks. The parameter η^2 prevents singularities. A good choice is normally $\eta^2=0.01h^2$. The choice of α and β should not be critical, although there are some aspects which you should take into account:

```
   In case of negligible changes in the density (weakly compressible SPH) it is almost only shear
```

Bar-parameters like c_{ab} and ρ_{ab} are mean values of particle a and b.
NOTE: μ_{ab} is calculated, whereas c_{ab} is a given parameter. In a next version speed of sound should be added as atom property.

By appending the keyword *tensCorr* you enable the tensile correction algorithm (Monaghan, 2000) which improves results in combination with negative pressures (e.g. EOS like Tait's equation). This method adds $R*(f_{ab})^n$ to the bracket term, where the factor R is related to the pressure and can be calculated by $R=R_a + R_b$. In case of negative pressures ($P_a < 0$) we use the rule

$$R_a = \frac{\epsilon |P_a|}{\rho_a^2}$$

otherwise R_a is zero. Typical values of *epsilon* are about 0.2.
f_{ab} is calculated by

$$f_{ab} = \frac{W(r_{ab})}{W(\Delta p)}$$

where Δp denotes the initial particle spacing.
NOTE: In a next version this calculation should be improved too.

Restart info:

Until now there is no restart option implemented.

Restrictions:

...

Related commands:

pair_coeff

Default: none

(Monaghan1992) J. J. Monaghan, Annu. Rev. Astron. Astrophys., 30, p. 543-574 (1992).

(Monaghan1994) J. J. Monaghan, Journal of Computational Physics, 110, p. 399-406 (1994).

(Monaghan2000) J. J. Monaghan, Journal of Computational Physics, 159, p. 290-311 (2000).

pair_style sph/idealgas command

Syntax:

```
pair_style sph/idealgas
```

Examples:

```
pair_style sph/idealgas
pair_coeff * * 1.0 2.4
```

Description:

The sph/idealgas style computes pressure forces between particles according to the ideal gas equation of state:

$$p = (\gamma - 1)\rho e$$

where gamma = 1.4 is the heat capacity ratio, rho is the local density, and e is the internal energy per unit mass. This pair style also computes Monaghan's artificial viscosity to prevent particles from interpentrating [(Monaghan)](#).

See [this PDF guide](#) to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- nu artificial viscosity (no units)
- h kernel function cutoff (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the [pair_modify](#) shift, table, and tail options.

This style does not write information to [binary restart files](#). Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#), pair_sph/rhosum

pair_style sph/idealgas command

Default: none

(Monaghan) Monaghan and Gingold, Journal of Computational Physics, 52, 374-389 (1983).

pair_style sph/lj command

Syntax:

```
pair_style sph/lj
```

Examples:

```
pair_style sph/lj
pair_coeff * * 1.0 2.4
```

Description:

The sph/lj style computes pressure forces between particles according to the Lennard-Jones equation of state, which is computed according to Ree's 1980 polynomial fit [(Ree)](#). The Lennard-Jones parameters epsilon and sigma are set to unity. This pair style also computes Monaghan's artificial viscosity to prevent particles from interpentrating [(Monaghan)](#).

See [this PDF guide](#) to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- nu artificial viscosity (no units)
- h kernel function cutoff (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the [pair_modify](#) shift, table, and tail options.

This style does not write information to [binary restart files](#). Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

As noted above, the Lennard-Jones parameters epsilon and sigma are set to unity.

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#), pair_sph/rhosum

Default: none

(Ree) Ree, Journal of Chemical Physics, 73, 5401 (1980).

(Monaghan) Monaghan and Gingold, Journal of Computational Physics, 52, 374-389 (1983).

pair_style sph/rhosum command

Syntax:

```
pair_style sph/rhosum Nstep
```

- Nstep = timestep interval

Examples:

```
pair_style sph/rhosum 10
pair_coeff * * 2.4
```

Description:

The sph/rhosum style computes the local particle mass density rho for SPH particles by kernel function interpolation, every Nstep timesteps.

See [this PDF guide](#) to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- h (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the [pair_modify](#) shift, table, and tail options.

This style does not write information to [binary restart files](#). Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](#), pair_sph/taitwater

Default: none

pair_style sph/taitwater command

Syntax:

```
pair_style sph/taitwater
```

Examples:

```
pair_style sph/taitwater
pair_coeff * * 1000.0 1430.0 1.0 2.4
```

Description:

The sph/taitwater style computes pressure forces between SPH particles according to Tait's equation of state:

$$p = B[(\frac{\rho}{\rho_0})^\gamma - 1]$$

where gamma = 7 and B = c_0^2 rho_0 / gamma, with rho_0 being the reference density and c_0 the reference speed of sound.

This pair style also computes Monaghan's artificial viscosity to prevent particles from interpentrating [(Monaghan)](#).

See [this PDF guide](#) to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- rho0 reference density (mass/volume units)
- c0 reference soundspeed (distance/time units)
- nu artificial viscosity (no units)
- h kernel function cutoff (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the [pair_modify](#) shift, table, and tail options.

This style does not write information to [binary restart files](#). Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the [Making LAMMPS](#) section for more info.

Related commands:

[pair_coeff](), pair_sph/rhosum

Default: none

(Monaghan) Monaghan and Gingold, Journal of Computational Physics, 52, 374-389 (1983).

pair_style sph/taitwater/morris command

Syntax:

```
pair_style sph/taitwater/morris
```

Examples:

```
pair_style sph/taitwater/morris
pair_coeff * * 1000.0 1430.0 1.0 2.4
```

Description:

The sph/taitwater/morris style computes pressure forces between SPH particles according to Tait's equation of state:

$$p = B[(\frac{\rho}{\rho_0})^{\gamma} - 1]$$

where gamma = 7 and B = c_0^2 rho_0 / gamma, with rho_0 being the reference density and c_0 the reference speed of sound.

This pair style also computes laminar viscosity (Morris).

See this PDF guide to using SPH in LAMMPS.

The following coefficients must be defined for each pair of atoms types via the pair_coeff command as in the examples above.

- rho0 reference density (mass/volume units)
- c0 reference soundspeed (distance/time units)
- nu dynamic viscosity (mass*distance/time units)
- h kernel function cutoff (distance units)

Mixing, shift, table, tail correction, restart, rRESPA info:

This style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

This style does not support the pair_modify shift, table, and tail options.

This style does not write information to binary restart files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the USER-SPH package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

pair_style sph/taitwater/morris command 927

Related commands:

[pair_coeff](), pair_sph/rhosum

Default: none

(**Morris**) Morris, Fox, Zhu, J Comp Physics, 136, 214-226 (1997).

pair_style command

Syntax:

```
pair_style style args
```

- style = one of the styles from the list below
- args = arguments used by a particular style

Examples:

```
pair_style lj/cut 2.5
pair_style eam/alloy
pair_style hybrid lj/charmm/coul/long 10.0 eam
pair_style table linear 1000
pair_style none
```

Description:

Set the formula(s) LAMMPS uses to compute pairwise interactions. In LAMMPS, pair potentials are defined between pairs of atoms that are within a cutoff distance and the set of active interactions typically changes over time. See the bond_style command to define potentials between pairs of bonded atoms, which typically remain in place for the duration of a simulation.

In LAMMPS, pairwise force fields encompass a variety of interactions, some of which include many-body effects, e.g. EAM, Stillinger-Weber, Tersoff, REBO potentials. They are still classified as "pairwise" potentials because the set of interacting atoms changes with time (unlike molecular bonds) and thus a neighbor list is used to find nearby interacting atoms.

Hybrid models where specified pairs of atom types interact via different pair potentials can be setup using the *hybrid* pair style.

The coefficients associated with a pair style are typically set for each pair of atom types, and are specified by the pair_coeff command or read from a file by the read_data or read_restart commands.

The pair_modify command sets options for mixing of type I-J interaction coefficients and adding energy offsets or tail corrections to Lennard-Jones potentials. Details on these options as they pertain to individual potentials are described on the doc page for the potential. Likewise, info on whether the potential information is stored in a restart file is listed on the potential doc page.

In the formulas listed for each pair style, *E* is the energy of a pairwise interaction between two atoms separated by a distance *r*. The force between the atoms is the negative derivative of this expression.

If the pair_style command has a cutoff argument, it sets global cutoffs for all pairs of atom types. The distance(s) can be smaller or larger than the dimensions of the simulation box.

Typically, the global cutoff value can be overridden for a specific pair of atom types by the pair_coeff command. The pair style settings (including global cutoffs) can be changed by a subsequent pair_style command using the same style. This will reset the cutoffs for all atom type pairs, including those previously set explicitly by a pair_coeff command. The exceptions to this are that pair_style *table* and *hybrid* settings cannot be reset. A new pair_style command for these styles will wipe out all previously specified pair_coeff values.

Here is an alphabetic list of pair styles defined in LAMMPS. Click on the style to display the formula it computes, arguments specified in the pair_style command, and coefficients specified by the associated pair_coeff command.

Note that there are also additional pair styles submitted by users which are included in the LAMMPS distribution. The list of these with links to the individual styles are given in the pair section of this page.

There are also additional accelerated pair styles included in the LAMMPS distribution for faster performance on CPUs and GPUs. The list of these with links to the individual styles are given in the pair section of this page.

- pair_style none - turn off pairwise interactions
- pair_style hybrid - multiple styles of pairwise interactions
- pair_style hybrid/overlay - multiple styles of superposed pairwise interactions

- pair_style adp - angular dependent potential (ADP) of Mishin
- pair_style airebo - AIREBO potential of Stuart
- pair_style beck - Beck potential
- pair_style body - interactions between body particles
- pair_style bop - BOP potential of Pettifor
- pair_style born - Born-Mayer-Huggins potential
- pair_style born/coul/long - Born-Mayer-Huggins with long-range Coulombics
- pair_style born/coul/msm - Born-Mayer-Huggins with long-range MSM Coulombics
- pair_style born/coul/wolf - Born-Mayer-Huggins with Coulombics via Wolf potential
- pair_style brownian - Brownian potential for Fast Lubrication Dynamics
- pair_style brownian/poly - Brownian potential for Fast Lubrication Dynamics with polydispersity
- pair_style buck - Buckingham potential
- pair_style buck/coul/cut - Buckingham with cutoff Coulomb
- pair_style buck/coul/long - Buckingham with long-range Coulombics
- pair_style buck/coul/msm - Buckingham long-range MSM Coulombics
- pair_style buck/long/coul/long - long-range Buckingham with long-range Coulombics
- pair_style colloid - integrated colloidal potential
- pair_style comb - charge-optimized many-body (COMB) potential
- pair_style coul/cut - cutoff Coulombic potential
- pair_style coul/debye - cutoff Coulombic potential with Debye screening
- pair_style coul/dsf - Coulombics via damped shifted forces
- pair_style coul/long - long-range Coulombic potential
- pair_style coul/msm - long-range MSM Coulombics
- pair_style coul/wolf - Coulombics via Wolf potential
- pair_style dipole/cut - point dipoles with cutoff
- pair_style dpd - dissipative particle dynamics (DPD)
- pair_style dpd/tstat - DPD thermostatting
- pair_style dsmc - Direct Simulation Monte Carlo (DSMC)
- pair_style eam - embedded atom method (EAM)
- pair_style eam/alloy - alloy EAM
- pair_style eam/fs - Finnis-Sinclair EAM
- pair_style eim - embedded ion method (EIM)
- pair_style gauss - Gaussian potential
- pair_style gayberne - Gay-Berne ellipsoidal potential
- pair_style gran/hertz/history - granular potential with Hertzian interactions
- pair_style gran/hooke - granular potential with history effects
- pair_style gran/hooke/history - granular potential without history effects
- pair_style hbond/dreiding/lj - DREIDING hydrogen bonding LJ potential
- pair_style hbond/dreiding/morse - DREIDING hydrogen bonding Morse potential

pair_style command

- [pair_style kim](#) - interface to potentials provided by KIM project
- [pair_style lcbop](#) - long-range bond-order potential (LCBOP)
- [pair_style line/lj](#) - LJ potential between line segments
- [pair_style lj/charmm/coul/charmm](#) - CHARMM potential with cutoff Coulomb
- [pair_style lj/charmm/coul/charmm/implicit](#) - CHARMM for implicit solvent
- [pair_style lj/charmm/coul/long](#) - CHARMM with long-range Coulomb
- [pair_style lj/charmm/coul/msm](#) - CHARMM with long-range MSM Coulombics
- [pair_style lj/class2](#) - COMPASS (class 2) force field with no Coulomb
- [pair_style lj/class2/coul/cut](#) - COMPASS with cutoff Coulomb
- [pair_style lj/class2/coul/long](#) - COMPASS with long-range Coulomb
- [pair_style lj/cut](#) - cutoff Lennard-Jones potential with no Coulomb
- [pair_style lj/cut/coul/cut](#) - LJ with cutoff Coulomb
- [pair_style lj/cut/coul/debye](#) - LJ with Debye screening added to Coulomb
- [pair_style lj/cut/coul/dsf](#) - LJ with Coulombics via damped shifted forces
- [pair_style lj/cut/coul/long](#) - LJ with long-range Coulombics
- [pair_style lj/cut/coul/msm](#) - LJ with long-range MSM Coulombics
- [pair_style lj/cut/tip4p/cut](#) - LJ with cutoff Coulomb for TIP4P water
- [pair_style lj/cut/tip4p/long](#) - LJ with long-range Coulomb for TIP4P water
- [pair_style lj/expand](#) - Lennard-Jones for variable size particles
- [pair_style lj/gromacs](#) - GROMACS-style Lennard-Jones potential
- [pair_style lj/gromacs/coul/gromacs](#) - GROMACS-style LJ and Coulombic potential
- [pair_style lj/long/coul/long](#) - long-range LJ and long-range Coulombics
- [pair_style lj/long/tip4p/long](#) - long-range LJ and long-range Coulomb for TIP4P water
- [pair_style lj/smooth](#) - smoothed Lennard-Jones potential
- [pair_style lj/smooth/linear](#) - linear smoothed Lennard-Jones potential
- [pair_style lj96/cut](#) - Lennard-Jones 9/6 potential
- [pair_style lubricate](#) - hydrodynamic lubrication forces
- [pair_style lubricate/poly](#) - hydrodynamic lubrication forces with polydispersity
- [pair_style lubricateU](#) - hydrodynamic lubrication forces for Fast Lubrication Dynamics
- [pair_style lubricateU/poly](#) - hydrodynamic lubrication forces for Fast Lubrication with polydispersity
- [pair_style meam](#) - modified embedded atom method (MEAM)
- [pair_style mie/cut](#) - Mie potential
- [pair_style morse](#) - Morse potential
- [pair_style peri/lps](#) - peridynamic LPS potential
- [pair_style peri/pmb](#) - peridynamic PMB potential
- [pair_style reax](#) - ReaxFF potential
- [pair_style rebo](#) - 2nd generation REBO potential of Brenner
- [pair_style resquared](#) - Everaers RE-Squared ellipsoidal potential
- [pair_style soft](#) - Soft (cosine) potential
- [pair_style sw](#) - Stillinger-Weber 3-body potential
- [pair_style table](#) - tabulated pair potential
- [pair_style tersoff](#) - Tersoff 3-body potential
- [pair_style tersoff/zbl](#) - Tersoff/ZBL 3-body potential
- [pair_style tri/lj](#) - LJ potential between triangles
- [pair_style yukawa](#) - Yukawa potential
- [pair_style yukawa/colloid](#) - screened Yukawa potential for finite-size particles
- [pair_style zbl](#) - Ziegler-Biersack-Littmark potential

Restrictions:

This command must be used before any coefficients are set by the [pair_coeff](#), [read_data](#), or [read_restart](#) commands.

Some pair styles are part of specific packages. They are only enabled if LAMMPS was built with that

package. See the [Making LAMMPS](#) section for more info on packages. The doc pages for individual pair potentials tell if it is part of a package.

Related commands:

[pair_coeff](#), [read_data](#), [pair_modify](#), [kspace_style](#), [dielectric](#), [pair_write](#)

Default:

```
pair_style none
```

pair_style sw command

pair_style sw/cuda command

pair_style sw/gpu command

pair_style sw/omp command

Syntax:

```
pair_style sw
```

Examples:

```
pair_style sw
pair_coeff * * si.sw Si
pair_coeff * * GaN.sw Ga N Ga
```

Description:

The *sw* style computes a 3-body [Stillinger-Weber](#) potential for the energy E of a system of atoms as

$$
E = \sum_i \sum_{j>i} \phi_2(r_{ij}) + \sum_i \sum_{j \neq i} \sum_{k>j} \phi_3(r_{ij}, r_{ik}, \theta_{ijk})
$$

$$
\phi_2(r_{ij}) = A_{ij}\epsilon_{ij}\left[B_{ij}(\frac{\sigma_{ij}}{r_{ij}})^{p_{ij}} - (\frac{\sigma_{ij}}{r_{ij}})^{q_{ij}}\right]\exp\left(\frac{\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right)
$$

$$
\phi_3(r_{ij}, r_{ik}, \theta_{ijk}) = \lambda_{ijk}\epsilon_{ijk}\left[\cos\theta_{ijk} - \cos\theta_{0ijk}\right]^2\exp\left(\frac{\gamma_{ij}\sigma_{ij}}{r_{ij} - a_{ij}\sigma_{ij}}\right)\exp\left(\frac{\gamma_{ik}\sigma}{r_{ik} - a}\right.
$$

where phi2 is a two-body term and phi3 is a three-body term. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = a*sigma.

Only a single pair_coeff command is used with the *sw* style which specifies a Stillinger-Weber potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of SW elements to atom types

See the [pair_coeff](#) doc page for alternate ways to specify the path for the potential file.

As an example, imagine a file SiC.sw has Stillinger-Weber values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.sw Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the SW file. The final C argument maps LAMMPS atom type 4 to the C element in the SW file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *sw* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Stillinger-Weber files in the *potentials* directory of the LAMMPS distribution have a ".sw" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to the two-body and three-body coefficients in the formula above:

- element 1 (the center atom in a 3-body interaction)
- element 2
- element 3
- epsilon (energy units)
- sigma (distance units)
- a
- lambda
- gamma
- costheta0
- A
- B
- p
- q
- tol

The A, B, p, and q parameters are used only for two-body interactions. The lambda and costheta0 parameters are used only for three-body interactions. The epsilon, sigma and a parameters are used for both two-body and three-body interactions. gamma is used only in the three-body interactions, but is defined for pairs of atoms. The non-annotated parameters are unitless.

LAMMPS introduces an additional performance-optimization parameter tol that is used for both two-body and three-body interactions. In the Stillinger-Weber potential, the interaction energies become negligibly small at atomic separations substantially less than the theoretical cutoff distances. LAMMPS therefore defines a virtual cutoff distance based on a user defined tolerance tol. The use of the virtual cutoff distance in constructing atom neighbor lists can significantly reduce the neighbor list sizes and therefore the computational cost. LAMMPS provides a *tol* value for each of the three-body entries so that they can be separately controlled. If tol = 0.0, then the standard Stillinger-Weber cutoff is used.

The Stillinger-Weber potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single-element simulation, only a single entry is required (e.g. SiSiSi). For a two-element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify SW parameters for all permutations of the two elements interacting in three-body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three-body interaction. Thus an entry for SiCC means a Si atom with 2 C atoms as neighbors. The parameter values used for the two-body interaction come from the entry where the 2nd and 3rd elements are the same. Thus the two-body parameters for Si interacting with C, comes from the SiCC entry. The three-body parameters can in principle be specific to the three elements of the configuration. In the literature, however, the three-body parameters are usually defined by simple formulas involving two sets of pair-wise parameters, corresponding to the ij and ik pairs,

where i is the center atom. The user must ensure that the correct combining rule is used to calculate the values of the threebody parameters for alloys. Note also that the function phi3 contains two exponential screening factors with parameter values from the ij pair and ik pairs. So phi3 for a C atom bonded to a Si atom and a second C atom will depend on the three-body parameters for the CSiC entry, and also on the two-body parameters for the CCC and CSiSi entries. Since the order of the two neighbors is arbitrary, the threebody parameters for entries CSiC and CCSi should be the same. Similarly, the two-body parameters for entries SiCC and CSiSi should also be the same. The parameters used only for two-body interactions (A, B, p, and q) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired. This is also true for the parameters in phi3 that are taken from the ij and ik pairs (sigma, a, gamma)

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Stillinger-Weber potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the SW potential with any LAMMPS units, but you would need to create your own SW potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

The sw/gpu style is currently limited to a single element.

Related commands:

pair_style sw/omp command

[pair_coeff](#)

Default: none

(Stillinger) Stillinger and Weber, Phys Rev B, 31, 5262 (1985).

pair_style table command

pair_style table/gpu command

pair_style table/omp command

Syntax:

```
pair_style table style N keyword ...
```

- style = *lookup* or *linear* or *spline* or *bitmap* = method of interpolation
- N = use N values in *lookup*, *linear*, *spline* tables
- N = use 2^N values in *bitmap* tables
- zero or more keywords may be appended
- keyword = *ewald* or *pppm* or *msm* or *dispersion* or *tip4p*

Examples:

```
pair_style table linear 1000
pair_style table linear 1000 pppm
pair_style table bitmap 12
pair_coeff * 3 morse.table ENTRY1
pair_coeff * 3 morse.table ENTRY1 7.0
```

Description:

Style *table* creates interpolation tables of length *N* from pair potential and force values listed in a file(s) as a function of distance. The files are read by the [pair_coeff](#) command.

The interpolation tables are created by fitting cubic splines to the file values and interpolating energy and force values at each of *N* distances. During a simulation, these tables are used to interpolate energy and force values as needed. The interpolation is done in one of 4 styles: *lookup*, *linear*, *spline*, or *bitmap*.

For the *lookup* style, the distance between 2 atoms is used to find the nearest table entry, which is the energy or force.

For the *linear* style, the pair distance is used to find 2 surrounding table values from which an energy or force is computed by linear interpolation.

For the *spline* style, a cubic spline coefficients are computed and stored at each of the *N* values in the table. The pair distance is used to find the appropriate set of coefficients which are used to evaluate a cubic polynomial which computes the energy or force.

For the *bitmap* style, the N means to create interpolation tables that are 2^N in length. (Wolff) and a linear interpolation is performed between adjacent table values.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above.

- filename
- keyword
- cutoff (distance units)

The filename specifies a file containing tabulated energy and force values. The keyword specifies a section of the file. The cutoff is an optional coefficient. If not specified, the outer cutoff in the table itself (see below) will be used to build an interpolation table that extend to the largest tabulated distance. If specified, only file values up to the cutoff are used to create the interpolation table. The format of this file is described below.

If your tabulated potential(s) are designed to be used as the short-range part of one of the long-range solvers specified by the kspace_style command, then you must use one or more of the optional keywords listed above for the pair_style command. These are *ewald* or *pppm* or *msm* or *dispersion* or *tip4p*. This is so LAMMPS can insure the short-range potential and long-range solver are compatible with each other, as it does for other short-range pair styles, such as pair_style lj/cut/coul/long. Note that it is up to you to insure the tabulated values for each pair of atom types has the correct functional form to be compatible with the matching long-range solver.

Here are some guidelines for using the pair_style table command to best effect:

- Vary the number of table points; you may need to use more than you think to get good resolution.
- Always use the pair_write command to produce a plot of what the final interpolated potential looks like. This can show up interpolation "features" you may not like.
- Start with the linear style; it's the style least likely to have problems.
- Use *N* in the pair_style command equal to the "N" in the tabulation file, and use the "RSQ" or "BITMAP" parameter, so additional interpolation is not needed. See discussion below.
- Make sure that your tabulated forces and tabulated energies are consistent (dE/dr = -F) along the entire range of r values.
- Use as large an inner cutoff as possible. This avoids fitting splines to very steep parts of the potential.

The format of a tabulated file is as follows (without the parenthesized comments):

```
# Morse potential for Fe    (one or more comment or blank lines)

MORSE_FE                    (keyword is first text on line)
N 500 R 1.0 10.0            (N, R, RSQ, BITMAP, FPRIME parameters)
                            (blank)
1 1.0 25.5 102.34           (index, r, energy, force)
2 1.02 23.4 98.5
...
500 10.0 0.001 0.003
```

A section begins with a non-blank line whose 1st character is not a "#"; blank lines or lines starting with "#" can be used as comments between sections. The first line begins with a keyword which identifies the section. The line can contain additional text, but the initial text must match the argument specified in the pair_coeff command. The next line lists (in any order) one or more parameters for the table. Each parameter is a keyword followed by one or more numeric values.

The parameter "N" is required and its value is the number of table entries that follow. Note that this may be different than the *N* specified in the pair_style table command. Let Ntable = *N* in the pair_style command, and Nfile = "N" in the tabulated file. What LAMMPS does is a preliminary interpolation by creating splines using the Nfile tabulated values as nodal points. It uses these to interpolate as needed to generate energy and force values at Ntable different points. The resulting tables of length Ntable are then used as described above, when computing energy and force for individual pair distances. This means that if you want the interpolation tables of length Ntable to match exactly what is in the tabulated file (with effectively no preliminary interpolation), you should set Ntable = Nfile, and use the "RSQ" or "BITMAP" parameter. The internal table abscissa is RSQ (separation distance squared).

All other parameters are optional. If "R" or "RSQ" or "BITMAP" does not appear, then the distances in each line of the table are used as-is to perform spline interpolation. In this case, the table values can be spaced in *r* uniformly or however you wish to position table values in regions of large gradients.

If used, the parameters "R" or "RSQ" are followed by 2 values *rlo* and *rhi*. If specified, the distance associated with each energy and force value is computed from these 2 values (at high accuracy), rather than using the (low-accuracy) value listed in each line of the table. The distance values in the table file are ignored in this case. For "R", distances uniformly spaced between *rlo* and *rhi* are computed; for "RSQ", squared distances uniformly spaced between *rlo*rlo* and *rhi*rhi* are computed.

If used, the parameter "BITMAP" is also followed by 2 values *rlo* and *rhi*. These values, along with the "N" value determine the ordering of the N lines that follow and what distance is associated with each. This ordering is complex, so it is not documented here, since this file is typically produced by the pair_write command with its *bitmap* option. When the table is in BITMAP format, the "N" parameter in the file must be equal to 2^M where M is the value specified in the pair_style command. Also, a cutoff parameter cannot be used as an optional 3rd argument in the pair_coeff command; the entire table extent as specified in the file must be used.

If used, the parameter "FPRIME" is followed by 2 values *fplo* and *fphi* which are the derivative of the force at the innermost and outermost distances listed in the table. These values are needed by the spline construction routines. If not specified by the "FPRIME" parameter, they are estimated (less accurately) by the first 2 and last 2 force values in the table. This parameter is not used by BITMAP tables.

Following a blank line, the next N lines list the tabulated values. On each line, the 1st value is the index from 1 to N, the 2nd value is r (in distance units), the 3rd value is the energy (in energy units), and the 4th is the force (in force units). The r values must increase from one line to the next (unless the BITMAP parameter is specified).

Note that one file can contain many sections, each with a tabulated potential. LAMMPS reads the file section by section until it finds one that matches the specified keyword.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support mixing. Thus, coefficients for all I,J pairs must be specified explicitly.

The pair_modify shift, table, and tail options are not relevant for this pair style.

This pair style writes the settings for the "pair_style table" command to binary restart files, so a pair_style command does not need to specified in an input script that reads a restart file. However, the coefficient information is not stored in the restart file, since it is tabulated in the potential files. Thus, pair_coeff commands do need to be specified in the restart input script.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

(Wolff) Wolff and Rudd, Comp Phys Comm, 120, 200-32 (1999).

pair_style tersoff command

pair_style tersoff/table command

pair_style tersoff/cuda

pair_style tersoff/omp

pair_style tersoff/table/omp command

Syntax:

```
pair_style style
```

style = *tersoff* or *tersoff/table* or *tersoff/cuda* or *tersoff/omp* or *tersoff/table/omp*

Examples:

```
pair_style tersoff
pair_coeff * * Si.tersoff Si
pair_coeff * * SiC.tersoff Si C Si

pair_style tersoff/table
pair_coeff * * SiCGe.tersoff Si(D)
```

Description:

The *tersoff* style computes a 3-body Tersoff potential [(Tersoff_1)](#) for the energy E of a system of atoms as

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} V_{ij}$$

$$V_{ij} = f_C(r_{ij})\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right]$$

$$f_C(r) = \begin{cases} 1 & : & r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin\left(\frac{\pi}{2}\frac{r-R}{D}\right) & : & R - D < r < R + D \\ 0 & : & r > R + D \end{cases}$$

$$f_R(r) = A\exp(-\lambda_1 r)$$

$$f_A(r) = -B\exp(-\lambda_2 r)$$

$$b_{ij} = (1 + \beta^n \zeta_{ij}^n)^{-\frac{1}{2n}}$$

$$\zeta_{ij} = \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp\left[\lambda_3^m (r_{ij} - r_{ik})^m\right]$$

$$g(\theta) = \gamma_{ijk}\left(1 + \frac{c^2}{d^2} - \frac{c^2}{[d^2 + (\cos\theta - \cos\theta_0)^2]}\right)$$

where f_R is a two-body term and f_A includes three-body interactions. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = R + D.

The *tersoff/table* style uses tabulated forms for the two-body, environment and angular functions. Linear interpolation is performed between adjacent table entries. The table length is chosen to be accurate within 10^{-6} with respect to the *tersoff* style energy. The *tersoff/table* should give better performance in terms of speed.

Only a single pair_coeff command is used with the *tersoff* style which specifies a Tersoff potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of Tersoff elements to atom types

See the [pair_coeff](#) doc page for alternate ways to specify the path for the potential file.

As an example, imagine the SiC.tersoff file has Tersoff values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.tersoff Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the Tersoff file. The final C argument maps LAMMPS atom type 4 to the C element in the Tersoff file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *tersoff* potential is used as part of the *hybrid* pair style. The NULL values

pair_style tersoff/table/omp command

are placeholders for atom types that will be used with other potentials.

Tersoff files in the *potentials* directory of the LAMMPS distribution have a ".tersoff" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3-body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1-2 bond in a bond-order sense)
- m
- gamma
- lambda3 (1/distance units)
- c
- d
- costheta0 (can be a value < -1 or > 1)
- n
- beta
- lambda2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- lambda1 (1/distance units)
- A (energy units)

The n, beta, lambda2, B, lambda1, and A parameters are only used for two-body interactions. The m, gamma, lambda3, c, d, and costheta0 parameters are only used for three-body interactions. The R and D parameters are used for both two-body and three-body interactions. The non-annotated parameters are unitless. The value of m must be 3 or 1.

The Tersoff potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single-element simulation, only a single entry is required (e.g. SiSiSi). For a two-element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify Tersoff parameters for all permutations of the two elements interacting in three-body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three-body interaction and it is bonded to the 2nd atom and the bond is influenced by the 3rd atom. Thus an entry for SiCC means Si bonded to a C with another C atom influencing the bond. Thus three-body parameters for SiCSi and SiSiC entries will not, in general, be the same. The parameters used for the two-body interaction come from the entry where the 2nd element is repeated. Thus the two-body parameters for Si interacting with C, comes from the SiCC entry.

The parameters used for a particular three-body interaction come from the entry with the corresponding three elements. The parameters used only for two-body interactions (n, beta, lambda2, B, lambda1, and A) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired.

Note that the twobody parameters in entries such as SiCC and CSiSi are often the same, due to the common use of symmetric mixing rules, but this is not always the case. For example, the beta and n parameters in Tersoff_2 [(Tersoff_2)](Tersoff_2) are not symmetric.

We chose the above form so as to enable users to define all commonly used variants of the Tersoff potential. In particular, our form reduces to the original Tersoff form when m = 3 and gamma = 1, while it reduces to

the form of Albe et al. when beta = 1 and m = 1. Note that in the current Tersoff implementation in LAMMPS, m must be specified as either 3 or 1. Tersoff used a slightly different but equivalent form for alloys, which we will refer to as Tersoff_2 potential (Tersoff_2). The *tersoff/table* style implements Tersoff_2 parameterization only.

LAMMPS parameter values for Tersoff_2 can be obtained as follows: gamma_ijk = omega_ik, lambda3 = 0 and the value of m has no effect. The parameters for species i and j can be calculated using the Tersoff_2 mixing rules:

$$
\begin{aligned}
\lambda_1^{i,j} &= \frac{1}{2}(\lambda_1^i + \lambda_1^j) \\
\lambda_2^{i,j} &= \frac{1}{2}(\lambda_2^i + \lambda_2^j) \\
A_{i,j} &= (A_i A_j)^{1/2} \\
B_{i,j} &= \chi_{ij}(B_i B_j)^{1/2} \\
R_{i,j} &= (R_i R_j)^{1/2} \\
S_{i,j} &= (S_i S_j)^{1/2}
\end{aligned}
$$

Tersoff_2 parameters R and S must be converted to the LAMMPS parameters R and D (R is different in both forms), using the following relations: R=(R'+S')/2 and D=(S'-R')/2, where the primes indicate the Tersoff_2 parameters.

In the potentials directory, the file SiCGe.tersoff provides the LAMMPS parameters for Tersoff's various versions of Si, as well as his alloy parameters for Si, C, and Ge. This file can be used for pure Si, (three different versions), pure C, pure Ge, binary SiC, and binary SiGe. LAMMPS will generate an error if this file is used with any combination involving C and Ge, since there are no entries for the GeC interactions (Tersoff did not publish parameters for this cross-interaction.) Tersoff files are also provided for the SiC alloy (SiC.tersoff) and the GaN (GaN.tersoff) alloys.

Many thanks to Rutuparna Narulkar, David Farrell, and Xiaowang Zhou for helping clarify how Tersoff parameters for alloys have been defined in various papers.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the [pair_modify](#) shift, table, and tail options.

This pair style does not write its information to [binary restart files](#), since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the [Making LAMMPS](#) section for more info.

This pair style requires the [newton](#) setting to be "on" for pair interactions.

The Tersoff potential files provided with LAMMPS (see the potentials directory) are parameterized for metal [units](#). You can use the Tersoff potential with any LAMMPS units, but you would need to create your own Tersoff potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

[pair_coeff](#)

Default: none

(Tersoff_1) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(Albe) J. Nord, K. Albe, P. Erhart, and K. Nordlund, J. Phys.: Condens. Matter, 15, 5649(2003).

(Tersoff_2) J. Tersoff, Phys Rev B, 39, 5566 (1989); errata (PRB 41, 3248)

pair_style tersoff/mod command

pair_style tersoff/mod/omp command

Syntax:

```
pair_style tersoff/mod
```

Examples:

```
pair_style tersoff/mod
pair_coeff * * Si.tersoff.mod Si Si
```

Description:

The *tersoff/mod* style computes a bond-order type interatomic potential [(Kumagai)](#) based on a 3-body Tersoff potential [(Tersoff_1)](#), [(Tersoff_2)](#) with modified cutoff function and angular-dependent term, giving the energy E of a system of atoms as

$$
\begin{aligned}
E &= \frac{1}{2} \sum_i \sum_{j \neq i} V_{ij} \\
V_{ij} &= f_C(r_{ij}) \left[f_R(r_{ij}) + b_{ij} f_A(r_{ij}) \right] \\
f_C(r) &= \begin{cases} 1 & : \quad r < R - D \\ \frac{1}{2} - \frac{9}{16} \sin\left(\frac{\pi}{2} \frac{r-R}{D}\right) - \frac{1}{16} \sin\left(\frac{3\pi}{2} \frac{r-R}{D}\right) & : \quad R - D < r < R + D \\ 0 & : \quad r > R + D \end{cases} \\
f_R(r) &= A \exp(-\lambda_1 r) \\
f_A(r) &= -B \exp(-\lambda_2 r) \\
b_{ij} &= \left(1 + \zeta_{ij}^{\eta}\right)^{-\frac{1}{2n}} \\
\zeta_{ij} &= \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp\left[\alpha(r_{ij} - r_{ik})^{\beta}\right] \\
g(\theta) &= c_1 + g_o(\theta) g_a(\theta) \\
g_o(\theta) &= \frac{c_2 (h - \cos\theta)^2}{c_3 + (h - \cos\theta)^2} \\
g_a(\theta) &= 1 + c_4 \exp\left[-c_5 (h - \cos\theta)^2\right]
\end{aligned}
$$

where f_R is a two-body term and f_A includes three-body interactions. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = R + D.

The modified cutoff function f_C proposed by (Murty) and having a continuous second-order differential is employed. The angular-dependent term g(theta) was modified to increase the flexibility of the potential.

The *tersoff/mod* potential is fitted to both the elastic constants and melting point by employing the modified Tersoff potential function form in which the angular-dependent term is improved. The model performs extremely well in describing the crystalline, liquid, and amorphous phases (Schelling).

Only a single pair_coeff command is used with the *tersoff/mod* style which specifies a Tersoff/MOD potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of Tersoff/MOD elements to atom types

As an example, imagine the Si.tersoff_mod file has Tersoff values for Si. If your LAMMPS simulation has 3 Si atoms types, you would use the following pair_coeff command:

```
pair_coeff * * Si.tersoff_mod Si Si Si
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the Tersoff/MOD file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *tersoff/mod* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Tersoff/MOD file in the *potentials* directory of the LAMMPS distribution have a ".tersoff.mod" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3-body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1-2 bond in a bond-order sense)
- beta
- alpha
- h
- eta
- beta_ters = 1 (dummy parameter)
- lambda2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- lambda1 (1/distance units)
- A (energy units)
- n
- c1
- c2
- c3
- c4
- c5

The n, eta, lambda2, B, lambda1, and A parameters are only used for two-body interactions. The beta, alpha, c1, c2, c3, c4, c5, h parameters are only used for three-body interactions. The R and D parameters are used for

pair_style tersoff/mod/omp command

both two-body and three-body interactions. The non-annotated parameters are unitless.

The Tersoff/MOD potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single-element simulation, only a single entry is required (e.g. SiSiSi). As annotated above, the first element in the entry is the center atom in a three-body interaction and it is bonded to the 2nd atom and the bond is influenced by the 3rd atom. Thus an entry for SiSiSi means Si bonded to a Si with another Si atom influencing the bond.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Tersoff/MOD potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the Tersoff/MOD potential with any LAMMPS units, but you would need to create your own Tersoff/MOD potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

Related commands:

pair_coeff

Default: none

(Kumagai) T. Kumagai, S. Izumi, S. Hara, S. Sakai, Comp. Mat. Science, 39, 457 (2007).

(Tersoff_1) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(Tersoff_2) J. Tersoff, Phys Rev B, 38, 9902 (1988).

(Murty) M.V.R. Murty, H.A. Atwater, Phys Rev B, 51, 4889 (1995).

(Schelling) Patrick K. Schelling, Comp. Mat. Science, 44, 274 (2008).

pair_style tersoff/zbl command

pair_style tersoff/zbl/omp command

Syntax:

```
pair_style tersoff/zbl
```

Examples:

```
pair_style tersoff/zbl
pair_coeff * * SiC.tersoff.zbl Si C Si
```

Description:

The *tersoff/zbl* style computes a 3-body Tersoff potential [(Tersoff_1)](#) with a close-separation pairwise modification based on a Coulomb potential and the Ziegler-Biersack-Littmark universal screening function [(ZBL)](#), giving the energy E of a system of atoms as

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} V_{ij}$$

$$V_{ij} = (1 - f_F(r_{ij}))V_{ij}^{ZBL} + f_F(r_{ij})V_{ij}^{Tersoff}$$

$$f_F(r_{ij}) = \frac{1}{1 + e^{-A_F(r_{ij} - r_C)}}$$

$$V_{ij}^{ZBL} = \frac{1}{4\pi\epsilon_0} \frac{Z_1 Z_2 e^2}{r_{ij}} \phi(r_{ij}/a)$$

$$a = \frac{0.8854\, a_0}{Z_1^{0.23} + Z_2^{0.23}}$$

$$\phi(x) = 0.1818 e^{-3.2x} + 0.5099 e^{-0.9423x} + 0.2802 e^{-0.4029x} + 0.02817 e^{-0.20}$$

$$V_{ij}^{Tersoff} = f_C(r_{ij})\left[f_R(r_{ij}) + b_{ij} f_A(r_{ij})\right]$$

$$f_C(r) = \begin{cases} 1 & : & r < R - D \\ \frac{1}{2} - \frac{1}{2}\sin\left(\frac{\pi}{2}\frac{r-R}{D}\right) & : & R - D < r < R + D \\ 0 & : & r > R + D \end{cases}$$

$$f_R(r) = A \exp(-\lambda_1 r)$$

$$f_A(r) = -B \exp(-\lambda_2 r)$$

$$b_{ij} = (1 + \beta^n \zeta_{ij}^n)^{-\frac{1}{2n}}$$

$$\zeta_{ij} = \sum_{k \neq i,j} f_C(r_{ik}) g(\theta_{ijk}) \exp\left[\lambda_3^3 (r_{ij} - r_{ik})^m\right]$$

$$g(\theta) = \gamma_{ijk}\left(1 + \frac{c^2}{d^2} - \frac{c^2}{[d^2 + (\cos\theta - \cos\theta_0)^2]}\right)$$

The f_F term is a fermi-like function used to smoothly connect the ZBL repulsive potential with the Tersoff potential. There are 2 parameters used to adjust it: A_F and r_C. A_F controls how "sharp" the transition is between the two, and r_C is essentially the cutoff for the ZBL potential.

pair_style tersoff/zbl/omp command

For the ZBL portion, there are two terms. The first is the Coulomb repulsive term, with Z1, Z2 as the number of protons in each nucleus, e as the electron charge (1 for metal and real units) and epsilon0 as the permittivity of vacuum. The second part is the ZBL universal screening function, with a0 being the Bohr radius (typically 0.529 Angstroms), and the remainder of the coefficients provided by the original paper. This screening function should be applicable to most systems. However, it is only accurate for small separations (i.e. less than 1 Angstrom).

For the Tersoff portion, f_R is a two-body term and f_A includes three-body interactions. The summations in the formula are over all neighbors J and K of atom I within a cutoff distance = R + D.

Only a single pair_coeff command is used with the *tersoff/zbl* style which specifies a Tersoff/ZBL potential file with parameters for all needed elements. These are mapped to LAMMPS atom types by specifying N additional arguments after the filename in the pair_coeff command, where N is the number of LAMMPS atom types:

- filename
- N element names = mapping of Tersoff/ZBL elements to atom types

See the [pair_coeff](#) doc page for alternate ways to specify the path for the potential file.

As an example, imagine the SiC.tersoff.zbl file has Tersoff/ZBL values for Si and C. If your LAMMPS simulation has 4 atoms types and you want the 1st 3 to be Si, and the 4th to be C, you would use the following pair_coeff command:

```
pair_coeff * * SiC.tersoff Si Si Si C
```

The 1st 2 arguments must be * * so as to span all LAMMPS atom types. The first three Si arguments map LAMMPS atom types 1,2,3 to the Si element in the Tersoff/ZBL file. The final C argument maps LAMMPS atom type 4 to the C element in the Tersoff/ZBL file. If a mapping value is specified as NULL, the mapping is not performed. This can be used when a *tersoff/zbl* potential is used as part of the *hybrid* pair style. The NULL values are placeholders for atom types that will be used with other potentials.

Tersoff/ZBL files in the *potentials* directory of the LAMMPS distribution have a ".tersoff.zbl" suffix. Lines that are not blank or comments (starting with #) define parameters for a triplet of elements. The parameters in a single entry correspond to coefficients in the formula above:

- element 1 (the center atom in a 3-body interaction)
- element 2 (the atom bonded to the center atom)
- element 3 (the atom influencing the 1-2 bond in a bond-order sense)
- m
- gamma
- lambda3 (1/distance units)
- c
- d
- costheta0 (can be a value < -1 or > 1)
- n
- beta
- lambda2 (1/distance units)
- B (energy units)
- R (distance units)
- D (distance units)
- lambda1 (1/distance units)
- A (energy units)
- Z_i
- Z_j

pair_style tersoff/zbl/omp command

- ZBLcut (distance units)
- ZBLexpscale (1/distance units)

The n, beta, lambda2, B, lambda1, and A parameters are only used for two-body interactions. The m, gamma, lambda3, c, d, and costheta0 parameters are only used for three-body interactions. The R and D parameters are used for both two-body and three-body interactions. The Z_i,Z_j, ZBLcut, ZBLexpscale parameters are used in the ZBL repulsive portion of the potential and in the Fermi-like function. The non-annotated parameters are unitless. The value of m must be 3 or 1.

The Tersoff/ZBL potential file must contain entries for all the elements listed in the pair_coeff command. It can also contain entries for additional elements not being used in a particular simulation; LAMMPS ignores those entries.

For a single-element simulation, only a single entry is required (e.g. SiSiSi). For a two-element simulation, the file must contain 8 entries (for SiSiSi, SiSiC, SiCSi, SiCC, CSiSi, CSiC, CCSi, CCC), that specify Tersoff parameters for all permutations of the two elements interacting in three-body configurations. Thus for 3 elements, 27 entries would be required, etc.

As annotated above, the first element in the entry is the center atom in a three-body interaction and it is bonded to the 2nd atom and the bond is influenced by the 3rd atom. Thus an entry for SiCC means Si bonded to a C with another C atom influencing the bond. Thus three-body parameters for SiCSi and SiSiC entries will not, in general, be the same. The parameters used for the two-body interaction come from the entry where the 2nd element is repeated. Thus the two-body parameters for Si interacting with C, comes from the SiCC entry. By symmetry, the twobody parameters in the SiCC and CSiSi entries should thus be the same. The parameters used for a particular three-body interaction come from the entry with the corresponding three elements. The parameters used only for two-body interactions (n, beta, lambda2, B, lambda1, and A) in entries whose 2nd and 3rd element are different (e.g. SiCSi) are not used for anything and can be set to 0.0 if desired.

We chose the above form so as to enable users to define all commonly used variants of the Tersoff portion of the potential. In particular, our form reduces to the original Tersoff form when m = 3 and gamma = 1, while it reduces to the form of Albe et al. when beta = 1 and m = 1. Note that in the current Tersoff implementation in LAMMPS, m must be specified as either 3 or 1. Tersoff used a slightly different but equivalent form for alloys, which we will refer to as Tersoff_2 potential (Tersoff_2).

LAMMPS parameter values for Tersoff_2 can be obtained as follows: gamma = 1, just as for Tersoff_1, but now lambda3 = 0 and the value of m has no effect. The parameters for species i and j can be calculated using the Tersoff_2 mixing rules:

$$\lambda_1^{i,j} = \frac{1}{2}(\lambda_1^i + \lambda_1^j)$$

$$\lambda_2^{i,j} = \frac{1}{2}(\lambda_2^i + \lambda_2^j)$$

$$A_{i,j} = (A_i A_j)^{1/2}$$

$$B_{i,j} = \chi_{ij}(B_i B_j)^{1/2}$$

$$R_{i,j} = (R_i R_j)^{1/2}$$

$$S_{i,j} = (S_i S_j)^{1/2}$$

Values not shown are determined by the first atom type. Finally, the Tersoff_2 parameters R and S must be converted to the LAMMPS parameters R and D (R is different in both forms), using the following relations: R=(R'+S')/2 and D=(S'-R')/2, where the primes indicate the Tersoff_2 parameters.

In the potentials directory, the file SiCGe.tersoff provides the LAMMPS parameters for Tersoff's various versions of Si, as well as his alloy parameters for Si, C, and Ge. This file can be used for pure Si, (three different versions), pure C, pure Ge, binary SiC, and binary SiGe. LAMMPS will generate an error if this file is used with any combination involving C and Ge, since there are no entries for the GeC interactions (Tersoff did not publish parameters for this cross-interaction.) Tersoff files are also provided for the SiC alloy (SiC.tersoff) and the GaN (GaN.tersoff) alloys.

Many thanks to Rutuparna Narulkar, David Farrell, and Xiaowang Zhou for helping clarify how Tersoff parameters for alloys have been defined in various papers. Also thanks to Ram Devanathan for providing the base ZBL implementation.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, where types I and J correspond to two different element types, mixing is performed by LAMMPS as described above from values in the potential file.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files, since it is stored in potential files. Thus, you need to re-specify the pair_style and pair_coeff commands in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This pair style is part of the MANYBODY package. It is only enabled if LAMMPS was built with that package (which it is by default). See the Making LAMMPS section for more info.

This pair style requires the newton setting to be "on" for pair interactions.

The Tersoff/ZBL potential files provided with LAMMPS (see the potentials directory) are parameterized for metal units. You can use the Tersoff potential with any LAMMPS units, but you would need to create your own Tersoff potential file with coefficients listed in the appropriate units if your simulation doesn't use "metal" units.

pair_style tersoff/zbl/omp command   954

Related commands:

pair_coeff

Default: none

(Tersoff_1) J. Tersoff, Phys Rev B, 37, 6991 (1988).

(ZBL) J.F. Ziegler, J.P. Biersack, U. Littmark, 'Stopping and Ranges of Ions in Matter' Vol 1, 1985, Pergamon Press.

(Albe) J. Nord, K. Albe, P. Erhartand K. Nordlund, J. Phys.: Condens. Matter, 15, 5649(2003).

(Tersoff_2) J. Tersoff, Phys Rev B, 39, 5566 (1989)

pair_style tri/lj command

pair_style tri/lj/omp command

Syntax:

```
pair_style tri/lj cutoff
```

cutoff = global cutoff for interactions (distance units)

Examples:

```
pair_style tri/lj 3.0
pair_coeff * * 1.0 1.0
pair_coeff 1 1 1.0 1.5 2.5
```

Description:

Style *tri/lj* treats particles which are triangles as a set of small spherical particles that tile the triangle surface as explained below. Interactions between two triangles, each with N1 and N2 spherical particles, are calculated as the pairwise sum of N1*N2 Lennard-Jones interactions. Interactions between a triangle with N spherical particles and a point particle are treated as the pairwise sum of N Lennard-Jones interactions. See the [pair_style lj/cut](#) doc page for the definition of Lennard-Jones interactions.

The cutoff distance for an interaction between 2 triangles, or between a triangle and a point particle, is calculated from the position of the triangle (its centroid), not between pairs of individual spheres comprising the triangle. Thus an interaction is either calculated in its entirety or not at all.

The set of non-overlapping spherical particles that represent a triangle, for purposes of this pair style, are generated in the following manner. Assume the triangle is of type I, and sigma_II has been specified. We want a set of spheres with centers in the plane of the triangle, none of them larger in diameter than sigma_II, which completely cover the triangle's area, but with minimial overlap and a minimal total number of spheres. This is done in a recursive manner. Place a sphere at the centroid of the original triangle. Calculate what diameter it must have to just cover all 3 corner points of the triangle. If that diameter is equal to or smaller than sigma_II, then include a sphere of the calculated diameter in the set of covering spheres. It the diameter is larger than sigma_II, then split the triangle into 2 triangles by bisecting its longest side. Repeat the process on each sub-triangle, recursing as far as needed to generate a set of covering spheres. When finished, the original criteria are met, and the set of covering spheres shoule be near minimal in number and overlap, at least for input triangles with a reasonable aspect-ratio.

The LJ interaction between 2 spheres on different triangles of types I,J is computed with an arithmetic mixing of the sigma values of the 2 spheres and using the specified epsilon value for I,J atom types. Note that because the sigma values for triangles spheres is computed using only sigma_II values, specific to the triangles's type, this means that any specified sigma_IJ values (for I != J) are effectively ignored.

For style *tri/lj*, the following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands:

- epsilon (energy units)
- sigma (distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the epsilon and sigma coefficients and cutoff distance for all of this pair style can be mixed. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style does not support the pair_modify shift, table, and tail options.

This pair style does not write its information to binary restart files.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the ASPHERE package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

Defining particles to be triangles so they participate in tri/tri or tri/particle interactions requires the use the atom_style tri command.

Related commands:

pair_coeff, pair_style line/lj

Default: none

pair_write command

Syntax:

```
pair_write itype jtype N style inner outer file keyword Qi Qj
```

- itype,jtype = 2 atom types
- N = # of values
- style = *r* or *rsq* or *bitmap*
- inner,outer = inner and outer cutoff (distance units)
- file = name of file to write values to
- keyword = section name in file for this set of tabulated values
- Qi,Qj = 2 atom charges (charge units) (optional)

Examples:

```
pair_write 1 3 500 r 1.0 10.0 table.txt LJ
pair_write 1 1 1000 rsq 2.0 8.0 table.txt Yukawa_1_1 -0.5 0.5
```

Description:

Write energy and force values to a file as a function of distance for the currently defined pair potential. This is useful for plotting the potential function or otherwise debugging its values. If the file already exists, the table of values is appended to the end of the file to allow multiple tables of energy and force to be included in one file.

The energy and force values are computed at distances from inner to outer for 2 interacting atoms of type itype and jtype, using the appropriate [pair_coeff](#) coefficients. If the style is *r*, then N distances are used, evenly spaced in r; if the style is *rsq*, N distances are used, evenly spaced in r^2.

For example, for N = 7, style = *r*, inner = 1.0, and outer = 4.0, values are computed at r = 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0.

If the style is *bitmap*, then 2^N values are written to the file in a format and order consistent with how they are read in by the [pair_coeff](#) command for pair style *table*. For reasonable accuracy in a bitmapped table, choose N >= 12, an *inner* value that is smaller than the distance of closest approach of 2 atoms, and an *outer* value <= cutoff of the potential.

If the pair potential is computed between charged atoms, the charges of the pair of interacting atoms can optionally be specified. If not specified, values of Qi = Qj = 1.0 are used.

The file is written in the format used as input for the [pair_style](#) *table* option with *keyword* as the section name. Each line written to the file lists an index number (1-N), a distance (in distance units), an energy (in energy units), and a force (in force units).

Restrictions:

All force field coefficients for pair and other kinds of interactions must be set before this command can be invoked.

Due to how the pairwise force is computed, an inner value > 0.0 must be specified even if the potential has a finite value at r = 0.0.

For EAM potentials, the pair_write command only tabulates the pairwise portion of the potential, not the embedding portion.

Related commands:

pair_style, pair_coeff

Default: none

pair_style yukawa/colloid command

pair_style yukawa/colloid/gpu command

pair_style yukawa/colloid/omp command

Syntax:

```
pair_style yukawa/colloid kappa cutoff
```

- kappa = screening length (inverse distance units)
- cutoff = global cutoff for colloidal Yukawa interactions (distance units)

Examples:

```
pair_style yukawa/colloid 2.0 2.5
pair_coeff 1 1 100.0 2.3
pair_coeff * * 100.0
```

Description:

Style *yukawa/colloid* computes pairwise interactions with the formula

$$E = \frac{A}{\kappa} e^{-\kappa(r-(r_i+r_j))} \qquad r < r_c$$

where Ri and Rj are the radii of the two particles and Rc is the cutoff.

In contrast to [pair_style yukawa](#), this functional form arises from the Coulombic interaction between two colloid particles, screened due to the presence of an electrolyte, see the book by [Safran](#) for a derivation in the context of DVLO theory. [Pair_style yukawa](#) is a screened Coulombic potential between two point-charges and uses no such approximation.

This potential applies to nearby particle pairs for which the Derjagin approximation holds, meaning h << Ri + Rj, where h is the surface-to-surface separation of the two particles.

When used in combination with [pair_style colloid](#), the two terms become the so-called DLVO potential, which combines electrostatic repulsion and van der Waals attraction.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- A (energy/distance units)
- cutoff (distance units)

The prefactor A is determined from the relationship between surface charge and surface potential due to the presence of electrolyte. Note that the A for this potential style has different units than the A used in [pair_style yukawa](#). For low surface potentials, i.e. less than about 25 mV, A can be written as:

```
A = 2 * PI * R*eps*eps0 * kappa * psi^2
```

where

- R = colloid radius (distance units)
- eps0 = permittivity of free space (charge^2/energy/distance units)
- eps = relative permittivity of fluid medium (dimensionless)
- kappa = inverse screening length (1/distance units)
- psi = surface potential (energy/charge units)

The last coefficient is optional. If not specified, the global yukawa/colloid cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in Section_accelerate of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the Making LAMMPS section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the -suffix command-line switch when you invoke LAMMPS, or you can use the suffix command in your input script.

See Section_accelerate of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the pair_modify shift option for the energy of the pair interaction.

The pair_modify table option is not relevant for this pair style.

This pair style does not support the pair_modify tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to binary restart files, so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the run_style respa command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions:

This style is part of the COLLOID package. It is only enabled if LAMMPS was built with that package. See the Making LAMMPS section for more info.

This pair style requires that atoms be finite-size spheres with a diameter, as defined by the atom_style sphere command.

pair_style yukawa/colloid/omp command   961

Per-particle polydispersity is not yet supported by this pair style; per-type polydispersity is allowed. This means all particles of the same type must have the same diameter. Each type can have a different diameter.

Related commands:

pair_coeff

Default: none

(Safran) Safran, Statistical Thermodynamics of Surfaces, Interfaces, And Membranes, Westview Press, ISBN: 978-0813340791 (2003).

pair_style yukawa command

pair_style yukawa/gpu command

pair_style yukawa/omp command

Syntax:

```
pair_style yukawa kappa cutoff
```

- kappa = screening length (inverse distance units)
- cutoff = global cutoff for Yukawa interactions (distance units)

Examples:

```
pair_style yukawa 2.0 2.5
pair_coeff 1 1 100.0 2.3
pair_coeff * * 100.0
```

Description:

Style *yukawa* computes pairwise interactions with the formula

$$E = A\frac{e^{-\kappa r}}{r} \qquad r < r_c$$

Rc is the cutoff.

The following coefficients must be defined for each pair of atoms types via the [pair_coeff](#) command as in the examples above, or in the data file or restart files read by the [read_data](#) or [read_restart](#) commands, or by mixing as described below:

- A (energy*distance units)
- cutoff (distance units)

The last coefficient is optional. If not specified, the global yukawa cutoff is used.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

For atom type pairs I,J and I != J, the A coefficient and cutoff distance for this pair style can be mixed. A is an energy value mixed like a LJ epsilon. The default mix value is *geometric*. See the "pair_modify" command for details.

This pair style supports the [pair_modify](#) shift option for the energy of the pair interaction.

The [pair_modify](#) table option is not relevant for this pair style.

This pair style does not support the [pair_modify](#) tail option for adding long-range tail corrections to energy and pressure.

This pair style writes its information to [binary restart files](#), so pair_style and pair_coeff commands do not need to be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

pair_style zbl command

pair_style zbl/omp command

Syntax:

```
pair_style zbl inner outer
```

- inner = distance where switching function begins
- outer = global cutoff for ZBL interaction

Examples:

```
pair_style zbl 3.0 4.0
pair_coeff * * 73.0
pair_coeff 1 1 14.0
```

Description:

Style *zbl* computes the Ziegler-Biersack-Littmark (ZBL) screened nuclear repulsion for describing high-energy collisions between atoms. [(Ziegler)](#). It includes an additional switching function that ramps the energy, force, and curvature smoothly to zero between an inner and outer cutoff. The potential energy due to a pair of atoms at a distance r_ij is given by:

$$
\begin{aligned}
E_{ij}^{ZBL} &= \frac{1}{4\pi\epsilon_0}\frac{Z_i Z_j\, e^2}{r_{ij}}\phi(r_{ij}/a) + S(r_{ij}) \\
a &= \frac{0.46850}{Z_i^{0.23} + Z_j^{0.23}} \\
\phi(x) &= 0.18175e^{-3.19980x} + 0.50986e^{-0.94229x} + 0.28022e^{-0.40290x} + 0.02817e^{-0.20162x}
\end{aligned}
$$

where e is the electron charge, epsilon_0 is the electrical permittivity of vacuum, and Z_i and Z_j are the nuclear charges of the two atoms in electron charge units. The switching function S(r) is identical to that used by [pair_style lj/gromacs](#). Here, the inner and outer cutoff are the same for all pairs of atom types.

The following coefficient must be defined for each pair of atom types via the [pair_coeff](#) command as in the examples above, or in the LAMMPS data file. Z can not be specified for two different atoms types. Therefore the lists of atom types I and atom types J must match.

- Z (electron charge)

Although Z must be defined for all atom type pairs I,J, it is only stored for individual atom types, i.e. when I = J. Z is normally equal to the atomic number of the atom type.

IMPORTANT NOTE: The numerical values of the exponential decay constants in the screening function depend on the unit of distance. In the above equation they are given for units of angstroms. LAMMPS will automatically convert these values to the distance unit of the specified LAMMPS [units](#) setting. The values of Z should always be given in units of electron charge.

Styles with a *cuda*, *gpu*, *omp*, or *opt* suffix are functionally the same as the corresponding style without the suffix. They have been optimized to run faster, depending on your available hardware, as discussed in [Section_accelerate](#) of the manual. The accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

These accelerated styles are part of the USER-CUDA, GPU, USER-OMP and OPT packages, respectively. They are only enabled if LAMMPS was built with those packages. See the [Making LAMMPS](#) section for more info.

You can specify the accelerated styles explicitly in your input script by including their suffix, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Mixing, shift, table, tail correction, restart, rRESPA info:

Mixing is not relevant for this pair style, since as explained above, Z values are stored on a per-type basis, and both Z_i and Z_j are used explicitly in the ZBL formula.

The ZBL pair style does not support the [pair_modify](#) shift option, since the ZBL interaction is already smoothed to 0.0 at the cutoff.

The [pair_modify](#) table option is not relevant for this pair style.

This pair style does not support the [pair_modify](#) tail option for adding long-range tail corrections to energy and pressure, since there are no corrections for a potential that goes to 0.0 at the cutoff.

This pair style does not write information to [binary restart files](#), so pair_style and pair_coeff commands must be specified in an input script that reads a restart file.

This pair style can only be used via the *pair* keyword of the [run_style respa](#) command. It does not support the *inner*, *middle*, *outer* keywords.

Restrictions: none

Related commands:

[pair_coeff](#)

Default: none

(Ziegler) J.F. Ziegler, J. P. Biersack and U. Littmark, "The Stopping and Range of Ions in Matter," Volume 1, Pergamon, 1985.

partition command

Syntax:

```
partition style N command ...
```

- style = *yes* or *no*
- N = partition number (see asterisk form below)
- command = any LAMMPS command

Examples:

```
partition yes 1 processors 4 10 6
partition no 5 print "Active partition"
partition yes *5 fix all nve
partition yes 6* fix all nvt temp 1.0 1.0 0.1
```

Description:

This command invokes the specified command on a subset of the partitions of processors you have defined via the -partition command-line switch. See [Section_start 6](#) for an explanation of the switch.

Normally, every input script command in your script is invoked by every partition. This behavior can be modified by defining world- or universe-style [variables](#) that have different values for each partition. This mechanism can be used to cause your script to jump to different input script files on different partitions, if such a variable is used in a [jump](#) command.

The "partition" command is another mechanism for having as input script operate differently on different partitions. It is basically a prefix on any LAMMPS command. The commmand will only be invoked on the partition(s) specified by the *style* and *N* arguments.

If the *style* is *yes*, the command will be invoked on any partition which matches the *N* argument. If the *style* is *no* the command will be invoked on all the partitions which do not match the Np argument.

Partitions are numbered from 1 to Np, where Np is the number of partitions specified by the [-partition command-line switch](#).

N can be specified in one of two ways. An explicit numeric value can be used, as in the 1st example above. Or a wild-card asterisk can be used to span a range of partition numbers. This takes the form "*" or "*n" or "n*" or "m*n". An asterisk with no numeric values means all partitions from 1 to Np. A leading asterisk means all partitions from 1 to n (inclusive). A trailing asterisk means all partitions from n to Np (inclusive). A middle asterisk means all partitions from m to n (inclusive).

This command can be useful for the "run_style verlet/split" command which imposed requirements on how the [processors](#) command lays out a 3d grid of processors in each of 2 partitions.

Restrictions: none

Related commands:

[run_style verlet/split](#)

Default: none

prd command

Syntax:

```
prd N t_event n_dephase t_dephase t_correlate compute-ID seed keyword value ...
```

- N = # of timesteps to run (not including dephasing/quenching)
- t_event = timestep interval between event checks
- n_dephase = number of velocity randomizations to perform in each dephase run
- t_dephase = number of timesteps to run dynamics after each velocity randomization during dephase
- t_correlate = number of timesteps within which 2 consecutive events are considered to be correlated
- compute-ID = ID of the compute used for event detection
- random_seed = random # seed (positive integer)
- zero or more keyword/value pairs may be appended
- keyword = *min* or *temp* or *vel*

```
  min values = etol ftol maxiter maxeval
     etol = stopping tolerance for energy, used in quenching
     ftol = stopping tolerance for force, used in quenching
     maxiter = max iterations of minimize, used in quenching
     maxeval = max number of force/energy evaluations, used in quenching
   temp value = Tdephase
     Tdephase = target temperature for velocity randomization, used in dephasing
   vel values = loop dist
     loop = all or local or geom, used in dephasing
     dist = uniform or gaussian, used in dephasing
```

Examples:

```
prd 5000 100 10 10 100 1 54982
prd 5000 100 10 10 100 1 54982 min 0.1 0.1 100 200
```

Description:

Run a parallel replica dynamics (PRD) simulation using multiple replicas of a system. One or more replicas can be used.

PRD is described in [this paper](#) by Art Voter. It is a method for performing accelerated dynamics that is suitable for infrequent-event systems that obey first-order kinetics. A good overview of accelerated dynamics methods for such systems in given in [this review paper](#) from the same group. To quote from the paper: "The dynamical evolution is characterized by vibrational excursions within a potential basin, punctuated by occasional transitions between basins." The transition probability is characterized by $p(t) = k*exp(-kt)$ where k is the rate constant. Running multiple replicas gives an effective enhancement in the timescale spanned by the multiple simulations, while waiting for an event to occur.

Each replica runs on a partition of one or more processors. Processor partitions are defined at run-time using the -partition command-line switch; see [Section_start 6](#) of the manual. Note that if you have MPI installed, you can run a multi-replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10-replica simulation on one or two processors. For PRD, this makes little sense, since this offers no effective parallel speed-up in searching for infrequent events. See [Section_howto 5](#) of the manual for further discussion.

When a PRD simulation is performed, it is assumed that each replica is running the same model, though LAMMPS does not check for this. I.e. the simulation domain, the number of atoms, the interaction potentials,

etc should be the same for every replica.

A PRD run has several stages, which are repeated each time an "event" occurs in one of the replicas, as defined below. The logic for a PRD run is as follows:

```
while (time remains):
  dephase for n_dephase*t_dephase steps
  until (event occurs on some replica):
    run dynamics for t_event steps
    quench
    check for uncorrelated event on any replica
  until (no correlated event occurs):
    run dynamics for t_correlate steps
    quench
    check for correlated event on this replica
  event replica shares state with all replicas
```

Before this loop begins, the state of the system on replica 0 is shared with all replicas, so that all replicas begin from the same initial state. The first potential energy basin is identified by quenching (an energy minimization, see below) the initial state and storing the resulting coordinates for reference.

In the first stage, dephasing is performed by each replica independently to eliminate correlations between replicas. This is done by choosing a random set of velocities, based on the *random_seed* that is specified, and running *t_dephase* timesteps of dynamics. This is repeated *n_dephase* times. If the *temp* keyword is not specified, the target temperature for velocity randomization for each replica is the current temperature of that replica. Otherwise, it is the specified *Tdephase* temperature. The style of velocity randomization is controlled using the keyword *vel* with arguments that have the same meaning as their counterparts in the [velocity](#) command.

In the second stage, each replica runs dynamics continuously, stopping every *t_event* steps to check if a transition event has occurred. This check is performed by quenching the system and comparing the resulting atom coordinates to the coordinates from the previous basin. The first time through the PRD loop, the "previous basin" is the set of quenched coordinates from the initial state of the system.

A quench is an energy minimization and is performed by whichever algorithm has been defined by the [min_style](#) command. Minimization parameters may be set via the [min_modify](#) command and by the *min* keyword of the PRD command. The latter are the settings that would be used with the [minimize](#) command. Note that typically, you do not need to perform a highly-converged minimization to detect a transition event.

The event check is performed by a compute with the specified *compute-ID*. Currently there is only one compute that works with the PRD commmand, which is the [compute event/displace](#) command. Other event-checking computes may be added. [Compute event/displace](#) checks whether any atom in the compute group has moved further than a specified threshold distance. If so, an "event" has occurred.

In the third stage, the replica on which the event occurred (event replica) continues to run dynamics to search for correlated events. This is done by running dynamics for *t_correlate* steps, quenching every *t_event* steps, and checking if another event has occurred. The first time no correlated event occurs, the final state of the event replica is shared with all replicas, the new basin reference coordinates are updated with the quenched state, and the outer loop begins again. While the replica event is searching for correlated events, all the other replicas also run dynamics and event checking with the same schedule, but the final states are always overwritten by the state of the event replica.

Four kinds of output can be generated during a PRD run: event statistics, thermodynamic output by each replica, dump files, and restart files.

When running with multiple partitions (each of which is a replica in this case), the print-out to the screen and

master log.lammps file is limited to event statistics. Note that if a PRD run is performed on only a single replica then the event statistics will be intermixed with the usual thermodynamic output discussed below.

The quantities printed each time an event occurs are the timestep, CPU time, clock, event number, a correlation flag, the number of coincident events, and the replica number of the chosen event.

The timestep is the usual LAMMPS timestep, except that time does not advance during dephasing or quenches, but only during dynamics. Note that are two kinds of dynamics in the PRD loop listed above. The first is when all replicas are performing independent dynamics. The second is when correlated events are being searched for and only one replica is running dynamics.

The CPU time is the total processor time since the start of the PRD run.

The clock is the same as the timestep except that it advances by M steps every timestep during the first kind of dynamics when the M replicas are running independently. The clock represents the real time that effectively elapses during a PRD simulation of N steps on M replicas. If most of the PRD run is spent in the second stage of the loop above, searching for infrequent events, then the clock will advance nearly N*M steps. Note the clock time between events will be drawn from p(t).

The event number is a counter that increments with each event, whether it is uncorrelated or correlated.

The correlation flag will be 0 when an uncorrelated event occurs during the second stage of the loop listed above, i.e. when all replicas are running independently. The correlation flag will be 1 when a correlated event occurs during the third stage of the loop listed above, i.e. when only one replica is running dynamics.

When more than one replica detects an event at the end of the second stage, then one of them is chosen at random. The number of coincident events is the number of replicas that detected an event. Normally, we expect this value to be 1. If it is often greater than 1, then either the number of replicas is too large, or *t_event* is too large.

The replica number is the ID of the replica (from 0 to M-1) that found the event.

When running on multiple partitions, LAMMPS produces additional log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For the PRD command, these contain the thermodynamic output for each replica. You will see short runs and minimizations corresponding to the dynamics and quench operations of the loop listed above. The timestep will be reset aprropriately depending on whether the operation advances time or not.

After the PRD command completes, timing statistics for the PRD run are printed in each replica's log file, giving a breakdown of how much CPU time was spent in each stage (dephasing, dynamics, quenching, etc).

Any dump files defined in the input script, will be written to during a PRD run at timesteps corresponding to both uncorrelated and correlated events. This means the the requested dump frequency in the dump command is ignored. There will be one dump file (per dump command) created for all partitions.

The atom coordinates of the dump snapshot are those of the minimum energy configuration resulting from quenching following a transition event. The timesteps written into the dump files correspond to the timestep at which the event occurred and NOT the clock. A dump snapshot corresponding to the initial minimum state used for event detection is written to the dump file at the beginning of each PRD run.

If the restart command is used, a single restart file for all the partitions is generated, which allows a PRD run to be continued by a new input script in the usual manner.

prd command

The restart file is generated at the end of the loop listed above. If no correlated events are found, this means it contains a snapshot of the system at time T + *t_correlate*, where T is the time at which the uncorrelated event occurred. If correlated events were found, then it contains a snapshot of the system at time T + *t_correlate*, where T is the time of the last correlated event.

The restart frequency specified in the restart command is interpreted differently when performing a PRD run. It does not mean the timestep interval between restart files. Instead it means an event interval for uncorrelated events. Thus a frequency of 1 means write a restart file every time an uncorrelated event occurs. A frequency of 10 means write a restart file every 10th uncorrelated event.

When an input script reads a restart file from a previous PRD run, the new script can be run on a different number of replicas or processors. However, it is assumed that *t_correlate* in the new PRD command is the same as it was previously. If not, the calculation of the "clock" value for the first event in the new run will be slightly off.

Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the Making LAMMPS section for more info on packages.

N and *t_correlate* settings must be integer multiples of *t_event*.

Runs restarted from restart file written during a PRD run will not produce identical results due to changes in the random numbers used for dephasing.

This command cannot be used when any fixes are defined that keep track of elapsed time to perform time-dependent operations. Examples include the "ave" fixes such as fix ave/spatial. Also fix dt/reset and fix deposit.

Related commands:

compute event/displace, min_modify, min_style, run_style, minimize, velocity, temper, [neb](neb), [tad](tad)

Default:

The option defaults are *min* = 0.1 0.1 40 50, no *temp* setting, and *vel = geom gaussian*.

(Voter) Voter, Phys Rev B, 57, 13985 (1998).

(Voter2) Voter, Montalenti, Germann, Annual Review of Materials Research 32, 321 (2002).

print command

Syntax:

print string keyword value:pre

- string = text string to print, which may contain variables
- zero or more keyword/value pairs may be appended
- keyword = *file* or *append* or *screen*

```
  file value = filename
    append value = filename
    screen value = yes or no
```

Examples:

```
print "Done with equilibration" file info.dat
print Vol=$v append info.dat screen no
print "The system volume is now $v"
print 'The system volume is now $v'
```

Description:

Print a text string to the screen and logfile. One line of output is generated. The text string must be a single argument, so it should be enclosed in double quotes if it is more than one word. If it contains variables, they will be evaluated and their current values printed.

If the *file* or *append* keyword is used, a filename is specified to which the output will be written. If *file* is used, then the filename is overwritten if it already exists. If *append* is used, then the filename is appended to if it already exists, or created if it does not exist.

If the *screen* keyword is used, output to the screen and logfile can be turned on or off as desired.

If you want the print command to be executed multiple times (with changing variable values), there are 3 options. First, consider using the [fix print](#) command, which will print a string periodically during a simulation. Second, the print command can be used as an argument to the *every* option of the [run](#) command. Third, the print command could appear in a section of the input script that is looped over (see the [jump](#) and [next](#) commands).

See the [variable](#) command for a description of *equal* style variables which are typically the most useful ones to use with the print command. Equal-style variables can calculate formulas involving mathematical operations, atom properties, group properties, thermodynamic properties, global values calculated by a [compute](#) or [fix](#), or references to other [variables](#).

Restrictions: none

Related commands:

[fix print](#), [variable](#)

Default:

The option defaults are no file output and screen = yes.

processors command

Syntax:

```
processors Px Py Pz keyword args ...
```

- Px,Py,Pz = # of processors in each dimension of 3d grid overlaying the simulation domain
- zero or more keyword/arg pairs may be appended
- keyword = *grid* or *map* or *part* or *file*

```
  grid arg = gstyle params ...
      gstyle = onelevel or twolevel or numa or custom
        onelevel params = none
        twolevel params = Nc Cx Cy Cz
          Nc = number of cores per node
          Cx,Cy,Cz = # of cores in each dimension of 3d sub-grid assigned to each node
        numa params = none
        custom params = infile
          infile = file containing grid layout
    map arg = cart or cart/reorder or xyz or xzy or yxz or yzx or zxy or zyx
        cart = use MPI_Cart() methods to map processors to 3d grid with reorder = 0
        cart/reorder = use MPI_Cart() methods to map processors to 3d grid with reorder = 1
        xyz,xzy,yxz,yzx,zxy,zyx = map procesors to 3d grid in IJK ordering
    numa arg = none
    part args = Psend Precv cstyle
      Psend = partition # (1 to Np) which will send its processor layout
      Precv = partition # (1 to Np) which will recv the processor layout
      cstyle = multiple
        multiple = Psend grid will be multiple of Precv grid in each dimension
    file arg = outfile
      outfile = name of file to write 3d grid of processors to
```

Examples:

```
processors * * 5
processors 2 4 4
processors * * 8 map xyz
processors * * * grid numa
processors * * * grid twolevel 4 * * 1
processors 4 8 16 grid custom myfile
processors * * * part 1 2 multiple
```

Description:

Specify how processors are mapped as a 3d logical grid to the global simulation box. This involves 2 steps. First if there are P processors it means choosing a factorization P = Px by Py by Pz so that there are Px processors in the x dimension, and similarly for the y and z dimensions. Second, the P processors are mapped to the logical 3d grid. The arguments to this command control each of these 2 steps.

The Px, Py, Pz parameters affect the factorization. Any of the 3 parameters can be specified with an asterisk "*", which means LAMMPS will choose the number of processors in that dimension of the grid. It will do this based on the size and shape of the global simulation box so as to minimize the surface-to-volume ratio of each processor's sub-domain.

Since LAMMPS does not load-balance by changing the grid of 3d processors on-the-fly, choosing explicit values for Px or Py or Pz can be used to override the LAMMPS default if it is known to be sub-optimal for a particular problem. E.g. a problem where the extent of atoms will change dramatically in a particular

dimension over the course of the simulation.

The product of Px, Py, Pz must equal P, the total # of processors LAMMPS is running on. For a [2d simulation](#), Pz must equal 1.

Note that if you run on a prime number of processors P, then a grid such as 1 x P x 1 will be required, which may incur extra communication costs due to the high surface area of each processor's sub-domain.

Also note that if multiple partitions are being used then P is the number of processors in this partition; see [this section](#) for an explanation of the -partition command-line switch. Also note that you can prefix the processors command with the [partition](#) command to easily specify different Px,Py,Pz values for different partitions.

You can use the [partition](#) command to specify different processor grids for different partitions, e.g.

```
partition yes 1 processors 4 4 4
partition yes 2 processors 2 3 2
```

The *grid* keyword affects the factorization of P into Px,Py,Pz and it can also affect how the P processor IDs are mapped to the 3d grid of processors.

The *onelevel* style creates a 3d grid that is compatible with the Px,Py,Pz settings, and which minimizes the surface-to-volume ratio of each processor's sub-domain, as described above. The mapping of processors to the grid is determined by the *map* keyword setting.

The *twolevel* style can be used on machines with multicore nodes to minimize off-node communication. It insures that contiguous sub-sections of the 3d grid are assigned to all the cores of a node. For example if *Nc* is 4, then 2x2x1 or 2x1x2 or 1x2x2 sub-sections of the 3d grid will correspond to the cores of each node. This affects both the factorization and mapping steps.

The *Cx*, *Cy*, *Cz* settings are similar to the *Px*, *Py*, *Pz* settings, only their product should equal *Nc*. Any of the 3 parameters can be specified with an asterisk "*", which means LAMMPS will choose the number of cores in that dimension of the node's sub-grid. As with Px,Py,Pz, it will do this based on the size and shape of the global simulation box so as to minimize the surface-to-volume ratio of each processor's sub-domain.

IMPORTANT NOTE: For the *twolevel* style to work correctly, it assumes the MPI ranks of processors LAMMPS is running on are ordered by core and then by node. E.g. if you are running on 2 quad-core nodes, for a total of 8 processors, then it assumes processors 0,1,2,3 are on node 1, and processors 4,5,6,7 are on node 2. This is the default rank ordering for most MPI implementations, but some MPIs provide options for this ordering, e.g. via environment variable settings.

The *numa* style operates similar to the *twolevel* keyword except that it auto-detects which cores are running on which nodes. Currently, it does this in only 2 levels, but it may be extended in the future to account for socket topology and other non-uniform memory access (NUMA) costs. It also uses a different algorithm than the *twolevel* keyword for doing the two-level factorization of the simulation box into a 3d processor grid to minimize off-node communication, and it does its own MPI-based mapping of nodes and cores to the logical 3d grid. Thus it may produce a different layout of the processors than the *twolevel* options.

The *numa* style will give an error if the number of MPI processes is not divisible by the number of cores used per node, or any of the Px or Py of Pz values is greater than 1.

IMPORTANT NOTE: Unlike the *twolevel* style, the *numa* style does not require any particular ordering of MPI ranks i norder to work correctly. This is because it auto-detects which processes are running on which nodes.

The *custom* style uses the file *infile* to define both the 3d factorization and the mapping of processors to the grid.

The file should have the following format. Any number of initial blank or comment lines (starting with a "#" character) can be present. The first non-blank, non-comment line should have 3 values:

```
Px Py Py
```

These must be compatible with the total number of processors and the Px, Py, Pz settings of the processors commmand.

This line should be immediately followed by P = Px*Py*Pz lines of the form:

```
ID I J K
```

where ID is a processor ID (from 0 to P-1) and I,J,K are the processors location in the 3d grid. I must be a number from 1 to Px (inclusive) and similarly for J and K. The P lines can be listed in any order, but no processor ID should appear more than once.

The *map* keyword affects how the P processor IDs (from 0 to P-1) are mapped to the 3d grid of processors. It is only used by the *onelevel* and *twolevel* grid settings.

The *cart* style uses the family of MPI Cartesian functions to perform the mapping, namely MPI_Cart_create(), MPI_Cart_get(), MPI_Cart_shift(), and MPI_Cart_rank(). It invokes the MPI_Cart_create() function with its reorder flag = 0, so that MPI is not free to reorder the processors.

The *cart/reorder* style does the same thing as the *cart* style except it sets the reorder flag to 1, so that MPI can reorder processors if it desires.

The *xyz*, *xzy*, *yxz*, *yzx*, *zxy*, and *zyx* styles are all similar. If the style is IJK, then it maps the P processors to the grid so that the processor ID in the I direction varies fastest, the processor ID in the J direction varies next fastest, and the processor ID in the K direction varies slowest. For example, if you select style *xyz* and you have a 2x2x2 grid of 8 processors, the assignments of the 8 octants of the simulation domain will be:

```
proc 0 = lo x, lo y, lo z octant
proc 1 = hi x, lo y, lo z octant
proc 2 = lo x, hi y, lo z octant
proc 3 = hi x, hi y, lo z octant
proc 4 = lo x, lo y, hi z octant
proc 5 = hi x, lo y, hi z octant
proc 6 = lo x, hi y, hi z octant
proc 7 = hi x, hi y, hi z octant
```

Note that, in principle, an MPI implementation on a particular machine should be aware of both the machine's network topology and the specific subset of processors and nodes that were assigned to your simulation. Thus its MPI_Cart calls can optimize the assignment of MPI processes to the 3d grid to minimize communication costs. In practice, however, few if any MPI implementations actually do this. So it is likely that the *cart* and *cart/reorder* styles simply give the same result as one of the IJK styles.

Also note, that for the *twolevel* grid style, the *map* setting is used to first map the nodes to the 3d grid, then again to the cores within each node. For the latter step, the *cart* and *cart/reorder* styles are not supported, so an *xyz* style is used in their place.

The *part* keyword affects the factorization of P into Px,Py,Pz.

It can be useful when running in multi-partition mode, e.g. with the <u>run_style verlet/split</u> command. It specifies a dependency bewteen a sending partition *Psend* and a receiving partition *Precv* which is enforced when each is setting up their own mapping of their processors to the simulation box. Each of *Psend* and *Precv* must be integers from 1 to Np, where Np is the number of partitions you have defined via the <u>-partition command-line switch</u>.

A "dependency" means that the sending partition will create its 3d logical grid as Px by Py by Pz and after it has done this, it will send the Px,Py,Pz values to the receiving partition. The receiving partition will wait to receive these values before creating its own 3d logical grid and will use the sender's Px,Py,Pz values as a constraint. The nature of the constraint is determined by the *cstyle* argument.

For a *cstyle* of *multiple*, each dimension of the sender's processor grid is required to be an integer multiple of the corresponding dimension in the receiver's processor grid. This is a requirement of the <u>run_style verlet/split</u> command.

For example, assume the sending partition creates a 4x6x10 grid = 240 processor grid. If the receiving partition is running on 80 processors, it could create a 4x2x10 grid, but it will not create a 2x4x10 grid, since in the y-dimension, 6 is not an integer multiple of 4.

IMPORTANT NOTE: If you use the <u>partition</u> command to invoke different "processsors" commands on different partitions, and you also use the *part* keyword, then you must insure that both the sending and receiving partitions invoke the "processors" command that connects the 2 partitions via the *part* keyword. LAMMPS cannot easily check for this, but your simulation will likely hang in its setup phase if this error has been made.

The *file* keyword writes the mapping of the factorization of P processors and their mapping to the 3d grid to the specified file *outfile*. This is useful to check that you assigned physical processors in the manner you desired, which can be tricky to figure out, especially when running on multiple partitions or on, a multicore machine or when the processor ranks were reordered by use of the <u>-reorder command-line switch</u> or due to use of MPI-specific launch options such as a config file.

If you have multiple partitions you should insure that each one writes to a different file, e.g. using a <u>world-style variable</u> for the filename. The file has a self-explanatory header, followed by one-line per processor in this format:

world-ID universe-ID original-ID: I J K: name

The IDs are the processor's rank in this simulation (the world), the universe (of multiple simulations), and the original MPI communicator used to instantiate LAMMPS, respectively. The world and universe IDs will only be different if you are running on more than one partition; see the <u>-partition command-line switch</u>. The universe and original IDs will only be different if you used the <u>-reorder command-line switch</u> to reorder the processors differently than their rank in the original communicator LAMMPS was instantiated with.

I,J,K are the indices of the processor in the 3d logical grid, each from 1 to Nd, where Nd is the number of processors in that dimension of the grid.

The *name* is what is returned by a call to MPI_Get_processor_name() and should represent an identifier relevant to the physical processors in your machine. Note that depending on the MPI implementation, multiple cores can have the same *name*.

Restrictions:

This command cannot be used after the simulation box is defined by a <u>read_data</u> or <u>create_box</u> command. It can be used before a restart file is read to change the 3d processor grid from what is specified in the restart

file.

The *grid numa* keyword only currently works with the *map cart* option.

The *part* keyword (for the receiving partition) only works with the *grid onelevel* or *grid twolevel* options.

Related commands:

partition, -reorder command-line switch

Default:

The option defaults are Px Py Pz = * * *, grid = onelevel, and map = cart.

quit command

Syntax:

```
quit
```

Examples:

```
quit
if "$n > 10000" then quit
```

Description:

This command causes LAMMPS to exit, after shutting down all output cleanly.

It can be used as a debug statement in an input script, to terminate the script at some intermediate point.

It can also be used as an invoked command inside the "then" or "else" portion of an [if](#) command.

Restrictions: none

Related commands:

[if](#)

Default: none

read_data command

Syntax:

```
read_data file keyword args ...
```

- file = name of data file to read in
- zero or more keyword/arg pairs may be appended
- keyword = *fix*

```
  fix args = fix-ID header-string section-string
      fix-ID = ID of fix to process header lines and sections of data file
      header-string = header lines containing this string will be passed to fix
      section-string = section names with this string will be passed to fix
```

Examples:

```
read_data data.lj
read_data ../run7/data.polymer.gz
read_data data.protein fix mycmap crossterm CMAP
```

Description:

Read in a data file containing information LAMMPS needs to run a simulation. The file can be ASCII text or a gzipped text file (detected by a .gz suffix). This is one of 3 ways to specify initial atom coordinates; see the [read_restart](#) and [create_atoms](#) commands for alternative methods.

The structure of the data file is important, though many settings and sections are optional or can come in any order. See the examples directory for sample data files for different problems.

A data file has a header and a body. The header appears first. The first line of the header is always skipped; it typically contains a description of the file. Then lines are read one at a time. Lines can have a trailing comment starting with '#' that is ignored. If the line is blank (only whitespace after comment is deleted), it is skipped. If the line contains a header keyword, the corresponding value(s) is read from the line. If it doesn't contain a header keyword, the line begins the body of the file.

The body of the file contains zero or more sections. The first line of a section has only a keyword. The next line is skipped. The remaining lines of the section contain values. The number of lines depends on the section keyword as described below. Zero or more blank lines can be used between sections. Sections can appear in any order, with a few exceptions as noted below.

The keyword *fix* can be used one or more times. Each usage specifies a fix that will be used to process a specific portion of the data file. Any header line containing *header-string* and any section with a name containing *section-string* will be passed to the specified fix. See the [fix property/atom](#) command for an example of a fix that operates in this manner. The doc page for the fix defines the syntax of the header line(s) and section(s) that it reads from the data file. Note that the *header-string* can be specified as NULL, in which case no header lines are passed to the fix. This means that it can infer the length of its Section from standard header settings, such as the number of atoms.

The formatting of individual lines in the data file (indentation, spacing between words and numbers) is not important except that header and section keywords (e.g. atoms, xlo xhi, Masses, Bond Coeffs) must be capitalized as shown and can't have extra white space between their words - e.g. two spaces or a tab between the 2 words in "xlo xhi" or the 2 words in "Bond Coeffs", is not valid.

These are the recognized header keywords. Header lines can come in any order. The value(s) are read from the beginning of the line. Thus the keyword *atoms* should be in a line like "1000 atoms"; the keyword *ylo yhi* should be in a line like "-10.0 10.0 ylo yhi"; the keyword *xy xz yz* should be in a line like "0.0 5.0 6.0 xy xz yz". All these settings have a default value of 0, except the lo/hi box size defaults are -0.5 and 0.5. A line need only appear if the value is different than the default.

- *atoms* = # of atoms in system
- *bonds* = # of bonds in system
- *angles* = # of angles in system
- *dihedrals* = # of dihedrals in system
- *impropers* = # of impropers in system
- *atom types* = # of atom types in system
- *bond types* = # of bond types in system
- *angle types* = # of angle types in system
- *dihedral types* = # of dihedral types in system
- *improper types* = # of improper types in system
- *extra bond per atom* = leave space for this many new bonds per atom
- *ellipsoids* = # of ellipsoids in system
- *lines* = # of line segments in system
- *triangles* = # of triangles in system
- *bodies* = # of bodies in system
- *xlo xhi* = simulation box boundaries in x dimension
- *ylo yhi* = simulation box boundaries in y dimension
- *zlo zhi* = simulation box boundaries in z dimension
- *xy xz yz* = simulation box tilt factors for triclinic system

The initial simulation box size is determined by the lo/hi settings. In any dimension, the system may be periodic or non-periodic; see the boundary command.

If the *xy xz yz* line does not appear, LAMMPS will set up an axis-aligned (orthogonal) simulation box. If the line does appear, LAMMPS creates a non-orthogonal simulation domain shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

The tilt factors (xy,xz,yz) can not skew the box more than half the distance of the corresponding parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

See Section howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

When a triclinic system is used, the simulation domain must be periodic in any dimensions with a non-zero tilt factor, as defined by the boundary command. I.e. if the xy tilt factor is non-zero, then both the x and y dimensions must be periodic. Similarly, x and z must be periodic if xz is non-zero and y and z must be periodic if yz is non-zero. Also note that if your simulation will tilt the box, e.g. via the fix deform command, the simulation box must be defined as triclinic, even if the tilt factors are initially 0.0.

For 2d simulations, the *zlo zhi* values should be set to bound the z coords for atoms that appear in the file; the default of -0.5 0.5 is valid if all z coords are 0.0. For 2d triclinic simulations, the xz and yz tilt factors must be

0.0.

If the system is periodic (in a dimension), then atom coordinates can be outside the bounds (in that dimension); they will be remapped (in a periodic sense) back inside the box.

IMPORTANT NOTE: If the system is non-periodic (in a dimension), then all atoms in the data file must have coordinates (in that dimension) that are "greater than or equal to" the lo value and "less than or equal to" the hi value. If the non-periodic dimension is of style "fixed" (see the boundary command), then the atom coords must be strictly "less than" the hi value, due to the way LAMMPS assign atoms to processors. Note that you should not make the lo/hi values radically smaller/larger than the extent of the atoms. For example, if your atoms extend from 0 to 50, you should not specify the box bounds as -10000 and 10000. This is because LAMMPS uses the specified box size to layout the 3d grid of processors. A huge (mostly empty) box will be sub-optimal for performance when using "fixed" boundary conditions (see the boundary command). When using "shrink-wrap" boundary conditions (see the boundary command), a huge (mostly empty) box may cause a parallel simulation to lose atoms the first time that LAMMPS shrink-wraps the box around the atoms.

The "extra bond per atom" setting should be used if new bonds will be added to the system when a simulation runs, e.g. by using the fix bond/create command. This will pre-allocate space in LAMMPS data structures for storing the new bonds.

The "ellipsoids" and "lines" and "triangles" and "bodies" settings are only used with atom_style ellipsoid or line or tri or body and specify how many of the atoms are finite-size ellipsoids or lines or triangles or bodies; the remainder are point particles. See the discussion of ellipsoidflag and the *Ellipsoids* section below. See the discussion of lineflag and the *Lines* section below. See the discussion of triangleflag and the *Triangles* section below. See the discussion of bodyflag and the *Bodies* section below.

These are the section keywords for the body of the file.

- *Atoms, Velocities, Masses, Ellipsoids, Lines, Triangles, Bodies* = atom-property sections
- *Bonds, Angles, Dihedrals, Impropers* = molecular topology sections
- *Pair Coeffs, PairIJ Coeffs, Bond Coeffs, Angle Coeffs, Dihedral Coeffs, Improper Coeffs* = force field sections
- *BondBond Coeffs, BondAngle Coeffs, MiddleBondTorsion Coeffs, EndBondTorsion Coeffs, AngleTorsion Coeffs, AngleAngleTorsion Coeffs, BondBond13 Coeffs, AngleAngle Coeffs* = class 2 force field sections

Each section is listed below in alphabetic order. The format of each section is described including the number of lines it must contain and rules (if any) for where it can appear in the data file.

Any individual line in the various sections can have a trailing comment starting with "#" for annotation purposes. E.g. in the Atoms section:

```
10 1 17 -1.0 10.0 5.0 6.0   # salt ion
```

Angle Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
    ID = angle type (1-N)
    coeffs = list of coeffs
```
- example:

```
    6 70 108.5 0 0
```

The number and meaning of the coefficients are specific to the defined angle style. See the <u>angle_style</u> and <u>angle_coeff</u> commands for details. Coefficients can also be set via the <u>angle_coeff</u> command in the input script.

AngleAngle Coeffs section:

- one line per improper type
- line syntax: ID coeffs

```
ID = improper type (1-N)
coeffs = list of coeffs (see improper_coeff)
```

AngleAngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see dihedral_coeff)
```

Angles section:

- one line per angle
- line syntax: ID type atom1 atom2 atom3

```
ID = number of angle (1-Nangles)
type = angle type (1-Nangletype)
atom1,atom2,atom3 = IDs of 1st,2nd,3rd atoms in angle
```

   example:

```
2 2 17 29 430
```

The 3 atoms are ordered linearly within the angle. Thus the central atom (around which the angle is computed) is the atom2 in the list. E.g. H,O,H for a water molecule. The *Angles* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

AngleTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see dihedral_coeff)
```

Atoms section:

- one line per atom
- line syntax: depends on atom style

An *Atoms* section must appear in the data file if natoms > 0 in the header section. The atoms can be listed in any order. These are the line formats for each <u>atom style</u> in LAMMPS. As discussed below, each line can optionally have 3 flags (nx,ny,nz) appended to it, which indicate which image of a periodic simulation box the atom is in. These may be important to include for some kinds of analysis.

---

angle	atom-ID molecule-ID atom-type x y z
atomic	atom-ID atom-type x y z
body	atom-ID atom-type bodyflag mass x y z
bond	atom-ID molecule-ID atom-type x y z
charge	atom-ID atom-type q x y z
dipole	atom-ID atom-type q x y z mux muy muz
electron	atom-ID atom-type q spin eradius x y z
ellipsoid	atom-ID atom-type ellipsoidflag density x y z
full	atom-ID molecule-ID atom-type q x y z
line	atom-ID molecule-ID atom-type lineflag density x y z
meso	atom-ID atom-type rho e cv x y z
molecular	atom-ID molecule-ID atom-type x y z
peri	atom-ID atom-type volume density x y z
sphere	atom-ID atom-type diameter density x y z
tri	atom-ID molecule-ID atom-type triangleflag density x y z
wavepacket	atom-ID atom-type charge spin eradius etag cs_re cs_im x y z
hybrid	atom-ID atom-type x y z sub-style1 sub-style2 ...

The keywords have these meanings:

- atom-ID = integer ID of atom
- molecule-ID = integer ID of molecule the atom belongs to
- atom-type = type of atom (1-Ntype)
- q = charge on atom (charge units)
- diameter = diameter of spherical atom (distance units)
- ellipsoidflag = 1 for ellipsoidal particles, 0 for point particles
- lineflag = 1 for line segment particles, 0 for point particles
- triangleflag = 1 for triangular particles, 0 for point particles
- bodyflag = 1 for body particles, 0 for point particles
- density = density of particle (mass/distance^3 or mass/distance^2 or mass/distance units, depending on dimensionality of particle)
- mass = mass of particle (mass units)
- volume = volume of particle (distance^3 units)
- x,y,z = coordinates of atom
- mux,muy,muz = components of dipole moment of atom (dipole units)
- rho = density (need units) for SPH particles
- e = energy (need units) for SPH particles
- cv = heat capacity (need units) for SPH particles
- spin = electron spin (+1/-1), 0 = nuclei, 2 = fixed-core, 3 = pseudo-cores (i.e. ECP)
- eradius = electron radius (or fixed-core radius)
- etag = integer ID of electron that each wavepacket belongs to
- cs_re,cs_im = real/imaginary parts of wavepacket coefficients

The units for these quantities depend on the unit style; see the units command for details.

For 2d simulations specify z as 0.0, or a value within the *zlo zhi* setting in the data file header.

The atom-ID is used to identify the atom throughout the simulation and in dump files. Normally, it is a unique value from 1 to Natoms for each atom. Unique values larger than Natoms can be used, but they will cause extra memory to be allocated on each processor, if an atom map array is used (see the atom_modify command). If an atom map array is not used (e.g. an atomic system with no bonds), and velocities are not assigned in the data file, and you don't care if unique atom IDs appear in dump files, then the atom-IDs can all

be set to 0.

The molecule ID is a 2nd identifier attached to an atom. Normally, it is a number from 1 to N, identifying which molecule the atom belongs to. It can be 0 if it is an unbonded atom or if you don't care to keep track of molecule assignments.

The diameter specifies the size of a finite-size spherical particle. It can be set to 0.0, which means that atom is a point particle.

The ellipsoidflag, lineflag, triangleflag, and bodyflag determine whether the particle is a finite-size ellipsoid or line or triangle or body of finite size, or whether the particle is a point particle. Additional attributes must be defined for each ellipsoid, line, triangle, or body in the corresponding *Ellipsoids*, *Lines*, *Triangles*, or *Bodies* section.

Some pair styles and fixes and computes that operate on finite-size particles allow for a mixture of finite-size and point particles. See the doc pages of individual commands for details.

For finite-size particles, the density is used in conjunction with the particle volume to set the mass of each particle as mass = density * volume. In this context, volume can be a 3d quantity (for spheres or ellipsoids), a 2d quantity (for triangles), or a 1d quantity (for line segments). If the volume is 0.0, meaning a point particle, then the density value is used as the mass. One exception is for the body atom style, in which case the mass of each particle (body or point particle) is specified explicitly. This is because the volume of the body is unknown.

For atom_style hybrid, following the 5 initial values (ID,type,x,y,z), specific values for each sub-style must be listed. The order of the sub-styles is the same as they were listed in the <u>atom_style</u> command. The sub-style specific values are those that are not the 5 standard ones (ID,type,x,y,z). For example, for the "charge" sub-style, a "q" value would appear. For the "full" sub-style, a "molecule-ID" and "q" would appear. These are listed in the same order they appear as listed above. Thus if

```
atom_style hybrid charge sphere
```

were used in the input script, each atom line would have these fields:

```
atom-ID atom-type x y z q diameter density
```

Note that if a non-standard value is defined by multiple sub-styles, it must appear mutliple times in the atom line. E.g. the atom line for atom_style hybrid dipole full would list "q" twice:

```
atom-ID atom-type x y z q mux muy myz molecule-ID q
```

Atom lines (all lines or none of them) can optionally list 3 trailing integer values: nx,ny,nz. For periodic dimensions, they specify which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the <u>dump</u> command.

If nx,ny,nz values are not set in the data file, LAMMPS initializes them to 0. If image information is needed for later analysis and they are not all initially 0, it's important to set them correctly in the data file. Also, if you plan to use the <u>replicate</u> command to generate a larger system, these flags must be listed correctly for bonded atoms when the bond crosses a periodic boundary. I.e. the values of the image flags should be different by 1 (in the appropriate dimension) for the two atoms in such a bond.

Atom velocities and other atom quantities not defined above are set to 0.0 when the *Atoms* section is read. Velocities can be set later by a *Velocities* section in the data file or by a <u>velocity</u> or <u>set</u> command in the input

script.

Bodies section:

- one or more lines per body
- first line syntax: atom-ID ninteger ndouble

```
ninteger = # of integer quantities for this particle
ndouble = # of floating-point quantities for this particle
```
- 0 or more integer lines: one line for every 10 integer quantities
- 0 or more double lines: one line for every 10 double quantities
- example:

```
12 3 6
2 3 2
1.0 2.0 3.0 1.0 2.0 4.0
```
- example:

```
12 0 14
1.0 2.0 3.0 1.0 2.0 4.0 1.0 2.0 3.0 1.0
2.0 4.0 4.0 2.0
```

The *Bodies* section must appear if [atom_style body](#) is used and any atoms listed in the *Atoms* section have a bodyflag = 1. The number of bodies should be specified in the header section via the "bodies" keyword.

Each body can have a variable number of integer and/or floating-point values. The number and meaning of the values is defined by the body style, as described in the [body](#) doc page. The body style is given as an argument to the [atom_style body](#) command.

The ninteger and ndouble values determine how many integer and floating-point values are specified for this particle. Ninteger and ndouble can be as large as needed and can be different for every body. Integer values are then listed on subsequent lines, 10 values per line. Floating-point values follow on subsequent lines, again 10 per line. If the number of lines is not evenly divisible by 10, the last line in that group contains the remaining values, e.g. 4 values out of 14 in the last example above, for floating-point values. If there are no values of a particular type, no lines appear for that type, e.g. there are no integer lines in the last example above.

The *Bodies* section must appear after the *Atoms* section.

Bond Coeffs section:

- one line per bond type
- line syntax: ID coeffs

```
ID = bond type (1-N)
coeffs = list of coeffs
```
- example:

```
4 250 1.49
```

The number and meaning of the coefficients are specific to the defined bond style. See the [bond_style](#) and [bond_coeff](#) commands for details. Coefficients can also be set via the [bond_coeff](#) command in the input script.

BondAngle Coeffs section:

---

read_data command   985

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond Coeffs section:

- one line per angle type
- line syntax: ID coeffs

```
ID = angle type (1-N)
coeffs = list of coeffs (see class 2 section of angle_coeff)
```

BondBond13 Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Bonds section:

- one line per bond
- line syntax: ID type atom1 atom2

```
ID = bond number (1-Nbonds)
type = bond type (1-Nbondtype)
atom1,atom2 = IDs of 1st,2nd atoms in bond
```
- example:

```
12 3 17 29
```

The *Bonds* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Dihedral Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs
```
- example:

```
3 0.6 1 0 1
```

The number and meaning of the coefficients are specific to the defined dihedral style. See the dihedral_style and dihedral_coeff commands for details. Coefficients can also be set via the dihedral_coeff command in the input script.

Dihedrals section:

- one line per dihedral
- line syntax: ID type atom1 atom2 atom3 atom4

```
ID = number of dihedral (1-Ndihedrals)
type = dihedral type (1-Ndihedraltype)
atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in dihedral
```

- example:

```
12 4 17 29 30 21
```

The 4 atoms are ordered linearly within the dihedral. The *Dihedrals* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Ellipsoids section:

- one line per ellipsoid
- line syntax: atom-ID shapex shapey shapez quatw quati quatj quatk

```
atom-ID = ID of atom which is an ellipsoid
shapex,shapey,shapez = 3 diameters of ellipsoid (distance units)
quatw,quati,quatj,quatk = quaternion components for orientation of atom
```

- example:

```
12 1 2 1 1 0 0 0
```

The *Ellipsoids* section must appear if atom_style ellipsoid is used and any atoms are listed in the *Atoms* section with an ellipsoidflag = 1. The number of ellipsoids should be specified in the header section via the "ellipsoids" keyword.

The 3 shape values specify the 3 diameters or aspect ratios of a finite-size ellipsoidal particle, when it is oriented along the 3 coordinate axes. They must all be non-zero values.

The values *quatw*, *quati*, *quatj*, and *quatk* set the orientation of the atom as a quaternion (4-vector). Note that the shape attributes specify the aspect ratios of an ellipsoidal particle, which is oriented by default with its x-axis along the simulation box's x-axis, and similarly for y and z. If this body is rotated (via the right-hand rule) by an angle theta around a unit vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). These 4 components are quatw, quati, quatj, and quatk as specified above. LAMMPS normalizes each atom's quaternion in case (a,b,c) is not specified as a unit vector.

The *Ellipsoids* section must appear after the *Atoms* section.

EndBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Improper Coeffs section:

- one line per improper type
- line syntax: ID coeffs

```
ID = improper type (1-N)
coeffs = list of coeffs
```

- example:

```
2 20 0.0548311
```

read_data command

The number and meaning of the coefficients are specific to the defined improper style. See the improper_style and improper_coeff commands for details. Coefficients can also be set via the improper_coeff command in the input script.

Impropers section:

- one line per improper
- line syntax: ID type atom1 atom2 atom3 atom4

```
ID = number of improper (1-Nimpropers)
type = improper type (1-Nimpropertype)
atom1,atom2,atom3,atom4 = IDs of 1st,2nd,3rd,4th atoms in improper
```

- example:

```
12 3 17 29 13 100
```

The ordering of the 4 atoms determines the definition of the improper angle used in the formula for each [improper style](improper%20style). See the doc pages for individual styles for details.

The *Impropers* section must appear after the *Atoms* section. All values in this section must be integers (1, not 1.0).

Lines section:

- one line per line segment
- line syntax: atom-ID x1 y1 x2 y2

```
atom-ID = ID of atom which is a line segment
x1,y1 = 1st end point
x2,y2 = 2nd end point
```

- example:

```
12 1.0 0.0 2.0 0.0
```

The *Lines* section must appear if [atom_style line](atom_style%20line) is used and any atoms are listed in the *Atoms* section with a lineflag = 1. The number of lines should be specified in the header section via the "lines" keyword.

The 2 end points are the end points of the line segment. The ordering of the 2 points should be such that using a right-hand rule to cross the line segment with a unit vector in the +z direction, gives an "outward" normal vector perpendicular to the line segment. I.e. normal = (c2-c1) x (0,0,1). This orientation may be important for defining some interactions.

The *Lines* section must appear after the *Atoms* section.

Masses section:

- one line per atom type
- line syntax: ID mass

```
ID = atom type (1-N)
mass = mass value
```

- example:

```
3 1.01
```

This defines the mass of each atom type. This can also be set via the mass command in the input script. This section cannot be used for atom styles that define a mass for individual atoms - e.g. atom_style sphere.

MiddleBondTorsion Coeffs section:

- one line per dihedral type
- line syntax: ID coeffs

```
ID = dihedral type (1-N)
coeffs = list of coeffs (see class 2 section of dihedral_coeff)
```

Pair Coeffs section:

- one line per atom type
- line syntax: ID coeffs

```
ID = atom type (1-N)
coeffs = list of coeffs
```
- example:

```
3 0.022 2.35197 0.022 2.35197
```

The number and meaning of the coefficients are specific to the defined pair style. See the pair_style and pair_coeff commands for details. Since pair coefficients for types I != J are not specified, these will be generated automatically by the pair style's mixing rule. See the individual pair_style doc pages and the pair_modify mix command for details. Pair coefficients can also be set via the pair_coeff command in the input script.

PairIJ Coeffs section:

- one line per pair of atom types for all I,J with I <= J
- line syntax: ID1 ID2 coeffs

```
ID1 = atom type I = 1-N
ID2 = atom type J = I-N, with I <= J
coeffs = list of coeffs
```
- examples:

```
3 3 0.022 2.35197 0.022 2.35197
3 5 0.022 2.35197 0.022 2.35197
```

This section must have N*(N+1)/2 lines where N = # of atom types. The number and meaning of the coefficients are specific to the defined pair style. See the pair_style and pair_coeff commands for details. Since pair coefficients for types I != J are all specified, these values will turn off the default mixing rule defined by the pair style. See the individual pair_style doc pages and the pair_modify mix command for details. Pair coefficients can also be set via the pair_coeff command in the input script.

Triangles section:

- one line per triangle
- line syntax: atom-ID x1 y1 x2 y2

```
atom-ID = ID of atom which is a line segment
x1,y1,z1 = 1st corner point
x2,y2,z2 = 2nd corner point
x3,y3,z3 = 3rd corner point
```

- example:

```
12 0.0 0.0 0.0 2.0 0.0 1.0 0.0 2.0 1.0
```

The *Triangles* section must appear if [atom_style tri](#) is used and any atoms are listed in the *Atoms* section with a triangleflag = 1. The number of lines should be specified in the header section via the "triangles" keyword.

The 3 corner points are the corner points of the triangle. The ordering of the 3 points should be such that using a right-hand rule to go from point1 to point2 to point3 gives an "outward" normal vector to the face of the triangle. I.e. normal = (c2-c1) x (c3-c1). This orientation may be important for defining some interactions.

The *Triangles* section must appear after the *Atoms* section.

Velocities section:

- one line per atom
- line syntax: depends on atom style

all styles except those listed	atom-ID vx vy vz
electron	atom-ID vx vy vz ervel
ellipsoid	atom-ID vx vy vz lx ly lz
sphere	atom-ID vx vy vz wx wy wz
hybrid	atom-ID vx vy vz sub-style1 sub-style2 ...

where the keywords have these meanings:

vx,vy,vz = translational velocity of atom lx,ly,lz = angular momentum of aspherical atom wx,wy,wz = angular velocity of spherical atom ervel = electron radial velocity (0 for fixed-core):ul

The velocity lines can appear in any order. This section can only be used after an *Atoms* section. This is because the *Atoms* section must have assigned a unique atom ID to each atom so that velocities can be assigned to them.

Vx, vy, vz, and ervel are in [units](#) of velocity. Lx, ly, lz are in units of angular momentum (distance-velocity-mass). Wx, Wy, Wz are in units of angular velocity (radians/time).

For atom_style hybrid, following the 4 initial values (ID,vx,vy,vz), specific values for each sub-style must be listed. The order of the sub-styles is the same as they were listed in the [atom_style](#) command. The sub-style specific values are those that are not the 5 standard ones (ID,vx,vy,vz). For example, for the "sphere" sub-style, "wx", "wy", "wz" values would appear. These are listed in the same order they appear as listed above. Thus if

```
atom_style hybrid electron sphere
```

were used in the input script, each velocity line would have these fields:

```
atom-ID vx vy vz ervel wx wy wz
```

Translational velocities can also be set by the [velocity](#) command in the input script.

Restrictions:

To read gzipped data files, you must compile LAMMPS with the -DLAMMPS_GZIP option - see the [Making LAMMPS](#) section of the documentation.

Related commands:

[read_dump](), [read_restart](), [create_atoms](), [write_data]()

Default: none

read_dump command

Syntax:

```
read_dump file Nstep field1 field2 ... keyword values ...
```

- file = name of dump file to read
- Nstep = snapshot timestep to read from file
- one or more fields may be appended

```
field = x or y or z or vx or vy or vz or q or ix or iy or iz
  x,y,z = atom coordinates
  vx,vy,vz = velocity components
  q = charge
  ix,iy,iz = image flags in each dimension
```

- zero or more keyword/value pairs may be appended
- keyword = *box* or *replace* or *purge* or *trim* or *add* or *label* or *scaled* or *wrapped* or *format*

```
box value = yes or no = replace simulation box with dump box
  replace value = yes or no = overwrite atoms with dump atoms
  purge value = yes or no = delete all atoms before adding dump atoms
  trim value = yes or no = trim atoms not in dump snapshot
  add value = yes or no = add new dump atoms to system
  label value = field column
    field = one of the listed fields or id or type
    column = label on corresponding column in dump file
  scaled value = yes or no = coords in dump file are scaled/unscaled
  wrapped value = yes or no = coords in dump file are wrapped/unwrapped
  format values = format of dump file, must be last keyword if used
    native = native LAMMPS dump file
    xyz = XYZ file
    molfile style path = VMD molfile plugin interface
      style = dcd or xyz or others supported my mofile
      path = optional path for location of molfile plugins
```

Examples:

```
read_dump dump.file 5000 x y z
read_dump dump.xyz 5 x y z format xyz box no
read_dump dump.xyz 10 x y z format molfile box no reader xyz "../plugins"
read_dump dump.dcd 0 x y z format molfile box yes reader dcd
read_dump dump.file 1000 x y z vx vy vz format molfile box yes reader lammpstrj /usr/local/lib/vm
read_dump dump.file 5000 x y vx vy trim yes
read_dump ../run7/dump.file.gz 10000 x y z box yes
read_dump dump.xyz 5 x y z box no format xyz
read_dump dump.xyz 10 x y z box no format molfile xyz ../plugins
read_dump dump.dcd 0 x y z format molfile dcd
read_dump dump.file 1000 x y z vx vy vz format molfile lammpstrj /usr/local/lib/vmd/plugins/LINUX
```

Description:

Read atom information from a dump file to overwrite the current atom coordinates, and optionally the atom velocities and image flags and the simluation box dimensions. This is useful for restarting a run from a particular snapshot in a dump file. See the read_restart and read_data commands for alternative methods to do this. Also see the rerun command for a means of reading multiple snapshots from a dump file.

Note that a simulation box must already be defined before using the read_dump command. This can be done by the create_box, read_data, or read_restart commands. The read_dump command can reset the simulation

box dimensions, as explained below.

Also note that reading per-atom information from a dump snapshot is limited to the atom coordinates, velocities and image flags, as explained below. Other atom properties, which may be necessary to run a valid simulation, such as atom charge, or bond topology information for a molecular system, are not read from (or even contained in) dump files. Thus this auxiliary information should be defined in the usual way, e.g. in a data file read in by a read_data command, before using the read_dump command, or by the set command, after the dump snapshot is read.

If the dump filename specified as *file* ends with ".gz", the dump file is read in gzipped format. You cannot (yet) read a dump file that was written in binary format with a ".bin" suffix, or to multiple files via the "%" option in the dump file name. See the dump command for details.

The format of the dump file is selected through the *format* keyword. If specified, it must be the last keyword used, since all remaining arguments are passed on to the dump reader. The *native* format is for native LAMMPS dump files, written with a "dump atom".html or dump custom command. The *xyz* format is for generic XYZ formatted dump files,

The *molfile* format supports reading data through using the VMD molfile plugin interface. This dump reader format is only available, if the USER-MOLFILE package has been installed when compiling LAMMPS.

The *molfile* format takes one or two additional values. The *style* value determines the file format to be used and can be any format that the molfile plugins support, such as DCD or XYZ. Note that DCD dump files can be written by LAMMPS via the dump dcd command. The *path* value specifies a list of directories which LAMMPS will search for the molfile plugins appropriate to the specified *style*. The syntax of the *path* value is like other search paths: it can contain multiple directories separated by a colon (or semi-colon on windows). The *path* keyword is optional and defaults to ".", i.e. the current directory.

Support for other dump format readers may be added in the future.

Global information is first read from the dump file, namely timestep and box information.

The dump file is scanned for a snapshot with a time stamp that matches the specified *Nstep*. This means the LAMMPS timestep the dump file snapshot was written on for the *native* format. However, the *xyz* and *molfile* formats do not store the timestep. For these formats, timesteps are numbered logically, in a sequential manner, starting from 0. Thus to access the 10th snapshot in an *xyz* or *mofile* formatted dump file, use *Nstep* = 9.

The dimensions of the simulation box for the selected snapshot are also read; see the *box* keyword discussion below. For the *native* format, an error is generated if the snapshot is for a triclinic box and the current simulation box is orthogonal or vice versa. A warning will be generated if the snapshot box boundary conditions (periodic, shrink-wrapped, etc) do not match the current simulation boundary conditions, but the boundary condition information in the snapshot is otherwise ignored. See the "boundary" command for more details.

For the *xyz* format, no information about the box is available, so you must set the *box* flag to *no*. See details below.

For the *molfile* format, reading simulation box information is typically supported, but the location of the simulation box origin is lost and no explicit information about periodicity or orthogonal/triclinic box shape is available. The USER-MOLFILE package makes a best effort to guess based on heuristics, but this may not always work perfectly.

Per-atom information from the dump file snapshot is then read from the dump file snapshot. This corresponds to the specified *fields* listed in the read_dump command. It is an error to specify a z-dimension field, namely *z*,

vz, or *iz*, for a 2d simulation.

For dump files in *native* format, each column of per-atom data has a text label listed in the file. A matching label for each field must appear, e.g. the label "vy" for the field *vy*. For the *x*, *y*, *z* fields any of the following labels are considered a match:

```
x, xs, xu, xsu for field x
y, ys, yu, ysu for field y
z, zs, zu, zsu for field z
```

The meaning of xs (scaled), xu (unwrapped), and xsu (scaled and unwrapped) is explained on the dump command doc page. These labels are searched for in the list of column labels in the dump file, in order, until a match is found.

The dump file must also contain atom IDs, with a column label of "id".

If the *add* keyword is specified with a value of *yes*, as discussed below, the dump file must contain atom types, with a column label of "type".

If a column label you want to read from the dump file is not a match to a specified field, the *label* keyword can be used to specify the specific column label from the dump file to associate with that field. An example is if a time-averaged coordinate is written to the dump file via the fix ave/atom command. The column will then have a label corresponding to the fix-ID rather than "x" or "xs". The *label* keyword can also be used to specify new column labels for fields *id* and *type*.

For dump files in *xyz* format, only the *x*, *y*, and *z* fields are supported. The dump file does not store atom IDs, so these are assigned consecutively to the atoms as they appear in the dump file, starting from 1. Thus you should insure that order of atoms is consistent from snapshot to snapshot in the the XYZ dump file. See the dump_modify sort command if the XYZ dump file was written by LAMMPS.

For dump files in *molfile* format, the *x*, *y*, *z*, *vx*, *vy*, and *vz* fields can be specified. However, not all molfile formats store velocities, or their respective plugins may not support reading of velocities. The molfile dump files do not store atom IDs, so these are assigned consecutively to the atoms as they appear in the dump file, starting from 1. Thus you should insure that order of atoms are consistent from snapshot to snapshot in the the molfile dump file. See the dump_modify sort command if the dump file was written by LAMMPS.

Information from the dump file snapshot is used to overwrite or replace properties of the current system. There are various options for how this is done, determined by the specified fields and optional keywords.

The timestep of the snapshot becomes the current timestep for the simulation. See the reset_timestep command if you wish to change this after the dump snapshot is read.

If the *box* keyword is specified with a *yes* value, then the current simulation box dimensions are replaced by the dump snapshot box dimensions. If the *box* keyword is specified with a *no* value, the current simulatoin box is unchanged.

If the *purge* keyword is specified with a *yes* value, then all current atoms in the system are deleted before any of the operations invoked by the *replace*, *trim*, or *add* keywords take place.

If the *replace* keyword is specified with a *yes* value, then atoms with IDs that are in both the current system and the dump snapshot have their properties overwritten by field values. If the *replace* keyword is specified with a *no* value, atoms with IDs that are in both the current system and the dump snapshot are not modified.

If the *trim* keyword is specified with a *yes* value, then atoms with IDs that are in the current system but not in the dump snapshot are deleted. These atoms are unaffected if the *trim* keyword is specified with a *no* value.

If the *add* keyword is specified with a *yes* value, then atoms with IDs that are in the dump snapshot, but not in the current system are added to the system. These dump atoms are ignored if the *add* keyword is specified with a *no* value.

Note that atoms added via the *add* keyword will have only the attributes read from the dump file due to the *field* arguments. If *x* or *y* or *z* is not specified as a field, a value of 0.0 is used for added atoms. Added atoms must have an atom type, so this value must appear in the dump file.

Any other attributes (e.g. charge or particle diameter for spherical particles) will be set to default values, the same as if the create_atoms command were used.

Note that atom IDs are not preserved for new dump snapshot atoms added via the *add* keyword. The procedure for assigning new atom IDS to added atoms is the same as is described for the create_atoms command.

Atom coordinates read from the dump file are first converted into unscaled coordinates, relative to the box dimensions of the snapshot. These coordinates are then be assigned to an existing or new atom in the current simulation. The coordinates will then be remapped to the simulation box, whether it is the original box or the dump snapshot box. If periodic boundary conditions apply, this means the atom will be remapped back into the simulation box if necessary. If shrink-wrap boundary conditions apply, the new coordinates may change the simulation box dimensions. If fixed boundary conditions apply, the atom will be lost if it is outside the simulation box.

For *native* format dump files, the 3 xyz image flags for an atom in the dump file are set to the corresponding values appearing in the dump file if the *ix*, *iy*, *iz* fields are specified. If not specified, the image flags for replaced atoms are not changed and image flags for new atoms are set to default values. If coordinates read from the dump file are in unwrapped format (e.g. *xu*) then the image flags for read-in atoms are also set to default values. The remapping procedure described in the previous paragraph will then change images flags for all atoms (old and new) if periodic boundary conditions are applied to remap an atom back into the simulation box.

IMPORTANT NOTE: If you get a warning about inconsistent image flags after reading in a dump snapshot, it means one or more pairs of bonded atoms now have inconsistent image flags. As discussed in Section errors this may or may not cause problems for subsequent simulations, One way this can happen is if you read image flag fields from the dump file but do not also use the dump file box parameters.

LAMMPS knows how to compute unscaled and remapped coordinates for the snapshot column labels discussed above, e.g. *x*, *xs*, *xu*, *xsu*. If another column label is assigned to the *x* or *y* or *z* field via the *label* keyword, e.g. for coordinates output by the fix ave/atom command, then LAMMPS needs to know whether the coordinate information in the dump file is scaled and/or wrapped. This can be set via the *scaled* and *wrapped* keywords. Note that the value of the *scaled* and *wrapped* keywords is ignored for fields *x* or *y* or *z* if the *label* keyword is not used to assign a column label to that field.

The scaled/unscaled and wrapped/unwrapped setting must be identical for any of the *x*, *y*, *z* fields that are specified. Thus you cannot read *xs* and *yu* from the dump file. Also, if the dump file coordinates are scaled and the simulation box is triclinic, then all 3 of the *x*, *y*, *z* fields must be specified, since they are all needed to generate absolute, unscaled coordinates.

Restrictions:

To read gzipped dump files, you must compile LAMMPS with the -DLAMMPS_GZIP option - see the Making LAMMPS section of the documentation.

The *molfile* dump file formats are part of the USER-MOLFILE package. They are only enabled if LAMMPS was built with that packages. See the [Making LAMMPS](#) section for more info.

Related commands:

[dump](#), [dump molfile](#), [read_data](#), [read_restart](#), [rerun](#)

Default:

The option defaults are box = yes, replace = yes, purge = no, trim = no, add = no, scaled = no, wrapped = yes, and format = native.

---

read_restart command

Syntax:

```
read_restart file
```

- file = name of binary restart file to read in

Examples:

```
read_restart save.10000
read_restart restart.*
read_restart poly.*.%
```

Description:

Read in a previously saved simulation from a restart file. This allows continuation of a previous run. Information about what is stored in a restart file is given below.

Restart files are saved in binary format to enable exact restarts, meaning that the trajectories of a restarted run will precisely match those produced by the original run had it continued on.

Several things can prevent exact restarts due to round-off effects, in which case the trajectories in the 2 runs will slowly diverge. These include running on a different number of processors or changing certain settings such as those set by the [newton](#) or [processors](#) commands. LAMMPS will issue a warning in these cases.

Certain fixes will not restart exactly, though they should provide statistically similar results. These include [fix shake](#) and [fix langevin](#).

Certain pair styles will not restart exactly, though they should provide statistically similar results. This is because the forces they compute depend on atom velocities, which are used at half-step values every timestep when forces are computed. When a run restarts, forces are initially evaluated with a full-step velocity, which is different than if the run had continued. These pair styles include [granular pair styles](#), [pair dpd](#), and [pair lubricate](#).

If a restarted run is immediately different than the run which produced the restart file, it could be a LAMMPS bug, so consider [reporting it](#) if you think the behavior is wrong.

Because restart files are binary, they may not be portable to other machines. In this case, you can use the [-r command-line switch](#) to convert a restart file to a data file.

Similar to how restart files are written (see the [write_restart](#) and [restart](#) commands), the restart filename can contain two wild-card characters. If a "*" appears in the filename, the directory is searched for all filenames that match the pattern where "*" is replaced with a timestep value. The file with the largest timestep value is read in. Thus, this effectively means, read the latest restart file. It's useful if you want your script to continue a run from where it left off. See the [run](#) command and its "upto" option for how to specify the run command so it doesn't need to be changed either.

If a "%" character appears in the restart filename, LAMMPS expects a set of multiple files to exist. The [restart](#) and [write_restart](#) commands explain how such sets are created. Read_restart will first read a filename where "%" is replaced by "base". This file tells LAMMPS how many processors created the set and how many files are in it. Read_restart then reads the additional files. For example, if the restart file was specified as save.%

when it was written, then read_restart reads the files save.base, save.0, save.1, ... save.P-1, where P is the number of processors that created the restart file. The processors in the current LAMMPS simulation share the work of reading these files; each reads a roughly equal subset of the files. The number of processors which created the set can be different the number of processors in the current LAMMPS simulation. This can be a fast mode of input on parallel machines that support parallel I/O.

A restart file stores the following information about a simulation: units and atom style, simulation box size and shape and boundary settings, group definitions, per-type atom settings such as mass, per-atom attributes including their group assignments and molecular topology attributes, force field styles and coefficients, and special_bonds settings. This means that commands for these quantities do not need to be re-specified in the input script that reads the restart file, though you can redefine settings after the restart file is read.

One exception is that some pair styles do not store their info in restart files. The doc pages for individual pair styles note if this is the case. This is also true of bond_style hybrid (and angle_style, dihedral_style, improper_style hybrid).

All settings made by the pair_modify command, such as the shift and tail settings, are stored in the restart file with the pair style. The one exception is the pair_modify compute setting is not stored.

Information about kspace_style settings are not stored in the restart file. Hence if you wish to use an Ewald or PPPM solver, these commands must be re-issued after the restart file is read.

The list of fixes used for a simulation is not stored in the restart file. This means the new input script should specify all fixes it will use. Note that some fixes store an internal "state" which is written to the restart file. This allows the fix to continue on with its calculations in a restarted simulation. To re-enable such a fix, the fix command in the new input script must use the same fix-ID and group-ID as was used in the input script that wrote the restart file. If a match is found, LAMMPS prints a message indicating that the fix is being re-enabled. If no match is found before the first run or minimization is performed by the new script, the "state" information for the saved fix is discarded. See the doc pages for individual fixes for info on which ones can be restarted in this manner.

Bond interactions (angle, etc) that have been turned off by the fix shake or delete_bonds command will be written to a restart file as if they are turned on. This means they will need to be turned off again in a new run after the restart file is read.

Bonds that are broken (e.g. by a bond-breaking potential) are written to the restart file as broken bonds with a type of 0. Thus these bonds will still be broken when the restart file is read.

IMPORTANT NOTE: No other information is stored in the restart file. This means that an input script that reads a restart file should specify settings for quantities like timestep size, thermodynamic, neighbor list criteria including settings made via the neigh_modify command, dump file output, geometric regions, etc.

Restrictions: none

Related commands:

read_data, read_dump, write_restart, restart

Default: none

region command

Syntax:

```
region ID style args keyword arg ...
```

- ID = user-assigned name for the region
- style = *delete* or *block* or *cone* or *cylinder* or *plane* or *prism* or *sphere* or *mesh/tet* or *union* or *intersect* or *wedge*

```
delete = no args
  block args = xlo xhi ylo yhi zlo zhi
    xlo,xhi,ylo,yhi,zlo,zhi = bounds of block in all dimensions (distance units)
  cone args = dim c1 c2 radlo radhi lo hi
    dim = x or y or z = axis of cone
    c1,c2 = coords of cone axis in other 2 dimensions (distance units)
    radlo,radhi = cone radii at lo and hi end (distance units)
    lo,hi = bounds of cone in dim (distance units)
  cylinder args = dim c1 c2 radius lo hi
    dim = x or y or z = axis of cylinder
    c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
    radius = cylinder radius (distance units)
      radius can be a variable (see below)
    lo,hi = bounds of cylinder in dim (distance units)
  plane args = px py pz nx ny nz
    px,py,pz = point on the plane (distance units)
    nx,ny,nz = direction normal to plane (distance units)
  prism args = xlo xhi ylo yhi zlo zhi xy xz yz
    xlo,xhi,ylo,yhi,zlo,zhi = bounds of untilted prism (distance units)
    xy = distance to tilt y in x direction (distance units)
    xz = distance to tilt z in x direction (distance units)
    yz = distance to tilt z in y direction (distance units)
  sphere args = x y z radius
    x,y,z = center of sphere (distance units)
    radius = radius of sphere (distance units)
  mesh/tet args = file filename scale s move offx offy offz rotate phix phiy phiz
    file = obligatory keyword
    filename = name of ASCII VTK file containing the VTK tet-mesh data
    scale = obligatory keyword
    s = scaling factor for the mesh (dimensionless)
    move = obligatory keyword
    offx,offy,offz = offset for the mesh (distance units)
    rotate = obligatory keyword
    phix,phiy,phiz = angle of mesh rotation around x-, y-, and z-axis (in grad)
  wedge args = axis dim center c1 c2 radius r bounds lo hi angle0 alpha0 angle alpha
    axis = obligatory keyword
    dim = x or y or z ... wedge is aligned to this dimension
    center = obligatory keyword
    c1 = first center-coordinate
          if dim == x then y-coord
          if dim == y then z-coord
          if dim == z then x-coord
    c2 = second center-coordinate
          if dim == x then z-coord
          if dim == y then x-coord
          if dim == z then y-coord
    radius = obligatory keyword
    r = radius of cylinder
    bounds = obligatory keyword
    lo = dim-coord of lower flat face of cylinder of wedge
    hi = dim-coord of higher flat face of cylinder of wedge
    angle0 = obligatory keyword
```

```
        alpha0 = mathematically positive angle of the wedge's starting face
                 if axis == x then starting from y-axis
                 if axis == y then starting from z-axis
                 if axis == z then starting from x=axis
        angle = obligatory keyword
        alpha = mathematically positive angle between the wedge's starting and ending face in
      union args = N reg-ID1 reg-ID2 ...
        N = # of regions to follow, must be 2 or greater
        reg-ID1,reg-ID2, ... = IDs of regions to join together
      intersect args = N reg-ID1 reg-ID2 ...
        N = # of regions to follow, must be 2 or greater
        reg-ID1,reg-ID2, ... = IDs of regions to intersect
```

- zero or more keyword/arg pairs may be appended
- keyword = *side* or *units* or *move* or *rotate*

```
  side value = in or out
      in = the region is inside the specified geometry
      out = the region is outside the specified geometry
   units value = lattice or box
     lattice = the geometry is defined in lattice units
     box = the geometry is defined in simulation box units
   move args = v_x v_y v_z
     v_x,v_y,v_z = equal-style variables for x,y,z displacement of region over time
   rotate args = v_theta Px Py Pz Rx Ry Rz
     v_theta = equal-style variable for rotaton of region over time (in radians)
     Px,Py,Pz = origin for axis of rotation (distance units)
     Rx,Ry,Rz = axis of rotation vector
```

Examples:

```
region 1 block -3.0 5.0 INF 10.0 INF INF
region 2 sphere 0.0 0.0 0.0 5 side out
region void cylinder y 2 3 5 -5.0 EDGE units box
region 1 prism 0 10 0 10 0 10 2 0 0
region outside union 4 side1 side2 side3 side4
region 2 sphere 0.0 0.0 0.0 5 side out move v_left v_up NULL
region 1 wedge axis y  center 0 0  radius 10 bounds 0 10 angle0 -22.5 angle 45 units box side in
```

Description:

This command defines a geometric region of space. Various other commands use regions. For example, the region can be filled with atoms via the [create_atoms] command. Or a bounding box around the region, can be used to define the simulation box via the [create_box] command. Or the atoms in the region can be identified as a group via the [group] command, or deleted via the [delete_atoms] command. Or the surface of the region can be used as a boundary wall via the [fix wall/region] command.

Commands which use regions typically test whether an atom's position is contained in the region or not. For this purpose, coordinates exactly on the region boundary are considered to be interior to the region. This means, for example, for a spherical region, an atom on the sphere surface would be part of the region if the sphere were defined with the *side in* keyword, but would not be part of the region if it were defined using the *side out* keyword. See more details on the *side* keyword below.

Normally, regions in LAMMPS are "static", meaning their geometric extent does not change with time. If the *move* or *rotate* keyword is used, as described below, the region becomes "dynamic", meaning it's location or orientation changes with time. This may be useful, for example, when thermostatting a region, via the compute temp/region command, or when the fix wall/region command uses a region surface as a bounding wall on particle motion, i.e. a rotating container.

The *delete* style removes the named region. Since there is little overhead to defining extra regions, there is normally no need to do this, unless you are defining and discarding large numbers of regions in your input

region command

script.

The lo/hi values for *block* or *cone* or *cylinder* or *prism* styles can be specified as EDGE or INF. EDGE means they extend all the way to the global simulation box boundary. Note that this is the current box boundary; if the box changes size during a simulation, the region does not. INF means a large negative or positive number (1.0e20), so it should encompass the simulation box even if it changes size. If a region is defined before the simulation box has been created (via create_box or read_data or read_restart commands), then an EDGE or INF parameter cannot be used. For a *prism* region, a non-zero tilt factor in any pair of dimensions cannot be used if both the lo/hi values in either of those dimensions are INF. E.g. if the xy tilt is non-zero, then xlo and xhi cannot both be INF, nor can ylo and yhi.

IMPORTANT NOTE: Regions in LAMMPS do not get wrapped across periodic boundaries, as specified by the boundary command. For example, a spherical region that is defined so that it overlaps a periodic boundary is not treated as 2 half-spheres, one on either side of the simulation box.

IMPORTANT NOTE: Regions in LAMMPS are always 3d geometric objects, regardless of whether the dimension of a simulation is 2d or 3d. Thus when using regions in a 2d simulation, you should be careful to define the region so that its intersection with the 2d x-y plane of the simulation has the 2d geometric extent you want.

For style *cone*, an axis-aligned cone is defined which is like a *cylinder* except that two different radii (one at each end) can be defined. Either of the radii (but not both) can be 0.0.

For style *cone* and *cylinder*, the c1,c2 params are coordinates in the 2 other dimensions besides the cylinder axis dimension. For dim = x, c1/c2 = y/z; for dim = y, c1/c2 = x/z; for dim = z, c1/c2 = x/y. Thus the third example above specifies a cylinder with its axis in the y-direction located at x = 2.0 and z = 3.0, with a radius of 5.0, and extending in the y-direction from -5.0 to the upper box boundary.

For style *plane*, a plane is defined which contain the point (px,py,pz) and has a normal vector (nx,ny,nz). The normal vector does not have to be of unit length. The "inside" of the plane is the half-space in the direction of the normal vector; see the discussion of the *side* option below.

For style *prism*, a parallelepiped is defined (it's too hard to spell parallelepiped in an input script!). The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by A = (xhi-xlo,0,0); B = (xy,yhi-ylo,0); C = (xz,yz,zhi-zlo). *Xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped.

A prism region that will be used with the create_box command to define a triclinic simulation box must have tilt factors (xy,xz,yz) that do not skew the box more than half the distance of corresponding the parallel box length. For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are all geometrically equivalent.

The *radius* value for style *sphere* and *cylinder* can be specified as an equal-style variable. If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated each timestep, and its value used to determine the radius of the region.

Equal-style variables can specify formulas with various mathematical functions, and include thermo_style command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a time-dependent radius.

See Section_howto 12 of the doc pages for a geometric description of triclinic boxes, as defined by

region command   1001

LAMMPS, and how to transform these parameters to and from other commonly used triclinic representations.

For style *mesh/tet*, a tetrahedral mesh can be read from an ASCII VTK file. You can apply offset, scaling and rotation to the imported mesh via dedicated keywords. If applying more then one of these operations, the offset is applied first and then the geometry is scaled. Then the geometry is rotated around the x-axis first, then around the y-axis, then around the z-axis.

IMPORTANT NOTE: Currently only ASCII VTK containing tetrahedra are supported. For periodic boundaries, the mesh is NOT mapped. Instead, a warning is generated if a vertex lies outside the simulation box.

The *union* style creates a region consisting of the volume of all the listed regions combined. The *intersect* style creates a region consisting of the volume that is common to all the listed regions.

IMPORTANT NOTE: The *union* and *intersect* regions operate by invoking methods from their list of sub-regions. Thus you cannot delete the sub-regions after defining the *union* or *intersection* region.

The *side* keyword determines whether the region is considered to be inside or outside of the specified geometry. Using this keyword in conjunction with *union* and *intersect* regions, complex geometries can be built up. For example, if the interior of two spheres were each defined as regions, and a *union* style with *side* = out was constructed listing the region-IDs of the 2 spheres, the resulting region would be all the volume in the simulation box that was outside both of the spheres.

The *units* keyword determines the meaning of the distance units used to define the region for any argument above listed as having distance units. It also affects the scaling of the velocity vector specfied with the *vel* keyword, the amplitude vector specified with the *wiggle* keyword, and the rotation point specified with the *rotate* keyword, since they each involve a distance metric.

A *box* value selects standard distance units as defined by the <u>units</u> command, e.g. Angstroms for units = real or metal. A *lattice* value means the distance units are in lattice spacings. The <u>lattice</u> command must have been previously used to define the lattice spacings which are used as follows:

- For style *block*, the lattice spacing in dimension x is applied to xlo and xhi, similarly the spacings in dimensions y,z are applied to ylo/yhi and zlo/zhi.
- For style *cone*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The two cone radii are scaled by the lattice spacing in the dimension corresponding to c1.
- For style *cylinder*, the lattice spacing in argument *dim* is applied to lo and hi. The spacings in the two radial dimensions are applied to c1 and c2. The cylinder radius is scaled by the lattice spacing in the dimension corresponding to c1.
- For style *plane*, the lattice spacing in dimension x is applied to px and nx, similarly the spacings in dimensions y,z are applied to py/ny and pz/nz.
- For style *prism*, the lattice spacing in dimension x is applied to xlo and xhi, similarly for ylo/yhi and zlo/zhi. The lattice spacing in dimension x is applied to xy and xz, and the spacing in dimension y to yz.
- For style *sphere*, the lattice spacing in dimensions x,y,z are applied to the sphere center x,y,z. The spacing in dimension x is applied to the sphere radius.

If the *move* or *rotate* keywords are used, the region is "dynamic", meaning its location or orientation changes with time. These keywords cannot be used with a *union* or *intersect* style region. Instead, the keywords should be used to make the individual sub-regions of the *union* or *intersect* region dynamic. Normally, each sub-region should be "dynamic" in the same manner (e.g. rotate around the same point), though this is not a requirement.

The *move* keyword allows one or more [equal-style variables](#) to be used to specify the x,y,z displacement of the region, typically as a function of time. A variable is specified as v_name, where name is the variable name. Any of the three variables can be specified as NULL, in which case no displacement is calculated in that dimension.

Note that equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. Thus it is easy to specify a region displacement that change as a function of time or spans consecutive runs in a continuous fashion. For the latter, see the *start* and *stop* keywords of the [run](#) command and the *elaplong* keyword of [thermo_style custom](#) for details.

For example, these commands would displace a region from its initial position, in the positive x direction, effectively at a constant velocity:

```
variable dx equal ramp(0,10)
region 2 sphere 10.0 10.0 0.0 5 move v_dx NULL NULL
```

Note that the initial displacemet is 0.0, though that is not required.

Either of these varaibles would "wiggle" the region back and forth in the y direction:

```
variable dy equal swiggle(0,5,100)
variable dysame equal 5*sin(2*PI*elaplong*dt/100)
region 2 sphere 10.0 10.0 0.0 5 move NULL v_dy NULL
```

The *rotate* keyword rotates the region around a rotation axis *R* = (Rx,Ry,Rz) that goes thru a point *P* = (Px,Py,Pz). The rotation angle is calculated, presumably as a function of time, by a variable specified as v_theta, where theta is the variable name. The variable should generate its result in radians. The direction of rotation for the region around the rotation axis is consistent with the right-hand rule: if your right-hand thumb points along *R*, then your fingers wrap around the axis in the direction of rotation.

The *move* and *rotate* keywords can be used together. In this case, the displacement specified by the *move* keyword is applied to the *P* point of the *rotate* keyword.

Restrictions:

A prism cannot be of 0.0 thickness in any dimension; use a small z thickness for 2d simulations. For 2d simulations, the xz and yz parameters must be 0.0.

Related commands:

[lattice](#), [create_atoms](#), [delete_atoms](#), [group](#)

Default:

The option defaults are side = in, units = lattice, and no move or rotation.

replicate command

Syntax:

```
replicate nx ny nz
```

- nx,ny,nz = replication factors in each dimension

Examples:

```
replicate 2 3 2
```

Description:

Replicate the current simulation one or more times in each dimension. For example, replication factors of 2,2,2 will create a simulation with 8x as many atoms by doubling the simulation domain in each dimension. A replication factor of 1 in a dimension leaves the simulation domain unchanged.

All properties of the atoms are replicated, including their velocities, which may or may not be desirable. New atom IDs are assigned to new atoms, as are molecule IDs. Bonds and other topology interactions are created between pairs of new atoms as well as between old and new atoms. This is done by using the image flag for each atom to "unwrap" it out of the periodic box before replicating it.

This means that any molecular bond you specify in the original data file that crosses a periodic boundary should be between two atoms with image flags that differ by 1. This will allow the bond to be unwrapped appropriately.

Restrictions:

A 2d simulation cannot be replicated in the z dimension.

If a simulation is non-periodic in a dimension, care should be used when replicating it in that dimension, as it may put atoms nearly on top of each other.

IMPORTANT NOTE: You cannot use the replicate command on a system which has a molecule that spans the box and is bonded to itself across a periodic boundary, so that the molecule is efffectively a loop. A simple example would be a linear polymer chain that spans the simulation box and bonds back to itself across the periodic boundary. More realistic examples would be a CNT (meant to be an infinitely long CNT) or a graphene sheet or a bulk periodic crystal where there are explicit bonds specified between near neighbors. (Note that this only applies to systems that have permanent bonds as specified in the data file. A CNT that is just atoms modeled with the AIREBO potential has no such permanent bonds, so it can be replicated.) The reason replication does not work with those systems is that the image flag settings described above cannot be made consistent. I.e. it is not possible to define images flags so that when every pair of bonded atoms is unwrapped (using the image flags), they will be close to each other. The only way the replicate command could work in this scenario is for it to break a bond, insert more atoms, and re-connect the loop for the larger simulation box. But it is not clever enough to do this. So you will have to construct a larger version of your molecule as a pre-processing step and input a new data file to LAMMPS.

If the current simulation was read in from a restart file (before a run is performed), there can have been no fix information stored in the file for individual atoms. Similarly, no fixes can be defined at the time the replicate command is used that require vectors of atom information to be stored. This is because the replicate command does not know how to replicate that information for new atoms it creates.

Replicating a system that has rigid bodies (defined via the fix rigid command), either currently defined or that created the restart file which was read in before replicating, can cause problems if there is a bond between a pair of rigid bodies that straddle a periodic boundary. This is because the periodic image information for particles in the rigid bodies are set differently than for a non-rigid system and can result in a new bond being created that spans the periodic box. Thus you cannot use the replicate command in this scenario.

Related commands: none

Default: none

rerun command

Syntax:

```
rerun file1 file2 ... keyword args ...
```

- file1,file2,... = dump file(s) to read
- one or more keywords may be appended, keyword *dump* must appear and be last

```
keyword = first or last or every or skip or start or stop or dump
  first args = Nfirts
    Nfirst = dump timestep to start on
  last args = Nlast
    Nlast = dumptimestep to stop on
  every args = Nevery
    Nevery = read snapshots matching every this many timesteps
  skip args = Nskip
    Nskip = read one out of every Nskip snapshots
  start args = Nstart
    Nstart = timestep on which pseudo run will start
  stop args = Nstop
    Nstop = timestep to which pseudo run will end
  dump args = same as read_dump command starting with its field arguments
```

Examples:

```
rerun dump.file dump x y z vx vy vz
rerun dump1.txt dump2.txt first 10000 every 1000 dump x y z
rerun dump.vels dump x y z vx vy vz box yes format molfile lammpstrj
rerun dump.dcd dump x y z box no format molfile dcd
rerun ../run7/dump.file.gz skip 2 dump x y z box yes
```

Description:

Perform a psuedo simulation run where atom information is read one snapshot at a time from a dump file(s), and energies and forces are computed on the shapshot to produce thermodynamic or other output.

This can be useful in the following kinds of scenarios, after an initial simulation produced the dump file:

- Compute the energy and forces of snaphots using a different potential.
- Calculate one or more diagnostic quantities on the snapshots that weren't computed in the initial run. These can also be computed with settings not used in the initial run, e.g. computing an RDF via the [compute rdf](#) command with a longer cutoff than was used initially.
- Calculate the portion of per-atom forces resulting from a subset of the potential. E.g. compute only Coulombic forces. This can be done by only defining only a Coulombic pair style in the rerun script. Doing this in the original script would result in different (bad) dynamics.

Conceptually, using the rerun command is like running an input script that has a loop in it (see the [next](#) and [jump](#) commands). Each iteration of the loop reads one snapshot from the dump file via the [read_dump](#) command, sets the timestep to the appropriate value, and then invokes a [run](#) command for zero timesteps to simply compute energy and forces, and any other [thermodynamic output](#) or diagnostic info you have defined. This computation also invokes any fixes you have defined that apply constraints to the system, such as [fix shake](#) or [fix indent](#).

Note that a simulation box must already be defined before using the rerun command. This can be done by the [create_box](#), [read_data](#), or [read_restart](#) commands.

Also note that reading per-atom information from dump snapshots is limited to the atom coordinates, velocities and image flags as explained in the read_dump command. Other atom properties, which may be necessary to compute energies and forces, such as atom charge, or bond topology information for a molecular system, are not read from (or even contained in) dump files. Thus this auxiliary information should be defined in the usual way, e.g. in a data file read in by a read_data command, before using the rerun command.

If more than one dump file is specified, the dump files are read one after the other. It is assumed that snapshot timesteps will be in ascending order. If a snapshot is encountered that is not in ascending order, it will cause the rerun command to complete.

The *first*, *last*, *every*, *skip* keywords determine which snapshots are read from the dump file(s). Snapshots are skipped until they have a timestamp >= *Nfirst*. When a snapshot with a timestamp > *Nlast* is encountered, the rerun command finishes. Note below that the defaults for *first* and *last* are to read all snapshots. If the *every* keyword is set to a value > 0, then only snapshots with timestamps that are a multiple of *Nevery* are read (the first snapshot is always read). If *Nevery* = 0, then this criterion is ignored, i.e. every snapshot is read that meets the other criteria. If the *skip* keyword is used, then after the first snapshot is read, every Nth snapshot is read, where N = *Nskip*. E.g. if *Nskip* = 3, then only 1 out of every 3 snapshots is read, assuming the snapshot timestamp is also consistent with the other criteria.

The *start* and *stop* keywords have the same meaning that they do for the run command. They only need to be defined if (a) you are using a fix command that changes some value over time, and (b) you want the reference point for elapsed time (from start to stop) to be different than the *first* and *last* settings. See the doc page for individual fixes to see which ones can be used with the *start/stop* keywords. Note that if you define neither of the *start/stop* or *first/last* keywords, then LAMMPS treats the pseudo run as going from 0 to a huge value (effectively infinity). This means that any quantity that a fix scales as a fraction of elapsed time in the run, will essentially remain at its intiial value.

The *dump* keyword is required and must be the last keyword specified. Its arguments are passed internally to the read_dump command. The first argument following the *dump* keyword should be the *field1* argument of the read_dump command. See the read_dump doc page for details on the various options it allows for extracting information from the dump file snapshots, and for using that information to alter the LAMMPS simulation.

In general, a LAMMPS input script that uses a rerun command can include and perform all the usual operations of an input script that uses the run command. There are a few exceptions and points to consider, as discussed here.

Fixes that perform time integration, such as fix nve or fix npt are not invoked, since no time integration is performed. Fixes that perturb or constrain the forces on atoms will be invoked, just as they would during a normal run. Examples are fix indent and fix langevin. So you should think carefully as to whether that makes sense for the manner in which you are reprocessing the dump snapshots.

If you only want the rerun script to perform analyses that do not involve pair interactions, such as use compute msd to calculated displacements over time, you do not need to define a pair style, which may also mean neighbor lists will not need to be calculated which saves time. The communicate cutoff command can also be used to insure ghost atoms are acquired from far enough away for operations like bond and angle evaluations, if no pair style is being used.

Every time a snapshot is read, the timestep for the simulation is reset, as if the >reset_timestep command were used. This command has some restrictions as to what fixes can be defined. See its doc page for details. For example, the fix deposit and fix dt/reset fixes are in this category. They also make no sense to use with a rerun command.

If time-averaging fixes like [fix ave/time](#) are used, they are invoked on timesteps that are a function of their *Nevery*, *Nrepeat*, and *Nfreq* settings. As an example, see the [fix ave/time](#) doc page for details. You must insure those settings are consistent with the snapshot timestamps that are read from the dump file(s). If an averaging fix is not invoked on a timestep it expects to be, LAMMPS will flag an error.

The various forms of LAMMPS output, as defined by the [thermo_style](#), [thermo](#), [dump](#), and [restart](#) commands occur on specific timesteps. If successvive dump snapshots skip those timesteps, then no output will be produced. E.g. if you request thermodynamic output every 100 steps, but the dump file snapshots are every 1000 steps, then you will only see thermodynamic output every 1000 steps.

Restrictions:

To read gzipped dump files, you must compile LAMMPS with the -DLAMMPS_GZIP option - see the [Making LAMMPS](#) section of the documentation.

Related commands:

[read_dump](#)

Default:

The option defaults are first = 0, last = a huge value (effectively infinity), start = same as first, stop = same as last, every = 0, skip = 1;

reset_timestep command

Syntax:

```
reset_timestep N
```

- N = timestep number

Examples:

```
reset_timestep 0
reset_timestep 4000000
```

Description:

Set the timestep counter to the specified value. This command normally comes after the timestep has been set by reading a restart file via the [read_restart](#) command, or a previous simulation advanced the timestep.

The [read_data](#) and [create_box](#) commands set the timestep to 0; the [read_restart](#) command sets the timestep to the value it had when the restart file was written.

Restrictions: none

This command cannot be used when any fixes are defined that keep track of elapsed time to perform certain kinds of time-dependent operations. Examples are the [fix deposit](#) and [fix dt/reset](#) commands. The former adds atoms on specific timesteps. The latter keeps track of accumulated time.

Various fixes use the current timestep to calculate related quantities. If the timestep is reset, this may produce unexpected behavior, but LAMMPS allows the fixes to be defined even if the timestep is reset. For example, commands which thermostat the system, e.g. [fix nvt](#), allow you to specify a target temperature which ramps from Tstart to Tstop which may persist over several runs. If you change the timestep, you may induce an instantaneous change in the target temperature.

Resetting the timestep clears flags for [computes](#) that may have calculated some quantity from a previous run. This means these quantity cannot be accessed by a variable in between runs until a new run is performed. See the [variable](#) command for more details.

Related commands:

[rerun](#)

Default: none

restart command

Syntax:

```
restart 0
restart N root keyword value ...
restart N file1 file2 keyword value ...
```

- N = write a restart file every this many timesteps
- N can be a variable (see below)
- root = filename to which timestep # is appended
- file1,file2 = two full filenames, toggle between them when writing file
- zero or more keyword/value pairs may be appended
- keyword = *fileper* or *nfile*

```
  fileper arg = Np
      Np = write one file for every this many processors
    nfile arg = Nf
      Nf = write this many files, one from each of Nf processors
```

Examples:

```
restart 0
restart 1000 poly.restart
restart 1000 restart.*.equil
restart 10000 poly.%.1 poly.%.2
restart v_mystep poly.restart
```

Description:

Write out a binary restart file every so many timesteps, in either or both of two modes, as a run proceeds. A value of 0 means do not write out any restart files. The two modes are as follows. If one filename is specified, a series of filenames will be created which include the timestep in the filename. If two filenames are specified, only 2 restart files will be created, with those names. LAMMPS will toggle between the 2 names as it writes successive restart files.

Note that you can specify the restart command twice, once with a single filename and once with two filenames. This would allow you, for example, to write out archival restart files every 100000 steps using a single filenname, and more frequent temporary restart files every 1000 steps, using two filenames. Using restart 0 will turn off both modes of output.

Similar to [dump](#) files, the restart filename(s) can contain two wild-card characters.

If a "*" appears in the single filename, it is replaced with the current timestep value. This is only recognized when a single filename is used (not when toggling back and forth). Thus, the 3rd example above creates restart files as follows: restart.1000.equil, restart.2000.equil, etc. If a single filename is used with no "*", then the timestep value is appended. E.g. the 2nd example above creates restart files as follows: poly.restart.1000, poly.restart.2000, etc.

If a "%" character appears in the restart filename(s), then one file is written for each processor and the "%" character is replaced with the processor ID from 0 to P-1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written on step 1000 for filename restart.% would be restart.base.1000, restart.0.1000, restart.1.1000, ..., restart.P-1.1000. This creates smaller files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O. The

optional *fileper* and *nfile* keywords discussed below can alter the number of files written.

Restart files are written on timesteps that are a multiple of N but not on the first timestep of a run or minimization. You can use the write_restart command to write a restart file before a run begins. A restart file is not written on the last timestep of a run unless it is a multiple of N. A restart file is written on the last timestep of a minimization if N > 0 and the minimization converges.

Instead of a numeric value, N can be specifed as an equal-style variable, which should be specified as v_name, where name is the variable name. In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which a restart file will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for equal-style variables, as examples of useful functions to use in this context. Other similar math functions could easily be added as options for equal-style variables.

For example, the following commands will write restart files every step from 1100 to 1200, and could be useful for debugging a simulation where something goes wrong at step 1163:

```
variable        s equal stride(1100,1200,1)
restart         v_s tmp.restart
```

See the read_restart command for information about what is stored in a restart file.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, you can use the -r command-line switch to convert a restart file to a data file.

The optional *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified restart file name(s). As explained above, the "%" character causes the restart file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a restart file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a restart file.

Restrictions: none

Related commands:

write_restart, read_restart

Default:

```
restart 0
```

run command

Syntax:

```
run N keyword values ...
```

- N = # of timesteps
- zero or more keyword/value pairs may be appended
- keyword = *upto* or *start* or *stop* or *pre* or *post* or *every*

```
upto value = none
  start value = N1
    N1 = timestep at which 1st run started
  stop value = N2
    N2 = timestep at which last run will end
  pre value = no or yes
  post value = no or yes
  every values = M c1 c2 ...
    M = break the run into M-timestep segments and invoke one or more commands between each
    c1,c2,...,cN = one or more LAMMPS commands, each enclosed in quotes
    c1 = NULL means no command will be invoked
```

Examples:

```
run 10000
run 1000000 upto
run 100 start 0 stop 1000
run 1000 pre no post yes
run 100000 start 0 stop 1000000 every 1000 "print 'Protein Rg = $r'"
run 100000 every 1000 NULL
```

Description:

Run or continue dynamics for a specified number of timesteps.

When the run style is *respa*, N refers to outer loop (largest) timesteps.

A value of N = 0 is acceptable; only the thermodynamics of the system are computed and printed without taking a timestep.

The *upto* keyword means to perform a run starting at the current timestep up to the specified timestep. E.g. if the current timestep is 10,000 and "run 100000 upto" is used, then an additional 90,000 timesteps will be run. This can be useful for very long runs on a machine that allocates chunks of time and terminate your job when time is exceeded. If you need to restart your script multiple times (reading in the last restart file), you can keep restarting your script with the same run command until the simulation finally completes.

The *start* or *stop* keywords can be used if multiple runs are being performed and you want a fix command that changes some value over time (e.g. temperature) to make the change across the entire set of runs and not just a single run. See the doc page for individual fixes to see which ones can be used with the *start/stop* keywords.

For example, consider this fix followed by 10 run commands:

```
fix        1 all nvt 200.0 300.0 1.0
run        1000 start 0 stop 10000
run        1000 start 0 stop 10000
...
```

```
run          1000 start 0 stop 10000
```

The NVT fix ramps the target temperature from 200.0 to 300.0 during a run. If the run commands did not have the start/stop keywords (just "run 1000"), then the temperature would ramp from 200.0 to 300.0 during the 1000 steps of each run. With the start/stop keywords, the ramping takes place over the 10000 steps of all runs together.

The *pre* and *post* keywords can be used to streamline the setup, clean-up, and associated output to the screen that happens before and after a run. This can be useful if you wish to do many short runs in succession (e.g. LAMMPS is being called as a library which is doing other computations between successive short LAMMPS runs).

By default (pre and post = yes), LAMMPS creates neighbor lists, computes forces, and imposes fix constraints before every run. And after every run it gathers and prints timings statistics. If a run is just a continuation of a previous run (i.e. no settings are changed), the initial computation is not necessary; the old neighbor list is still valid as are the forces. So if *pre* is specified as "no" then the initial setup is skipped, except for printing thermodynamic info. Note that if *pre* is set to "no" for the very 1st run LAMMPS performs, then it is overridden, since the initial setup computations must be done.

IMPORTANT NOTE: If your input script changes settings between 2 runs (e.g. adds a [fix] or [dump] or [compute] or changes a [neighbor] list parameter), then the initial setup must be performed. LAMMPS does not check for this, but it would be an error to use the *pre no* option in this case.

If *post* is specified as "no", the full timing summary is skipped; only a one-line summary timing is printed.

The *every* keyword provides a means of breaking a LAMMPS run into a series of shorter runs. Optionally, one or more LAMMPS commands (c1, c2, ..., cN) will be executed in between the short runs. If used, the *every* keyword must be the last keyword, since it has a variable number of arguments. Each of the trailing arguments is a single LAMMPS command, and each command should be enclosed in quotes, so that the entire command will be treated as a single argument. This will also prevent any variables in the command from being evaluated until it is executed multiple times during the run. Note that if a command itself needs one of its arguments quoted (e.g. the [print] command), then you can use a combination of single and double quotes, as in the example above or below.

The *every* keyword is a means to avoid listing a long series of runs and interleaving commands in your input script. For example, a [print] command could be invoked or a [fix] could be redefined, e.g. to reset a thermostat temperature. Or this could be useful for invoking a command you have added to LAMMPS that wraps some other code (e.g. as a library) to perform a computation periodically during a long LAMMPS run. See [this section] of the documentation for info about how to add new commands to LAMMPS. See [this section] of the documentation for ideas about how to couple LAMMPS to other codes.

With the *every* option, N total steps are simulated, in shorter runs of M steps each. After each M-length run, the specified commands are invoked. If only a single command is specified as NULL, then no command is invoked. Thus these lines:

```
variable q equal x[100]
run 6000 every 2000 "print Coord = $q"
```

are the equivalent of:

```
variable q equal x[100]
run 2000
print Coord = $q
run 2000
print Coord = $q
run 2000
```

run command   1013

```
print Coord = $q
```

which does 3 runs of 2000 steps and prints the x-coordinate of a particular atom between runs. Note that the variable "$q" will be evaluated afresh each time the print command is executed.

Note that by using the line continuation character "&", the run every command can be spread across many lines, though it is still a single command:

```
run 100000 every 1000 &
  "print 'Minimum value = $a'" &
  "print 'Maximum value = $b'" &
  "print 'Temp = $c'" &
  "print 'Press = $d'"
```

If the *pre* and *post* options are set to "no" when used with the *every* keyword, then the 1st run will do the full setup and the last run will print the full timing summary, but these operations will be skipped for intermediate runs.

IMPORTANT NOTE: You might hope to specify a command that exits the run by jumping out of the loop, e.g.

```
variable t equal temp
run 10000 every 100 "if '$t <300.0' then 'jump SELF afterrun'"
```

Unfortunately this will not currently work. The run command simply executes each command one at a time each time it pauses, then continues the run. You can replace the jump command with a simple quit command and cause LAMMPS to exit during the middle of a run when the condition is met.

Restrictions:

The number of specified timesteps N must fit in a signed 32-bit integer, so you are limited to slightly more than 2 billion steps (2^31) in a single run. However, you can perform successive runs to run a simulation for any number of steps (ok, up to 2^63 steps).

Related commands:

minimize, run_style, temper

Default:

The option defaults are start = the current timestep, stop = current timestep + N, pre = yes, and post = yes.

run_style command

Syntax:

```
run_style style args
```

- style = *verlet* or *verlet/split* or *respa* or *respa/omp*

```
  verlet args = none
    verlet/split args = none
    respa args = N n1 n2 ... keyword values ...
      N = # of levels of rRESPA
      n1, n2, ... = loop factor between rRESPA levels (N-1 values)
      zero or more keyword/value pairings may be appended to the loop factors
      keyword = bond or angle or dihedral or improper or
        pair or inner or middle or outer or kspace
        bond value = M
          M = which level (1-N) to compute bond forces in
        angle value = M
          M = which level (1-N) to compute angle forces in
        dihedral value = M
          M = which level (1-N) to compute dihedral forces in
        improper value = M
          M = which level (1-N) to compute improper forces in
        pair value = M
          M = which level (1-N) to compute pair forces in
        inner values = M cut1 cut2
          M = which level (1-N) to compute pair inner forces in
          cut1 = inner cutoff between pair inner and
                 pair middle or outer  (distance units)
          cut2 = outer cutoff between pair inner and
                 pair middle or outer  (distance units)
        middle values = M cut1 cut2
          M = which level (1-N) to compute pair middle forces in
          cut1 = inner cutoff between pair middle and pair outer (distance units)
          cut2 = outer cutoff between pair middle and pair outer (distance units)
        outer value = M
          M = which level (1-N) to compute pair outer forces in
        kspace value = M
          M = which level (1-N) to compute kspace forces in
```

Examples:

```
run_style verlet
run_style respa 4 2 2 2 bond 1 dihedral 2 pair 3 kspace 4
run_style respa 4 2 2 2 bond 1 dihedral 2 inner 3 5.0 6.0 outer 4 kspace 4
```

Description:

Choose the style of time integrator used for molecular dynamics simulations performed by LAMMPS.

The *verlet* style is a standard velocity-Verlet integrator.

The *verlet/split* style is also a velocity-Verlet integrator, but it splits the force calculation within each timestep over 2 partitions of processors. See [Section_start 6](#) for an explanation of the -partition command-line switch.

Specifically, this style performs all computation except the [kspace_style](#) portion of the force field on the 1st partition. This include the [pair style](#), [bond style](#), [neighbor list building](#), [fixes](#) including time intergration, and

output. The kspace_style portion of the calculation is performed on the 2nd partition.

This is most useful for the PPPM kspace_style when its performance on a large number of processors degrades due to the cost of communication in its 3d FFTs. In this scenario, splitting your P total processors into 2 subsets of processors, P1 in the 1st partition and P2 in the 2nd partition, can enable your simulation to run faster. This is because the long-range forces in PPPM can be calculated at the same time as pair-wise and bonded forces are being calculated, and the FFTs can actually speed up when running on fewer processors.

To use this style, you must define 2 partitions where P1 is a multiple of P2. Typically having P1 be 3x larger than P2 is a good choice. The 3d processor layouts in each partition must overlay in the following sense. If P1 is a Px1 by Py1 by Pz1 grid, and P2 = Px2 by Py2 by Pz2, then Px1 must be an integer multiple of Px2, and similarly for Py1 a multiple of Py2, and Pz1 a multiple of Pz2.

Typically the best way to do this is to let the 1st partition choose its onn optimal layout, then require the 2nd partition's layout to match the integer multiple constraint. See the processors command with its *part* keyword for a way to control this, e.g.

```
procssors * * * part 1 2 multiple
```

You can also use the partition command to explicitly specity the processor layout on each partition. E.g. for 2 partitions of 60 and 15 processors each:

```
partition yes 1 processors 3 4 5
partition yes 2 processors 3 1 5
```

When you run in 2-partition mode with the *verlet/split* style, the thermodyanmic data for the entire simulation will be output to the log and screen file of the 1st partition, which are log.lammps.0 and screen.0 by default; see the "-plog and -pscreen command-line switches"Section_start.html#start_7 to change this. The log and screen file for the 2nd partition will not contain thermodynamic output beyone the 1st timestep of the run.

See Section_accelerate of the manual for performance details of the speed-up offered by the *verlet/split* style. One important performance consideration is the assignemnt of logical processors in the 2 partitions to the physical cores of a parallel machine. The processors command has options to support this, and strategies are discussed in Section_accelerate of the manual.

The *respa* style implements the rRESPA multi-timescale integrator [(Tuckerman)](Tuckerman) with N hierarchical levels, where level 1 is the innermost loop (shortest timestep) and level N is the outermost loop (largest timestep). The loop factor arguments specify what the looping factor is between levels. N1 specifies the number of iterations of level 1 for a single iteration of level 2, N2 is the iterations of level 2 per iteration of level 3, etc. N-1 looping parameters must be specified.

The timestep command sets the timestep for the outermost rRESPA level. Thus if the example command above for a 4-level rRESPA had an outer timestep of 4.0 fmsec, the inner timestep would be 8x smaller or 0.5 fmsec. All other LAMMPS commands that specify number of timesteps (e.g. neigh_modify parameters, dump every N timesteps, etc) refer to the outermost timesteps.

The rRESPA keywords enable you to specify at what level of the hierarchy various forces will be computed. If not specified, the defaults are that bond forces are computed at level 1 (innermost loop), angle forces are computed where bond forces are, dihedral forces are computed where angle forces are, improper forces are computed where dihedral forces are, pair forces are computed at the outermost level, and kspace forces are computed where pair forces are. The inner, middle, outer forces have no defaults.

The *inner* and *middle* keywords take additional arguments for cutoffs that are used by the pairwise force computations. If the 2 cutoffs for *inner* are 5.0 and 6.0, this means that all pairs up to 6.0 apart are computed by the inner force. Those between 5.0 and 6.0 have their force go ramped to 0.0 so the overlap with the next

regime (middle or outer) is smooth. The next regime (middle or outer) will compute forces for all pairs from 5.0 outward, with those from 5.0 to 6.0 having their value ramped in an inverse manner.

Only some pair potentials support the use of the *inner* and *middle* and *outer* keywords. If not, only the *pair* keyword can be used with that pair style, meaning all pairwise forces are computed at the same rRESPA level. See the doc pages for individual pair styles for details.

When using rRESPA (or for any MD simulation) care must be taken to choose a timestep size(s) that insures the Hamiltonian for the chosen ensemble is conserved. For the constant NVE ensemble, total energy must be conserved. Unfortunately, it is difficult to know *a priori* how well energy will be conserved, and a fairly long test simulation (~10 ps) is usually necessary in order to verify that no long-term drift in energy occurs with the trial set of parameters.

With that caveat, a few rules-of-thumb may be useful in selecting *respa* settings. The following applies mostly to biomolecular simulations using the CHARMM or a similar all-atom force field, but the concepts are adaptable to other problems. Without SHAKE, bonds involving hydrogen atoms exhibit high-frequency vibrations and require a timestep on the order of 0.5 fmsec in order to conserve energy. The relatively inexpensive force computations for the bonds, angles, impropers, and dihedrals can be computed on this innermost 0.5 fmsec step. The outermost timestep cannot be greater than 4.0 fmsec without risking energy drift. Smooth switching of forces between the levels of the rRESPA hierarchy is also necessary to avoid drift, and a 1-2 angstrom "healing distance" (the distance between the outer and inner cutoffs) works reasonably well. We thus recommend the following settings for use of the *respa* style without SHAKE in biomolecular simulations:

```
timestep  4.0
run_style respa 4 2 2 2 inner 2 4.5 6.0 middle 3 8.0 10.0 outer 4
```

With these settings, users can expect good energy conservation and roughly a 2.5 fold speedup over the *verlet* style with a 0.5 fmsec timestep.

If SHAKE is used with the *respa* style, time reversibility is lost, but substantially longer time steps can be achieved. For biomolecular simulations using the CHARMM or similar all-atom force field, bonds involving hydrogen atoms exhibit high frequency vibrations and require a time step on the order of 0.5 fmsec in order to conserve energy. These high frequency modes also limit the outer time step sizes since the modes are coupled. It is therefore desirable to use SHAKE with respa in order to freeze out these high frequency motions and increase the size of the time steps in the respa hierarchy. The following settings can be used for biomolecular simulations with SHAKE and rRESPA:

```
fix            2 all shake 0.000001 500 0 m 1.0 a 1
timestep       4.0
run_style      respa 2 2 inner 1 4.0 5.0 outer 2
```

With these settings, users can expect good energy conservation and roughly a 1.5 fold speedup over the *verlet* style with SHAKE and a 2.0 fmsec timestep.

For non-biomolecular simulations, the *respa* style can be advantageous if there is a clear separation of time scales - fast and slow modes in the simulation. Even a LJ system can benefit from rRESPA if the interactions are divided by the inner, middle and outer keywords. A 2-fold or more speedup can be obtained while maintaining good energy conservation. In real units, for a pure LJ fluid at liquid density, with a sigma of 3.0 angstroms, and epsilon of 0.1 Kcal/mol, the following settings seem to work well:

```
timestep  36.0
run_style respa 3 3 4 inner 1 3.0 4.0 middle 2 6.0 7.0 outer 3
```

The *respa/omp* styles is a variant of *respa* adapted for use with pair, bond, angle, dihedral, improper, or kspace styles with an *omp* suffix. It is functionally to *respa* but performs additional required operations. For

more on *omp* styles see the [Section_accelerate](#) of the manual. Accelerated styles take the same arguments and should produce the same results, except for round-off and precision issues.

The *respa/omp* style is part of the USER-OMP packages. It is only enabled if LAMMPS was built with this package included. See the [Making LAMMPS](#) section for more info.

You can specify *respa/omp* explicitly in your input script, or you can use the [-suffix command-line switch](#) when you invoke LAMMPS, or you can use the [suffix](#) command in your input script.

See [Section_accelerate](#) of the manual for more instructions on how to use the accelerated styles effectively.

Restrictions:

The *verlet/split* style can only be used if LAMMPS was built with the REPLICA package. See the [Making LAMMPS](#) section for more info on packages.

Whenever using rRESPA, the user should experiment with trade-offs in speed and accuracy for their system, and verify that they are conserving energy to adequate precision.

Related commands:

[timestep](#), [run](#)

Default:

```
run_style verlet
```

(Tuckerman) Tuckerman, Berne and Martyna, J Chem Phys, 97, p 1990 (1992).

5. Accelerating LAMMPS performance

This section describes various methods for improving LAMMPS performance for different classes of problems running on different kinds of machines.

5.1 Measuring performance

Before trying to make your simulation run faster, you should understand how it currently performs and where the bottlenecks are.

The best way to do this is run the your system (actual number of atoms) for a modest number of timesteps (say 100, or a few 100 at most) on several different processor counts, including a single processor if possible. Do this for an equilibrium version of your system, so that the 100-step timings are representative of a much longer run. There is typically no need to run for 1000s or timesteps to get accurate timings; you can simply extrapolate from short runs.

For the set of runs, look at the timing data printed to the screen and log file at the end of each LAMMPS run. This section of the manual has an overview.

Running on one (or a few processors) should give a good estimate of the serial performance and what portions of the timestep are taking the most time. Running the same problem on a few different processor counts should give an estimate of parallel scalability. I.e. if the simulation runs 16x faster on 16 processors, its 100% parallel efficient; if it runs 8x faster on 16 processors, it's 50% efficient.

The most important data to look at in the timing info is the timing breakdown and relative percentages. For example, trying different options for speeding up the long-range solvers will have little impact if they only consume 10% of the run time. If the pairwise time is dominating, you may want to look at GPU or OMP versions of the pair style, as discussed below. Comparing how the percentages change as you increase the processor count gives you a sense of how different operations within the timestep are scaling. Note that if you are running with a Kspace solver, there is additional output on the breakdown of the Kspace time. For PPPM, this includes the fraction spent on FFTs, which can be communication intensive.

Another important detail in the timing info are the histograms of atoms counts and neighbor counts. If these vary widely across processors, you have a load-imbalance issue. This often results in inaccurate relative timing data, because processors have to wait when communication occurs for other processors to catch up. Thus the reported times for "Communication" or "Other" may be higher than they really are, due to load-imbalance. If this is an issue, you can uncomment the MPI_Barrier() lines in src/timer.cpp, and recompile LAMMPS, to obtain synchronized timings.

5.2 General strategies

NOTE: this sub-section is still a work in progress

Here is a list of general ideas for improving simulation performance. Most of them are only applicable to certain models and certain bottlenecks in the current performance, so let the timing data you intially generate be your guide. It is hard, if not impossible, to predict how much difference these options will make, since it is a function of your problem and your machine. There is no substitute for simply trying them out.

- rRESPA
- 2-FFT PPPM
- Staggered PPPM
- single vs double PPPM
- partial charge PPPM
- verlet/split
- processor mapping via processors numa command
- load-balancing: balance and fix balance
- processor command for layout
- OMP when lots of cores

2-FFT PPPM, also called *analytic differentiation* or *ad* PPPM, uses 2 FFTs instead of the 4 FFTs used by the default *ik differentiation* PPPM. However, 2-FFT PPPM also requires a slightly larger mesh size to achieve the same accuracy as 4-FFT PPPM. For problems where the FFT cost is the performance bottleneck (typically large problems running on many processors), 2-FFT PPPM may be faster than 4-FFT PPPM.

Staggered PPPM performs calculations using two different meshes, one shifted slightly with respect to the other. This can reduce force aliasing errors and increase the accuracy of the method, but also doubles the amount of work required. For high relative accuracy, using staggered PPPM allows one to half the mesh size in each dimension as compared to regular PPPM, which can give around a 4x speedup in the kspace time. However, for low relative accuracy, using staggered PPPM gives little benefit and can be up to 2x slower in the kspace time. For example, the rhodopsin benchmark was run on a single processor, and results for kspace time vs. relative accuracy for the different methods are shown in the figure below. For this system, staggered PPPM (using ik differentiation) becomes useful when using a relative accuracy of slightly greater than 1e-5 and above.
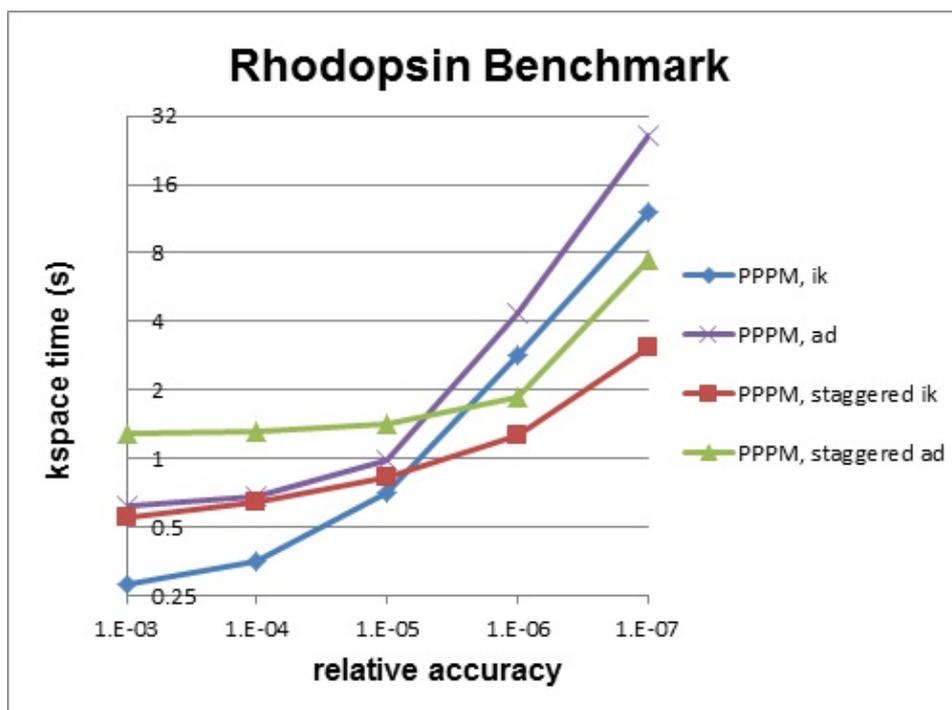
IMPORTANT NOTE: Using staggered PPPM may not give the same increase in accuracy of energy and pressure as it does in forces, so some caution must be used if energy and/or pressure are quantities of interest, such as when using a barostat.

5.3 Packages with optimized styles

Accelerated versions of various pair_style, fixes, computes, and other commands have been added to LAMMPS, which will typically run faster than the standard non-accelerated versions, if you have the appropriate hardware on your system.

The accelerated styles have the same name as the standard styles, except that a suffix is appended. Otherwise, the syntax for the command is identical, their functionality is the same, and the numerical results it produces should also be identical, except for precision and round-off issues.

For example, all of these variants of the basic Lennard-Jones pair style exist in LAMMPS:

- pair_style lj/cut
- pair_style lj/cut/opt
- pair_style lj/cut/omp
- pair_style lj/cut/gpu
- pair_style lj/cut/cuda

Assuming you have built LAMMPS with the appropriate package, these styles can be invoked by specifying them explicitly in your input script. Or you can use the -suffix command-line switch to invoke the accelerated versions automatically, without changing your input script. The suffix command allows you to set a suffix explicitly and to turn off/on the comand-line switch setting, both from within your input script.

Styles with an "opt" suffix are part of the OPT package and typically speed-up the pairwise calculations of your simulation by 5-25%.

Styles with an "omp" suffix are part of the USER-OMP package and allow a pair-style to be run in multi-threaded mode using OpenMP. This can be useful on nodes with high-core counts when using less MPI processes than cores is advantageous, e.g. when running with PPPM so that FFTs are run on fewer MPI

processors or when the many MPI tasks would overload the available bandwidth for communication.

Styles with a "gpu" or "cuda" suffix are part of the GPU or USER-CUDA packages, and can be run on NVIDIA GPUs associated with your CPUs. The speed-up due to GPU usage depends on a variety of factors, as discussed below.

To see what styles are currently available in each of the accelerated packages, see Section_commands 5 of the manual. A list of accelerated styles is included in the pair, fix, compute, and kspace sections.

The following sections explain:

- what hardware and software the accelerated styles require
- how to build LAMMPS with the accelerated packages in place
- what changes (if any) are needed in your input scripts
- guidelines for best performance
- speed-ups you can expect

The final section compares and contrasts the GPU and USER-CUDA packages, since they are both designed to use NVIDIA GPU hardware.

5.4 OPT package

The OPT package was developed by James Fischer (High Performance Technologies), David Richie, and Vincent Natoli (Stone Ridge Technologies). It contains a handful of pair styles whose compute() methods were rewritten in C++ templated form to reduce the overhead due to if tests and other conditional code.

The procedure for building LAMMPS with the OPT package is simple. It is the same as for any other package which has no additional library dependencies:

```
make yes-opt
make machine
```

If your input script uses one of the OPT pair styles, you can run it as follows:

```
lmp_machine -sf opt <in.script
mpirun -np 4 lmp_machine -sf opt  <in.script
```

You should see a reduction in the "Pair time" printed out at the end of the run. On most machines and problems, this will typically be a 5 to 20% savings.

5.5 USER-OMP package

The USER-OMP package was developed by Axel Kohlmeyer at Temple University. It provides multi-threaded versions of most pair styles, all dihedral styles and a few fixes in LAMMPS. The package currently uses the OpenMP interface which requires using a specific compiler flag in the makefile to enable multiple threads; without this flag the corresponding pair styles will still be compiled and work, but do not support multi-threading.

Building LAMMPS with the USER-OMP package:

The procedure for building LAMMPS with the USER-OMP package is simple. You have to edit your machine specific makefile to add the flag to enable OpenMP support to the CCFLAGS and LINKFLAGS variables. For the GNU compilers for example this flag is called *-fopenmp*. Check your compiler documentation to find out which flag you need to add. The rest of the compilation is the same as for any other package which has no additional library dependencies:

```
make yes-user-omp
make machine
```

Please note that this will only install accelerated versions of styles that are already installed, so you want to install this package as the last package, or else you may be missing some accelerated styles. If you plan to uninstall some package, you should first uninstall the USER-OMP package then the other package and then re-install USER-OMP, to make sure that there are no orphaned *omp* style files present, which would lead to compilation errors.

If your input script uses one of regular styles that are also exist as an OpenMP version in the USER-OMP package you can run it as follows:

```
env OMP_NUM_THREADS=4 lmp_serial -sf omp -in in.script
env OMP_NUM_THREADS=2 mpirun -np 2 lmp_machine -sf omp -in in.script
mpirun -x OMP_NUM_THREADS=2 -np 2 lmp_machine -sf omp -in in.script
```

The value of the environment variable OMP_NUM_THREADS determines how many threads per MPI task are launched. All three examples above use a total of 4 CPU cores. For different MPI implementations the method to pass the OMP_NUM_THREADS environment variable to all processes is different. Two different variants, one for MPICH and OpenMPI, respectively are shown above. Please check the documentation of your MPI installation for additional details. Alternatively, the value provided by OMP_NUM_THREADS can be overridded with the [package omp](#) command. Depending on which styles are accelerated in your input, you should see a reduction in the "Pair time" and/or "Bond time" and "Loop time" printed out at the end of the run. The optimal ratio of MPI to OpenMP can vary a lot and should always be confirmed through some benchmark runs for the current system and on the current machine.

Restrictions:

None of the pair styles in the USER-OMP package support the "inner", "middle", "outer" options for r-RESPA integration, only the "pair" option is supported.

Parallel efficiency and performance tips:

In most simple cases the MPI parallelization in LAMMPS is more efficient than multi-threading implemented in the USER-OMP package. Also the parallel efficiency varies between individual styles. On the other hand, in many cases you still want to use the *omp* version - even when compiling or running without OpenMP support - since they all contain optimizations similar to those in the OPT package, which can result in serial speedup.

Using multi-threading is most effective under the following circumstances:

- Individual compute nodes have a significant number of CPU cores but the CPU itself has limited memory bandwidth, e.g. Intel Xeon 53xx (Clovertown) and 54xx (Harpertown) quad core processors. Running one MPI task per CPU core will result in significant performance degradation, so that running with 4 or even only 2 MPI tasks per nodes is faster. Running in hybrid MPI+OpenMP mode will reduce the inter-node communication bandwidth contention in the same way, but offers and additional speedup from utilizing the otherwise idle CPU cores.
- The interconnect used for MPI communication is not able to provide sufficient bandwidth for a large number of MPI tasks per node. This applies for example to running over gigabit ethernet or on Cray XT4 or XT5 series supercomputers. Same as in the aforementioned case this effect worsens with using an increasing number of nodes.
- The input is a system that has an inhomogeneous particle density which cannot be mapped well to the domain decomposition scheme that LAMMPS employs. While this can be to some degree alleviated through using the [processors](#) keyword, multi-threading provides a parallelism that parallelizes over the number of particles not their distribution in space.

- Finally, multi-threaded styles can improve performance when running LAMMPS in "capability mode", i.e. near the point where the MPI parallelism scales out. This can happen in particular when using as kspace style for long-range electrostatics. Here the scaling of the kspace style is the performance limiting factor and using multi-threaded styles allows to operate the kspace style at the limit of scaling and then increase performance parallelizing the real space calculations with hybrid MPI+OpenMP. Sometimes additional speedup can be achived by increasing the real-space coulomb cutoff and thus reducing the work in the kspace part.

The best parallel efficiency from *omp* styles is typically achieved when there is at least one MPI task per physical processor, i.e. socket or die.

Using threads on hyper-threading enabled cores is usually counterproductive, as the cost in additional memory bandwidth requirements is not offset by the gain in CPU utilization through hyper-threading.

A description of the multi-threading strategy and some performance examples are presented here

5.6 GPU package

The GPU package was developed by Mike Brown at ORNL. It provides GPU versions of several pair styles and for long-range Coulombics via the PPPM command. It has the following features:

- The package is designed to exploit common GPU hardware configurations where one or more GPUs are coupled with many cores of a multi-core CPUs, e.g. within a node of a parallel machine.
- Atom-based data (e.g. coordinates, forces) moves back-and-forth between the CPU(s) and GPU every timestep.
- Neighbor lists can be constructed on the CPU or on the GPU
- The charge assignement and force interpolation portions of PPPM can be run on the GPU. The FFT portion, which requires MPI communication between processors, runs on the CPU.
- Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
- LAMMPS-specific code is in the GPU package. It makes calls to a generic GPU library in the lib/gpu directory. This library provides NVIDIA support as well as more general OpenCL support, so that the same functionality can eventually be supported on a variety of GPU hardware.

NOTE: discuss 3 precisions if change, also have to re-link with LAMMPS always use newton off expt with differing numbers of CPUs vs GPU - can't tell what is fastest give command line switches in examples

I am not very clear to the meaning of "Max Mem / Proc" in the "GPU Time Info (average)". Is it the maximal of GPU memory used by one CPU core?

It is the maximum memory used at one time on the GPU for data storage by a single MPI process. - Mike

Hardware and software requirements:

To use this package, you currently need to have specific NVIDIA hardware and install specific NVIDIA CUDA software on your system:

- Check if you have an NVIDIA card: cat /proc/driver/nvidia/cards/0
- Go to http://www.nvidia.com/object/cuda_get.html
- Install a driver and toolkit appropriate for your system (SDK is not necessary)
- Follow the instructions in lammps/lib/gpu/README to build the library (see below)
- Run lammps/lib/gpu/nvc_get_devices to list supported devices and properties

Building LAMMPS with the GPU package:

As with other packages that include a separately compiled library, you need to first build the GPU library, before building LAMMPS itself. General instructions for doing this are in this section of the manual. For this package, do the following, using a Makefile in lib/gpu appropriate for your system:

```
cd lammps/lib/gpu
make -f Makefile.linux
(see further instructions in lammps/lib/gpu/README)
```

If you are successful, you will produce the file lib/libgpu.a.

Now you are ready to build LAMMPS with the GPU package installed:

```
cd lammps/src
make yes-gpu
make machine
```

Note that the lo-level Makefile (e.g. src/MAKE/Makefile.linux) has these settings: gpu_SYSINC, gpu_SYSLIB, gpu_SYSPATH. These need to be set appropriately to include the paths and settings for the CUDA system software on your machine. See src/MAKE/Makefile.g++ for an example.

GPU configuration

When using GPUs, you are restricted to one physical GPU per LAMMPS process, which is an MPI process running on a single core or processor. Multiple MPI processes (CPU cores) can share a single GPU, and in many cases it will be more efficient to run this way.

Input script requirements:

Additional input script requirements to run pair or PPPM styles with a *gpu* suffix are as follows:

- To invoke specific styles from the GPU package, you can either append "gpu" to the style name (e.g. pair_style lj/cut/gpu), or use the -suffix command-line switch, or use the suffix command.
- The newton pair setting must be *off*.
- The package gpu command must be used near the beginning of your script to control the GPU selection and initialization settings. It also has an option to enable asynchronous splitting of force computations between the CPUs and GPUs.

As an example, if you have two GPUs per node and 8 CPU cores per node, and would like to run on 4 nodes (32 cores) with dynamic balancing of force calculation across CPU and GPU cores, you could specify

```
package gpu force/neigh 0 1 -1
```

In this case, all CPU cores and GPU devices on the nodes would be utilized. Each GPU device would be shared by 4 CPU cores. The CPU cores would perform force calculations for some fraction of the particles at the same time the GPUs performed force calculation for the other particles.

Timing output:

As described by the package gpu command, GPU accelerated pair styles can perform computations asynchronously with CPU computations. The "Pair" time reported by LAMMPS will be the maximum of the time required to complete the CPU pair style computations and the time required to complete the GPU pair style computations. Any time spent for GPU-enabled pair styles for computations that run simultaneously with bond, angle, dihedral, improper, and long-range calculations will not be included in the "Pair" time.

When the *mode* setting for the package gpu command is force/neigh, the time for neighbor list calculations on the GPU will be added into the "Pair" time, not the "Neigh" time. An additional breakdown of the times

required for various tasks on the GPU (data copy, neighbor calculations, force computations, etc) are output only with the LAMMPS screen output (not in the log file) at the end of each run. These timings represent total time spent on the GPU for each routine, regardless of asynchronous CPU calculations.

Performance tips:

Generally speaking, for best performance, you should use multiple CPUs per GPU, as provided my most multi-core CPU/GPU configurations.

Because of the large number of cores within each GPU device, it may be more efficient to run on fewer processes per GPU when the number of particles per MPI process is small (100's of particles); this can be necessary to keep the GPU cores busy.

See the lammps/lib/gpu/README file for instructions on how to build the GPU library for single, mixed, or double precision. The latter requires that your GPU card support double precision.

5.7 USER-CUDA package

The USER-CUDA package was developed by Christian Trott at U Technology Ilmenau in Germany. It provides NVIDIA GPU versions of many pair styles, many fixes, a few computes, and for long-range Coulombics via the PPPM command. It has the following features:

- The package is designed to allow an entire LAMMPS calculation, for many timesteps, to run entirely on the GPU (except for inter-processor MPI communication), so that atom-based data (e.g. coordinates, forces) do not have to move back-and-forth between the CPU and GPU.
- The speed-up advantage of this approach is typically better when the number of atoms per GPU is large
- Data will stay on the GPU until a timestep where a non-GPU-ized fix or compute is invoked. Whenever a non-GPU operation occurs (fix, compute, output), data automatically moves back to the CPU as needed. This may incur a performance penalty, but should otherwise work transparently.
- Neighbor lists for GPU-ized pair styles are constructed on the GPU.
- The package only supports use of a single CPU (core) with each GPU.

Hardware and software requirements:

To use this package, you need to have specific NVIDIA hardware and install specific NVIDIA CUDA software on your system.

Your NVIDIA GPU needs to support Compute Capability 1.3. This list may help you to find out the Compute Capability of your card:

http://en.wikipedia.org/wiki/Comparison_of_Nvidia_graphics_processing_units

Install the Nvidia Cuda Toolkit in version 3.2 or higher and the corresponding GPU drivers. The Nvidia Cuda SDK is not required for LAMMPSCUDA but we recommend it be installed. You can then make sure that its sample projects can be compiled without problems.

Building LAMMPS with the USER-CUDA package:

As with other packages that include a separately compiled library, you need to first build the USER-CUDA library, before building LAMMPS itself. General instructions for doing this are in this section of the manual. For this package, do the following, using settings in the lib/cuda Makefiles appropriate for your system:

- Go to the lammps/lib/cuda directory

- If your *CUDA* toolkit is not installed in the default system directoy */usr/local/cuda* edit the file *lib/cuda/Makefile.common* accordingly.
- Type "make OPTIONS", where *OPTIONS* are one or more of the following options. The settings will be written to the *lib/cuda/Makefile.defaults* and used in the next step.

```
precision=N to set the precision level
  N = 1 for single precision (default)
  N = 2 for double precision
  N = 3 for positions in double precision
  N = 4 for positions and velocities in double precision
arch=M to set GPU compute capability
  M = 20 for CC2.0 (GF100/110, e.g. C2050,GTX580,GTX470) (default)
  M = 21 for CC2.1 (GF104/114,  e.g. GTX560, GTX460, GTX450)
  M = 13 for CC1.3 (GF200, e.g. C1060, GTX285)
prec_timer=0/1 to use hi-precision timers
  0 = do not use them (default)
  1 = use these timers
  this is usually only useful for Mac machines
dbg=0/1 to activate debug mode
  0 = no debug mode (default)
  1 = yes debug mode
  this is only useful for developers
cufft=1 to determine usage of CUDA FFT library
  0 = no CUFFT support (default)
  in the future other CUDA-enabled FFT libraries might be supported
```

- Type "make" to build the library. If you are successful, you will produce the file lib/libcuda.a.

Now you are ready to build LAMMPS with the USER-CUDA package installed:

```
cd lammps/src
make yes-user-cuda
make machine
```

Note that the LAMMPS build references the lib/cuda/Makefile.common file to extract setting specific CUDA settings. So it is important that you have first built the cuda library (in lib/cuda) using settings appropriate to your system.

Input script requirements:

Additional input script requirements to run styles with a *cuda* suffix are as follows:

- To invoke specific styles from the USER-CUDA package, you can either append "cuda" to the style name (e.g. pair_style lj/cut/cuda), or use the -suffix command-line switch, or use the suffix command. One exception is that the kspace_style pppm/cuda command has to be requested explicitly.
- To use the USER-CUDA package with its default settings, no additional command is needed in your input script. This is because when LAMMPS starts up, it detects if it has been built with the USER-CUDA package. See the -cuda command-line switch for more details.
- To change settings for the USER-CUDA package at run-time, the package cuda command can be used near the beginning of your input script. See the package command doc page for details.

Performance tips:

The USER-CUDA package offers more speed-up relative to CPU performance when the number of atoms per GPU is large, e.g. on the order of tens or hundreds of 1000s.

As noted above, this package will continue to run a simulation entirely on the GPU(s) (except for inter-processor MPI communication), for multiple timesteps, until a CPU calculation is required, either by a fix or compute that is non-GPU-ized, or until output is performed (thermo or dump snapshot or restart file). The less often this occurs, the faster your simulation will run.

5.8 Comparison of GPU and USER-CUDA packages

Both the GPU and USER-CUDA packages accelerate a LAMMPS calculation using NVIDIA hardware, but they do it in different ways.

As a consequence, for a particular simulation on specific hardware, one package may be faster than the other. We give guidelines below, but the best way to determine which package is faster for your input script is to try both of them on your machine. See the benchmarking section below for examples where this has been done.

Guidelines for using each package optimally:

- The GPU package allows you to assign multiple CPUs (cores) to a single GPU (a common configuration for "hybrid" nodes that contain multicore CPU(s) and GPU(s)) and works effectively in this mode. The USER-CUDA package does not allow this; you can only use one CPU per GPU.
- The GPU package moves per-atom data (coordinates, forces) back-and-forth between the CPU and GPU every timestep. The USER-CUDA package only does this on timesteps when a CPU calculation is required (e.g. to invoke a fix or compute that is non-GPU-ized). Hence, if you can formulate your input script to only use GPU-ized fixes and computes, and avoid doing I/O too often (thermo output, dump file snapshots, restart files), then the data transfer cost of the USER-CUDA package can be very low, causing it to run faster than the GPU package.
- The GPU package is often faster than the USER-CUDA package, if the number of atoms per GPU is "small". The crossover point, in terms of atoms/GPU at which the USER-CUDA package becomes faster depends strongly on the pair style. For example, for a simple Lennard Jones system the crossover (in single precision) is often about 50K-100K atoms per GPU. When performing double precision calculations the crossover point can be significantly smaller.
- Both packages compute bonded interactions (bonds, angles, etc) on the CPU. This means a model with bonds will force the USER-CUDA package to transfer per-atom data back-and-forth between the CPU and GPU every timestep. If the GPU package is running with several MPI processes assigned to one GPU, the cost of computing the bonded interactions is spread across more CPUs and hence the GPU package can run faster.
- When using the GPU package with multiple CPUs assigned to one GPU, its performance depends to some extent on high bandwidth between the CPUs and the GPU. Hence its performance is affected if full 16 PCIe lanes are not available for each GPU. In HPC environments this can be the case if S2050/70 servers are used, where two devices generally share one PCIe 2.0 16x slot. Also many multi-GPU mainboards do not provide full 16 lanes to each of the PCIe 2.0 16x slots.

Differences between the two packages:

- The GPU package accelerates only pair force, neighbor list, and PPPM calculations. The USER-CUDA package currently supports a wider range of pair styles and can also accelerate many fix styles and some compute styles, as well as neighbor list and PPPM calculations.
- The USER-CUDA package does not support acceleration for minimization.
- The USER-CUDA package does not support hybrid pair styles.
- The USER-CUDA package can order atoms in the neighbor list differently from run to run resulting in a different order for force accumulation.
- The USER-CUDA package has a limit on the number of atom types that can be used in a simulation.
- The GPU package requires neighbor lists to be built on the CPU when using exclusion lists or a triclinic simulation box.
- The GPU package uses more GPU memory than the USER-CUDA package. This is generally not a problem since typical runs are computation-limited rather than memory-limited.

Examples:

The LAMMPS distribution has two directories with sample input scripts for the GPU and USER-CUDA packages.

- lammps/examples/gpu = GPU package files
- lammps/examples/USER/cuda = USER-CUDA package files

These contain input scripts for identical systems, so they can be used to benchmark the performance of both packages on your system.

3. Commands

This section describes how a LAMMPS input script is formatted and the input script commands used to define a LAMMPS simulation.

3.1 LAMMPS input script

LAMMPS executes by reading commands from a input script (text file), one line at a time. When the input script ends, LAMMPS exits. Each command causes LAMMPS to take some action. It may set an internal variable, read in a file, or run a simulation. Most commands have default settings, which means you only need to use the command if you wish to change the default.

In many cases, the ordering of commands in an input script is not important. However the following rules apply:

(1) LAMMPS does not read your entire input script and then perform a simulation with all the settings. Rather, the input script is read one line at a time and each command takes effect when it is read. Thus this sequence of commands:

```
timestep 0.5
run     100
run     100
```

does something different than this sequence:

```
run     100
timestep 0.5
run     100
```

In the first case, the specified timestep (0.5 fmsec) is used for two simulations of 100 timesteps each. In the 2nd case, the default timestep (1.0 fmsec) is used for the 1st 100 step simulation and a 0.5 fmsec timestep is used for the 2nd one.

(2) Some commands are only valid when they follow other commands. For example you cannot set the temperature of a group of atoms until atoms have been defined and a group command is used to define which atoms belong to the group.

(3) Sometimes command B will use values that can be set by command A. This means command A must precede command B in the input script if it is to have the desired effect. For example, the read_data command initializes the system by setting up the simulation box and assigning atoms to processors. If default values are not desired, the processors and boundary commands need to be used before read_data to tell LAMMPS how to map processors to the simulation box.

Many input script errors are detected by LAMMPS and an ERROR or WARNING message is printed. This section gives more information on what errors mean. The documentation for each command lists restrictions

on how the command can be used.

3.2 Parsing rules

Each non-blank line in the input script is treated as a command. LAMMPS commands are case sensitive. Command names are lower-case, as are specified command arguments. Upper case letters may be used in file names or user-chosen ID strings.

Here is how each line in the input script is parsed by LAMMPS:

(1) If the last printable character on the line is a "&" character (with no surrounding quotes), the command is assumed to continue on the next line. The next line is concatenated to the previous line by removing the "&" character and newline. This allows long commands to be continued across two or more lines.

(2) All characters from the first "#" character onward are treated as comment and discarded. See an exception in (6). Note that a comment after a trailing "&" character will prevent the command from continuing on the next line. Also note that for multi-line commands a single leading "#" will comment out the entire command.

(3) The line is searched repeatedly for $ characters, which indicate variables that are replaced with a text string. See an exception in (6).

If the $ is followed by curly brackets, then the variable name is the text inside the curly brackets. If no curly brackets follow the $, then the variable name is the single character immediately following the $. Thus ${myTemp} and $x refer to variable names "myTemp" and "x".

If the $ is followed by parenthesis, then the text inside the parenthesis is treated as an "immediate" variable and evaluated as an [equal-style variable](). This is a way to use numeric formulas in an input script without having to assign them to variable names. For example, these 3 input script lines:

```
variable X equal (xlo+xhi)/2+sqrt(v_area)
region 1 block $X 2 INF INF EDGE EDGE
variable X delete
```

can be replaced by

```
region 1 block $((xlo+xhi)/2+sqrt(v_area)) 2 INF INF EDGE EDGE
```

so that you do not have to define (or discard) a temporary variable X.

Note that neither the curly-bracket or immediate form of variables can contain nested $ characters for other variables to substitute for. Thus you cannot do this:

```
variable        a equal 2
variable        b2 equal 4
print           "B2 = ${b$a}"
```

Nor can you specify this $($x-1.0) for an immediate variable, but you could use $(v_x-1.0), since the latter is valid syntax for an [equal-style variable]().

See the [variable]() command for more details of how strings are assigned to variables and evaluated, and how they can be used in input script commands.

(4) The line is broken into "words" separated by whitespace (tabs, spaces). Note that words can thus contain letters, digits, underscores, or punctuation characters.

(5) The first word is the command name. All successive words in the line are arguments.

(6) If you want text with spaces to be treated as a single argument, it can be enclosed in either double or single quotes. E.g.

```
print "Volume = $v"
print 'Volume = $v'
if "$steps > 1000" then quit
```

The quotes are removed when the single argument is stored internally. See the dump modify format or print or if commands for examples. A "#" or "$" character that is between quotes will not be treated as a comment indicator in (2) or substituted for as a variable in (3).

IMPORTANT NOTE: If the argument is itself a command that requires a quoted argument (e.g. using a print command as part of an if or run every command), then the double and single quotes can be nested in the usual manner. See the doc pages for those commands for examples. Only one of level of nesting is allowed, but that should be sufficient for most use cases.

3.3 Input script structure

This section describes the structure of a typical LAMMPS input script. The "examples" directory in the LAMMPS distribution contains many sample input scripts; the corresponding problems are discussed in Section_example, and animated on the LAMMPS WWW Site.

A LAMMPS input script typically has 4 parts:

1. Initialization
2. Atom definition
3. Settings
4. Run a simulation

The last 2 parts can be repeated as many times as desired. I.e. run a simulation, change some settings, run some more, etc. Each of the 4 parts is now described in more detail. Remember that almost all the commands need only be used if a non-default value is desired.

(1) Initialization

Set parameters that need to be defined before atoms are created or read-in from a file.

The relevant commands are units, dimension, newton, processors, boundary, atom_style, atom_modify.

If force-field parameters appear in the files that will be read, these commands tell LAMMPS what kinds of force fields are being used: pair_style, bond_style, angle_style, dihedral_style, improper_style.

(2) Atom definition

There are 3 ways to define atoms in LAMMPS. Read them in from a data or restart file via the read_data or read_restart commands. These files can contain molecular topology information. Or create atoms on a lattice (with no molecular topology), using these commands: lattice, region, create_box, create_atoms. The entire set of atoms can be duplicated to make a larger simulation using the replicate command.

(3) Settings

Once atoms and molecular topology are defined, a variety of settings can be specified: force field coefficients, simulation parameters, output options, etc.

Force field coefficients are set by these commands (they can also be set in the read-in files): pair_coeff, bond_coeff, angle_coeff, dihedral_coeff, improper_coeff, kspace_style, dielectric, special_bonds.

Various simulation parameters are set by these commands: neighbor, neigh_modify, group, timestep, reset_timestep, run_style, min_style, min_modify.

Fixes impose a variety of boundary conditions, time integration, and diagnostic options. The fix command comes in many flavors.

Various computations can be specified for execution during a simulation using the compute, compute_modify, and variable commands.

Output options are set by the thermo, dump, and restart commands.

(4) Run a simulation

A molecular dynamics simulation is run using the run command. Energy minimization (molecular statics) is performed using the minimize command. A parallel tempering (replica-exchange) simulation can be run using the temper command.

3.4 Commands listed by category

This section lists all LAMMPS commands, grouped by category. The next section lists the same commands alphabetically. Note that some style options for some commands are part of specific LAMMPS packages, which means they cannot be used unless the package was included when LAMMPS was built. Not all packages are included in a default LAMMPS build. These dependencies are listed as Restrictions in the command's documentation.

Initialization:

atom_modify, atom_style, boundary, dimension, newton, processors, units

Atom definition:

create_atoms, create_box, lattice, read_data, read_dump, read_restart, region, replicate

Force fields:

angle_coeff, angle_style, bond_coeff, bond_style, dielectric, dihedral_coeff, dihedral_style, improper_coeff, improper_style, kspace_modify, kspace_style, pair_coeff, pair_modify, pair_style, pair_write, special_bonds

Settings:

communicate, group, mass, min_modify, min_style, neigh_modify, neighbor, reset_timestep, run_style, set, timestep, velocity

Fixes:

fix, fix_modify, unfix

Computes:

compute, compute_modify, uncompute

Output:

[dump](), [dump image](), [dump_modify](), [dump movie](), [restart](), [thermo](), [thermo_modify](), [thermo_style](), [undump](), [write_data](), [write_dump](), [write_restart]()

Actions:

[delete_atoms](), [delete_bonds](), [displace_atoms](), [change_box](), [minimize](), [neb]() [prd](), [rerun](), [run](), [temper]()

Miscellaneous:

[clear](), [echo](), [if](), [include](), [jump](), [label](), [log](), [next](), [print](), [shell](), [variable]()

3.5 Individual commands

This section lists all LAMMPS and LIGGGHTS commands alphabetically, with a separate listing below of styles within certain commands. Note that some style options for some commands are part of specific LAMMPS packages, which means they cannot be used unless the package was included when LAMMPS was built. Not all packages are included in a default LAMMPS build. These dependencies are listed as Restrictions in the command's documentation.

angle_coeff	angle_style	atom_modify	atom_style	bond_coeff	bond_style
boundary	box	change_box	clear	communicate	compute
compute_modify	create_atoms	create_box	delete_atoms	delete_bonds	dielectric
dihedral_coeff	dihedral_style	dimension	displace_atoms	dump	dump_modify
echo	fix	fix_modify	group	group2ndx	if
improper_coeff	improper_style	include	jump	kspace_modify	kspace_style
label	lattice	log	mass	min_modify	min_style
minimize	neb	neigh_modify	neighbor	newton	next
package	pair_coeff	pair_modify	pair_style	pair_write	partition
prd	print	processors	quit	read_data	read_dump
read_restart	region	replicate	rerun	reset_timestep	restart
run	run_style	set	shell	special_bonds	suffix
tad	temper	thermo	thermo_modify	thermo_style	timestep
uncompute	undump	unfix	units	variable	velocity
write_data	write_dump	write_restart			

angle_style potentials

See the [angle_style]() command for an overview of angle potentials. Click on the style itself for a full description:

charmm	class2	cosine	cosine/delta
cosine/periodic	cosine/shift	cosine/shift/exp	cosine/squared
dipole	fourier	fourier/simple	harmonic
hybrid	none	quartic	sdk
table			

These are accelerated angle styles, which can be used if LAMMPS is built with the [appropriate accelerated package]().

charmm/omp	class2/omp	cosine/delta/omp	cosine/omp
cosine/periodic/omp	cosine/shift/exp/omp	cosine/shift/omp	cosine/squared/omp
dipole/omp	fourier/omp	fourier/simple/omp	harmonic/omp
quartic/omp	table/omp		

bond_style potentials

See the bond_style command for an overview of bond potentials. Click on the style itself for a full description:

class2	fene	fene/expand	harmonic
harmonic/shift	harmonic/shift/cut	hybrid	morse
none	nonlinear	quartic	table

These are accelerated bond styles, which can be used if LAMMPS is built with the appropriate accelerated package.

class2/omp	fene/expand/omp	fene/omp	harmonic/omp
harmonic/shift/cut/omp	harmonic/shift/omp	morse/omp	nonlinear/omp
quartic/omp	table/omp		

compute syles

See the compute command for one-line descriptions of each style or click on the style itself for a full description:

ackland/atom	angle/local	atom/molecule	basal/atom
body/local	bond/local	centro/atom	cluster/atom
cna/atom	com	com/molecule	contact/atom
coord/atom	damage/atom	dihedral/local	displace/atom
erotate/asphere	erotate/sphere	erotate/sphere/atom	event/displace
group/group	gyration	gyration/molecule	heat/flux
improper/local	inertia/molecule	ke	ke/atom
ke/atom/eff	ke/eff	meso_e/atom	meso_rho/atom
meso_t/atom	msd	msd/molecule	msd/nongauss
nparticles/tracer/region	pair	pair/gran/local	pair/local
pe	pe/atom	pressure	property/atom
property/local	property/molecule	rdf	reduce
reduce/region	slice	stress/atom	temp
temp/asphere	temp/com	temp/deform	temp/deform/eff
temp/eff	temp/partial	temp/profile	temp/ramp
temp/region	temp/region/eff	temp/rotate	temp/sphere
ti	voronoi/atom	wall/gran/local	

These are accelerated compute styles, which can be used if LAMMPS is built with the appropriate accelerated package.

pe/cuda	pressure/cuda	temp/cuda	temp/partial/cuda

angle_style potentials 1035

dihedral_style potentials

See the dihedral_style command for an overview of dihedral potentials. Click on the style itself for a full description:

charmm	class2	cosine/shift/exp	fourier
harmonic	helix	hybrid	multi/harmonic
nharmonic	none	opls	quadratic
table			

These are accelerated dihedral styles, which can be used if LAMMPS is built with the [appropriate accelerated package](appropriate_accelerated_package).

charmm/omp	class2/omp	cosine/shift/exp/omp	fourier/omp
harmonic/omp	helix/omp	multi/harmonic/omp	nharmonic/omp
opls/omp	quadratic/omp	table/omp	

---

dump syles

Click on the style itself for a full description:

image	molfile	movie

These are accelerated styles, which can be used if LAMMPS is built with the [appropriate accelerated package](appropriate_accelerated_package).

fix syles

See the [fix](fix) command for one-line descriptions of each style or click on the style itself for a full description:

adapt	addforce	addtorque	append/atoms
[atc](atc)	ave/atom	ave/correlate	ave/euler
ave/histo	ave/spatial	ave/time	aveforce
bond/break	bond/create	bond/swap	box/relax
check/timestep/gran	colvars	deform	deposit
drag	dt/reset	efield	enforce2d
evaporate	external	freeze	gcmc
[gld](gld)	gravity	heat	heat/gran
heat/gran/conduction	[imd](imd)	indent	insert/pack
insert/rate/region	insert/stream	langevin	langevin/eff
lb/fluid	lb/momentum	[lb/pc](lb/pc)	lb/rigid/pc/sphere
lb/viscous	lineforce	massflow/mesh	mesh/surface
mesh/surface/planar	mesh/surface/stress	mesh/surface/stress/servo	meso
meso/stationary	momentum	move	move/mesh
msst	[neb](neb)	[nph](nph)	nph/asphere
[nph/eff](nph/eff)	nph/sphere	nphug	[npt](npt)
npt/asphere	[npt/eff](npt/eff)	npt/sphere	[nve](nve)
nve/asphere	nve/asphere/noforce	nve/body	[nve/eff](nve/eff)
nve/limit	nve/line	nve/noforce	nve/sphere
[nve/tri](nve/tri)	[nvt](nvt)	nvt/asphere	[nvt/eff](nvt/eff)
nvt/sllod	nvt/sllod/eff	nvt/sphere	orient/fcc

particledistribution/discrete	particletemplate/sphere	phonon	planeforce
pour/legacy	press/berendsen	print	property/atom
property/atom/tracer	property/atom/tracer/stream	property/global	qeq/comb
qeq/reax	reax/bonds	reax/c/bonds	reax/c/species
recenter	restrain	rigid	rigid/nph
rigid/npt	rigid/nve	rigid/nvt	rigid/small
setforce	shake	smd	sph/density/continuity
sph/density/corr	sph/density/summation	sph/pressure	spring
spring/rg	spring/self	srd	store/force
store/state	temp/berendsen	temp/rescale	temp/rescale/eff
thermal/conductivity	ti/rs	ti/spring	tmd
ttm	tune/kspace	viscosity	viscous
wall/colloid	wall/gran	wall/harmonic	wall/lj1043
wall/lj126	wall/lj93	wall/piston	wall/reflect
wall/region	wall/region/sph	wall/srd	

These are accelerated fix styles, which can be used if LAMMPS is built with the appropriate accelerated package.

addforce/cuda	aveforce/cuda	enforce2d/cuda	freeze/cuda
gravity/cuda	gravity/omp	nph/asphere/omp	nph/omp
nph/sphere/omp	nphug/omp	npt/asphere/omp	npt/cuda
npt/omp	npt/sphere/omp	nve/cuda	nve/omp
nve/sphere/omp	nvt/asphere/omp	nvt/cuda	nvt/omp
nvt/sllod/omp	nvt/sphere/omp	qeq/comb/omp	setforce/cuda
shake/cuda	temp/berendsen/cuda	temp/rescale/cuda	temp/rescale/limit/cuda
viscous/cuda			

improper_style potentials

See the improper_style command for an overview of improper potentials. Click on the style itself for a full description:

class2	cossq	cvff	fourier
harmonic	hybrid	none	ring
umbrella			

These are accelerated improper styles, which can be used if LAMMPS is built with the appropriate accelerated package.

class2/omp	cossq/omp	cvff/omp	fourier/omp
harmonic/omp	ring/omp	umbrella/omp	

pair_style potentials

See the pair_style command for an overview of pair potentials. Click on the style itself for a full description:

adp	airebo	awpmd/cut	beck
body	bop	born	born/coul/long

born/coul/msm	born/coul/wolf	brownian	brownian/poly
buck	buck/coul/cut	buck/coul/long	buck/coul/msm
buck/long/coul/long	colloid	comb	comb3
coul/cut	coul/debye	coul/diel	coul/dsf
coul/long	coul/msm	coul/wolf	dpd
dpd/tstat	dsmc	eam	eam/alloy
eam/cd	eam/fs	edip	eff/cut
eim	gauss	gauss/cut	gayberne
gran	hbond/dreiding/lj	hbond/dreiding/morse	hybrid
hybrid/overlay	kim	lcbop	line/lj
list	lj/charmm/coul/charmm	lj/charmm/coul/charmm/implicit	lj/charmm/coul/long
lj/charmm/coul/msm	lj/class2	lj/class2/coul/cut	lj/class2/coul/long
lj/cubic	lj/cut	lj/cut/coul/cut	lj/cut/coul/debye
lj/cut/coul/dsf	lj/cut/coul/long	lj/cut/coul/msm	lj/cut/dipole/cut
lj/cut/dipole/long	lj/cut/tip4p/cut	lj/cut/tip4p/long	lj/expand
lj/gromacs	lj/gromacs/coul/gromacs	lj/long/coul/long	lj/long/dipole/long
lj/long/tip4p/long	lj/sdk	lj/sdk/coul/long	lj/sf
lj/sf/dipole/sf	lj/smooth	lj/smooth/linear	lj96/cut
lubricate	lubricate/poly	lubricateU	lubricateU/poly
meam	mie/cut	morse	nb3b/harmonic
nm/cut	nm/cut/coul/cut	nm/cut/coul/long	none
peri/lps	peri/pmb	peri/ves	reax
reax/c	rebo	resquared	soft
sph	sph/artVisc/tensCorr	sph/heatconduction	sph/idealgas
sph/lj	sph/rhosum	sph/taitwater	sph/taitwater/morris
sw	table	tersoff	tersoff/mod
tersoff/table	tersoff/zbl	tip4p/cut	tip4p/long
tri/lj	yukawa	yukawa/colloid	zbl

These are accelerated pair styles, which can be used if LAMMPS is built with the appropriate accelerated package.

adp/omp	airebo/omp	beck/gpu	beck/omp
born/coul/long/cuda	born/coul/long/gpu	born/coul/long/omp	born/coul/msm/omp
born/coul/wolf/gpu	born/coul/wolf/omp	born/gpu	born/omp
brownian/omp	brownian/poly/omp	buck/coul/cut/cuda	buck/coul/cut/gpu
buck/coul/cut/omp	buck/coul/long/cuda	buck/coul/long/gpu	buck/coul/long/omp
buck/coul/msm/omp	buck/cuda	buck/gpu	buck/long/coul/long/omp
buck/omp	colloid/gpu	colloid/omp	comb/omp
coul/cut/omp	coul/debye/omp	coul/dsf/gpu	coul/dsf/omp
coul/long/gpu	coul/long/omp	coul/msm/omp	coul/wolf/omp
dpd/omp	dpd/tstat/omp	eam/alloy/cuda	eam/alloy/gpu
eam/alloy/omp	eam/alloy/opt	eam/cd/omp	eam/cuda
eam/fs/cuda	eam/fs/gpu	eam/fs/omp	eam/fs/opt
eam/gpu	eam/omp	eam/opt	eim/omp
gauss/cut/omp	gauss/gpu	gauss/omp	gayberne/gpu

gayberne/omp	hbond/dreiding/lj/omp	hbond/dreiding/morse/omp	hybrid/omp
hybrid/overlay/omp	line/lj/omp	lj/charmm/coul/charmm/cuda	lj/charmm/coul/charmm/implicit/
lj/charmm/coul/charmm/implicit/omp	lj/charmm/coul/charmm/omp	lj/charmm/coul/long/cuda	lj/charmm/coul/long/gpu
lj/charmm/coul/long/omp	lj/charmm/coul/long/opt	lj/charmm/coul/msm/omp	lj/class2/coul/cut/cuda
lj/class2/coul/cut/omp	lj/class2/coul/long/cuda	lj/class2/coul/long/gpu	lj/class2/coul/long/omp
lj/class2/cuda	lj/class2/gpu	lj/class2/omp	lj/cubic/omp
lj/cut/coul/cut/cuda	lj/cut/coul/cut/gpu	lj/cut/coul/cut/omp	lj/cut/coul/debye/cuda
lj/cut/coul/debye/gpu	lj/cut/coul/debye/omp	lj/cut/coul/dsf/gpu	lj/cut/coul/dsf/omp
lj/cut/coul/long/cuda	lj/cut/coul/long/gpu	lj/cut/coul/long/omp	lj/cut/coul/long/opt
lj/cut/coul/msm/gpu	lj/cut/coul/msm/omp	lj/cut/cuda	lj/cut/dipole/cut/gpu
lj/cut/dipole/cut/omp	lj/cut/experimental/cuda	lj/cut/gpu	lj/cut/omp
lj/cut/opt	lj/cut/tip4p/cut/omp	lj/cut/tip4p/long/omp	lj/cut/tip4p/long/opt
lj/expand/cuda	lj/expand/gpu	lj/expand/omp	lj/gromacs/coul/gromacs/cud
lj/gromacs/coul/gromacs/omp	lj/gromacs/cuda	lj/gromacs/omp	lj/long/coul/long/omp
lj/long/coul/long/opt	lj/sdk/coul/long/gpu	lj/sdk/coul/long/omp	lj/sdk/gpu
lj/sdk/omp	lj/sf/dipole/sf/gpu	lj/sf/dipole/sf/omp	lj/sf/omp
lj/smooth/cuda	lj/smooth/linear/omp	lj/smooth/omp	lj96/cut/cuda
lj96/cut/gpu	lj96/cut/omp	lubricate/omp	lubricate/poly/omp
mie/cut/gpu	morse/cuda	morse/gpu	morse/omp
morse/opt	nb3b/harmonic/omp	nm/cut/coul/cut/omp	nm/cut/coul/long/omp
nm/cut/omp	peri/lps/omp	peri/pmb/omp	rebo/omp
resquared/gpu	resquared/omp	soft/gpu	soft/omp
sw/cuda	sw/gpu	sw/omp	table/gpu
table/omp	tersoff/mod/omp	tersoff/table/omp	tersoff/zbl/omp
tip4p/cut/omp	tip4p/long/omp	tri/lj/omp	yukawa/colloid/gpu
yukawa/colloid/omp	yukawa/gpu	yukawa/omp	zbl/omp

12. Errors

This section describes the errors you can encounter when using LAMMPS, either conceptually, or as printed out by the program.

12.1 Common problems

If two LAMMPS runs do not produce the same answer on different machines or different numbers of processors, this is typically not a bug. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round-off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories within a few 100s or few 1000s of timesteps. However, the statistical properties of the two runs (e.g. average energy or temperature) should still be the same.

If the velocity command is used to set initial atom velocities, a particular atom can be assigned a different velocity when the problem is run on a different number of processors or on different machines. If this happens, the phase space trajectories of the two simulations will rapidly diverge. See the discussion of the *loop* option in the velocity command for details and options that avoid this issue.

Similarly, the create_atoms command generates a lattice of atoms. For the same physical system, the ordering and numbering of atoms by atom ID may be different depending on the number of processors.

Some commands use random number generators which may be setup to produce different random number streams on each processor and hence will produce different effects when run on different numbers of processors. A commonly-used example is the fix langevin command for thermostatting.

A LAMMPS simulation typically has two stages, setup and run. Most LAMMPS errors are detected at setup time; others like a bond stretching too far may not occur until the middle of a run.

LAMMPS tries to flag errors and print informative error messages so you can fix the problem. Of course, LAMMPS cannot figure out your physics or numerical mistakes, like choosing too big a timestep, specifying erroneous force field coefficients, or putting 2 atoms on top of each other! If you run into errors that LAMMPS doesn't catch that you think it should flag, please send an email to the developers.

If you get an error message about an invalid command in your input script, you can determine what command is causing the problem by looking in the log.lammps file or using the echo command to see it on the screen. For a given command, LAMMPS expects certain arguments in a specified order. If you mess this up, LAMMPS will often flag the error, but it may read a bogus argument and assign a value that is valid, but not what you wanted. E.g. trying to read the string "abc" as an integer value and assigning the associated variable a value of 0.

Generally, LAMMPS will print a message to the screen and logfile and exit gracefully when it encounters a fatal error. Sometimes it will print a WARNING to the screen and logfile and continue on; you can decide if the WARNING is important or not. A WARNING message that is generated in the middle of a run is only printed to the screen, not to the logfile, to avoid cluttering up thermodynamic output. If LAMMPS crashes or hangs without spitting out an error message first then it could be a bug (see this section) or one of the following cases:

LAMMPS runs in the available memory a processor allows to be allocated. Most reasonable MD runs are compute limited, not memory limited, so this shouldn't be a bottleneck on most platforms. Almost all large memory allocations in the code are done via C-style malloc's which will generate an error message if you run out of memory. Smaller chunks of memory are allocated via C++ "new" statements. If you are unlucky you could run out of memory just when one of these small requests is made, in which case the code will crash or hang (in parallel), since LAMMPS doesn't trap on those errors.

Illegal arithmetic can cause LAMMPS to run slow or crash. This is typically due to invalid physics and numerics that your simulation is computing. If you see wild thermodynamic values or NaN values in your LAMMPS output, something is wrong with your simulation. If you suspect this is happening, it is a good idea to print out thermodynamic info frequently (e.g. every timestep) via the thermo so you can monitor what is happening. Visualizing the atom movement is also a good idea to insure your model is behaving as you expect.

In parallel, one way LAMMPS can hang is due to how different MPI implementations handle buffering of messages. If the code hangs without an error message, it may be that you need to specify an MPI setting or two (usually via an environment variable) to enable buffering or boost the sizes of messages that can be buffered.

12.2 Reporting bugs

If you are confident that you have found a bug in LAMMPS, follow these steps.

Check the New features and bug fixes section of the LAMMPS WWW site to see if the bug has already been reported or fixed or the Unfixed bug to see if a fix is pending.

Check the mailing list to see if it has been discussed before.

If not, send an email to the mailing list describing the problem with any ideas you have as to what is causing it or where in the code the problem might be. The developers will ask for more info if needed, such as an input script or data files.

The most useful thing you can do to help us fix the bug is to isolate the problem. Run it on the smallest number of atoms and fewest number of processors and with the simplest input script that reproduces the bug and try to identify what command or combination of commands is causing the problem.

As a last resort, you can send an email directly to the developers.

12.3 Error & warning messages

These are two alphabetic lists of the ERROR and WARNING messages LAMMPS prints out and the reason why. If the explanation here is not sufficient, the documentation for the offending command may help. Error and warning messages also list the source file and line number where the error was generated. For example, this message

ERROR: Illegal velocity command (velocity.cpp:78)

means that line #78 in the file src/velocity.cpp generated the error. Looking in the source code may help you figure out what went wrong.

Note that error messages from user-contributed packages are not listed here. If such an error occurs and is not self-explanatory, you'll need to look in the source code or contact the author of the package.

Errors:

1-3 bond count is inconsistent

An inconsistency was detected when computing the number of 1-3 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

1-4 bond count is inconsistent

An inconsistency was detected when computing the number of 1-4 neighbors for each atom. This likely means something is wrong with the bond topologies you have defined.

Accelerator sharing is not currently supported on system

Multiple MPI processes cannot share the accelerator on your system. For NVIDIA GPUs, see the nvidia-smi command to change this setting.

All angle coeffs are not set

All angle coefficients must be set in the data file or by the angle_coeff command before running a simulation.

All bond coeffs are not set

All bond coefficients must be set in the data file or by the bond_coeff command before running a simulation.

All dihedral coeffs are not set

All dihedral coefficients must be set in the data file or by the dihedral_coeff command before running a simulation.

All improper coeffs are not set

All improper coefficients must be set in the data file or by the improper_coeff command before running a simulation.

All masses are not set

For atom styles that define masses for each atom type, all masses must be set in the data file or by the mass command before running a simulation. They must also be set before using the velocity command.

All mol IDs should be set for fix gcmc group atoms

The molecule flag is on, yet not all molecule ids in the fix group have been set to non-zero positive values by the user. This is an error since all atoms in the fix gcmc group are eligible for deletion, rotation, and translation and therefore must have valid molecule ids.

All pair coeffs are not set

All pair coefficients must be set in the data file or by the pair_coeff command before running a simulation.

All read_dump x,y,z fields must be specified for scaled, triclinic coords

For triclinic boxes and scaled coordinates you must specify all 3 of the x,y,z fields, else LAMMPS cannot reconstruct the unscaled coordinates.

All universe/uloop variables must have same # of values

Self-explanatory.

All variables in next command must be same style

Self-explanatory.

Angle atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atom missing in set command

The set command cannot find one or more atoms in a particular angle on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid angle.

Angle atoms %d %d %d missing on proc %d at step %ld

One or more of 3 atoms needed to compute a particular angle are missing on this processor. Typically this is because the pairwise cutoff is set too short or the angle has blown apart and an atom is too far away.

Angle coeff for hybrid has invalid style

Angle style hybrid uses another angle style as one of its coefficients. The angle style used in the angle_coeff command or read from a restart file is not recognized.

Angle coeffs are not set

No angle coefficients have been assigned in the data file or via the angle_coeff command.

Angle extent > half of periodic box length

This error was detected by the neigh_modify check yes setting. It is an error because the angle atoms are so far apart it is ambiguous how it should be defined.

Angle potential must be defined for SHAKE

When shaking angles, an angle_style potential must be used.

Angle style hybrid cannot have hybrid as an argument

Self-explanatory.

Angle style hybrid cannot have none as an argument

Self-explanatory.

Angle style hybrid cannot use same angle style twice

Self-explanatory.

Angle table must range from 0 to 180 degrees

Self-explanatory.

Angle table parameters did not set N

List of angle table parameters must include N setting.

Angle_coeff command before angle_style is defined

Coefficients cannot be set in the data file or via the angle_coeff command until an angle_style has been assigned.

Angle_coeff command before simulation box is defined

The angle_coeff command cannot be used before a read_data, read_restart, or create_box command.

Angle_coeff command when no angles allowed

The chosen atom style does not allow for angles to be defined.

Angle_style command when no angles allowed

The chosen atom style does not allow for angles to be defined.

Angles assigned incorrectly

Angles read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Angles defined but no angle types

The data file header lists angles but no angle types.

Another input script is already being processed

Cannot attempt to open a 2nd input script, when the original file is still being processed.

Append boundary must be shrink/minimum

The boundary style of the face where atoms are added must be of type m (shrink/minimum).

Arccos of invalid value in variable formula

Argument of arccos() must be between -1 and 1.

Arcsin of invalid value in variable formula

Argument of arcsin() must be between -1 and 1.

Assigning body parameters to non-body atom

Self-explanatory.

Assigning ellipsoid parameters to non-ellipsoid atom

Self-explanatory.

Assigning line parameters to non-line atom

Self-explanatory.

Assigning tri parameters to non-tri atom

Self-explanatory.

Atom IDs must be consecutive for velocity create loop all

Self-explanatory.

Atom count changed in fix neb

This is not allowed in a NEB calculation.

Atom count is inconsistent, cannot write restart file

Sum of atoms across processors does not equal initial total count. This is probably because you have lost some atoms.

Atom in too many rigid bodies - boost MAXBODY

Fix poems has a parameter MAXBODY (in fix_poems.cpp) which determines the maximum number of rigid bodies a single atom can belong to (i.e. a multibody joint). The bodies you have defined exceed this limit.

Atom sort did not operate correctly

This is an internal LAMMPS error. Please report it to the developers.

Atom sorting has bin size = 0.0

The neighbor cutoff is being used as the bin size, but it is zero. Thus you must explicitly list a bin size in the atom_modify sort command or turn off sorting.

Atom style hybrid cannot have hybrid as an argument

Self-explanatory.

Atom style hybrid cannot use same atom style twice

Self-explanatory.

Atom vector in equal-style variable formula

Atom vectors generate one value per atom which is not allowed in an equal-style variable.

Atom-style variable in equal-style variable formula

Atom-style variables generate one value per atom which is not allowed in an equal-style variable.

Atom_modify map command after simulation box is defined

The atom_modify map command cannot be used after a read_data, read_restart, or create_box command.

Atom_modify sort and first options cannot be used together

Self-explanatory.

Atom_style command after simulation box is defined

The atom_style command cannot be used after a read_data, read_restart, or create_box command.

Atom_style line can only be used in 2d simulations

Self-explanatory.

Atom_style tri can only be used in 3d simulations

Self-explanatory.

Attempt to pop empty stack in fix box/relax

Internal LAMMPS error. Please report it to the developers.

Attempt to push beyond stack limit in fix box/relax

Internal LAMMPS error. Please report it to the developers.

Attempting to rescale a 0.0 temperature

Cannot rescale a temperature that is already 0.0.

Bad FENE bond

Two atoms in a FENE bond have become so far apart that the bond cannot be computed.

Bad TIP4P angle type for PPPM/TIP4P

Specified angle type is not valid.

Bad TIP4P angle type for PPPMDisp/TIP4P

Specified angle type is not valid.

Bad TIP4P bond type for PPPM/TIP4P

Specified bond type is not valid.

Bad TIP4P bond type for PPPMDisp/TIP4P

Specified bond type is not valid.

Bad fix ID in fix append/atoms command

The value of the fix_id for keyword spatial must start with the suffix f_.

Bad grid of processors

The 3d grid of processors defined by the processors command does not match the number of processors LAMMPS is being run on.

Bad kspace_modify slab parameter

Kspace_modify value for the slab/volume keyword must be >= 2.0.

Bad matrix inversion in mldivide3

This error should not occur unless the matrix is badly formed.

Bad principal moments

Fix rigid did not compute the principal moments of inertia of a rigid group of atoms correctly.

Bad quadratic solve for particle/line collision

This is an internal error. It should nornally not occur.

Bad quadratic solve for particle/tri collision

This is an internal error. It should nornally not occur.

Balance command before simulation box is defined

The balance command cannot be used before a read_data, read_restart, or create_box command.

Balance dynamic string is invalid

The string can only contain the characters "x", "y", or "z".

Balance produced bad splits

This should not occur. It means two or more cutting plane locations are on top of each other or out of order. Report the problem to the developers.

Bias compute does not calculate a velocity bias

The specified compute must compute a bias for temperature.

Bias compute does not calculate temperature

The specified compute must compute temperature.

Bias compute group does not match compute group

The specified compute must operate on the same group as the parent compute.

Big particle in fix srd cannot be point particle

Big particles must be extended spheriods or ellipsoids.

Bigint setting in lmptype.h is invalid

Size of bigint is less than size of tagint.

Bigint setting in lmptype.h is not compatible

Bigint stored in restart file is not consistent with LAMMPS version you are running.

Bitmapped lookup tables require int/float be same size

Cannot use pair tables on this machine, because of word sizes. Use the pair_modify command with table 0 instead.

Bitmapped table in file does not match requested table

Setting for bitmapped table in pair_coeff command must match table in file exactly.

Bitmapped table is incorrect length in table file

Number of table entries is not a correct power of 2.

Bond and angle potentials must be defined for TIP4P

Cannot use TIP4P pair potential unless bond and angle potentials are defined.

Bond atom missing in box size check

The 2nd atoms needed to compute a particular bond is missing on this processor. Typically this is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.

Bond atom missing in image check

The 2nd atom in a particular bond is missing on this processor. Typically this is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond atom missing in set command

The set command cannot find one or more atoms in a particular bond on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid bond.

Bond atoms %d %d missing on proc %d at step %ld

The 2nd atom needed to compute a particular bond is missing on this processor. Typically this is because the pairwise cutoff is set too short or the bond has blown apart and an atom is too far away.

Bond coeff for hybrid has invalid style

Bond style hybrid uses another bond style as one of its coefficients. The bond style used in the bond_coeff command or read from a restart file is not recognized.

Bond coeffs are not set

No bond coefficients have been assigned in the data file or via the bond_coeff command.

Bond extent > half of periodic box length

This error was detected by the neigh_modify check yes setting. It is an error because the bond atoms are so far apart it is ambiguous how it should be defined.

Bond potential must be defined for SHAKE

Cannot use fix shake unless bond potential is defined.

Bond style hybrid cannot have hybrid as an argument

Self-explanatory.

Bond style hybrid cannot have none as an argument

Self-explanatory.

Bond style hybrid cannot use same bond style twice

Self-explanatory.

Bond style quartic cannot be used with 3,4-body interactions

No angle, dihedral, or improper styles can be defined when using bond style quartic.

Bond style quartic requires special_bonds = 1,1,1

This is a restriction of the current bond quartic implementation.

Bond table parameters did not set N

List of bond table parameters must include N setting.

Bond table values are not increasing

The values in the tabulated file must be monotonically increasing.

Bond_coeff command before bond_style is defined

Coefficients cannot be set in the data file or via the bond_coeff command until an bond_style has been assigned.

Bond_coeff command before simulation box is defined

The bond_coeff command cannot be used before a read_data, read_restart, or create_box command.

Bond_coeff command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bond_style command when no bonds allowed

The chosen atom style does not allow for bonds to be defined.

Bonds assigned incorrectly

Bonds read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Bonds defined but no bond types

The data file header lists bonds but no bond types.

Both sides of boundary must be periodic

Cannot specify a boundary as periodic only on the lo or hi side. Must be periodic on both sides.

Boundary command after simulation box is defined

The boundary command cannot be used after a read_data, read_restart, or create_box command.

Box bounds are invalid

The box boundaries specified in the read_data file are invalid. The lo value must be less than the hi value for all 3 dimensions.

Box command after simulation box is defined

The box command cannot be used after a read_data, read_restart, or create_box command.

CPU neighbor lists must be used for ellipsoid/sphere mix

When using Gay-Berne or RE-squared pair styles with both ellipsoidal and spherical particles, the neighbor list must be built on the CPU

Can not specify Pxy/Pxz/Pyz in fix box/relax with non-triclinic box

Only triclinic boxes can be used with off-diagonal pressure components. See the region prism command for details.

Can not specify Pxy/Pxz/Pyz in fix nvt/npt/nph with non-triclinic box

Only triclinic boxes can be used with off-diagonal pressure components. See the region prism command for details.

Can only use -plog with multiple partitions

Self-explanatory. See doc page discussion of command-line switches.

Can only use -pscreen with multiple partitions

Self-explanatory. See doc page discussion of command-line switches.

Can only use NEB with 1-processor replicas
>This is current restriction for NEB as implemented in LAMMPS.

Can only use TAD with 1-processor replicas for NEB
>This is current restriction for NEB as implemented in LAMMPS.

Cannot (yet) do analytic differentiation with pppm/gpu
>This is a current restriction of this command.

Cannot (yet) use K-space slab correction with compute group/group
>This option is not yet supported.

Cannot (yet) use Kspace slab correction with compute group/group
>This option is not yet supported.

Cannot (yet) use MSM with 2d simulation
>This feature is not yet supported.

Cannot (yet) use MSM with triclinic box
>This feature is not yet supported.

Cannot (yet) use PPPM with triclinic box
>This feature is not yet supported.

Cannot (yet) use PPPMDisp with triclinic box
>This feature is not yet supported.

Cannot (yet) use single precision with MSM (remove -DFFT_SINGLE from Makefile and recompile)
>Single precision cannot be used with MSM.

Cannot add atoms to fix move variable
>Atoms can not be added afterwards to this fix option.

Cannot append atoms to a triclinic box
>The simulation box must be defined with edges alligned with the Cartesian axes.

Cannot balance in z dimension for 2d simulation
>Self-explanatory.

Cannot change box ortho/triclinic with certain fixes defined
>This is because those fixes store the shape of the box. You need to use unfix to discard the fix, change the box, then redefine a new fix.

Cannot change box ortho/triclinic with dumps defined
>This is because some dumps store the shape of the box. You need to use undump to discard the dump, change the box, then redefine a new dump.

Cannot change box tilt factors for orthogonal box
>Cannot use tilt factors unless the simulation box is non-orthogonal.

Cannot change box to orthogonal when tilt is non-zero
>Self-explanatory.

Cannot change box z boundary to nonperiodic for a 2d simulation
>Self-explanatory.

Cannot change dump_modify every for dump dcd
>The frequency of writing dump dcd snapshots cannot be changed.

Cannot change dump_modify every for dump xtc
>The frequency of writing dump xtc snapshots cannot be changed.

Cannot change timestep once fix srd is setup
>This is because various SRD properties depend on the timestep size.

Cannot change timestep with fix pour
>This fix pre-computes some values based on the timestep, so it cannot be changed during a simulation run.

Cannot change_box after reading restart file with per-atom info
>This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0-timestep run which will assign the restart file info to actual atoms.

Cannot change_box in xz or yz for 2d simulation
>Self-explanatory.

Cannot change_box in z dimension for 2d simulation
>Self-explanatory.

Cannot compute initial g_ewald_disp

LAMMPS failed to compute an initial guess for the PPPM_disp g_ewald_6 factor that partitions the computation between real space and k-space for Disptersion interactions.

Cannot create an atom map unless atoms have IDs

The simulation requires a mapping from global atom IDs to local atoms, but the atoms that have been defined have no IDs.

Cannot create atoms with undefined lattice

Must use the lattice command before using the create_atoms command.

Cannot create/grow a vector/array of pointers for %s

LAMMPS code is making an illegal call to the templated memory allocaters, to create a vector or array of pointers.

Cannot create_atoms after reading restart file with per-atom info

The per-atom info was stored to be used when by a fix that you may re-define. If you add atoms before re-defining the fix, then there will not be a correct amount of per-atom info.

Cannot create_box after simulation box is defined

The create_box command cannot be used after a read_data, read_restart, or create_box command.

Cannot currently use pair reax with pair hybrid

This is not yet supported.

Cannot currently use pppm/gpu with fix balance.

Self-explanatory.

Cannot delete group all

Self-explanatory.

Cannot delete group currently used by a compute

Self-explanatory.

Cannot delete group currently used by a dump

Self-explanatory.

Cannot delete group currently used by a fix

Self-explanatory.

Cannot delete group currently used by atom_modify first

Self-explanatory.

Cannot displace_atoms after reading restart file with per-atom info

This is because the restart file info cannot be migrated with the atoms. You can get around this by performing a 0-timestep run which will assign the restart file info to actual atoms.

Cannot do GCMC on atoms in atom_modify first group

This is a restriction due to the way atoms are organized in a list to enable the atom_modify first command.

Cannot dump JPG file

LAMMPS was not built with the -DLAMMPS_JPEG switch in the Makefile.

Cannot dump sort on atom IDs with no atom IDs defined

Self-explanatory.

Cannot evaporate atoms in atom_modify first group

This is a restriction due to the way atoms are organized in a list to enable the atom_modify first command.

Cannot find delete_bonds group ID

Group ID used in the delete_bonds command does not exist.

Cannot have both pair_modify shift and tail set to yes

These 2 options are contradictory.

Cannot open -reorder file

Self-explanatory.

Cannot open ADP potential file %s

The specified ADP potential file cannot be opened. Check that the path and name are correct.

Cannot open AIREBO potential file %s

The specified AIREBO potential file cannot be opened. Check that the path and name are correct.

Cannot open BOP potential file %s

The specified BOP potential file cannot be opened. Check that the path and name are correct.

Cannot open COMB potential file %s

The specified COMB potential file cannot be opened. Check that the path and name are correct.

Cannot open EAM potential file %s

The specified EAM potential file cannot be opened. Check that the path and name are correct.

Cannot open EIM potential file %s

The specified EIM potential file cannot be opened. Check that the path and name are correct.

Cannot open LCBOP potential file %s

The specified LCBOP potential file cannot be opened. Check that the path and name are correct.

Cannot open MEAM potential file %s

The specified MEAM potential file cannot be opened. Check that the path and name are correct.

Cannot open Stillinger-Weber potential file %s

The specified SW potential file cannot be opened. Check that the path and name are correct.

Cannot open Tersoff potential file %s

The specified Tersoff potential file cannot be opened. Check that the path and name are correct.

Cannot open balance output file

Self-explanatory.

Cannot open custom file

Self-explanatory.

Cannot open dir to search for restart file

Using a "*" in the name of the restart file will open the current directory to search for matching file names.

Cannot open dump file

The output file for the dump command cannot be opened. Check that the path and name are correct.

Cannot open file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open file variable file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/correlate file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/histo file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/spatial file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix ave/time file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix balance output file

Self-explanatory.

Cannot open fix poems file %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix print file %s

The output file generated by the fix print command cannot be opened

Cannot open fix qeq/comb file %s

The output file for the fix qeq/combs command cannot be opened. Check that the path and name are correct.

Cannot open fix reax/bonds file %s

The output file for the fix reax/bonds command cannot be opened. Check that the path and name are correct.

Cannot open fix rigid infile %s

The specified file cannot be opened. Check that the path and name are correct.

Cannot open fix tmd file %s

The output file for the fix tmd command cannot be opened. Check that the path and name are correct.

Cannot open fix ttm file %s

The output file for the fix ttm command cannot be opened. Check that the path and name are correct.

Cannot open gzipped file
> LAMMPS is attempting to open a gzipped version of the specified file but was unsuccessful. Check that the path and name are correct.

Cannot open input script %s
> Self-explanatory.

Cannot open log.lammps
> The default LAMMPS log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open logfile
> The LAMMPS log file named in a command-line argument cannot be opened. Check that the path and name are correct.

Cannot open logfile %s
> The LAMMPS log file specified in the input script cannot be opened. Check that the path and name are correct.

Cannot open pair_write file
> The specified output file for pair energies and forces cannot be opened. Check that the path and name are correct.

Cannot open processors output file
> Self-explanatory.

Cannot open restart file %s
> Self-explanatory.

Cannot open screen file
> The screen file specified as a command-line argument cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe log file
> For a multi-partition run, the master log file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot open universe screen file
> For a multi-partition run, the master screen file cannot be opened. Check that the directory you are running in allows for files to be created.

Cannot read_data after simulation box is defined
> The read_data command cannot be used after a read_data, read_restart, or create_box command.

Cannot read_restart after simulation box is defined
> The read_restart command cannot be used after a read_data, read_restart, or create_box command.

Cannot redefine variable as a different style
> An equal-style variable can be re-defined but only if it was originally an equal-style variable.

Cannot replicate 2d simulation in z dimension
> The replicate command cannot replicate a 2d simulation in the z dimension.

Cannot replicate with fixes that store atom quantities
> Either fixes are defined that create and store atom-based vectors or a restart file was read which included atom-based vectors for fixes. The replicate command cannot duplicate that information for new atoms. You should use the replicate command before fixes are applied to the system.

Cannot reset timestep with a dynamic region defined
> Dynamic regions (see the region command) have a time dependence. Thus you cannot change the timestep when one or more of these are defined.

Cannot reset timestep with a time-dependent fix defined
> You cannot reset the timestep when a fix that keeps track of elapsed time is in place.

Cannot run 2d simulation with nonperiodic Z dimension
> Use the boundary command to make the z dimension periodic in order to run a 2d simulation.

Cannot set both respa pair and inner/middle/outer
> In the rRESPA integrator, you must compute pairwise potentials either all together (pair), or in pieces (inner/middle/outer). You can't do both.

Cannot set dump_modify flush for dump xtc
> Self-explanatory.

Cannot set mass for this atom style
> This atom style does not support mass settings for each atom type. Instead they are defined on a per-atom basis in the data file.

Cannot set meso_rho for this atom style
> Self-explanatory.

Cannot set non-zero image flag for non-periodic dimension
> Self-explanatory.

Cannot set non-zero z velocity for 2d simulation
> Self-explanatory.

Cannot set quaternion for atom that has none
> Self-explanatory.

Cannot set respa middle without inner/outer
> In the rRESPA integrator, you must define both a inner and outer setting in order to use a middle setting.

Cannot set temperature for fix rigid/nph
> The temp keyword cannot be specified.

Cannot set theta for atom that is not a line
> Self-explanatory.

Cannot set this attribute for this atom style
> The attribute being set does not exist for the defined atom style.

Cannot set variable z velocity for 2d simulation
> Self-explanatory.

Cannot skew triclinic box in z for 2d simulation
> Self-explanatory.

Cannot use -cuda on without USER-CUDA installed
> The USER-CUDA package must be installed via "make yes-user-cuda" before LAMMPS is built.

Cannot use -reorder after -partition
> Self-explanatory. See doc page discussion of command-line switches.

Cannot use Ewald with 2d simulation
> The kspace style ewald cannot be used in 2d simulations. You can use 2d Ewald in a 3d simulation; see the kspace_modify command.

Cannot use Ewald with triclinic box
> This feature is not yet supported.

Cannot use Ewald/disp solver on system with no charge or LJ particles
> No atoms in system have a non-zero charge or are LJ particles. Change charges or change options of the kspace solver/pair style.

Cannot use EwaldDisp with 2d simulation
> This is a current restriction of this command.

Cannot use NEB unless atom map exists
> Use the atom_modify command to create an atom map.

Cannot use NEB with a single replica
> Self-explanatory.

Cannot use NEB with atom_modify sort enabled
> This is current restriction for NEB implemented in LAMMPS.

Cannot use PPPM with 2d simulation
> The kspace style pppm cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

Cannot use PPPMDisp with 2d simulation
> The kspace style pppm/disp cannot be used in 2d simulations. You can use 2d PPPM in a 3d simulation; see the kspace_modify command.

Cannot use PRD with a time-dependent fix defined
> PRD alters the timestep in ways that will mess up these fixes.

Cannot use PRD with a time-dependent region defined
> PRD alters the timestep in ways that will mess up these regions.

Cannot use PRD with atom_modify sort enabled
> This is a current restriction of PRD. You must turn off sorting, which is enabled by default, via the atom_modify command.

Cannot use PRD with multi-processor replicas unless atom map exists
> Use the atom_modify command to create an atom map.

Cannot use TAD unless atom map exists for NEB
> See atom_modify map command to set this.

Cannot use TAD with a single replica for NEB
> NEB requires multiple replicas.

Cannot use TAD with atom_modify sort enabled for NEB
> This is a current restriction of NEB.

Cannot use a damped dynamics min style with fix box/relax
> This is a current restriction in LAMMPS. Use another minimizer style.

Cannot use a damped dynamics min style with per-atom DOF
> This is a current restriction in LAMMPS. Use another minimizer style.

Cannot use append/atoms in periodic dimension
> The boundary style of the face where atoms are added can not be of type p (periodic).

Cannot use compute cluster/atom unless atoms have IDs
> Atom IDs are used to identify clusters.

Cannot use cwiggle in variable formula between runs
> This is a function of elapsed time.

Cannot use delete_atoms unless atoms have IDs
> Your atoms do not have IDs, so the delete_atoms command cannot be used.

Cannot use delete_bonds with non-molecular system
> Your choice of atom style does not have bonds.

Cannot use fix GPU with USER-CUDA mode enabled
> You cannot use both the GPU and USER-CUDA packages together. Use one or the other.

Cannot use fix TMD unless atom map exists
> Using this fix requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non-molecular problems. Using the atom_modify map command will force an atom map to be created.

Cannot use fix ave/spatial z for 2 dimensional model
> Self-explanatory.

Cannot use fix bond/break with non-molecular systems
> Self-explanatory.

Cannot use fix bond/create with non-molecular systems
> Self-explanatory.

Cannot use fix box/relax on a 2nd non-periodic dimension
> When specifying an off-diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.

Cannot use fix box/relax on a non-periodic dimension
> When specifying a diagonal pressure component, the dimension must be periodic.

Cannot use fix box/relax with both relaxation and scaling on a tilt factor
> When specifying scaling on a tilt factor component, that component can not also be controlled by the barostat. E.g. if scalexy yes is specified and also keyword tri or xy, this is wrong.

Cannot use fix box/relax with tilt factor scaling on a 2nd non-periodic dimension
> When specifying scaling on a tilt factor component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.

Cannot use fix deform on a shrink-wrapped boundary
> The x, y, z options cannot be applied to shrink-wrapped dimensions.

Cannot use fix deform tilt on a shrink-wrapped 2nd dim
> This is because the shrink-wrapping will change the value of the strain implied by the tilt factor.

Cannot use fix deform trate on a box with zero tilt
> The trate style alters the current strain.

Cannot use fix enforce2d with 3d simulation
>   Self-explanatory.

Cannot use fix gcmc in a 2d simulation
>   Fix gcmc is set up to run in 3d only. No 2d simulations with fix gcmc are allowed.

Cannot use fix gcmc with a triclinic box
>   Fix gcmc is set up to run with othogonal boxes only. Simulations with triclinic boxes and fix gcmc are not allowed.

Cannot use fix msst without per-type mass defined
>   Self-explanatory.

Cannot use fix npt and fix deform on same component of stress tensor
>   This would be changing the same box dimension twice.

Cannot use fix nvt/npt/nph on a 2nd non-periodic dimension
>   When specifying an off-diagonal pressure component, the 2nd of the two dimensions must be periodic. E.g. if the xy component is specified, then the y dimension must be periodic.

Cannot use fix nvt/npt/nph on a non-periodic dimension
>   When specifying a diagonal pressure component, the dimension must be periodic.

Cannot use fix nvt/npt/nph with both xy dynamics and xy scaling
>   Self-explanatory.

Cannot use fix nvt/npt/nph with both xz dynamics and xz scaling
>   Self-explanatory.

Cannot use fix nvt/npt/nph with both yz dynamics and yz scaling
>   Self-explanatory.

Cannot use fix nvt/npt/nph with xy scaling when y is non-periodic dimension
>   The 2nd dimension in the barostatted tilt factor must be periodic.

Cannot use fix nvt/npt/nph with xz scaling when z is non-periodic dimension
>   The 2nd dimension in the barostatted tilt factor must be periodic.

Cannot use fix nvt/npt/nph with yz scaling when z is non-periodic dimension
>   The 2nd dimension in the barostatted tilt factor must be periodic.

Cannot use fix pour with triclinic box
>   This feature is not yet supported.

Cannot use fix press/berendsen and fix deform on same component of stress tensor
>   These commands both change the box size/shape, so you cannot use both together.

Cannot use fix press/berendsen on a non-periodic dimension
>   Self-explanatory.

Cannot use fix press/berendsen with triclinic box
>   Self-explanatory.

Cannot use fix reax/bonds without pair_style reax
>   Self-explantory.

Cannot use fix rigid npt/nph and fix deform on same component of stress tensor
>   This would be changing the same box dimension twice.

Cannot use fix rigid npt/nph on a non-periodic dimension
>   When specifying a diagonal pressure component, the dimension must be periodic.

Cannot use fix shake with non-molecular system
>   Your choice of atom style does not have bonds.

Cannot use fix ttm with 2d simulation
>   This is a current restriction of this fix due to the grid it creates.

Cannot use fix ttm with triclinic box
>   This is a current restriction of this fix due to the grid it creates.

Cannot use fix wall in periodic dimension
>   Self-explanatory.

Cannot use fix wall zlo/zhi for a 2d simulation
>   Self-explanatory.

Cannot use fix wall/reflect in periodic dimension
>   Self-explanatory.

Cannot use fix wall/reflect zlo/zhi for a 2d simulation
    Self-explanatory.
Cannot use fix wall/srd in periodic dimension
    Self-explanatory.
Cannot use fix wall/srd more than once
    Nor is their a need to since multiple walls can be specified in one command.
Cannot use fix wall/srd without fix srd
    Self-explanatory.
Cannot use fix wall/srd zlo/zhi for a 2d simulation
    Self-explanatory.
Cannot use force/hybrid_neigh with triclinic box
    Self-explanatory.
Cannot use force/neigh with triclinic box
    This is a current limitation of the GPU implementation in LAMMPS.
Cannot use kspace solver on system with no charge
    No atoms in system have a non-zero charge.
Cannot use kspace solver with selected options on system with no charge
    No atoms in system have a non-zero charge. Change charges or change options of the kspace
    solver/pair style.
Cannot use lines with fix srd unless overlap is set
    This is because line segements are connected to each other.
Cannot use multiple fix wall commands with pair brownian
    Self-explanatory.
Cannot use multiple fix wall commands with pair lubricate
    Self-explanatory.
Cannot use multiple fix wall commands with pair lubricate/poly
    Self-explanatory.
Cannot use multiple fix wall commands with pair lubricateU
    Self-explanatory.
Cannot use neigh_modify exclude with GPU neighbor builds
    This is a current limitation of the GPU implementation in LAMMPS.
Cannot use neighbor bins - box size << cutoff
    Too many neighbor bins will be created. This typically happens when the simulation box is very small
    in some dimension, compared to the neighbor cutoff. Use the "nsq" style instead of "bin" style.
Cannot use newton pair with born/coul/long/gpu pair style
    Self-explanatory.
Cannot use newton pair with born/coul/wolf/gpu pair style
    Self-explanatory.
Cannot use newton pair with born/gpu pair style
    Self-explantory.
Cannot use newton pair with buck/coul/cut/gpu pair style
    Self-explanatory.
Cannot use newton pair with buck/coul/long/gpu pair style
    Self-explanatory.
Cannot use newton pair with buck/gpu pair style
    Self-explanatory.
Cannot use newton pair with colloid/gpu pair style
    Self-explanatory.
Cannot use newton pair with coul/dsf/gpu pair style
    Self-explanatory.
Cannot use newton pair with coul/long/gpu pair style
    Self-explanatory.
Cannot use newton pair with dipole/cut/gpu pair style
    Self-explanatory.

Cannot use newton pair with eam/gpu pair style
> Self-explanatory.

Cannot use newton pair with gauss/gpu pair style
> Self-explanatory.

Cannot use newton pair with gayberne/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/charmm/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/class2/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/class2/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/debye/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/dsf/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/coul/long/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj/expand/gpu pair style
> Self-explanatory.

Cannot use newton pair with lj96/cut/gpu pair style
> Self-explanatory.

Cannot use newton pair with morse/gpu pair style
> Self-explanatory.

Cannot use newton pair with resquared/gpu pair style
> Self-explanatory.

Cannot use newton pair with table/gpu pair style
> Self-explanatory.

Cannot use newton pair with yukawa/colloid/gpu pair style
> Self-explanatory.

Cannot use newton pair with yukawa/gpu pair style
> Self-explanatory.

Cannot use non-zero forces in an energy minimization
> Fix setforce cannot be used in this manner. Use fix addforce instead.

Cannot use nonperiodic boundares with fix ttm
> This fix requires a fully periodic simulation box.

Cannot use nonperiodic boundaries with Ewald
> For kspace style ewald, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with EwaldDisp
> For kspace style ewald/disp, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with PPPM
> For kspace style pppm, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use nonperiodic boundaries with PPPMDisp
> For kspace style pppm/disp, all 3 dimensions must have periodic boundaries unless you use the kspace_modify command to define a 2d slab with a non-periodic z dimension.

Cannot use order greater than 8 with pppm/gpu.
> Self-explanatory.

Cannot use pair hybrid with GPU neighbor builds
> See documentation for fix gpu.

Cannot use pair tail corrections with 2d simulations
> The correction factors are only currently defined for 3d systems.

Cannot use processors part command without using partitions
> See the command-line -partition switch.

Cannot use ramp in variable formula between runs
> This is because the ramp() function is time dependent.

Cannot use region INF or EDGE when box does not exist
> Regions that extend to the box boundaries can only be used after the create_box command has been used.

Cannot use set atom with no atom IDs defined
> Atom IDs are not defined, so they cannot be used to identify an atom.

Cannot use set mol with no molecule IDs defined
> Self-explanatory.

Cannot use slab correction with MSM
> Slab correction can only be used with Ewald and PPPM, not MSM.

Cannot use swiggle in variable formula between runs
> This is a function of elapsed time.

Cannot use tris with fix srd unless overlap is set
> This is because triangles are connected to each other.

Cannot use variable energy with constant force in fix addforce
> This is because for constant force, LAMMPS can compute the change in energy directly.

Cannot use variable every setting for dump dcd
> The format of DCD dump files requires snapshots be output at a constant frequency.

Cannot use variable every setting for dump xtc
> The format of this file requires snapshots at regular intervals.

Cannot use vdisplace in variable formula between runs
> This is a function of elapsed time.

Cannot use velocity create loop all unless atoms have IDs
> Atoms in the simulation to do not have IDs, so this style of velocity creation cannot be performed.

Cannot use wall in periodic dimension
> Self-explanatory.

Cannot wiggle and shear fix wall/gran
> Cannot specify both options at the same time.

Cannot zero Langevin force of 0 atoms
> The group has zero atoms, so you cannot request its force be zeroed.

Cannot zero momentum of 0 atoms
> The collection of atoms for which momentum is being computed has no atoms.

Change_box command before simulation box is defined
> Self-explanatory.

Change_box volume used incorrectly
> The "dim volume" option must be used immediately following one or two settings for "dim1 ..." (and optionally "dim2 ...") and must be for a different dimension, i.e. dim != dim1 and dim != dim2.

Communicate group != atom_modify first group
> Self-explanatory.

Compute ID for compute atom/molecule does not exist
> Self-explanatory.

Compute ID for compute reduce does not exist
> Self-explanatory.

Compute ID for compute slice does not exist
> Self-explanatory.

Compute ID for fix ave/atom does not exist
> Self-explanatory.

Compute ID for fix ave/correlate does not exist
      Self-explanatory.
Compute ID for fix ave/histo does not exist
      Self-explanatory.
Compute ID for fix ave/spatial does not exist
      Self-explanatory.
Compute ID for fix ave/time does not exist
      Self-explanatory.
Compute ID for fix store/state does not exist
      Self-explanatory.
Compute ID must be alphanumeric or underscore characters
      Self-explanatory.
Compute angle/local used when angles are not allowed
      The atom style does not support angles.
Compute atom/molecule compute array is accessed out-of-range
      Self-explanatory.
Compute atom/molecule compute does not calculate a per-atom array
      Self-explanatory.
Compute atom/molecule compute does not calculate a per-atom vector
      Self-explanatory.
Compute atom/molecule compute does not calculate per-atom values
      Self-explanatory.
Compute atom/molecule fix array is accessed out-of-range
      Self-explanatory.
Compute atom/molecule fix does not calculate a per-atom array
      Self-explanatory.
Compute atom/molecule fix does not calculate a per-atom vector
      Self-explanatory.
Compute atom/molecule fix does not calculate per-atom values
      Self-explanatory.
Compute atom/molecule requires molecular atom style
      Self-explanatory.
Compute atom/molecule variable is not atom-style variable
      Self-explanatory.
Compute body/local requires atom style body
      Self-explanatory.
Compute bond/local used when bonds are not allowed
      The atom style does not support bonds.
Compute centro/atom requires a pair style be defined
      This is because the computation of the centro-symmetry values uses a pairwise neighbor list.
Compute cluster/atom cutoff is longer than pairwise cutoff
      Cannot identify clusters beyond cutoff.
Compute cluster/atom requires a pair style be defined
      This is so that the pair style defines a cutoff distance which is used to find clusters.
Compute cna/atom cutoff is longer than pairwise cutoff
      Self-explantory.
Compute cna/atom requires a pair style be defined
      Self-explantory.
Compute com/molecule requires molecular atom style
      Self-explanatory.
Compute contact/atom requires a pair style be defined
      Self-explantory.
Compute contact/atom requires atom style sphere
      Self-explanatory.

Compute coord/atom cutoff is longer than pairwise cutoff
    Cannot compute coordination at distances longer than the pair cutoff, since those atoms are not in the neighbor list.
Compute coord/atom requires a pair style be defined
    Self-explantory.
Compute damage/atom requires peridynamic potential
    Damage is a Peridynamic-specific metric. It requires you to be running a Peridynamics simulation.
Compute dihedral/local used when dihedrals are not allowed
    The atom style does not support dihedrals.
Compute does not allow an extra compute or fix to be reset
    This is an internal LAMMPS error. Please report it to the developers.
Compute erotate/asphere requires atom style ellipsoid or line or tri
    Self-explanatory.
Compute erotate/asphere requires extended particles
    This compute cannot be used with point paritlces.
Compute erotate/sphere requires atom style sphere
    Self-explanatory.
Compute erotate/sphere/atom requires atom style sphere
    Self-explanatory.
Compute event/displace has invalid fix event assigned
    This is an internal LAMMPS error. Please report it to the developers.
Compute group/group group ID does not exist
    Self-explanatory.
Compute gyration/molecule requires molecular atom style
    Self-explanatory.
Compute heat/flux compute ID does not compute ke/atom
    Self-explanatory.
Compute heat/flux compute ID does not compute pe/atom
    Self-explanatory.
Compute heat/flux compute ID does not compute stress/atom
    Self-explanatory.
Compute improper/local used when impropers are not allowed
    The atom style does not support impropers.
Compute inertia/molecule requires molecular atom style
    Self-explanatory.
Compute msd/molecule requires molecular atom style
    Self-explanatory.
Compute nve/asphere requires atom style ellipsoid
    Self-explanatory.
Compute nvt/nph/npt asphere requires atom style ellipsoid
    Self-explanatory.
Compute pair must use group all
    Pair styles accumlate energy on all atoms.
Compute pe must use group all
    Energies computed by potentials (pair, bond, etc) are computed on all atoms.
Compute pressure must use group all
    Virial contributions computed by potentials (pair, bond, etc) are computed on all atoms.
Compute pressure temperature ID does not compute temperature
    The compute ID assigned to a pressure computation must compute temperature.
Compute property/atom for atom property that isn't allocated
    Self-explanatory.
Compute property/local cannot use these inputs together
    Only inputs that generate the same number of datums can be used togther. E.g. bond and angle quantities cannot be mixed.

Compute property/local for property that isn't allocated
Self-explanatory.
Compute property/molecule requires molecular atom style
Self-explanatory.
Compute rdf requires a pair style be defined
Self-explanatory.
Compute reduce compute array is accessed out-of-range
An index for the array is out of bounds.
Compute reduce compute calculates global values
A compute that calculates peratom or local values is required.
Compute reduce compute does not calculate a local array
Self-explanatory.
Compute reduce compute does not calculate a local vector
Self-explanatory.
Compute reduce compute does not calculate a per-atom array
Self-explanatory.
Compute reduce compute does not calculate a per-atom vector
Self-explanatory.
Compute reduce fix array is accessed out-of-range
An index for the array is out of bounds.
Compute reduce fix calculates global values
A fix that calculates peratom or local values is required.
Compute reduce fix does not calculate a local array
Self-explanatory.
Compute reduce fix does not calculate a local vector
Self-explanatory.
Compute reduce fix does not calculate a per-atom array
Self-explanatory.
Compute reduce fix does not calculate a per-atom vector
Self-explanatory.
Compute reduce replace requires min or max mode
Self-explanatory.
Compute reduce variable is not atom-style variable
Self-explanatory.
Compute slice compute array is accessed out-of-range
An index for the array is out of bounds.
Compute slice compute does not calculate a global array
Self-explanatory.
Compute slice compute does not calculate a global vector
Self-explanatory.
Compute slice compute does not calculate global vector or array
Self-explanatory.
Compute slice compute vector is accessed out-of-range
The index for the vector is out of bounds.
Compute slice fix array is accessed out-of-range
An index for the array is out of bounds.
Compute slice fix does not calculate a global array
Self-explanatory.
Compute slice fix does not calculate a global vector
Self-explanatory.
Compute slice fix does not calculate global vector or array
Self-explanatory.
Compute slice fix vector is accessed out-of-range
The index for the vector is out of bounds.

Compute temp/asphere requires atom style ellipsoid
> Self-explanatory.

Compute temp/asphere requires extended particles
> This compute cannot be used with point paritlces.

Compute temp/partial cannot use vz for 2d systemx
> Self-explanatory.

Compute temp/profile cannot bin z for 2d systems
> Self-explanatory.

Compute temp/profile cannot use vz for 2d systemx
> Self-explanatory.

Compute temp/sphere requires atom style sphere
> Self-explanatory.

Compute ti kspace style does not exist
> Self-explanatory.

Compute ti pair style does not exist
> Self-explanatory.

Compute ti tail when pair style does not compute tail corrections
> Self-explanatory.

Compute used in variable between runs is not current
> Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Compute used in variable thermo keyword between runs is not current
> Some thermo keywords rely on a compute to calculate their value(s). Computes cannot be invoked by a variable in between runs. Thus they must have been evaluated on the last timestep of the previous run in order for their value(s) to be accessed. See the doc page for the variable command for more info.

Compute voronoi/atom not allowed for triclinic boxes
> This is a current restriction of this command.

Computed temperature for fix temp/berendsen cannot be 0.0
> Self-explanatory.

Computed temperature for fix temp/rescale cannot be 0.0
> Cannot rescale the temperature to a new value if the current temperature is 0.0.

Could not adjust g_ewald_6
> The Newton-Raphson solver failed to converge to a good value for g_ewald_6. This error should not occur for typical problems. Please send an email to the developers.

Could not compute g_ewald
> The Newton-Raphson solver failed to converge to a good value for g_ewald. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size for Coulomb interaction
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not compute grid size for dispersion
> The code is unable to compute a grid size consistent with the desired accuracy. This error should not occur for typical problems. Please send an email to the developers.

Could not count initial bonds in fix bond/create
> Could not find one of the atoms in a bond on this processor.

Could not create 3d FFT plan
> The FFT setup for the PPPM solver failed, typically due to lack of memory. This is an unusual error. Check the size of the FFT grid you are requesting.

Could not create 3d grid of processors

The specified constraints did not allow a Px by Py by Pz grid to be created where Px * Py * Pz = P = total number of processors.

Could not create 3d remap plan
The FFT setup in pppm failed.

Could not create numa grid of processors
The specified constraints did not allow this style of grid to be created. Usually this is because the total processor count is not a multiple of the cores/node or the user specified processor count is > 1 in one of the dimensions.

Could not create twolevel 3d grid of processors
The specified constraints did not allow this style of grid to be created.

Could not find atom_modify first group ID
Self-explanatory.

Could not find change_box group ID
Group ID used in the change_box command does not exist.

Could not find compute ID for PRD
Self-explanatory.

Could not find compute ID for TAD
Self-explanatory.

Could not find compute ID for temperature bias
Self-explanatory.

Could not find compute ID to delete
Self-explanatory.

Could not find compute displace/atom fix ID
Self-explanatory.

Could not find compute event/displace fix ID
Self-explanatory.

Could not find compute group ID
Self-explanatory.

Could not find compute heat/flux compute ID
Self-explanatory.

Could not find compute msd fix ID
Self-explanatory.

Could not find compute pressure temperature ID
The compute ID for calculating temperature does not exist.

Could not find compute_modify ID
Self-explanatory.

Could not find delete_atoms group ID
Group ID used in the delete_atoms command does not exist.

Could not find delete_atoms region ID
Region ID used in the delete_atoms command does not exist.

Could not find displace_atoms group ID
Group ID used in the displace_atoms command does not exist.

Could not find dump custom compute ID
The compute ID needed by dump custom to compute a per-atom quantity does not exist.

Could not find dump custom fix ID
Self-explanatory.

Could not find dump custom variable name
Self-explanatory.

Could not find dump group ID
A group ID used in the dump command does not exist.

Could not find dump local compute ID
Self-explanatory.

Could not find dump local fix ID
Self-explanatory.

Could not find dump modify compute ID
>   Self-explanatory.

Could not find dump modify fix ID
>   Self-explanatory.

Could not find dump modify variable name
>   Self-explanatory.

Could not find fix ID to delete
>   Self-explanatory.

Could not find fix gcmc rotation group ID
>   Self-explanatory.

Could not find fix group ID
>   A group ID used in the fix command does not exist.

Could not find fix msst compute ID
>   Self-explanatory.

Could not find fix poems group ID
>   A group ID used in the fix poems command does not exist.

Could not find fix recenter group ID
>   A group ID used in the fix recenter command does not exist.

Could not find fix rigid group ID
>   A group ID used in the fix rigid command does not exist.

Could not find fix srd group ID
>   Self-explanatory.

Could not find fix_modify ID
>   A fix ID used in the fix_modify command does not exist.

Could not find fix_modify pressure ID
>   The compute ID for computing pressure does not exist.

Could not find fix_modify temperature ID
>   The compute ID for computing temperature does not exist.

Could not find group delete group ID
>   Self-explanatory.

Could not find set group ID
>   Group ID specified in set command does not exist.

Could not find thermo compute ID
>   Compute ID specified in thermo_style command does not exist.

Could not find thermo custom compute ID
>   The compute ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom fix ID
>   The fix ID needed by thermo style custom to compute a requested quantity does not exist.

Could not find thermo custom variable name
>   Self-explanatory.

Could not find thermo fix ID
>   Fix ID specified in thermo_style command does not exist.

Could not find thermo variable name
>   Self-explanatory.

Could not find thermo_modify pressure ID
>   The compute ID needed by thermo style custom to compute pressure does not exist.

Could not find thermo_modify temperature ID
>   The compute ID needed by thermo style custom to compute temperature does not exist.

Could not find undump ID
>   A dump ID used in the undump command does not exist.

Could not find velocity group ID
>   A group ID used in the velocity command does not exist.

Could not find velocity temperature ID
>   The compute ID needed by the velocity command to compute temperature does not exist.

Could not find/initialize a specified accelerator device
        Could not initialize at least one of the devices specified for the gpu package
Could not grab element entry from EIM potential file
        Self-explanatory
Could not grab global entry from EIM potential file
        Self-explanatory.
Could not grab pair entry from EIM potential file
        Self-explanatory.
Coulomb PPPMDisp order < minimum allowed order
        The default minimum order is 2. This can be reset by the kspace_modify minorder command.
Coulomb cut not supported in pair_style buck/long/coul/coul
        Must use long-range Coulombic interactions.
Coulomb cut not supported in pair_style lj/long/coul/long
        Must use long-range Coulombic interactions.
Coulomb cut not supported in pair_style lj/long/tip4p/long
        Must use long-range Coulombic interactions.
Coulomb cutoffs of pair hybrid sub-styles do not match
        If using a Kspace solver, all Coulomb cutoffs of long pair styles must be the same.
Cound not find dump_modify ID
        Self-explanatory.
Create_atoms command before simulation box is defined
        The create_atoms command cannot be used before a read_data, read_restart, or create_box command.
Create_atoms region ID does not exist
        A region ID used in the create_atoms command does not exist.
Create_box region ID does not exist
        A region ID used in the create_box command does not exist.
Create_box region does not support a bounding box
        Not all regions represent bounded volumes. You cannot use such a region with the create_box
        command.
Cutoffs missing in pair_style buck/long/coul/long
        Self-exlanatory.
Cutoffs missing in pair_style lj/long/coul/long
        Self-explanatory.
Cyclic loop in joint connections
        Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a ring (or cycle).
Degenerate lattice primitive vectors
        Invalid set of 3 lattice vectors for lattice command.
Delete region ID does not exist
        Self-explanatory.
Delete_atoms command before simulation box is defined
        The delete_atoms command cannot be used before a read_data, read_restart, or create_box command.
Delete_atoms cutoff > neighbor cutoff
        Cannot delete atoms further away than a processor knows about.
Delete_atoms requires a pair style be defined
        This is because atom deletion within a cutoff uses a pairwise neighbor list.
Delete_bonds command before simulation box is defined
        The delete_bonds command cannot be used before a read_data, read_restart, or create_box command.
Delete_bonds command with no atoms existing
        No atoms are yet defined so the delete_bonds command cannot be used.
Deposition region extends outside simulation box
        Self-explanatory.
Did not assign all atoms correctly
        Atoms read in from a data file were not assigned correctly to processors. This is likely due to some
        atom coordinates being outside a non-periodic simulation box.

Did not find all elements in MEAM library file
> The requested elements were not found in the MEAM file.

Did not find fix shake partner info
> Could not find bond partners implied by fix shake command. This error can be triggered if the delete_bonds command was used before fix shake, and it removed bonds without resetting the 1-2, 1-3, 1-4 weighting list via the special keyword.

Did not find keyword in table file
> Keyword used in pair_coeff command was not found in table file.

Did not set pressure for fix rigid/nph
> The press keyword must be specified.

Did not set temperature for fix rigid/nvt
> The temp keyword must be specified.

Did not set temperature or pressure for fix rigid/npt
> The temp and press keywords must be specified.

Dihedral atom missing in delete_bonds
> The delete_bonds command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atom missing in set command
> The set command cannot find one or more atoms in a particular dihedral on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid dihedral.

Dihedral atoms %d %d %d %d missing on proc %d at step %ld
> One or more of 4 atoms needed to compute a particular dihedral are missing on this processor. Typically this is because the pairwise cutoff is set too short or the dihedral has blown apart and an atom is too far away.

Dihedral charmm is incompatible with Pair style
> Dihedral style charmm must be used with a pair style charmm in order for the 1-4 epsilon/sigma parameters to be defined.

Dihedral coeff for hybrid has invalid style
> Dihedral style hybrid uses another dihedral style as one of its coefficients. The dihedral style used in the dihedral_coeff command or read from a restart file is not recognized.

Dihedral coeffs are not set
> No dihedral coefficients have been assigned in the data file or via the dihedral_coeff command.

Dihedral style hybrid cannot have hybrid as an argument
> Self-explanatory.

Dihedral style hybrid cannot have none as an argument
> Self-explanatory.

Dihedral style hybrid cannot use same dihedral style twice
> Self-explanatory.

Dihedral/improper extent > half of periodic box length
> This error was detected by the neigh_modify check yes setting. It is an error because the dihedral atoms are so far apart it is ambiguous how it should be defined.

Dihedral_coeff command before dihedral_style is defined
> Coefficients cannot be set in the data file or via the dihedral_coeff command until an dihedral_style has been assigned.

Dihedral_coeff command before simulation box is defined
> The dihedral_coeff command cannot be used before a read_data, read_restart, or create_box command.

Dihedral_coeff command when no dihedrals allowed
> The chosen atom style does not allow for dihedrals to be defined.

Dihedral_style command when no dihedrals allowed
> The chosen atom style does not allow for dihedrals to be defined.

Dihedrals assigned incorrectly
> Dihedrals read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Dihedrals defined but no dihedral types
> The data file header lists dihedrals but no dihedral types.

Dimension command after simulation box is defined
> The dimension command cannot be used after a read_data, read_restart, or create_box command.

Dispersion PPPMDisp order has been reduced below minorder
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the dipsersion order.

Displace_atoms command before simulation box is defined
> The displace_atoms command cannot be used before a read_data, read_restart, or create_box command.

Distance must be > 0 for compute event/displace
> Self-explanatory.

Divide by 0 in influence function of pair peri/lps
> This should not normally occur. It is likely a problem with your model.

Divide by 0 in variable formula
> Self-explanatory.

Domain too large for neighbor bins
> The domain has become extremely large so that neighbor bins cannot be used. Most likely, one or more atoms have been blown out of the simulation box to a great distance.

Double precision is not supported on this accelerator
> Self-explanatory

Dump cfg arguments can not mix xs|ys|zs with xsu|ysu|zsu
> Self-explanatory.

Dump cfg arguments must start with 'id type xs ys zs' or 'id type xsu ysu zsu'
> This is a requirement of the CFG output format.

Dump cfg requires one snapshot per file
> Use the wildcard "*" character in the filename.

Dump custom and fix not computed at compatible times
> The fix must produce per-atom quantities on timesteps that dump custom needs them.

Dump custom compute does not calculate per-atom array
> Self-explanatory.

Dump custom compute does not calculate per-atom vector
> Self-explanatory.

Dump custom compute does not compute per-atom info
> Self-explanatory.

Dump custom compute vector is accessed out-of-range
> Self-explanatory.

Dump custom fix does not compute per-atom array
> Self-explanatory.

Dump custom fix does not compute per-atom info
> Self-explanatory.

Dump custom fix does not compute per-atom vector
> Self-explanatory.

Dump custom fix vector is accessed out-of-range
> Self-explanatory.

Dump custom variable is not atom-style variable
> Only atom-style variables generate per-atom quantities, needed for dump output.

Dump dcd of non-matching # of atoms
> Every snapshot written by dump dcd must contain the same # of atoms.

Dump dcd requires sorting by atom ID
> Use the dump_modify sort command to enable this.

Dump every variable returned a bad timestep
> The variable must return a timestep greater than the current timestep.

Dump file does not contain requested snapshot

Self-explanatory.
Dump file is incorrectly formatted
Self-explanatory.
Dump image bond not allowed with no bond types
Self-explanatory.
Dump image cannot perform sorting
Self-explanatory.
Dump image persp option is not yet supported
Self-explanatory.
Dump image requires one snapshot per file
Use a "*" in the filename.
Dump local and fix not computed at compatible times
The fix must produce per-atom quantities on timesteps that dump local needs them.
Dump local attributes contain no compute or fix
Self-explanatory.
Dump local cannot sort by atom ID
This is because dump local does not really dump per-atom info.
Dump local compute does not calculate local array
Self-explanatory.
Dump local compute does not calculate local vector
Self-explanatory.
Dump local compute does not compute local info
Self-explanatory.
Dump local compute vector is accessed out-of-range
Self-explanatory.
Dump local count is not consistent across input fields
Every column of output must be the same length.
Dump local fix does not compute local array
Self-explanatory.
Dump local fix does not compute local info
Self-explanatory.
Dump local fix does not compute local vector
Self-explanatory.
Dump local fix vector is accessed out-of-range
Self-explanatory.
Dump modify bcolor not allowed with no bond types
Self-explanatory.
Dump modify bdiam not allowed with no bond types
Self-explanatory.
Dump modify compute ID does not compute per-atom array
Self-explanatory.
Dump modify compute ID does not compute per-atom info
Self-explanatory.
Dump modify compute ID does not compute per-atom vector
Self-explanatory.
Dump modify compute ID vector is not large enough
Self-explanatory.
Dump modify element names do not match atom types
Number of element names must equal number of atom types.
Dump modify fix ID does not compute per-atom array
Self-explanatory.
Dump modify fix ID does not compute per-atom info
Self-explanatory.
Dump modify fix ID does not compute per-atom vector

Self-explanatory.

Dump modify fix ID vector is not large enough

Self-explanatory.

Dump modify variable is not atom-style variable

Self-explanatory.

Dump sort column is invalid

Self-explanatory.

Dump xtc requires sorting by atom ID

Use the dump_modify sort command to enable this.

Dump_modify format string is too short

There are more fields to be dumped in a line of output than your format string specifies.

Dump_modify region ID does not exist

Self-explanatory.

Dumping an atom property that isn't allocated

The chosen atom style does not define the per-atom quantity being dumped.

Dumping an atom quantity that isn't allocated

Only per-atom quantities that are defined for the atom style being used are allowed.

Duplicate fields in read_dump command

Self-explanatory.

Duplicate particle in PeriDynamic bond - simulation box is too small

This is likely because your box length is shorter than 2 times the bond length.

Electronic temperature dropped below zero

Something has gone wrong with the fix ttm electron temperature model.

Empty brackets in variable

There is no variable syntax that uses empty brackets. Check the variable doc page.

Energy was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Epsilon or sigma reference not set by pair style in PPPMDisp

The pair style is not providing the needed epsilon or sigma values.

Epsilon or sigma reference not set by pair style in ewald/n

The pair style is not providing the needed epsilon or sigma values.

Expected floating point parameter in input script or data file

The quantity being read is an integer on non-numeric value.

Expected floating point parameter in variable definition

The quantity being read is a non-numeric value.

Expected integer parameter in input script or data file

The quantity being read is a floating point or non-numeric value.

Expected integer parameter in variable definition

The quantity being read is a floating point or non-numeric value.

Failed to allocate %ld bytes for array %s

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Failed to reallocate %ld bytes for array %s

Your LAMMPS simulation has run out of memory. You need to run a smaller simulation or on more processors.

Fewer SRD bins than processors in some dimension

This is not allowed. Make your SRD bin size smaller.

File variable could not read value

Check the file assigned to the variable.

Final box dimension due to fix deform is < 0.0

Self-explanatory.

Fix GPU split must be positive for hybrid pair styles

Self-explanatory.

Fix ID for compute atom/molecule does not exist
        Self-explanatory.
Fix ID for compute reduce does not exist
        Self-explanatory.
Fix ID for compute slice does not exist
        Self-explanatory.
Fix ID for fix ave/atom does not exist
        Self-explanatory.
Fix ID for fix ave/correlate does not exist
        Self-explanatory.
Fix ID for fix ave/histo does not exist
        Self-explanatory.
Fix ID for fix ave/spatial does not exist
        Self-explanatory.
Fix ID for fix ave/time does not exist
        Self-explanatory.
Fix ID for fix store/state does not exist
        Self-explanatory
Fix ID for read_data does not exist
        Self-explanatory.
Fix ID must be alphanumeric or underscore characters
        Self-explanatory.
Fix SRD no-slip requires atom attribute torque
        This is because the SRD collisions will impart torque to the solute particles.
Fix SRD: bad bin assignment for SRD advection
        Something has gone wrong in your SRD model; try using more conservative settings.
Fix SRD: bad search bin assignment
        Something has gone wrong in your SRD model; try using more conservative settings.
Fix SRD: bad stencil bin for big particle
        Something has gone wrong in your SRD model; try using more conservative settings.
Fix SRD: too many big particles in bin
        Reset the ATOMPERBIN parameter at the top of fix_srd.cpp to a larger value, and re-compile the
        code.
Fix SRD: too many walls in bin
        This should not happen unless your system has been setup incorrectly.
Fix adapt kspace style does not exist
        Self-explanatory.
Fix adapt pair style does not exist
        Self-explanatory
Fix adapt pair style param not supported
        The pair style does not know about the parameter you specified.
Fix adapt requires atom attribute charge
        The atom style being used does not specify an atom charge.
Fix adapt requires atom attribute diameter
        The atom style being used does not specify an atom diameter.
Fix adapt type pair range is not valid for pair hybrid sub-style
        Self-explanatory.
Fix append/atoms requires a lattice be defined
        Use the lattice command for this purpose.
Fix ave/atom compute array is accessed out-of-range
        Self-explanatory.
Fix ave/atom compute does not calculate a per-atom array
        Self-explanatory.
Fix ave/atom compute does not calculate a per-atom vector

A compute used by fix ave/atom must generate per-atom values.

Fix ave/atom compute does not calculate per-atom values

A compute used by fix ave/atom must generate per-atom values.

Fix ave/atom fix array is accessed out-of-range

Self-explanatory.

Fix ave/atom fix does not calculate a per-atom array

Self-explanatory.

Fix ave/atom fix does not calculate a per-atom vector

A fix used by fix ave/atom must generate per-atom values.

Fix ave/atom fix does not calculate per-atom values

A fix used by fix ave/atom must generate per-atom values.

Fix ave/atom missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/atom variable is not atom-style variable

A variable used by fix ave/atom must generate per-atom values.

Fix ave/correlate compute does not calculate a scalar

Self-explanatory.

Fix ave/correlate compute does not calculate a vector

Self-explanatory.

Fix ave/correlate compute vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/correlate fix does not calculate a scalar

Self-explanatory.

Fix ave/correlate fix does not calculate a vector

Self-explanatory.

Fix ave/correlate fix vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/correlate missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/correlate variable is not equal-style variable

Self-explanatory.

Fix ave/histo cannot input local values in scalar mode

Self-explanatory.

Fix ave/histo cannot input per-atom values in scalar mode

Self-explanatory.

Fix ave/histo compute array is accessed out-of-range

Self-explanatory.

Fix ave/histo compute does not calculate a global array

Self-explanatory.

Fix ave/histo compute does not calculate a global scalar

Self-explanatory.

Fix ave/histo compute does not calculate a global vector

Self-explanatory.

Fix ave/histo compute does not calculate a local array

Self-explanatory.

Fix ave/histo compute does not calculate a local vector

Self-explanatory.

Fix ave/histo compute does not calculate a per-atom array

Self-explanatory.

Fix ave/histo compute does not calculate a per-atom vector

Self-explanatory.

Fix ave/histo compute does not calculate local values

Self-explanatory.

Fix ave/histo compute does not calculate per-atom values

Self-explanatory.

Fix ave/histo compute vector is accessed out-of-range
Self-explanatory.

Fix ave/histo fix array is accessed out-of-range
Self-explanatory.

Fix ave/histo fix does not calculate a global array
Self-explanatory.

Fix ave/histo fix does not calculate a global scalar
Self-explanatory.

Fix ave/histo fix does not calculate a global vector
Self-explanatory.

Fix ave/histo fix does not calculate a local array
Self-explanatory.

Fix ave/histo fix does not calculate a local vector
Self-explanatory.

Fix ave/histo fix does not calculate a per-atom array
Self-explanatory.

Fix ave/histo fix does not calculate a per-atom vector
Self-explanatory.

Fix ave/histo fix does not calculate local values
Self-explanatory.

Fix ave/histo fix does not calculate per-atom values
Self-explanatory.

Fix ave/histo fix vector is accessed out-of-range
Self-explanatory.

Fix ave/histo input is invalid compute
Self-explanatory.

Fix ave/histo input is invalid fix
Self-explanatory.

Fix ave/histo input is invalid variable
Self-explanatory.

Fix ave/histo inputs are not all global, peratom, or local
All inputs in a single fix ave/histo command must be of the same style.

Fix ave/histo missed timestep
You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/spatial compute does not calculate a per-atom array
Self-explanatory.

Fix ave/spatial compute does not calculate a per-atom vector
A compute used by fix ave/spatial must generate per-atom values.

Fix ave/spatial compute does not calculate per-atom values
A compute used by fix ave/spatial must generate per-atom values.

Fix ave/spatial compute vector is accessed out-of-range
The index for the vector is out of bounds.

Fix ave/spatial fix does not calculate a per-atom array
Self-explanatory.

Fix ave/spatial fix does not calculate a per-atom vector
A fix used by fix ave/spatial must generate per-atom values.

Fix ave/spatial fix does not calculate per-atom values
A fix used by fix ave/spatial must generate per-atom values.

Fix ave/spatial fix vector is accessed out-of-range
The index for the vector is out of bounds.

Fix ave/spatial for triclinic boxes requires units reduced
Self-explanatory.

Fix ave/spatial missed timestep

Errors: 1070

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/spatial settings invalid with changing box

If the ave setting is "running" or "window" and the box size/shape changes during the simulation, then the units setting must be "reduced", else the number of bins may change.

Fix ave/spatial variable is not atom-style variable

A variable used by fix ave/spatial must generate per-atom values.

Fix ave/time cannot set output array intensive/extensive from these inputs

One of more of the vector inputs has individual elements which are flagged as intensive or extensive. Such an input cannot be flagged as all intensive/extensive when turned into an array by fix ave/time.

Fix ave/time cannot use variable with vector mode

Variables produce scalar values.

Fix ave/time columns are inconsistent lengths

Self-explanatory.

Fix ave/time compute array is accessed out-of-range

An index for the array is out of bounds.

Fix ave/time compute does not calculate a scalar

Self-explantory.

Fix ave/time compute does not calculate a vector

Self-explantory.

Fix ave/time compute does not calculate an array

Self-explanatory.

Fix ave/time compute vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/time fix array is accessed out-of-range

An index for the array is out of bounds.

Fix ave/time fix does not calculate a scalar

Self-explanatory.

Fix ave/time fix does not calculate a vector

Self-explanatory.

Fix ave/time fix does not calculate an array

Self-explanatory.

Fix ave/time fix vector is accessed out-of-range

The index for the vector is out of bounds.

Fix ave/time missed timestep

You cannot reset the timestep to a value beyond where the fix expects to next perform averaging.

Fix ave/time variable is not equal-style variable

Self-explanatory.

Fix balance string is invalid

The string can only contain the characters "x", "y", or "z".

Fix balance string is invalid for 2d simulation

The string cannot contain the letter "z".

Fix bond/break requires special_bonds = 0,1,1

This is a restriction of the current fix bond/break implementation.

Fix bond/create cutoff is longer than pairwise cutoff

This is not allowed because bond creation is done using the pairwise neighbor list.

Fix bond/create requires special_bonds coul = 0,1,1

Self-explanatory.

Fix bond/create requires special_bonds lj = 0,1,1

Self-explanatory.

Fix bond/swap cannot use dihedral or improper styles

These styles cannot be defined when using this fix.

Fix bond/swap requires pair and bond styles

Self-explanatory.

Fix bond/swap requires special_bonds = 0,1,1

Self-explanatory.

Fix box/relax generated negative box length

The pressure being applied is likely too large. Try applying it incrementally, to build to the high pressure.

Fix command before simulation box is defined

The fix command cannot be used before a read_data, read_restart, or create_box command.

Fix deform cannot use yz variable with xy

The yz setting cannot be a variable if xy deformation is also specified. This is because LAMMPS cannot determine if the yz setting will induce a box flip which would be invalid if xy is also changing.

Fix deform is changing yz too much with xy

When both yz and xy are changing, it induces changes in xz if the box must flip from one tilt extreme to another. Thus it is not allowed for yz to grow so much that a flip is induced.

Fix deform tilt factors require triclinic box

Cannot deform the tilt factors of a simulation box unless it is a triclinic (non-orthogonal) box.

Fix deform volume setting is invalid

Cannot use volume style unless other dimensions are being controlled.

Fix deposit region cannot be dynamic

Only static regions can be used with fix deposit.

Fix deposit region does not support a bounding box

Not all regions represent bounded volumes. You cannot use such a region with the fix deposit command.

Fix efield requires atom attribute q

Self-explanatory.

Fix evaporate molecule requires atom attribute molecule

The atom style being used does not define a molecule ID.

Fix external callback function not set

This must be done by an external program in order to use this fix.

Fix for fix ave/atom not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/atom is requesting a value on a non-allowed timestep.

Fix for fix ave/correlate not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/correlate is requesting a value on a non-allowed timestep.

Fix for fix ave/histo not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/histo is requesting a value on a non-allowed timestep.

Fix for fix ave/spatial not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/spatial is requesting a value on a non-allowed timestep.

Fix for fix ave/time not computed at compatible time

Fixes generate their values on specific timesteps. Fix ave/time is requesting a value on a non-allowed timestep.

Fix for fix store/state not computed at compatible time

Fixes generate their values on specific timesteps. Fix store/state is requesting a value on a non-allowed timestep.

Fix freeze requires atom attribute torque

The atom style defined does not have this attribute.

Fix gcmc cannot exchange individual atoms belonging to a molecule

This is an error since you should not delete only one atom of a molecule. The user has specified atomic (non-molecular) gas exchanges, but an atom belonging to a molecule could be deleted.

Fix gcmc could not find any atoms in the user-supplied template molecule

When using the molecule option with fix gcmc, the user must supply a template molecule in the usual LAMMPS data file with its molecule id specified in the fix gcmc command as the "type" of the exchanged gas.

Fix gcmc incompatible with given pair_style
> Some pair_styles do not provide single-atom energies, which are needed by fix gcmc.

Fix gcmc incorrect number of atoms per molecule
> The number of atoms in each gas molecule was not computed correctly.

Fix gcmc molecule command requires that atoms have molecule attributes
> Should not choose the GCMC molecule feature if no molecules are being simulated. The general molecule flag is off, but GCMC's molecule flag is on.

Fix gcmc ran out of available molecule IDs
> This is a code limitation where more than MAXSMALLINT (usually around two billion) molecules have been created. The code needs to be modified to either allow molecule ID recycling or use bigger ints for molecule IDs. A work-around is to run shorter simulations.

Fix gcmc region cannot be dynamic
> Only static regions can be used with fix gcmc.

Fix gcmc region does not support a bounding box
> Not all regions represent bounded volumes. You cannot use such a region with the fix gcmc command.

Fix gcmc region extends outside simulation box
> Self-explanatory.

Fix heat group has no atoms
> Self-explanatory.

Fix heat kinetic energy of an atom went negative
> This will cause the velocity rescaling about to be performed by fix heat to be invalid.

Fix heat kinetic energy went negative
> This will cause the velocity rescaling about to be performed by fix heat to be invalid.

Fix in variable not computed at compatible time
> Fixes generate their values on specific timesteps. The variable is requesting the values on a non-allowed timestep.

Fix langevin angmom requires atom style ellipsoid
> Self-explanatory.

Fix langevin angmom requires extended particles
> This fix option cannot be used with point paritlces.

Fix langevin omega requires atom style sphere
> Self-explanatory.

Fix langevin omega requires extended particles
> One of the particles has radius 0.0.

Fix langevin period must be > 0.0
> The time window for temperature relaxation must be > 0

Fix langevin variable returned negative temperature
> Self-explanatory.

Fix momentum group has no atoms
> Self-explanatory.

Fix move cannot define z or vz variable for 2d problem
> Self-explanatory.

Fix move cannot rotate aroung non z-axis for 2d problem
> Self-explanatory.

Fix move cannot set linear z motion for 2d problem
> Self-explanatory.

Fix move cannot set wiggle z motion for 2d problem
> Self-explanatory.

Fix msst compute ID does not compute potential energy
> Self-explanatory.

Fix msst compute ID does not compute pressure
> Self-explanatory.

Fix msst compute ID does not compute temperature

Self-explanatory.

Fix msst requires a periodic box

Self-explanatory.

Fix msst tscale must satisfy 0 <= tscale < 1

Self-explanatory.

Fix npt/nph has tilted box too far in one step - periodic cell is too far from equilibrium state

Self-explanatory. The change in the box tilt is too extreme on a short timescale.

Fix nve/asphere requires extended particles

This fix can only be used for particles with a shape setting.

Fix nve/asphere/noforce requires atom style ellipsoid

Self-explanatory.

Fix nve/asphere/noforce requires extended particles

One of the particles is not an ellipsoid.

Fix nve/body requires atom style body

Self-explanatory.

Fix nve/body requires bodies

This fix can only be used for particles that are bodies.

Fix nve/line can only be used for 2d simulations

Self-explanatory.

Fix nve/line requires atom style line

Self-explanatory.

Fix nve/line requires line particles

Self-explanatory.

Fix nve/sphere requires atom attribute mu

An atom style with this attribute is needed.

Fix nve/sphere requires atom style sphere

Self-explanatory.

Fix nve/sphere requires extended particles

This fix can only be used for particles of a finite size.

Fix nve/tri can only be used for 3d simulations

Self-explanatory.

Fix nve/tri requires atom style tri

Self-explanatory.

Fix nve/tri requires tri particles

Self-explanatory.

Fix nvt/nph/npt asphere requires extended particles

The shape setting for a particle in the fix group has shape = 0.0, which means it is a point particle.

Fix nvt/nph/npt sphere requires atom style sphere

Self-explanatory.

Fix nvt/npt/nph damping parameters must be > 0.0

Self-explanatory.

Fix nvt/npt/nph dilate group ID does not exist

Self-explanatory.

Fix nvt/sphere requires extended particles

This fix can only be used for particles of a finite size.

Fix orient/fcc file open failed

The fix orient/fcc command could not open a specified file.

Fix orient/fcc file read failed

The fix orient/fcc command could not read the needed parameters from a specified file.

Fix orient/fcc found self twice

The neighbor lists used by fix orient/fcc are messed up. If this error occurs, it is likely a bug, so send an email to the developers.

Fix peri neigh does not exist

Somehow a fix that the pair style defines has been deleted.

Fix pour region ID does not exist
>Self-explanatory.

Fix pour region cannot be dynamic
>Only static regions can be used with fix pour.

Fix pour region does not support a bounding box
>Not all regions represent bounded volumes. You cannot use such a region with the fix pour command.

Fix pour requires atom attributes radius, rmass
>The atom style defined does not have these attributes.

Fix press/berendsen damping parameters must be > 0.0
>Self-explanatory.

Fix qeq/comb group has no atoms
>Self-explanatory.

Fix qeq/comb requires atom attribute q
>An atom style with charge must be used to perform charge equilibration.

Fix reax/bonds numbonds > nsbmax_most
>The limit of the number of bonds expected by the ReaxFF force field was exceeded.

Fix recenter group has no atoms
>Self-explanatory.

Fix restrain requires an atom map, see atom_modify
>Self-explanatory.

Fix rigid atom has non-zero image flag in a non-periodic dimension
>Image flags for non-periodic dimensions should not be set.

Fix rigid langevin period must be > 0.0
>Self-explanatory.

Fix rigid molecule requires atom attribute molecule
>Self-explanatory.

Fix rigid npt/nph dilate group ID does not exist
>Self-explanatory.

Fix rigid npt/nph does not yet allow triclinic box
>Self-explanatory.

Fix rigid npt/nph period must be > 0.0
>Self-explanatory.

Fix rigid nvt/npt/nph damping parameters must be > 0.0
>Self-explanatory.

Fix rigid xy torque cannot be on for 2d simulation
>Self-explanatory.

Fix rigid z force cannot be on for 2d simulation
>Self-explanatory.

Fix rigid/npt period must be > 0.0
>Self-explanatory.

Fix rigid/npt temperature order must be 3 or 5
>Self-explanatory.

Fix rigid/nvt period must be > 0.0
>Self-explanatory.

Fix rigid/nvt temperature order must be 3 or 5
>Self-explanatory.

Fix rigid/small atom has non-zero image flag in a non-periodic dimension
>Image flags for non-periodic dimensions should not be set.

Fix rigid/small langevin period must be > 0.0
>Self-explanatory.

Fix rigid/small requires atom attribute molecule
>Self-explanatory.

Fix rigid: Bad principal moments
>The principal moments of inertia computed for a rigid body are not within the required tolerances.

Fix shake cannot be used with minimization
> Cannot use fix shake while doing an energy minimization since it turns off bonds that should contribute to the energy.

Fix spring couple group ID does not exist
> Self-explanatory.

Fix srd lamda must be >= 0.6 of SRD grid size
> This is a requirement for accuracy reasons.

Fix srd requires SRD particles all have same mass
> Self-explanatory.

Fix srd requires ghost atoms store velocity
> Use the communicate vel yes command to enable this.

Fix srd requires newton pair on
> Self-explanatory.

Fix store/state compute array is accessed out-of-range
> Self-explanatory.

Fix store/state compute does not calculate a per-atom array
> The compute calculates a per-atom vector.

Fix store/state compute does not calculate a per-atom vector
> The compute calculates a per-atom vector.

Fix store/state compute does not calculate per-atom values
> Computes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state fix array is accessed out-of-range
> Self-explanatory.

Fix store/state fix does not calculate a per-atom array
> The fix calculates a per-atom vector.

Fix store/state fix does not calculate a per-atom vector
> The fix calculates a per-atom array.

Fix store/state fix does not calculate per-atom values
> Fixes that calculate global or local quantities cannot be used with fix store/state.

Fix store/state for atom property that isn't allocated
> Self-explanatory.

Fix store/state variable is not atom-style variable
> Only atom-style variables calculate per-atom quantities.

Fix temp/berendsen period must be > 0.0
> Self-explanatory.

Fix temp/berendsen variable returned negative temperature
> Self-explanatory.

Fix temp/rescale variable returned negative temperature
> Self-explanatory.

Fix thermal/conductivity swap value must be positive
> Self-explanatory.

Fix tmd must come after integration fixes
> Any fix tmd command must appear in the input script after all time integration fixes (nve, nvt, npt). See the fix tmd documentation for details.

Fix ttm electron temperatures must be > 0.0
> Self-explanatory.

Fix ttm electronic_density must be > 0.0
> Self-explanatory.

Fix ttm electronic_specific_heat must be > 0.0
> Self-explanatory.

Fix ttm electronic_thermal_conductivity must be >= 0.0
> Self-explanatory.

Fix ttm gamma_p must be > 0.0
> Self-explanatory.

Fix ttm gamma_s must be >= 0.0
    Self-explanatory.
Fix ttm number of nodes must be > 0
    Self-explanatory.
Fix ttm v_0 must be >= 0.0
    Self-explanatory.
Fix used in compute atom/molecule not computed at compatible time
    The fix must produce per-atom quantities on timesteps that the compute needs them.
Fix used in compute reduce not computed at compatible time
    Fixes generate their values on specific timesteps. Compute reduce is requesting a value on a
    non-allowed timestep.
Fix used in compute slice not computed at compatible time
    Fixes generate their values on specific timesteps. Compute slice is requesting a value on a
    non-allowed timestep.
Fix viscosity swap value must be positive
    Self-explanatory.
Fix viscosity vtarget value must be positive
    Self-explanatory.
Fix wall cutoff <= 0.0
    Self-explanatory.
Fix wall/colloid requires atom style sphere
    Self-explanatory.
Fix wall/colloid requires extended particles
    One of the particles has radius 0.0.
Fix wall/gran is incompatible with Pair style
    Must use a granular pair style to define the parameters needed for this fix.
Fix wall/gran requires atom style sphere
    Self-explanatory.
Fix wall/piston command only available at zlo
    The face keyword must be zlo.
Fix wall/region colloid requires atom style sphere
    Self-explanatory.
Fix wall/region colloid requires extended particles
    One of the particles has radius 0.0.
Fix wall/region cutoff <= 0.0
    Self-explanatory.
Fix_modify pressure ID does not compute pressure
    The compute ID assigned to the fix must compute pressure.
Fix_modify temperature ID does not compute temperature
    The compute ID assigned to the fix must compute temperature.
For triclinic deformation, specified target stress must be hydrostatic
    Triclinic pressure control is allowed using the tri keyword, but non-hydrostatic pressure control can
    not be used in this case.
Found no restart file matching pattern
    When using a "*" in the restart file name, no matching file was found.
GPU library not compiled for this accelerator
    Self-explanatory.
GPU particle split must be set to 1 for this pair style.
    For this pair style, you cannot run part of the force calculation on the host. See the package command.
Gmask function in equal-style variable formula
    Gmask is per-atom operation.
Gravity changed since fix pour was created
    Gravity must be static and not dynamic for use with fix pour.
Gravity must point in -y to use with fix pour in 2d

Gravity must be pointing "down" in a 2d box.

Gravity must point in -z to use with fix pour in 3d

Gravity must be pointing "down" in a 3d box, i.e. theta = 180.0.

Grmask function in equal-style variable formula

Grmask is per-atom operation.

Group ID does not exist

A group ID used in the group command does not exist.

Group ID in variable formula does not exist

Self-explanatory.

Group command before simulation box is defined

The group command cannot be used before a read_data, read_restart, or create_box command.

Group region ID does not exist

A region ID used in the group command does not exist.

If read_dump purges it cannot replace or trim

These operations are not compatible. See the read_dump doc page for details.

Illegal ... command

Self-explanatory. Check the input script syntax and compare to the documentation for the command. You can use -echo screen as a command-line option when running LAMMPS to see the offending line.

Illegal COMB parameter

One or more of the coefficients defined in the potential file is invalid.

Illegal Stillinger-Weber parameter

One or more of the coefficients defined in the potential file is invalid.

Illegal Tersoff parameter

One or more of the coefficients defined in the potential file is invalid.

Illegal fix gcmc gas mass <= 0

The computed mass of the designated gas molecule or atom type was less than or equal to zero.

Illegal fix wall/piston velocity

The piston velocity must be positive.

Illegal integrate style

Self-explanatory.

Illegal number of angle table entries

There must be at least 2 table entries.

Illegal number of bond table entries

There must be at least 2 table entries.

Illegal number of pair table entries

There must be at least 2 table entries.

Illegal simulation box

The lower bound of the simulation box is greater than the upper bound.

Improper atom missing in delete_bonds

The delete_bonds command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atom missing in set command

The set command cannot find one or more atoms in a particular improper on a particular processor. The pairwise cutoff is too short or the atoms are too far apart to make a valid improper.

Improper atoms %d %d %d %d missing on proc %d at step %ld

One or more of 4 atoms needed to compute a particular improper are missing on this processor. Typically this is because the pairwise cutoff is set too short or the improper has blown apart and an atom is too far away.

Improper coeff for hybrid has invalid style

Improper style hybrid uses another improper style as one of its coefficients. The improper style used in the improper_coeff command or read from a restart file is not recognized.

Improper coeffs are not set

No improper coefficients have been assigned in the data file or via the improper_coeff command.

Improper style hybrid cannot have hybrid as an argument
> Self-explanatory.

Improper style hybrid cannot have none as an argument
> Self-explanatory.

Improper style hybrid cannot use same improper style twice
> Self-explanatory.

Improper_coeff command before improper_style is defined
> Coefficients cannot be set in the data file or via the improper_coeff command until an improper_style has been assigned.

Improper_coeff command before simulation box is defined
> The improper_coeff command cannot be used before a read_data, read_restart, or create_box command.

Improper_coeff command when no impropers allowed
> The chosen atom style does not allow for impropers to be defined.

Improper_style command when no impropers allowed
> The chosen atom style does not allow for impropers to be defined.

Impropers assigned incorrectly
> Impropers read in from the data file were not assigned correctly to atoms. This means there is something invalid about the topology definitions.

Impropers defined but no improper types
> The data file header lists improper but no improper types.

Inconsistent iparam/jparam values in fix bond/create command
> If itype and jtype are the same, then their maxbond and newtype settings must also be the same.

Inconsistent line segment in data file
> The end points of the line segment are not equal distances from the center point which is the atom coordinate.

Inconsistent triangle in data file
> The centroid of the triangle as defined by the corner points is not the atom coordinate.

Incorrect # of floating-point values in Bodies section of data file
> See doc page for body style.

Incorrect # of integer values in Bodies section of data file
> See doc page for body style.

Incorrect args for angle coefficients
> Self-explanatory. Check the input script or data file.

Incorrect args for bond coefficients
> Self-explanatory. Check the input script or data file.

Incorrect args for dihedral coefficients
> Self-explanatory. Check the input script or data file.

Incorrect args for improper coefficients
> Self-explanatory. Check the input script or data file.

Incorrect args for pair coefficients
> Self-explanatory. Check the input script or data file.

Incorrect args in pair_style command
> Self-explanatory.

Incorrect atom format in data file
> Number of values per atom line in the data file is not consistent with the atom style.

Incorrect bonus data format in data file
> See the read_data doc page for a description of how various kinds of bonus data must be formatted for certain atom styles.

Incorrect boundaries with slab Ewald
> Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with Ewald.

Incorrect boundaries with slab EwaldDisp
> Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with Ewald.

Incorrect boundaries with slab PPPM

Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with PPPM.

Incorrect boundaries with slab PPPMDisp

Must have periodic x,y dimensions and non-periodic z dimension to use 2d slab option with PPPM.

Incorrect element names in ADP potential file

The element names in the ADP file do not match those requested.

Incorrect element names in EAM potential file

The element names in the EAM file do not match those requested.

Incorrect format in COMB potential file

Incorrect number of words per line in the potential file.

Incorrect format in MEAM potential file

Incorrect number of words per line in the potential file.

Incorrect format in NEB coordinate file

Self-explanatory.

Incorrect format in Stillinger-Weber potential file

Incorrect number of words per line in the potential file.

Incorrect format in TMD target file

Format of file read by fix tmd command is incorrect.

Incorrect format in Tersoff potential file

Incorrect number of words per line in the potential file.

Incorrect integer value in Bodies section of data file

See doc page for body style.

Incorrect multiplicity arg for dihedral coefficients

Self-explanatory. Check the input script or data file.

Incorrect rigid body format in fix rigid file

The number of fields per line is not what expected.

Incorrect sign arg for dihedral coefficients

Self-explanatory. Check the input script or data file.

Incorrect velocity format in data file

Each atom style defines a format for the Velocity section of the data file. The read-in lines do not match.

Incorrect weight arg for dihedral coefficients

Self-explanatory. Check the input script or data file.

Index between variable brackets must be positive

Self-explanatory.

Indexed per-atom vector in variable formula without atom map

Accessing a value from an atom vector requires the ability to lookup an atom index, which is provided by an atom map. An atom map does not exist (by default) for non-molecular problems. Using the atom_modify map command will force an atom map to be created.

Initial temperatures not all set in fix ttm

Self-explantory.

Input line quote not followed by whitespace

An end quote must be followed by whitespace.

Insertion region extends outside simulation box

Region specified with fix pour command extends outside the global simulation box.

Insufficient Jacobi rotations for POEMS body

Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for body nparticle

Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for rigid body

Eigensolve for rigid body was not sufficiently accurate.

Insufficient Jacobi rotations for triangle

The calculation of the intertia tensor of the triangle failed. This should not happen if it is a reasonably shaped triangle.

Insufficient memory on accelerator

There is insufficient memory on one of the devices specified for the gpu package

Internal error in atom_style body

This error should not occur. Contact the developers.

Invalid -reorder N value

Self-explanatory.

Invalid Boolean syntax in if command

Self-explanatory.

Invalid REAX atom type

There is a mis-match between LAMMPS atom types and the elements listed in the ReaxFF force field file.

Invalid angle style

The choice of angle style is unknown.

Invalid angle table length

Length must be 2 or greater.

Invalid angle type in Angles section of data file

Angle type must be positive integer and within range of specified angle types.

Invalid angle type index for fix shake

Self-explanatory.

Invalid args for non-hybrid pair coefficients

"NULL" is only supported in pair_coeff calls when using pair hybrid

Invalid atom ID in Angles section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Atoms section of data file

Atom IDs must be positive integers.

Invalid atom ID in Bodies section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Bonds section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Bonus section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Dihedrals section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Impropers section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom ID in Velocities section of data file

Atom IDs must be positive integers and within range of defined atoms.

Invalid atom mass for fix shake

Mass specified in fix shake command must be > 0.0.

Invalid atom style

The choice of atom style is unknown.

Invalid atom type in Atoms section of data file

Atom types must range from 1 to specified # of types.

Invalid atom type in create_atoms command

The create_box command specified the range of valid atom types. An invalid type is being requested.

Invalid atom type in fix bond/create command

Self-explanatory.

Invalid atom type in fix gcmc command

The atom type specified in the GCMC command does not exist.

Invalid atom type in neighbor exclusion list

Atom types must range from 1 to Ntypes inclusive.

Invalid atom type index for fix shake

Atom types must range from 1 to Ntypes inclusive.

Invalid atom types in pair_write command

Atom types must range from 1 to Ntypes inclusive.

Invalid atom vector in variable formula

      The atom vector is not recognized.

Invalid atom_style body command

      No body style argument was provided.

Invalid atom_style command

      Self-explanatory.

Invalid attribute in dump custom command

      Self-explantory.

Invalid attribute in dump local command

      Self-explantory.

Invalid attribute in dump modify command

      Self-explantory.

Invalid body nparticle command

      Arguments in atom-style command are not correct.

Invalid body style

      The choice of body style is unknown.

Invalid bond style

      The choice of bond style is unknown.

Invalid bond table length

      Length must be 2 or greater.

Invalid bond type in Bonds section of data file

      Bond type must be positive integer and within range of specified bond types.

Invalid bond type in fix bond/break command

      Self-explanatory.

Invalid bond type in fix bond/create command

      Self-explanatory.

Invalid bond type index for fix shake

      Self-explanatory. Check the fix shake command in the input script.

Invalid coeffs for this dihedral style

      Cannot set class 2 coeffs in data file for this dihedral style.

Invalid color in dump_modify command

      The specified color name was not in the list of recognized colors. See the dump_modify doc page.

Invalid command-line argument

      One or more command-line arguments is invalid. Check the syntax of the command you are using to launch LAMMPS.

Invalid compute ID in variable formula

      The compute is not recognized.

Invalid compute style

      Self-explanatory.

Invalid cutoff in communicate command

      Specified cutoff must be >= 0.0.

Invalid cutoffs in pair_write command

      Inner cutoff must be larger than 0.0 and less than outer cutoff.

Invalid d1 or d2 value for pair colloid coeff

      Neither d1 or d2 can be < 0.

Invalid data file section: Angle Coeffs

      Atom style does not allow angles.

Invalid data file section: AngleAngle Coeffs

      Atom style does not allow impropers.

Invalid data file section: AngleAngleTorsion Coeffs

      Atom style does not allow dihedrals.

Invalid data file section: AngleTorsion Coeffs

      Atom style does not allow dihedrals.

Invalid data file section: Angles

Atom style does not allow angles.

Invalid data file section: Bodies

Atom style does not allow bodies.

Invalid data file section: Bond Coeffs

Atom style does not allow bonds.

Invalid data file section: BondAngle Coeffs

Atom style does not allow angles.

Invalid data file section: BondBond Coeffs

Atom style does not allow angles.

Invalid data file section: BondBond13 Coeffs

Atom style does not allow dihedrals.

Invalid data file section: Bonds

Atom style does not allow bonds.

Invalid data file section: Dihedral Coeffs

Atom style does not allow dihedrals.

Invalid data file section: Dihedrals

Atom style does not allow dihedrals.

Invalid data file section: Ellipsoids

Atom style does not allow ellipsoids.

Invalid data file section: EndBondTorsion Coeffs

Atom style does not allow dihedrals.

Invalid data file section: Improper Coeffs

Atom style does not allow impropers.

Invalid data file section: Impropers

Atom style does not allow impropers.

Invalid data file section: Lines

Atom style does not allow lines.

Invalid data file section: MiddleBondTorsion Coeffs

Atom style does not allow dihedrals.

Invalid data file section: Triangles

Atom style does not allow triangles.

Invalid delta_conf in tad command

The value must be between 0 and 1 inclusive.

Invalid density in Atoms section of data file

Density value cannot be <= 0.0.

Invalid diameter in set command

Self-explanatory.

Invalid dihedral style

The choice of dihedral style is unknown.

Invalid dihedral type in Dihedrals section of data file

Dihedral type must be positive integer and within range of specified dihedral types.

Invalid dipole length in set command

Self-explanatory.

Invalid displace_atoms rotate axis for 2d

Axis must be in z direction.

Invalid dump dcd filename

Filenames used with the dump dcd style cannot be binary or compressed or cause multiple files to be written.

Invalid dump frequency

Dump frequency must be 1 or greater.

Invalid dump image element name

The specified element name was not in the standard list of elements. See the dump_modify doc page.

Invalid dump image filename

The file produced by dump image cannot be binary and must be for a single processor.

Invalid dump image persp value

Persp value must be >= 0.0.

Invalid dump image theta value

Theta must be between 0.0 and 180.0 inclusive.

Invalid dump image zoom value

Zoom value must be > 0.0.

Invalid dump reader style

Self-explanatory.

Invalid dump style

The choice of dump style is unknown.

Invalid dump xtc filename

Filenames used with the dump xtc style cannot be binary or compressed or cause multiple files to be written.

Invalid dump xyz filename

Filenames used with the dump xyz style cannot be binary or cause files to be written by each processor.

Invalid dump_modify threshhold operator

Operator keyword used for threshold specification in not recognized.

Invalid entry in -reorder file

Self-explanatory.

Invalid fix ID in variable formula

The fix is not recognized.

Invalid fix ave/time off column

Self-explantory.

Invalid fix box/relax command for a 2d simulation

Fix box/relax styles involving the z dimension cannot be used in a 2d simulation.

Invalid fix box/relax command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix box/relax pressure settings

Settings for coupled dimensions must be the same.

Invalid fix nvt/npt/nph command for a 2d simulation

Cannot control z dimension in a 2d model.

Invalid fix nvt/npt/nph command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix nvt/npt/nph pressure settings

Settings for coupled dimensions must be the same.

Invalid fix press/berendsen for a 2d simulation

The z component of pressure cannot be controlled for a 2d model.

Invalid fix press/berendsen pressure settings

Settings for coupled dimensions must be the same.

Invalid fix rigid npt/nph command for a 2d simulation

Cannot control z dimension in a 2d model.

Invalid fix rigid npt/nph command pressure settings

If multiple dimensions are coupled, those dimensions must be specified.

Invalid fix rigid npt/nph pressure settings

Settings for coupled dimensions must be the same.

Invalid fix style

The choice of fix style is unknown.

Invalid flag in force field section of restart file

Unrecognized entry in restart file.

Invalid flag in header section of restart file

Unrecognized entry in restart file.

Invalid flag in type arrays section of restart file

Unrecognized entry in restart file.

Invalid format in Bodies section of data file
        The specified number of integer or floating point values does not appear.
Invalid frequency in temper command
        Nevery must be > 0.
Invalid group ID in neigh_modify command
        A group ID used in the neigh_modify command does not exist.
Invalid group function in variable formula
        Group function is not recognized.
Invalid group in communicate command
        Self-explanatory.
Invalid image color range
        The lo value in the range is larger than the hi value.
Invalid image up vector
        Up vector cannot be (0,0,0).
Invalid immediate variable
        Syntax of immediate value is incorrect.
Invalid improper style
        The choice of improper style is unknown.
Invalid improper type in Impropers section of data file
        Improper type must be positive integer and within range of specified improper types.
Invalid index for non-body particles in compute body/local command
        Only indices 1,2,3 can be used for non-body particles.
Invalid index in compute body/local command
        Self-explanatory.
Invalid keyword in angle table parameters
        Self-explanatory.
Invalid keyword in bond table parameters
        Self-explanatory.
Invalid keyword in compute angle/local command
        Self-explanatory.
Invalid keyword in compute bond/local command
        Self-explanatory.
Invalid keyword in compute dihedral/local command
        Self-explanatory.
Invalid keyword in compute improper/local command
        Self-explanatory.
Invalid keyword in compute pair/local command
        Self-explanatory.
Invalid keyword in compute property/atom command
        Self-explanatory.
Invalid keyword in compute property/local command
        Self-explanatory.
Invalid keyword in compute property/molecule command
        Self-explanatory.
Invalid keyword in dump cfg command
        Self-explanatory.
Invalid keyword in pair table parameters
        Keyword used in list of table parameters is not recognized.
Invalid keyword in thermo_style custom command
        One or more specified keywords are not recognized.
Invalid kspace style
        The choice of kspace style is unknown.
Invalid length in set command
        Self-explanatory.

Invalid mass in set command
Self-explanatory.
Invalid mass line in data file
Self-explanatory.
Invalid mass value
Self-explanatory.
Invalid math function in variable formula
Self-explanatory.
Invalid math/group/special function in variable formula
Self-explanatory.
Invalid option in lattice command for non-custom style
Certain lattice keywords are not supported unless the lattice style is "custom".
Invalid order of forces within respa levels
For respa, ordering of force computations within respa levels must obey certain rules. E.g. bonds cannot be compute less frequently than angles, pairwise forces cannot be computed less frequently than kspace, etc.
Invalid pair style
The choice of pair style is unknown.
Invalid pair table cutoff
Cutoffs in pair_coeff command are not valid with read-in pair table.
Invalid pair table length
Length of read-in pair table is invalid
Invalid partitions in processors part command
Valid partitions are numbered 1 to N and the sender and receiver cannot be the same partition.
Invalid radius in Atoms section of data file
Radius must be >= 0.0.
Invalid random number seed in fix ttm command
Random number seed must be > 0.
Invalid random number seed in set command
Random number seed must be > 0.
Invalid region style
The choice of region style is unknown.
Invalid replace values in compute reduce
Self-explanatory.
Invalid rigid body ID in fix rigid file
The ID does not match the number or an existing ID of rigid bodies that are defined by the fix rigid command.
Invalid run command N value
The number of timesteps must fit in a 32-bit integer. If you want to run for more steps than this, perform multiple shorter runs.
Invalid run command start/stop value
Self-explanatory.
Invalid run command upto value
Self-explanatory.
Invalid seed for Marsaglia random # generator
The initial seed for this random number generator must be a positive integer less than or equal to 900 million.
Invalid seed for Park random # generator
The initial seed for this random number generator must be a positive integer.
Invalid shape in Ellipsoids section of data file
Self-explanatory.
Invalid shape in Triangles section of data file
Two or more of the triangle corners are duplicate points.
Invalid shape in set command

Self-explanatory.

Invalid shear direction for fix wall/gran

Self-explanatory.

Invalid special function in variable formula

Self-explanatory.

Invalid style in pair_write command

Self-explanatory. Check the input script.

Invalid syntax in variable formula

Self-explanatory.

Invalid t_event in prd command

Self-explanatory.

Invalid t_event in tad command

The value must be greater than 0.

Invalid thermo keyword in variable formula

The keyword is not recognized.

Invalid tmax in tad command

The value must be greater than 0.0.

Invalid type for mass set

Mass command must set a type from 1-N where N is the number of atom types.

Invalid value in set command

The value specified for the setting is invalid, likely because it is too small or too large.

Invalid variable evaluation in variable formula

A variable used in a formula could not be evaluated.

Invalid variable in next command

Self-explanatory.

Invalid variable in special function next

Only file-style variables can be used with the next() function.

Invalid variable name

Variable name used in an input script line is invalid.

Invalid variable name in variable formula

Variable name is not recognized.

Invalid variable style with next command

Variable styles *equal* and *world* cannot be used in a next command.

Invalid wiggle direction for fix wall/gran

Self-explanatory.

Invoked angle equil angle on angle style none

Self-explanatory.

Invoked angle single on angle style none

Self-explanatory.

Invoked bond equil distance on bond style none

Self-explanatory.

Invoked bond single on bond style none

Self-explanatory.

Invoked pair single on pair style none

A command (e.g. a dump) attempted to invoke the single() function on a pair style none, which is illegal. You are probably attempting to compute per-atom quantities with an undefined pair style.

KIM neighbor iterator exceeded range

This should not happen. It likely indicates a bug in the KIM implementation of the interatomic potential where it is requesting neighbors incorrectly.

KSpace accuracy must be > 0

The kspace accuracy designated in the input must be greater than zero.

KSpace accuracy too large to estimate G vector

Reduce the accuracy request or specify gwald explicitly via the kspace_modify command.

KSpace accuracy too low

Requested accuracy must be less than 1.0.

KSpace solver requires a pair style

No pair style is defined.

KSpace style has not yet been set

Cannot use kspace_modify command until a kspace style is set.

KSpace style is incompatible with Pair style

Setting a kspace style requires that a pair style with a long-range Coulombic or dispersion component be used.

Keyword %s in MEAM parameter file not recognized

Self-explanatory.

Kspace style does not support compute group/group

Self-explanatory.

Kspace style pppm/disp/tip4p requires newton on

Self-explanatory.

Kspace style pppm/tip4p requires newton on

Self-explanatory.

Kspace style requires atom attribute q

The atom style defined does not have these attributes.

Kspace style with selected options requires atom attribute q

The atom style defined does not have these attributes. Change the atom style or switch of the coulomb solver.

LAMMPS unit_style lj not supported by KIM models

Self-explanatory. Check the input script or data file.

LJ6 off not supported in pair_style buck/long/coul/long

Self-exlanatory.

Label wasn't found in input script

Self-explanatory.

Lattice orient vectors are not orthogonal

The three specified lattice orientation vectors must be mutually orthogonal.

Lattice orient vectors are not right-handed

The three specified lattice orientation vectors must create a right-handed coordinate system such that a1 cross a2 = a3.

Lattice primitive vectors are collinear

The specified lattice primitive vectors do not for a unit cell with non-zero volume.

Lattice settings are not compatible with 2d simulation

One or more of the specified lattice vectors has a non-zero z component.

Lattice spacings are invalid

Each x,y,z spacing must be > 0.

Lattice style incompatible with simulation dimension

2d simulation can use sq, sq2, or hex lattice. 3d simulation can use sc, bcc, or fcc lattice.

Log of zero/negative value in variable formula

Self-explanatory.

Lost atoms via balance: original %ld current %ld

This should not occur. Report the problem to the developers.

Lost atoms: original %ld current %ld

Lost atoms are checked for each time thermo output is done. See the thermo_modify lost command for options. Lost atoms usually indicate bad dynamics, e.g. atoms have been blown far out of the simulation box, or moved futher than one processor's sub-domain away before reneighboring.

MEAM library error %d

A call to the MEAM Fortran library returned an error.

MPI_LMP_BIGINT and bigint in lmptype.h are not compatible

The size of the MPI datatype does not match the size of a bigint.

MPI_LMP_TAGINT and tagint in lmptype.h are not compatible

The size of the MPI datatype does not match the size of a tagint.

MSM grid is too large

The global MSM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 16384. You likely need to decrease the requested accuracy.

MSM order must be 4, 6, 8, or 10

This is a limitation of the MSM implementation in LAMMPS: the MSM order can only be 4, 6, 8, or 10.

Mass command before simulation box is defined

The mass command cannot be used before a read_data, read_restart, or create_box command.

Min_style command before simulation box is defined

The min_style command cannot be used before a read_data, read_restart, or create_box command.

Minimization could not find thermo_pe compute

This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.

Minimize command before simulation box is defined

The minimize command cannot be used before a read_data, read_restart, or create_box command.

Mismatched brackets in variable

Self-explanatory.

Mismatched compute in variable formula

A compute is referenced incorrectly or a compute that produces per-atom values is used in an equal-style variable formula.

Mismatched fix in variable formula

A fix is referenced incorrectly or a fix that produces per-atom values is used in an equal-style variable formula.

Mismatched variable in variable formula

A variable is referenced incorrectly or an atom-style variable that produces per-atom values is used in an equal-style variable formula.

Modulo 0 in variable formula

Self-explanatory.

Molecular data file has too many atoms

These kids of data files are currently limited to a number of atoms that fits in a 32-bit integer.

Molecule count changed in compute atom/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute com/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute gyration/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute inertia/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute msd/molecule

Number of molecules must remain constant over time.

Molecule count changed in compute property/molecule

Number of molecules must remain constant over time.

More than one fix deform

Only one fix deform can be defined at a time.

More than one fix freeze

Only one of these fixes can be defined, since the granular pair potentials access it.

More than one fix shake

Only one fix shake can be defined.

Must define angle_style before Angle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondAngle Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define angle_style before BondBond Coeffs

Must use an angle_style command before reading a data file that defines Angle Coeffs.

Must define bond_style before Bond Coeffs
> Must use a bond_style command before reading a data file that defines Bond Coeffs.

Must define dihedral_style before AngleAngleTorsion Coeffs
> Must use a dihedral_style command before reading a data file that defines AngleAngleTorsion Coeffs.

Must define dihedral_style before AngleTorsion Coeffs
> Must use a dihedral_style command before reading a data file that defines AngleTorsion Coeffs.

Must define dihedral_style before BondBond13 Coeffs
> Must use a dihedral_style command before reading a data file that defines BondBond13 Coeffs.

Must define dihedral_style before Dihedral Coeffs
> Must use a dihedral_style command before reading a data file that defines Dihedral Coeffs.

Must define dihedral_style before EndBondTorsion Coeffs
> Must use a dihedral_style command before reading a data file that defines EndBondTorsion Coeffs.

Must define dihedral_style before MiddleBondTorsion Coeffs
> Must use a dihedral_style command before reading a data file that defines MiddleBondTorsion Coeffs.

Must define improper_style before AngleAngle Coeffs
> Must use an improper_style command before reading a data file that defines AngleAngle Coeffs.

Must define improper_style before Improper Coeffs
> Must use an improper_style command before reading a data file that defines Improper Coeffs.

Must define pair_style before Pair Coeffs
> Must use a pair_style command before reading a data file that defines Pair Coeffs.

Must have more than one processor partition to temper
> Cannot use the temper command with only one processor partition. Use the -partition command-line option.

Must read Atoms before Angles
> The Atoms section of a data file must come before an Angles section.

Must read Atoms before Bodies
> The Atoms section of a data file must come before a Bodies section.

Must read Atoms before Bonds
> The Atoms section of a data file must come before a Bonds section.

Must read Atoms before Dihedrals
> The Atoms section of a data file must come before a Dihedrals section.

Must read Atoms before Ellipsoids
> The Atoms section of a data file must come before a Ellipsoids section.

Must read Atoms before Impropers
> The Atoms section of a data file must come before an Impropers section.

Must read Atoms before Lines
> The Atoms section of a data file must come before a Lines section.

Must read Atoms before Triangles
> The Atoms section of a data file must come before a Triangles section.

Must read Atoms before Velocities
> The Atoms section of a data file must come before a Velocities section.

Must set both respa inner and outer
> Cannot use just the inner or outer option with respa without using the other.

Must shrink-wrap piston boundary
> The boundary style of the face where the piston is applied must be of type s (shrink-wrapped).

Must specify a region in fix deposit
> The region keyword must be specified with this fix.

Must specify a region in fix pour
> The region keyword must be specified with this fix.

Must use -in switch with multiple partitions
> A multi-partition simulation cannot read the input script from stdin. The -in command-line option must be used to specify a file.

Must use a block or cylinder region with fix pour

Self-explanatory.

Must use a block region with fix pour for 2d simulations

Self-explanatory.

Must use a bond style with TIP4P potential

TIP4P potentials assume bond lengths in water are constrained by a fix shake command.

Must use a molecular atom style with fix poems molecule

Self-explanatory.

Must use a z-axis cylinder with fix pour

The axis of the cylinder region used with the fix pour command must be oriented along the z dimension.

Must use an angle style with TIP4P potential

TIP4P potentials assume angles in water are constrained by a fix shake command.

Must use atom style with molecule IDs with fix bond/swap

Self-explanatory.

Must use pair_style comb with fix qeq/comb

Self-explanatory.

Must use variable energy with fix addforce

Must define an energy vartiable when appllyting a dynamic force during minimization.

NEB command before simulation box is defined

Self-explanatory.

NEB requires damped dynamics minimizer

Use a different minimization style.

NEB requires use of fix neb

Self-explanatory.

NL ramp in wall/piston only implemented in zlo for now

The ramp keyword can only be used for piston applied to face zlo.

Needed bonus data not in data file

Some atom styles require bonus data. See the read_data doc page for details.

Needed topology not in data file

The header of the data file indicated that bonds or angles or dihedrals or impropers would be included, but they were not present.

Neigh_modify exclude molecule requires atom attribute molecule

Self-explanatory.

Neigh_modify include group != atom_modify first group

Self-explanatory.

Neighbor delay must be 0 or multiple of every setting

The delay and every parameters set via the neigh_modify command are inconsistent. If the delay setting is non-zero, then it must be a multiple of the every setting.

Neighbor include group not allowed with ghost neighbors

This is a current restriction within LAMMPS.

Neighbor list overflow, boost neigh_modify one

There are too many neighbors of a single atom. Use the neigh_modify command to increase the max number of neighbors allowed for one atom. You may also want to boost the page size.

Neighbor list overflow, boost neigh_modify one or page

There are too many neighbors of a single atom. Use the neigh_modify command to increase the neighbor page size and the max number of neighbors allowed for one atom.

Neighbor multi not yet enabled for ghost neighbors

This is a current restriction within LAMMPS.

Neighbor multi not yet enabled for granular

Self-explanatory.

Neighbor multi not yet enabled for rRESPA

Self-explanatory.

Neighbor page size must be >= 10x the one atom setting

This is required to prevent wasting too much memory.

New bond exceeded bonds per atom in fix bond/create
> See the read_data command for info on setting the "extra bond per atom" header value to allow for additional bonds to be formed.

New bond exceeded special list size in fix bond/create
> See the special_bonds extra command for info on how to leave space in the special bonds list to allow for additional bonds to be formed.

Newton bond change after simulation box is defined
> The newton command cannot be used to change the newton bond value after a read_data, read_restart, or create_box command.

No Kspace style defined for compute group/group
> Self-explanatory.

No OpenMP support compiled in
> An OpenMP flag is set, but LAMMPS was not built with OpenMP support.

No angle style is defined for compute angle/local
> Self-explanatory.

No angles allowed with this atom style
> Self-explanatory. Check data file.

No atoms in data file
> The header of the data file indicated that atoms would be included, but they were not present.

No basis atoms in lattice
> Basis atoms must be defined for lattice style user.

No bodies allowed with this atom style
> Self-explanatory. Check data file.

No bond style is defined for compute bond/local
> Self-explanatory.

No bonds allowed with this atom style
> Self-explanatory. Check data file.

No box information in dump. You have to use 'box no'
> Self-explanatory.

No dihedral style is defined for compute dihedral/local
> Self-explanatory.

No dihedrals allowed with this atom style
> Self-explanatory. Check data file.

No dump custom arguments specified
> The dump custom command requires that atom quantities be specified to output to dump file.

No dump local arguments specified
> Self-explanatory.

No ellipsoids allowed with this atom style
> Self-explanatory. Check data file.

No fix gravity defined for fix pour
> Cannot add poured particles without gravity to move them.

No improper style is defined for compute improper/local
> Self-explanatory.

No impropers allowed with this atom style
> Self-explanatory. Check data file.

No lines allowed with this atom style
> Self-explanatory. Check data file.

No matching element in ADP potential file
> The ADP potential file does not contain elements that match the requested elements.

No matching element in EAM potential file
> The EAM potential file does not contain elements that match the requested elements.

No overlap of box and region for create_atoms
> Self-explanatory.

No pair hbond/dreiding coefficients set

Self-explanatory.

No pair style defined for compute group/group

Cannot calculate group interactions without a pair style defined.

No pair style is defined for compute pair/local

Self-explanatory.

No pair style is defined for compute property/local

Self-explanatory.

No rigid bodies defined

The fix specification did not end up defining any rigid bodies.

No triangles allowed with this atom style

Self-explanatory. Check data file.

Non digit character between brackets in variable

Self-explantory.

Non integer # of swaps in temper command

Swap frequency in temper command must evenly divide the total # of timesteps.

Nprocs not a multiple of N for -reorder

Self-explanatory.

Numeric index is out of bounds

A command with an argument that specifies an integer or range of integers is using a value that is less than 1 or greater than the maximum allowed limit.

One or more atoms belong to multiple rigid bodies

Two or more rigid bodies defined by the fix rigid command cannot contain the same atom.

One or zero atoms in rigid body

Any rigid body defined by the fix rigid command must contain 2 or more atoms.

Only one cutoff allowed when requesting all long

Self-explanatory.

Only zhi currently implemented for fix append/atoms

Self-explanatory.

Out of range atoms - cannot compute MSM

One or more atoms are attempting to map their charge to a MSM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re-neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Out of range atoms - cannot compute PPPM

One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the re-neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Out of range atoms - cannot compute PPPMDisp

One or more atoms are attempting to map their charge to a PPPM grid point that is not owned by a processor. This is likely for one of two reasons, both of them bad. First, it may mean that an atom near the boundary of a processor's sub-domain has moved more than 1/2 the neighbor skin distance without neighbor lists being rebuilt and atoms being migrated to new processors. This also means you may be missing pairwise interactions that need to be computed. The solution is to change the

re-neighboring criteria via the neigh_modify command. The safest settings are "delay 0 every 1 check yes". Second, it may mean that an atom has moved far outside a processor's sub-domain or even the entire simulation box. This indicates bad physics, e.g. due to highly overlapping atoms, too large a timestep, etc.

Overlapping large/large in pair colloid
>   This potential is infinite when there is an overlap.

Overlapping small/large in pair colloid
>   This potential is infinite when there is an overlap.

POEMS fix must come before NPT/NPH fix
>   NPT/NPH fix must be defined in input script after all poems fixes, else the fix contribution to the pressure virial is incorrect.

PPPM grid is too large
>   The global PPPM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPM grid stencil extends beyond nearest neighbor processor
>   This is not allowed if the kspace_modify overlap setting is no.

PPPM order < minimum allowed order
>   The default minimum order is 2. This can be reset by the kspace_modify minorder command.

PPPM order cannot be < 2 or > than %d
>   This is a limitation of the PPPM implementation in LAMMPS.

PPPMDisp Coulomb grid is too large
>   The global PPPM grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPMDisp Dispersion grid is too large
>   The global dispersion grid is larger than OFFSET in one or more dimensions. OFFSET is currently set to 4096. You likely need to decrease the requested accuracy.

PPPMDisp coulomb order cannot be greater than %d
>   This is a limitation of the PPPM implementation in LAMMPS.

PRD command before simulation box is defined
>   The prd command cannot be used before a read_data, read_restart, or create_box command.

PRD nsteps must be multiple of t_event
>   Self-explanatory.

PRD t_corr must be multiple of t_event
>   Self-explanatory.

Package command after simulation box is defined
>   The package command cannot be used afer a read_data, read_restart, or create_box command.

Package cuda command without USER-CUDA installed
>   The USER-CUDA package must be installed via "make yes-user-cuda" before LAMMPS is built.

Pair body requires atom style body
>   Self-explanatory.

Pair body requires body style nparticle
>   This pair style is specific to the nparticle body style.

Pair brownian requires atom style sphere
>   Self-explanatory.

Pair brownian requires extended particles
>   One of the particles has radius 0.0.

Pair brownian requires monodisperse particles
>   All particles must be the same finite size.

Pair brownian/poly requires atom style sphere
>   Self-explanatory.

Pair brownian/poly requires extended particles
>   One of the particles has radius 0.0.

Pair brownian/poly requires newton pair off
>   Self-explanatory.

Pair coeff for hybrid has invalid style

Style in pair coeff must have been listed in pair_style command.

Pair colloid/poly requires atom style sphere

Self-explanatory.

Pair coul/wolf requires atom attribute q

The atom style defined does not have this attribute.

Pair cutoff < Respa interior cutoff

One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair dipole/cut requires atom attributes q, mu, torque

The atom style defined does not have these attributes.

Pair dipole/cut/gpu requires atom attributes q, mu, torque

The atom style defined does not have this attribute.

Pair distance < table inner cutoff

Two atoms are closer together than the pairwise table allows.

Pair distance > table outer cutoff

Two atoms are further apart than the pairwise table allows.

Pair dpd requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair gayberne epsilon a,b,c coeffs are not all set

Each atom type involved in pair_style gayberne must have these 3 coefficients set at least once.

Pair gayberne requires atom style ellipsoid

Self-explanatory.

Pair gayberne requires atoms with same type have same shape

Self-explanatory.

Pair gayberne/gpu requires atom style ellipsoid

Self-explanatory.

Pair gayberne/gpu requires atoms with same type have same shape

Self-explanatory.

Pair granular requires atom style sphere

Self-explanatory.

Pair granular requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair granular with shear history requires newton pair off

This is a current restriction of the implementation of pair granular styles with history.

Pair hybrid sub-style does not support single call

You are attempting to invoke a single() call on a pair style that doesn't support it.

Pair hybrid sub-style is not used

No pair_coeff command used a sub-style specified in the pair_style command.

Pair inner cutoff < Respa interior cutoff

One or more pairwise cutoffs are too short to use with the specified rRESPA cutoffs.

Pair inner cutoff >= Pair outer cutoff

The specified cutoffs for the pair style are inconsistent.

Pair line/lj requires atom style line

Self-explanatory.

Pair lubricate requires atom style sphere

Self-explanatory.

Pair lubricate requires ghost atoms store velocity

Use the communicate vel yes command to enable this.

Pair lubricate requires monodisperse particles

All particles must be the same finite size.

Pair lubricate/poly requires atom style sphere

Self-explanatory.

Pair lubricate/poly requires extended particles

One of the particles has radius 0.0.

Errors: 1095

Pair lubricate/poly requires ghost atoms store velocity
　　　Use the communicate vel yes command to enable this.
Pair lubricate/poly requires newton pair off
　　　Self-explanatory.
Pair lubricateU requires atom style sphere
　　　Self-explanatory.
Pair lubricateU requires ghost atoms store velocity
　　　Use the communicate vel yes command to enable this.
Pair lubricateU requires monodisperse particles
　　　All particles must be the same finite size.
Pair lubricateU/poly requires ghost atoms store velocity
　　　Use the communicate vel yes command to enable this.
Pair lubricateU/poly requires newton pair off
　　　Self-explanatory.
Pair peri lattice is not identical in x, y, and z
　　　The lattice defined by the lattice command must be cubic.
Pair peri requires a lattice be defined
　　　Use the lattice command for this purpose.
Pair peri requires an atom map, see atom_modify
　　　Even for atomic systems, an atom map is required to find Peridynamic bonds. Use the atom_modify
　　　command to define one.
Pair resquared epsilon a,b,c coeffs are not all set
　　　Self-explanatory.
Pair resquared epsilon and sigma coeffs are not all set
　　　Self-explanatory.
Pair resquared requires atom style ellipsoid
　　　Self-explanatory.
Pair resquared requires atoms with same type have same shape
　　　Self-explanatory.
Pair resquared/gpu requires atom style ellipsoid
　　　Self-explanatory.
Pair resquared/gpu requires atoms with same type have same shape
　　　Self-explanatory.
Pair style AIREBO requires atom IDs
　　　This is a requirement to use the AIREBO potential.
Pair style AIREBO requires newton pair on
　　　See the newton command. This is a restriction to use the AIREBO potential.
Pair style BOP requires atom IDs
　　　This is a requirement to use the BOP potential.
Pair style BOP requires newton pair on
　　　See the newton command. This is a restriction to use the BOP potential.
Pair style COMB requires atom IDs
　　　This is a requirement to use the AIREBO potential.
Pair style COMB requires atom attribute q
　　　Self-explanatory.
Pair style COMB requires newton pair on
　　　See the newton command. This is a restriction to use the COMB potential.
Pair style LCBOP requires atom IDs
　　　This is a requirement to use the LCBOP potential.
Pair style LCBOP requires newton pair on
　　　See the newton command. This is a restriction to use the LCBOP potential.
Pair style MEAM requires newton pair on
　　　See the newton command. This is a restriction to use the MEAM potential.
Pair style Stillinger-Weber requires atom IDs

This is a requirement to use the SW potential.

Pair style Stillinger-Weber requires newton pair on

See the newton command. This is a restriction to use the SW potential.

Pair style Tersoff requires atom IDs

This is a requirement to use the Tersoff potential.

Pair style Tersoff requires newton pair on

See the newton command. This is a restriction to use the Tersoff potential.

Pair style bop requires comm ghost cutoff at least 3x larger than %g

Use the communicate ghost command to set this. See the pair bop doc page for more details.

Pair style born/coul/long requires atom attribute q

An atom style that defines this attribute must be used.

Pair style born/coul/long/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style born/coul/wolf requires atom attribute q

The atom style defined does not have this attribute.

Pair style buck/coul/cut requires atom attribute q

The atom style defined does not have this attribute.

Pair style buck/coul/long requires atom attribute q

The atom style defined does not have these attributes.

Pair style buck/coul/long/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style buck/long/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style coul/cut requires atom attribute q

The atom style defined does not have these attributes.

Pair style coul/dsf requires atom attribute q

The atom style defined does not have this attribute.

Pair style coul/dsf/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style coul/long/gpu requires atom attribute q

The atom style defined does not have these attributes.

Pair style does not have extra field requested by compute pair/local

The pair style does not support the pN value requested by the compute pair/local command.

Pair style does not support bond_style quartic

The pair style does not have a single() function, so it can not be invoked by bond_style quartic.

Pair style does not support compute group/group

The pair_style does not have a single() function, so it cannot be invokded by the compute group/group command.

Pair style does not support compute pair/local

The pair style does not have a single() function, so it can not be invoked by compute pair/local.

Pair style does not support compute property/local

The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support fix bond/swap

The pair style does not have a single() function, so it can not be invoked by fix bond/swap.

Pair style does not support pair_write

The pair style does not have a single() function, so it can not be invoked by pair write.

Pair style does not support rRESPA inner/middle/outer

You are attempting to use rRESPA options with a pair style that does not support them.

Pair style granular with history requires atoms have IDs

Atoms in the simulation do not have IDs, so history effects cannot be tracked by the granular pair potential.

Pair style hbond/dreiding requires an atom map, see atom_modify

Self-explanatory.

Pair style hbond/dreiding requires atom IDs

Self-explanatory.

Pair style hbond/dreiding requires molecular system

Self-explanatory.

Pair style hbond/dreiding requires newton pair on

See the newton command for details.

Pair style hybrid cannot have hybrid as an argument

Self-explanatory.

Pair style hybrid cannot have none as an argument

Self-explanatory.

Pair style is incompatible with KSpace style

If a pair style with a long-range Coulombic component is selected, then a kspace style must also be used.

Pair style lj/charmm/coul/charmm requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/charmm/coul/long requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/charmm/coul/long/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/class2/coul/cut requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/class2/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/class2/coul/long/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/cut requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/cut/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/debye/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/dsf requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/cut/coul/dsf/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/coul/long/gpu requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/cut/tip4p/long requires atom IDs

There are no atom IDs defined in the system and the TIP4P potential requires them to find O,H atoms with a water molecule.

Pair style lj/cut/tip4p/long requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/cut/tip4p/long requires newton pair on

This is because the computation of constraint forces within a water molecule adds forces to atoms owned by other processors.

Pair style lj/cut/coul/msm requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/gromacs/coul/gromacs requires atom attribute q

An atom_style with this attribute is needed.

Pair style lj/long/coul/long requires atom attribute q

The atom style defined does not have this attribute.

Pair style lj/long/tip4p/long requires atom IDs

There are no atom IDs defined in the system and the TIP4P potential requires them to find O,H atoms

with a water molecule.

Pair style lj/long/tip4p/long requires atom attribute q

The atom style defined does not have these attributes.

Pair style lj/long/tip4p/long requires newton pair on

This is because the computation of constraint forces within a water molecule adds forces to atoms owned by other processors.

Pair style peri requires atom style peri

Self-explanatory.

Pair style reax requires atom IDs

This is a requirement to use the ReaxFF potential.

Pair style reax requires newton pair on

This is a requirement to use the ReaxFF potential.

Pair style requires a KSpace style

No kspace style is defined.

Pair table cutoffs must all be equal to use with KSpace

When using pair style table with a long-range KSpace solver, the cutoffs for all atom type pairs must all be the same, since the long-range solver starts at that cutoff.

Pair table parameters did not set N

List of pair table parameters must include N setting.

Pair tersoff/zbl requires metal or real units

This is a current restriction of this pair potential.

Pair tri/lj requires atom style tri

Self-explanatory.

Pair yukawa/colloid requires atom style sphere

Self-explantory.

Pair yukawa/colloid requires atoms with same type have same radius

Self-explantory.

Pair yukawa/colloid/gpu requires atom style sphere

Self-explanatory.

PairKIM only works with 3D problems.

This is a current restriction of this pair style.

Pair_coeff command before pair_style is defined

Self-explanatory.

Pair_coeff command before simulation box is defined

The pair_coeff command cannot be used before a read_data, read_restart, or create_box command.

Pair_modify command before pair_style is defined

Self-explanatory.

Pair_write command before pair_style is defined

Self-explanatory.

Particle on or inside fix wall surface

Particles must be "exterior" to the wall in order for energy/force to be calculated.

Particle on or inside surface of region used in fix wall/region

Particles must be "exterior" to the region surface in order for energy/force to be calculated.

Per-atom compute in equal-style variable formula

Equal-style variables cannot use per-atom quantities.

Per-atom energy was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied energy, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Per-atom fix in equal-style variable formula

Equal-style variables cannot use per-atom quantities.

Per-atom virial was not tallied on needed timestep

You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Per-processor system is too big

The number of owned atoms plus ghost atoms on a single processor must fit in 32-bit integer.

Potential energy ID for fix neb does not exist
> Self-explanatory.

Potential energy ID for fix nvt/nph/npt does not exist
> A compute for potential energy must be defined.

Potential file has duplicate entry
> The potential file for a SW or Tersoff potential has more than one entry for the same 3 ordered elements.

Potential file is missing an entry
> The potential file for a SW or Tersoff potential does not have a needed entry.

Power by 0 in variable formula
> Self-explanatory.

Pressure ID for fix box/relax does not exist
> The compute ID needed to compute pressure for the fix does not exist.

Pressure ID for fix modify does not exist
> Self-explanatory.

Pressure ID for fix npt/nph does not exist
> Self-explanatory.

Pressure ID for fix press/berendsen does not exist
> The compute ID needed to compute pressure for the fix does not exist.

Pressure ID for fix rigid npt/nph does not exist
> Self-explanatory.

Pressure ID for thermo does not exist
> The compute ID needed to compute pressure for thermodynamics does not exist.

Pressure control can not be used with fix nvt
> Self-explanatory.

Pressure control can not be used with fix nvt/asphere
> Self-explanatory.

Pressure control can not be used with fix nvt/sllod
> Self-explanatory.

Pressure control can not be used with fix nvt/sphere
> Self-explanatory.

Pressure control must be used with fix nph
> Self-explanatory.

Pressure control must be used with fix nph/asphere
> Self-explanatory.

Pressure control must be used with fix nph/sphere
> Self-explanatory.

Pressure control must be used with fix nphug
> A pressure control keyword (iso, aniso, tri, x, y, or z) must be provided.

Pressure control must be used with fix npt
> Self-explanatory.

Pressure control must be used with fix npt/asphere
> Self-explanatory.

Pressure control must be used with fix npt/sphere
> Self-explanatory.

Processor count in z must be 1 for 2d simulation
> Self-explanatory.

Processor partitions are inconsistent
> The total number of processors in all partitions must match the number of processors LAMMPS is running on.

Processors command after simulation box is defined
> The processors command cannot be used after a read_data, read_restart, or create_box command.

Processors custom grid file is inconsistent

The vales in the custom file are not consistent with the number of processors you are running on or the Px,Py,Pz settings of the processors command. Or there was not a setting for every processor.

Processors grid numa and map style are incompatible

Using numa for gstyle in the processors command requires using cart for the map option.

Processors part option and grid style are incompatible

Cannot use gstyle numa or custom with the part option.

Processors twogrid requires proc count be a multiple of core count

Self-explanatory.

Pstart and Pstop must have the same value

Self-explanatory.

R0 < 0 for fix spring command

Equilibrium spring length is invalid.

Read_dump command before simulation box is defined

The read_dump command cannot be used before a read_data, read_restart, or create_box command.

Read_dump field not found in dump file

Self-explanatory.

Read_dump triclinic status does not match simulation

Both the dump snapshot and the current LAMMPS simulation must be using either an orthogonal or triclinic box.

Read_dump x,y,z fields do not have consistent scaling

Self-explanatory.

Reax_defs.h setting for NATDEF is too small

Edit the setting in the ReaxFF library and re-compile the library and re-build LAMMPS.

Reax_defs.h setting for NNEIGHMAXDEF is too small

Edit the setting in the ReaxFF library and re-compile the library and re-build LAMMPS.

Receiving partition in processors part command is already a receiver

Cannot specify a partition to be a receiver twice.

Region ID for compute reduce/region does not exist

Self-explanatory.

Region ID for compute temp/region does not exist

Self-explanatory.

Region ID for dump custom does not exist

Self-explanatory.

Region ID for fix addforce does not exist

Self-explanatory.

Region ID for fix ave/spatial does not exist

Self-explanatory.

Region ID for fix aveforce does not exist

Self-explanatory.

Region ID for fix deposit does not exist

Self-explanatory.

Region ID for fix evaporate does not exist

Self-explanatory.

Region ID for fix gcmc does not exist

Self-explanatory.

Region ID for fix heat does not exist

Self-explanatory.

Region ID for fix setforce does not exist

Self-explanatory.

Region ID for fix wall/region does not exist

Self-explanatory.

Region ID in variable formula does not exist

Self-explanatory.

Region cannot have 0 length rotation vector

Self-explanatory.

Region intersect region ID does not exist

Self-explanatory.

Region union or intersect cannot be dynamic

The sub-regions can be dynamic, but not the combined region.

Region union region ID does not exist

One or more of the region IDs specified by the region union command does not exist.

Replacing a fix, but new style != old style

A fix ID can be used a 2nd time, but only if the style matches the previous fix. In this case it is assumed you with to reset a fix's parameters. This error may mean you are mistakenly re-using a fix ID when you do not intend to.

Replicate command before simulation box is defined

The replicate command cannot be used before a read_data, read_restart, or create_box command.

Replicate did not assign all atoms correctly

Atoms replicated by the replicate command were not assigned correctly to processors. This is likely due to some atom coordinates being outside a non-periodic simulation box.

Replicated molecular system atom IDs are too big

See the setting for the allowed atom ID size in the src/lmptype.h file.

Replicated system is too big

See the setting for bigint in the src/lmptype.h file.

Rerun command before simulation box is defined

The rerun command cannot be used before a read_data, read_restart, or create_box command.

Rerun dump file does not contain requested snapshot

Self-explanatory.

Resetting timestep is not allowed with fix move

This is because fix move is moving atoms based on elapsed time.

Respa inner cutoffs are invalid

The first cutoff must be <= the second cutoff.

Respa levels must be >= 1

Self-explanatory.

Respa middle cutoffs are invalid

The first cutoff must be <= the second cutoff.

Restart variable returned a bad timestep

The variable must return a timestep greater than the current timestep.

Restrain atoms %d %d %d %d missing on proc %d at step %ld

The 4 atoms in a restrain dihedral specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Restrain atoms %d %d %d missing on proc %d at step %ld

The 3 atoms in a restrain angle specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Restrain atoms %d %d missing on proc %d at step %ld

The 2 atoms in a restrain bond specified by the fix restrain command are not all accessible to a processor. This probably means an atom has moved too far.

Reuse of compute ID

A compute ID cannot be used twice.

Reuse of dump ID

A dump ID cannot be used twice.

Reuse of region ID

A region ID cannot be used twice.

Rigid body atoms %d %d missing on proc %d at step %ld

This means that an atom cannot find the atom that owns the rigid body it is part of, or vice versa. The solution is to use the communicate cutoff command to insure ghost atoms are acquired from far enough away to encompass the max distance printed when the fix rigid/small command was invoked.

Rigid body has degenerate moment of inertia

Fix poems will only work with bodies (collections of atoms) that have non-zero principal moments of inertia. This means they must be 3 or more non-collinear atoms, even with joint atoms removed.

Rigid fix must come before NPT/NPH fix

NPT/NPH fix must be defined in input script after all rigid fixes, else the rigid fix contribution to the pressure virial is incorrect.

Rmask function in equal-style variable formula

Rmask is per-atom operation.

Run command before simulation box is defined

The run command cannot be used before a read_data, read_restart, or create_box command.

Run command start value is after start of run

Self-explanatory.

Run command stop value is before end of run

Self-explanatory.

Run_style command before simulation box is defined

The run_style command cannot be used before a read_data, read_restart, or create_box command.

SRD bin size for fix srd differs from user request

Fix SRD had to adjust the bin size to fit the simulation box. See the cubic keyword if you want this message to be an error vs warning.

SRD bins for fix srd are not cubic enough

The bin shape is not within tolerance of cubic. See the cubic keyword if you want this message to be an error vs warning.

SRD particle %d started inside big particle %d on step %ld bounce %d

See the inside keyword if you want this message to be an error vs warning.

Same dimension twice in fix ave/spatial

Self-explanatory.

Sending partition in processors part command is already a sender

Cannot specify a partition to be a sender twice.

Set command before simulation box is defined

The set command cannot be used before a read_data, read_restart, or create_box command.

Set command with no atoms existing

No atoms are yet defined so the set command cannot be used.

Set region ID does not exist

Region ID specified in set command does not exist.

Shake angles have different bond types

All 3-atom angle-constrained SHAKE clusters specified by the fix shake command that are the same angle type, must also have the same bond types for the 2 bonds in the angle.

Shake atoms %d %d %d %d missing on proc %d at step %ld

The 4 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d %d missing on proc %d at step %ld

The 3 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake atoms %d %d missing on proc %d at step %ld

The 2 atoms in a single shake cluster specified by the fix shake command are not all accessible to a processor. This probably means an atom has moved too far.

Shake cluster of more than 4 atoms

A single cluster specified by the fix shake command can have no more than 4 atoms.

Shake clusters are connected

A single cluster specified by the fix shake command must have a single central atom with up to 3 other atoms bonded to it.

Shake determinant = 0.0

The determinant of the matrix being solved for a single cluster specified by the fix shake command is numerically invalid.

Shake fix must come before NPT/NPH fix

NPT fix must be defined in input script after SHAKE fix, else the SHAKE fix contribution to the pressure virial is incorrect.

Small, tag, big integers are not sized correctly

See description of these 3 data types in src/lmptype.h.

Smallint setting in lmptype.h is invalid

It has to be the size of an integer.

Smallint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LAMMPS version you are running.

Specified processors != physical processors

The 3d grid of processors defined by the processors command does not match the number of processors LAMMPS is being run on.

Specified target stress must be uniaxial or hydrostatic

Self-explanatory.

Sqrt of negative value in variable formula

Self-explanatory.

Substitution for illegal variable

Input script line contained a variable that could not be substituted for.

System in data file is too big

See the setting for bigint in the src/lmptype.h file.

System is not charge neutral, net charge = %g

The total charge on all atoms on the system is not 0.0, which is not valid for the long-range Coulombic solvers.

TAD nsteps must be multiple of t_event

Self-explanatory.

TIP4P hydrogen has incorrect atom type

The TIP4P pairwise computation found an H atom whose type does not agree with the specified H type.

TIP4P hydrogen is missing

The TIP4P pairwise computation failed to find the correct H atom within a water molecule.

TMD target file did not list all group atoms

The target file for the fix tmd command did not list all atoms in the fix group.

Tad command before simulation box is defined

Self-explanatory.

Tagint setting in lmptype.h is invalid

Tagint must be as large or larger than smallint.

Tagint setting in lmptype.h is not compatible

Smallint stored in restart file is not consistent with LAMMPS version you are running.

Target temperature for fix nvt/npt/nph cannot be 0.0

Self-explanatory.

Target temperature for fix rigid/npt cannot be 0.0

Self-explanatory.

Target temperature for fix rigid/nvt cannot be 0.0

Self-explanatory.

Temper command before simulation box is defined

The temper command cannot be used before a read_data, read_restart, or create_box command.

Temperature ID for fix bond/swap does not exist

Self-explanatory.

Temperature ID for fix box/relax does not exist

Self-explanatory.

Temperature ID for fix nvt/npt does not exist

Self-explanatory.

Temperature ID for fix press/berendsen does not exist

Self-explanatory.

Temperature ID for fix rigid nvt/npt/nph does not exist

Self-explanatory.

Temperature ID for fix temp/berendsen does not exist

Self-explanatory.

Temperature ID for fix temp/rescale does not exist

Self-explanatory.

Temperature control can not be used with fix nph

Self-explanatory.

Temperature control can not be used with fix nph/asphere

Self-explanatory.

Temperature control can not be used with fix nph/sphere

Self-explanatory.

Temperature control must be used with fix nphug

The temp keyword must be provided.

Temperature control must be used with fix npt

Self-explanatory.

Temperature control must be used with fix npt/asphere

Self-explanatory.

Temperature control must be used with fix npt/sphere

Self-explanatory.

Temperature control must be used with fix nvt

Self-explanatory.

Temperature control must be used with fix nvt/asphere

Self-explanatory.

Temperature control must be used with fix nvt/sllod

Self-explanatory.

Temperature control must be used with fix nvt/sphere

Self-explanatory.

Temperature for fix nvt/sllod does not have a bias

The specified compute must compute temperature with a bias.

Tempering could not find thermo_pe compute

This compute is created by the thermo command. It must have been explicitly deleted by a uncompute command.

Tempering fix ID is not defined

The fix ID specified by the temper command does not exist.

Tempering temperature fix is not valid

The fix specified by the temper command is not one that controls temperature (nvt or langevin).

Test_descriptor_string already allocated

This should not happen. It likely indicates a bug in the pair_kim implementation.

The package gpu command is required for gpu styles

Self-explanatory.

Thermo and fix not computed at compatible times

Fixes generate values on specific timesteps. The thermo output does not match these timesteps.

Thermo compute array is accessed out-of-range

Self-explanatory.

Thermo compute does not compute array

Self-explanatory.

Thermo compute does not compute scalar

Self-explanatory.

Thermo compute does not compute vector

Self-explanatory.

Thermo compute vector is accessed out-of-range

Self-explanatory.

Thermo custom variable cannot be indexed

Self-explanatory.

Thermo custom variable is not equal-style variable
> Only equal-style variables can be output with thermodynamics, not atom-style variables.

Thermo every variable returned a bad timestep
> The variable must return a timestep greater than the current timestep.

Thermo fix array is accessed out-of-range
> Self-explanatory.

Thermo fix does not compute array
> Self-explanatory.

Thermo fix does not compute scalar
> Self-explanatory.

Thermo fix does not compute vector
> Self-explanatory.

Thermo fix vector is accessed out-of-range
> Self-explanatory.

Thermo keyword in variable requires lattice be defined
> The xlat, ylat, zlat keywords refer to lattice properties.

Thermo keyword in variable requires thermo to use/init pe
> You are using a thermo keyword in a variable that requires potential energy to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword in variable requires thermo to use/init press
> You are using a thermo keyword in a variable that requires pressure to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword in variable requires thermo to use/init temp
> You are using a thermo keyword in a variable that requires temperature to be calculated, but your thermo output does not use it. Add it to your thermo output.

Thermo keyword requires lattice be defined
> The xlat, ylat, zlat keywords refer to lattice properties.

Thermo style does not use press
> Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.

Thermo style does not use temp
> Cannot use thermo_modify to set this parameter since the thermo_style is not computing this quantity.

Thermo_modify int format does not contain d character
> Self-explanatory.

Thermo_modify pressure ID does not compute pressure
> The specified compute ID does not compute pressure.

Thermo_modify temperature ID does not compute temperature
> The specified compute ID does not compute temperature.

Thermo_style command before simulation box is defined
> The thermo_style command cannot be used before a read_data, read_restart, or create_box command.

This variable thermo keyword cannot be used between runs
> Keywords that refer to time (such as cpu, elapsed) do not make sense in between runs.

Threshhold for an atom property that isn't allocated
> A dump threshhold has been requested on a quantity that is not defined by the atom style used in this simulation.

Timestep must be >= 0
> Specified timestep is invalid.

Too big a problem to use velocity create loop all
> The system size must fit in a 32-bit integer to use this option.

Too big a timestep
> Specified timestep is too large.

Too big a timestep for dump dcd
> The timestep must fit in a 32-bit integer to use this dump style.

Too big a timestep for dump xtc
> The timestep must fit in a 32-bit integer to use this dump style.

Too few bits for lookup table
> Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many MSM grid levels
> The max number of MSM grid levels is hardwired to 10.

Too many atom pairs for pair bop
> The number of atomic pairs exceeds the expected number. Check your atomic structure to ensure that it is realistic.

Too many atom sorting bins
> This is likely due to an immense simulation box that has blown up to a large size.

Too many atom triplets for pair bop
> The number of three atom groups for angle determinations exceeds the expected number. Check your atomic structrure to ensure that it is realistic.

Too many atoms for dump dcd
> The system size must fit in a 32-bit integer to use this dump style.

Too many atoms for dump xtc
> The system size must fit in a 32-bit integer to use this dump style.

Too many atoms to dump sort
> Cannot sort when running with more than 2^31 atoms.

Too many exponent bits for lookup table
> Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many groups
> The maximum number of atom groups (including the "all" group) is given by MAX_GROUP in group.cpp and is 32.

Too many iterations
> You must use a number of iterations that fit in a 32-bit integer for minimization.

Too many lines in one body in data file - boost MAXBODY
> MAXBODY is a setting at the top of the src/read_data.cpp file. Set it larger and re-compile the code.

Too many local+ghost atoms for neighbor list
> The number of nlocal + nghost atoms on a processor is limited by the size of a 32-bit integer with 2 bits removed for masking 1-2, 1-3, 1-4 neighbors.

Too many mantissa bits for lookup table
> Table size specified via pair_modify command does not work with your machine's floating point representation.

Too many masses for fix shake
> The fix shake command cannot list more masses than there are atom types.

Too many neighbor bins
> This is likely due to an immense simulation box that has blown up to a large size.

Too many timesteps
> The cummulative timesteps must fit in a 64-bit integer.

Too many timesteps for NEB
> You must use a number of timesteps that fit in a 32-bit integer for NEB.

Too many total atoms
> See the setting for bigint in the src/lmptype.h file.

Too many total bits for bitmapped lookup table
> Table size specified via pair_modify command is too large. Note that a value of N generates a 2^N size table.

Too many touching neighbors - boost MAXTOUCH
> A granular simulation has too many neighbors touching one atom. The MAXTOUCH parameter in fix_shear_history.cpp must be set larger and LAMMPS must be re-built.

Too much per-proc info for dump
> Number of local atoms times number of columns must fit in a 32-bit integer for dump.

Tree structure in joint connections

Fix poems cannot (yet) work with coupled bodies whose joints connect the bodies in a tree structure.

Triclinic box skew is too large

The displacement in a skewed direction must be less than half the box length in that dimension. E.g. the xy tilt must be between -half and +half of the x box length. This constraint can be relaxed by using the box tilt command.

Tried to convert a double to int, but input_double > INT_MAX

Self-explanatory.

Two groups cannot be the same in fix spring couple

Self-explanatory.

USER-CUDA mode requires CUDA variant of min style

CUDA mode is enabled, so the min style must include a cuda suffix.

USER-CUDA mode requires CUDA variant of run style

CUDA mode is enabled, so the run style must include a cuda suffix.

USER-CUDA package requires a cuda enabled atom_style

Self-explanatory.

Unable to initialize accelerator for use

There was a problem initializing an accelerator for the gpu package

Unbalanced quotes in input line

No matching end double quote was found following a leading double quote.

Unexpected end of -reorder file

Self-explanatory.

Unexpected end of custom file

Self-explanatory.

Unexpected end of data file

LAMMPS hit the end of the data file while attempting to read a section. Something is wrong with the format of the data file.

Unexpected end of dump file

A read operation from the file failed.

Unexpected end of fix rigid file

A read operation from the file failed.

Units command after simulation box is defined

The units command cannot be used after a read_data, read_restart, or create_box command.

Universe/uloop variable count < # of partitions

A universe or uloop style variable must specify a number of values >= to the number of processor partitions.

Unknown command: %s

The command is not known to LAMMPS. Check the input script.

Unknown error in GPU library

Self-explanatory.

Unknown identifier in data file: %s

A section of the data file cannot be read by LAMMPS.

Unknown table style in angle style table

Self-explanatory.

Unknown table style in bond style table

Self-explanatory.

Unknown table style in pair_style command

Style of table is invalid for use with pair_style table command.

Unknown unit_style

Self-explanatory. Check the input script or data file.

Unrecognized lattice type in MEAM file 1

The lattice type in an entry of the MEAM library file is not valid.

Unrecognized lattice type in MEAM file 2

The lattice type in an entry of the MEAM parameter file is not valid.

Unrecognized pair style in compute pair command

Self-explanatory.

Unrecognized virial argument in pair_style command

Only two options are supported: LAMMPSvirial and KIMvirial

Unsupported mixing rule in kspace_style ewald/disp

Only geometric mixing is supported.

Unsupported mixing rule in kspace_style pppm/disp for pair_style %s

Only geometric mixing is supported.

Unsupported order in kspace_style ewald/disp

Only 1/r^6 dispersion terms are supported.

Unsupported order in kspace_style pppm/disp pair_style %s

Only 1/r^6 dispersion terms are supported.

Use of change_box with undefined lattice

Must use lattice command with displace_box command if units option is set to lattice.

Use of compute temp/ramp with undefined lattice

Must use lattice command with compute temp/ramp command if units option is set to lattice.

Use of displace_atoms with undefined lattice

Must use lattice command with displace_atoms command if units option is set to lattice.

Use of fix append/atoms with undefined lattice

A lattice must be defined before using this fix.

Use of fix ave/spatial with undefined lattice

A lattice must be defined to use fix ave/spatial with units = lattice.

Use of fix deform with undefined lattice

A lattice must be defined to use fix deform with units = lattice.

Use of fix deposit with undefined lattice

Must use lattice command with compute fix deposit command if units option is set to lattice.

Use of fix dt/reset with undefined lattice

Must use lattice command with fix dt/reset command if units option is set to lattice.

Use of fix indent with undefined lattice

The lattice command must be used to define a lattice before using the fix indent command.

Use of fix move with undefined lattice

Must use lattice command with fix move command if units option is set to lattice.

Use of fix recenter with undefined lattice

Must use lattice command with fix recenter command if units option is set to lattice.

Use of fix wall with undefined lattice

Must use lattice command with fix wall command if units option is set to lattice.

Use of fix wall/piston with undefined lattice

A lattice must be defined before using this fix.

Use of region with undefined lattice

If units = lattice (the default) for the region command, then a lattice must first be defined via the lattice command.

Use of velocity with undefined lattice

If units = lattice (the default) for the velocity set or velocity ramp command, then a lattice must first be defined via the lattice command.

Using fix nvt/sllod with inconsistent fix deform remap option

Fix nvt/sllod requires that deforming atoms have a velocity profile provided by "remap v" as a fix deform option.

Using fix nvt/sllod with no fix deform defined

Self-explanatory.

Using fix srd with inconsistent fix deform remap option

When shearing the box in an SRD simulation, the remap v option for fix deform needs to be used.

Using pair lubricate with inconsistent fix deform remap option

Must use remap v option with fix deform with this pair style.

Using pair lubricate/poly with inconsistent fix deform remap option

If fix deform is used, the remap v option is required.

Variable ID in variable formula does not exist
> Self-explanatory.

Variable evaluation before simulation box is defined
> Cannot evaluate a compute or fix or atom-based value in a variable before the simulation has been setup.

Variable evaluation in fix wall gave bad value
> The returned value for epsilon or sigma < 0.0.

Variable evaluation in region gave bad value
> Variable returned a radius < 0.0.

Variable for compute ti is invalid style
> Self-explanatory.

Variable for dump every is invalid style
> Only equal-style variables can be used.

Variable for dump image center is invalid style
> Must be an equal-style variable.

Variable for dump image persp is invalid style
> Must be an equal-style variable.

Variable for dump image phi is invalid style
> Must be an equal-style variable.

Variable for dump image theta is invalid style
> Must be an equal-style variable.

Variable for dump image zoom is invalid style
> Must be an equal-style variable.

Variable for fix adapt is invalid style
> Only equal-style variables can be used.

Variable for fix addforce is invalid style
> Self-explanatory.

Variable for fix aveforce is invalid style
> Only equal-style variables can be used.

Variable for fix deform is invalid style
> The variable must be an equal-style variable.

Variable for fix efield is invalid style
> Only equal-style variables can be used.

Variable for fix gravity is invalid style
> Only equal-style variables can be used.

Variable for fix heat is invalid style
> Only equal-style or atom-style variables can be used.

Variable for fix indent is invalid style
> Only equal-style variables can be used.

Variable for fix indent is not equal style
> Only equal-style variables can be used.

Variable for fix langevin is invalid style
> It must be an equal-style variable.

Variable for fix move is invalid style
> Only equal-style variables can be used.

Variable for fix setforce is invalid style
> Only equal-style variables can be used.

Variable for fix temp/berendsen is invalid style
> Only equal-style variables can be used.

Variable for fix temp/rescale is invalid style
> Only equal-style variables can be used.

Variable for fix wall is invalid style
> Only equal-style variables can be used.

Variable for fix wall/reflect is invalid style

Only equal-style variables can be used.

Variable for fix wall/srd is invalid style

Only equal-style variables can be used.

Variable for group is invalid style

Only atom-style variables can be used.

Variable for region cylinder is invalid style

Only equal-style varaibles are allowed.

Variable for region is invalid style

Only equal-style variables can be used.

Variable for region is not equal style

Self-explanatory.

Variable for region sphere is invalid style

Only equal-style varaibles are allowed.

Variable for restart is invalid style

Only equal-style variables can be used.

Variable for thermo every is invalid style

Only equal-style variables can be used.

Variable for velocity set is invalid style

Only atom-style variables can be used.

Variable formula compute array is accessed out-of-range

Self-explanatory.

Variable formula compute vector is accessed out-of-range

Self-explanatory.

Variable formula fix array is accessed out-of-range

Self-explanatory.

Variable formula fix vector is accessed out-of-range

Self-explanatory.

Variable has circular dependency

A circular dependency is when variable "a" in used by variable "b" and variable "b" is also used by varaible "a". Circular dependencies with longer chains of dependence are also not allowed.

Variable name for compute atom/molecule does not exist

Self-explanatory.

Variable name for compute reduce does not exist

Self-explanatory.

Variable name for compute ti does not exist

Self-explanatory.

Variable name for dump every does not exist

Self-explanatory.

Variable name for dump image center does not exist

Self-explanatory.

Variable name for dump image persp does not exist

Self-explanatory.

Variable name for dump image phi does not exist

Self-explanatory.

Variable name for dump image theta does not exist

Self-explanatory.

Variable name for dump image zoom does not exist

Self-explanatory.

Variable name for fix adapt does not exist

Self-explanatory.

Variable name for fix addforce does not exist

Self-explanatory.

Variable name for fix ave/atom does not exist

Self-explanatory.

Variable name for fix ave/correlate does not exist
    Self-explanatory.
Variable name for fix ave/histo does not exist
    Self-explanatory.
Variable name for fix ave/spatial does not exist
    Self-explanatory.
Variable name for fix ave/time does not exist
    Self-explanatory.
Variable name for fix aveforce does not exist
    Self-explanatory.
Variable name for fix deform does not exist
    Self-explantory.
Variable name for fix efield does not exist
    Self-explanatory.
Variable name for fix gravity does not exist
    Self-explanatory.
Variable name for fix heat does not exist
    Self-explanatory.
Variable name for fix indent does not exist
    Self-explanatory.
Variable name for fix langevin does not exist
    Self-explanatory.
Variable name for fix move does not exist
    Self-explanatory.
Variable name for fix setforce does not exist
    Self-explanatory.
Variable name for fix store/state does not exist
    Self-explanatory.
Variable name for fix temp/berendsen does not exist
    Self-explanatory.
Variable name for fix temp/rescale does not exist
    Self-explanatory.
Variable name for fix wall does not exist
    Self-explanatory.
Variable name for fix wall/reflect does not exist
    Self-explanatory.
Variable name for fix wall/srd does not exist
    Self-explanatory.
Variable name for group does not exist
    Self-explanatory.
Variable name for region cylinder does not exist
    Self-explanatory.
Variable name for region does not exist
    Self-explanatory.
Variable name for region sphere does not exist
    Self-explanatory.
Variable name for restart does not exist
    Self-explanatory.
Variable name for thermo every does not exist
    Self-explanatory.
Variable name for velocity set does not exist
    Self-explanatory.
Variable name must be alphanumeric or underscore characters
    Self-explanatory.

Velocity command before simulation box is defined
> The velocity command cannot be used before a read_data, read_restart, or create_box command.

Velocity command with no atoms existing
> A velocity command has been used, but no atoms yet exist.

Velocity ramp in z for a 2d problem
> Self-explanatory.

Velocity temperature ID does not compute temperature
> The compute ID given to the velocity command must compute temperature.

Verlet/split requires 2 partitions
> See the -partition command-line switch.

Verlet/split requires Rspace partition layout be multiple of Kspace partition layout in each dim
> This is controlled by the processors command.

Verlet/split requires Rspace partition size be multiple of Kspace partition size
> This is so there is an equal number of Rspace processors for every Kspace processor.

Virial was not tallied on needed timestep
> You are using a thermo keyword that requires potentials to have tallied the virial, but they didn't on this timestep. See the variable doc page for ideas on how to make this work.

Wall defined twice in fix wall command
> Self-explanatory.

Wall defined twice in fix wall/reflect command
> Self-explanatory.

Wall defined twice in fix wall/srd command
> Self-explanatory.

Water H epsilon must be 0.0 for pair style lj/cut/tip4p/long
> This is because LAMMPS does not compute the Lennard-Jones interactions with these particles for efficiency reasons.

Water H epsilon must be 0.0 for pair style lj/long/tip4p/long
> This is because LAMMPS does not compute the Lennard-Jones interactions with these particles for efficiency reasons.

World variable count doesn't match # of partitions
> A world-style variable must specify a number of values equal to the number of processor partitions.

Write_restart command before simulation box is defined
> The write_restart command cannot be used before a read_data, read_restart, or create_box command.

Zero length rotation vector with displace_atoms
> Self-explanatory.

Zero length rotation vector with fix move
> Self-explanatory.

Zero-length lattice orient vector
> Self-explanatory.

Warnings:

Adjusting Coulombic cutoff for MSM, new cutoff = %g
> The adjust/cutoff command is turned on and the Coulombic cutoff has been adjusted to match the user-specified accuracy.

Atom with molecule ID = 0 included in compute molecule group
> The group used in a compute command that operates on moleclues includes atoms with no molecule ID. This is probably not what you want.

Bond/angle/dihedral extent > half of periodic box length
> This is a restriction because LAMMPS can be confused about which image of an atom in the bonded interaction is the correct one to use. "Extent" in this context means the maximum end-to-end length of the bond/angle/dihedral. LAMMPS computes this by taking the maximum bond length, multiplying by the number of bonds in the interaction (e.g. 3 for a dihedral) and adding a small amount of stretch.

Both groups in compute group/group have a net charge; the Kspace boundary correction to energy will be

non-zero
> Self-explantory.

Broken bonds will not alter angles, dihedrals, or impropers
> See the doc page for fix bond/break for more info on this restriction.

Building an occasional neighobr list when atoms may have moved too far
> This can cause LAMMPS to crash when the neighbor list is built. The solution is to check for building the regular neighbor lists more frequently.

Cannot include log terms without 1/r terms; setting flagHI to 1
> Self-explanatory.

Cannot include log terms without 1/r terms; setting flagHI to 1.
> Self-explanatory.

Charges are set, but coulombic solver is not used
> The atom style supports charge, but this KSpace style does not include long-range Coulombics.

Compute cna/atom cutoff may be too large to find ghost atom neighbors
> The neighbor cutoff used may not encompass enough ghost atoms to perform this operation correctly.

Computing temperature of portions of rigid bodies
> The group defined by the temperature compute does not encompass all the atoms in one or more rigid bodies, so the change in degrees-of-freedom for the atoms in those partial rigid bodies will not be accounted for.

Created bonds will not create angles, dihedrals, or impropers
> See the doc page for fix bond/create for more info on this restriction.

Dihedral problem: %d %ld %d %d %d %d
> Conformation of the 4 listed dihedral atoms is extreme; you may want to check your simulation geometry.

Dump dcd/xtc timestamp may be wrong with fix dt/reset
> If the fix changes the timestep, the dump dcd file will not reflect the change.

Ewald/disp Newton solver failed, using old method to estimate g_ewald
> Self-explanatory.

FENE bond too long: %ld %d %d %g
> A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

FENE bond too long: %ld %g
> A FENE bond has stretched dangerously far. It's interaction strength will be truncated to attempt to prevent the bond from blowing up.

Fix SRD walls overlap but fix srd overlap not set
> You likely want to set this in your input script.

Fix bond/swap will ignore defined angles
> See the doc page for fix bond/swap for more info on this restriction.

Fix evaporate may delete atom with non-zero molecule ID
> This is probably an error, since you should not delete only one atom of a molecule.

Fix move does not update angular momentum
> Atoms store this quantity, but fix move does not (yet) update it.

Fix move does not update quaternions
> Atoms store this quantity, but fix move does not (yet) update it.

Fix recenter should come after all other integration fixes
> Other fixes may change the position of the center-of-mass, so fix recenter should come last.

Fix shake with rRESPA computes invalid pressures
> This is a known bug in LAMMPS that has not yet been fixed. If you use SHAKE with rRESPA and perform a constant volume simulation (e.g. using fix npt) this only affects the output pressure, not the dynamics of the simulation. If you use SHAKE with rRESPA and perform a constant pressure simulation (e.g. using fix npt) then you will be equilibrating to the wrong volume.

Fix srd SRD moves may trigger frequent reneighboring
> This is because the SRD particles may move long distances.

Fix srd grid size > 1/4 of big particle diameter

This may cause accuracy problems.

Fix srd particle moved outside valid domain

This may indicate a problem with your simulation parameters.

Fix srd particles may move > big particle diameter

This may cause accuracy problems.

Fix srd viscosity < 0.0 due to low SRD density

This may cause accuracy problems.

Fix thermal/conductivity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix thermal/conductivity comes first. If you are using fix ave/spatial to measure the temperature profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of the 2 fixes typically helps.

Fix viscosity comes before fix ave/spatial

The order of these 2 fixes in your input script is such that fix viscosity comes first. If you are using fix ave/spatial to measure the velocity profile induced by fix viscosity, then this may cause a glitch in the profile since you are averaging immediately after swaps have occurred. Flipping the order of the 2 fixes typically helps.

For better accuracy use 'pair_modify table 0'

The user-specified force accuracy cannot be achieved unless the table feature is disabled by using 'pair_modify table 0'.

Geometric mixing assumed for 1/r^6 coefficients

Self-explanatory.

Group for fix_modify temp != fix group

The fix_modify command is specifying a temperature computation that computes a temperature on a different group of atoms than the fix itself operates on. This is probably not what you want to do.

Improper problem: %d %ld %d %d %d %d

Conformation of the 4 listed improper atoms is extreme; you may want to check your simulation geometry.

Inconsistent image flags

The image flags for a pair on bonded atoms appear to be inconsistent. Inconsistent means that when the coordinates of the two atoms are unwrapped using the image flags, the two atoms are far apart. Specifically they are further apart than half a periodic box length. Or they are more than a box length apart in a non-periodic dimension. This is usually due to the initial data file not having correct image flags for the 2 atoms in a bond that straddles a periodic boundary. They should be different by 1 in that case. This is a warning because inconsistent image flags will not cause problems for dynamics or most LAMMPS simulations. However they can cause problems when such atoms are used with the fix rigid or replicate commands.

KIM Model does not provide `energy'; Potential energy will be zero

Self-explanatory.

KIM Model does not provide `forces'; Forces will be zero

Self-explanatory.

KIM Model does not provide `particleEnergy'; energy per atom will be zero

Self-explanatory.

KIM Model does not provide `particleVirial'; virial per atom will be zero

Self-explanatory.

Kspace_modify slab param < 2.0 may cause unphysical behavior

The kspace_modify slab parameter should be larger to insure periodic grids padded with empty space do not overlap.

Less insertions than requested

Less atom insertions occurred on this timestep due to the fix pour command than were scheduled. This is probably because there were too many overlaps detected.

Library error in lammps_gather_atoms

This library function cannot be used if atom IDs are not defined or are not consecutively numbered.

Library error in lammps_scatter_atoms

This library function cannot be used if atom IDs are not defined or are not consecutively numbered, or if no atom map is defined. See the atom_modify command for details about atom maps.

Lost atoms via change_box: original %ld current %ld

The command options you have used caused atoms to be lost.

Lost atoms via displace_atoms: original %ld current %ld

The command options you have used caused atoms to be lost.

Lost atoms: original %ld current %ld

Lost atoms are checked for each time thermo output is done. See the thermo_modify lost command for options. Lost atoms usually indicate bad dynamics, e.g. atoms have been blown far out of the simulation box, or moved futher than one processor's sub-domain away before reneighboring.

MSM mesh too small, increasing to 2 points in each direction

Self-explanatory.

Mismatch between velocity and compute groups

The temperature computation used by the velocity command will not be on the same group of atoms that velocities are being set for.

Mixing forced for LJ coefficients

Self-explanatory.

Mixing forced for lj coefficients

Self-explanatory.

More than one compute centro/atom

It is not efficient to use compute centro/atom more than once.

More than one compute cluster/atom

It is not efficient to use compute cluster/atom more than once.

More than one compute cna/atom defined

It is not efficient to use compute cna/atom more than once.

More than one compute contact/atom

It is not efficient to use compute contact/atom more than once.

More than one compute coord/atom

It is not efficient to use compute coord/atom more than once.

More than one compute damage/atom

It is not efficient to use compute ke/atom more than once.

More than one compute erotate/sphere/atom

It is not efficient to use compute erorate/sphere/atom more than once.

More than one compute ke/atom

It is not efficient to use compute ke/atom more than once.

More than one compute voronoi/atom command

It is not efficient to use compute voronoi/atom more than once.

More than one fix poems

It is not efficient to use fix poems more than once.

More than one fix rigid

It is not efficient to use fix rigid more than once.

Neighbor exclusions used with KSpace solver may give inconsistent Coulombic energies

This is because excluding specific pair interactions also excludes them from long-range interactions which may not be the desired effect. The special_bonds command handles this consistently by insuring excluded (or weighted) 1-2, 1-3, 1-4 interactions are treated consistently by both the short-range pair style and the long-range solver. This is not done for exclusions of charged atom pairs via the neigh_modify exclude command.

New thermo_style command, previous thermo_modify settings will be lost

If a thermo_style command is used after a thermo_modify command, the settings changed by the thermo_modify command will be reset to their default values. This is because the thermo_modify commmand acts on the currently defined thermo style, and a thermo_style command creates a new style.

No Kspace calculation with verlet/split

The 2nd partition performs a kspace calculation so the kspace_style command must be used.

No fixes defined, atoms won't move
> If you are not using a fix like nve, nvt, npt then atom velocities and coordinates will not be updated during timestepping.

No joints between rigid bodies, use fix rigid instead
> The bodies defined by fix poems are not connected by joints. POEMS will integrate the body motion, but it would be more efficient to use fix rigid.

Not using real units with pair reax
> This is most likely an error, unless you have created your own ReaxFF parameter file in a different set of units.

Number of MSM mesh points increased to be a multiple of 2
> MSM requires that the number of grid points in each direction be a multiple of two and the number of grid points in one or more directions have been adjusted to meet this requirement.

OMP_NUM_THREADS environment is not set.
> This environment variable must be set appropriately to use the USER-OMP pacakge.

One or more atoms are time integrated more than once
> This is probably an error since you typically do not want to advance the positions or velocities of an atom more than once per timestep.

One or more compute molecules has atoms not in group
> The group used in a compute command that operates on moleclues does not include all the atoms in some molecules. This is probably not what you want.

One or more respa levels compute no forces
> This is computationally inefficient.

Pair COMB charge %.10f with force %.10f hit max barrier
> Something is possibly wrong with your model.

Pair COMB charge %.10f with force %.10f hit min barrier
> Something is possibly wrong with your model.

Pair brownian needs newton pair on for momentum conservation
> Self-explanatory.

Pair dpd needs newton pair on for momentum conservation
> Self-explanatory.

Pair dsmc: num_of_collisions > number_of_A
> Collision model in DSMC is breaking down.

Pair dsmc: num_of_collisions > number_of_B
> Collision model in DSMC is breaking down.

Particle deposition was unsuccessful
> The fix deposit command was not able to insert as many atoms as needed. The requested volume fraction may be too high, or other atoms may be in the insertion region.

Reducing PPPM order b/c stencil extends beyond nearest neighbor processor
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Reducing PPPMDisp Coulomb order b/c stencil extends beyond neighbor processor.
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Reducing PPPMDisp Dispersion order b/c stencil extends beyond neighbor processor
> This may lead to a larger grid than desired. See the kspace_modify overlap command to prevent changing of the PPPM order.

Replacing a fix, but new group != old group
> The ID and style of a fix match for a fix you are changing with a fix command, but the new group you are specifying does not match the old group.

Replicating in a non-periodic dimension
> The parameters for a replicate command will cause a non-periodic dimension to be replicated; this may cause unwanted behavior.

Resetting reneighboring criteria during PRD
> A PRD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since

these settings were not in place, LAMMPS changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during TAD

A TAD simulation requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the PRD simulation.

Resetting reneighboring criteria during minimization

Minimization requires that neigh_modify settings be delay = 0, every = 1, check = yes. Since these settings were not in place, LAMMPS changed them and will restore them to their original values after the minimization.

Restart file used different # of processors

The restart file was written out by a LAMMPS simulation running on a different number of processors. Due to round-off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different 3d processor grid

The restart file was written out by a LAMMPS simulation running on a different 3d grid of processors. Due to round-off, the trajectories of your restarted simulation may diverge a little more quickly than if you ran on the same # of processors.

Restart file used different boundary settings, using restart file values

Your input script cannot change these restart file settings.

Restart file used different newton bond setting, using restart file value

The restart file value will override the setting in the input script.

Restart file used different newton pair setting, using input script value

The input script value will override the setting in the restart file.

Restart file version does not match LAMMPS version

This may cause problems when reading the restart file.

Restrain problem: %d %ld %d %d %d %d

Conformation of the 4 listed dihedral atoms is extreme; you may want to check your simulation geometry.

Running PRD with only one replica

This is allowed, but you will get no parallel speed-up.

SRD bin shifting turned on due to small lamda

This is done to try to preserve accuracy.

SRD bin size for fix srd differs from user request

Fix SRD had to adjust the bin size to fit the simulation box. See the cubic keyword if you want this message to be an error vs warning.

SRD bins for fix srd are not cubic enough

The bin shape is not within tolerance of cubic. See the cubic keyword if you want this message to be an error vs warning.

SRD particle %d started inside big particle %d on step %ld bounce %d

See the inside keyword if you want this message to be an error vs warning.

Shake determinant < 0.0

The determinant of the quadratic equation being solved for a single cluster specified by the fix shake command is numerically suspect. LAMMPS will set it to 0.0 and continue.

Should not allow rigid bodies to bounce off reflecting walls

LAMMPS allows this, but their dynamics are not computed correctly.

System is not charge neutral, net charge = %g

The total charge on all atoms on the system is not 0.0, which is not valid for the long-range Coulombic solvers.

Table inner cutoff >= outer cutoff

You specified an inner cutoff for a Coulombic table that is longer than the global cutoff. Probably not what you wanted.

Temperature for MSST is not for group all

User-assigned temperature to MSST fix does not compute temperature for all atoms. Since MSST

computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by MSST could be inaccurate.

Temperature for NPT is not for group all

User-assigned temperature to NPT fix does not compute temperature for all atoms. Since NPT computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure used by NPT could be inaccurate.

Temperature for fix modify is not for group all

The temperature compute is being used with a pressure calculation which does operate on group all, so this may be inconsistent.

Temperature for thermo pressure is not for group all

User-assigned temperature to thermo via the thermo_modify command does not compute temperature for all atoms. Since thermo computes a global pressure, the kinetic energy contribution from the temperature is assumed to also be for all atoms. Thus the pressure printed by thermo could be inaccurate.

Too many common neighbors in CNA %d times

More than the maximum # of neighbors was found multiple times. This was unexpected.

Too many inner timesteps in fix ttm

Self-explanatory.

Too many neighbors in CNA for %d atoms

More than the maximum # of neighbors was found multiple times. This was unexpected.

Triclinic box skew is large

The displacement in a skewed direction is normally required to be less than half the box length in that dimension. E.g. the xy tilt must be between -half and +half of the x box length. You have relaxed the constraint using the box tilt command, but the warning means that a LAMMPS simulation may be inefficient as a result.

Use special bonds = 0,1,1 with bond style fene

Most FENE models need this setting for the special_bonds command.

Use special bonds = 0,1,1 with bond style fene/expand

Most FENE models need this setting for the special_bonds command.

Using compute temp/deform with inconsistent fix deform remap option

Fix nvt/sllod assumes deforming atoms have a velocity profile provided by "remap v" or "remap none" as a fix deform option.

Using compute temp/deform with no fix deform defined

This is probably an error, since it makes little sense to use compute temp/deform in this case.

Using fix srd with box deformation but no SRD thermostat

The deformation will heat the SRD particles so this can be dangerous.

Using largest cutoff for buck/long/coul/long

Self-exlanatory.

Using largest cutoff for pair_style lj/long/coul/long

Self-explanatory.

Using largest cutoff for pair_style lj/long/tip4p/long

Self-explanatory.

Using pair tail corrections with nonperiodic system

This is probably a bogus thing to do, since tail corrections are computed by integrating the density of a periodic system out to infinity.

7. Example problems

The LAMMPS distribution includes an examples sub-directory with several sample problems. Each problem is in a sub-directory of its own. Most are 2d models so that they run quickly, requiring at most a couple of minutes to run on a desktop machine. Each problem has an input script (in.*) and produces a log file (log.*) and dump file (dump.*) when it runs. Some use a data file (data.*) of initial coordinates as additional input. A few sample log file outputs on different machines and different numbers of processors are included in the directories to compare your answers to. E.g. a log file like log.crack.foo.P means it ran on P processors of machine "foo".

For examples that use input data files, many of them were produced by Pizza.py or setup tools described in the Additional Tools section of the LAMMPS documentation and provided with the LAMMPS distribution.

If you uncomment the dump command in the input script, a text dump file will be produced, which can be animated by various visualization programs. It can also be animated using the xmovie tool described in the Additional Tools section of the LAMMPS documentation.

If you uncomment the dump image command in the input script, and assuming you have built LAMMPS with a JPG library, JPG snapshot images will be produced when the simulation runs. They can be quickly post-processed into a movie using commands described on the dump image doc page.

Animations of many of these examples can be viewed on the Movies section of the LAMMPS WWW Site.

These are the sample problems in the examples sub-directories:

body	body particles, 2d system
colloid	big colloid particles in a small particle solvent, 2d system
comb	models using the COMB potential
crack	crack propagation in a 2d solid
dipole	point dipolar particles, 2d system
dreiding	methanol via Dreiding FF
eim	NaCl using the EIM potential
ellipse	ellipsoidal particles in spherical solvent, 2d system
flow	Couette and Poiseuille flow in a 2d channel
friction	frictional contact of spherical asperities between 2d surfaces
gpu	use of the GPU package for GPU acceleration
hugoniostat	Hugoniostat shock dynamics
indent	spherical indenter into a 2d solid
kim	use of potentials in Knowledge Base for Interatomic Models (KIM)
line	line segment particles in 2d rigid bodies
meam	MEAM test for SiC and shear (same as shear examples)
melt	rapid melt of 3d LJ system
micelle	self-assembly of small lipid-like molecules into 2d bilayers
min	energy minimization of 2d LJ melt
msst	MSST shock dynamics
neb	nudged elastic band (NEB) calculation for barrier finding
nemd	non-equilibrium MD of 2d sheared system

obstacle	flow around two voids in a 2d channel
peptide	dynamics of a small solvated peptide chain (5-mer)
peri	Peridynamic model of cylinder impacted by indenter
pour	pouring of granular particles into a 3d box, then chute flow
prd	parallel replica dynamics of vacancy diffusion in bulk Si
reax	RDX and TATB models using the ReaxFF
rigid	rigid bodies modeled as independent or coupled
shear	sideways shear applied to 2d solid, with and without a void
srd	stochastic rotation dynamics (SRD) particles as solvent
tad	temperature-accelerated dynamics of vacancy diffusion in bulk Si
tri	triangular particles in rigid bodies

Here is how you might run and visualize one of the sample problems:

```
cd indent
cp ../../src/lmp_linux .          # copy LAMMPS executable to this dir
lmp_linux <in.indent              # run the problem
```

Running the simulation produces the files *dump.indent* and *log.lammps*. You can visualize the dump file as follows:

```
../../tools/xmovie/xmovie -scale dump.indent
```

If you uncomment the dump image line(s) in the input script a series of JPG images will be produced by the run. These can be viewed individually or turned into a movie or animated by tools like ImageMagick or QuickTime or various Windows-based tools. See the dump image doc page for more details. E.g. this Imagemagick command would create a GIF file suitable for viewing in a browser.

```
% convert -loop 1 *.jpg foo.gif
```

There is also a COUPLE directory with examples of how to use LAMMPS as a library, either by itself or in tandem with another code or library. See the COUPLE/README file to get started.

There is also an ELASTIC directory with an example script for computing elastic constants, using a zero temperature Si example. See the in.elastic file for more info.

There is also a USER directory which contains subdirectories of user-provided examples for user packages. See the README files in those directories for more info. See the Section_start.html file for more info about user packages.

LIGGGHTS Users Manual

3. Commands

This section describes what granular models can be used along with [pair gran](#) and [fix wall/gran](#)

cohesion commands

Click on the style itself for a full description:

[sjkr](#) [sjkr2](#)

model commands

Click on the style itself for a full description:

[hertz](#) [hertz/stiffness](#) [hooke](#) [hooke/stiffness](#)

rolling_friction commands

Click on the style itself for a full description:

[cdt](#) [epsd](#) [epsd2](#)

tangential commands

Click on the style itself for a full description:

[history](#) [no_history](#)

13. Future and history

This section lists features we plan to add to LAMMPS, features of previous versions of LAMMPS, and features of other parallel molecular dynamics codes our group has distributed.

13.1 Coming attractions

The Wish list link on the LAMMPS WWW page gives a list of features we are hoping to add to LAMMPS in the future, including contact names of individuals you can email if you are interested in contributing to the developement or would be a future user of that feature.

You can also send email to the developers if you want to add your wish to the list.

13.2 Past versions

LAMMPS development began in the mid 1990s under a cooperative research & development agreement (CRADA) between two DOE labs (Sandia and LLNL) and 3 companies (Cray, Bristol Myers Squibb, and Dupont). The goal was to develop a large-scale parallel classical MD code; the coding effort was led by Steve Plimpton at Sandia.

After the CRADA ended, a final F77 version, LAMMPS 99, was released. As development of LAMMPS continued at Sandia, its memory management was converted to F90; a final F90 version was released as LAMMPS 2001.

The current LAMMPS is a rewrite in C++ and was first publicly released as an open source code in 2004. It includes many new features beyond those in LAMMPS 99 or 2001. It also includes features from older parallel MD codes written at Sandia, namely ParaDyn, Warp, and GranFlow (see below).

In late 2006 we began merging new capabilities into LAMMPS that were developed by Aidan Thompson at Sandia for his MD code GRASP, which has a parallel framework similar to LAMMPS. Most notably, these have included many-body potentials - Stillinger-Weber, Tersoff, ReaxFF - and the associated charge-equilibration routines needed for ReaxFF.

The History link on the LAMMPS WWW page gives a timeline of features added to the C++ open-source version of LAMMPS over the last several years.

These older codes are available for download from the LAMMPS WWW site, except for Warp & GranFlow which were primarily used internally. A brief listing of their features is given here.

LAMMPS 2001

- F90 + MPI
- dynamic memory
- spatial-decomposition parallelism
- NVE, NVT, NPT, NPH, rRESPA integrators
- LJ and Coulombic pairwise force fields
- all-atom, united-atom, bead-spring polymer force fields
- CHARMM-compatible force fields

tangential commands   1123

- class 2 force fields
- 3d/2d Ewald & PPPM
- various force and temperature constraints
- SHAKE
- Hessian-free truncated-Newton minimizer
- user-defined diagnostics

LAMMPS 99

- F77 + MPI
- static memory allocation
- spatial-decomposition parallelism
- most of the LAMMPS 2001 features with a few exceptions
- no 2d Ewald & PPPM
- molecular force fields are missing a few CHARMM terms
- no SHAKE

Warp

- F90 + MPI
- spatial-decomposition parallelism
- embedded atom method (EAM) metal potentials + LJ
- lattice and grain-boundary atom creation
- NVE, NVT integrators
- boundary conditions for applying shear stresses
- temperature controls for actively sheared systems
- per-atom energy and centro-symmetry computation and output

ParaDyn

- F77 + MPI
- atom- and force-decomposition parallelism
- embedded atom method (EAM) metal potentials
- lattice atom creation
- NVE, NVT, NPT integrators
- all serial DYNAMO features for controls and constraints

GranFlow

- F90 + MPI
- spatial-decomposition parallelism
- frictional granular potentials
- NVE integrator
- boundary conditions for granular flow and packing and walls
- particle insertion

6. How-to discussions

This section describes how to perform common tasks using LAMMPS.

The example input scripts included in the LAMMPS distribution and highlighted in Section_example also show how to setup and run various kinds of simulations.

6.1 Restarting a simulation

There are 3 ways to continue a long LAMMPS simulation. Multiple run commands can be used in the same input script. Each run will continue from where the previous run left off. Or binary restart files can be saved to disk using the restart command. At a later time, these binary files can be read via a read_restart command in a new script. Or they can be converted to text data files using the -r command-line switch and read by a read_data command in a new script.

Here we give examples of 2 scripts that read either a binary restart file or a converted data file and then issue a new run command to continue where the previous run left off. They illustrate what settings must be made in the new script. Details are discussed in the documentation for the read_restart and read_data commands.

Look at the *in.chain* input script provided in the *bench* directory of the LAMMPS distribution to see the original script that these 2 scripts are based on. If that script had the line

```
restart         50 tmp.restart
```

added to it, it would produce 2 binary restart files (tmp.restart.50 and tmp.restart.100) as it ran.

This script could be used to read the 1st restart file and re-run the last 50 timesteps:

```
read_restart    tmp.restart.50

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

run             50
```

Note that the following commands do not need to be repeated because their settings are included in the restart file: *units, atom_style, special_bonds, pair_style, bond_style*. However these commands do need to be used, since their settings are not in the restart file: *neighbor, fix, timestep*.

If you actually use this script to perform a restarted run, you will notice that the thermodynamic data match at step 50 (if you also put a "thermo 50" command in the original script), but do not match at step 100. This is because the fix langevin command uses random numbers in a way that does not allow for perfect restarts.

As an alternate approach, the restart file could be converted to a data file as follows:

```
lmp_g++ -r tmp.restart.50 tmp.restart.data
```

Then, this script could be used to re-run the last 50 steps:

```
units           lj
atom_style      bond
pair_style      lj/cut 1.12
pair_modify     shift yes
bond_style      fene
special_bonds   0.0 1.0 1.0

read_data       tmp.restart.data

neighbor        0.4 bin
neigh_modify    every 1 delay 1

fix             1 all nve
fix             2 all langevin 1.0 1.0 10.0 904297

timestep        0.012

reset_timestep  50
run             50
```

Note that nearly all the settings specified in the original *in.chain* script must be repeated, except the *pair_coeff* and *bond_coeff* commands since the new data file lists the force field coefficients. Also, the reset_timestep command is used to tell LAMMPS the current timestep. This value is stored in restart files, but not in data files.

6.2 2d simulations

Use the dimension command to specify a 2d simulation.

Make the simulation box periodic in z via the boundary command. This is the default.

If using the create box command to define a simulation box, set the z dimensions narrow, but finite, so that the create_atoms command will tile the 3d simulation box with a single z plane of atoms - e.g.

```
create box 1 -10 10 -10 10 -0.25 0.25
```

If using the read data command to read in a file of atom coordinates, set the "zlo zhi" values to be finite but narrow, similar to the create_box command settings just described. For each atom in the file, assign a z coordinate so it falls inside the z-boundaries of the box - e.g. 0.0.

Use the fix enforce2d command as the last defined fix to insure that the z-components of velocities and forces are zeroed out every timestep. The reason to make it the last fix is so that any forces induced by other fixes will be zeroed out.

Many of the example input scripts included in the LAMMPS distribution are for 2d models.

IMPORTANT NOTE: Some models in LAMMPS treat particles as finite-size spheres, as opposed to point particles. In 2d, the particles will still be spheres, not disks, meaning their moment of inertia will be the same as in 3d.

6.3 CHARMM, AMBER, and DREIDING force fields

A force field has 2 parts: the formulas that define it and the coefficients used for a particular system. Here we only discuss formulas implemented in LAMMPS that correspond to formulas commonly used in the CHARMM, AMBER, and DREIDING force fields. Setting coefficients is done in the input data file via the read_data command or in the input script with commands like pair_coeff or bond_coeff. See Section_tools for additional tools that can use CHARMM or AMBER to assign force field coefficients and convert their output into LAMMPS input.

See (MacKerell) for a description of the CHARMM force field. See (Cornell) for a description of the AMBER force field.

These style choices compute force field formulas that are consistent with common options in CHARMM or AMBER. See each command's documentation for the formula it computes.

- bond_style harmonic
- angle_style charmm
- dihedral_style charmm
- pair_style lj/charmm/coul/charmm
- pair_style lj/charmm/coul/charmm/implicit
- pair_style lj/charmm/coul/long

- special_bonds charmm
- special_bonds amber

DREIDING is a generic force field developed by the Goddard group at Caltech and is useful for predicting structures and dynamics of organic, biological and main-group inorganic molecules. The philosophy in DREIDING is to use general force constants and geometry parameters based on simple hybridization considerations, rather than individual force constants and geometric parameters that depend on the particular combinations of atoms involved in the bond, angle, or torsion terms. DREIDING has an explicit hydrogen bond term to describe interactions involving a hydrogen atom on very electronegative atoms (N, O, F).

See (Mayo) for a description of the DREIDING force field

These style choices compute force field formulas that are consistent with the DREIDING force field. See each command's documentation for the formula it computes.

- [bond_style](#) harmonic
- [bond_style](#) morse

- [angle_style](#) harmonic
- [angle_style](#) cosine
- [angle_style](#) cosine/periodic

- [dihedral_style](#) charmm
- [improper_style](#) umbrella

- [pair_style](#) buck
- [pair_style](#) buck/coul/cut
- [pair_style](#) buck/coul/long
- [pair_style](#) lj/cut
- [pair_style](#) lj/cut/coul/cut
- [pair_style](#) lj/cut/coul/long

- [pair_style](#) hbond/dreiding/lj
- [pair_style](#) hbond/dreiding/morse

- [special_bonds](#) dreiding

6.4 Running multiple simulations from one input script

This can be done in several ways. See the documentation for individual commands for more details on how these examples work.

If "multiple simulations" means continue a previous simulation for more timesteps, then you simply use the [run](#) command multiple times. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
run 10000
run 10000
run 10000
run 10000
```

would run 5 successive simulations of the same system for a total of 50,000 timesteps.

If you wish to run totally different simulations, one after the other, the [clear](#) command can be used in between them to re-initialize LAMMPS. For example, this script

```
units lj
atom_style atomic
read_data data.lj
run 10000
clear
units lj
atom_style atomic
read_data data.lj.new
run 10000
```

would run 2 independent simulations, one after the other.

For large numbers of independent simulations, you can use [variables](#) and the [next](#) and [jump](#) commands to loop over the same input script multiple times with different settings. For example, this script, named in.polymer

```
variable d index run1 run2 run3 run4 run5 run6 run7 run8
shell cd $d
read_data data.polymer
run 10000
shell cd ..
clear
next d
jump in.polymer
```

would run 8 simulations in different directories, using a data.polymer file in each directory. The same concept could be used to run the same system at 8 different temperatures, using a temperature variable and storing the output in different log and dump files, for example

```
variable a loop 8
variable t index 0.8 0.85 0.9 0.95 1.0 1.05 1.1 1.15
log log.$a
read data.polymer
velocity all create $t 352839
fix 1 all nvt $t $t 100.0
dump 1 all atom 1000 dump.$a
run 100000
next t
next a
jump in.polymer
```

All of the above examples work whether you are running on 1 or multiple processors, but assumed you are running LAMMPS on a single partition of processors. LAMMPS can be run on multiple partitions via the "-partition" command-line switch as described in [this section](#) of the manual.

In the last 2 examples, if LAMMPS were run on 3 partitions, the same scripts could be used if the "index" and "loop" variables were replaced with *universe*-style variables, as described in the [variable](#) command. Also, the "next t" and "next a" commands would need to be replaced with a single "next a t" command. With these modifications, the 8 simulations of each script would run on the 3 partitions one after the other until all were finished. Initially, 3 simulations would be started simultaneously, one on each partition. When one finished, that partition would then start the 4th simulation, and so forth, until all 8 were completed.

6.5 Multi-replica simulations

Several commands in LAMMPS run mutli-replica simulations, meaning that multiple instances (replicas) of your simulation are run simultaneously, with small amounts of data exchanged between replicas periodically.

These are the relevant commands:

- [neb](#) for nudged elastic band calculations
- [prd](#) for parallel replica dynamics
- [tad](#) for temperature accelerated dynamics
- [temper](#) for parallel tempering

NEB is a method for finding transition states and barrier energies. PRD and TAD are methods for performing accelerated dynamics to find and perform infrequent events. Parallel tempering or replica exchange runs different replicas at a series of temperature to facilitate rare-event sampling.

These command can only be used if LAMMPS was built with the "replica" package. See the Making LAMMPS section for more info on packages.

In all these cases, you must run with one or more processors per replica. The processors assigned to each replica are determined at run-time by using the -partition command-line switch to launch LAMMPS on multiple partitions, which in this context are the same as replicas. E.g. these commands:

```
mpirun -np 16 lmp_linux -partition 8x2 -in in.temper
mpirun -np 8 lmp_linux -partition 8x1 -in in.neb
```

would each run 8 replicas, on either 16 or 8 processors. Note the use of the -in command-line switch to specify the input script which is required when running in multi-replica mode.

Also note that with MPI installed on a machine (e.g. your desktop), you can run on more (virtual) processors than you have physical processors. Thus the above commands could be run on a single-processor (or few-processor) desktop so that you can run a multi-replica simulation on more replicas than you have physical processors.

6.6 Granular models

Granular system are composed of spherical particles with a diameter, as opposed to point particles. This means they have an angular velocity and torque can be imparted to them to cause them to rotate.

To run a simulation of a granular model, you will want to use the following commands:

- atom_style sphere
- fix nve/sphere
- fix gravity

This compute

- compute erotate/sphere

calculates rotational kinetic energy which can be output with thermodynamic info.

Use one of these 3 pair potentials, which compute forces and torques between interacting pairs of particles:

- pair_style gran/history
- pair_style gran/no_history
- pair_style gran/hertzian

These commands implement fix options specific to granular systems:

- fix freeze
- fix pour
- fix viscous
- fix wall/gran

The fix style *freeze* zeroes both the force and torque of frozen atoms, and should be used for granular system instead of the fix style *setforce*.

For computational efficiency, you can eliminate needless pairwise computations between frozen atoms by using this command:

- [neigh_modify](#) exclude

6.7 TIP3P water model

The TIP3P water model as implemented in CHARMM [(MacKerell)](#) specifies a 3-site rigid water molecule with charges and Lennard-Jones parameters assigned to each of the 3 atoms. In LAMMPS the [fix shake](#) command can be used to hold the two O-H bonds and the H-O-H angle rigid. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid TIP3P-CHARMM model with a cutoff. The K values can be used if a flexible TIP3P model (without fix shake) is desired. If the LJ epsilon and sigma for HH and OH are set to 0.0, it corresponds to the original 1983 TIP3P model [(Jorgensen)](#).

O mass = 15.9994
H mass = 1.008
O charge = -0.834
H charge = 0.417
LJ epsilon of OO = 0.1521
LJ sigma of OO = 3.1507
LJ epsilon of HH = 0.0460
LJ sigma of HH = 0.4000
LJ epsilon of OH = 0.0836
LJ sigma of OH = 1.7753
K of OH bond = 450
r0 of OH bond = 0.9572
K of HOH angle = 55
theta of HOH angle = 104.52

These are the parameters to use for TIP3P with a long-range Coulombic solver (e.g. Ewald or PPPM in LAMMPS), see [(Price)](#) for details:

O mass = 15.9994
H mass = 1.008
O charge = -0.830
H charge = 0.415
LJ epsilon of OO = 0.102
LJ sigma of OO = 3.188
LJ epsilon, sigma of OH, HH = 0.0
K of OH bond = 450
r0 of OH bond = 0.9572
K of HOH angle = 55
theta of HOH angle = 104.52

Wikipedia also has a nice article on [water models](#).

6.8 TIP4P water model

The four-point TIP4P rigid water model extends the traditional three-point TIP3P model by adding an additional site, usually massless, where the charge associated with the oxygen atom is placed. This site M is located at a fixed distance away from the oxygen along the bisector of the HOH bond angle. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

A TIP4P model is run with LAMMPS using either this command for a cutoff model:

pair_style lj/cut/tip4p/cut

or these two commands for a long-range model:

- pair_style lj/cut/tip4p/long
- kspace_style pppm/tip4p

For both models, the bond lengths and bond angles should be held fixed using the fix shake command.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid TIP4P model with a cutoff (Jorgensen). Note that the OM distance is specified in the pair_style command, not as part of the pair coefficients.

O mass = 15.9994
H mass = 1.008
O charge = -1.040
H charge = 0.520
r0 of OH bond = 0.9572
theta of HOH angle = 104.52
OM distance = 0.15
LJ epsilon of O-O = 0.1550
LJ sigma of O-O = 3.1536
LJ epsilon, sigma of OH, HH = 0.0
Coulombic cutoff = 8.5

For the TIP4/Ice model (J Chem Phys, 122, 234511 (2005); http://dx.doi.org/10.1063/1.1931662) these values can be used:

O mass = 15.9994
H mass = 1.008
O charge = -1.1794
H charge = 0.5897
r0 of OH bond = 0.9572
theta of HOH angle = 104.52
OM distance = 0.1577
LJ epsilon of O-O = 0.21084
LJ sigma of O-O = 3.1668
LJ epsilon, sigma of OH, HH = 0.0
Coulombic cutoff = 8.5

For the TIP4P/2005 model (J Chem Phys, 123, 234505 (2005); http://dx.doi.org/10.1063/1.2121687), these values can be used:

O mass = 15.9994
H mass = 1.008
O charge = -1.1128
H charge = 0.5564
r0 of OH bond = 0.9572
theta of HOH angle = 104.52
OM distance = 0.1546
LJ epsilon of O-O = 0.1852
LJ sigma of O-O = 3.1589

LJ epsilon, sigma of OH, HH = 0.0
Coulombic cutoff = 8.5

These are the parameters to use for TIP4P with a long-range Coulombic solver (e.g. Ewald or PPPM in LAMMPS):

O mass = 15.9994
H mass = 1.008
O charge = -1.0484
H charge = 0.5242
r0 of OH bond = 0.9572
theta of HOH angle = 104.52
OM distance = 0.1250
LJ epsilon of O-O = 0.16275
LJ sigma of O-O = 3.16435
LJ epsilon, sigma of OH, HH = 0.0

Note that the when using the TIP4P pair style, the neighobr list cutoff for Coulomb interactions is effectively extended by a distance 2 * (OM distance), to account for the offset distance of the fictitious charges on O atoms in water molecules. Thus it is typically best in an efficiency sense to use a LJ cutoff >= Coulomb cutoff + 2*(OM distance), to shrink the size of the neighbor list. This leads to slightly larger cost for the long-range calculation, so you can test the trade-off for your model. The OM distance and the LJ and Coulombic cutoffs are set in the pair_style lj/cut/tip4p/long command.

Wikipedia also has a nice article on water models.

6.9 SPC water model

The SPC water model specifies a 3-site rigid water molecule with charges and Lennard-Jones parameters assigned to each of the 3 atoms. In LAMMPS the fix shake command can be used to hold the two O-H bonds and the H-O-H angle rigid. A bond style of *harmonic* and an angle style of *harmonic* or *charmm* should also be used.

These are the additional parameters (in real units) to set for O and H atoms and the water molecule to run a rigid SPC model.

O mass = 15.9994
H mass = 1.008
O charge = -0.820
H charge = 0.410
LJ epsilon of OO = 0.1553
LJ sigma of OO = 3.166
LJ epsilon, sigma of OH, HH = 0.0
r0 of OH bond = 1.0
theta of HOH angle = 109.47

Note that as originally proposed, the SPC model was run with a 9 Angstrom cutoff for both LJ and Coulommbic terms. It can also be used with long-range Coulombics (Ewald or PPPM in LAMMPS), without changing any of the parameters above, though it becomes a different model in that mode of usage.

The SPC/E (extended) water model is the same, except the partial charge assignemnts change:

O charge = -0.8476

H charge = 0.4238

See the [(Berendsen)](#) reference for more details on both the SPC and SPC/E models.

Wikipedia also has a nice article on [water models](#).

6.10 Coupling LAMMPS to other codes

LAMMPS is designed to allow it to be coupled to other codes. For example, a quantum mechanics code might compute forces on a subset of atoms and pass those forces to LAMMPS. Or a continuum finite element (FE) simulation might use atom positions as boundary conditions on FE nodal points, compute a FE solution, and return interpolated forces on MD atoms.

LAMMPS can be coupled to other codes in at least 3 ways. Each has advantages and disadvantages, which you'll have to think about in the context of your application.

(1) Define a new [fix](#) command that calls the other code. In this scenario, LAMMPS is the driver code. During its timestepping, the fix is invoked, and can make library calls to the other code, which has been linked to LAMMPS as a library. This is the way the [POEMS](#) package that performs constrained rigid-body motion on groups of atoms is hooked to LAMMPS. See the [fix_poems](#) command for more details. See [this section](#) of the documentation for info on how to add a new fix to LAMMPS.

(2) Define a new LAMMPS command that calls the other code. This is conceptually similar to method (1), but in this case LAMMPS and the other code are on a more equal footing. Note that now the other code is not called during the timestepping of a LAMMPS run, but between runs. The LAMMPS input script can be used to alternate LAMMPS runs with calls to the other code, invoked via the new command. The [run](#) command facilitates this with its *every* option, which makes it easy to run a few steps, invoke the command, run a few steps, invoke the command, etc.

In this scenario, the other code can be called as a library, as in (1), or it could be a stand-alone code, invoked by a system() call made by the command (assuming your parallel machine allows one or more processors to start up another program). In the latter case the stand-alone code could communicate with LAMMPS thru files that the command writes and reads.

See [Section_modify](#) of the documentation for how to add a new command to LAMMPS.

(3) Use LAMMPS as a library called by another code. In this case the other code is the driver and calls LAMMPS as needed. Or a wrapper code could link and call both LAMMPS and another code as libraries. Again, the [run](#) command has options that allow it to be invoked with minimal overhead (no setup or clean-up) if you wish to do multiple short runs, driven by another program.

Examples of driver codes that call LAMMPS as a library are included in the examples/COUPLE directory of the LAMMPS distribution; see examples/COUPLE/README for more details:

- simple: simple driver programs in C++ and C which invoke LAMMPS as a library
- lammps_quest: coupling of LAMMPS and [Quest](#), to run classical MD with quantum forces calculated by a density functional code
- lammps_spparks: coupling of LAMMPS and [SPPARKS](#), to couple a kinetic Monte Carlo model for grain growth using MD to calculate strain induced across grain boundaries

[This section](#) of the documentation describes how to build LAMMPS as a library. Once this is done, you can interface with LAMMPS either via C++, C, Fortran, or Python (or any other language that supports a vanilla C-like interface). For example, from C++ you could create one (or more) "instances" of LAMMPS, pass it an

input script to process, or execute individual commands, all by invoking the correct class methods in LAMMPS. From C or Fortran you can make function calls to do the same things. See Section_python of the manual for a description of the Python wrapper provided with LAMMPS that operates through the LAMMPS library interface.

The files src/library.cpp and library.h contain the C-style interface to LAMMPS. See Section_howto 19 of the manual for a description of the interface and how to extend it for your needs.

Note that the lammps_open() function that creates an instance of LAMMPS takes an MPI communicator as an argument. This means that instance of LAMMPS will run on the set of processors in the communicator. Thus the calling code can run LAMMPS on all or a subset of processors. For example, a wrapper script might decide to alternate between LAMMPS and another code, allowing them both to run on all the processors. Or it might allocate half the processors to LAMMPS and half to the other code and run both codes simultaneously before syncing them up periodically. Or it might instantiate multiple instances of LAMMPS to perform different calculations.

6.11 Visualizing LAMMPS snapshots

LAMMPS itself does not do visualization, but snapshots from LAMMPS simulations can be visualized (and analyzed) in a variety of ways.

LAMMPS snapshots are created by the dump command which can create files in several formats. The native LAMMPS dump format is a text file (see "dump atom" or "dump custom") which can be visualized by the xmovie program, included with the LAMMPS package. This produces simple, fast 2d projections of 3d systems, and can be useful for rapid debugging of simulation geometry and atom trajectories.

Several programs included with LAMMPS as auxiliary tools can convert native LAMMPS dump files to other formats. See the Section_tools doc page for details. The first is the ch2lmp tool, which contains a lammps2pdb Perl script which converts LAMMPS dump files into PDB files. The second is the lmp2arc tool which converts LAMMPS dump files into Accelrys' Insight MD program files. The third is the lmp2cfg tool which converts LAMMPS dump files into CFG files which can be read into the AtomEye visualizer.

A Python-based toolkit distributed by our group can read native LAMMPS dump files, including custom dump files with additional columns of user-specified atom information, and convert them to various formats or pipe them into visualization software directly. See the Pizza.py WWW site for details. Specifically, Pizza.py can convert LAMMPS dump files into PDB, XYZ, Ensight, and VTK formats. Pizza.py can pipe LAMMPS dump files directly into the Raster3d and RasMol visualization programs. Pizza.py has tools that do interactive 3d OpenGL visualization and one that creates SVG images of dump file snapshots.

LAMMPS can create XYZ files directly (via "dump xyz") which is a simple text-based file format used by many visualization programs including VMD.

LAMMPS can create DCD files directly (via "dump dcd") which can be read by VMD in conjunction with a CHARMM PSF file. Using this form of output avoids the need to convert LAMMPS snapshots to PDB files. See the dump command for more information on DCD files.

LAMMPS can create XTC files directly (via "dump xtc") which is GROMACS file format which can also be read by VMD for visualization. See the dump command for more information on XTC files.

6.12 Triclinic (non-orthogonal) simulation boxes

By default, LAMMPS uses an orthogonal simulation box to encompass the particles. The boundary command sets the boundary conditions of the box (periodic, non-periodic, etc). The orthogonal box has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by **a** = (xhi-xlo,0,0); **b** = (0,yhi-ylo,0); **c** = (0,0,zhi-zlo). The 6 parameters (xlo,xhi,ylo,yhi,zlo,zhi) are defined at the time the simulation box is created, e.g. by the create_box or read_data or read_restart commands. Additionally, LAMMPS defines box size parameters lx,ly,lz where lx = xhi-xlo, and similarly in the y and z dimensions. The 6 parameters, as well as lx,ly,lz, can be output via the thermo_style custom command.

LAMMPS also allows simulations to be performed in triclinic (non-orthogonal) simulation boxes shaped as a parallelepiped with triclinic symmetry. The parallelepiped has its "origin" at (xlo,ylo,zlo) and is defined by 3 edge vectors starting from the origin given by **a** = (xhi-xlo,0,0); **b** = (xy,yhi-ylo,0); **c** = (xz,yz,zhi-zlo). *xy,xz,yz* can be 0.0 or positive or negative values and are called "tilt factors" because they are the amount of displacement applied to faces of an originally orthogonal box to transform it into the parallelepiped. In LAMMPS the triclinic simulation box edge vectors **a**, **b**, and **c** cannot be arbitrary vectors. As indicated, **a** must lie on the positive x axis. **b** must lie in the xy plane, with strictly positive y component. **c** may have any orientation with strictly positive z component. The requirement that **a**, **b**, and **c** have strictly positive x, y, and z components, respectively, ensures that **a**, **b**, and **c** form a complete right-handed basis. These restrictions impose no loss of generality, since it is possible to rotate/invert any set of 3 crystal basis vectors so that they conform to the restrictions.

For example, assume that the 3 vectors **A,B,C** are the edge vectors of a general parallelepiped, where there is no restriction on **A,B,C** other than they form a complete right-handed basis i.e. **A** x **B** . **C** > 0. The equivalent LAMMPS **a,b,c** are a linear rotation of **A**, **B**, and **C** and can be computed as follows:

$$
(\mathbf{a} \quad \mathbf{b} \quad \mathbf{c}) = \begin{pmatrix} a_x & b_x & c_x \\ 0 & b_y & c_y \\ 0 & 0 & c_z \end{pmatrix}
$$

$$
a_x = A
$$

$$
b_x = \mathbf{B} \cdot \hat{\mathbf{A}} = B \cos \gamma
$$

$$
b_y = |\hat{\mathbf{A}} \times \mathbf{B}| = B \sin \gamma = \sqrt{B^2 - b_x{}^2}
$$

$$
c_x = \mathbf{C} \cdot \hat{\mathbf{A}} = C \cos \beta
$$

$$
c_y = \mathbf{C} \cdot (\widehat{\mathbf{A} \times \mathbf{B}}) \times \hat{\mathbf{A}} = \frac{\mathbf{B} \cdot \mathbf{C} - b_x c_x}{b_y}
$$

$$
c_z = |\mathbf{C} \cdot (\widehat{\mathbf{A} \times \mathbf{B}})| = \sqrt{C^2 - c_x{}^2 - c_y{}^2}
$$

where A = |**A**| indicates the scalar length of **A**. The ^ hat symbol indicates the corresponding unit vector. *beta* and *gamma* are angles between the vectors described below. Note that by construction, **a**, **b**, and **c** have

strictly positive x, y, and z components, respectively. If it should happen that **A**, **B**, and **C** form a left-handed basis, then the above equations are not valid for **c**. In this case, it is necessary to first apply an inversion. This can be achieved by interchanging two basis vectors or by changing the sign of one of them.

For consistency, the same rotation/inversion applied to the basis vectors must also be applied to atom positions, velocities, and any other vector quantities. This can be conveniently achieved by first converting to fractional coordinates in the old basis and then converting to distance coordinates in the new basis. The transformation is given by the following equation:

$$\mathbf{x} = \begin{pmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{pmatrix} \cdot \frac{1}{V} \begin{pmatrix} \mathbf{B} \times \mathbf{C} \\ \mathbf{C} \times \mathbf{A} \\ \mathbf{A} \times \mathbf{B} \end{pmatrix} \cdot \mathbf{X}$$

where V is the volume of the box, **X** is the original vector quantity and **x** is the vector in the LAMMPS basis.

There is no requirement that a triclinic box be periodic in any dimension, though it typically should be in at least the 2nd dimension of the tilt (y in xy) if you want to enforce a shift in periodic boundary conditions across that boundary. Some commands that work with triclinic boxes, e.g. the fix deform and fix npt commands, require periodicity or non-shrink-wrap boundary conditions in specific dimensions. See the command doc pages for details.

The 9 parameters (xlo,xhi,ylo,yhi,zlo,zhi,xy,xz,yz) are defined at the time the simluation box is created. This happens in one of 3 ways. If the create_box command is used with a region of style *prism*, then a triclinic box is setup. See the region command for details. If the read_data command is used to define the simulation box, and the header of the data file contains a line with the "xy xz yz" keyword, then a triclinic box is setup. See the read_data command for details. Finally, if the read_restart command reads a restart file which was written from a simulation using a triclinic box, then a triclinic box will be setup for the restarted simulation.

Note that you can define a triclinic box with all 3 tilt factors = 0.0, so that it is initially orthogonal. This is necessary if the box will become non-orthogonal, e.g. due to the fix npt or fix deform commands. Alternatively, you can use the change_box command to convert a simulation box from orthogonal to triclinic and vice versa.

As with orthogonal boxes, LAMMPS defines triclinic box size parameters lx,ly,lz where lx = xhi-xlo, and similarly in the y and z dimensions. The 9 parameters, as well as lx,ly,lz, can be output via the thermo_style custom command.

To avoid extremely tilted boxes (which would be computationally inefficient), LAMMPS normally requires that no tilt factor can skew the box more than half the distance of the parallel box length, which is the 1st dimension in the tilt factor (x for xz). This is required both when the simulation box is created, e.g. via the create_box or read_data commands, as well as when the box shape changes dynamically during a simulation, e.g. via the fix deform or fix npt commands.

For example, if xlo = 2 and xhi = 12, then the x box length is 10 and the xy tilt factor must be between -5 and 5. Similarly, both xz and yz must be between -(xhi-xlo)/2 and +(yhi-ylo)/2. Note that this is not a limitation, since if the maximum tilt factor is 5 (as in this example), then configurations with tilt = ..., -15, -5, 5, 15, 25, ... are geometrically all equivalent. If the box tilt exceeds this limit during a dynamics run (e.g. via the fix deform command), then the box is "flipped" to an equivalent shape with a tilt factor within the bounds, so the run can continue. See the fix deform doc page for further details.

6.12 Triclinic (non-orthogonal) simulation boxes   1137

One exception to this rule is if the 1st dimension in the tilt factor (x for xy) is non-periodic. In that case, the limits on the tilt factor are not enforced, since flipping the box in that dimension does not change the atom positions due to non-periodicity. In this mode, if you tilt the system to extreme angles, the simulation will simply become inefficient, due to the highly skewed simulation box.

The limitation on not creating a simulation box with a tilt factor skewing the box more than half the distance of the parallel box length can be overridden via the box command. Setting the *tilt* keyword to *large* allows any tilt factors to be specified.

Box flips that may occur using the fix deform or fix npt commands can be turned off using the *flip no* option with either of the commands.

Note that if a simulation box has a large tilt factor, LAMMPS will run less efficiently, due to the large volume of communication needed to acquire ghost atoms around a processor's irregular-shaped sub-domain. For extreme values of tilt, LAMMPS may also lose atoms and generate an error.

Triclinic crystal structures are often defined using three lattice constants *a*, *b*, and *c*, and three angles *alpha*, *beta* and *gamma*. Note that in this nomenclature, the a, b, and c lattice constants are the scalar lengths of the edge vectors **a**, **b**, and **c** defined above. The relationship between these 6 quantities (a,b,c,alpha,beta,gamma) and the LAMMPS box sizes (lx,ly,lz) = (xhi-xlo,yhi-ylo,zhi-zlo) and tilt factors (xy,xz,yz) is as follows:

$$
\begin{aligned}
a &= \mathrm{lx} \\
b^2 &= \mathrm{ly}^2 + \mathrm{xy}^2 \\
c^2 &= \mathrm{lz}^2 + \mathrm{xz}^2 + \mathrm{yz}^2 \\
\cos\alpha &= \frac{\mathrm{xy} * \mathrm{xz} + \mathrm{ly} * \mathrm{yz}}{b * c} \\
\cos\beta &= \frac{\mathrm{xz}}{c} \\
\cos\gamma &= \frac{\mathrm{xy}}{b}
\end{aligned}
$$

The inverse relationship can be written as follows:

$$\mathrm{lx} = a$$

$$\mathrm{xy} = b\cos\gamma$$

$$\mathrm{xz} = c\cos\beta$$

$$\mathrm{ly}^2 = b^2 - \mathrm{xy}^2$$

$$\mathrm{yz} = \frac{b * c\cos\alpha - \mathrm{xy} * \mathrm{xz}}{\mathrm{ly}}$$

$$\mathrm{lz}^2 = c^2 - \mathrm{xz}^2 - \mathrm{yz}^2$$

The values of *a*, *b*, *c* , *alpha*, *beta* , and *gamma* can be printed out or accessed by computes using the thermo_style custom keywords *cella*, *cellb*, *cellc*, *cellalpha*, *cellbeta*, *cellgamma*, respectively.

As discussed on the dump command doc page, when the BOX BOUNDS for a snapshot is written to a dump file for a triclinic box, an orthogonal bounding box which encloses the triclinic simulation box is output, along with the 3 tilt factors (xy, xz, yz) of the triclinic box, formatted as follows:

```
ITEM: BOX BOUNDS xy xz yz
xlo_bound xhi_bound xy
ylo_bound yhi_bound xz
zlo_bound zhi_bound yz
```

This bounding box is convenient for many visualization programs and is calculated from the 9 triclinic box parameters (xlo,xhi,ylo,yhi,zlo,zhi,xy,xz,yz) as follows:

```
xlo_bound = xlo + MIN(0.0,xy,xz,xy+xz)
xhi_bound = xhi + MAX(0.0,xy,xz,xy+xz)
ylo_bound = ylo + MIN(0.0,yz)
yhi_bound = yhi + MAX(0.0,yz)
zlo_bound = zlo
zhi_bound = zhi
```

These formulas can be inverted if you need to convert the bounding box back into the triclinic box parameters, e.g. xlo = xlo_bound - MIN(0.0,xy,xz,xy+xz).

One use of triclinic simulation boxes is to model solid-state crystals with triclinic symmetry. The lattice command can be used with non-orthogonal basis vectors to define a lattice that will tile a triclinic simulation box via the create_atoms command.

A second use is to run Parinello-Rahman dyanamics via the fix npt command, which will adjust the xy, xz, yz tilt factors to compensate for off-diagonal components of the pressure tensor. The analalog for an energy minimization is the fix box/relax command.

A third use is to shear a bulk solid to study the response of the material. The fix deform command can be used for this purpose. It allows dynamic control of the xy, xz, yz tilt factors as a simulation runs. This is discussed in the next section on non-equilibrium MD (NEMD) simulations.

6.13 NEMD simulations

Non-equilibrium molecular dynamics or NEMD simulations are typically used to measure a fluid's rheological properties such as viscosity. In LAMMPS, such simulations can be performed by first setting up a non-orthogonal simulation box (see the preceding Howto section).

A shear strain can be applied to the simulation box at a desired strain rate by using the fix deform command. The fix nvt/sllod command can be used to thermostat the sheared fluid and integrate the SLLOD equations of motion for the system. Fix nvt/sllod uses compute temp/deform to compute a thermal temperature by subtracting out the streaming velocity of the shearing atoms. The velocity profile or other properties of the fluid can be monitored via the fix ave/spatial command.

As discussed in the previous section on non-orthogonal simulation boxes, the amount of tilt or skew that can be applied is limited by LAMMPS for computational efficiency to be 1/2 of the parallel box length. However, fix deform can continuously strain a box by an arbitrary amount. As discussed in the fix deform command, when the tilt value reaches a limit, the box is flipped to the opposite limit which is an equivalent tiling of periodic space. The strain rate can then continue to change as before. In a long NEMD simulation these box re-shaping events may occur many times.

In a NEMD simulation, the "remap" option of fix deform should be set to "remap v", since that is what fix nvt/sllod assumes to generate a velocity profile consistent with the applied shear strain rate.

An alternative method for calculating viscosities is provided via the fix viscosity command.

6.14 Finite-size spherical and aspherical particles

Typical MD models treat atoms or particles as point masses. Sometimes it is desirable to have a model with finite-size particles such as spheroids or ellipsoids or generalized aspherical bodies. The difference is that such particles have a moment of inertia, rotational energy, and angular momentum. Rotation is induced by torque coming from interactions with other particles.

LAMMPS has several options for running simulations with these kinds of particles. The following aspects are discussed in turn:

- atom styles
- pair potentials
- time integration
- computes, thermodynamics, and dump output
- rigid bodies composed of finite-size particles

Example input scripts for these kinds of models are in the body, colloid, dipole, ellipse, line, peri, pour, and tri directories of the examples directory in the LAMMPS distribution.

Atom styles

There are several atom styles that allow for definition of finite-size particles: sphere, dipole, ellipsoid, line, tri, peri, and body.

The sphere style defines particles that are spheriods and each particle can have a unique diameter and mass (or density). These particles store an angular velocity (omega) and can be acted upon by torque. The "set" command can be used to modify the diameter and mass of individual particles, after then are created.

The dipole style does not actually define finite-size particles, but is often used in conjunction with spherical particles, via a command like

```
atom_style hybrid sphere dipole
```

This is because when dipoles interact with each other, they induce torques, and a particle must be finite-size (i.e. have a moment of inertia) in order to respond and rotate. See the atom_style dipole command for details. The "set" command can be used to modify the orientation and length of the dipole moment of individual particles, after then are created.

The ellipsoid style defines particles that are ellipsoids and thus can be aspherical. Each particle has a shape, specified by 3 diameters, and mass (or density). These particles store an angular momentum and their orientation (quaternion), and can be acted upon by torque. They do not store an angular velocity (omega), which can be in a different direction than angular momentum, rather they compute it as needed. The "set" command can be used to modify the diameter, orientation, and mass of individual particles, after then are created. It also has a brief explanation of what quaternions are.

The line style defines line segment particles with two end points and a mass (or density). They can be used in 2d simulations, and they can be joined together to form rigid bodies which represent arbitrary polygons.

The tri style defines triangular particles with three corner points and a mass (or density). They can be used in 3d simulations, and they can be joined together to form rigid bodies which represent arbitrary particles with a triangulated surface.

The peri style is used with Peridynamic models and defines particles as having a volume, that is used internally in the pair_style peri potentials.

The body style allows for definition of particles which can represent complex entities, such as surface meshes of discrete points, collections of sub-particles, deformable objects, etc. The body style is discussed in more detail on the body doc page.

Note that if one of these atom styles is used (or multiple styles via the atom_style hybrid command), not all particles in the system are required to be finite-size or aspherical.

For example, in the ellipsoid style, if the 3 shape parameters are set to the same value, the particle will be a sphere rather than an ellipsoid. If the 3 shape parameters are all set to 0.0 or if the diameter is set to 0.0, it will be a point particle. In the line or tri style, if the lineflag or triflag is specified as 0, then it will be a point particle.

Some of the pair styles used to compute pairwise interactions between finite-size particles also compute the correct interaction with point particles as well, e.g. the interaction between a point particle and a finite-size particle or between two point particles. If necessary, pair_style hybrid can be used to insure the correct interactions are computed for the appropriate style of interactions. Likewise, using groups to partition particles (ellipsoids versus spheres versus point particles) will allow you to use the appropriate time integrators and temperature computations for each class of particles. See the doc pages for various commands for details.

Also note that for 2d simulations, atom styles sphere and ellipsoid still use 3d particles, rather than as circular disks or ellipses. This means they have the same moment of inertia as the 3d object. When temperature is computed, the correct degrees of freedom are used for rotation in a 2d versus 3d system.

Pair potentials

When a system with finite-size particles is defined, the particles will only rotate and experience torque if the force field computes such interactions. These are the various pair styles that generate torque:

- pair_style gran/history
- pair_style gran/hertzian
- pair_style gran/no_history

- pair_style dipole/cut
- pair_style gayberne
- pair_style resquared
- pair_style brownian
- pair_style lubricate
- pair_style line/lj
- pair_style tri/lj
- pair_style body

The granular pair styles are used with spherical particles. The dipole pair style is used with the dipole atom style, which could be applied to spherical or ellipsoidal particles. The GayBerne and REsquared potentials require ellipsoidal particles, though they will also work if the 3 shape parameters are the same (a sphere). The Brownian and lubrication potentials are used with spherical particles. The line, tri, and body potentials are used with line segment, triangular, and body particles respectively.

Time integration

There are several fixes that perform time integration on finite-size spherical particles, meaning the integrators update the rotational orientation and angular velocity or angular momentum of the particles:

- fix nve/sphere
- fix nvt/sphere
- fix npt/sphere

Likewise, there are 3 fixes that perform time integration on ellipsoidal particles:

- fix nve/asphere
- fix nvt/asphere
- fix npt/asphere

The advantage of these fixes is that those which thermostat the particles include the rotational degrees of freedom in the temperature calculation and thermostatting. The fix langevin command can also be used with its *omgea* or *angmom* options to thermostat the rotational degrees of freedom for spherical or ellipsoidal particles. Other thermostatting fixes only operate on the translational kinetic energy of finite-size particles.

These fixes perform constant NVE time integration on line segment, triangular, and body particles:

- fix nve/line
- fix nve/tri
- fix nve/body

Note that for mixtures of point and finite-size particles, these integration fixes can only be used with groups which contain finite-size particles.

Computes, thermodynamics, and dump output

There are several computes that calculate the temperature or rotational energy of spherical or ellipsoidal particles:

- compute temp/sphere
- compute temp/asphere
- compute erotate/sphere
- compute erotate/asphere

These include rotational degrees of freedom in their computation. If you wish the thermodynamic output of temperature or pressure to use one of these computes (e.g. for a system entirely composed of finite-size particles), then the compute can be defined and the [thermo_modify](#) command used. Note that by default thermodynamic quantities will be calculated with a temperature that only includes translational degrees of freedom. See the [thermo_style](#) command for details.

These commands can be used to output various attributes of finite-size particles:

- [dump custom](#)
- [compute property/atom](#)
- [dump local](#)
- [compute body/local](#)

Attributes include the dipole moment, the angular velocity, the angular momentum, the quaternion, the torque, the end-point and corner-point coordinates (for line and tri particles), and sub-particle attributes of body particles.

Rigid bodies composed of finite-size particles

The [fix rigid](#) command treats a collection of particles as a rigid body, computes its inertia tensor, sums the total force and torque on the rigid body each timestep due to forces on its constituent particles, and integrates the motion of the rigid body.

If any of the constituent particles of a rigid body are finite-size particles (spheres or ellipsoids or line segments or triangles), then their contribution to the inertia tensor of the body is different than if they were point particles. This means the rotational dynamics of the rigid body will be different. Thus a model of a dimer is different if the dimer consists of two point masses versus two spheroids, even if the two particles have the same mass. Finite-size particles that experience torque due to their interaction with other particles will also impart that torque to a rigid body they are part of.

See the "fix rigid" command for example of complex rigid-body models it is possible to define in LAMMPS.

Note that the [fix shake](#) command can also be used to treat 2, 3, or 4 particles as a rigid body, but it always assumes the particles are point masses.

Also note that body particles cannot be modeled with the [fix rigid](#) command. Body particles are treated by LAMMPS as single particles, though they can store internal state, such as a list of sub-particles. Individual body partices are typically treated as rigid bodies, and their motion integrated with a command like [fix nve/body](#). Interactions between pairs of body particles are computed via a command like [pair_style body](#).

6.15 Output from LAMMPS (thermo, dumps, computes, fixes, variables)

There are four basic kinds of LAMMPS output:

- [Thermodynamic output](#), which is a list of quantities printed every few timesteps to the screen and logfile.
- [Dump files](#), which contain snapshots of atoms and various per-atom values and are written at a specified frequency.
- Certain fixes can output user-specified quantities to files: [fix ave/time](#) for time averaging, [fix ave/spatial](#) for spatial averaging, and [fix print](#) for single-line output of [variables](#). Fix print can also output to the screen.
- [Restart files](#).

A simulation prints one set of thermodynamic output and (optionally) restart files. It can generate any number of dump files and fix output files, depending on what [dump](#) and [fix](#) commands you specify.

As discussed below, LAMMPS gives you a variety of ways to determine what quantities are computed and printed when the thermodynamics, dump, or fix commands listed above perform output. Throughout this discussion, note that users can also [add their own computes and fixes to LAMMPS](#) which can then generate values that can then be output with these commands.

The following sub-sections discuss different LAMMPS command related to output and the kind of data they operate on and produce:

- [Global/per-atom/local data](#)
- [Scalar/vector/array data](#)
- [Thermodynamic output](#)
- [Dump file output](#)
- [Fixes that write output files](#)
- [Computes that process output quantities](#)
- [Fixes that process output quantities](#)
- [Computes that generate values to output](#)
- [Fixes that generate values to output](#)
- [Variables that generate values to output](#)
- [Summary table of output options and data flow between commands](#)

Global/per-atom/local data

Various output-related commands work with three different styles of data: global, per-atom, or local. A global datum is one or more system-wide values, e.g. the temperature of the system. A per-atom datum is one or more values per atom, e.g. the kinetic energy of each atom. Local datums are calculated by each processor based on the atoms it owns, but there may be zero or more per atom, e.g. a list of bond distances.

Scalar/vector/array data

Global, per-atom, and local datums can each come in three kinds: a single scalar value, a vector of values, or a 2d array of values. The doc page for a "compute" or "fix" or "variable" that generates data will specify both the style and kind of data it produces, e.g. a per-atom vector.

When a quantity is accessed, as in many of the output commands discussed below, it can be referenced via the following bracket notation, where ID in this case is the ID of a compute. The leading "c_" would be replaced by "f_" for a fix, or "v_" for a variable:

c_ID	entire scalar, vector, or array
c_ID[I]	one element of vector, one column of array
c_ID[I][J]	one element of array

In other words, using one bracket reduces the dimension of the data once (vector -> scalar, array -> vector). Using two brackets reduces the dimension twice (array -> scalar). Thus a command that uses scalar values as input can typically also process elements of a vector or array.

Thermodynamic output

The frequency and format of thermodynamic output is set by the [thermo](#), [thermo_style](#), and [thermo_modify](#) commands. The [thermo_style](#) command also specifies what values are calculated and written out. Pre-defined keywords can be specified (e.g. press, etotal, etc). Three additional kinds of keywords can also be specified (c_ID, f_ID, v_name), where a [compute](#) or [fix](#) or [variable](#) provides the value to be output. In each case, the compute, fix, or variable must generate global values for input to the [thermo_style custom](#) command.

Note that thermodynamic output values can be "extensive" or "intensive". The former scale with the number of atoms in the system (e.g. total energy), the latter do not (e.g. temperature). The setting for thermo_modify norm determines whether extensive quantities are normalized or not. Computes and fixes produce either extensive or intensive values; see their individual doc pages for details. Equal-style variables produce only intensive values; you can include a division by "natoms" in the formula if desired, to make an extensive calculation produce an intensive result.

Dump file output

Dump file output is specified by the dump and dump_modify commands. There are several pre-defined formats (dump atom, dump xtc, etc).

There is also a dump custom format where the user specifies what values are output with each atom. Pre-defined atom attributes can be specified (id, x, fx, etc). Three additional kinds of keywords can also be specified (c_ID, f_ID, v_name), where a compute or fix or variable provides the values to be output. In each case, the compute, fix, or variable must generate per-atom values for input to the dump custom command.

There is also a dump local format where the user specifies what local values to output. A pre-defined index keyword can be specified to enumuerate the local values. Two additional kinds of keywords can also be specified (c_ID, f_ID), where a compute or fix or variable provides the values to be output. In each case, the compute or fix must generate local values for input to the dump local command.

Fixes that write output files

Several fixes take various quantities as input and can write output files: fix ave/time, fix ave/spatial, fix ave/histo, fix ave/correlate, and fix print.

The fix ave/time command enables direct output to a file and/or time-averaging of global scalars or vectors. The user specifies one or more quantities as input. These can be global compute values, global fix values, or variables of any style except the atom style which produces per-atom values. Since a variable can refer to keywords used by the thermo_style custom command (like temp or press) and individual per-atom values, a wide variety of quantities can be time averaged and/or output in this way. If the inputs are one or more scalar values, then the fix generate a global scalar or vector of output. If the inputs are one or more vector values, then the fix generates a global vector or array of output. The time-averaged output of this fix can also be used as input to other output commands.

The fix ave/spatial command enables direct output to a file of spatial-averaged per-atom quantities like those output in dump files, within 1d layers of the simulation box. The per-atom quantities can be atom density (mass or number) or atom attributes such as position, velocity, force. They can also be per-atom quantities calculated by a compute, by a fix, or by an atom-style variable. The spatial-averaged output of this fix can also be used as input to other output commands.

The fix ave/histo command enables direct output to a file of histogrammed quantities, which can be global or per-atom or local quantities. The histogram output of this fix can also be used as input to other output commands.

The fix ave/correlate command enables direct output to a file of time-correlated quantities, which can be global scalars. The correlation matrix output of this fix can also be used as input to other output commands.

The fix print command can generate a line of output written to the screen and log file or to a separate file, periodically during a running simulation. The line can contain one or more variable values for any style variable except the atom style). As explained above, variables themselves can contain references to global values generated by thermodynamic keywords, computes, fixes, or other variables, or to per-atom values for a specific atom. Thus the fix print command is a means to output a wide variety of quantities separate from normal thermodynamic or dump file output.

Computes that process output quantities

The [compute reduce](#) and [compute reduce/region](#) commands take one or more per-atom or local vector quantities as inputs and "reduce" them (sum, min, max, ave) to scalar quantities. These are produced as output values which can be used as input to other output commands.

The [compute slice](#) command take one or more global vector or array quantities as inputs and extracts a subset of their values to create a new vector or array. These are produced as output values which can be used as input to other output commands.

The [compute property/atom](#) command takes a list of one or more pre-defined atom attributes (id, x, fx, etc) and stores the values in a per-atom vector or array. These are produced as output values which can be used as input to other output commands. The list of atom attributes is the same as for the [dump custom](#) command.

The [compute property/local](#) command takes a list of one or more pre-defined local attributes (bond info, angle info, etc) and stores the values in a local vector or array. These are produced as output values which can be used as input to other output commands.

The [compute atom/molecule](#) command takes a list of one or more per-atom quantities (from a compute, fix, per-atom variable) and sums the quantities on a per-molecule basis. It produces a global vector or array as output values which can be used as input to other output commands.

Fixes that process output quantities

The [fix ave/atom](#) command performs time-averaging of per-atom vectors. The per-atom quantities can be atom attributes such as position, velocity, force. They can also be per-atom quantities calculated by a [compute](#), by a [fix](#), or by an atom-style [variable](#). The time-averaged per-atom output of this fix can be used as input to other output commands.

The [fix store/state](#) command can archive one or more per-atom attributes at a particular time, so that the old values can be used in a future calculation or output. The list of atom attributes is the same as for the [dump custom](#) command, including per-atom quantities calculated by a [compute](#), by a [fix](#), or by an atom-style [variable](#). The output of this fix can be used as input to other output commands.

Computes that generate values to output

Every [compute](#) in LAMMPS produces either global or per-atom or local values. The values can be scalars or vectors or arrays of data. These values can be output using the other commands described in this section. The doc page for each compute command describes what it produces. Computes that produce per-atom or local values have the word "atom" or "local" in their style name. Computes without the word "atom" or "local" produce global values.

Fixes that generate values to output

Some [fixes](#) in LAMMPS produces either global or per-atom or local values which can be accessed by other commands. The values can be scalars or vectors or arrays of data. These values can be output using the other commands described in this section. The doc page for each fix command tells whether it produces any output quantities and describes them.

Variables that generate values to output

Every [variables](#) defined in an input script generates either a global scalar value or a per-atom vector (only atom-style variables) when it is accessed. The formulas used to define equal- and atom-style variables can contain references to the thermodynamic keywords and to global and per-atom data generated by computes, fixes, and other variables. The values generated by variables can be output using the other commands

described in this section.

Summary table of output options and data flow between commands

This table summarizes the various commands that can be used for generating output from LAMMPS. Each command produces output data of some kind and/or writes data to a file. Most of the commands can take data from other commands as input. Thus you can link many of these commands together in pipeline form, where data produced by one command is used as input to another command and eventually written to the screen or to a file. Note that to hook two commands together the output and input data types must match, e.g. global/per-atom/local data and scalar/vector/array data.

Also note that, as described above, when a command takes a scalar as input, that could be an element of a vector or array. Likewise a vector input could be a column of an array.

Command	Input	Output
thermo_style custom	global scalars	screen, log file
dump custom	per-atom vectors	dump file
dump local	local vectors	dump file
fix print	global scalar from variable	screen, file
print	global scalar from variable	screen
computes	N/A	global/per-atom/local scalar/vector/array
fixes	N/A	global/per-atom/local scalar/vector/array
variables	global scalars, per-atom vectors	global scalar, per-atom vector
compute reduce	per-atom/local vectors	global scalar/vector
compute slice	global vectors/arrays	global vector/array
compute property/atom	per-atom vectors	per-atom vector/array
compute property/local	local vectors	local vector/array
compute atom/molecule	per-atom vectors	global vector/array
fix ave/atom	per-atom vectors	per-atom vector/array
fix ave/time	global scalars/vectors	global scalar/vector/array, file
fix ave/spatial	per-atom vectors	global array, file
fix ave/histo	global/per-atom/local scalars and vectors	global array, file
fix ave/correlate	global scalars	global array, file
fix store/state	per-atom vectors	per-atom vector/array

6.16 Thermostatting, barostatting, and computing temperature

Thermostatting means controlling the temperature of particles in an MD simulation. Barostatting means controlling the pressure. Since the pressure includes a kinetic component due to particle velocities, both these operations require calculation of the temperature. Typically a target temperature (T) and/or pressure (P) is specified by the user, and the thermostat or barostat attempts to equilibrate the system to the requested T and/or P.

Temperature is computed as kinetic energy divided by some number of degrees of freedom (and the Boltzmann constant). Since kinetic energy is a function of particle velocity, there is often a need to distinguish between a particle's advection velocity (due to some aggregate motiion of particles) and its thermal velocity. The sum of the two is the particle's total velocity, but the latter is often what is wanted to compute a temperature.

LAMMPS has several options for computing temperatures, any of which can be used in thermostatting and barostatting. These compute commands calculate temperature, and the compute pressure command calculates pressure.

- compute temp
- compute temp/sphere
- compute temp/asphere
- compute temp/com
- compute temp/deform
- compute temp/partial
- compute temp/profile
- compute temp/ramp
- compute temp/region

All but the first 3 calculate velocity biases (i.e. advection velocities) that are removed when computing the thermal temperature. Compute temp/sphere and compute temp/asphere compute kinetic energy for finite-size particles that includes rotational degrees of freedom. They both allow, as an extra argument, which is another temperature compute that subtracts a velocity bias. This allows the translational velocity of spherical or aspherical particles to be adjusted in prescribed ways.

Thermostatting in LAMMPS is performed by fixes, or in one case by a pair style. Four thermostatting fixes are currently available: Nose-Hoover (nvt), Berendsen, Langevin, and direct rescaling (temp/rescale). Dissipative particle dynamics (DPD) thermostatting can be invoked via the *dpd/tstat* pair style:

- fix nvt
- fix nvt/sphere
- fix nvt/asphere
- fix nvt/sllod
- fix temp/berendsen
- fix langevin
- fix temp/rescale
- pair_style dpd/tstat

Fix nvt only thermostats the translational velocity of particles. Fix nvt/sllod also does this, except that it subtracts out a velocity bias due to a deforming box and integrates the SLLOD equations of motion. See the NEMD simulations section of this page for further details. Fix nvt/sphere and fix nvt/asphere thermostat not only translation velocities but also rotational velocities for spherical and aspherical particles.

DPD thermostatting alters pairwise interactions in a manner analagous to the per-particle thermostatting of fix langevin.

Any of the thermostatting fixes can use temperature computes that remove bias for two purposes: (a) computing the current temperature to compare to the requested target temperature, and (b) adjusting only the thermal temperature component of the particle's velocities. See the doc pages for the individual fixes and for the fix_modify command for instructions on how to assign a temperature compute to a thermostatting fix. For example, you can apply a thermostat to only the x and z components of velocity by using it in conjunction with compute temp/partial.

IMPORTANT NOTE: Only the nvt fixes perform time integration, meaning they update the velocities and positions of particles due to forces and velocities respectively. The other thermostat fixes only adjust velocities; they do NOT perform time integration updates. Thus they should be used in conjunction with a constant NVE integration fix such as these:

- fix nve

- fix nve/sphere
- fix nve/asphere

Barostatting in LAMMPS is also performed by fixes. Two barosttating methods are currently available: Nose-Hoover (npt and nph) and Berendsen:

- fix npt
- fix npt/sphere
- fix npt/asphere
- fix nph
- fix press/berendsen

The fix npt commands include a Nose-Hoover thermostat and barostat. Fix nph is just a Nose/Hoover barostat; it does no thermostatting. Both fix nph and fix press/bernendsen can be used in conjunction with any of the thermostatting fixes.

As with the thermostats, fix npt and fix nph only use translational motion of the particles in computing T and P and performing thermo/barostatting. Fix npt/sphere and fix npt/asphere thermo/barostat using not only translation velocities but also rotational velocities for spherical and aspherical particles.

All of the barostatting fixes use the compute pressure compute to calculate a current pressure. By default, this compute is created with a simple compute temp (see the last argument of the compute pressure command), which is used to calculated the kinetic componenet of the pressure. The barostatting fixes can also use temperature computes that remove bias for the purpose of computing the kinetic componenet which contributes to the current pressure. See the doc pages for the individual fixes and for the fix_modify command for instructions on how to assign a temperature or pressure compute to a barostatting fix.

IMPORTANT NOTE: As with the thermostats, the Nose/Hoover methods (fix npt and fix nph) perform time integration. Fix press/berendsen does NOT, so it should be used with one of the constant NVE fixes or with one of the NVT fixes.

Finally, thermodynamic output, which can be setup via the thermo_style command, often includes temperature and pressure values. As explained on the doc page for the thermo_style command, the default T and P are setup by the thermo command itself. They are NOT the ones associated with any thermostatting or barostatting fix you have defined or with any compute that calculates a temperature or pressure. Thus if you want to view these values of T and P, you need to specify them explicitly via a thermo_style custom command. Or you can use the thermo_modify command to re-define what temperature or pressure compute is used for default thermodynamic output.

6.17 Walls

Walls in an MD simulation are typically used to bound particle motion, i.e. to serve as a boundary condition.

Walls in LAMMPS can be of rough (made of particles) or idealized surfaces. Ideal walls can be smooth, generating forces only in the normal direction, or frictional, generating forces also in the tangential direction.

Rough walls, built of particles, can be created in various ways. The particles themselves can be generated like any other particle, via the lattice and create_atoms commands, or read in via the read_data command.

Their motion can be constrained by many different commands, so that they do not move at all, move together as a group at constant velocity or in response to a net force acting on them, move in a prescribed fashion (e.g. rotate around a point), etc. Note that if a time integration fix like fix nve or fix nvt is not used with the group that contains wall particles, their positions and velocities will not be updated.

- [fix aveforce](#) - set force on particles to average value, so they move together
- [fix setforce](#) - set force on particles to a value, e.g. 0.0
- [fix freeze](#) - freeze particles for use as granular walls
- [fix nve/noforce](#) - advect particles by their velocity, but without force
- [fix move](#) - prescribe motion of particles by a linear velocity, oscillation, rotation, variable

The [fix move](#) command offers the most generality, since the motion of individual particles can be specified with [variable](#) formula which depends on time and/or the particle position.

For rough walls, it may be useful to turn off pairwise interactions between wall particles via the [neigh_modify exclude](#) command.

Rough walls can also be created by specifying frozen particles that do not move and do not interact with mobile particles, and then tethering other particles to the fixed particles, via a [bond](#). The bonded particles do interact with other mobile particles.

Idealized walls can be specified via several fix commands. [Fix wall/gran](#) creates frictional walls for use with granular particles; all the other commands create smooth walls.

- [fix wall/reflect](#) - reflective flat walls
- [fix wall/lj93](#) - flat walls, with Lennard-Jones 9/3 potential
- [fix wall/lj126](#) - flat walls, with Lennard-Jones 12/6 potential
- [fix wall/colloid](#) - flat walls, with [pair_style colloid](#) potential
- [fix wall/harmonic](#) - flat walls, with repulsive harmonic spring potential
- [fix wall/region](#) - use region surface as wall
- [fix wall/gran](#) - flat or curved walls with [pair_style granular](#) potential

The *lj93*, *lj126*, *colloid*, and *harmonic* styles all allow the flat walls to move with a constant velocity, or oscillate in time. The [fix wall/region](#) command offers the most generality, since the region surface is treated as a wall, and the geometry of the region can be a simple primitive volume (e.g. a sphere, or cube, or plane), or a complex volume made from the union and intersection of primitive volumes. [Regions](#) can also specify a volume "interior" or "exterior" to the specified primitive shape or *union* or *intersection*. [Regions](#) can also be "dynamic" meaning they move with constant velocity, oscillate, or rotate.

The only frictional idealized walls currently in LAMMPS are flat or curved surfaces specified by the [fix wall/gran](#) command. At some point we plan to allow regoin surfaces to be used as frictional walls, as well as triangulated surfaces.

6.18 Elastic constants

Elastic constants characterize the stiffness of a material. The formal definition is provided by the linear relation that holds between the stress and strain tensors in the limit of infinitesimal deformation. In tensor notation, this is expressed as $s\_ij = C\_ijkl * e\_kl$, where the repeated indices imply summation. s_ij are the elements of the symmetric stress tensor. e_kl are the elements of the symmetric strain tensor. C_ijkl are the elements of the fourth rank tensor of elastic constants. In three dimensions, this tensor has $3^4=81$ elements. Using Voigt notation, the tensor can be written as a 6x6 matrix, where C_ij is now the derivative of s_i w.r.t. e_j. Because s_i is itself a derivative w.r.t. e_i, it follows that C_ij is also symmetric, with at most $7*6/2 = 21$ distinct elements.

At zero temperature, it is easy to estimate these derivatives by deforming the simulation box in one of the six directions using the [change_box](#) command and measuring the change in the stress tensor. A general-purpose script that does this is given in the examples/elastic directory described in [this section](#).

Calculating elastic constants at finite temperature is more challenging, because it is necessary to run a simulation that perfoms time averages of differential properties. One way to do this is to measure the change in average stress tensor in an NVT simulations when the cell volume undergoes a finite deformation. In order to balance the systematic and statistical errors in this method, the magnitude of the deformation must be chosen judiciously, and care must be taken to fully equilibrate the deformed cell before sampling the stress tensor. Another approach is to sample the triclinic cell fluctuations that occur in an NPT simulation. This method can also be slow to converge and requires careful post-processing (Shinoda)

6.19 Library interface to LAMMPS

As described in Section_start 5, LAMMPS can be built as a library, so that it can be called by another code, used in a coupled manner with other codes, or driven through a Python interface.

All of these methodologies use a C-style interface to LAMMPS that is provided in the files src/library.cpp and src/library.h. The functions therein have a C-style argument list, but contain C++ code you could write yourself in a C++ application that was invoking LAMMPS directly. The C++ code in the functions illustrates how to invoke internal LAMMPS operations. Note that LAMMPS classes are defined within a LAMMPS namespace (LAMMPS_NS) if you use them from another C++ application.

Library.cpp contains these 4 functions:

```
void lammps_open(int, char **, MPI_Comm, void **);
void lammps_close(void *);
void lammps_file(void *, char *);
char *lammps_command(void *, char *);
```

The lammps_open() function is used to initialize LAMMPS, passing in a list of strings as if they were command-line arguments when LAMMPS is run in stand-alone mode from the command line, and a MPI communicator for LAMMPS to run under. It returns a ptr to the LAMMPS object that is created, and which is used in subsequent library calls. The lammps_open() function can be called multiple times, to create multiple instances of LAMMPS.

LAMMPS will run on the set of processors in the communicator. This means the calling code can run LAMMPS on all or a subset of processors. For example, a wrapper script might decide to alternate between LAMMPS and another code, allowing them both to run on all the processors. Or it might allocate half the processors to LAMMPS and half to the other code and run both codes simultaneously before syncing them up periodically. Or it might instantiate multiple instances of LAMMPS to perform different calculations.

The lammps_close() function is used to shut down an instance of LAMMPS and free all its memory.

The lammps_file() and lammps_command() functions are used to pass a file or string to LAMMPS as if it were an input script or single command in an input script. Thus the calling code can read or generate a series of LAMMPS commands one line at a time and pass it thru the library interface to setup a problem and then run it, interleaving the lammps_command() calls with other calls to extract information from LAMMPS, perform its own operations, or call another code's library.

Other useful functions are also included in library.cpp. For example:

```
void *lammps_extract_global(void *, char *)
void *lammps_extract_atom(void *, char *)
void *lammps_extract_compute(void *, char *, int, int)
void *lammps_extract_fix(void *, char *, int, int, int, int)
void *lammps_extract_variable(void *, char *, char *)
int lammps_get_natoms(void *)
void lammps_get_coords(void *, double *)
void lammps_put_coords(void *, double *)
```

These can extract various global or per-atom quantities from LAMMPS as well as values calculated by a compute, fix, or variable. The "get" and "put" operations can retrieve and reset atom coordinates. See the library.cpp file and its associated header file library.h for details.

The key idea of the library interface is that you can write any functions you wish to define how your code talks to LAMMPS and add them to src/library.cpp and src/library.h, as well as to the Python interface. The routines you add can access or change any LAMMPS data you wish. The examples/COUPLE and python directories have example C++ and C and Python codes which show how a driver code can link to LAMMPS as a library, run LAMMPS on a subset of processors, grab data from LAMMPS, change it, and put it back into LAMMPS.

6.20 Calculating thermal conductivity

The thermal conductivity kappa of a material can be measured in at least 4 ways using various options in LAMMPS. See the examples/KAPPA directory for scripts that implement the 4 methods discussed here for a simple Lennard-Jones fluid model. Also, see this section of the manual for an analogous discussion for viscosity.

The thermal conducitivity tensor kappa is a measure of the propensity of a material to transmit heat energy in a diffusive manner as given by Fourier's law

J = -kappa grad(T)

where J is the heat flux in units of energy per area per time and grad(T) is the spatial gradient of temperature. The thermal conductivity thus has units of energy per distance per time per degree K and is often approximated as an isotropic quantity, i.e. as a scalar.

The first method is to setup two thermostatted regions at opposite ends of a simulation box, or one in the middle and one at the end of a periodic box. By holding the two regions at different temperatures with a thermostatting fix, the energy added to the hot region should equal the energy subtracted from the cold region and be proportional to the heat flux moving between the regions. See the paper by Ikeshoji and Hafskjold for details of this idea. Note that thermostatting fixes such as fix nvt, fix langevin, and fix temp/rescale store the cumulative energy they add/subtract.

Alternatively, as a second method, the fix heat command can used in place of thermostats on each of two regions to add/subtract specified amounts of energy to both regions. In both cases, the resulting temperatures of the two regions can be monitored with the "compute temp/region" command and the temperature profile of the intermediate region can be monitored with the fix ave/spatial and compute ke/atom commands.

The third method is to perform a reverse non-equilibrium MD simulation using the fix thermal/conductivity command which implements the rNEMD algorithm of Muller-Plathe. Kinetic energy is swapped between atoms in two different layers of the simulation box. This induces a temperature gradient between the two layers which can be monitored with the fix ave/spatial and compute ke/atom commands. The fix tallies the cumulative energy transfer that it performs. See the fix thermal/conductivity command for details.

The fourth method is based on the Green-Kubo (GK) formula which relates the ensemble average of the auto-correlation of the heat flux to kappa. The heat flux can be calculated from the fluctuations of per-atom potential and kinetic energies and per-atom stress tensor in a steady-state equilibrated simulation. This is in contrast to the two preceding non-equilibrium methods, where energy flows continuously between hot and cold regions of the simulation box.

The compute heat/flux command can calculate the needed heat flux and describes how to implement the Green_Kubo formalism using additional LAMMPS commands, such as the fix ave/correlate command to

calculate the needed auto-correlation. See the doc page for the compute heat/flux command for an example input script that calculates the thermal conductivity of solid Ar via the GK formalism.

6.21 Calculating viscosity

The shear viscosity eta of a fluid can be measured in at least 4 ways using various options in LAMMPS. See the examples/VISCOSITY directory for scripts that implement the 4 methods discussed here for a simple Lennard-Jones fluid model. Also, see this section of the manual for an analogous discussion for thermal conductivity.

Eta is a measure of the propensity of a fluid to transmit momentum in a direction perpendicular to the direction of velocity or momentum flow. Alternatively it is the resistance the fluid has to being sheared. It is given by

J = -eta grad(Vstream)

where J is the momentum flux in units of momentum per area per time. and grad(Vstream) is the spatial gradient of the velocity of the fluid moving in another direction, normal to the area through which the momentum flows. Viscosity thus has units of pressure-time.

The first method is to perform a non-equlibrium MD (NEMD) simulation by shearing the simulation box via the fix deform command, and using the fix nvt/sllod command to thermostat the fluid via the SLLOD equations of motion. Alternatively, as a second method, one or more moving walls can be used to shear the fluid in between them, again with some kind of thermostat that modifies only the thermal (non-shearing) components of velocity to prevent the fluid from heating up.

In both cases, the velocity profile setup in the fluid by this procedure can be monitored by the fix ave/spatial command, which determines grad(Vstream) in the equation above. E.g. the derivative in the y-direction of the Vx component of fluid motion or grad(Vstream) = dVx/dy. The Pxy off-diagonal component of the pressure or stress tensor, as calculated by the compute pressure command, can also be monitored, which is the J term in the equation above. See this section of the manual for details on NEMD simulations.

The third method is to perform a reverse non-equilibrium MD simulation using the fix viscosity command which implements the rNEMD algorithm of Muller-Plathe. Momentum in one dimension is swapped between atoms in two different layers of the simulation box in a different dimension. This induces a velocity gradient which can be monitored with the fix ave/spatial command. The fix tallies the cummulative momentum transfer that it performs. See the fix viscosity command for details.

The fourth method is based on the Green-Kubo (GK) formula which relates the ensemble average of the auto-correlation of the stress/pressure tensor to eta. This can be done in a steady-state equilibrated simulation which is in contrast to the two preceding non-equilibrium methods, where momentum flows continuously through the simulation box.

Here is an example input script that calculates the viscosity of liquid Ar via the GK formalism:

```
# Sample LAMMPS input script for viscosity of liquid Ar

units       real
variable    T equal 86.4956
variable    V equal vol
variable    dt equal 4.0
variable    p equal 400      # correlation length
variable    s equal 5        # sample interval
variable    d equal $p*$s    # dump interval
```

```
# convert from LAMMPS real units to SI


variable    kB equal 1.3806504e-23    # [J/K/ Boltzmann
variable    atm2Pa equal 101325.0
variable    A2m equal 1.0e-10
variable    fs2s equal 1.0e-15
variable    convert equal ${atm2Pa}*${atm2Pa}*${fs2s}*${A2m}*${A2m}*${A2m}


# setup problem


dimension    3
boundary     p p p
lattice      fcc 5.376 orient x 1 0 0 orient y 0 1 0 orient z 0 0 1
region       box block 0 4 0 4 0 4
create_box   1 box
create_atoms 1 box
mass         1 39.948
pair_style   lj/cut 13.0
pair_coeff   * * 0.2381 3.405
timestep     ${dt}
thermo       $d


# equilibration and thermalization


velocity     all create $T 102486 mom yes rot yes dist gaussian
fix          NVT all nvt temp $T $T 10 drag 0.2
run          8000


# viscosity calculation, switch to NVE if desired


#unfix       NVT
#fix         NVE all nve


reset_timestep 0
variable     pxy equal pxy
variable     pxz equal pxz
variable     pyz equal pyz
fix          SS all ave/correlate $s $p $d &
             v_pxy v_pxz v_pyz type auto file S0St.dat ave running
variable     scale equal ${convert}/(${kB}*$T)*$V*$s*${dt}
variable     v11 equal trap(f_SS[3])*${scale}
variable     v22 equal trap(f_SS[4])*${scale}
variable     v33 equal trap(f_SS[5])*${scale}
thermo_style custom step temp press v_pxy v_pxz v_pyz v_v11 v_v22 v_v33
run          100000
variable     v equal (v_v11+v_v22+v_v33)/3.0
variable     ndens equal count(all)/vol
print        "average viscosity: $v [Pa.s/ @ $T K, ${ndens} /A^3"
```

(Berendsen) Berendsen, Grigera, Straatsma, J Phys Chem, 91, 6269-6271 (1987).

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179-5197 (1995).

(Horn) Horn, Swope, Pitera, Madura, Dick, Hura, and Head-Gordon, J Chem Phys, 120, 9665 (2004).

(Ikeshoji) Ikeshoji and Hafskjold, Molecular Physics, 81, 251-261 (1994).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897-8909 (1990).

(Jorgensen) Jorgensen, Chandrasekhar, Madura, Impey, Klein, J Chem Phys, 79, 926 (1983).

(Price) Price and Brooks, J Chem Phys, 121, 10096 (2004).

(Shinoda) Shinoda, Shiga, and Mikami, Phys Rev B, 69, 134103 (2004).

1. Introduction

This section provides an overview of what LAMMPS can and can't do, describes what it means for LAMMPS to be an open-source code, and acknowledges the funding and people who have contributed to LAMMPS over the years.

1.1 What is LAMMPS
1.2 LAMMPS features
1.3 LAMMPS non-features
1.4 Open source distribution
1.5 Acknowledgments and citations

1.1 What is LAMMPS

LAMMPS is a classical molecular dynamics code that models an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and coarse-grained systems using a variety of force fields and boundary conditions.

For examples of LAMMPS simulations, see the Publications page of the LAMMPS WWW Site.

LAMMPS runs efficiently on single-processor desktop or laptop machines, but is designed for parallel computers. It will run on any parallel machine that compiles C++ and supports the MPI message-passing library. This includes distributed- or shared-memory parallel machines and Beowulf-style clusters.

LAMMPS can model systems with only a few particles up to millions or billions. See Section_perf for information on LAMMPS performance and scalability, or the Benchmarks section of the LAMMPS WWW Site.

LAMMPS is a freely-available open-source code, distributed under the terms of the GNU Public License, which means you can use or modify the code however you wish. See this section for a brief discussion of the open-source philosophy.

LAMMPS is designed to be easy to modify or extend with new capabilities, such as new force fields, atom types, boundary conditions, or diagnostics. See Section_modify for more details.

The current version of LAMMPS is written in C++. Earlier versions were written in F77 and F90. See Section_history for more information on different versions. All versions can be downloaded from the LAMMPS WWW Site.

LAMMPS was originally developed under a US Department of Energy CRADA (Cooperative Research and Development Agreement) between two DOE labs and 3 companies. It is distributed by Sandia National Labs. See this section for more information on LAMMPS funding and individuals who have contributed to LAMMPS.

In the most general sense, LAMMPS integrates Newton's equations of motion for collections of atoms, molecules, or macroscopic particles that interact via short- or long-range forces with a variety of initial and/or boundary conditions. For computational efficiency LAMMPS uses neighbor lists to keep track of nearby particles. The lists are optimized for systems with particles that are repulsive at short distances, so that the local density of particles never becomes too large. On parallel machines, LAMMPS uses spatial-decomposition techniques to partition the simulation domain into small 3d sub-domains, one of which

is assigned to each processor. Processors communicate and store "ghost" atom information for atoms that border their sub-domain. LAMMPS is most efficient (in a parallel sense) for systems whose particles fill a 3d rectangular box with roughly uniform density. Papers with technical details of the algorithms used in LAMMPS are listed in [this section](#).

1.2 LAMMPS features

This section highlights LAMMPS features, with pointers to specific commands which give more details. If LAMMPS doesn't have your favorite interatomic potential, boundary condition, or atom type, see [Section_modify](#), which describes how you can add it to LAMMPS.

General features

- runs on a single processor or in parallel
- distributed-memory message-passing parallelism (MPI)
- spatial-decomposition of simulation domain for parallelism
- open-source distribution
- highly portable C++
- optional libraries used: MPI and single-processor FFT
- GPU (CUDA and OpenCL) and OpenMP support for many code features
- easy to extend with new features and functionality
- runs from an input script
- syntax for defining and using variables and formulas
- syntax for looping over runs and breaking out of loops
- run one or multiple simulations simultaneously (in parallel) from one script
- build as library, invoke LAMMPS thru library interface or provided Python wrapper
- couple with other codes: LAMMPS calls other code, other code calls LAMMPS, umbrella code calls both

Particle and model types

([atom style](#) command)

- atoms
- coarse-grained particles (e.g. bead-spring polymers)
- united-atom polymers or organic molecules
- all-atom polymers, organic molecules, proteins, DNA
- metals
- granular materials
- coarse-grained mesoscale models
- finite-size spherical and ellipsoidal particles
- finite-size line segment (2d) and triangle (3d) particles
- point dipole particles
- rigid collections of particles
- hybrid combinations of these

Force fields

([pair style](#), [bond style](#), [angle style](#), [dihedral style](#), [improper style](#), [kspace style](#) commands)

- pairwise potentials: Lennard-Jones, Buckingham, Morse, Born-Mayer-Huggins, Yukawa, soft, class 2 (COMPASS), hydrogen bond, tabulated
- charged pairwise potentials: Coulombic, point-dipole

- manybody potentials: EAM, Finnis/Sinclair EAM, modified EAM (MEAM), embedded ion method (EIM), EDIP, ADP, Stillinger-Weber, Tersoff, REBO, AIREBO, ReaxFF, COMB
- electron force field (eFF, AWPMD)
- coarse-grained potentials: DPD, GayBerne, REsquared, colloidal, DLVO
- mesoscopic potentials: granular, Peridynamics, SPH
- bond potentials: harmonic, FENE, Morse, nonlinear, class 2, quartic (breakable)
- angle potentials: harmonic, CHARMM, cosine, cosine/squared, cosine/periodic, class 2 (COMPASS)
- dihedral potentials: harmonic, CHARMM, multi-harmonic, helix, class 2 (COMPASS), OPLS
- improper potentials: harmonic, cvff, umbrella, class 2 (COMPASS)
- polymer potentials: all-atom, united-atom, bead-spring, breakable
- water potentials: TIP3P, TIP4P, SPC
- implicit solvent potentials: hydrodynamic lubrication, Debye
- [KIM archive](#) of potentials
- long-range interactions for charge, point-dipoles, and LJ dispersion: Ewald, Wolf, PPPM (similar to particle-mesh Ewald)
- force-field compatibility with common CHARMM, AMBER, DREIDING, OPLS, GROMACS, COMPASS options
- handful of GPU-enabled pair styles
- hybrid potentials: multiple pair, bond, angle, dihedral, improper potentials can be used in one simulation
- overlaid potentials: superposition of multiple pair potentials

Atom creation

([read_data](#), [lattice](#), [create_atoms](#), [delete_atoms](#), [displace_atoms](#), [replicate](#) commands)

- read in atom coords from files
- create atoms on one or more lattices (e.g. grain boundaries)
- delete geometric or logical groups of atoms (e.g. voids)
- replicate existing atoms multiple times
- displace atoms

Ensembles, constraints, and boundary conditions

([fix](#) command)

- 2d or 3d systems
- orthogonal or non-orthogonal (triclinic symmetry) simulation domains
- constant NVE, NVT, NPT, NPH, Parinello/Rahman integrators
- thermostatting options for groups and geometric regions of atoms
- pressure control via Nose/Hoover or Berendsen barostatting in 1 to 3 dimensions
- simulation box deformation (tensile and shear)
- harmonic (umbrella) constraint forces
- rigid body constraints
- SHAKE bond and angle constraints
- bond breaking, formation, swapping
- walls of various kinds
- non-equilibrium molecular dynamics (NEMD)
- variety of additional boundary conditions and constraints

Integrators

([run](#), [run_style](#), [minimize](#) commands)

- velocity-Verlet integrator
- Brownian dynamics
- rigid body integration
- energy minimization via conjugate gradient or steepest descent relaxation
- rRESPA hierarchical timestepping
- rerun command for post-processing of dump files

Diagnostics

- see the various flavors of the [fix](#) and [compute](#) commands

Output

([dump](#), [restart](#) commands)

- log file of thermodynamic info
- text dump files of atom coords, velocities, other per-atom quantities
- binary restart files
- parallel I/O of dump and restart files
- per-atom quantities (energy, stress, centro-symmetry parameter, CNA, etc)
- user-defined system-wide (log file) or per-atom (dump file) calculations
- spatial and time averaging of per-atom quantities
- time averaging of system-wide quantities
- atom snapshots in native, XYZ, XTC, DCD, CFG formats

Multi-replica models

[nudged elastic band](#) [parallel replica dynamics](#) [temperature accelerated dynamics](#) [parallel tempering](#)

Pre- and post-processing

- Various pre- and post-processing serial tools are packaged with LAMMPS; see these [doc pages](#).
- Our group has also written and released a separate toolkit called [Pizza.py](#) which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in [Python](#) and is available for download from [the Pizza.py WWW site](#).

Specialized features

These are LAMMPS capabilities which you may not think of as typical molecular dynamics options:

- [generalized aspherical particles](#)
- [stochastic rotation dynamics (SRD)](#)
- [real-time visualization and interactive MD](#)
- [atom-to-continuum coupling](#) with finite elements
- coupled rigid body integration via the [POEMS](#) library
- [grand canonical Monte Carlo](#) insertions/deletions
- [Direct Simulation Monte Carlo](#) for low-density fluids
- [Peridynamics mesoscale modeling](#)
- [targeted](#) and [steered](#) molecular dynamics
- [two-temperature electron model](#)

1.3 LAMMPS non-features

LAMMPS is designed to efficiently compute Newton's equations of motion for a system of interacting particles. Many of the tools needed to pre- and post-process the data for such simulations are not included in the LAMMPS kernel for several reasons:

- the desire to keep LAMMPS simple
- they are not parallel operations
- other codes already do them
- limited development resources

Specifically, LAMMPS itself does not:

- run thru a GUI
- build molecular systems
- assign force-field coefficients automagically
- perform sophisticated analyses of your MD simulation
- visualize your MD simulation
- plot your output data

A few tools for pre- and post-processing tasks are provided as part of the LAMMPS package; they are described in this section. However, many people use other codes or write their own tools for these tasks.

As noted above, our group has also written and released a separate toolkit called Pizza.py which addresses some of the listed bullets. It provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

LAMMPS requires as input a list of initial atom coordinates and types, molecular topology information, and force-field coefficients assigned to all atoms and bonds. LAMMPS will not build molecular systems and assign force-field parameters for you.

For atomic systems LAMMPS provides a create_atoms command which places atoms on solid-state lattices (fcc, bcc, user-defined, etc). Assigning small numbers of force field coefficients can be done via the pair coeff, bond coeff, angle coeff, etc commands. For molecular systems or more complicated simulation geometries, users typically use another code as a builder and convert its output to LAMMPS input format, or write their own code to generate atom coordinate and molecular topology for LAMMPS to read in.

For complicated molecular systems (e.g. a protein), a multitude of topology information and hundreds of force-field coefficients must typically be specified. We suggest you use a program like CHARMM or AMBER or other molecular builders to setup such problems and dump its information to a file. You can then reformat the file as LAMMPS input. Some of the tools in this section can assist in this process.

Similarly, LAMMPS creates output files in a simple format. Most users post-process these files with their own analysis tools or re-format them for input into other programs, including visualization packages. If you are convinced you need to compute something on-the-fly as LAMMPS runs, see Section_modify for a discussion of how you can use the dump and compute and fix commands to print out data of your choosing. Keep in mind that complicated computations can slow down the molecular dynamics timestepping, particularly if the computations are not parallel, so it is often better to leave such analysis to post-processing codes.

A very simple (yet fast) visualizer is provided with the LAMMPS package - see the xmovie tool in this section. It creates xyz projection views of atomic coordinates and animates them. We find it very useful for debugging purposes. For high-quality visualization we recommend the following packages:

- VMD
- AtomEye
- PyMol
- Raster3d
- RasMol

Other features that LAMMPS does not yet (and may never) support are discussed in Section_history.

Finally, these are freely-available molecular dynamics codes, most of them parallel, which may be well-suited to the problems you want to model. They can also be used in conjunction with LAMMPS to perform complementary modeling tasks.

- CHARMM
- AMBER
- NAMD
- NWCHEM
- DL_POLY
- Tinker

CHARMM, AMBER, NAMD, NWCHEM, and Tinker are designed primarily for modeling biological molecules. CHARMM and AMBER use atom-decomposition (replicated-data) strategies for parallelism; NAMD and NWCHEM use spatial-decomposition approaches, similar to LAMMPS. Tinker is a serial code. DL_POLY includes potentials for a variety of biological and non-biological materials; both a replicated-data and spatial-decomposition version exist.

1.4 Open source distribution

LAMMPS comes with no warranty of any kind. As each source file states in its header, it is a copyrighted code that is distributed free-of- charge, under the terms of the GNU Public License (GPL). This is often referred to as open-source distribution - see www.gnu.org or www.opensource.org for more details. The legal text of the GPL is in the LICENSE file that is included in the LAMMPS distribution.

Here is a summary of what the GPL means for LAMMPS users:

(1) Anyone is free to use, modify, or extend LAMMPS in any way they choose, including for commercial purposes.

(2) If you distribute a modified version of LAMMPS, it must remain open-source, meaning you distribute it under the terms of the GPL. You should clearly annotate such a code as a derivative version of LAMMPS.

(3) If you release any code that includes LAMMPS source code, then it must also be open-sourced, meaning you distribute it under the terms of the GPL.

(4) If you give LAMMPS files to someone else, the GPL LICENSE file and source file headers (including the copyright and GPL notices) should remain part of the code.

In the spirit of an open-source code, these are various ways you can contribute to making LAMMPS better. You can send email to the developers on any of these items.

- Point prospective users to the LAMMPS WWW Site. Mention it in talks or link to it from your WWW site.
- If you find an error or omission in this manual or on the LAMMPS WWW Site, or have a suggestion for something to clarify or include, send an email to the developers.

- If you find a bug, Section_errors 2 describes how to report it.
- If you publish a paper using LAMMPS results, send the citation (and any cool pictures or movies if you like) to add to the Publications, Pictures, and Movies pages of the LAMMPS WWW Site, with links and attributions back to you.
- Create a new Makefile.machine that can be added to the src/MAKE directory.
- The tools sub-directory of the LAMMPS distribution has various stand-alone codes for pre- and post-processing of LAMMPS data. More details are given in Section_tools. If you write a new tool that users will find useful, it can be added to the LAMMPS distribution.
- LAMMPS is designed to be easy to extend with new code for features like potentials, boundary conditions, diagnostic computations, etc. This section gives details. If you add a feature of general interest, it can be added to the LAMMPS distribution.
- The Benchmark page of the LAMMPS WWW Site lists LAMMPS performance on various platforms. The files needed to run the benchmarks are part of the LAMMPS distribution. If your machine is sufficiently different from those listed, your timing data can be added to the page.
- You can send feedback for the User Comments page of the LAMMPS WWW Site. It might be added to the page. No promises.
- Cash. Small denominations, unmarked bills preferred. Paper sack OK. Leave on desk. VISA also accepted. Chocolate chip cookies encouraged.

1.5 Acknowledgments and citations

LAMMPS development has been funded by the US Department of Energy (DOE), through its CRADA, LDRD, ASCI, and Genomes-to-Life programs and its OASCR and OBER offices.

Specifically, work on the latest version was funded in part by the US Department of Energy's Genomics:GTL program (www.doegenomestolife.org) under the project, "Carbon Sequestration in Synechococcus Sp.: From Molecular Machines to Hierarchical Modeling".

The following paper describe the basic parallel algorithms used in LAMMPS. If you use LAMMPS results in your published work, please cite this paper and include a pointer to the LAMMPS WWW Site (http://lammps.sandia.gov):

S. J. Plimpton, **Fast Parallel Algorithms for Short-Range Molecular Dynamics**, J Comp Phys, 117, 1-19 (1995).

Other papers describing specific algorithms used in LAMMPS are listed under the Citing LAMMPS link of the LAMMPS WWW page.

The Publications link on the LAMMPS WWW page lists papers that have cited LAMMPS. If your paper is not listed there for some reason, feel free to send us the info. If the simulations in your paper produced cool pictures or animations, we'll be pleased to add them to the Pictures or Movies pages of the LAMMPS WWW site.

The core group of LAMMPS developers is at Sandia National Labs:

- Steve Plimpton, sjplimp at sandia.gov
- Aidan Thompson, athomps at sandia.gov
- Paul Crozier, pscrozi at sandia.gov

The following folks are responsible for significant contributions to the code, or other aspects of the LAMMPS development effort. Many of the packages they have written are somewhat unique to LAMMPS and the code would not be as general-purpose as it is without their expertise and efforts.

- Axel Kohlmeyer (Temple U), akohlmey at gmail.com, SVN and Git repositories, indefatigable mail

list responder, USER-CG-CMM and USER-OMP packages
- Roy Pollock (LLNL), Ewald and PPPM solvers
- Mike Brown (ORNL), brownw at ornl.gov, GPU package
- Greg Wagner (Sandia), gjwagne at sandia.gov, MEAM package for MEAM potential
- Mike Parks (Sandia), mlparks at sandia.gov, PERI package for Peridynamics
- Rudra Mukherjee (JPL), Rudranarayan.M.Mukherjee at jpl.nasa.gov, POEMS package for articulated rigid body motion
- Reese Jones (Sandia) and collaborators, rjones at sandia.gov, USER-ATC package for atom/continuum coupling
- Ilya Valuev (JIHT), valuev at physik.hu-berlin.de, USER-AWPMD package for wave-packet MD
- Christian Trott (U Tech Ilmenau), christian.trott at tu-ilmenau.de, USER-CUDA package
- Andres Jaramillo-Botero (Caltech), ajaramil at wag.caltech.edu, USER-EFF package for electron force field
- Christoph Kloss (JKU), Christoph.Kloss at jku.at, USER-LIGGGHTS package for granular models and granular/fluid coupling
- Metin Aktulga (LBL), hmaktulga at lbl.gov, USER-REAXC package for C version of ReaxFF
- Georg Gunzenmuller (EMI), georg.ganzenmueller at emi.fhg.de, USER-SPH package

As discussed in Section_history, LAMMPS originated as a cooperative project between DOE labs and industrial partners. Folks involved in the design and testing of the original version of LAMMPS were the following:

- John Carpenter (Mayo Clinic, formerly at Cray Research)
- Terry Stouch (Lexicon Pharmaceuticals, formerly at Bristol Myers Squibb)
- Steve Lustig (Dupont)
- Jim Belak (LLNL)

10. Modifying & extending LAMMPS

This section describes how to customize LAMMPS by modifying and extending its source code.

LAMMPS is designed in a modular fashion so as to be easy to modify and extend with new functionality. In fact, about 75% of its source code is files added in this fashion.

In this section, changes and additions users can make are listed along with minimal instructions. If you add a new feature to LAMMPS and think it will be of interest to general users, we encourage you to submit it to the developers for inclusion in the released version of LAMMPS. Information about how to do this is provided below.

The best way to add a new feature is to find a similar feature in LAMMPS and look at the corresponding source and header files to figure out what it does. You will need some knowledge of C++ to be able to understand the hi-level structure of LAMMPS and its class organization, but functions (class methods) that do actual computations are written in vanilla C-style code and operate on simple C-style data structures (vectors and arrays).

Most of the new features described in this section require you to write a new C++ derived class (except for exceptions described below, where you can make small edits to existing files). Creating a new class requires 2 files, a source code file (*.cpp) and a header file (*.h). The derived class must provide certain methods to work as a new option. Depending on how different your new feature is compared to existing features, you can either derive from the base class itself, or from a derived class that already exists. Enabling LAMMPS to invoke the new class is as simple as putting the two source files in the src dir and re-building LAMMPS.

The advantage of C++ and its object-orientation is that all the code and variables needed to define the new feature are in the 2 files you write, and thus shouldn't make the rest of LAMMPS more complex or cause side-effect bugs.

Here is a concrete example. Suppose you write 2 files pair_foo.cpp and pair_foo.h that define a new class PairFoo that computes pairwise potentials described in the classic 1997 paper by Foo, et al. If you wish to invoke those potentials in a LAMMPS input script with a command like

```
pair_style foo 0.1 3.5
```

then your pair_foo.h file should be structured as follows:

```
#ifdef PAIR_CLASS
PairStyle(foo,PairFoo)
#else
...
(class definition for PairFoo)
...
#endif
```

where "foo" is the style keyword in the pair_style command, and PairFoo is the class name defined in your pair_foo.cpp and pair_foo.h files.

When you re-build LAMMPS, your new pairwise potential becomes part of the executable and can be invoked with a pair_style command like the example above. Arguments like 0.1 and 3.5 can be defined and processed by your new class.

As illustrated by this pairwise example, many kinds of options are referred to in the LAMMPS documentation as the "style" of a particular command.

The instructions below give the header file for the base class that these styles are derived from. Public variables in that file are ones used and set by the derived classes which are also used by the base class. Sometimes they are also used by the rest of LAMMPS. Virtual functions in the base class header file which are set = 0 are ones you must define in your new derived class to give it the functionality LAMMPS expects. Virtual functions that are not set to 0 are functions you can optionally define.

Additionally, new output options can be added directly to the thermo.cpp, dump_custom.cpp, and variable.cpp files as explained below.

Here are additional guidelines for modifying LAMMPS and adding new functionality:

- Think about whether what you want to do would be better as a pre- or post-processing step. Many computations are more easily and more quickly done that way.
- Don't do anything within the timestepping of a run that isn't parallel. E.g. don't accumulate a bunch of data on a single processor and analyze it. You run the risk of seriously degrading the parallel efficiency.
- If your new feature reads arguments or writes output, make sure you follow the unit conventions discussed by the **units** command.
- If you add something you think is truly useful and doesn't impact LAMMPS performance when it isn't used, send an email to the **developers**. We might be interested in adding it to the LAMMPS distribution. See further details on this at the bottom of this page.

10.1 Atom styles

Classes that define an **atom style** are derived from the AtomVec class and managed by the Atom class. The atom style determines what attributes are associated with an atom. A new atom style can be created if one of the existing atom styles does not define all the attributes you need to store and communicate with atoms.

Atom_vec_atomic.cpp is a simple example of an atom style.

Here is a brief description of methods you define in your new derived class. See atom_vec.h for details.

init	one time setup (optional)

grow	re-allocate atom arrays to longer lengths (required)
grow_reset	make array pointers in Atom and AtomVec classes consistent (required)
copy	copy info for one atom to another atom's array locations (required)
pack_comm	store an atom's info in a buffer communicated every timestep (required)
pack_comm_vel	add velocity info to communication buffer (required)
pack_comm_hybrid	store extra info unique to this atom style (optional)
unpack_comm	retrieve an atom's info from the buffer (required)
unpack_comm_vel	also retrieve velocity info (required)
unpack_comm_hybrid	retreive extra info unique to this atom style (optional)
pack_reverse	store an atom's info in a buffer communicating partial forces (required)
pack_reverse_hybrid	store extra info unique to this atom style (optional)
unpack_reverse	retrieve an atom's info from the buffer (required)
unpack_reverse_hybrid	retreive extra info unique to this atom style (optional)
pack_border	store an atom's info in a buffer communicated on neighbor re-builds (required)
pack_border_vel	add velocity info to buffer (required)
pack_border_hybrid	store extra info unique to this atom style (optional)
unpack_border	retrieve an atom's info from the buffer (required)
unpack_border_vel	also retrieve velocity info (required)
unpack_border_hybrid	retreive extra info unique to this atom style (optional)
pack_exchange	store all an atom's info to migrate to another processor (required)
unpack_exchange	retrieve an atom's info from the buffer (required)
size_restart	number of restart quantities associated with proc's atoms (required)
pack_restart	pack atom quantities into a buffer (required)
unpack_restart	unpack atom quantities from a buffer (required)
create_atom	create an individual atom of this style (required)
data_atom	parse an atom line from the data file (required)
data_atom_hybrid	parse additional atom info unique to this atom style (optional)
data_vel	parse one line of velocity information from data file (optional)
data_vel_hybrid	parse additional velocity data unique to this atom style (optional)
memory_usage	tally memory allocated by atom arrays (required)

The constructor of the derived class sets values for several variables that you must set when defining a new atom style, which are documented in atom_vec.h. New atom arrays are defined in atom.cpp. Search for the word "customize" and you will find locations you will need to modify.

IMPORTANT NOTE: It is possible to add some attributes, such as a molecule ID, to atom styles that do not have them via the fix property/atom command. This command also allows new custom attributes consisting of extra integer or floating-point values to be added to atoms. See the fix property/atom doc page for examples of cases where this is useful and details on how to initialize, access, and output the custom values.

New pair styles, fixes, or computes can be added to LAMMPS, as discussed below. The code for these classes can use the per-atom properties defined by fix property/atom. The Atom class has a find_custom() method that is useful in this context:

```
int index = atom->find_custom(char *name, int &flag);
```

The "name" of a custom attribute, as specified in the fix property/atom command, is checked to verify that it exists and its index is returned. The method also sets flag = 0/1 depending on whether it is an integer or floating-point attribute. The vector of values associated with the attribute can then be accessed using the

returned index as

```
int *ivector = atom->ivector[index];
double *dvector = atom->dvector[index];
```

Ivector or dvector are vectors of length Nlocal = # of owned atoms, which store the attributes of individual atoms.

10.2 Bond, angle, dihedral, improper potentials

Classes that compute molecular interactions are derived from the Bond, Angle, Dihedral, and Improper classes. New styles can be created to add new potentials to LAMMPS.

Bond_harmonic.cpp is the simplest example of a bond style. Ditto for the harmonic forms of the angle, dihedral, and improper style commands.

Here is a brief description of common methods you define in your new derived class. See bond.h, angle.h, dihedral.h, and improper.h for details and specific additional methods.

init	check if all coefficients are set, calls *init_style* (optional)
init_style	check if style specific conditions are met (optional)
compute	compute the molecular interactions (required)
settings	apply global settings for all types (optional)
coeff	set coefficients for one type (required)
equilibrium_distance	length of bond, used by SHAKE (required, bond only)
equilibrium_angle	opening of angle, used by SHAKE (required, angle only)
write & read_restart	writes/reads coeffs to restart files (required)
single	force and energy of a single bond or angle (required, bond or angle only)
memory_usage	tally memory allocated by the style (optional)

10.3 Compute styles

Classes that compute scalar and vector quantities like temperature and the pressure tensor, as well as classes that compute per-atom quantities like kinetic energy and the centro-symmetry parameter are derived from the Compute class. New styles can be created to add new calculations to LAMMPS.

Compute_temp.cpp is a simple example of computing a scalar temperature. Compute_ke_atom.cpp is a simple example of computing per-atom kinetic energy.

Here is a brief description of methods you define in your new derived class. See compute.h for details.

init	perform one time setup (required)
init_list	neighbor list setup, if needed (optional)
compute_scalar	compute a scalar quantity (optional)
compute_vector	compute a vector of quantities (optional)
compute_peratom	compute one or more quantities per atom (optional)
compute_local	compute one or more quantities per processor (optional)
pack_comm	pack a buffer with items to communicate (optional)
unpack_comm	unpack the buffer (optional)

pack_reverse	pack a buffer with items to reverse communicate (optional)
unpack_reverse	unpack the buffer (optional)
remove_bias	remove velocity bias from one atom (optional)
remove_bias_all	remove velocity bias from all atoms in group (optional)
restore_bias	restore velocity bias for one atom after remove_bias (optional)
restore_bias_all	same as before, but for all atoms in group (optional)
memory_usage	tally memory usage (optional)

10.4 Dump styles

10.5 Dump custom output options

Classes that dump per-atom info to files are derived from the Dump class. To dump new quantities or in a new format, a new derived dump class can be added, but it is typically simpler to modify the DumpCustom class contained in the dump_custom.cpp file.

Dump_atom.cpp is a simple example of a derived dump class.

Here is a brief description of methods you define in your new derived class. See dump.h for details.

write_header	write the header section of a snapshot of atoms
count	count the number of lines a processor will output
pack	pack a proc's output data into a buffer
write_data	write a proc's data to a file

See the dump command and its *custom* style for a list of keywords for atom information that can already be dumped by DumpCustom. It includes options to dump per-atom info from Compute classes, so adding a new derived Compute class is one way to calculate new quantities to dump.

Alternatively, you can add new keywords to the dump custom command. Search for the word "customize" in dump_custom.cpp to see the half-dozen or so locations where code will need to be added.

10.6 Fix styles

In LAMMPS, a "fix" is any operation that is computed during timestepping that alters some property of the system. Essentially everything that happens during a simulation besides force computation, neighbor list construction, and output, is a "fix". This includes time integration (update of coordinates and velocities), force constraints or boundary conditions (SHAKE or walls), and diagnostics (compute a diffusion coefficient). New styles can be created to add new options to LAMMPS.

Fix_setforce.cpp is a simple example of setting forces on atoms to prescribed values. There are dozens of fix options already in LAMMPS; choose one as a template that is similar to what you want to implement.

Here is a brief description of methods you can define in your new derived class. See fix.h for details.

setmask	determines when the fix is called during the timestep (required)
init	initialization before a run (optional)
setup_pre_exchange	called before atom exchange in setup (optional)
setup_pre_force	called before force computation in setup (optional)

setup	called immediately before the 1st timestep and after forces are computed (optional)
min_setup_pre_force	like setup_pre_force, but for minimizations instead of MD runs (optional)
min_setup	like setup, but for minimizations instead of MD runs (optional)
initial_integrate	called at very beginning of each timestep (optional)
pre_exchange	called before atom exchange on re-neighboring steps (optional)
pre_neighbor	called before neighbor list build (optional)
pre_force	called after pair & molecular forces are computed (optional)
post_force	called after pair & molecular forces are computed and communicated (optional)
final_integrate	called at end of each timestep (optional)
end_of_step	called at very end of timestep (optional)
write_restart	dumps fix info to restart file (optional)
restart	uses info from restart file to re-initialize the fix (optional)
grow_arrays	allocate memory for atom-based arrays used by fix (optional)
copy_arrays	copy atom info when an atom migrates to a new processor (optional)
pack_exchange	store atom's data in a buffer (optional)
unpack_exchange	retrieve atom's data from a buffer (optional)
pack_restart	store atom's data for writing to restart file (optional)
unpack_restart	retrieve atom's data from a restart file buffer (optional)
size_restart	size of atom's data (optional)
maxsize_restart	max size of atom's data (optional)
setup_pre_force_respa	same as setup_pre_force, but for rRESPA (optional)
initial_integrate_respa	same as initial_integrate, but for rRESPA (optional)
post_integrate_respa	called after the first half integration step is done in rRESPA (optional)
pre_force_respa	same as pre_force, but for rRESPA (optional)
post_force_respa	same as post_force, but for rRESPA (optional)
final_integrate_respa	same as final_integrate, but for rRESPA (optional)
min_pre_force	called after pair & molecular forces are computed in minimizer (optional)
min_post_force	called after pair & molecular forces are computed and communicated in minmizer (optional)
min_store	store extra data for linesearch based minimization on a LIFO stack (optional)
min_pushstore	push the minimization LIFO stack one element down (optional)
min_popstore	pop the minimization LIFO stack one element up (optional)
min_clearstore	clear minimization LIFO stack (optional)
min_step	reset or move forward on line search minimization (optional)
min_dof	report number of degrees of freedom *added* by this fix in minimization (optional)
max_alpha	report maximum allowed step size during linesearch minimization (optional)
pack_comm	pack a buffer to communicate a per-atom quantity (optional)
unpack_comm	unpack a buffer to communicate a per-atom quantity (optional)
pack_reverse_comm	pack a buffer to reverse communicate a per-atom quantity (optional)
unpack_reverse_comm	unpack a buffer to reverse communicate a per-atom quantity (optional)
dof	report number of degrees of freedom *removed* by this fix during MD (optional)
compute_scalar	return a global scalar property that the fix computes (optional)
compute_vector	return a component of a vector property that the fix computes (optional)
compute_array	return a component of an array property that the fix computes (optional)
deform	called when the box size is changed (optional)

reset_target	called when a change of the target temperature is requested during a run (optional)
reset_dt	is called when a change of the time step is requested during a run (optional)
modify_param	called when a fix_modify request is executed (optional)
memory_usage	report memory used by fix (optional)
thermo	compute quantities for thermodynamic output (optional)

Typically, only a small fraction of these methods are defined for a particular fix. Setmask is mandatory, as it determines when the fix will be invoked during the timestep. Fixes that perform time integration (*nve*, *nvt*, *npt*) implement initial_integrate() and final_integrate() to perform velocity Verlet updates. Fixes that constrain forces implement post_force().

Fixes that perform diagnostics typically implement end_of_step(). For an end_of_step fix, one of your fix arguments must be the variable "nevery" which is used to determine when to call the fix and you must set this variable in the constructor of your fix. By convention, this is the first argument the fix defines (after the ID, group-ID, style).

If the fix needs to store information for each atom that persists from timestep to timestep, it can manage that memory and migrate the info with the atoms as they move from processors to processor by implementing the grow_arrays, copy_arrays, pack_exchange, and unpack_exchange methods. Similarly, the pack_restart and unpack_restart methods can be implemented to store information about the fix in restart files. If you wish an integrator or force constraint fix to work with rRESPA (see the run_style command), the initial_integrate, post_force_integrate, and final_integrate_respa methods can be implemented. The thermo method enables a fix to contribute values to thermodynamic output, as printed quantities and/or to be summed to the potential energy of the system.

10.7 Input script commands

New commands can be added to LAMMPS input scripts by adding new classes that have a "command" method. For example, the create_atoms, read_data, velocity, and run commands are all implemented in this fashion. When such a command is encountered in the LAMMPS input script, LAMMPS simply creates a class with the corresponding name, invokes the "command" method of the class, and passes it the arguments from the input script. The command method can perform whatever operations it wishes on LAMMPS data structures.

The single method your new class must define is as follows:

command	operations performed by the new command

Of course, the new class can define other methods and variables as needed.

10.8 Kspace computations

Classes that compute long-range Coulombic interactions via K-space representations (Ewald, PPPM) are derived from the KSpace class. New styles can be created to add new K-space options to LAMMPS.

Ewald.cpp is an example of computing K-space interactions.

Here is a brief description of methods you define in your new derived class. See kspace.h for details.

init	initialize the calculation before a run
setup	computation before the 1st timestep of a run
compute	every-timestep computation

| memory_usage | tally of memory usage |

10.9 Minimization styles

Classes that perform energy minimization derived from the Min class. New styles can be created to add new minimization algorithms to LAMMPS.

Min_cg.cpp is an example of conjugate gradient minimization.

Here is a brief description of methods you define in your new derived class. See min.h for details.

init	initialize the minimization before a run
run	perform the minimization
memory_usage	tally of memory usage

10.10 Pairwise potentials

Classes that compute pairwise interactions are derived from the Pair class. In LAMMPS, pairwise calculation include manybody potentials such as EAM or Tersoff where particles interact without a static bond topology. New styles can be created to add new pair potentials to LAMMPS.

Pair_lj_cut.cpp is a simple example of a Pair class, though it includes some optional methods to enable its use with rRESPA.

Here is a brief description of the class methods in pair.h:

compute	workhorse routine that computes pairwise interactions
settings	reads the input script line with arguments you define
coeff	set coefficients for one i,j type pair
init_one	perform initialization for one i,j type pair
init_style	initialization specific to this pair style
write & read_restart	write/read i,j pair coeffs to restart files
write & read_restart_settings	write/read global settings to restart files
single	force and energy of a single pairwise interaction between 2 atoms
compute_inner/middle/outer	versions of compute used by rRESPA

The inner/middle/outer routines are optional.

10.11 Region styles

Classes that define geometric regions are derived from the Region class. Regions are used elsewhere in LAMMPS to group atoms, delete atoms to create a void, insert atoms in a specified region, etc. New styles can be created to add new region shapes to LAMMPS.

Region_sphere.cpp is an example of a spherical region.

Here is a brief description of methods you define in your new derived class. See region.h for details.

| match | determine whether a point is in the region |

10.11 Body styles

Classes that define body particles are derived from the Body class. Body particles can represent complex entities, such as surface meshes of discrete points, collections of sub-particles, deformable objects, etc.

See [Section_howto 14](#) of the manual for an overview of using body particles and the [body](#) doc page for details on the various body styles LAMMPS supports. New styles can be created to add new kinds of body particles to LAMMPS.

Body_nparticle.cpp is an example of a body particle that is treated as a rigid body containing N sub-particles.

Here is a brief description of methods you define in your new derived class. See body.h for details.

data_body	process a line from the Bodies section of a data file
noutrow	number of sub-particles output is generated for
noutcol	number of values per-sub-particle output is generated for
output	output values for the Mth sub-particle
pack_comm_body	body attributes to communicate every timestep
unpack_comm_body	unpacking of those attributes
pack_border_body	body attributes to communicate when reneighboring is done
unpack_border_body	unpacking of those attributes

10.13 Thermodynamic output options

There is one class that computes and prints thermodynamic information to the screen and log file; see the file thermo.cpp.

There are two styles defined in thermo.cpp: "one" and "multi". There is also a flexible "custom" style which allows the user to explicitly list keywords for quantities to print when thermodynamic info is output. See the [thermo_style](#) command for a list of defined quantities.

The thermo styles (one, multi, etc) are simply lists of keywords. Adding a new style thus only requires defining a new list of keywords. Search for the word "customize" with references to "thermo style" in thermo.cpp to see the two locations where code will need to be added.

New keywords can also be added to thermo.cpp to compute new quantities for output. Search for the word "customize" with references to "keyword" in thermo.cpp to see the several locations where code will need to be added.

Note that the [thermo_style custom](#) command already allows for thermo output of quantities calculated by [fixes](#), [computes](#), and [variables](#). Thus, it may be simpler to compute what you wish via one of those constructs, than by adding a new keyword to the thermo command.

10.14 Variable options

There is one class that computes and stores [variable](#) information in LAMMPS; see the file variable.cpp. The value associated with a variable can be periodically printed to the screen via the [print](#), [fix print](#), or [thermo_style custom](#) commands. Variables of style "equal" can compute complex equations that involve the

following types of arguments:

thermo keywords = ke, vol, atoms, ... other variables = v_a, v_myvar, ... math functions = div(x,y), mult(x,y), add(x,y), ... group functions = mass(group), xcm(group,x), ... atom values = x**123**, y**3**, vx**34**, ... compute values = c_mytemp**0**, c_thermo_press**3**, ...

Adding keywords for the thermo_style custom command (which can then be accessed by variables) was discussed here on this page.

Adding a new math function of one or two arguments can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding a new group function can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location. You may need to add a new method to the Group class as well (see the group.cpp file).

Accessing a new atom-based vector can be done by editing one section of the Variable::evaulate() method. Search for the word "customize" to find the appropriate location.

Adding new compute styles (whose calculated values can then be accessed by variables) was discussed here on this page.

10.15 Submitting new features for inclusion in LAMMPS

We encourage users to submit new features that they add to LAMMPS to the developers, especially if you think the features will be of interest to other users. If they are broadly useful we may add them as core files to LAMMPS or as part of a standard package. Else we will add them as a user-contributed package or file. Examples of user packages are in src sub-directories that start with USER. The USER-MISC package is simply a collection of (mostly) unrelated single files, which is the simplest way to have your contribution quickly added to the LAMMPS distribution. You can see a list of the both standard and user packages by typing "make package" in the LAMMPS src directory.

With user packages and files, all we are really providing (aside from the fame and fortune that accompanies having your name in the source code and on the Authors page of the LAMMPS WWW site), is a means for you to distribute your work to the LAMMPS user community and a mechanism for others to easily try out your new feature. This may help you find bugs or make contact with new collaborators. Note that you're also implicitly agreeing to support your code which means answer questions, fix bugs, and maintain it if LAMMPS changes.

The previous sections of this doc page describe how to add new features of various kinds to LAMMPS. Packages are simply collections of one or more new class files which are invoked as a new "style" within a LAMMPS input script. If designed correctly, these additions typically do not require changes to the main core of LAMMPS; they are simply add-on files. If you think your new feature requires non-trivial changes in core LAMMPS files, you'll need to communicate with the developers, since we may or may not want to make those changes. An example of a trivial change is making a parent-class method "virtual" when you derive a new child class from it.

Here is what you need to do to submit a user package or single file for our consideration. Following these steps will save time for both you and us. See existing package files for examples.

- All source files you provide must compile with the most current version of LAMMPS.

- If you want your file(s) to be added to main LAMMPS or one of its standard packages, then it needs to be written in a style compatible with other LAMMPS source files. This is so the developers can understand it and hopefully maintain it. This basically means that the code accesses data structures, performs its operations, and is formatted similar to other LAMMPS source files, including the use of the error class for error and warning messages.
- If your contribution is a single file (actually a *.cpp and *.h file) it can most rapidly be added to the USER-MISC directory. Send us the one-line entry to add to the USER-MISC/README file in that dir, along with the 2 source files. You can do this multiple times if you wish to contribute several individual features.
- If your contribution is several related featues, it is probably best to make it a user package directory with a name like USER-FOO. In addition to your new files, the directory should contain a README, and Install.csh file. The README text file should contain your name and contact information and a brief description of what your new package does. The Install.csh file enables LAMMPS to include and exclude your package. See other README and Install.sh files in other USER directories as examples. Send us a tarball of this USER-FOO directory.
- Your new source files need to have the LAMMPS copyright, GPL notice, and your name at the top, like other LAMMPS source files. They need to create a class that is inside the LAMMPS namespace. If the file is for one of the USER packages, including USER-MISC, then we are not as picky about the coding style (see above). I.e. they do not need to be in the same stylistic format and syntax as other LAMMPS files, though that would be nice.
- Finally, you must also send a documentation file for each new command or style you are adding to LAMMPS. This will be one file for a single-file feature. For a package, it might be several files. These are simple text files which we will convert to HTML. They must be in the same format as other *.txt files in the lammps/doc directory for similar commands and styles. The "Restrictions" section of the doc page should indicate that your command is only available if LAMMPS is built with the appropriate USER-MISC or USER-FOO package. See other user package doc files for an example of how to do this. The txt2html tool we use to do the conversion can be downloaded from this site, so you can perform the HTML conversion yourself to proofread your doc page.

Note that the more clear and self-explanatory you make your doc and README files, the more likely it is that users will try out your new feature.

(Foo) Foo, Morefoo, and Maxfoo, J of Classic Potentials, 75, 345 (1997).

4. Packages

This section gives a quick overview of the add-on packages that extend LAMMPS functionality.

LAMMPS includes many optional packages, which are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of all packages by typing "make package" from within the src directory of the LAMMPS distribution.

See Section_start 3 of the manual for details on how to include/exclude specific packages as part of the LAMMPS build process, and for more details about the differences between standard packages and user packages in LAMMPS.

Below, the packages currently availabe in LAMMPS are listed. For standard packages, just a one-line description is given. For user packages, more details are provided.

4.1 Standard packages

The current list of standard packages is as follows:

Package	Description	Author(s)	Doc page	Example	Library
ASPHERE	aspherical particles	-	Section_howto	ellipse	-
CLASS2	class 2 force fields	-	pair_style lj/class2	-	-
COLLOID	colloidal particles	-	atom_style colloid	colloid	-
DIPOLE	point dipole particles	-	pair_style dipole/cut	dipole	-
FLD	Fast Lubrication Dynamics	Kumar & Bybee & Higdon (1)	pair_style lubricateU	-	-
GPU	GPU-enabled potentials	Mike Brown (ORNL)	Section accelerate	gpu	lib/gpu
GRANULAR	granular systems	-	Section_howto	pour	-
KIM	openKIM potentials	Smirichinski & Elliot & Tadmor (3)	pair_style kim	kim	KIM
KSPACE	long-range Coulombic solvers	-	kspace_style	peptide	-
MANYBODY	many-body potentials	-	pair_style tersoff	shear	-
MEAM	modified EAM potential	Greg Wagner (Sandia)	pair_style meam	meam	lib/meam
MC	Monte Carlo options	-	fix gcmc	-	-
MOLECULE	molecular system force fields	-	Section_howto	peptide	-
OPT	optimized pair potentials	Fischer & Richie & Natoli (2)	Section accelerate	-	-
PERI	Peridynamics models	Mike Parks (Sandia)	pair_style peri	peri	-
POEMS		Rudra Mukherjee (JPL)	fix poems	rigid	lib/poems

	coupled rigid body motion				
REAX	ReaxFF potential	Aidan Thompson (Sandia)	pair_style reax	reax	lib/reax
REPLICA	multi-replica methods	-	Section_howto	tad	-
RIGID	rigid bodies	-	fix rigid	rigid	-
SHOCK	shock loading methods	-	fix msst	-	-
SRD	stochastic rotation dynamics	-	fix srd	srd	-
VORONOI	Voronoi tesselations	Daniel Schwen (LANL)	compute voronoi/atom	-	Voro++
XTC	dumps in XTC format	-	dump	-	-

The "Authors" column lists a name(s) if a specific person is responible for creating and maintaining the package.

(1) The FLD package was created by Amit Kumar and Michael Bybee from Jonathan Higdon's group at UIUC.

(2) The OPT package was created by James Fischer (High Performance Technologies), David Richie, and Vincent Natoli (Stone Ridge Technolgy).

(3) The KIM package was created by Valeriu Smirichinski, Ryan Elliott, and Ellad Tadmor (U Minn).

The "Doc page" column links to either a portion of the Section_howto of the manual, or an input script command implemented as part of the package.

The "Example" column is a sub-directory in the examples directory of the distribution which has an input script that uses the package. E.g. "peptide" refers to the examples/peptide directory.

The "Library" column lists an external library which must be built first and which LAMMPS links to when it is built. If it is listed as lib/package, then the code for the library is under the lib directory of the LAMMPS distribution. See the lib/package/README file for info on how to build the library. If it is not listed as lib/package, then it is a third-party library not included in the LAMMPS distribution. See the src/package/README or src/package/Makefile.lammps file for info on where to download the library. Section start of the manual also gives details on how to build LAMMPS with both kinds of auxiliary libraries.

4.2 User packages

The current list of user-contributed packages is as follows:

Package	Description	Author(s)	Doc page	Example	Pic/movie	Library
USER-ATC	atom-to-continuum coupling	Jones & Templeton & Zimmerman (2)	fix atc	USER/atc	atc	lib/atc
USER-AWPMD	wave-packet MD	Ilya Valuev (JIHT)	pair_style awpmd/cut	USER/awpmd	-	lib/awpmd
USER-CG-CMM	coarse-graining model	Axel Kohlmeyer (Temple U)	pair_style lj/sdk	USER/cg-cmm	cg	-
USER-COLVARS	collective variables	Fiorin & Henin & Kohlmeyer (3)	fix colvars	USER/colvars	colvars	lib/colvars
USER-CUDA	NVIDIA GPU styles	Christian Trott (U Tech Ilmenau)	Section accelerate	USER/cuda	-	lib/cuda
USER-EFF	electron force field		pair_style eff/cut	USER/eff	eff	-

		Andres Jaramillo-Botero (Caltech)				
USER-LB	Lattice Boltzmann fluid	Colin Denniston (U Western Ontario)	fix lb/fluid	USER/lb	-	-
USER-MISC	single-file contributions	USER-MISC/README	USER-MISC/README	-	-	-
USER-MOLFILE	VMD molfile plug-ins	Axel Kohlmeyer (Temple U)	dump molfile	-	-	VMD-MOL
USER-OMP	OpenMP threaded styles	Axel Kohlmeyer (Temple U)	Section accelerate	-	-	-
USER-REAXC	C version of ReaxFF	Metin Aktulga (LBNL)	pair_style reaxc	reax	-	-
USER-SPH	smoothed particle hydrodynamics	Georg Ganzenmuller (EMI)	userguide.pdf	USER/sph	sph	-

The "Authors" column lists a name(s) if a specific person is responible for creating and maintaining the package.

If the Library is not listed as lib/package, then it is a third-party library not included in the LAMMPS distribution. See the src/package/Makefile.lammps file for info on where to download the library from.

(2) The ATC package was created by Reese Jones, Jeremy Templeton, and Jon Zimmerman (Sandia).

(3) The COLVARS package was created by Axel Kohlmeyer (Temple U) using the colvars module library written by Giacomo Fiorin (Temple U) and Jerome Henin (LISM, Marseille, France).

The "Doc page" column links to either a portion of the Section_howto of the manual, or an input script command implemented as part of the package, or to additional documentation provided witht he package.

The "Example" column is a sub-directory in the examples directory of the distribution which has an input script that uses the package. E.g. "peptide" refers to the examples/peptide directory. USER/cuda refers to the examples/USER/cuda directory.

The "Library" column lists an external library which must be built first and which LAMMPS links to when it is built. If it is listed as lib/package, then the code for the library is under the lib directory of the LAMMPS distribution. See the lib/package/README file for info on how to build the library. If it is not listed as lib/package, then it is a third-party library not included in the LAMMPS distribution. See the src/package/Makefile.lammps file for info on where to download the library. Section start of the manual also gives details on how to build LAMMPS with both kinds of auxiliary libraries.

More details on each package, from the USER-*/README file is given below.

USER-MISC package

The files in this package are a potpourri of (mostly) unrelated features contributed to LAMMPS by users. Each feature is a single pair of files (*.cpp and *.h).

More information about each feature can be found by reading its doc page in the LAMMPS doc directory. The doc page which lists all LAMMPS input script commands is as follows:

Section_commands

User-contributed features are listed at the bottom of the fix, compute, pair, etc sections.

The list of features and author of each is given in the src/USER-MISC/README file.

You should contact the author directly if you have specific questions about the feature or its coding.

USER-ATC package

This package implements a "fix atc" command which can be used in a LAMMPS input script. This fix can be employed to either do concurrent coupling of MD with FE-based physics surrogates or on-the-fly post-processing of atomic information to continuum fields.

See the doc page for the fix atc command to get started. At the bottom of the doc page are many links to additional documentation contained in the doc/USER/atc directory.

There are example scripts for using this package in examples/USER/atc.

This package uses an external library in lib/atc which must be compiled before making LAMMPS. See the lib/atc/README file and the LAMMPS manual for information on building LAMMPS with external libraries.

The primary people who created this package are Reese Jones (rjones at sandia.gov), Jeremy Templeton (jatempl at sandia.gov) and Jon Zimmerman (jzimmer at sandia.gov) at Sandia. Contact them directly if you have questions.

USER-AWPMD package

This package contains a LAMMPS implementation of the Antisymmetrized Wave Packet Molecular Dynamics (AWPMD) method.

See the doc page for the pair_style awpmd/cut command to get started.

There are example scripts for using this package in examples/USER/awpmd.

This package uses an external library in lib/awpmd which must be compiled before making LAMMPS. See the lib/awpmd/README file and the LAMMPS manual for information on building LAMMPS with external libraries.

The person who created this package is Ilya Valuev at the JIHT in Russia (valuev at physik.hu-berlin.de). Contact him directly if you have questions.

USER-COLVARS package

This package implements the "fix colvars" command which can be used in a LAMMPS input script.

This fix allows to use "collective variables" to implement Adaptive Biasing Force, Metadynamics, Steered MD, Umbrella Sampling and Restraints. This code consists of two parts:

- A portable collective variable module library written and maintained by Giacomo Fiorin (ICMS, Temple University, Philadelphia, PA, USA) and Jerome Henin (LISM, CNRS, Marseille, France). This code is located in the directory lib/colvars and needs to be compiled first.
- The colvars fix and an interface layer, exchanges information between LAMMPS and the collective variable module.

See the doc page of fix colvars for more details.

USER-MISC package 1178

There are example scripts for using this package in examples/USER/colvars

This is a very new interface that does not yet support all features in the module and will see future optimizations and improvements. The colvars module library is also available in NAMD has been thoroughly used and tested there. Bugs and problems are likely due to the interface layers code. Thus the current version of this package should be considered beta quality.

The person who created this package is Axel Kohlmeyer at Temple U (akohlmey at gmail.com). Contact him directly if you have questions.

USER-CG-CMM package

This package implements 3 commands which can be used in a LAMMPS input script:

- pair_style lj/sdk
- pair_style lj/sdk/coul/long
- angle_style sdk

These styles allow coarse grained MD simulations with the parametrization of Shinoda, DeVane, Klein, Mol Sim, 33, 27 (2007) (SDK), with extensions to simulate ionic liquids, electrolytes, lipids and charged amino acids.

See the doc pages for these commands for details.

There are example scripts for using this package in examples/USER/cg-cmm.

This is the second generation implementation reducing the the clutter of the previous version. For many systems with electrostatics, it will be faster to use pair_style hybrid/overlay with lj/sdk and coul/long instead of the combined lj/sdk/coul/long style. since the number of charged atom types is usually small. For any other coulomb interactions this is now required. To exploit this property, the use of the kspace_style pppm/cg is recommended over regular pppm. For all new styles, input file backward compatibility is provided. The old implementation is still available through appending the /old suffix. These will be discontinued and removed after the new implementation has been fully validated.

The current version of this package should be considered beta quality. The CG potentials work correctly for "normal" situations, but have not been testing with all kinds of potential parameters and simulation systems.

The person who created this package is Axel Kohlmeyer at Temple U (akohlmey at gmail.com). Contact him directly if you have questions.

USER-CUDA package

This package provides acceleration of various LAMMPS pair styles, fix styles, compute styles, and long-range Coulombics via PPPM for NVIDIA GPUs.

See this section of the manual to get started:

Section_accelerate

There are example scripts for using this package in examples/USER/cuda.

This package uses an external library in lib/cuda which must be compiled before making LAMMPS. See the lib/cuda/README file and the LAMMPS manual for information on building LAMMPS with external libraries.

The person who created this package is Christian Trott at the University of Technology Ilmenau, Germany (christian.trott at tu-ilmenau.de). Contact him directly if you have questions.

USER-EFF package

This package contains a LAMMPS implementation of the electron Force Field (eFF) currently under development at Caltech, as described in A. Jaramillo-Botero, J. Su, Q. An, and W.A. Goddard III, JCC, 2010. The eFF potential was first introduced by Su and Goddard, in 2007.

eFF can be viewed as an approximation to QM wave packet dynamics and Fermionic molecular dynamics, combining the ability of electronic structure methods to describe atomic structure, bonding, and chemistry in materials, and of plasma methods to describe nonequilibrium dynamics of large systems with a large number of highly excited electrons. We classify it as a mixed QM-classical approach rather than a conventional force field method, which introduces QM-based terms (a spin-dependent repulsion term to account for the Pauli exclusion principle and the electron wavefunction kinetic energy associated with the Heisenberg principle) that reduce, along with classical electrostatic terms between nuclei and electrons, to the sum of a set of effective pairwise potentials. This makes eFF uniquely suited to simulate materials over a wide range of temperatures and pressures where electronically excited and ionized states of matter can occur and coexist.

The necessary customizations to the LAMMPS core are in place to enable the correct handling of explicit electron properties during minimization and dynamics.

See the doc page for the pair_style eff/cut command to get started.

There are example scripts for using this package in examples/USER/eff.

There are auxiliary tools for using this package in tools/eff.

The person who created this package is Andres Jaramillo-Botero at CalTech (ajaramil at wag.caltech.edu). Contact him directly if you have questions.

USER-OMP package

This package provides OpenMP multi-threading support and other optimizations of various LAMMPS pair styles, dihedral styles, and fix styles.

See this section of the manual to get started:

Section_accelerate

The person who created this package is Axel Kohlmeyer at Temple U (akohlmey at gmail.com). Contact him directly if you have questions.

USER-REAXC package

This package contains a implementation for LAMMPS of the ReaxFF force field. ReaxFF uses distance-dependent bond-order functions to represent the contributions of chemical bonding to the potential energy. It was originally developed by Adri van Duin and the Goddard group at CalTech.

The USER-REAXC version of ReaxFF (pair_style reax/c), implemented in C, should give identical or very similar results to pair_style reax, which is a ReaxFF implementation on top of a Fortran library, a version of which library was originally authored by Adri van Duin.

USER-CUDA package

The reax/c version should be somewhat faster and more scalable, particularly with respect to the charge equilibration calculation. It should also be easier to build and use since there are no complicating issues with Fortran memory allocation or linking to a Fortran library.

For technical details about this implemention of ReaxFF, see this paper:

Parallel and Scalable Reactive Molecular Dynamics: Numerical Methods and Algorithmic Techniques, H. M. Aktulga, J. C. Fogarty, S. A. Pandit, A. Y. Grama, Parallel Computing, in press (2011).

See the doc page for the pair_style reax/c command for details of how to use it in LAMMPS.

The person who created this package is Hasan Metin Aktulga (hmaktulga at lbl.gov), while at Purdue University. Contact him directly, or Aidan Thompson at Sandia (athomps at sandia.gov), if you have questions.

USER-SPH package

This package implements smoothed particle hydrodynamics (SPH) in LAMMPS. Currently, the package has the following features:

* Tait, ideal gas, Lennard-Jones equation of states, full support for complete (i.e. internal-energy dependent) equations of state * plain or Monaghans XSPH integration of the equations of motion * density continuity or density summation to propagate the density field * commands to set internal energy and density of particles from the input script * output commands to access internal energy and density for dumping and thermo output

See the file doc/USER/sph/SPH_LAMMPS_userguide.pdf to get started.

There are example scripts for using this package in examples/USER/sph.

The person who created this package is Georg Ganzenmuller at the Fraunhofer-Institute for High-Speed Dynamics, Ernst Mach Institute in Germany (georg.ganzenmueller at emi.fhg.de). Contact him directly if you have questions.

8. Performance & scalability

LAMMPS performance on several prototypical benchmarks and machines is discussed on the Benchmarks page of the LAMMPS WWW Site where CPU timings and parallel efficiencies are listed. Here, the benchmarks are described briefly and some useful rules of thumb about their performance are highlighted.

These are the 5 benchmark problems:

1. LJ = atomic fluid, Lennard-Jones potential with 2.5 sigma cutoff (55 neighbors per atom), NVE integration
2. Chain = bead-spring polymer melt of 100-mer chains, FENE bonds and LJ pairwise interactions with a 2^(1/6) sigma cutoff (5 neighbors per atom), NVE integration
3. EAM = metallic solid, Cu EAM potential with 4.95 Angstrom cutoff (45 neighbors per atom), NVE integration
4. Chute = granular chute flow, frictional history potential with 1.1 sigma cutoff (7 neighbors per atom), NVE integration
5. Rhodo = rhodopsin protein in solvated lipid bilayer, CHARMM force field with a 10 Angstrom LJ cutoff (440 neighbors per atom), particle-particle particle-mesh (PPPM) for long-range Coulombics, NPT integration

The input files for running the benchmarks are included in the LAMMPS distribution, as are sample output files. Each of the 5 problems has 32,000 atoms and runs for 100 timesteps. Each can be run as a serial benchmarks (on one processor) or in parallel. In parallel, each benchmark can be run as a fixed-size or scaled-size problem. For fixed-size benchmarking, the same 32K atom problem is run on various numbers of processors. For scaled-size benchmarking, the model size is increased with the number of processors. E.g. on 8 processors, a 256K-atom problem is run; on 1024 processors, a 32-million atom problem is run, etc.

A useful metric from the benchmarks is the CPU cost per atom per timestep. Since LAMMPS performance scales roughly linearly with problem size and timesteps, the run time of any problem using the same model (atom style, force field, cutoff, etc) can then be estimated. For example, on a 1.7 GHz Pentium desktop machine (Intel icc compiler under Red Hat Linux), the CPU run-time in seconds/atom/timestep for the 5 problems is

Problem:	LJ	Chain	EAM	Chute	Rhodopsin
CPU/atom/step:	4.55E-6	2.18E-6	9.38E-6	2.18E-6	1.11E-4
Ratio to LJ:	1.0	0.48	2.06	0.48	24.5

The ratios mean that if the atomic LJ system has a normalized cost of 1.0, the bead-spring chains and granular systems run 2x faster, while the EAM metal and solvated protein models run 2x and 25x slower respectively. The bulk of these cost differences is due to the expense of computing a particular pairwise force field for a given number of neighbors per atom.

Performance on a parallel machine can also be predicted from the one-processor timings if the parallel efficiency can be estimated. The communication bandwidth and latency of a particular parallel machine affects the efficiency. On most machines LAMMPS will give fixed-size parallel efficiencies on these benchmarks above 50% so long as the atoms/processor count is a few 100 or greater - i.e. on 64 to 128 processors. Likewise, scaled-size parallel efficiencies will typically be 80% or greater up to very large processor counts. The benchmark data on the LAMMPS WWW Site gives specific examples on some different machines, including a run of 3/4 of a billion LJ atoms on 1500 processors that ran at 85% parallel efficiency.

11. Python interface to LAMMPS

This section describes how to build and use LAMMPS via a Python interface.

- 11.1 Building LAMMPS as a shared library
- 11.2 Installing the Python wrapper into Python
- 11.3 Extending Python with MPI to run in parallel
- 11.4 Testing the Python-LAMMPS interface
- 11.5 Using LAMMPS from Python
- 11.6 Example Python scripts that use LAMMPS

The LAMMPS distribution includes the file python/lammps.py which wraps the library interface to LAMMPS. This file makes it is possible to run LAMMPS, invoke LAMMPS commands or give it an input script, extract LAMMPS results, an modify internal LAMMPS variables, either from a Python script or interactively from a Python prompt. You can do the former in serial or parallel. Running Python interactively in parallel does not generally work, unless you have a package installed that extends your Python to enable multiple instances of Python to read what you type.

Python is a powerful scripting and programming language which can be used to wrap software like LAMMPS and other packages. It can be used to glue multiple pieces of software together, e.g. to run a coupled or multiscale model. See Section section of the manual and the couple directory of the distribution for more ideas about coupling LAMMPS to other codes. See Section_start 4 about how to build LAMMPS as a library, and Section_howto 19 for a description of the library interface provided in src/library.cpp and src/library.h and how to extend it for your needs. As described below, that interface is what is exposed to Python. It is designed to be easy to add functions to. This can easily extend the Python inteface as well. See details below.

By using the Python interface, LAMMPS can also be coupled with a GUI or other visualization tools that display graphs or animations in real time as LAMMPS runs. Examples of such scripts are inlcluded in the python directory.

Two advantages of using Python are how concise the language is, and that it can be run interactively, enabling rapid development and debugging of programs. If you use it to mostly invoke costly operations within LAMMPS, such as running a simulation for a reasonable number of timesteps, then the overhead cost of invoking LAMMPS thru Python will be negligible.

Before using LAMMPS from a Python script, you need to do two things. You need to build LAMMPS as a dynamic shared library, so it can be loaded by Python. And you need to tell Python how to find the library and the Python wrapper file python/lammps.py. Both these steps are discussed below. If you wish to run LAMMPS in parallel from Python, you also need to extend your Python with MPI. This is also discussed below.

The Python wrapper for LAMMPS uses the amazing and magical (to me) "ctypes" package in Python, which auto-generates the interface code needed between Python and a set of C interface routines for a library. Ctypes is part of standard Python for versions 2.5 and later. You can check which version of Python you have installed, by simply typing "python" at a shell prompt.

11.1 Building LAMMPS as a shared library

Instructions on how to build LAMMPS as a shared library are given in Section_start 5. A shared library is one that is dynamically loadable, which is what Python requires. On Linux this is a library file that ends in ".so",

not ".a".

From the src directory, type

```
make makeshlib
make -f Makefile.shlib foo
```

where foo is the machine target name, such as linux or g++ or serial. This should create the file liblammps_foo.so in the src directory, as well as a soft link liblammps.so, which is what the Python wrapper will load by default. Note that if you are building multiple machine versions of the shared library, the soft link is always set to the most recently built version.

If this fails, see [Section_start 5](#) for more details, especially if your LAMMPS build uses auxiliary libraries like MPI or FFTW which may not be built as shared libraries on your system.

11.2 Installing the Python wrapper into Python

For Python to invoke LAMMPS, there are 2 files it needs to know about:

- python/lammps.py
- src/liblammps.so

Lammps.py is the Python wrapper on the LAMMPS library interface. Liblammps.so is the shared LAMMPS library that Python loads, as described above.

You can insure Python can find these files in one of two ways:

- set two environment variables
- run the python/install.py script

If you set the paths to these files as environment variables, you only have to do it once. For the csh or tcsh shells, add something like this to your ~/.cshrc file, one line for each of the two files:

```
setenv PYTHONPATH $PYTHONPATH:/home/sjplimp/lammps/python
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:/home/sjplimp/lammps/src
```

If you use the python/install.py script, you need to invoke it every time you rebuild LAMMPS (as a shared library) or make changes to the python/lammps.py file.

You can invoke install.py from the python directory as

```
% python install.py [libdir] [pydir]
```

The optional libdir is where to copy the LAMMPS shared library to; the default is /usr/local/lib. The optional pydir is where to copy the lammps.py file to; the default is the site-packages directory of the version of Python that is running the install script.

Note that libdir must be a location that is in your default LD_LIBRARY_PATH, like /usr/local/lib or /usr/lib. And pydir must be a location that Python looks in by default for imported modules, like its site-packages dir. If you want to copy these files to non-standard locations, such as within your own user space, you will need to set your PYTHONPATH and LD_LIBRARY_PATH environment variables accordingly, as above.

If the install.py script does not allow you to copy files into system directories, prefix the python command with "sudo". If you do this, make sure that the Python that root runs is the same as the Python you run. E.g.

you may need to do something like

```
% sudo /usr/local/bin/python install.py [libdir] [pydir]
```

You can also invoke install.py from the make command in the src directory as

```
% make install-python
```

In this mode you cannot append optional arguments. Again, you may need to prefix this with "sudo". In this mode you cannot control which Python is invoked by root.

Note that if you want Python to be able to load different versions of the LAMMPS shared library (see this section below), you will need to manually copy files like liblammps_g++.so into the appropriate system directory. This is not needed if you set the LD_LIBRARY_PATH environment variable as described above.

11.3 Extending Python with MPI to run in parallel

If you wish to run LAMMPS in parallel from Python, you need to extend your Python with an interface to MPI. This also allows you to make MPI calls directly from Python in your script, if you desire.

There are several Python packages available that purport to wrap MPI as a library and allow MPI functions to be called from Python.

These include

- pyMPI
- maroonmpi
- mpi4py
- myMPI
- Pypar

All of these except pyMPI work by wrapping the MPI library and exposing (some portion of) its interface to your Python script. This means Python cannot be used interactively in parallel, since they do not address the issue of interactive input to multiple instances of Python running on different processors. The one exception is pyMPI, which alters the Python interpreter to address this issue, and (I believe) creates a new alternate executable (in place of "python" itself) as a result.

In principle any of these Python/MPI packages should work to invoke LAMMPS in parallel and MPI calls themselves from a Python script which is itself running in parallel. However, when I downloaded and looked at a few of them, their documentation was incomplete and I had trouble with their installation. It's not clear if some of the packages are still being actively developed and supported.

The one I recommend, since I have successfully used it with LAMMPS, is Pypar. Pypar requires the ubiquitous Numpy package be installed in your Python. After launching python, type

```
import numpy
```

to see if it is installed. If not, here is how to install it (version 1.3.0b1 as of April 2009). Unpack the numpy tarball and from its top-level directory, type

```
python setup.py build
sudo python setup.py install
```

The "sudo" is only needed if required to copy Numpy files into your Python distribution's site-packages directory.

To install Pypar (version pypar-2.1.4_94 as of Aug 2012), unpack it and from its "source" directory, type

```
python setup.py build
sudo python setup.py install
```

Again, the "sudo" is only needed if required to copy Pypar files into your Python distribution's site-packages directory.

If you have successully installed Pypar, you should be able to run Python and type

```
import pypar
```

without error. You should also be able to run python in parallel on a simple test script

```
% mpirun -np 4 python test.py
```

where test.py contains the lines

```
import pypar
print "Proc %d out of %d procs" % (pypar.rank(),pypar.size())
```

and see one line of output for each processor you run on.

IMPORTANT NOTE: To use Pypar and LAMMPS in parallel from Python, you must insure both are using the same version of MPI. If you only have one MPI installed on your system, this is not an issue, but it can be if you have multiple MPIs. Your LAMMPS build is explicit about which MPI it is using, since you specify the details in your lo-level src/MAKE/Makefile.foo file. Pypar uses the "mpicc" command to find information about the MPI it uses to build against. And it tries to load "libmpi.so" from the LD_LIBRARY_PATH. This may or may not find the MPI library that LAMMPS is using. If you have problems running both Pypar and LAMMPS together, this is an issue you may need to address, e.g. by moving other MPI installations so that Pypar finds the right one.

11.4 Testing the Python-LAMMPS interface

To test if LAMMPS is callable from Python, launch Python interactively and type:

```
>>> from lammps import lammps
>>> lmp = lammps()
```

If you get no errors, you're ready to use LAMMPS from Python. If the 2nd command fails, the most common error to see is

```
OSError: Could not load LAMMPS dynamic library
```

which means Python was unable to load the LAMMPS shared library. This typically occurs if the system can't find the LAMMPS shared library or one of the auxiliary shared libraries it depends on, or if something about the library is incompatible with your Python. The error message should give you an indication of what went wrong.

You can also test the load directly in Python as follows, without first importing from the lammps.py file:

```
>>> from ctypes import CDLL
```

```
>>> CDLL("liblammps.so")
```

If an error occurs, carefully go thru the steps in <u>Section_start 5</u> and above about building a shared library and about insuring Python can find the necessary two files it needs.

Test LAMMPS and Python in serial:

To run a LAMMPS test in serial, type these lines into Python interactively from the bench directory:

```
>>> from lammps import lammps
>>> lmp = lammps()
>>> lmp.file("in.lj")
```

Or put the same lines in the file test.py and run it as

```
% python test.py
```

Either way, you should see the results of running the in.lj benchmark on a single processor appear on the screen, the same as if you had typed something like:

```
lmp_g++ <in.lj
```

Test LAMMPS and Python in parallel:

To run LAMMPS in parallel, assuming you have installed the <u>Pypar</u> package as discussed above, create a test.py file containing these lines:

```
import pypar
from lammps import lammps
lmp = lammps()
lmp.file("in.lj")
print "Proc %d out of %d procs has" % (pypar.rank(),pypar.size()),lmp
pypar.finalize()
```

You can then run it in parallel as:

```
% mpirun -np 4 python test.py
```

and you should see the same output as if you had typed

```
% mpirun -np 4 lmp_g++ <in.lj
```

Note that if you leave out the 3 lines from test.py that specify Pypar commands you will instantiate and run LAMMPS independently on each of the P processors specified in the mpirun command. In this case you should get 4 sets of output, each showing that a LAMMPS run was made on a single processor, instead of one set of output showing that LAMMPS ran on 4 processors. If the 1-processor outputs occur, it means that Pypar is not working correctly.

Also note that once you import the PyPar module, Pypar initializes MPI for you, and you can use MPI calls directly in your Python script, as described in the Pypar documentation. The last line of your Python script should be pypar.finalize(), to insure MPI is shut down correctly.

Running Python scripts:

Note that any Python script (not just for LAMMPS) can be invoked in one of several ways:

```
% python foo.script
% python -i foo.script
```

```
% foo.script
```

The last command requires that the first line of the script be something like this:

```
#!/usr/local/bin/python
#!/usr/local/bin/python -i
```

where the path points to where you have Python installed, and that you have made the script file executable:

```
% chmod +x foo.script
```

Without the "-i" flag, Python will exit when the script finishes. With the "-i" flag, you will be left in the Python interpreter when the script finishes, so you can type subsequent commands. As mentioned above, you can only run Python interactively when running Python on a single processor, not in parallel.

11.5 Using LAMMPS from Python

The Python interface to LAMMPS consists of a Python "lammps" module, the source code for which is in python/lammps.py, which creates a "lammps" object, with a set of methods that can be invoked on that object. The sample Python code below assumes you have first imported the "lammps" module in your Python script, as follows:

```
from lammps import lammps
```

These are the methods defined by the lammps module. If you look at the file src/library.cpp you will see that they correspond one-to-one with calls you can make to the LAMMPS library from a C++ or C or Fortran program.

```
lmp = lammps()              # create a LAMMPS object using the default liblammps.so library
lmp = lammps("g++")         # create a LAMMPS object using the liblammps_g++.so library
lmp = lammps("",list)       # ditto, with command-line args, e.g. list = ["-echo","screen"]
lmp = lammps("g++",list)

lmp.close()                 # destroy a LAMMPS object

lmp.file(file)              # run an entire input script, file = "in.lj"
lmp.command(cmd)            # invoke a single LAMMPS command, cmd = "run 100"

xlo = lmp.extract_global(name,type)   # extract a global quantity
                                      # name = "boxxlo", "nlocal", etc
                                      # type = 0 = int
                                      #        1 = double

coords = lmp.extract_atom(name,type)       # extract a per-atom quantity
                                           # name = "x", "type", etc
                                           # type = 0 = vector of ints
                                           #        1 = array of ints
                                           #        2 = vector of doubles
                                           #        3 = array of doubles

eng = lmp.extract_compute(id,style,type)  # extract value(s) from a compute
v3 = lmp.extract_fix(id,style,type,i,j)   # extract value(s) from a fix
                                          # id = ID of compute or fix
                                          # style = 0 = global data
                                          #         1 = per-atom data
                                          #         2 = local data
                                          # type = 0 = scalar
                                          #        1 = vector
```

```
                                        #        2 = array
                                        # i,j = indices of value in global vector or array

var = lmp.extract_variable(name,group,flag)  # extract value(s) from a variable
                                        # name = name of variable
                                        # group = group ID (ignored for equal-style variable
                                        # flag = 0 = equal-style variable
                                        #        1 = atom-style variable


natoms = lmp.get_natoms()               # total # of atoms as int
data = lmp.gather_atoms(name,type,count)  # return atom attribute of all atoms gathered into data
                                        # name = "x", "charge", "type", etc
                                        # count = # of per-atom values, 1 or 3, etc
lmp.scatter_atoms(name,type,count,data)  # scatter atom attribute of all atoms from data, ordere
                                        # name = "x", "charge", "type", etc
                                        # count = # of per-atom values, 1 or 3, etc
```

IMPORTANT NOTE: Currently, the creation of a LAMMPS object from within lammps.py does not take an MPI communicator as an argument. There should be a way to do this, so that the LAMMPS instance runs on a subset of processors if desired, but I don't know how to do it from Pypar. So for now, it runs with MPI_COMM_WORLD, which is all the processors. If someone figures out how to do this with one or more of the Python wrappers for MPI, like Pypar, please let us know and we will amend these doc pages.

Note that you can create multiple LAMMPS objects in your Python script, and coordinate and run multiple simulations, e.g.

```
from lammps import lammps
lmp1 = lammps()
lmp2 = lammps()
lmp1.file("in.file1")
lmp2.file("in.file2")
```

The file() and command() methods allow an input script or single commands to be invoked.

The extract_global(), extract_atom(), extract_compute(), extract_fix(), and extract_variable() methods return values or pointers to data structures internal to LAMMPS.

For extract_global() see the src/library.cpp file for the list of valid names. New names could easily be added. A double or integer is returned. You need to specify the appropriate data type via the type argument.

For extract_atom(), a pointer to internal LAMMPS atom-based data is returned, which you can use via normal Python subscripting. See the extract() method in the src/atom.cpp file for a list of valid names. Again, new names could easily be added. A pointer to a vector of doubles or integers, or a pointer to an array of doubles (double **) or integers (int **) is returned. You need to specify the appropriate data type via the type argument.

For extract_compute() and extract_fix(), the global, per-atom, or local data calulated by the compute or fix can be accessed. What is returned depends on whether the compute or fix calculates a scalar or vector or array. For a scalar, a single double value is returned. If the compute or fix calculates a vector or array, a pointer to the internal LAMMPS data is returned, which you can use via normal Python subscripting. The one exception is that for a fix that calculates a global vector or array, a single double value from the vector or array is returned, indexed by I (vector) or I and J (array). I,J are zero-based indices. The I,J arguments can be left out if not needed. See Section_howto 15 of the manual for a discussion of global, per-atom, and local data, and of scalar, vector, and array data types. See the doc pages for individual computes and fixes for a description of what they calculate and store.

For extract_variable(), an equal-style or atom-style variable is evaluated and its result returned.

For equal-style variables a single double value is returned and the group argument is ignored. For atom-style variables, a vector of doubles is returned, one value per atom, which you can use via normal Python subscripting. The values will be zero for atoms not in the specified group.

The get_natoms() method returns the total number of atoms in the simulation, as an int.

The gather_atoms() method returns a ctypes vector of ints or doubles as specified by type, of length count*natoms, for the property of all the atoms in the simulation specified by name, ordered by count and then by atom ID. The vector can be used via normal Python subscripting. If atom IDs are not consecutively ordered within LAMMPS, a None is returned as indication of an error.

Note that the data structure gather_atoms("x") returns is different from the data structure returned by extract_atom("x") in four ways. (1) Gather_atoms() returns a vector which you index as x[i]; extract_atom() returns an array which you index as x[i][j]. (2) Gather_atoms() orders the atoms by atom ID while extract_atom() does not. (3) Gathert_atoms() returns a list of all atoms in the simulation; extract_atoms() returns just the atoms local to each processor. (4) Finally, the gather_atoms() data structure is a copy of the atom coords stored internally in LAMMPS, whereas extract_atom() returns an array that effectively points directly to the internal data. This means you can change values inside LAMMPS from Python by assigning a new values to the extract_atom() array. To do this with the gather_atoms() vector, you need to change values in the vector, then invoke the scatter_atoms() method.

The scatter_atoms() method takes a vector of ints or doubles as specified by type, of length count*natoms, for the property of all the atoms in the simulation specified by name, ordered by bount and then by atom ID. It uses the vector of data to overwrite the corresponding properties for each atom inside LAMMPS. This requires LAMMPS to have its "map" option enabled; see the [atom_modify](#) command for details. If it is not, or if atom IDs are not consecutively ordered, no coordinates are reset.

The array of coordinates passed to scatter_atoms() must be a ctypes vector of ints or doubles, allocated and initialized something like this:

```
from ctypes import *
natoms = lmp.get_natoms()
n3 = 3*natoms
x = (n3*c_double)()
x0 = x coord of atom with ID 1
x1 = y coord of atom with ID 1
x2 = z coord of atom with ID 1
x3 = x coord of atom with ID 2
...
xn3-1 = z coord of atom with ID natoms
lmp.scatter_coords("x",1,3,x)
```

Alternatively, you can just change values in the vector returned by gather_atoms("x",1,3), since it is a ctypes vector of doubles.

As noted above, these Python class methods correspond one-to-one with the functions in the LAMMPS library interface in src/library.cpp and library.h. This means you can extend the Python wrapper via the following steps:

- Add a new interface function to src/library.cpp and src/library.h.
- Rebuild LAMMPS as a shared library.
- Add a wrapper method to python/lammps.py for this interface function.
- You should now be able to invoke the new interface function from a Python script. Isn't ctypes amazing?

11.6 Example Python scripts that use LAMMPS

These are the Python scripts included as demos in the python/examples directory of the LAMMPS distribution, to illustrate the kinds of things that are possible when Python wraps LAMMPS. If you create your own scripts, send them to us and we can include them in the LAMMPS distribution.

trivial.py	read/run a LAMMPS input script thru Python
demo.py	invoke various LAMMPS library interface routines
simple.py	mimic operation of couple/simple/simple.cpp in Python
gui.py	GUI go/stop/temperature-slider to control LAMMPS
plot.py	real-time temeperature plot with GnuPlot via Pizza.py
viz_tool.py	real-time viz via some viz package
vizplotgui_tool.py	combination of viz_tool.py and plot.py and gui.py

For the viz_tool.py and vizplotgui_tool.py commands, replace "tool" with "gl" or "atomeye" or "pymol" or "vmd", depending on what visualization package you have installed.

Note that for GL, you need to be able to run the Pizza.py GL tool, which is included in the pizza sub-directory. See the Pizza.py doc pages for more info:

Note that for AtomEye, you need version 3, and there is a line in the scripts that specifies the path and name of the executable. See the AtomEye WWW pages here or here for more details:

```
http://mt.seas.upenn.edu/Archive/Graphics/A
http://mt.seas.upenn.edu/Archive/Graphics/A3/A3.html
```

The latter link is to AtomEye 3 which has the scriping capability needed by these Python scripts.

Note that for PyMol, you need to have built and installed the open-source version of PyMol in your Python, so that you can import it from a Python script. See the PyMol WWW pages here or here for more details:

```
http://www.pymol.org
http://sourceforge.net/scm/?type=svn&group_id=4546
```

The latter link is to the open-source version.

Note that for VMD, you need a fairly current version (1.8.7 works for me) and there are some lines in the pizza/vmd.py script for 4 PIZZA variables that have to match the VMD installation on your system.

See the python/README file for instructions on how to run them and the source code for individual scripts for comments about what they do.

Here are screenshots of the vizplotgui_tool.py script in action for different visualization package options. Click to see larger images:
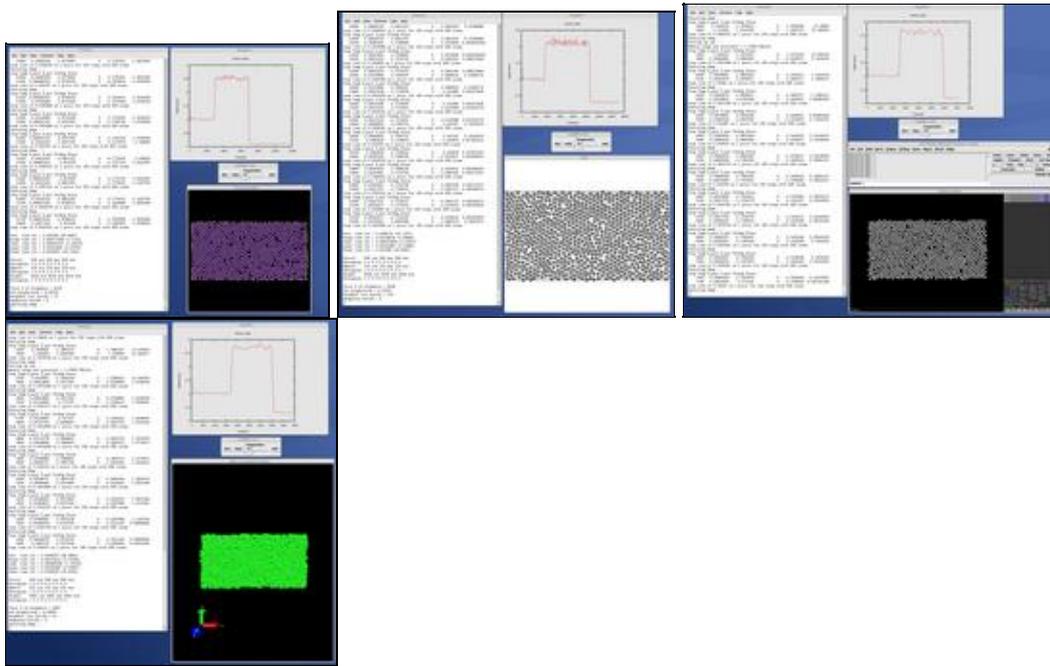
2. Getting Started

This section describes how to build and run LAMMPS, for both new and experienced users.

2.1 What's in the LAMMPS distribution

When you download LAMMPS you will need to unzip and untar the downloaded file with the following commands, after placing the file in an appropriate directory.

```
gunzip lammps*.tar.gz
tar xvf lammps*.tar
```

This will create a LAMMPS directory containing two files and several sub-directories:

README	text file
LICENSE	the GNU General Public License (GPL)
bench	benchmark problems
doc	documentation
examples	simple test problems
potentials	embedded atom method (EAM) potential files
src	source files
tools	pre- and post-processing tools

If you download one of the Windows executables from the download page, then you just get a single file:

```
lmp_windows.exe
```

Skip to the Running LAMMPS sections for info on how to launch these executables on a Windows box.

The Windows executables for serial or parallel only include certain packages and bug-fixes/upgrades listed on this page up to a certain date, as stated on the download page. If you want something with more packages or that is more current, you'll have to download the source tarball and build it yourself from source code using Microsoft Visual Studio, as described in the next section.

2.2 Making LAMMPS

This section has the following sub-sections:

- Read this first
- Steps to build a LAMMPS executable

- [Common errors that can occur when making LAMMPS](#)
- [Additional build tips](#)
- [Building for a Mac](#)
- [Building for Windows](#)

Read this first:

Building LAMMPS can be non-trivial. You may need to edit a makefile, there are compiler options to consider, additional libraries can be used (MPI, FFT, JPEG, PNG), LAMMPS packages may be included or excluded, some of these packages use auxiliary libraries which need to be pre-built, etc.

Please read this section carefully. If you are not comfortable with makefiles, or building codes on a Unix platform, or running an MPI job on your machine, please find a local expert to help you. Many compiling, linking, and run problems that users have are often not LAMMPS issues - they are peculiar to the user's system, compilers, libraries, etc. Such questions are better answered by a local expert.

If you have a build problem that you are convinced is a LAMMPS issue (e.g. the compiler complains about a line of LAMMPS source code), then please post a question to the [LAMMPS mail list](#).

If you succeed in building LAMMPS on a new kind of machine, for which there isn't a similar Makefile for in the src/MAKE directory, send it to the developers and we can include it in the LAMMPS distribution.

Steps to build a LAMMPS executable:

Step 0

The src directory contains the C++ source and header files for LAMMPS. It also contains a top-level Makefile and a MAKE sub-directory with low-level Makefile.* files for many machines. From within the src directory, type "make" or "gmake". You should see a list of available choices. If one of those is the machine and options you want, you can type a command like:

```
make linux
or
gmake mac
```

Note that on a multi-processor or multi-core platform you can launch a parallel make, by using the "-j" switch with the make command, which will build LAMMPS more quickly.

If you get no errors and an executable like lmp_linux or lmp_mac is produced, you're done; it's your lucky day.

Note that by default only a few of LAMMPS optional packages are installed. To build LAMMPS with optional packages, see [this section](#) below.

Step 1

If Step 0 did not work, you will need to create a low-level Makefile for your machine, like Makefile.foo. You should make a copy of an existing src/MAKE/Makefile.* as a starting point. The only portions of the file you need to edit are the first line, the "compiler/linker settings" section, and the "LAMMPS-specific settings" section.

Step 2

Change the first line of src/MAKE/Makefile.foo to list the word "foo" after the "#", and whatever other options it will set. This is the line you will see if you just type "make".

Step 3

The "compiler/linker settings" section lists compiler and linker settings for your C++ compiler, including optimization flags. You can use g++, the open-source GNU compiler, which is available on all Unix systems. You can also use mpicc which will typically be available if MPI is installed on your system, though you should check which actual compiler it wraps. Vendor compilers often produce faster code. On boxes with Intel CPUs, we suggest using the commercial Intel icc compiler, which can be downloaded from Intel's compiler site.

If building a C++ code on your machine requires additional libraries, then you should list them as part of the LIB variable.

The DEPFLAGS setting is what triggers the C++ compiler to create a dependency list for a source file. This speeds re-compilation when source (*.cpp) or header (*.h) files are edited. Some compilers do not support dependency file creation, or may use a different switch than -D. GNU g++ works with -D. If your compiler can't create dependency files, then you'll need to create a Makefile.foo patterned after Makefile.storm, which uses different rules that do not involve dependency files. Note that when you build LAMMPS for the first time on a new platform, a long list of *.d files will be printed out rapidly. This is not an error; it is the Makefile doing its normal creation of dependencies.

Step 4

The "system-specific settings" section has several parts. Note that if you change any -D setting in this section, you should do a full re-compile, after typing "make clean" (which will describe different clean options).

The LMP_INC variable is used to include options that turn on ifdefs within the LAMMPS code. The options that are currently recogized are:

- -DLAMMPS_GZIP
- -DLAMMPS_JPEG
- -DLAMMPS_PNG
- -DLAMMPS_FFMPEG
- -DLAMMPS_MEMALIGN
- -DLAMMPS_XDR
- -DLAMMPS_SMALLBIG
- -DLAMMPS_BIGBIG
- -DLAMMPS_SMALLSMALL
- -DLAMMPS_LONGLONG_TO_LONG
- -DPACK_ARRAY
- -DPACK_POINTER
- -DPACK_MEMCPY

The read_data and dump commands will read/write gzipped files if you compile with -DLAMMPS_GZIP. It requires that your machine supports the "popen" function in the standard runtime library and that a gzip executable can be found by LAMMPS during a run.

If you use -DLAMMPS_JPEG, the dump image command will be able to write out JPEG image files. For JPEG files, you must also link LAMMPS with a JPEG library, as described below. If you use -DLAMMPS_PNG, the dump image command will be able to write out PNG image files. For PNG files, you must also link LAMMPS with a PNG library, as described below. If neither of those two defines are used, LAMMPS will only be able to write out uncompressed PPM image files.

If you use -DLAMMPS_FFMPEG, the dump movie command will be available to support on-the-fly generation of rendered movies the need to store intermediate image files. It requires that your machines

supports the "popen" function in the standard runtime library and that an FFmpeg executable can be found by LAMMPS during the run.

Using -DLAMMPS_MEMALIGN= enables the use of the posix_memalign() call instead of malloc() when large chunks or memory are allocated by LAMMPS. This can help to make more efficient use of vector instructions of modern CPUS, since dynamically allocated memory has to be aligned on larger than default byte boundaries (e.g. 16 bytes instead of 8 bytes on x86 type platforms) for optimal performance.

If you use -DLAMMPS_XDR, the build will include XDR compatibility files for doing particle dumps in XTC format. This is only necessary if your platform does have its own XDR files available. See the Restrictions section of the dump command for details.

Use at most one of the -DLAMMPS_SMALLBIG, -DLAMMPS_BIGBIG, -D-DLAMMPS_SMALLSMALL settings. The default is -DLAMMPS_SMALLBIG. These settings refer to use of 4-byte (small) vs 8-byte (big) integers within LAMMPS, as specified in src/lmptype.h. The only reason to use the BIGBIG setting is to enable simulation of huge molecular systems with more than 2 billion atoms or to allow moving atoms to wrap back through a periodic box more than 512 times. The only reason to use the SMALLSMALL setting is if your machine does not support 64-bit integers. See the [Additional build tips](#) section below for more details.

The -DLAMMPS_LONGLONG_TO_LONG setting may be needed if your system or MPI version does not recognize "long long" data types. In this case a "long" data type is likely already 64-bits, in which case this setting will convert to that data type.

Using one of the -DPACK_ARRAY, -DPACK_POINTER, and -DPACK_MEMCPY options can make for faster parallel FFTs (in the PPPM solver) on some platforms. The -DPACK_ARRAY setting is the default. See the kspace_style command for info about PPPM. See Step 6 below for info about building LAMMPS with an FFT library.

Step 5

The 3 MPI variables are used to specify an MPI library to build LAMMPS with.

If you want LAMMPS to run in parallel, you must have an MPI library installed on your platform. If you use an MPI-wrapped compiler, such as "mpicc" to build LAMMPS, you should be able to leave these 3 variables blank; the MPI wrapper knows where to find the needed files. If not, and MPI is installed on your system in the usual place (under /usr/local), you also may not need to specify these 3 variables. On some large parallel machines which use "modules" for their compile/link environements, you may simply need to include the correct module in your build environment. Or the parallel machine may have a vendor-provided MPI which the compiler has no trouble finding.

Failing this, with these 3 variables you can specify where the mpi.h file (MPI_INC) and the MPI library file (MPI_PATH) are found and the name of the library file (MPI_LIB).

If you are installing MPI yourself, we recommend Argonne's MPICH2 or OpenMPI. MPICH can be downloaded from the [Argonne MPI site](#). OpenMPI can be downloaded from the [OpenMPI site](#). Other MPI packages should also work. If you are running on a big parallel platform, your system people or the vendor should have already installed a version of MPI, which is likely to be faster than a self-installed MPICH or OpenMPI, so find out how to build and link with it. If you use MPICH or OpenMPI, you will have to configure and build it for your platform. The MPI configure script should have compiler options to enable you to use the same compiler you are using for the LAMMPS build, which can avoid problems that can arise when linking LAMMPS to the MPI library.

If you just want to run LAMMPS on a single processor, you can use the dummy MPI library provided in src/STUBS, since you don't need a true MPI library installed on your system. See the

src/MAKE/Makefile.serial file for how to specify the 3 MPI variables in this case. You will also need to build the STUBS library for your platform before making LAMMPS itself. To build from the src directory, type "make stubs", or from the STUBS dir, type "make". This should create a libmpi_stubs.a file suitable for linking to LAMMPS. If the build fails, you will need to edit the STUBS/Makefile for your platform.

The file STUBS/mpi.c provides a CPU timer function called MPI_Wtime() that calls gettimeofday() . If your system doesn't support gettimeofday() , you'll need to insert code to call another timer. Note that the ANSI-standard function clock() rolls over after an hour or so, and is therefore insufficient for timing long LAMMPS simulations.

Step 6

The 3 FFT variables allow you to specify an FFT library which LAMMPS uses (for performing 1d FFTs) when running the particle-particle particle-mesh (PPPM) option for long-range Coulombics via the [kspace_style](#) command.

LAMMPS supports various open-source or vendor-supplied FFT libraries for this purpose. If you leave these 3 variables blank, LAMMPS will use the open-source [KISS FFT library](#), which is included in the LAMMPS distribution. This library is portable to all platforms and for typical LAMMPS simulations is almost as fast as FFTW or vendor optimized libraries. If you are not including the KSPACE package in your build, you can also leave the 3 variables blank.

Otherwise, select which kinds of FFTs to use as part of the FFT_INC setting by a switch of the form -DFFT_XXX. Recommended values for XXX are: MKL, SCSL, FFTW2, and FFTW3. Legacy options are: INTEL, SGI, ACML, and T3E. For backward compatability, using -DFFT_FFTW will use the FFTW2 library. Using -DFFT_NONE will use the KISS library described above.

You may also need to set the FFT_INC, FFT_PATH, and FFT_LIB variables, so the compiler and linker can find the needed FFT header and library files. Note that on some large parallel machines which use "modules" for their compile/link environements, you may simply need to include the correct module in your build environment. Or the parallel machine may have a vendor-provided FFT library which the compiler has no trouble finding.

FFTW is a fast, portable library that should also work on any platform. You can download it from [www.fftw.org](#). Both the legacy version 2.1.X and the newer 3.X versions are supported as -DFFT_FFTW2 or -DFFT_FFTW3. Building FFTW for your box should be as simple as ./configure; make. Note that on some platforms FFTW2 has been pre-installed, and uses renamed files indicating the precision it was compiled with, e.g. sfftw.h, or dfftw.h instead of fftw.h. In this case, you can specify an additional define variable for FFT_INC called -DFFTW_SIZE, which will select the correct include file. In this case, for FFT_LIB you must also manually specify the correct library, namely -lsfftw or -ldfftw.

The FFT_INC variable also allows for a -DFFT_SINGLE setting that will use single-precision FFTs with PPPM, which can speed-up long-range calulations, particularly in parallel or on GPUs. Fourier transform and related PPPM operations are somewhat insensitive to floating point truncation errors and thus do not always need to be performed in double precision. Using the -DFFT_SINGLE setting trades off a little accuracy for reduced memory use and parallel communication costs for transposing 3d FFT data. Note that single precision FFTs have only been tested with the FFTW3, FFTW2, MKL, and KISS FFT options.

Step 7

The 3 JPG variables allow you to specify a JPEG and/or PNG library which LAMMPS uses when writing out JPEG or PNG files via the [dump image](#) command. These can be left blank if you do not use the -DLAMMPS_JPEG or -DLAMMPS_PNG switches discussed above in Step 4, since in that case JPEG/PNG output will be disabled.

A standard JPEG library usually goes by the name libjpeg.a or libjpeg.so and has an associated header file jpeglib.h. Whichever JPEG library you have on your platform, you'll need to set the appropriate JPG_INC, JPG_PATH, and JPG_LIB variables, so that the compiler and linker can find it.

A standard PNG library usually goes by the name libpng.a or libpng.so and has an associated header file png.h. Whichever PNG library you have on your platform, you'll need to set the appropriate JPG_INC, JPG_PATH, and JPG_LIB variables, so that the compiler and linker can find it.

As before, if these header and library files are in the usual place on your machine, you may not need to set these variables.

Step 8

Note that by default only a few of LAMMPS optional packages are installed. To build LAMMPS with optional packages, see this section below, before proceeding to Step 9.

Step 9

That's it. Once you have a correct Makefile.foo, you have installed the optional LAMMPS packages you want to include in your build, and you have pre-built any other needed libraries (e.g. MPI, FFT, package libraries), all you need to do from the src directory is type something like this:

```
make foo
or
gmake foo
```

You should get the executable lmp_foo when the build is complete.

Errors that can occur when making LAMMPS:

IMPORTANT NOTE: If an error occurs when building LAMMPS, the compiler or linker will state very explicitly what the problem is. The error message should give you a hint as to which of the steps above has failed, and what you need to do in order to fix it. Building a code with a Makefile is a very logical process. The compiler and linker need to find the appropriate files and those files need to be compatible with LAMMPS source files. When a make fails, there is usually a very simple reason, which you or a local expert will need to fix.

Here are two non-obvious errors that can occur:

(1) If the make command breaks immediately with errors that indicate it can't find files with a "*" in their names, this can be because your machine's native make doesn't support wildcard expansion in a makefile. Try gmake instead of make. If that doesn't work, try using a -f switch with your make command to use a pre-generated Makefile.list which explicitly lists all the needed files, e.g.

```
make makelist
make -f Makefile.list linux
gmake -f Makefile.list mac
```

The first "make" command will create a current Makefile.list with all the file names in your src dir. The 2nd "make" command (make or gmake) will use it to build LAMMPS. Note that you should include/exclude any desired optional packages before using the "make makelist" command.

(2) If you get an error that says something like 'identifier "atoll" is undefined', then your machine does not support "long long" integers. Try using the -DLAMMPS_LONGLONG_TO_LONG setting described above in Step 4.

Additional build tips:

(1) Building LAMMPS for multiple platforms.

You can make LAMMPS for multiple platforms from the same src directory. Each target creates its own object sub-directory called Obj_target where it stores the system-specific *.o files.

(2) Cleaning up.

Typing "make clean-all" or "make clean-machine" will delete *.o object files created when LAMMPS is built, for either all builds or for a particular machine.

(3) Changing the LAMMPS size limits via -DLAMMPS_SMALLBIG or -DLAMMPS_BIBIG or -DLAMMPS_SMALLSMALL

As explained above, any of these 3 settings can be specified on the LMP_INC line in your low-level src/MAKE/Makefile.foo.

The default is -DLAMMPS_SMALLBIG which allows for systems with up to 2^{63} atoms and timesteps (about 9 billion billion). The atom limit is for atomic systems that do not require atom IDs. For molecular models, which require atom IDs, the limit is 2^{31} atoms (about 2 billion). With this setting, image flags are stored in 32-bit integers, which means for 3 dimensions that atoms can only wrap around a periodic box at most 512 times. If atoms move through the periodic box more than this limit, the image flags will "roll over", e.g. from 511 to -512, which can cause diagnostics like the mean-squared displacement, as calculated by the compute msd command, to be faulty.

To allow for larger molecular systems or larger image flags, compile with -DLAMMPS_BIGBIG. This enables molecular systems with up to 2^{63} atoms (about 9 billion billion). And image flags will not "roll over" until they reach $2^{20} = 1048576$.

IMPORTANT NOTE: As of 6/2012, the BIGBIG setting does not yet enable molecular systems to grow as large as 2^{63}. Only the image flag roll over is currently affected by this compile option.

If your system does not support 8-byte integers, you will need to compile with the -DLAMMPS_SMALLSMALL setting. This will restrict your total number of atoms (for atomic or molecular models) and timesteps to 2^{31} (about 2 billion). Image flags will roll over at $2^9 = 512$.

Note that in src/lmptype.h there are also settings for the MPI data types associated with the integers that store atom IDs and total system sizes. These need to be consistent with the associated C data types, or else LAMMPS will generate a run-time error.

In all cases, the size of problem that can be run on a per-processor basis is limited by 4-byte integer storage to 2^{31} atoms per processor (about 2 billion). This should not normally be a restriction since such a problem would have a huge per-processor memory footprint due to neighbor lists and would run very slowly in terms of CPU secs/timestep.

Building for a Mac:

OS X is BSD Unix, so it should just work. See the src/MAKE/Makefile.mac file.

Building for Windows:

The LAMMPS download page has an option to download both a serial and parallel pre-built Windows exeutable. See the [Running LAMMPS](#) section for instructions for running these executables on a Windows box.

The pre-built executables are built with a subset of the available pacakges; see the download page for the list. If you want a Windows version with specific packages included and excluded, you can build it yourself.

One way to do this is install and use cygwin to build LAMMPS with a standard Linus make, just as you would on any Linux box; see src/MAKE/Makefile.cygwin.

The other way to do this is using Visual Studio and project files. See the src/WINDOWS directory and its README.txt file for instructions on both a basic build and a customized build with pacakges you select.

2.3 Making LAMMPS with optional packages

This section has the following sub-sections:

- [Package basics](#)
- [Including/excluding packages](#)
- [Packages that require extra libraries](#)
- [Additional Makefile settings for extra libraries](#)

Package basics:

The source code for LAMMPS is structured as a set of core files which are always included, plus optional packages. Packages are groups of files that enable a specific set of features. For example, force fields for molecular systems or granular systems are in packages. You can see the list of all packages by typing "make package" from within the src directory of the LAMMPS distribution.

If you use a command in a LAMMPS input script that is specific to a particular package, you must have built LAMMPS with that package, else you will get an error that the style is invalid or the command is unknown. Every command's doc page specfies if it is part of a package. You can also type

```
lmp_machine -h
```

to run your executable with the optional [-h command-line switch](#) for "help", which will list the styles and commands known to your executable.

There are two kinds of packages in LAMMPS, standard and user packages. More information about the contents of standard and user packages is given in [Section_packages](#) of the manual. The difference between standard and user packages is as follows:

Standard packages are supported by the LAMMPS developers and are written in a syntax and style consistent with the rest of LAMMPS. This means we will answer questions about them, debug and fix them if necessary, and keep them compatible with future changes to LAMMPS.

User packages have been contributed by users, and always begin with the user prefix. If they are a single command (single file), they are typically in the user-misc package. Otherwise, they are a a set of files grouped together which add a specific functionality to the code.

User packages don't necessarily meet the requirements of the standard packages. If you have problems using a feature provided in a user package, you will likely need to contact the contributor directly to get help. Information on how to submit additions you make to LAMMPS as a user-contributed package is given in [this section](#) of the documentation.

Some packages (both standard and user) require additional libraries. See more details below.

Including/excluding packages:

To use or not use a package you must include or exclude it before building LAMMPS. From the src directory, this is typically as simple as:

```
make yes-colloid
make g++
```

or

```
make no-manybody
make g++
```

IMPORTANT NOTE: You should NOT include/exclude packages and build LAMMPS in a single make command by using multiple targets, e.g. make yes-colloid g++. This is because the make procedure creates a list of source files that will be out-of-date for the build if the package configuration changes during the same command.

Some packages have individual files that depend on other packages being included. LAMMPS checks for this and does the right thing. I.e. individual files are only included if their dependencies are already included. Likewise, if a package is excluded, other files dependent on that package are also excluded.

The reason to exclude packages is if you will never run certain kinds of simulations. For some packages, this will keep you from having to build auxiliary libraries (see below), and will also produce a smaller executable which may run a bit faster.

When you download a LAMMPS tarball, these packages are pre-installed in the src directory: KSPACE, MANYBODY,MOLECULE. When you download LAMMPS source files from the SVN or Git repositories, no packages are pre-installed.

Packages are included or excluded by typing "make yes-name" or "make no-name", where "name" is the name of the package in lower-case, e.g. name = kspace for the KSPACE package or name = user-atc for the USER-ATC package. You can also type "make yes-standard", "make no-standard", "make yes-user", "make no-user", "make yes-all" or "make no-all" to include/exclude various sets of packages. Type "make package" to see the all of the package-related make options.

IMPORTANT NOTE: Inclusion/exclusion of a package works by simply moving files back and forth between the main src directory and sub-directories with the package name (e.g. src/KSPACE, src/USER-ATC), so that the files are seen or not seen when LAMMPS is built. After you have included or excluded a package, you must re-build LAMMPS.

Additional package-related make options exist to help manage LAMMPS files that exist in both the src directory and in package sub-directories. You do not normally need to use these commands unless you are editing LAMMPS files or have downloaded a patch from the LAMMPS WWW site.

Typing "make package-update" will overwrite src files with files from the package sub-directories if the package has been included. It should be used after a patch is installed, since patches only update the files in the package sub-directory, but not the src files. Typing "make package-overwrite" will overwrite files in the package sub-directories with src files.

Typing "make package-status" will show which packages are currently included. Of those that are included, it will list files that are different in the src directory and package sub-directory. Typing "make package-diff" lists all differences between these files. Again, type "make package" to see all of the package-related make

options.

Packages that require extra libraries:

A few of the standard and user packages require additional auxiliary libraries. They must be compiled first, before LAMMPS is built. If you get a LAMMPS build error about a missing library, this is likely the reason. See the Section_packages doc page for a list of packages that have auxiliary libraries.

Code for some of these auxiliary libraries is included in the LAMMPS distribution under the lib directory. Examples are the USER-ATC and MEAM packages. Some auxiliary libraries are not included with LAMMPS; to use the associated package you must download and install the auxiliary library yourself. Examples are the KIM and VORONOI and USER-MOLFILE packages.

For libraries with provided source code, each lib directory has a README file (e.g. lib/reax/README) with instructions on how to build that library. Typically this is done by typing something like:

```
make -f Makefile.g++
```

If one of the provided Makefiles is not appropriate for your system you will need to edit or add one. Note that all the Makefiles have a setting for EXTRAMAKE at the top that names a Makefile.lammps.* file.

If successful, this will produce 2 files in the lib directory:

```
libpackage.a
Makefile.lammps
```

The Makefile.lammps file is a copy of the EXTRAMAKE file specified in the Makefile you used.

You MUST insure that the settings in Makefile.lammps are appropriate for your system. If they are not, the LAMMPS build will fail.

As explained in the lib/package/README files, they are used to specify additional system libraries and their locations so that LAMMPS can build with the auxiliary library. For example, if the MEAM or REAX packages are used, the auxiliary libraries consist of F90 code, build with a F90 complier. To link that library with LAMMPS (a C++ code) via whatever C++ compiler LAMMPS is built with, typically requires additional Fortran-to-C libraries be included in the link. Another example are the BLAS and LAPACK libraries needed to use the USER-ATC or USER-AWPMD packages.

For libraries without provided source code, see the src/package/Makefile.lammps file for information on where to find the library and how to build it. E.g. the file src/KIM/Makefile.lammps. This file serves the same purpose as the lib/package/Makefile.lammps file described above. It has settings needed when LAMMPS is built to link with the auxiliary library. Again, you MUST insure that the settings in src/package/Makefile.lammps are appropriate for your system and where you installed the auxiliary library. If they are not, the LAMMPS build will fail.

2.4 Building LAMMPS via the Make.py script

The src directory includes a Make.py script, written in Python, which can be used to automate various steps of the build process.

You can run the script from the src directory by typing either:

```
Make.py
python Make.py
```

which will give you info about the tool. For the former to work, you may need to edit the 1st line of the script to point to your local Python. And you may need to insure the script is executable:

```
chmod +x Make.py
```

The following options are supported as switches:

- -i file1 file2 ...
- -p package1 package2 ...
- -u package1 package2 ...
- -e package1 arg1 arg2 package2 ...
- -o dir
- -b machine
- -s suffix1 suffix2 ...
- -l dir
- -j N
- -h switch1 switch2 ...

Help on any switch can be listed by using -h, e.g.

```
Make.py -h -i -p
```

At a hi-level, these are the kinds of package management and build tasks that can be performed easily, using the Make.py tool:

- install/uninstall packages and build the associated external libs (use -p and -u and -e)
- install packages needed for one or more input scripts (use -i and -p)
- build LAMMPS, either in the src dir or new dir (use -b)
- create a new dir with only the source code needed for one or more input scripts (use -i and -o)

The last bullet can be useful when you wish to build a stripped-down version of LAMMPS to run a specific script(s). Or when you wish to move the minimal amount of files to another platform for a remote LAMMPS build.

Note that using Make.py is not a substitute for insuring you have a valid src/MAKE/Makefile.foo for your system, or that external library Makefiles in any lib/* directories you use are also valid for your system. But once you have done that, you can use Make.py to quickly include/exclude the packages and external libraries needed by your input scripts.

2.5 Building LAMMPS as a library

LAMMPS can be built as either a static or shared library, which can then be called from another application or a scripting language. See this section for more info on coupling LAMMPS to other codes. See this section for more info on wrapping and running LAMMPS from Python.

Static library:

To build LAMMPS as a static library (*.a file on Linux), type

```
make makelib
make -f Makefile.lib foo
```

where foo is the machine name. This kind of library is typically used to statically link a driver application to LAMMPS, so that you can insure all dependencies are satisfied at compile time. Note that inclusion or exclusion of any desired optional packages should be done before typing "make makelib". The first "make"

command will create a current Makefile.lib with all the file names in your src dir. The second "make" command will use it to build LAMMPS as a static library, using the ARCHIVE and ARFLAGS settings in src/MAKE/Makefile.foo. The build will create the file liblammps_foo.a which another application can link to.

Shared library:

To build LAMMPS as a shared library (*.so file on Linux), which can be dynamically loaded, e.g. from Python, type

```
make makeshlib
make -f Makefile.shlib foo
```

where foo is the machine name. This kind of library is required when wrapping LAMMPS with Python; see [Section_python](#) for details. Again, note that inclusion or exclusion of any desired optional packages should be done before typing "make makelib". The first "make" command will create a current Makefile.shlib with all the file names in your src dir. The second "make" command will use it to build LAMMPS as a shared library, using the SHFLAGS and SHLIBFLAGS settings in src/MAKE/Makefile.foo. The build will create the file liblammps_foo.so which another application can link to dyamically. It will also create a soft link liblammps.so, which the Python wrapper uses by default.

Note that for a shared library to be usable by a calling program, all the auxiliary libraries it depends on must also exist as shared libraries. This will be the case for libraries included with LAMMPS, such as the dummy MPI library in src/STUBS or any package libraries in lib/packges, since they are always built as shared libraries with the -fPIC switch. However, if a library like MPI or FFTW does not exist as a shared library, the second make command will generate an error. This means you will need to install a shared library version of the package. The build instructions for the library should tell you how to do this.

As an example, here is how to build and install the [MPICH library](#), a popular open-source version of MPI, distributed by Argonne National Labs, as a shared library in the default /usr/local/lib location:

```
./configure --enable-shared
make
make install
```

You may need to use "sudo make install" in place of the last line if you do not have write privileges for /usr/local/lib. The end result should be the file /usr/local/lib/libmpich.so.

Additional requirement for using a shared library:

The operating system finds shared libraries to load at run-time using the environment variable LD_LIBRARY_PATH. So you may wish to copy the file src/liblammps.so or src/liblammps_g++.so (for example) to a place the system can find it by default, such as /usr/local/lib, or you may wish to add the LAMMPS src directory to LD_LIBRARY_PATH, so that the current version of the shared library is always available to programs that use it.

For the csh or tcsh shells, you would add something like this to your ~/.cshrc file:

```
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:/home/sjplimp/lammps/src
```

Calling the LAMMPS library:

Either flavor of library (static or shared0 allows one or more LAMMPS objects to be instantiated from the calling program.

When used from a C++ program, all of LAMMPS is wrapped in a LAMMPS_NS namespace; you can safely use any of its classes and methods from within the calling code, as needed.

Static library:

When used from a C or Fortran program or a scripting language like Python, the library has a simple function-style interface, provided in src/library.cpp and src/library.h.

See the sample codes in examples/COUPLE/simple for examples of C++ and C and Fortran codes that invoke LAMMPS thru its library interface. There are other examples as well in the COUPLE directory which are discussed in Section_howto 10 of the manual. See Section_python of the manual for a description of the Python wrapper provided with LAMMPS that operates through the LAMMPS library interface.

The files src/library.cpp and library.h define the C-style API for using LAMMPS as a library. See Section_howto 19 of the manual for a description of the interface and how to extend it for your needs.

2.6 Running LAMMPS

By default, LAMMPS runs by reading commands from stdin; e.g. lmp_linux < in.file. This means you first create an input script (e.g. in.file) containing the desired commands. This section describes how input scripts are structured and what commands they contain.

You can test LAMMPS on any of the sample inputs provided in the examples or bench directory. Input scripts are named in.* and sample outputs are named log.*.name.P where name is a machine and P is the number of processors it was run on.

Here is how you might run a standard Lennard-Jones benchmark on a Linux box, using mpirun to launch a parallel job:

```
cd src
make linux
cp lmp_linux ../bench
cd ../bench
mpirun -np 4 lmp_linux <in.lj
```

See this page for timings for this and the other benchmarks on various platforms.

On a Windows box, you can skip making LAMMPS and simply download an executable, as described above, though the pre-packaged executables include only certain packages.

To run a LAMMPS executable on a Windows machine, first decide whether you want to download the non-MPI (serial) or the MPI (parallel) version of the executable. Download and save the version you have chosen.

For the non-MPI version, follow these steps:

- Get a command prompt by going to Start->Run... , then typing "cmd".
- Move to the directory where you have saved lmp_win_no-mpi.exe (e.g. by typing: cd "Documents").
- At the command prompt, type "lmp_win_no-mpi -in in.lj", replacing in.lj with the name of your LAMMPS input script.

For the MPI version, which allows you to run LAMMPS under Windows on multiple processors, follow these steps:

- Download and install MPICH2 for Windows.
- You'll need to use the mpiexec.exe and smpd.exe files from the MPICH2 package. Put them in same directory (or path) as the LAMMPS Windows executable.
- Get a command prompt by going to Start->Run... , then typing "cmd".
- Move to the directory where you have saved lmp_win_mpi.exe (e.g. by typing: cd "Documents").

- Then type something like this: "mpiexec -np 4 -localonly lmp_win_mpi -in in.lj", replacing in.lj with the name of your LAMMPS input script.
- Note that you may need to provide smpd with a passphrase --- it doesn't matter what you type.
- In this mode, output may not immediately show up on the screen, so if your input script takes a long time to execute, you may need to be patient before the output shows up.
- Alternatively, you can still use this executable to run on a single processor by typing something like: "lmp_win_mpi -in in.lj".

The screen output from LAMMPS is described in the next section. As it runs, LAMMPS also writes a log.lammps file with the same information.

Note that this sequence of commands copies the LAMMPS executable (lmp_linux) to the directory with the input files. This may not be necessary, but some versions of MPI reset the working directory to where the executable is, rather than leave it as the directory where you launch mpirun from (if you launch lmp_linux on its own and not under mpirun). If that happens, LAMMPS will look for additional input files and write its output files to the executable directory, rather than your working directory, which is probably not what you want.

If LAMMPS encounters errors in the input script or while running a simulation it will print an ERROR message and stop or a WARNING message and continue. See [Section_errors](#) for a discussion of the various kinds of errors LAMMPS can or can't detect, a list of all ERROR and WARNING messages, and what to do about them.

LAMMPS can run a problem on any number of processors, including a single processor. In theory you should get identical answers on any number of processors and on any machine. In practice, numerical round-off can cause slight differences and eventual divergence of molecular dynamics phase space trajectories.

LAMMPS can run as large a problem as will fit in the physical memory of one or more processors. If you run out of memory, you must run on more processors or setup a smaller problem.

2.7 Command-line options

At run time, LAMMPS recognizes several optional command-line switches which may be used in any order. Either the full word or a one-or-two letter abbreviation can be used:

- -c or -cuda
- -e or -echo
- -i or -in
- -h or -help
- -l or -log
- -nc or -nocite
- -p or -partition
- -pl or -plog
- -ps or -pscreen
- -r or -restart
- -ro or -reorder
- -sc or -screen
- -sf or -suffix
- -v or -var

For example, lmp_ibm might be launched as follows:

```
mpirun -np 16 lmp_ibm -v f tmp.out -l my.log -sc none <in.alloy
mpirun -np 16 lmp_ibm -var f tmp.out -log my.log -screen none  <in.alloy
```

Here are the details on the options:

```
-cuda on/off
```

Explicitly enable or disable CUDA support, as provided by the USER-CUDA package. If LAMMPS is built with this package, as described above in <u>Section 2.3</u>, then by default LAMMPS will run in CUDA mode. If this switch is set to "off", then it will not, even if it was built with the USER-CUDA package, which means you can run standard LAMMPS or with the GPU package for testing or benchmarking purposes. The only reason to set the switch to "on", is to check if LAMMPS was built with the USER-CUDA package, since an error will be generated if it was not.

```
-echo style
```

Set the style of command echoing. The style can be *none* or *screen* or *log* or *both*. Depending on the style, each command read from the input script will be echoed to the screen and/or logfile. This can be useful to figure out which line of your script is causing an input error. The default value is *log*. The echo style can also be set by using the <u>echo</u> command in the input script itself.

```
-in file
```

Specify a file to use as an input script. This is an optional switch when running LAMMPS in one-partition mode. If it is not specified, LAMMPS reads its input script from stdin - e.g. lmp_linux < in.run. This is a required switch when running LAMMPS in multi-partition mode, since multiple processors cannot all read from stdin.

```
-help
```

Print a list of options compiled into this executable for each LAMMPS style (atom_style, fix, compute, pair_style, bond_style, etc). This can help you know if the command you want to use was included via the appropriate package. LAMMPS will print the info and immediately exit if this switch is used.

```
-log file
```

Specify a log file for LAMMPS to write status information to. In one-partition mode, if the switch is not used, LAMMPS writes to the file log.lammps. If this switch is used, LAMMPS writes to the specified file. In multi-partition mode, if the switch is not used, a log.lammps file is created with hi-level status information. Each partition also writes to a log.lammps.N file where N is the partition ID. If the switch is specified in multi-partition mode, the hi-level logfile is named "file" and each partition also logs information to a file.N. For both one-partition and multi-partition mode, if the specified file is "none", then no log files are created. Using a <u>log</u> command in the input script will override this setting. Option -plog will override the name of the partition log files file.N.

```
-nocite
```

Disable writing the log.cite file which is normally written to list references for specific cite-able features used during a LAMMPS run. See the <u>citation page</u> for more details.

```
-partition 8x2 4 5 ...
```

Invoke LAMMPS in multi-partition mode. When LAMMPS is run on P processors and this switch is not used, LAMMPS runs in one partition, i.e. all P processors run a single simulation. If this switch is used, the P processors are split into separate partitions and each partition runs its own simulation. The arguments to the switch specify the number of processors in each partition. Arguments of the form MxN mean M partitions, each with N processors. Arguments of the form N mean a single partition with N processors. The sum of processors in all partitions must equal P. Thus the command "-partition 8x2 4 5" has 10 partitions and runs on a total of 25 processors.

Running with multiple partitions can e useful for running [multi-replica simulations,](#) where each replica runs on on one or a few processors. Note that with MPI installed on a machine (e.g. your desktop), you can run on more (virtual) processors than you have physical processors.

To run multiple independent simulatoins from one input script, using multiple partitions, see [Section_howto 4](#) of the manual. World- and universe-style [variables](#) are useful in this context.

```
-plog file
```

Specify the base name for the partition log files, so partition N writes log information to file.N. If file is none, then no partition log files are created. This overrides the filename specified in the -log command-line option. This option is useful when working with large numbers of partitions, allowing the partition log files to be suppressed (-plog none) or placed in a sub-directory (-plog replica_files/log.lammps) If this option is not used the log file for partition N is log.lammps.N or whatever is specified by the -log command-line option.

```
-pscreen file
```

Specify the base name for the partition screen file, so partition N writes screen information to file.N. If file is none, then no partition screen files are created. This overrides the filename specified in the -screen command-line option. This option is useful when working with large numbers of partitions, allowing the partition screen files to be suppressed (-pscreen none) or placed in a sub-directory (-pscreen replica_files/screen). If this option is not used the screen file for partition N is screen.N or whatever is specified by the -screen command-line option.

```
-restart restartfile datafile
```

Convert the restart file into a data file and immediately exit. This is the same operation as if the following 2-line input script were run:

```
read_restart restartfile
write_data datafile
```

Note that the specified restartfile and datafile can have wild-card characters ("*",%") as described by the [read_restart](#) and [write_data](#) commands. But a filename such as file.* will need to be enclosed in quotes to avoid shell expansion of the "*" character.

```
-reorder nth N
-reorder custom filename
```

Reorder the processors in the MPI communicator used to instantiate LAMMPS, in one of several ways. The original MPI communicator ranks all P processors from 0 to P-1. The mapping of these ranks to physical processors is done by MPI before LAMMPS begins. It may be useful in some cases to alter the rank order. E.g. to insure that cores within each node are ranked in a desired order. Or when using the [run_style](#) [verlet/split](#) command with 2 partitions to insure that a specific Kspace processor (in the 2nd partition) is matched up with a specific set of processors in the 1st partition. See the [Section_accelerate](#) doc pages for more details.

If the keyword *nth* is used with a setting *N*, then it means every Nth processor will be moved to the end of the ranking. This is useful when using the [run_style verlet/split](#) command with 2 partitions via the -partition command-line switch. The first set of processors will be in the first partition, the 2nd set in the 2nd partition. The -reorder command-line switch can alter this so that the 1st N procs in the 1st partition and one proc in the 2nd partition will be ordered consecutively, e.g. as the cores on one physical node. This can boost performance. For example, if you use "-reorder nth 4" and "-partition 9 3" and you are running on 12 processors, the processors will be reordered from

```
0 1 2 3 4 5 6 7 8 9 10 11
```

to

```
0 1 2 4 5 6 8 9 10 3 7 11
```

so that the processors in each partition will be

```
0 1 2 4 5 6 8 9 10
3 7 11
```

See the "processors" command for how to insure processors from each partition could then be grouped optimally for quad-core nodes.

If the keyword is *custom*, then a file that specifies a permutation of the processor ranks is also specified. The format of the reorder file is as follows. Any number of initial blank or comment lines (starting with a "#" character) can be present. These should be followed by P lines of the form:

```
I J
```

where P is the number of processors LAMMPS was launched with. Note that if running in multi-partition mode (see the -partition switch above) P is the total number of processors in all partitions. The I and J values describe a permutation of the P processors. Every I and J should be values from 0 to P-1 inclusive. In the set of P I values, every proc ID should appear exactly once. Ditto for the set of P J values. A single I,J pairing means that the physical processor with rank I in the original MPI communicator will have rank J in the reordered communicator.

Note that rank ordering can also be specified by many MPI implementations, either by environment variables that specify how to order physical processors, or by config files that specify what physical processors to assign to each MPI rank. The -reorder switch simply gives you a portable way to do this without relying on MPI itself. See the processors out command for how to output info on the final assignment of physical processors to the LAMMPS simulation domain.

```
-screen file
```

Specify a file for LAMMPS to write its screen information to. In one-partition mode, if the switch is not used, LAMMPS writes to the screen. If this switch is used, LAMMPS writes to the specified file instead and you will see no screen output. In multi-partition mode, if the switch is not used, hi-level status information is written to the screen. Each partition also writes to a screen.N file where N is the partition ID. If the switch is specified in multi-partition mode, the hi-level screen dump is named "file" and each partition also writes screen information to a file.N. For both one-partition and multi-partition mode, if the specified file is "none", then no screen output is performed. Option -pscreen will override the name of the partition screen files file.N.

```
-suffix style
```

Use variants of various styles if they exist. The specified style can be *opt*, *omp*, *gpu*, or *cuda*. These refer to optional packages that LAMMPS can be built with, as described above in Section 2.3. The "opt" style corrsponds to the OPT package, the "omp" style to the USER-OMP package, the "gpu" style to the GPU package, and the "cuda" style to the USER-CUDA package.

As an example, all of the packages provide a pair_style lj/cut variant, with style names lj/cut/opt, lj/cut/omp, lj/cut/gpu, or lj/cut/cuda. A variant styles can be specified explicitly in your input script, e.g. pair_style lj/cut/gpu. If the -suffix switch is used, you do not need to modify your input script. The specified suffix (opt,omp,gpu,cuda) is automatically appended whenever your input script command creates a new atom, pair, fix, compute, or run style. If the variant version does not exist, the standard version is created.

For the GPU package, using this command-line switch also invokes the default GPU settings, as if the command "package gpu force/neigh 0 0 1" were used at the top of your input script. These settings can be

changed by using the [package gpu](#) command in your script if desired.

For the OMP package, using this command-line switch also invokes the default OMP settings, as if the command "package omp *" were used at the top of your input script. These settings can be changed by using the [package omp](#) command in your script if desired.

The [suffix](#) command can also set a suffix and it can also turn off/on any suffix setting made via the command line.

```
-var name value1 value2 ...
```

Specify a variable that will be defined for substitution purposes when the input script is read. "Name" is the variable name which can be a single character (referenced as $x in the input script) or a full string (referenced as ${abc}). An [index-style variable](#) will be created and populated with the subsequent values, e.g. a set of filenames. Using this command-line option is equivalent to putting the line "variable name index value1 value2 ..." at the beginning of the input script. Defining an index variable as a command-line argument overrides any setting for the same index variable in the input script, since index variables cannot be re-defined. See the [variable](#) command for more info on defining index and other kinds of variables and [this section](#) for more info on using variables in input scripts.

NOTE: Currently, the command-line parser looks for arguments that start with "-" to indicate new switches. Thus you cannot specify multiple variable values if any of they start with a "-", e.g. a negative numeric value. It is OK if the first value1 starts with a "-", since it is automatically skipped.

2.8 LAMMPS screen output

As LAMMPS reads an input script, it prints information to both the screen and a log file about significant actions it takes to setup a simulation. When the simulation is ready to begin, LAMMPS performs various initializations and prints the amount of memory (in MBytes per processor) that the simulation requires. It also prints details of the initial thermodynamic state of the system. During the run itself, thermodynamic information is printed periodically, every few timesteps. When the run concludes, LAMMPS prints the final thermodynamic state and a total run time for the simulation. It then appends statistics about the CPU time and storage requirements for the simulation. An example set of statistics is shown here:

```
Loop time of 49.002 on 2 procs for 2004 atoms

Pair   time (%) = 35.0495 (71.5267)
Bond   time (%) = 0.092046 (0.187841)
Kspce  time (%) = 6.42073 (13.103)
Neigh  time (%) = 2.73485 (5.5811)
Comm   time (%) = 1.50291 (3.06703)
Outpt  time (%) = 0.013799 (0.0281601)
Other  time (%) = 2.13669 (4.36041)

Nlocal:    1002 ave, 1015 max, 989 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Nghost:    8720 ave, 8724 max, 8716 min
Histogram: 1 0 0 0 0 0 0 0 0 1
Neighs:    354141 ave, 361422 max, 346860 min
Histogram: 1 0 0 0 0 0 0 0 0 1

Total # of neighbors = 708282
Ave neighs/atom = 353.434
Ave special neighs/atom = 2.34032
Number of reneighborings = 42
Dangerous reneighborings = 2
```

The first section gives the breakdown of the CPU run time (in seconds) into major categories. The second section lists the number of owned atoms (Nlocal), ghost atoms (Nghost), and pair-wise neighbors stored per processor. The max and min values give the spread of these values across processors with a 10-bin histogram showing the distribution. The total number of histogram counts is equal to the number of processors.

The last section gives aggregate statistics for pair-wise neighbors and special neighbors that LAMMPS keeps track of (see the special_bonds command). The number of times neighbor lists were rebuilt during the run is given as well as the number of potentially "dangerous" rebuilds. If atom movement triggered neighbor list rebuilding (see the neigh_modify command), then dangerous reneighborings are those that were triggered on the first timestep atom movement was checked for. If this count is non-zero you may wish to reduce the delay factor to insure no force interactions are missed by atoms moving beyond the neighbor skin distance before a rebuild takes place.

If an energy minimization was performed via the minimize command, additional information is printed, e.g.

```
Minimization stats:
  E initial, next-to-last, final = -0.895962 -2.94193 -2.94342
  Gradient 2-norm init/final= 1920.78 20.9992
  Gradient inf-norm init/final= 304.283 9.61216
  Iterations = 36
  Force evaluations = 177
```

The first line lists the initial and final energy, as well as the energy on the next-to-last iteration. The next 2 lines give a measure of the gradient of the energy (force on all atoms). The 2-norm is the "length" of this force vector; the inf-norm is the largest component. The last 2 lines are statistics on how many iterations and force-evaluations the minimizer required. Multiple force evaluations are typically done at each iteration to perform a 1d line minimization in the search direction.

If a kspace_style long-range Coulombics solve was performed during the run (PPPM, Ewald), then additional information is printed, e.g.

```
FFT time (% of Kspce) = 0.200313 (8.34477)
FFT Gflps 3d 1d-only = 2.31074 9.19989
```

The first line gives the time spent doing 3d FFTs (4 per timestep) and the fraction it represents of the total KSpace time (listed above). Each 3d FFT requires computation (3 sets of 1d FFTs) and communication (transposes). The total flops performed is $5N\log_2(N)$, where N is the number of points in the 3d grid. The FFTs are timed with and without the communication and a Gflop rate is computed. The 3d rate is with communication; the 1d rate is without (just the 1d FFTs). Thus you can estimate what fraction of your FFT time was spent in communication, roughly 75% in the example above.

2.9 Tips for users of previous LAMMPS versions

The current C++ began with a complete rewrite of LAMMPS 2001, which was written in F90. Features of earlier versions of LAMMPS are listed in Section_history. The F90 and F77 versions (2001 and 99) are also freely distributed as open-source codes; check the LAMMPS WWW Site for distribution information if you prefer those versions. The 99 and 2001 versions are no longer under active development; they do not have all the features of C++ LAMMPS.

If you are a previous user of LAMMPS 2001, these are the most significant changes you will notice in C++ LAMMPS:

(1) The names and arguments of many input script commands have changed. All commands are now a single word (e.g. read_data instead of read data).

(2) All the functionality of LAMMPS 2001 is included in C++ LAMMPS, but you may need to specify the relevant commands in different ways.

(3) The format of the data file can be streamlined for some problems. See the read_data command for details. The data file section "Nonbond Coeff" has been renamed to "Pair Coeff" in C++ LAMMPS.

(4) Binary restart files written by LAMMPS 2001 cannot be read by C++ LAMMPS with a read_restart command. This is because they were output by F90 which writes in a different binary format than C or C++ writes or reads. Use the *restart2data* tool provided with LAMMPS 2001 to convert the 2001 restart file to a text data file. Then edit the data file as necessary before using the C++ LAMMPS read_data command to read it in.

(5) There are numerous small numerical changes in C++ LAMMPS that mean you will not get identical answers when comparing to a 2001 run. However, your initial thermodynamic energy and MD trajectory should be close if you have setup the problem for both codes the same.

2.9 Tips for users of previous LAMMPS versions   1212

9. Additional tools

LAMMPS is designed to be a computational kernel for performing molecular dynamics computations. Additional pre- and post-processing steps are often necessary to setup and analyze a simulation. A few additional tools are provided with the LAMMPS distribution and are described in this section.

Our group has also written and released a separate toolkit called Pizza.py which provides tools for doing setup, analysis, plotting, and visualization for LAMMPS simulations. Pizza.py is written in Python and is available for download from the Pizza.py WWW site.

Note that many users write their own setup or analysis tools or use other existing codes and convert their output to a LAMMPS input format or vice versa. The tools listed here are included in the LAMMPS distribution as examples of auxiliary tools. Some of them are not actively supported by Sandia, as they were contributed by LAMMPS users. If you have problems using them, we can direct you to the authors.

The source code for each of these codes is in the tools sub-directory of the LAMMPS distribution. There is a Makefile (which you may need to edit for your platform) which will build several of the tools which reside in that directory. Some of them are larger packages in their own sub-directories with their own Makefiles.

- amber2lmp
- binary2txt
- ch2lmp
- chain
- createatoms
- data2xmovie
- eam database
- eam generate
- eff
- emacs
- ipp
- kate
- lmp2arc
- lmp2cfg
- lmp2vmd
- matlab
- micelle2d
- moltemplate
- msi2lmp
- phonon
- polymer bonding
- pymol_asphere
- python
- reax
- restart2data
- vim
- xmgrace
- xmovie

amber2lmp tool

The amber2lmp sub-directory contains two Python scripts for converting files back-and-forth between the AMBER MD code and LAMMPS. See the README file in amber2lmp for more information.

These tools were written by Keir Novik while he was at Queen Mary University of London. Keir is no longer there and cannot support these tools which are out-of-date with respect to the current LAMMPS version (and maybe with respect to AMBER as well). Since we don't use these tools at Sandia, you'll need to experiment with them and make necessary modifications yourself.

---

binary2txt tool

The file binary2txt.cpp converts one or more binary LAMMPS dump file into ASCII text files. The syntax for running the tool is

```
binary2txt file1 file2 ...
```

which creates file1.txt, file2.txt, etc. This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

---

ch2lmp tool

The ch2lmp sub-directory contains tools for converting files back-and-forth between the CHARMM MD code and LAMMPS.

They are intended to make it easy to use CHARMM as a builder and as a post-processor for LAMMPS. Using charmm2lammps.pl, you can convert an ensemble built in CHARMM into its LAMMPS equivalent. Using lammps2pdb.pl you can convert LAMMPS atom dumps into pdb files.

See the README file in the ch2lmp sub-directory for more information.

These tools were created by Pieter in't Veld (pjintve at sandia.gov) and Paul Crozier (pscrozi at sandia.gov) at Sandia.

---

chain tool

The file chain.f creates a LAMMPS data file containing bead-spring polymer chains and/or monomer solvent atoms. It uses a text file containing chain definition parameters as an input. The created chains and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un-overlap it. The syntax for running the tool is

```
chain <def.chain > data.file
```

See the def.chain or def.chain.ab files in the tools directory for examples of definition files. This tool was used to create the system for the chain benchmark.

---

createatoms tool

The tools/createatoms directory contains a Fortran program called createAtoms.f which can generate a variety of interesting crystal structures and geometries and output the resulting list of atom coordinates in LAMMPS or other formats.

See the included Manual.pdf for details.

The tool is authored by Xiaowang Zhou (Sandia), xzhou at sandia.gov.

---

data2xmovie tool

The file data2xmovie.c converts a LAMMPS data file into a snapshot suitable for visualizing with the [xmovie](#) tool, as if it had been output with a dump command from LAMMPS itself. The syntax for running the tool is

```
data2xmovie options <infile > outfile
```

See the top of the data2xmovie.c file for a discussion of the options.

---

eam database tool

The tools/eam_database directory contains a Fortran program that will generate EAM alloy setfl potential files for any combination of 16 elements: Cu, Ag, Au, Ni, Pd, Pt, Al, Pb, Fe, Mo, Ta, W, Mg, Co, Ti, Zr. The files can then be used with the [pair_style eam/alloy](#) command.

The tool is authored by Xiaowang Zhou (Sandia), xzhou at sandia.gov, and is based on his paper:

X. W. Zhou, R. A. Johnson, and H. N. G. Wadley, Phys. Rev. B, 69, 144113 (2004).

---

eam generate tool

The tools/eam_generate directory contains several one-file C programs that convert an analytic formula into a tabulated [embedded atom method (EAM)](#) setfl potential file. The potentials they produce are in the potentials directory, and can be used with the [pair_style eam/alloy](#) command.

The source files and potentials were provided by Gerolf Ziegenhain (gerolf at ziegenhain.com).

---

eff tool

The tools/eff directory contains various scripts for generating structures and post-processing output for simulations using the electron force field (eFF).

These tools were provided by Andres Jaramillo-Botero at CalTech (ajaramil at wag.caltech.edu).

---

emacs tool

The tools/emacs directory contains a Lips add-on file for Emacs that enables a lammps-mode for editing of input scripts when using Emacs, with various highlighting options setup.

These tools were provided by Aidan Thompson at Sandia (athomps at sandia.gov).

---

ipp tool

The tools/ipp directory contains a Perl script ipp which can be used to facilitate the creation of a complicated file (say, a lammps input script or tools/createatoms input file) using a template file.

ipp was created and is maintained by Reese Jones (Sandia), rjones at sandia.gov.

See two examples in the tools/ipp directory. One of them is for the tools/createatoms tool's input file.

---

kate tool

The file in the tools/kate directory is an add-on to the Kate editor in the KDE suite that allow syntax highlighting of LAMMPS input scripts. See the README.txt file for details.

The file was provided by Alessandro Luigi Sellerio (alessandro.sellerio at ieni.cnr.it).

lmp2arc tool

The lmp2arc sub-directory contains a tool for converting LAMMPS output files to the format for Accelrys' Insight MD code (formerly MSI/Biosym and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec at mayo.edu), but still fields questions about the tool.

This tool was updated for the current LAMMPS C++ version by Jeff Greathouse at Sandia (jagreat at sandia.gov).

lmp2cfg tool

The lmp2cfg sub-directory contains a tool for converting LAMMPS output files into a series of *.cfg files which can be read into the [AtomEye](#) visualizer. See the README file for more information.

This tool was written by Ara Kooser at Sandia (askoose at sandia.gov).

lmp2vmd tool

The lmp2vmd sub-directory contains a README.txt file that describes details of scripts and plugin support within the [VMD package](#) for visualizing LAMMPS dump files.

The VMD plugins and other supporting scripts were written by Axel Kohlmeyer (akohlmey at cmm.chem.upenn.edu) at U Penn.

matlab tool

The matlab sub-directory contains several [MATLAB](#) scripts for post-processing LAMMPS output. The scripts include readers for log and dump files, a reader for EAM potential files, and a converter that reads LAMMPS dump files and produces CFG files that can be visualized with the [AtomEye](#) visualizer.

See the README.pdf file for more information.

These scripts were written by Arun Subramaniyan at Purdue Univ (asubrama at purdue.edu).

micelle2d tool

The file micelle2d.f creates a LAMMPS data file containing short lipid chains in a monomer solution. It uses a text file containing lipid definition parameters as an input. The created molecules and solvent atoms can strongly overlap, so LAMMPS needs to run the system initially with a "soft" pair potential to un-overlap it. The syntax for running the tool is

```
micelle2d <def.micelle2d > data.file
```

See the def.micelle2d file in the tools directory for an example of a definition file. This tool was used to create the system for the micelle example.

---

moltemplate tool

The moltemplate sub-directory contains a Python-based tool for building molecular systems based on a text-file description, and creating LAMMPS data files that encode their molecular topology as lists of bonds, angles, dihedrals, etc. See the README.TXT file for more information.

This tool was written by Andrew Jewett (jewett.aij at gmail.com), who supports it. It has its own WWW page at http://moltemplate.org.

---

msi2lmp tool

The msi2lmp sub-directory contains a tool for creating LAMMPS input data files from Accelrys' Insight MD code (formerly MSI/Biosym and its Discover MD code). See the README file for more information.

This tool was written by John Carpenter (Cray), Michael Peachey (Cray), and Steve Lustig (Dupont). John is now at the Mayo Clinic (jec at mayo.edu), but still fields questions about the tool.

This tool may be out-of-date with respect to the current LAMMPS and Insight versions. Since we don't use it at Sandia, you'll need to experiment with it yourself.

---

phonon tool

The phonon sub-directory contains a post-processing tool useful for analyzing the output of the fix phonon command in the USER-PHONON package.

See the README file for instruction on building the tool and what library it needs. And see the examples/USER/phonon directory for example problems that can be post-processed with this tool.

This tool was written by Ling-Ti Kong at Shanghai Jiao Tong University.

---

polymer bonding tool

The polybond sub-directory contains a Python-based tool useful for performing "programmable polymer bonding". The Python file lmpsdata.py provides a "Lmpsdata" class with various methods which can be invoked by a user-written Python script to create data files with complex bonding topologies.

See the Manual.pdf for details and example scripts.

This tool was written by Zachary Kraus at Georgia Tech.

---

pymol_asphere tool

The pymol_asphere sub-directory contains a tool for converting a LAMMPS dump file that contains orientation info for ellipsoidal particles into an input file for the PyMol visualization package.

Specifically, the tool triangulates the ellipsoids so they can be viewed as true ellipsoidal particles within PyMol. See the README and examples directory within pymol_asphere for more information.

This tool was written by Mike Brown at Sandia.

---

python tool

The python sub-directory contains several Python scripts that perform common LAMMPS post-processing tasks, such as:

- extract thermodynamic info from a log file as columns of numbers
- plot two columns of thermodynamic info from a log file using GnuPlot
- sort the snapshots in a dump file by atom ID
- convert multiple [NEB](#) dump files into one dump file for viz
- convert dump files into XYZ, CFG, or PDB format for viz by other packages

These are simple scripts built on [Pizza.py](#) modules. See the README for more info on Pizza.py and how to use these scripts.

reax tool

The reax sub-directory contains stand-alond codes that can post-process the output of the [fix reax/bonds](#) command from a LAMMPS simulation using [ReaxFF](#). See the README.txt file for more info.

These tools were written by Aidan Thompson at Sandia.

restart2data tool

IMPORTANT NOTE: This tool is now obsolete and is not included in the current LAMMPS distribution. This is becaues there is now a [write_data](#) command, which can create a data file from within an input script. Running LAMMPS with the "-r" [command-line switch](#) as follows:

lmp_g++ -r restartfile datafile

is the same as running a 2-line input script:

read_restart restartfile write_data datafile

which will produce the same data file that the restart2data tool used to create. The following information is included in case you have an older version of LAMMPS which still includes the restart2data tool.

The file restart2data.cpp converts a binary LAMMPS restart file into an ASCII data file. The syntax for running the tool is

```
restart2data restart-file data-file (input-file)
```

Input-file is optional and if specified will contain LAMMPS input commands for the masses and force field parameters, instead of putting those in the data-file. Only a few force field styles currently support this option.

This tool must be compiled on a platform that can read the binary file created by a LAMMPS run, since binary files are not compatible across all platforms.

Note that a text data file has less precision than a binary restart file. Hence, continuing a run from a converted data file will typically not conform as closely to a previous run as will restarting from a binary restart file.

If a "%" appears in the specified restart-file, the tool expects a set of multiple files to exist. See the [restart](#) and [write_restart](#) commands for info on how such sets of files are written by LAMMPS, and how the files are named.

vim tool

The files in the tools/vim directory are add-ons to the VIM editor that allow easier editing of LAMMPS input scripts. See the README.txt file for details.

These files were provided by Gerolf Ziegenhain (gerolf at ziegenhain.com)

---

xmgrace tool

The files in the tools/xmgrace directory can be used to plot the thermodynamic data in LAMMPS log files via the xmgrace plotting package. There are several tools in the directory that can be used in post-processing mode. The lammpsplot.cpp file can be compiled and used to create plots from the current state of a running LAMMPS simulation.

See the README file for details.

These files were provided by Vikas Varshney (vv0210 at gmail.com)

---

xmovie tool

The xmovie tool is an X-based visualization package that can read LAMMPS dump files and animate them. It is in its own sub-directory with the tools directory. You may need to modify its Makefile so that it can find the appropriate X libraries to link against.

The syntax for running xmovie is

```
xmovie options dump.file1 dump.file2 ...
```

If you just type "xmovie" you will see a list of options. Note that by default, LAMMPS dump files are in scaled coordinates, so you typically need to use the -scale option with xmovie. When xmovie runs it opens a visualization window and a control window. The control options are straightforward to use.

Xmovie was mostly written by Mike Uttormark (U Wisconsin) while he spent a summer at Sandia. It displays 2d projections of a 3d domain. While simple in design, it is an amazingly fast program that can render large numbers of atoms very quickly. It's a useful tool for debugging LAMMPS input and output and making sure your simulation is doing what you think it should. The animations on the Examples page of the LAMMPS WWW site were created with xmovie.

I've lost contact with Mike, so I hope he's comfortable with us distributing his great tool!

set command

Syntax:

```
set style ID keyword values ...
```

- style = *atom* or *type* or *mol* or *group* or *region*
- ID = atom ID range or type range or mol ID range or group ID or region ID
- one or more keyword/value pairs may be appended
- keyword = *type* or *type/fraction* or *mol* or *x* or *y* or *z* or *charge* or *dipole* or *dipole/random* or *quat* or *quat/random* or *diameter* or *shape* or *length* or *tri* or *theta* or *angmom* or *mass* or *density* or *volume* or *image* or *bond* or *angle* or *dihedral* or *improper* or *meso_e* or *meso_cv* or *meso_rho* or *property/atom*

```
type value = atom type
    value can be an atom-style variable (see below)
  type/fraction values = type fraction seed
    type = new atom type
    fraction = fraction of selected atoms to set to new atom type
    seed = random # seed (positive integer)
  mol value = molecule ID
    value can be an atom-style variable (see below)
  x,y,z value = atom coordinate (distance units)
    value can be an atom-style variable (see below)
  charge value = atomic charge (charge units)
    value can be an atom-style variable (see below)
  dipole values = x y z
    x,y,z = orientation of dipole moment vector
    any of x,y,z can be an atom-style variable (see below)
  dipole/random value = seed Dlen
    seed = random # seed (positive integer) for dipole moment orientations
    Dlen = magnitude of dipole moment (dipole units)
  quat values = a b c theta
    a,b,c = unit vector to rotate particle around via right-hand rule
    theta = rotation angle (degrees)
    any of a,b,c,theta can be an atom-style variable (see below)
  quat/random value = seed
    seed = random # seed (positive integer) for quaternion orientations
  diameter value = diameter of spherical particle (distance units)
    value can be an atom-style variable (see below)
  shape value = Sx Sy Sz
    Sx,Sy,Sz = 3 diameters of ellipsoid (distance units)
  length value = len
    len = length of line segment (distance units)
    len can be an atom-style variable (see below)
  tri value = side
    side = side length of equilateral triangle (distance units)
    side can be an atom-style variable (see below)
  theta value = angle (degrees)
    angle = orientation of line segment with respect to x-axis
    angle can be an atom-style variable (see below)
  angmom values = Lx Ly Lz
    Lx,Ly,Lz = components of angular momentum vector (distance-mass-velocity units)
    any of Lx,Ly,Lz can be an atom-style variable (see below)
  mass value = per-atom mass (mass units)
    value can be an atom-style variable (see below)
  density value = particle density for sphere or ellipsoid (mass/distance^3 or mass/distan
    value can be an atom-style variable (see below)
  volume value = particle volume for Peridynamic particle (distance^3 units)
    value can be an atom-style variable (see below)
  image nx ny nz
    nx,ny,nz = which periodic image of the simulation box the atom is in
```

```
bond value = bond type for all bonds between selected atoms
angle value = angle type for all angles between selected atoms
dihedral value = dihedral type for all dihedrals between selected atoms
improper value = improper type for all impropers between selected atoms
meso_e value = energy of SPH particles (need units)
  value can be an atom-style variable (see below)
meso_cv value = heat capacity of SPH particles (need units)
  value can be an atom-style variable (see below)
meso_rho value = density of SPH particles (need units)
  value can be an atom-style variable (see below)
add value = yes no
  yes = add per-atom quantities to a region or a group
until value = final timestep
  final timestep = the final timestep value until which the per-atom quantity is to be a
property/atom value = varname var_value0 var_value1 ....
  varname = name of the variable to be set
  var_value0, var_value1... = values of the property to be set.
  value can be an atom-style variable (see below)
```

Examples:

```
set group solvent type 2
set group solvent type/fraction 2 0.5 12393
set group edge bond 4
set region half charge 0.5
set type 3 charge 0.5
set type 1*3 charge 0.5
set atom 100*200 x 0.5 y 1.0
set atom 1492 type 3
set region reg add yes until 1000 property/atom liqOnParticle 5e-5
set property/atom Temp 273.15
```

LIGGGHTS vs. LAMMPS Info:

This command offers a new keyword property/atom to enable the set command to assign values to properties defined by a fix property/atom

Description:

Set one or more properties of one or more atoms. Since atom properties are initially assigned by the read_data, read_restart or create_atoms commands, this command changes those assignments. This can be useful for overriding the default values assigned by the create_atoms command (e.g. charge = 0.0). It can be useful for altering pairwise and molecular force interactions, since force-field coefficients are defined in terms of types. It can be used to change the labeling of atoms by atom type or molecule ID when they are output in dump files. It can also be useful for debugging purposes; i.e. positioning an atom at a precise location to compute subsequent forces or energy.

Note that the *style* and *ID* arguments determine which atoms have their properties reset. The remaining keywords specify which properties to reset and what the new values are. Some strings like *type* or *mol* can be used as a style and/or a keyword.

This section describes how to select which atoms to change the properties of, via the *style* and *ID* arguments.

The style *atom* selects all the atoms in a range of atom IDs. The style *type* selects all the atoms in a range of types. The style *mol* selects all the atoms in a range of molecule IDs.

In each of the range cases, the range can be specified as a single numeric value, or a wildcard asterisk can be used to specify a range of values. This takes the form "*" or "*n" or "n*" or "m*n". For example, for the style *type*, if N = the number of atom types, then an asterisk with no numeric values means all types from 1 to N. A

leading asterisk means all types from 1 to n (inclusive). A trailing asterisk means all types from n to N (inclusive). A middle asterisk means all types from m to n (inclusive). For all the styles except *mol*, the lowest value for the wildcard is 1; for *mol* it is 0.

The style *group* selects all the atoms in the specified group. The style *region* selects all the atoms in the specified geometric region. See the [group](#) and [region](#) commands for details of how to specify a group or region.

This section describes the keyword options for which properties to change, for the selected atoms.

Note that except where explicitly prohibited below, all of the keywords allow an [atom-style variable](#) to be used as the specified value(s). If the value is a variable, it should be specified as v_name, where name is the variable name. In this case, the variable will be evaluated, and its resulting per-atom value used to determine the value assigned to each selected atom.

Atom-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters and timestep and elapsed time. They can also include per-atom values, such as atom coordinates. Thus it is easy to specify a time-dependent or spatially-dependent set of per-atom values. As explained on the [variable](#) doc page, atomfile-style variables can be used in place of atom-style variables, and thus as arguments to the set command. Atomfile-style variables read their per-atoms values from a file.

IMPORTANT NOTE: Atom-style and atomfile-style variables return floating point per-atom values. If the values are assigned to an integer variable, such as the molecule ID, then the floating point value is truncated to its integer portion, e.g. a value of 2.6 would become 2.

Keyword *type* sets the atom type for all selected atoms. The specified value must be from 1 to ntypes, where ntypes was set by the [create_box](#) command or the *atom types* field in the header of the data file read by the [read_data](#) command.

Keyword *type/fraction* sets the atom type for a fraction of the selected atoms. The actual number of atoms changed is not guaranteed to be exactly the requested fraction, but should be statistically close. Random numbers are used in such a way that a particular atom is changed or not changed, regardless of how many processors are being used. This keyword does not allow use of an atom-style variable.

Keyword *mol* sets the molecule ID for all selected atoms. The [atom style](#) being used must support the use of molecule IDs.

Keywords *x*, *y*, *z*, and *charge* set the coordinates or charge of all selected atoms. For *charge*, the [atom style](#) being used must support the use of atomic charge.

Keyword *dipole* uses the specified x,y,z values as components of a vector to set as the orientation of the dipole moment vectors of the selected atoms. The magnitude of the dipole moment is set by the length of this orientation vector.

Keyword *dipole/random* randomizes the orientation of the dipole moment vectors of the selected atoms and sets the magnitude of each to the specified *Dlen* value. For 2d systems, the z component of the orientation is set to 0.0. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used. This keyword does not allow use of an atom-style variable.

Keyword *quat* uses the specified values to create a quaternion (4-vector) that represents the orientation of the selected atoms. The particles must be ellipsoids as defined by the [atom_style ellipsoid](#) command or triangles as defined by the [atom_style tri](#) command. Note that particles defined by [atom_style ellipsoid](#) have 3 shape parameters. The 3 values must be non-zero for each particle set by this command. They are used to specify the

aspect ratios of an ellipsoidal particle, which is oriented by default with its x-axis along the simulation box's x-axis, and similarly for y and z. If this body is rotated (via the right-hand rule) by an angle theta around a unit rotation vector (a,b,c), then the quaternion that represents its new orientation is given by (cos(theta/2), a*sin(theta/2), b*sin(theta/2), c*sin(theta/2)). The theta and a,b,c values are the arguments to the *quat* keyword. LAMMPS normalizes the quaternion in case (a,b,c) was not specified as a unit vector. For 2d systems, the a,b,c values are ignored, since a rotation vector of (0,0,1) is the only valid choice.

Keyword *quat/random* randomizes the orientation of the quaternion of the selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command or triangles as defined by the atom_style tri command. Random numbers are used in such a way that the orientation of a particular atom is the same, regardless of how many processors are being used. For 2d systems, only orientations in the xy plane are generated. As with keyword *quat*, for ellipsoidal particles, the 3 shape values must be non-zero for each particle set by this command. This keyword does not allow use of an atom-style variable.

Keyword *diameter* sets the size of the selected atoms. The particles must be finite-size spheres as defined by the atom_style sphere command. The diameter of a particle can be set to 0.0, which means they will be treated as point particles. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *shape* sets the size and shape of the selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command. The *Sx*, *Sy*, *Sz* settings are the 3 diameters of the ellipsoid in each direction. All 3 can be set to the same value, which means the ellipsoid is effectively a sphere. They can also all be set to 0.0 which means the particle will be treated as a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *length* sets the length of selected atoms. The particles must be line segments as defined by the atom_style line command. If the specified value is non-zero the line segment is (re)set to a length = the specified value, centered around the particle position, with an orientation along the x-axis. If the specified value is 0.0, the particle will become a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *tri* sets the size of selected atoms. The particles must be triangles as defined by the atom_style tri command. If the specified value is non-zero the triangle is (re)set to be an equilateral triangle in the xy plane with side length = the specified value, with a centroid at the particle position, with its base parallel to the x axis, and the y-axis running from the center of the base to the top point of the triangle. If the specified value is 0.0, the particle will become a point particle. Note that this command does not adjust the particle mass, even if it was defined with a density, e.g. via the read_data command.

Keyword *theta* sets the orientation of selected atoms. The particles must be line segments as defined by the atom_style line command. The specified value is used to set the orientation angle of the line segments with respect to the x axis.

Keyword *angmom* sets the angular momentum of selected atoms. The particles must be ellipsoids as defined by the atom_style ellipsoid command or triangles as defined by the atom_style tri command. The angular momentum vector of the particles is set to the 3 specified components.

Keyword *mass* sets the mass of all selected particles. The particles must have a per-atom mass attribute, as defined by the atom_style command. See the "mass" command for how to set mass values on a per-type basis.

Keyword *density* also sets the mass of all selected particles, but in a different way. The particles must have a per-atom mass attribute, as defined by the atom_style command. If the atom has a radius attribute (see atom_style sphere) and its radius is non-zero, its mass is set from the density and particle volume. If the atom has a shape attribute (see atom_style ellipsoid) and its 3 shape parameters are non-zero, then its mass is set from the density and particle volume. If the atom has a length attribute (see atom_style line) and its length is

non-zero, then its mass is set from the density and line segment length (the input density is assumed to be in mass/distance units). If the atom has an area attribute (see atom_style tri) and its area is non-zero, then its mass is set from the density and triangle area (the input density is assumed to be in mass/distance^2 units). If none of these cases are valid, then the mass is set to the density value directly (the input density is assumed to be in mass units).

Keyword *volume* sets the volume of all selected particles. Currently, only the atom_style peri command defines particles with a volume attribute. Note that this command does not adjust the particle mass.

Keyword *image* sets which image of the simulation box the atom is considered to be in. An image of 0 means it is inside the box as defined. A value of 2 means add 2 box lengths to get the true value. A value of -1 means subtract 1 box length to get the true value. LAMMPS updates these flags as atoms cross periodic boundaries during the simulation. The flags can be output with atom snapshots via the dump command. If a value of NULL is specified for any of nx,ny,nz, then the current image value for that dimension is unchanged. For non-periodic dimensions only a value of 0 can be specified. This keyword does not allow use of atom-style variables.

This command can be useful after a system has been equilibrated and atoms have diffused one or more box lengths in various directions. This command can then reset the image values for atoms so that they are effectively inside the simulation box, e.g if a diffusion coefficient is about to be measured via the compute msd command. Care should be taken not to reset the image flags of two atoms in a bond to the same value if the bond straddles a periodic boundary (rather they should be different by +/- 1). This will not affect the dynamics of a simulation, but may mess up analysis of the trajectories if a LAMMPS diagnostic or your own analysis relies on the image flags to unwrap a molecule which straddles the periodic box.

Keywords *bond*, *angle*, *dihedral*, and *improper*, set the bond type (angle type, etc) of all bonds (angles, etc) of selected atoms to the specified value from 1 to nbondtypes (nangletypes, etc). All atoms in a particular bond (angle, etc) must be selected atoms in order for the change to be made. The value of nbondtype (nangletypes, etc) was set by the *bond types* (*angle types*, etc) field in the header of the data file read by the read_data command. These keywords do not allow use of an atom-style variable.

Keywords *meso_e*, *meso_cv*, and *meso_rho* set the energy, heat capacity, and density of smmothed particle hydrodynamics (SPH) particles. See this PDF guide to using SPH in LAMMPS.

Keyword *property/atom* can update per-particle properties defined by a fix property/atom command. varname is the name of the variable you want to set, it is followed by the values that should be assigned to this variable. The number of values provided as var_value**i** must correspond to the number of values needed for the update, i.e. only var_value**0** if the defined property is a scalar, and the appropriate number if the property is a vector. See fix property/atom for details on how to define such a per-particle property.

Keyword *add* and *until* are used to add a per-atom quantity (or property) in addition to the keyword *property/atom*. The *add* keyword is used to turn on the addition of a quantity in a region or a group until the timestep defined by the *until* keyword.

Restrictions:

You cannot set an atom attribute (e.g. *mol* or *q* or *volume*) if the atom_style does not have that attribute.

This command requires inter-processor communication to coordinate the setting of bond types (angle types, etc). This means that your system must be ready to perform a simulation before using one of these keywords (force fields set, atom mass set, etc). This is not necessary for other keywords.

Using the *region* style with the bond (angle, etc) keywords can give unpredictable results if there are bonds (angles, etc) that straddle periodic boundaries. This is because the region may only extend up to the boundary

and partner atoms in the bond (angle, etc) may have coordinates outside the simulation box if they are ghost atoms.

Related commands:

create_box, create_atoms, read_data

Default: none

shell command

Syntax:

```
shell cmd args
```

- cmd = *cd* or *mkdir* or *mv* or *rm* or *rmdir* or *putenv* or arbitrary command

```
  cd arg = dir
     dir = directory to change to
   mkdir args = dir1 dir2 ...
     dir1,dir2 = one or more directories to create
   mv args = old new
     old = old filename
     new = new filename
   rm args = file1 file2 ...
     file1,file2 = one or more filenames to delete
   rmdir args = dir1 dir2 ...
     dir1,dir2 = one or more directories to delete
   putenv args = var1=value1 var2=value2
     var=value = one of more definitions of environment variables
   anything else is passed as a command to the shell for direct execution
```

Examples:

```
shell cd sub1
shell cd ..
shell mkdir tmp1 tmp2 tmp3
shell rmdir tmp1
shell mv log.lammps hold/log.1
shell rm TMP/file1 TMP/file2
shell putenv LAMMPS_POTENTIALS=../../potentials
shell my_setup file1 10 file2
shell my_post_process 100 dump.out
```

Description:

Execute a shell command. A few simple file-based shell commands are supported directly, in Unix-style syntax. Any command not listed above is passed as-is to the C-library system() call, which invokes the command in a shell.

This is means to invoke other commands from your input script. For example, you can move files around in preparation for the next section of the input script. Or you can run a program that pre-processes data for input into LAMMPS. Or you can run a program that post-processes LAMMPS output data.

With the exception of *cd*, all commands, including ones invoked via a system() call, are executed by only a single processor, so that files/directories are not being manipulated by multiple processors.

The *cd* cmd executes the Unix "cd" command to change the working directory. All subsequent LAMMPS commands that read/write files will use the new directory. All processors execute this command.

The *mkdir* cmd executes the Unix "mkdir" command to create one or more directories.

The *mv* cmd executes the Unix "mv" command to rename a file and/or move it to a new directory.

The *rm* cmd executes the Unix "rm" command to remove one or more files.

The *rmdir* cmd executes the Unix "rmdir" command to remove one or more directories. A directory must be empty to be successfully removed.

The *putenv* cmd defines or updates an environment variable directly. Since this command does not pass through the shell, no shell variable expansion or globbing is performed, only the usual substitution for LAMMPS variables defined with the [variable](#) command is performed. The resulting string is then used literally.

Any other cmd is passed as-is to the shell along with its arguments as one string, invoked by the C-library system() call. For example, these lines in your input script:

```
variable n equal 10
variable foo string file2
shell my_setup file1 $n ${foo}
```

would be the same as invoking

```
% my_setup file1 10 file2
```

from a command-line prompt. The executable program "my_setup" is run with 3 arguments: file1 10 file2.

Restrictions:

LAMMPS does not detect errors or print warnings when any of these commands execute. E.g. if the specified directory does not exist, executing the *cd* command will silently do nothing.

Related commands: none

Default: none

special_bonds command

Syntax:

```
special_bonds keyword values ...
```

- one or more keyword/value pairs may be appended
- keyword = *amber* or *charmm* or *dreiding* or *fene* or *lj/coul* or *lj* or *coul* or *angle* or *dihedral* or *extra*

```
  amber values = none
    charmm values = none
    dreiding values = none
    fene values = none
    lj/coul values = w1,w2,w3
      w1,w2,w3 = weights (0.0 to 1.0) on pairwise Lennard-Jones and Coulombic interactions
    lj values = w1,w2,w3
      w1,w2,w3 = weights (0.0 to 1.0) on pairwise Lennard-Jones interactions
    coul values = w1,w2,w3
      w1,w2,w3 = weights (0.0 to 1.0) on pairwise Coulombic interactions
    angle value = yes or no
    dihedral value = yes or no
    extra value = N
      N = number of extra 1-2,1-3,1-4 interactions to save space for
```

Examples:

```
special_bonds amber
special_bonds charmm
special_bonds fene dihedral no
special_bonds lj/coul 0.0 0.0 0.5 angle yes dihedral yes
special_bonds lj 0.0 0.0 0.5 coul 0.0 0.0 0.0 dihedral yes
special_bonds lj/coul 0 1 1 extra 2
```

Description:

Set weighting coefficients for pairwise energy and force contributions between pairs of atoms that are also permanently bonded to each other, either directly or via one or two intermediate bonds. These weighting factors are used by nearly all [pair styles](#) in LAMMPS that compute simple pairwise interactions. Permanent bonds between atoms are specified by defining the bond topology in the data file read by the [read_data](#) command. Typically a [bond_style](#) command is also used to define a bond potential. The rationale for using these weighting factors is that the interaction between a pair of bonded atoms is all (or mostly) specified by the bond, angle, dihedral potentials, and thus the non-bonded Lennard-Jones or Coulombic interaction between the pair of atoms should be excluded (or reduced by a weighting factor).

IMPORTANT NOTE: These weighting factors are NOT used by [pair styles](#) that compute many-body interactions, since the "bonds" that result from such interactions are not permanent, but are created and broken dynamically as atom conformations change. Examples of pair styles in this category are EAM, MEAM, Stillinger-Weber, Tersoff, COMB, AIREBO, and ReaxFF. In fact, it generally makes no sense to define permanent bonds between atoms that interact via these potentials, though such bonds may exist elsewhere in your system, e.g. when using the [pair_style hybrid](#) command. Thus LAMMPS ignores special_bonds settings when manybody potentials are calculated.

The Coulomb factors are applied to any Coulomb (charge interaction) term that the potential calculates. The LJ factors are applied to the remaining terms that the potential calculates, whether they represent LJ interactions or not. The weighting factors are a scaling pre-factor on the energy and force between the pair of atoms. A value of 1.0 means include the full interaction; a value of 0.0 means exclude it completely.

The 1st of the 3 coefficients (LJ or Coulombic) is the weighting factor on 1-2 atom pairs, which are pairs of atoms directly bonded to each other. The 2nd coefficient is the weighting factor on 1-3 atom pairs which are those separated by 2 bonds (e.g. the two H atoms in a water molecule). The 3rd coefficient is the weighting factor on 1-4 atom pairs which are those separated by 3 bonds (e.g. the 1st and 4th atoms in a dihedral interaction). Thus if the 1-2 coefficient is set to 0.0, then the pairwise interaction is effectively turned off for all pairs of atoms bonded to each other. If it is set to 1.0, then that interaction will be at full strength.

IMPORTANT NOTE: For purposes of computing weighted pairwise interactions, 1-3 and 1-4 interactions are not defined from the list of angles or dihedrals used by the simulation. Rather, they are inferred topologically from the set of bonds specified when the simulation is defined from a data or restart file (see read_data or read_restart commands). Thus the set of 1-2,1-3,1-4 interactions that the weights apply to is the same whether angle and dihedral potentials are computed or not, and remains the same even if bonds are constrained, or turned off, or removed during a simulation.

The two exceptions to this rule are (a) if the *angle* or *dihedral* keywords are set to *yes* (see below), or (b) if the delete_bonds command is used with the *special* option that recomputes the 1-2,1-3,1-4 topologies after bonds are deleted; see the delete_bonds command for more details.

The *amber* keyword sets the 3 coefficients to 0.0, 0.0, 0.5 for LJ interactions and to 0.0, 0.0, 0.8333 for Coulombic interactions, which is the default for a commonly used version of the AMBER force field, where the last value is really 5/6. See (Cornell) for a description of the AMBER force field.

The *charmm* keyword sets the 3 coefficients to 0.0, 0.0, 0.0 for both LJ and Coulombic interactions, which is the default for a commonly used version of the CHARMM force field. Note that in pair styles *lj/charmm/coul/charmm* and *lj/charmm/coul/long* the 1-4 coefficients are defined explicitly, and these pairwise contributions are computed as part of the charmm dihedral style - see the pair_coeff and dihedral_style commands for more information. See (MacKerell) for a description of the CHARMM force field.

The *dreiding* keyword sets the 3 coefficients to 0.0, 0.0, 1.0 for both LJ and Coulombic interactions, which is the default for the Dreiding force field, as discussed in (Mayo).

The *fene* keyword sets the 3 coefficients to 0.0, 1.0, 1.0 for both LJ and Coulombic interactions, which is consistent with a coarse-grained polymer model with FENE bonds. See (Kremer) for a description of FENE bonds.

The *lj/coul*, *lj*, and *coul* keywords allow the 3 coefficients to be set explicitly. The *lj/coul* keyword sets both the LJ and Coulombic coefficients to the same 3 values. The *lj* and *coul* keywords only set either the LJ or Coulombic coefficients. Use both of them if you wish to set the LJ coefficients to different values than the Coulombic coefficients.

The *angle* keyword allows the 1-3 weighting factor to be ignored for individual atom pairs if they are not listed as the first and last atoms in any angle defined in the simulation or as 1,3 or 2,4 atoms in any dihedral defined in the simulation. For example, imagine the 1-3 weighting factor is set to 0.5 and you have a linear molecule with 4 atoms and bonds as follows: 1-2-3-4. If your data file defines 1-2-3 as an angle, but does not define 2-3-4 as an angle or 1-2-3-4 as a dihedral, then the pairwise interaction between atoms 1 and 3 will always be weighted by 0.5, but different force fields use different rules for weighting the pairwise interaction between atoms 2 and 4. If the *angle* keyword is specified as *yes*, then the pairwise interaction between atoms 2 and 4 will be unaffected (full weighting of 1.0). If the *angle* keyword is specified as *no* which is the default, then the 2,4 interaction will also be weighted by 0.5.

The *dihedral* keyword allows the 1-4 weighting factor to be ignored for individual atom pairs if they are not listed as the first and last atoms in any dihedral defined in the simulation. For example, imagine the 1-4 weighting factor is set to 0.5 and you have a linear molecule with 5 atoms and bonds as follows: 1-2-3-4-5. If

your data file defines 1-2-3-4 as a dihedral, but does not define 2-3-4-5 as a dihedral, then the pairwise interaction between atoms 1 and 4 will always be weighted by 0.5, but different force fields use different rules for weighting the pairwise interaction between atoms 2 and 5. If the *dihedral* keyword is specified as *yes*, then the pairwise interaction between atoms 2 and 5 will be unaffected (full weighting of 1.0). If the *dihedral* keyword is specified as *no* which is the default, then the 2,5 interaction will also be weighted by 0.5.

The *extra* keyword is used when additional bonds will be created during a simulation run, e.g. by the fix bond/create command. A list of 1-2,1-3,1-4 neighbors for each atom is calculated and stored by LAMMPS. If new bonds are created, the list needs to grow. Using the *extra* keyword leaves empty space in the list for N additional bonds to be added. If you do not do this, you may get an error when bonds are added.

Restrictions: none

Related commands:

delete_bonds, fix bond/create

Default:

All 3 Lennard-Jones and 3 Coulobmic weighting coefficients = 0.0, angle = no, dihedral = no, and extra = 0.

(Cornell) Cornell, Cieplak, Bayly, Gould, Merz, Ferguson, Spellmeyer, Fox, Caldwell, Kollman, JACS 117, 5179-5197 (1995).

(Kremer) Kremer, Grest, J Chem Phys, 92, 5057 (1990).

(MacKerell) MacKerell, Bashford, Bellott, Dunbrack, Evanseck, Field, Fischer, Gao, Guo, Ha, et al, J Phys Chem, 102, 3586 (1998).

(Mayo) Mayo, Olfason, Goddard III, J Phys Chem, 94, 8897-8909 (1990).

suffix command

Syntax:

```
suffix style
```

- style = *off* or *on* or *opt* or *omp* or *gpu* or *cuda*

Examples:

```
suffix off
suffix on
suffix gpu
```

Description:

This command allows you to use variants of various styles if they exist. In that respect it operates the same as the -suffix command-line switch. It also has options to turn off/on any suffix setting made via the command line.

The specified style can be *opt*, *omp*, *gpu*, or *cuda*. These refer to optional packages that LAMMPS can be built with, as described in this section of the manual. The "opt" style corrsponds to the OPT package, the "omp" style to the USER-OMP package, the "gpu" style to the GPU package, and the "cuda" style to the USER-CUDA package.

These are the variants these packages provide:

- OPT = a handful of pair styles, cache-optimized for faster CPU performance
- USER-OMP = a collection of pair, bond, angle, dihedral, improper, kspace, compute, and fix styles with support for OpenMP multi-threading
- GPU = a handful of pair styles and the PPPM kspace_style, optimized to run on one or more GPUs or multicore CPU/GPU nodes
- USER-CUDA = a collection of atom, pair, fix, compute, and intergrate styles, optimized to run on one or more NVIDIA GPUs

As an example, all of the packages provide a pair_style lj/cut variant, with style names lj/cut/opt, lj/cut/omp, lj/cut/gpu, or lj/cut/cuda. A variant styles can be specified explicitly in your input script, e.g. pair_style lj/cut/gpu. If the suffix command is used with the appropriate style, you do not need to modify your input script. The specified suffix (opt,omp,gpu,cuda) is automatically appended whenever your input script command creates a new atom, pair, bond, angle, dihedral, improper, kspace, fix, compute, or run style. If the variant version does not exist, the standard version is created.

If the specified style is *off*, then any previously specified suffix is temporarily disabled, whether it was specified by a command-line switch or a previous suffix command. If the specified style is *on*, a disabled suffix is turned back on. The use of these 2 commands lets your input script use a standard LAMMPS style (i.e. a non-accelerated variant), which can be useful for testing or benchmarking purposes. Of course this is also possible by not using any suffix commands, and explictly appending or not appending the suffix to the relevant commands in your input script.

Restrictions: none

Related commands:

Command-line switch -suffix

Default: none

tad command

Syntax:

```
tad N t_event T_lo T_hi delta tmax compute-ID keyword value ...
```

- N = # of timesteps to run (not including dephasing/quenching)
- t_event = timestep interval between event checks
- T_lo = temperature at which event times are desired
- T_hi = temperature at which MD simulation is performed
- delta = desired confidence level for stopping criterion
- tmax = reciprocal of lowest expected preexponential factor (time units)
- compute-ID = ID of the compute used for event detection
- zero or more keyword/value pairs may be appended
- keyword = *min* or *neb* or *min_style* or *neb_style* or *neb_log*

```
  min values = etol ftol maxiter maxeval
      etol = stopping tolerance for energy (energy units)
      ftol = stopping tolerance for force (force units)
      maxiter = max iterations of minimize
      maxeval = max number of force/energy evaluations
    neb values = ftol N1 N2 Nevery
      etol = stopping tolerance for energy (energy units)
      ftol = stopping tolerance for force (force units)
      N1 = max # of iterations (timesteps) to run initial NEB
      N2 = max # of iterations (timesteps) to run barrier-climbing NEB
      Nevery = print NEB statistics every this many timesteps
    min_style value = cg or hftn or sd or quickmin or fire
    neb_style value = quickmin or fire
    neb_log value = file where NEB statistics are printed
```

Examples:

```
tad 2000 50 1800 2300 0.01 0.01 event
tad 2000 50 1800 2300 0.01 0.01 event &
    min 1e-05 1e-05 100 100 &
    neb 0.0 0.01 200 200 20 &
    min_style cg &
    neb_style fire &
    neb_log log.neb
```

Description:

Run a temperature accelerated dynamics (TAD) simulation. This method requires two or more partitions to perform NEB transition state searches.

TAD is described in [this paper](#) by Art Voter. It is a method that uses accelerated dynamics at an elevated temperature to generate results at a specified lower temperature. A good overview of accelerated dynamics methods for such systems is given in [this review paper](#) from the same group. In general, these methods assume that the long-time dynamics is dominated by infrequent events i.e. the system is is confined to low energy basins for long periods, punctuated by brief, randomly-occurring transitions to adjacent basins. TAD is suitable for infrequent-event systems, where in addition, the transition kinetics are well-approximated by harmonic transition state theory (hTST). In hTST, the temperature dependence of transition rates follows the Arrhenius relation. As a consequence a set of event times generated in a high-temperature simulation can be mapped to a set of much longer estimated times in the low-temperature system. However, because this mapping involves the energy barrier of the transition event, which is different for each event, the first event at

the high temperature may not be the earliest event at the low temperature. TAD handles this by first generating a set of possible events from the current basin. After each event, the simulation is reflected backwards into the current basin. This is repeated until the stopping criterion is satisfied, at which point the event with the earliest low-temperature occurrence time is selected. The stopping criterion is that the confidence measure be greater than 1-*delta*. The confidence measure is the probability that no earlier low-temperature event will occur at some later time in the high-temperature simulation. hTST provides an lower bound for this probability, based on the user-specified minimum pre-exponential factor (reciprocal of *tmax*).

In order to estimate the energy barrier for each event, the TAD method invokes the NEB method. Each NEB replica runs on a partition of processors. The current NEB implementation in LAMMPS restricts you to having exactly one processor per replica. For more information, see the documentation for the neb command. In the current LAMMPS implementation of TAD, all the non-NEB TAD operations are performed on the first partition, while the other partitions remain idle. See Section_howto 5 of the manual for further discussion of multi-replica simulations.

A TAD run has several stages, which are repeated each time an event is performed. The logic for a TAD run is as follows:

```
while (time remains):
  while (time <tstop):
    until (event occurs):
      run dynamics for t_event steps
      quench
    run neb calculation using all replicas
    compute tlo from energy barrier
    update earliest event
    update tstop
    reflect back into current basin
  execute earliest event
```

Before this outer loop begins, the initial potential energy basin is identified by quenching (an energy minimization, see below) the initial state and storing the resulting coordinates for reference.

Inside the inner loop, dynamics is run continuously according to whatever integrator has been specified by the user, stopping every *t_event* steps to check if a transition event has occurred. This check is performed by quenching the system and comparing the resulting atom coordinates to the coordinates from the previous basin.

A quench is an energy minimization and is performed by whichever algorithm has been defined by the *min* and *min_style* keywords or their default values. Note that typically, you do not need to perform a highly-converged minimization to detect a transition event.

The event check is performed by a compute with the specified *compute-ID*. Currently there is only one compute that works with the TAD commmand, which is the compute event/displace command. Other event-checking computes may be added. Compute event/displace checks whether any atom in the compute group has moved further than a specified threshold distance. If so, an "event" has occurred.

The neb calculation is similar to that invoked by the neb command, except that the final state is generated internally, instead of being read in from a file. The TAD implementation provides default values for the NEB settings, which can be overridden using the *neb* and *neb_style* keywords.

A key aspect of the TAD method is setting the stopping criterion appropriately. If this criterion is too conservative, then many events must be generated before one is finally executed. Conversely, if this criterion is too aggressive, high-entropy high-barrier events will be over-sampled, while low-entropy low-barrier events will be under-sampled. If the lowest pre-exponential factor is known fairly accurately, then it can be used to

estimate *tmax*, and the value of *delta* can be set to the desired confidence level e.g. *delta* = 0.05 corresponds to 95% confidence. However, for systems where the dynamics are not well characterized (the most common case), it will be necessary to experiment with the values of *delta* and *tmax* to get a good trade-off between accuracy and performance.

A second key aspect is the choice of *t_hi*. A larger value greatly increases the rate at which new events are generated. However, too large a value introduces errors due to anharmonicity (not accounted for within hTST). Once again, for any given system, experimentation is necessary to determine the best value of *t_hi*.

Five kinds of output can be generated during a TAD run: event statistics, NEB statistics, thermodynamic output by each replica, dump files, and restart files.

Event statistics are printed to the screen and master log.lammps file each time an event is executed. The quantities are the timestep, CPU time, global event number *N*, local event number *M*, event status, energy barrier, time margin, *t_lo* and *delt_lo*. The timestep is the usual LAMMPS timestep, which corresponds to the high-temperature time at which the event was detected, in units of timestep. The CPU time is the total processor time since the start of the TAD run. The global event number *N* is a counter that increments with each executed event. The local event number *M* is a counter that resets to zero upon entering each new basin. The event status is *E* when an event is executed, and is *D* for an event that is detected, while *DF* is for a detected event that is also the earliest (first) event at the low temperature.

The time margin is the ratio of the high temperature time in the current basin to the stopping time. This last number can be used to judge whether the stopping time is too short or too long (see above).

t_lo is the low-temperature event time when the current basin was entered, in units of timestep. del*t_lo* is the time of each detected event, measured relative to *t_lo*. *delt_lo* is equal to the high-temperature time since entering the current basin, scaled by an exponential factor that depends on the hi/lo temperature ratio and the energy barrier for that event.

On lines for executed events, with status *E*, the global event number is incremented by one, the local event number and time margin are reset to zero, while the global event number, energy barrier, and *delt_lo* match the last event with status *DF* in the immediately preceding block of detected events. The low-temperature event time *t_lo* is incremented by *delt_lo*.

The NEB statistics are written to the file specified by the *neb_log* keyword. If the keyword value is "none", then no NEB statistics are printed out. The statistics are written every *Nevery* timesteps. See the [neb](neb) command for a full description of the NEB statistics. When invoked from TAD, NEB statistics are never printed to the screen.

Because the NEB calculation must run on multiple partitions, LAMMPS produces additional screen and log files for each partition, e.g. log.lammps.0, log.lammps.1, etc. For the TAD command, these contain the thermodynamic output of each NEB replica. In addition, the log file for the first partition, log.lammps.0, will contain thermodynamic output from short runs and minimizations corresponding to the dynamics and quench operations, as well as a line for each new detected event, as described above.

After the TAD command completes, timing statistics for the TAD run are printed in each replica's log file, giving a breakdown of how much CPU time was spent in each stage (NEB, dynamics, quenching, etc).

Any dump files defined in the input script will be written to during a TAD run at timesteps when an event is executed. This means the the requested dump frequency in the dump command is ignored. There will be one dump file (per dump command) created for all partitions. The atom coordinates of the dump snapshot are those of the minimum energy configuration resulting from quenching following the executed event. The timesteps written into the dump files correspond to the timestep at which the event occurred and NOT the clock. A dump snapshot corresponding to the initial minimum state used for event detection is written to the

dump file at the beginning of each TAD run.

If the restart command is used, a single restart file for all the partitions is generated, which allows a TAD run to be continued by a new input script in the usual manner. The restart file is generated after an event is executed. The restart file contains a snapshot of the system in the new quenched state, including the event number and the low-temperature time. The restart frequency specified in the restart command is interpreted differently when performing a TAD run. It does not mean the timestep interval between restart files. Instead it means an event interval for executed events. Thus a frequency of 1 means write a restart file every time an event is executed. A frequency of 10 means write a restart file every 10th executed event. When an input script reads a restart file from a previous TAD run, the new script can be run on a different number of replicas or processors.

Note that within a single state, the dynamics will typically temporarily continue beyond the event that is ultimately chosen, until the stopping criterionis satisfied. When the event is eventually executed, the timestep counter is reset to the value when the event was detected. Similarly, after each quench and NEB minimization, the timestep counter is reset to the value at the start of the minimization. This means that the timesteps listed in the replica log files do not always increase monotonically. However, the timestep values printed to the master log file, dump files, and restart files are always monotonically increasing.

Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the Making LAMMPS section for more info on packages.

N setting must be integer multiple of *t_event*.

Runs restarted from restart files written during a TAD run will only produce identical results if the user-specified integrator supports exact restarts. So fix nvt will produce an exact restart, but fix langevin will not.

This command cannot be used when any fixes are defined that keep track of elapsed time to perform time-dependent operations. Examples include the "ave" fixes such as fix ave/spatial. Also fix dt/reset and fix deposit.

Related commands:

compute event/displace, min_modify, min_style, run_style, minimize, temper, neb, prd

Default:

The option defaults are *min* = 0.1 0.1 40 50, *neb* = 0.01 100 100 10, *min_style* = *cg*, *neb_style* = *quickmin*, and *neb_log* = "none"

(Voter) Sorensen and Voter, J Chem Phys, 112, 9599 (2000)

(Voter2) Voter, Montalenti, Germann, Annual Review of Materials Research 32, 321 (2002).

temper command

Syntax:

```
temper N M temp fix-ID seed1 seed2 index
```

- N = total # of timesteps to run
- M = attempt a tempering swap every this many steps
- temp = initial temperature for this ensemble
- fix-ID = ID of the fix that will control temperature during the run
- seed1 = random # seed used to decide on adjacent temperature to partner with
- seed2 = random # seed for Boltzmann factor in Metropolis swap
- index = which temperature (0 to N-1) I am simulating (optional)

Examples:

```
temper 100000 100 $t tempfix 0 58728
temper 40000 100 $t tempfix 0 32285 $w
```

Description:

Run a parallel tempering or replica exchange simulation using multiple replicas (ensembles) of a system. Two or more replicas must be used.

Each replica runs on a partition of one or more processors. Processor partitions are defined at run-time using the -partition command-line switch; see [Section_start 6](#) of the manual. Note that if you have MPI installed, you can run a multi-replica simulation with more replicas (partitions) than you have physical processors, e.g you can run a 10-replica simulation on one or two processors. You will simply not get the performance speed-up you would see with one or more physical processors per replica. See [this section](#) of the manual for further discussion.

Each replica's temperature is controlled at a different value by a fix with *fix-ID* that controls temperature. Possible fix styles are [nvt](#), [temp/berendsen](#), [langevin](#) and [temp/rescale](#). The desired temperature is specified by *temp*, which is typically a variable previously set in the input script, so that each partition is assigned a different temperature. See the [variable](#) command for more details. For example:

```
variable t world 300.0 310.0 320.0 330.0
fix myfix all nvt $t $t 100.0
temper 100000 100 $t myfix 3847 58382
```

would define 4 temperatures, and assign one of them to the thermostat used by each replica, and to the temper command.

As the tempering simulation runs for *N* timesteps, a temperature swap between adjacent ensembles will be attempted every *M* timesteps. If *seed1* is 0, then the swap attempts will alternate between odd and even pairings. If *seed1* is non-zero then it is used as a seed in a random number generator to randomly choose an odd or even pairing each time. Each attempted swap of temperatures is either accepted or rejected based on a Boltzmann-weighted Metropolis criterion which uses *seed2* in the random number generator.

As a tempering run proceeds, multiple log files and screen output files are created, one per replica. By default these files are named log.lammps.M and screen.M where M is the replica number from 0 to N-1, with N = # of replicas. See the [section on command-line switches](#) for info on how to change these names.

The main screen and log file (log.lammps) will list information about which temperature is assigned to each replica at each thermodynamic output timestep. E.g. for a simulation with 16 replicas:

```
Running on 16 partitions of processors
Step T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
0    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
500  1 0 3 2 5 4 6 7 8 9 10 11 12 13 14 15
1000 2 0 4 1 5 3 6 7 8 9 10 11 12 14 13 15
1500 2 1 4 0 5 3 6 7 9 8 10 11 12 14 13 15
2000 2 1 3 0 6 4 5 7 10 8 9 11 12 14 13 15
2500 2 1 3 0 6 4 5 7 11 8 9 10 12 14 13 15
...
```

The column headings T0 to TN-1 mean which temperature is currently assigned to the replica 0 to N-1. Thus the columns represent replicas and the value in each column is its temperature (also numbered 0 to N-1). For example, a 0 in the 4th column (column T3, step 2500) means that the 4th replica is assigned temperature 0, i.e. the lowest temperature. You can verify this time sequence of temperature assignments for the Nth replica by comparing the Nth column of screen output to the thermodynamic data in the corresponding log.lammps.N or screen.N files as time proceeds.

The last argument *index* in the temper command is optional and is used when restarting a tempering run from a set of restart files (one for each replica) which had previously swapped to new temperatures. The *index* value (from 0 to N-1, where N is the # of replicas) identifies which temperature the replica was simulating on the timestep the restart files were written. Obviously, this argument must be a variable so that each partition has the correct value. Set the variable to the *N* values listed in the log file for the previous run for the replica temperatures at that timestep. For example if the log file listed the following for a simulation with 5 replicas:

```
500000 2 4 0 1 3
```

then a setting of

```
variable w world 2 4 0 1 3
```

would be used to restart the run with a tempering command like the example above with $w as the last argument.

Restrictions:

This command can only be used if LAMMPS was built with the REPLICA package. See the Making LAMMPS section for more info on packages.

Related commands:

variable, prd, neb

Default: none

thermo command

Syntax:

```
thermo N
```

- N = output thermodynamics every N timesteps
- N can be a variable (see below)

Examples:

```
thermo 100
```

Description:

Compute and print thermodynamic info (e.g. temperature, energy, pressure) on timesteps that are a multiple of N and at the beginning and end of a simulation. A value of 0 will only print thermodynamics at the beginning and end.

The content and format of what is printed is controlled by the [thermo_style](#) and [thermo_modify](#) commands.

Instead of a numeric value, N can be specifed as an [equal-style variable](#), which should be specified as v_name, where name is the variable name. In this case, the variable is evaluated at the beginning of a run to determine the next timestep at which thermodynamic info will be written out. On that timestep, the variable will be evaluated again to determine the next timestep, etc. Thus the variable should return timestep values. See the stagger() and logfreq() and stride() math functions for [equal-style variables](#), as examples of useful functions to use in this context. Other similar math functions could easily be added as options for [equal-style variables](#).

For example, the following commands will output thermodynamic info at timesteps 0,10,20,30,100,200,300,1000,2000,etc:

```
variable        s equal logfreq(10,3,10)
thermo          v_s
```

Restrictions: none

Related commands:

[thermo_style](#), [thermo_modify](#)

Default:

```
thermo 0
```

thermo_modify command

Syntax:

```
thermo_modify keyword value ...
```

- one or more keyword/value pairs may be listed

```
keyword = lost or norm or flush or line or format or temp or press:l
  lost value = error or warn or ignore
  norm value = yes or no
  flush value = yes or no
  line value = one or multi
  format values = int string or float string or M string
    M = integer from 1 to N, where N = # of quantities being printed
    string = C-style format string
  temp value = compute ID that calculates a temperature
  press value = compute ID that calculates a pressure
```

Examples:

```
thermo_modify lost ignore flush yes
thermo_modify temp myTemp format 3 %15.8g
thermo_modify line multi format float %g
```

Description:

Set options for how thermodynamic information is computed and printed by LAMMPS.

IMPORTANT NOTE: These options apply to the currently defined thermo style. When you specify a thermo_style command, all thermodynamic settings are restored to their default values, including those previously reset by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

The *lost* keyword determines whether LAMMPS checks for lost atoms each time it computes thermodynamics and what it does if atoms are lost. An atom can be "lost" if it moves across a non-periodic simulation box boundary or if it moves more than a box length outside the simulation domain (or more than a processor sub-domain length) before reneighboring occurs. The latter case is typically due to bad dynamics, e.g. too large a timestep or huge forces and velocities. If the value is *ignore*, LAMMPS does not check for lost atoms. If the value is *error* or *warn*, LAMMPS checks and either issues an error or warning. The code will exit with an error and continue with a warning. A warning will only be issued once, the first time an atom is lost. This can be a useful debugging option.

The *norm* keyword determines whether various thermodynamic output values are normalized by the number of atoms or not, depending on whether it is set to *yes* or *no*. Different unit styles have different defaults for this setting (see below). Even if *norm* is set to *yes*, a value is only normalized if it is an "extensive" quantity, meaning that it scales with the number of atoms in the system. For the thermo keywords described by the doc page for the thermo_style command, all energy-related keywords are extensive, such as *pe* or *ebond* or *enthalpy*. Other keywords such as *temp* or *press* are "intensive" meaning their value is independent (in a statistical sense) of the number of atoms in the system and thus are never normalized. For thermodynamic output values extracted from fixes and computes in a thermo_style custom command, the doc page for the individual fix or compute lists whether the value is "extensive" or "intensive" and thus whether it is normalized. Thermodynamic output values calculated by a variable formula are assumed to be "intensive" and thus are never normalized. You can always include a divide by the number of atoms in the variable formula if this is not the case.

The *flush* keyword invokes a flush operation after thermodynamic info is written to the log file. This insures the output in that file is current (no buffering by the OS), even if LAMMPS halts before the simulation completes.

The *line* keyword determines whether thermodynamics will be printed as a series of numeric values on one line or in a multi-line format with 3 quantities with text strings per line and a dashed-line header containing the timestep and CPU time. This modify option overrides the *one* and *multi* thermo_style settings.

The *format* keyword sets the numeric format of individual printed quantities. The *int* and *float* keywords set the format for all integer or floating-point quantities printed. The setting with a numeric value M (e.g. format 5 %10.4g) sets the format of the Mth value printed in each output line, e.g. the 5th column of output in this case. If the format for a specific column has been set, it will take precedent over the *int* or *float* setting.

IMPORTANT NOTE: The thermo output values *step* and *atoms* are stored internally as 8-byte signed integers, rather than the usual 4-byte signed integers. When specifying the "format int" keyword you can use a "%d"-style format identifier in the format string and LAMMPS will convert this to the corresponding "%lu" form when it is applied to those keywords. However, when specifying the "format M string" keyword for *step* and *natoms*, you should specify a string appropriate for an 8-byte signed integer, e.g. one with "%ld".

The *temp* keyword is used to determine how thermodynamic temperature is calculated, which is used by all thermo quantities that require a temperature ("temp", "press", "ke", "etotal", "enthalpy", "pxx", etc). The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a temperature. As described in the [thermo_style](#) command, thermo output uses a default compute for temperature with ID = *thermo_temp*. This option allows the user to override the default.

The *press* keyword is used to determine how thermodynamic pressure is calculated, which is used by all thermo quantities that require a pressure ("press", "enthalpy", "pxx", etc). The specified compute ID must have been previously defined by the user via the [compute](#) command and it must be a style of compute that calculates a pressure. As described in the [thermo_style](#) command, thermo output uses a default compute for pressure with ID = *thermo_press*. This option allows the user to override the default.

IMPORTANT NOTE: If both the *temp* and *press* keywords are used in a single thermo_modify command (or in two separate commands), then the order in which the keywords are specified is important. Note that a [pressure compute](#) defines its own temperature compute as an argument when it is specified. The *temp* keyword will override this (for the pressure compute being used by thermodynamics), but only if the *temp* keyword comes after the *press* keyword. If the *temp* keyword comes before the *press* keyword, then the new pressure compute specified by the *press* keyword will be unaffected by the *temp* setting.

Restrictions: none

Related commands:

[thermo](#), [thermo_style](#)

Default:

The option defaults are lost = error, norm = yes for unit style of *lj*, norm = no for unit style of *real* and *metal*, flush = no, and temp/press = compute IDs defined by thermo_style.

The defaults for the line and format options depend on the thermo style. For styles "one" and "custom", the line and format defaults are "one", "%8d", and "%12.8g". For style "multi", the line and format defaults are "multi", "%8d", and "%14.4f".

thermo_style command

Syntax:

```
thermo_style style args
```

- style = *one* or *multi* or *custom*
- args = list of arguments for a particular style

```
  one args = none
    multi args = none
    custom args = list of attributes
      possible attributes = step, elapsed, elaplong, dt, time,
                            cpu, tpcpu, spcpu, cpuremain, part, cu
                            atoms, temp, press, pe, ke, etotal, enthalpy,
                            evdwl, ecoul, epair, ebond, eangle, edihed, eimp,
                            emol, elong, etail,
                            vol, lx, ly, lz, xlo, xhi, ylo, yhi, zlo, zhi,
                            xy, xz, yz, xlat, ylat, zlat,
                            pxx, pyy, pzz, pxy, pxz, pyz,
                            fmax, fnorm,
                            cella, cellb, cellc, cellalpha, cellbeta, cellgamma,
                            c_ID, c_ID[I], c_ID[I][J],
                            f_ID, f_ID[I], f_ID[I][J],
                            v_name
        step = timestep
        elapsed = timesteps since start of this run
        elaplong = timesteps since start of initial run in a series of runs
        dt = timestep size
        time = simulation time
        cpu = elapsed CPU time in seconds
        tpcpu = time per CPU second
        spcpu = timesteps per CPU second
        cpuremain = estimated CPU time remaining in run
        part = which partition (0 to Npartition-1) this is
        cu = timesteps per CPU second
        atoms = # of atoms
        temp = temperature
        press = pressure
        pe = total potential energy
        ke = kinetic energy
        etotal = total energy (pe + ke)
        enthalpy = enthalpy (etotal + press*vol)
        evdwl = VanderWaal pairwise energy
        ecoul = Coulombic pairwise energy
        epair = pairwise energy (evdwl + ecoul + elong + etail)
        ebond = bond energy
        eangle = angle energy
        edihed = dihedral energy
        eimp = improper energy
        emol = molecular energy (ebond + eangle + edihed + eimp)
        elong = long-range kspace energy
        etail = VanderWaal energy long-range tail correction
        vol = volume
        lx,ly,lz = box lengths in x,y,z
        xlo,xhi,ylo,yhi,zlo,zhi = box boundaries
        xy,xz,yz = box tilt for triclinic (non-orthogonal) simulation boxes
        xlat,ylat,zlat = lattice spacings as calculated by lattice command
        pxx,pyy,pzz,pxy,pxz,pyz = 6 components of pressure tensor
        fmax = max component of force on any atom in any dimension
        fnorm = length of force vector for all atoms
        cella,cellb,cellc = periodic cell lattice constants a,b,c
```

```
        cellalpha, cellbeta, cellgamma = periodic cell angles alpha,beta,gamma
        c_ID = global scalar value calculated by a compute with ID
        c_ID[I] = Ith component of global vector calculated by a compute with ID
        c_ID[I][J] = I,J component of global array calculated by a compute with ID
        f_ID = global scalar value calculated by a fix with ID
        f_ID[I] = Ith component of global vector calculated by a fix with ID
        f_ID[I][J] = I,J component of global array calculated by a fix with ID
        v_name = scalar value calculated by an equal-style variable with name
```

Examples:

```
thermo_style multi
thermo_style custom step temp pe etotal press vol
thermo_style custom step temp etotal c_myTemp v_abc
```

Description:

Set the style and content for printing thermodynamic data to the screen and log file.

Style *one* prints a one-line summary of thermodynamic info that is the equivalent of "thermo_style custom step temp epair emol etotal press". The line contains only numeric values.

Style *multi* prints a multiple-line listing of thermodynamic info that is the equivalent of "thermo_style custom etotal ke temp pe ebond eangle edihed eimp evdwl ecoul elong press". The listing contains numeric values and a string ID for each quantity.

Style *custom* is the most general setting and allows you to specify which of the keywords listed above you want printed on each thermodynamic timestep. Note that the keywords c_ID, f_ID, v_name are references to computes, fixes, and equal-style variables that have been defined elsewhere in the input script or can even be new styles which users have added to LAMMPS (see the Section_modify section of the documentation). Thus the *custom* style provides a flexible means of outputting essentially any desired quantity as a simulation proceeds.

All styles except *custom* have *vol* appended to their list of outputs if the simulation box volume changes during the simulation.

The values printed by the various keywords are instantaneous values, calculated on the current timestep. Time-averaged quantities, which include values from previous timesteps, can be output by using the f_ID keyword and accessing a fix that does time-averaging such as the fix ave/time command.

Options invoked by the thermo_modify command can be used to set the one- or multi-line format of the print-out, the normalization of thermodynamic output (total values versus per-atom values for extensive quantities (ones which scale with the number of atoms in the system), and the numeric precision of each printed value.

IMPORTANT NOTE: When you use a "thermo_style" command, all thermodynamic settings are restored to their default values, including those previously set by a thermo_modify command. Thus if your input script specifies a thermo_style command, you should use the thermo_modify command after it.

Several of the thermodynamic quantities require a temperature to be computed: "temp", "press", "ke", "etotal", "enthalpy", "pxx", etc. By default this is done by using a *temperature* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_temp all temp
```

See the compute temp command for details. Note that the ID of this compute is *thermo_temp* and the group is *all*. You can change the attributes of this temperature (e.g. its degrees-of-freedom) via the compute_modify

command. Alternatively, you can directly assign a new compute (that calculates temperature) which you have defined, to be used for calculating any thermodynamic quantity that requires a temperature. This is done via the thermo_modify command.

Several of the thermodynamic quantities require a pressure to be computed: "press", "enthalpy", "pxx", etc. By default this is done by using a *pressure* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_press all pressure thermo_temp
```

See the compute pressure command for details. Note that the ID of this compute is *thermo_press* and the group is *all*. You can change the attributes of this pressure via the compute_modify command. Alternatively, you can directly assign a new compute (that calculates pressure) which you have defined, to be used for calculating any thermodynamic quantity that requires a pressure. This is done via the thermo_modify command.

Several of the thermodynamic quantities require a potential energy to be computed: "pe", "etotal", "ebond", etc. This is done by using a *pe* compute which is created when LAMMPS starts up, as if this command had been issued:

```
compute thermo_pe all pe
```

See the compute pe command for details. Note that the ID of this compute is *thermo_pe* and the group is *all*. You can change the attributes of this potential energy via the compute_modify command.

The kinetic energy of the system *ke* is inferred from the temperature of the system with 1/2 Kb T of energy for each degree of freedom. Thus, using different compute commands for calculating temperature, via the thermo_modify temp command, may yield different kinetic energies, since different computes that calculate temperature can subtract out different non-thermal components of velocity and/or include different degrees of freedom (translational, rotational, etc).

The potential energy of the system *pe* will include contributions from fixes if the fix_modify thermo option is set for a fix that calculates such a contribution. For example, the fix wall/lj93 fix calculates the energy of atoms interacting with the wall. See the doc pages for "individual fixes" to see which ones contribute.

A long-range tail correction *etail* for the VanderWaal pairwise energy will be non-zero only if the pair_modify tail option is turned on. The *etail* contribution is included in *evdwl*, *pe*, and *etotal*, and the corresponding tail correction to the pressure is included in *press* and *pxx*, *pyy*, etc.

The *step*, *elapsed*, and *elaplong* keywords refer to timestep count. *Step* is the current timestep, or iteration count when a minimization is being performed. *Elapsed* is the number of timesteps elapsed since the beginning of this run. *Elaplong* is the number of timesteps elapsed since the beginning of an initial run in a series of runs. See the *start* and *stop* keywords for the run for info on how to invoke a series of runs that keep track of an initial starting time. If these keywords are not used, then *elapsed* and *elaplong* are the same value.

The *dt* keyword is the current timestep size in time units. The *time* keyword is the current elapsed simulation time, also in time units, which is simply (step*dt) if the timestep size has not changed and the timestep has not been reset. If the timestep has changed (e.g. via fix dt/reset) or the timestep has been reset (e.g. via the "reset_timestep" command), then the simulation time is effectively a cummulative value up to the current point.

The *cpu* keyword is elapsed CPU seconds since the beginning of this run. The *tpcpu* and *spcpu* keywords are measures of how fast your simulation is currently running. The *tpcpu* keyword is simulation time per CPU second, where simulation time is in time units. E.g. for metal units, the *tpcpu* value would be picoseconds per CPU second. The *spcpu* keyword is the number of timesteps per CPU second. Both quantities are on-the-fly

thermo_style command

metrics, measured relative to the last time they were invoked. Thus if you are printing out thermodyamic output every 100 timesteps, the two keywords will continually output the time and timestep rate for the last 100 steps. The *tpcpu* keyword does not attempt to track any changes in timestep size, e.g. due to using the fix dt/reset command.

The *cpuremain* keyword estimates the CPU time remaining in the current run, based on the time elapsed thus far. It will only be a good estimate if the CPU time/timestep for the rest of the run is similar to the preceding timesteps. On the initial timestep the value will be 0.0 since there is no history to estimate from. For a minimization run performed by the "minimize" command, the estimate is based on the *maxiter* parameter, assuming the minimization will proceed for the maximum number of allowed iterations.

The *part* keyword is useful for multi-replica or multi-partition simulations to indicate which partition this output and this file corresponds to, or for use in a variable to append to a filename for output specific to this partition. See Section_start 7 of the manual for details on running in multi-partition mode.

The *fmax* and *fnorm* keywords are useful for monitoring the progress of an energy minimization. The *fmax* keyword calculates the maximum force in any dimension on any atom in the system, or the infinity-norm of the force vector for the system. The *fnorm* keyword calculates the 2-norm or length of the force vector.

The keywords *cella*, *cellb*, *cellc*, *cellalpha*, *cellbeta*, *cellgamma*, correspond to the usual crystallographic quantities that define the periodic unit cell of a crystal. See this section of the doc pages for a geometric description of triclinic periodic cells, including a precise defintion of these quantities in terms of the internal LAMMPS cell dimensions *lx*, *ly*, *lz*, *yz*, *xz*, *xy*.

The *c_ID* and *c_ID[I]* and *c_ID[I][J]* keywords allow global values calculated by a compute to be output. As discussed on the compute doc page, computes can calculate global, per-atom, or local values. Only global values can be referenced by this command. However, per-atom compute values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a compute that has been defined elsewhere in the input script. See the compute command for details. If the compute calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the compute.

Note that some computes calculate "intensive" global quantities like temperature; others calculate "extensive" global quantities like kinetic energy that are summed over all atoms in the compute group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the compute group) when output, depending on the thermo_modify norm option being used.

The *f_ID* and *f_ID[I]* and *f_ID[I][J]* keywords allow global values calculated by a fix to be output. As discussed on the fix doc page, fixes can calculate global, per-atom, or local values. Only global values can be referenced by this command. However, per-atom fix values can be referenced in a variable and the variable referenced by thermo_style custom, as discussed below.

The ID in the keyword should be replaced by the actual ID of a fix that has been defined elsewhere in the input script. See the fix command for details. If the fix calculates a global scalar, vector, or array, then the keyword formats with 0, 1, or 2 brackets will reference a scalar value from the fix.

Note that some fixes calculate "intensive" global quantities like timestep size; others calculate "extensive" global quantities like energy that are summed over all atoms in the fix group. Intensive quantities are printed directly without normalization by thermo_style custom. Extensive quantities may be normalized by the total number of atoms in the simulation (NOT the number of atoms in the fix group) when output, depending on the thermo_modify norm option being used.

The *v_name* keyword allow the current value of a variable to be output. The name in the keyword should be replaced by the variable name that has been defined elsewhere in the input script. Only equal-style variables can be referenced. See the variable command for details. Variables of style *equal* can reference per-atom properties or thermodynamic keywords, or they can invoke other computes, fixes, or variables when evaluated, so this is a very general means of creating thermodynamic output.

Note that equal-style variables are assumed to be "intensive" global quantities, which are thus printed as-is, without normalization by thermo_style custom. You can include a division by "natoms" in the variable formula if this is not the case.

Restrictions:

This command must come after the simulation box is defined by a read_data, read_restart, or create_box command.

Related commands:

thermo, thermo_modify, fix_modify, compute temp, compute pressure

Default:

```
thermo_style one
```

timestep command

Syntax:

```
timestep dt
```

- dt = timestep size (time units)

Examples:

```
timestep 2.0
timestep 0.003
```

Description:

Set the timestep size for subsequent molecular dynamics simulations. See the [units](#) command for a discussion of time units. The default value for the timestep also depends on the choice of units for the simulation; see the default values below.

When the [run style](#) is *respa*, dt is the timestep for the outer loop (largest) timestep.

Restrictions: none

Related commands:

[fix dt/reset](#), [run](#), [run_style](#) respa, [units](#)

Default:

timestep = 0.005 tau for units = lj
timestep = 1.0 fmsec for units = real
timestep = 0.001 psec for units = metal
timestep = 1.0e-8 sec (10 nsec) for units = si or cgs

uncompute command

Syntax:

```
uncompute compute-ID
```

- compute-ID = ID of a previously defined compute

Examples:

```
uncompute 2
uncompute lower-boundary
```

Description:

Delete a compute that was previously defined with a [compute](#) command. This also wipes out any additional changes made to the compute via the [compute_modify](#) command.

Restrictions: none

Related commands:

[compute](#)

Default: none

undump command

Syntax:

```
undump dump-ID
```

- dump-ID = ID of previously defined dump

Examples:

```
undump mine
undump 2
```

Description:

Turn off a previously defined dump so that it is no longer active. This closes the file associated with the dump.

Restrictions: none

Related commands:

[dump](#)

Default: none

unfix command

Syntax:

```
unfix fix-ID
```

- fix-ID = ID of a previously defined fix

Examples:

```
unfix 2
unfix lower-boundary
```

Description:

Delete a fix that was previously defined with a [fix](#) command. This also wipes out any additional changes made to the fix via the [fix_modify](#) command.

Restrictions: none

Related commands:

[fix](#)

Default: none

units command

Syntax:

```
units style
```

- style = *lj* or *real* or *metal* or *si* or *cgs* or *electron* or *micro* or *nano*

Examples:

```
units metal
units lj
```

Description:

This command sets the style of units used for a simulation. It determines the units of all quantities specified in the input script and data file, as well as quantities output to the screen, log file, and dump files. Typically, this command is used at the very beginning of an input script.

For all units except *lj*, LAMMPS uses physical constants from www.physics.nist.gov. For the definition of Kcal in real units, LAMMPS uses the thermochemical calorie = 4.184 J.

For style *lj*, all quantities are unitless. Without loss of generality, LAMMPS sets the fundamental quantities mass, sigma, epsilon, and the Boltzmann constant = 1. The masses, distances, energies you specify are multiples of these fundamental values. The formulas relating the reduced or unitless quantity (with an asterisk) to the same quantity with units is also given. Thus you can use the mass & sigma & epsilon values for a specific material and convert the results from a unitless LJ simulation into physical quantities.

- mass = mass or m
- distance = sigma, where $x^* = x / sigma$
- time = tau, where tau = $t^* = t (epsilon / m / sigma^2)^{1/2}$
- energy = epsilon, where $E^* = E / epsilon$
- velocity = sigma/tau, where $v^* = v \, tau / sigma$
- force = epsilon/sigma, where $f^* = f \, sigma / epsilon$
- torque = epsilon, where $t^* = t / epsilon$
- temperature = reduced LJ temperature, where $T^* = T \, Kb / epsilon$
- pressure = reduced LJ pressure, where $P^* = P \, sigma^3 / epsilon$
- dynamic viscosity = reduced LJ viscosity, where eta* = eta $sigma^3$ / epsilon / tau
- charge = reduced LJ charge, where $q^* = q / (4 \, pi \, perm0 \, sigma \, epsilon)^{1/2}$
- dipole = reduced LJ dipole, moment where *mu = mu / (4 pi perm0 $sigma^3$ epsilon)^{1/2}
- electric field = force/charge, where $E^* = E (4 \, pi \, perm0 \, sigma \, epsilon)^{1/2} sigma / epsilon$
- density = mass/volume, where rho* = rho $sigma^{dim}$

Note that for LJ units, the default mode of thermodyamic output via the thermo_style command is to normalize energies by the number of atoms, i.e. energy/atom. This can be changed via the thermo_modify norm command.

For style *real*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = femtoseconds
- energy = Kcal/mole

- velocity = Angstroms/femtosecond
- force = Kcal/mole-Angstrom
- torque = Kcal/mole
- temperature = Kelvin
- pressure = atmospheres
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *metal*, these are the units:

- mass = grams/mole
- distance = Angstroms
- time = picoseconds
- energy = eV
- velocity = Angstroms/picosecond
- force = eV/Angstrom
- torque = eV
- temperature = Kelvin
- pressure = bars
- dynamic viscosity = Poise
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge*Angstroms
- electric field = volts/Angstrom
- density = gram/cm^dim

For style *si*, these are the units:

- mass = kilograms
- distance = meters
- time = seconds
- energy = Joules
- velocity = meters/second
- force = Newtons
- torque = Newton-meters
- temperature = Kelvin
- pressure = Pascals
- dynamic viscosity = Pascal*second
- charge = Coulombs
- dipole = Coulombs*meters
- electric field = volts/meter
- density = kilograms/meter^dim

For style *cgs*, these are the units:

- mass = grams
- distance = centimeters
- time = seconds
- energy = ergs
- velocity = centimeters/second
- force = dynes
- torque = dyne-centimeters

- temperature = Kelvin
- pressure = dyne/cm^2 or barye = 1.0e-6 bars
- dynamic viscosity = Poise
- charge = statcoulombs or esu
- dipole = statcoul-cm = 10^18 debye
- electric field = statvolt/cm or dyne/esu
- density = grams/cm^dim

For style *electron*, these are the units:

- mass = atomic mass units
- distance = Bohr
- time = femtoseconds
- energy = Hartrees
- velocity = Bohr/atomic time units [1.03275e-15 seconds]
- force = Hartrees/Bohr
- temperature = Kelvin
- pressure = Pascals
- charge = multiple of electron charge (+1.0 is a proton)
- dipole moment = Debye
- electric field = volts/cm

For style *micro*, these are the units:

- mass = picograms
- distance = micrometers
- time = microseconds
- energy = picogram-micrometer^2/microsecond^2
- velocity = micrometers/microsecond
- force = picogram-micrometer/microsecond^2
- torque = picogram-micrometer^2/microsecond^2
- temperature = Kelvin
- pressure = picogram/(micrometer-microsecond^2)
- dynamic viscosity = picogram/(micrometer-microsecond)
- charge = picocoulombs
- dipole = picocoulomb-micrometer
- electric field = volt/micrometer
- density = picograms/micrometer^dim

For style *nano*, these are the units:

- mass = attograms
- distance = nanometers
- time = nanoseconds
- energy = attogram-nanometer^2/nanosecond^2
- velocity = nanometers/nanosecond
- force = attogram-nanometer/nanosecond^2
- torque = attogram-nanometer^2/nanosecond^2
- temperature = Kelvin
- pressure = attogram/(nanometer-nanosecond^2)
- dynamic viscosity = attogram/(nanometer-nanosecond)
- charge = multiple of electron charge (+1.0 is a proton)
- dipole = charge-nanometer
- electric field = volt/nanometer

- density = attograms/nanometer^dim

The units command also sets the timestep size and neighbor skin distance to default values for each style:

- For style *lj* these are dt = 0.005 tau and skin = 0.3 sigma.
- For style *real* these are dt = 1.0 fmsec and skin = 2.0 Angstroms.
- For style *metal* these are dt = 0.001 psec and skin = 2.0 Angstroms.
- For style *si* these are dt = 1.0e-8 sec and skin = 0.001 meters.
- For style *cgs* these are dt = 1.0e-8 sec and skin = 0.1 cm.
- For style *electron* these are dt = 0.001 fmsec and skin = 2.0 Bohr.
- For style *micro* these are dt = 2.0 microsec and skin = 0.1 micrometers.
- For style *nano* these are dt = 0.00045 nanosec and skin = 0.1 nanometers.

Restrictions:

This command cannot be used after the simulation box is defined by a read_data or create_box command.

Related commands: none

Default:

```
units lj
```

variable command

Syntax:

```
variable name style args ...
```

- name = name of variable to define
- style = *delete* or *index* or *loop* or *world* or *universe* or *uloop* or *string* or *getenv* or *file* or *atomfile* or *equal* or *atom*

```
delete = no args
  index args = one or more strings
  loop args = N
    N = integer size of loop, loop from 1 to N inclusive
  loop args = N pad
    N = integer size of loop, loop from 1 to N inclusive
    pad = all values will be same length, e.g. 001, 002, ..., 100
  loop args = N1 N2
    N1,N2 = loop from N1 to N2 inclusive
  loop args = N1 N2 pad
    N1,N2 = loop from N1 to N2 inclusive
    pad = all values will be same length, e.g. 050, 051, ..., 100
  world args = one string for each partition of processors
  universe args = one or more strings
  uloop args = N
    N = integer size of loop
  uloop args = N pad
    N = integer size of loop
    pad = all values will be same length, e.g. 001, 002, ..., 100
  string arg = one string
  getenv arg = one string
  file arg = filename
  atomfile arg = filename
  equal or atom args = one formula containing numbers, thermo keywords, math operations, g
    numbers = 0.0, 100, -5.4, 2.8e-4, etc
    constants = PI
    thermo keywords = vol, ke, press, etc from thermo style
    math operators = (), -x, x+y, x-y, x*y, x/y, x^y,
                     x==y, x!=y, xy, x>=y, x&&y, x||y, !x
    math functions = sqrt(x), exp(x), ln(x), log(x), abs(x),
                     sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x),
                     random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x)
                     ramp(x,y), stagger(x,y), logfreq(x,y,z), stride(x,y,z), vdisplace(x,y
    group functions = count(group), mass(group), charge(group),
                      xcm(group,dim), vcm(group,dim), fcm(group,dim),
                      bound(group,xmin), gyration(group), ke(group),
                      angmom(group,dim), torque(group,dim),
                      inertia(group,dimdim), omega(group,dim)
    region functions = count(group,region), mass(group,region), charge(group,region),
                       xcm(group,dim,region), vcm(group,dim,region), fcm(group,dim,region),
                       bound(group,xmin,region), gyration(group,region), ke(group,reign),
                       angmom(group,dim,region), torque(group,dim,region),
                       inertia(group,dimdim,region), omega(group,dim,region)
    special functions = sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmas
    atom value = id[i], mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy
    atom vector = id, mass, type, x, y, z, vx, vy, vz, fx, fy, fz
    compute references = c_ID, c_ID[i], c_ID[i][j]
    fix references = f_ID, f_ID[i], f_ID[i][j]
    variable references = v_name, v_name[i]
```

Examples:

variable command   1255

```
variable x index run1 run2 run3 run4 run5 run6 run7 run8
variable LoopVar loop $n
variable beta equal temp/3.0
variable b1 equal x[234]+0.5*vol
variable b1 equal "x[234] + 0.5*vol"
variable b equal xcm(mol1,x)/2.0
variable b equal c_myTemp
variable b atom x*y/vol
variable foo string myfile
variable f file values.txt
variable temp world 300.0 310.0 320.0 ${Tfinal}
variable x universe 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
variable x uloop 15 pad
variable x delete
```

Description:

This command assigns one or more strings to a variable name for evaluation later in the input script or during a simulation.

Variables can thus be useful in several contexts. A variable can be defined and then referenced elsewhere in an input script to become part of a new input command. For variable styles that store multiple strings, the next command can be used to increment which string is assigned to the variable. Variables of style *equal* store a formula which when evaluated produces a single numeric value which can be output either directly (see the print, fix print, and run every commands) or as part of thermodynamic output (see the thermo_style command), or used as input to an averaging fix (see the fix ave/time command). Variables of style *atom* store a formula which when evaluated produces one numeric value per atom which can be output to a dump file (see the dump custom command) or used as input to an averaging fix (see the fix ave/spatial and fix ave/atom commands). Variables of style *atomfile* can be used anywhere in an input script that atom-style variables are used; they get their per-atom values from a file rather than from a formula.

In the discussion that follows, the "name" of the variable is the arbitrary string that is the 1st argument in the variable command. This name can only contain alphanumeric characters and underscores. The "string" is one or more of the subsequent arguments. The "string" can be simple text as in the 1st example above, it can contain other variables as in the 2nd example, or it can be a formula as in the 3rd example. The "value" is the numeric quantity resulting from evaluation of the string. Note that the same string can generate different values when it is evaluated at different times during a simulation.

IMPORTANT NOTE: When the input script line is encountered that defines a variable of style *equal* or *atom* that contains a formula, the formula is NOT immediately evaluated and the result stored. See the discussion below about "Immediate Evaluation of Variables" if you want to do this.

IMPORTANT NOTE: When a variable command is encountered in the input script and the variable name has already been specified, the command is ignored. This means variables can NOT be re-defined in an input script (with 2 exceptions, read further). This is to allow an input script to be processed multiple times without resetting the variables; see the jump or include commands. It also means that using the command-line switch -var will override a corresponding index variable setting in the input script.

There are two exceptions to this rule. First, variables of style *string*, *getenv*, *equal* and *atom* ARE redefined each time the command is encountered. This allows these style of variables to be redefined multiple times in an input script. In a loop, this means the formula associated with an *equal* or *atom* style variable can change if it contains a substitution for another variable, e.g. $x.

Second, as described below, if a variable is iterated on to the end of its list of strings via the next command, it is removed from the list of active variables, and is thus available to be re-defined in a subsequent variable command. The *delete* style does the same thing.

[This section](#) of the manual explains how occurrences of a variable name in an input script line are replaced by the variable's string. The variable name can be referenced as $x if the name "x" is a single character, or as ${LoopVar} if the name "LoopVar" is one or more characters.

As described below, for variable styles *index*, *loop*, *file*, *universe*, and *uloop*, which string is assigned to a variable can be incremented via the [next](#) command. When there are no more strings to assign, the variable is exhausted and a flag is set that causes the next [jump](#) command encountered in the input script to be skipped. This enables the construction of simple loops in the input script that are iterated over and then exited from.

As explained above, an exhausted variable can be re-used in an input script. The *delete* style also removes the variable, the same as if it were exhausted, allowing it to be redefined later in the input script or when the input script is looped over. This can be useful when breaking out of a loop via the [if](#) and [jump](#) commands before the variable would become exhausted. For example,

```
label       loop
variable    a loop 5
print       "A = $a"
if          "$a > 2" then "jump in.script break"
next        a
jump        in.script loop
label       break
variable    a delete
```

For the *index* style, one or more strings are specified. Initially, the 1st string is assigned to the variable. Each time a [next](#) command is used with the variable name, the next string is assigned. All processors assign the same string to the variable.

Index style variables with a single string value can also be set by using the command-line switch -var; see [this section](#) for details.

The *loop* style is identical to the *index* style except that the strings are the integers from 1 to N inclusive, if only one argument N is specified. This allows generation of a long list of runs (e.g. 1000) without having to list N strings in the input script. Initially, the string "1" is assigned to the variable. Each time a [next](#) command is used with the variable name, the next string ("2", "3", etc) is assigned. All processors assign the same string to the variable. The *loop* style can also be specified with two arguments N1 and N2. In this case the loop runs from N1 to N2 inclusive, and the string N1 is initially assigned to the variable. N1 <= N2 and N2 >= 0 is required.

For the *world* style, one or more strings are specified. There must be one string for each processor partition or "world". See [this section](#) of the manual for information on running LAMMPS with multiple partitions via the "-partition" command-line switch. This variable command assigns one string to each world. All processors in the world are assigned the same string. The next command cannot be used with *equal* style variables, since there is only one value per world. This style of variable is useful when you wish to run different simulations on different partitions, or when performing a parallel tempering simulation (see the [temper](#) command), to assign different temperatures to different partitions.

For the *universe* style, one or more strings are specified. There must be at least as many strings as there are processor partitions or "worlds". See [this page](#) for information on running LAMMPS with multiple partitions via the "-partition" command-line switch. This variable command initially assigns one string to each world. When a [next](#) command is encountered using this variable, the first processor partition to encounter it, is assigned the next available string. This continues until all the variable strings are consumed. Thus, this command can be used to run 50 simulations on 8 processor partitions. The simulations will be run one after the other on whatever partition becomes available, until they are all finished. *Universe* style variables are incremented using the files "tmp.lammps.variable" and "tmp.lammps.variable.lock" which you will see in your directory during such a LAMMPS run.

The *uloop* style is identical to the *universe* style except that the strings are the integers from 1 to N. This allows generation of long list of runs (e.g. 1000) without having to list N strings in the input script.

For the *string* style, a single string is assigned to the variable. The only difference between this and using the *index* style with a single string is that a variable with *string* style can be redefined. E.g. by another command later in the input script, or if the script is read again in a loop.

For the *getenv* style, a single string is assigned to the variable which should be the name of an environment variable. When the variable is evaluated, it returns the value of the environment variable, or an empty string if it not defined. This style of variable can be used to adapt the behavior of LAMMPS input scripts via environment variable settings, or to retrieve information that has been previously stored with the [shell putenv](#) command. Note that because environment variable settings are stored by the operating systems, they persist beyond a [clear](#) command.

For the *file* style, a filename is provided which contains a list of strings to assign to the variable, one per line. The strings can be numeric values if desired. See the discussion of the next() function below for equal-style variables, which will convert the string of a file-style variable into a numeric value in a formula.

When a file-style variable is defined, the file is opened and the string on the first line is read and stored with the variable. This means the variable can then be evaluated as many times as desired and will return that string. There are two ways to cause the next string from the file to be read: use the [next](#) command or the next() function in an equal- or atom-style variable, as discussed below.

The rules for formatting the file are as follows. A comment character "#" can be used anywhere on a line; text starting with the comment character is stripped. Blank lines are skipped. The first "word" of a non-blank line, delimited by white space, is the "string" assigned to the variable.

For the *atomfile* style, a filename is provided which contains one or more sets of values, to assign on a per-atom basis to the variable. The format of the file is described below.

When an atomfile-style variable is defined, the file is opened and the first set of per-atom values are read and stored with the variable. This means the variable can then be evaluated as many times as desired and will return those values. There are two ways to cause the next set of per-atom values from the file to be read: use the [next](#) command or the next() function in an atom-style variable, as discussed below.

The rules for formatting the file are as follows. Each time a set of per-atom values is read, a non-blank line is searched for in the file. A comment character "#" can be used anywhere on a line; text starting with the comment character is stripped. Blank lines are skipped. The first "word" of a non-blank line, delimited by white space, is read as the count N of per-atom lines to immediately follow. N can be be the total number of atoms in the system, or only a subset. The next N lines have the following format

```
ID value
```

where ID is an atom ID and value is the per-atom numeric value that will be assigned to that atom. IDs can be listed in any order.

IMPORTANT NOTE: Every time a set of per-atom lines is read, the value for all atoms is first set to 0.0. Thus values for atoms whose ID does not appear in the set, will remain 0.0.

For the *equal* and *atom* styles, a single string is specified which represents a formula that will be evaluated afresh each time the variable is used. If you want spaces in the string, enclose it in double quotes so the parser will treat it as a single argument. For *equal* style variables the formula computes a scalar quantity, which becomes the value of the variable whenever it is evaluated. For *atom* style variables the formula computes one quantity for each atom whenever it is evaluated.

Note that *equal* and *atom* variables can produce different values at different stages of the input script or at different times during a run. For example, if an *equal* variable is used in a [fix print](#) command, different values could be printed each timestep it was invoked. If you want a variable to be evaluated immediately, so that the result is stored by the variable instead of the string, see the section below on "Immediate Evaluation of Variables".

The next command cannot be used with *equal* or *atom* style variables, since there is only one string.

The formula for an *equal* or *atom* variable can contain a variety of quantities. The syntax for each kind of quantity is simple, but multiple quantities can be nested and combined in various ways to build up formulas of arbitrary complexity. For example, this is a valid (though strange) variable formula:

```
variable x equal "pe + c_MyTemp / vol^(1/3)"
```

Specifically, an formula can contain numbers, thermo keywords, math operators, math functions, group functions, region functions, atom values, atom vectors, compute references, fix references, and references to other variables.

Number	0.2, 100, 1.0e20, -15.4, etc
Constant	PI
Thermo keywords	vol, pe, ebond, etc
Math operators	(), -x, x+y, x-y, x*y, x/y, x^y, x==y, x!=y, xy, x>=y, x&&y, x||y, !x
Math functions	sqrt(x), exp(x), ln(x), log(x), abs(x), sin(x), cos(x), tan(x), asin(x), acos(x), atan(x), atan2(y,x), random(x,y,z), normal(x,y,z), ceil(x), floor(x), round(x), ramp(x,y), stagger(x,y), logfreq(x,y,z), stride(x,y,z), vdisplace(x,y), swiggle(x,y,z), cwiggle(x,y,z)
Group functions	count(ID), mass(ID), charge(ID), xcm(ID,dim), vcm(ID,dim), fcm(ID,dim), bound(ID,dir), gyration(ID), ke(ID), angmom(ID,dim), torque(ID,dim), inertia(ID,dimdim), omega(ID,dim)
Region functions	count(ID,IDR), mass(ID,IDR), charge(ID,IDR), xcm(ID,dim,IDR), vcm(ID,dim,IDR), fcm(ID,dim,IDR), bound(ID,dir,IDR), gyration(ID,IDR), ke(ID,IDR), angmom(ID,dim,IDR), torque(ID,dim,IDR), inertia(ID,dimdim,IDR), omega(ID,dim,IDR)
Special functions	sum(x), min(x), max(x), ave(x), trap(x), gmask(x), rmask(x), grmask(x,y), next(x)
Atom values	id[i], mass[i], type[i], x[i], y[i], z[i], vx[i], vy[i], vz[i], fx[i], fy[i], fz[i]
Atom vectors	id, mass, type, x, y, z, vx, vy, vz, fx, fy, fz
Compute references	c_ID, c_ID[i], c_ID[i][j]
Fix references	f_ID, f_ID[i], f_ID[i][j]
Other variables	v_name, v_name[i]

Most of the formula elements produce a scalar value. A few produce a per-atom vector of values. These are the atom vectors, compute references that represent a per-atom vector, fix references that represent a per-atom vector, and variables that are atom-style variables. Math functions that operate on scalar values produce a scalar value; math function that operate on per-atom vectors do so element-by-element and produce a per-atom vector.

A formula for equal-style variables cannot use any formula element that produces a per-atom vector. A formula for an atom-style variable can use formula elements that produce either a scalar value or a per-atom vector. Atom-style variables are evaluated by other commands that define a [group](#) on which they operate, e.g.

variable command   1259

a [dump](#) or [compute](#) or [fix](#) command. When they invoke the atom-style variable, only atoms in the group are inlcuded in the formula evaluation. The variable evaluates to 0.0 for atoms not in the group.

The thermo keywords allowed in a formula are those defined by the [thermo_style custom](#) command. Thermo keywords that require a [compute](#) to calculate their values such as "temp" or "press", use computes stored and invoked by the [thermo_style](#) command. This means that you can only use those keywords in a variable if the style you are using with the thermo_style command (and the thermo keywords associated with that style) also define and use the needed compute. Note that some thermo keywords use a compute indirectly to calculate their value (e.g. the enthalpy keyword uses temp, pe, and pressure). If a variable is evaluated directly in an input script (not during a run), then the values accessed by the thermo keyword must be current. See the discussion below about "Variable Accuracy".

Math Operators

Math operators are written in the usual way, where the "x" and "y" in the examples can themselves be arbitrarily complex formulas, as in the examples above. In this syntax, "x" and "y" can be scalar values or per-atom vectors. For example, "ke/natoms" is the division of two scalars, where "vy+vz" is the element-by-element sum of two per-atom vectors of y and z velocities.

Operators are evaluated left to right and have the usual C-style precedence: unary minus and unary logical NOT operator "!" have the highest precedence, exponentiation "^" is next; multiplication and division are next; addition and subtraction are next; the 4 relational operators "", and ">=" are next; the two remaining relational operators "==" and "!=" are next; then the logical AND operator "&&"; and finally the logical OR operator "‖" has the lowest precedence. Parenthesis can be used to group one or more portions of a formula and/or enforce a different order of evaluation than what would occur with the default precedence.

IMPORTANT NOTE: Because a unary minus is higher precedence than exponentiation, the formula "-2^2" will evaluate to 4, not -4. This convention is compatible with some programming languages, but not others. As mentioned, this behavior can be easily overridden with parenthesis; the formula "-(2^2)" will evaluate to -4.

The 6 relational operators return either a 1.0 or 0.0 depending on whether the relationship between x and y is TRUE or FALSE. For example the expression x

These relational and logical operators can be used as a masking or selection operation in a formula. For example, the number of atoms whose properties satifsy one or more criteria could be calculated by taking the returned per-atom vector of ones and zeroes and passing it to the [compute reduce](#) command.

Math Functions

Math functions are specified as keywords followed by one or more parenthesized arguments "x", "y", "z", each of which can themselves be arbitrarily complex formulas. In this syntax, the arguments can represent scalar values or per-atom vectors. In the latter case, the math operation is performed on each element of the vector. For example, "sqrt(natoms)" is the sqrt() of a scalar, where "sqrt(y*z)" yields a per-atom vector with each element being the sqrt() of the product of one atom's y and z coordinates.

Most of the math functions perform obvious operations. The ln() is the natural log; log() is the base 10 log.

The random(x,y,z) function takes 3 arguments: x = lo, y = hi, and z = seed. It generates a uniform random number between lo and hi. The normal(x,y,z) function also takes 3 arguments: x = mu, y = sigma, and z = seed. It generates a Gaussian variate centered on mu with variance sigma^2. In both cases the seed is used the first time the internal random number generator is invoked, to initialize it. For equal-style variables, every processor uses the same seed so that they each generate the same sequence of random numbers. For atom-style variables, a unique seed is created for each processor, based on the specified seed. This effectively

generates a different random number for each atom being looped over in the atom-style variable.

IMPORTANT NOTE: Internally, there is just one random number generator for all equal-style variables and one for all atom-style variables. If you define multiple variables (of each style) which use the random() or normal() math functions, then the internal random number generators will only be initialized once, which means only one of the specified seeds will determine the sequence of generated random numbers.

The ceil(), floor(), and round() functions are those in the C math library. Ceil() is the smallest integer not less than its argument. Floor() if the largest integer not greater than its argument. Round() is the nearest integer to its argument.

The ramp(x,y) function uses the current timestep to generate a value linearly intepolated between the specified x,y values over the course of a run, according to this formula:

```
value = x + (y-x) * (timestep-startstep) / (stopstep-startstep)
```

The run begins on startstep and ends on stopstep. Startstep and stopstep can span multiple runs, using the *start* and *stop* keywords of the run command. See the run command for details of how to do this.

The stagger(x,y) function uses the current timestep to generate a new timestep. X,y > 0 and x > y are required. The generated timesteps increase in a staggered fashion, as the sequence x,x+y,2x,2x+y,3x,3x+y,etc. For any current timestep, the next timestep in the sequence is returned. Thus if stagger(1000,100) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,1000,1100,2000,2100,3000,etc
```

The logfreq(x,y,z) function uses the current timestep to generate a new timestep. X,y,z > 0 and y < z are required. The generated timesteps increase in a logarithmic fashion, as the sequence x,2x,3x,...y*x,z*x,2*z*x,3*z*x,...y*z*x,z*z*x,2*z*x*x,etc. For any current timestep, the next timestep in the sequence is returned. Thus if logfreq(100,4,10) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
100,200,300,400,1000,2000,3000,4000,10000,20000,etc
```

The stride(x,y,z) function uses the current timestep to generate a new timestep. X,y >= 0 and z > 0 and x <= y are required. The generated timesteps increase in increments of z, from x to y, I.e. it generates the sequece x,x+z,x+2z,...,y. If y-x is not a multiple of z, then similar to the way a for loop operates, the last value will be one that does not exceed y. For any current timestep, the next timestep in the sequence is returned. Thus if stagger(1000,2000,100) is used in a variable by the dump_modify every command, it will generate the sequence of output timesteps:

```
1000,1100,1200, ... ,1900,2000
```

The vdisplace(x,y) function takes 2 arguments: x = value0 and y = velocity, and uses the elapsed time to change the value by a linear displacement due to the applied velocity over the course of a run, according to this formula:

```
value = value0 + velocity*(timestep-startstep)*dt
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command. See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep-startstep.

Math Functions   1261

The swiggle(x,y,z) and cwiggle(x,y,z) functions each take 3 arguments: x = value0, y = amplitude, z = period. They use the elapsed time to oscillate the value by a sin() or cos() function over the course of a run, according to one of these formulas, where omega = 2 PI / period:

```
value = value0 + Amplitude * sin(omega*(timestep-startstep)*dt)
value = value0 + Amplitude * (1 - cos(omega*(timestep-startstep)*dt))
```

where dt = the timestep size.

The run begins on startstep. Startstep can span multiple runs, using the *start* keyword of the run command. See the run command for details of how to do this. Note that the thermo_style keyword elaplong = timestep-startstep.

Group and Region Functions

Group functions are specified as keywords followed by one or two parenthesized arguments. The first argument is the group-ID. The *dim* argument, if it exists, is *x* or *y* or *z*. The *dir* argument, if it exists, is *xmin*, *xmax*, *ymin*, *ymax*, *zmin*, or *zmax*. The *dimdim* argument, if it exists, is *xx* or *yy* or *zz* or *xy* or *yz* or *xz*.

The group function count() is the number of atoms in the group. The group functions mass() and charge() are the total mass and charge of the group. Xcm() and vcm() return components of the position and velocity of the center of mass of the group. Fcm() returns a component of the total force on the group of atoms. Bound() returns the min/max of a particular coordinate for all atoms in the group. Gyration() computes the radius-of-gyration of the group of atoms. See the compute gyration command for a definition of the formula. Angmom() returns components of the angular momentum of the group of atoms around its center of mass. Torque() returns components of the torque on the group of atoms around its center of mass, based on current forces on the atoms. Inertia() returns one of 6 components of the symmetric inertia tensor of the group of atoms around its center of mass, ordered as Ixx,Iyy,Izz,Ixy,Iyz,Ixz. Omega() returns components of the angular velocity of the group of atoms around its center of mass.

Region functions are specified exactly the same way as group functions except they take an extra argument which is the region ID. The function is computed for all atoms that are in both the group and the region. If the group is "all", then the only criteria for atom inclusion is that it be in the region.

Special Functions

Special functions take specific kinds of arguments, meaning their arguments cannot be formulas themselves.

The sum(x), min(x), max(x), ave(x), and trap(x) functions each take 1 argument which is of the form "c_ID" or "c_ID[N]" or "f_ID" or "f_ID[N]". The first two are computes and the second two are fixes; the ID in the reference should be replaced by the ID of a compute or fix defined elsewhere in the input script. The compute or fix must produce either a global vector or array. If it produces a global vector, then the notation without "[N]" should be used. If it produces a global array, then the notation with "[N]" should be used, when N is an integer, to specify which column of the global array is being referenced.

These functions operate on the global vector of inputs and reduce it to a single scalar value. This is analagous to the operation of the compute reduce command, which invokes the same functions on per-atom and local vectors.

The sum() function calculates the sum of all the vector elements. The min() and max() functions find the minimum and maximum element respectively. The ave() function is the same as sum() except that it divides the result by the length of the vector. The trap() function is the same as sum() except the first and last elements are multiplied by a weighting factor of 1/2 when performing the sum. This effectively implements an integratiion via the trapezoidal rule on the global vector of data. I.e. consider a set of points, equally spaced by 1 in their x coordinate: (1,V1), (2,V2), ..., (N,VN), where the Vi are the values in the global vector of length

N. The integral from 1 to N of these points is trap(). When appropriately normalized by the timestep size, this function is useful for calculating integrals of time-series data, like that generated by the fix ave/correlate command.

The gmask(x) function takes 1 argument which is a group ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in the group, and a 0 for atoms that are not.

The rmask(x) function takes 1 argument which is a region ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in the geometric region, and a 0 for atoms that are not.

The grmask(x,y) function takes 2 arguments. The first is a group ID, and the second is a region ID. It can only be used in atom-style variables. It returns a 1 for atoms that are in both the group and region, and a 0 for atoms that are not in both.

The next(x) function takes 1 argument which is a variable ID (not "v_foo", just "foo"). It must be for a file-style or atomfile-style variable. Each time the next() function is invoked (i.e. each time the equal-style or atom-style variable is evaluated), the following steps occur.

For file-style variables, the current string value stored by the file-style variable is converted to a numeric value and returned by the function. And the next string value in the file is read and stored. Note that if the line previously read from the file was not a numeric string, then it will typically evaluate to 0.0, which is likely not what you want.

For atomfile-style variables, the current per-atom values stored by the atomfile-style variable are returned by the function. And the next set of per-atom values in the file is read and stored.

Since file-style and atomfile-style variables read and store the first line of the file or first set of per-atoms values when they are defined in the input script, these are the value(s) that will be returned the first time the next() function is invoked. If next() is invoked more times than there are lines or sets of lines in the file, the variable is deleted, similar to how the next command operates.

Atom Values and Vectors

Atom values take a single integer argument I from 1 to N, where I is the an atom-ID, e.g. x[243], which means use the x coordinate of the atom with ID = 243.

Atom vectors generate one value per atom, so that a reference like "vx" means the x-component of each atom's velocity will be used when evaluating the variable. Note that other atom attributes can be used as inputs to a variable by using the compute property/atom command and then specifying a quantity from that compute.

Compute References

Compute references access quantities calculated by a compute. The ID in the reference should be replaced by the ID of a compute defined elsewhere in the input script. As discussed in the doc page for the compute command, computes can produce global, per-atom, or local values. Only global and per-atom values can be used in a variable. Computes can also produce a scalar, vector, or array. An equal-style variable can only use scalar values, which means a global scalar, or an element of a global or per-atom vector or array. Atom-style variables can use the same scalar values. They can also use per-atom vector values. A vector value can be a per-atom vector itself, or a column of an per-atom array. See the doc pages for individual computes to see what kind of values they produce.

Examples of different kinds of compute references are as follows. There is no ambiguity as to what a reference means, since computes only produce global or per-atom quantities, never both.

c_ID	global scalar, or per-atom vector
c_ID[I]	Ith element of global vector, or atom I's value in per-atom vector, or Ith column from per-atom array
c_ID[I][J]	I,J element of global array, or atom I's Jth value in per-atom array

If a variable containing a compute is evaluated directly in an input script (not during a run), then the values accessed by the compute must be current. See the discussion below about "Variable Accuracy".

Fix References

Fix references access quantities calculated by a fix. The ID in the reference should be replaced by the ID of a fix defined elsewhere in the input script. As discussed in the doc page for the fix command, fixes can produce global, per-atom, or local values. Only global and per-atom values can be used in a variable. Fixes can also produce a scalar, vector, or array. An equal-style variable can only use scalar values, which means a global scalar, or an element of a global or per-atom vector or array. Atom-style variables can use the same scalar values. They can also use per-atom vector values. A vector value can be a per-atom vector itself, or a column of an per-atom array. See the doc pages for individual fixes to see what kind of values they produce.

The different kinds of fix references are exactly the same as the compute references listed in the above table, where "c_" is replaced by "f_". Again, there is no ambiguity as to what a reference means, since fixes only produce global or per-atom quantities, never both.

f_ID	global scalar, or per-atom vector
f_ID[I]	Ith element of global vector, or atom I's value in per-atom vector, or Ith column from per-atom array
f_ID[I][J]	I,J element of global array, or atom I's Jth value in per-atom array

If a variable containing a fix is evaluated directly in an input script (not during a run), then the values accessed by the fix should be current. See the discussion below about "Variable Accuracy".

Note that some fixes only generate quantities on certain timesteps. If a variable attempts to access the fix on non-allowed timesteps, an error is generated. For example, the fix ave/time command may only generate averaged quantities every 100 steps. See the doc pages for individual fix commands for details.

Variable References

Variable references access quantities stored or calculated by other variables, which will cause those variables to be evaluated. The name in the reference should be replaced by the name of a variable defined elsewhere in the input script.

As discussed on this doc page, equal-style variables generate a global scalar numeric value; atom-style and atomfile-style variables generate a per-atom vector of numeric values; all other variables store a string. The formula for an equal-style variable can use any style of variable except an atom-style or atomfile-style (unless only a single value from the variable is accessed via a subscript). If a string-storing variable is used, the string is converted to a numeric value. Note that this will typically produce a 0.0 if the string is not a numeric string, which is likely not what you want. The formula for an atom-style variable can use any style of variable, including other atom-style or atomfile-style variables.

Examples of different kinds of variable references are as follows. There is no ambiguity as to what a reference means, since variables produce only a global scalar or a per-atom vector, never both.

v_name	scalar, or per-atom vector
v_name[I]	atom I's value in per-atom vector

Immediate Evaluation of Variables:

There is a difference between referencing a variable with a leading $ sign (e.g. $x or ${abc}) versus with a leading "v_" (e.g. v_x or v_abc). The former can be used in any input script command, including a variable command. The input script parser evaluates the reference variable immediately and substitutes its value into the command. As explained in Section commands 3.2 for "Parsing rules", you can also use un-named "immediate" variables for this purpose. An variable reference such as $((xlo+xhi)/2+sqrt(v_area)) evaluates the string between the parenthesis as an equal-style variable.

Referencing a variable with a leading "v_" is an optional or required kind of argument for some commands (e.g. the fix ave/spatial or dump custom or thermo_style commands) if you wish it to evaluate a variable periodically during a run. It can also be used in a variable formula if you wish to reference a second variable. The second variable will be evaluated whenever the first variable is evaluated.

As an example, suppose you use this command in your input script to define the variable "v" as

```
variable v equal vol
```

before a run where the simulation box size changes. You might think this will assign the initial volume to the variable "v". That is not the case. Rather it assigns a formula which evaluates the volume (using the thermo_style keyword "vol") to the variable "v". If you use the variable "v" in some other command like fix ave/time then the current volume of the box will be evaluated continuously during the run.

If you want to store the initial volume of the system, you can do it this way:

```
variable v equal vol
variable v0 equal $v
```

The second command will force "v" to be evaluated (yielding the initial volume) and assign that value to the variable "v0". Thus the command

```
thermo_style custom step v_v v_v0
```

would print out both the current and initial volume periodically during the run.

Note that it is a mistake to enclose a variable formula in double quotes if it contains variables preceeded by $ signs. For example,

```
variable vratio equal "${vfinal}/${v0}"
```

This is because the quotes prevent variable substitution (see this section on parsing input script commands), and thus an error will occur when the formula for "vratio" is evaluated later.

Variable Accuracy:

Obviously, LAMMPS attempts to evaluate variables containing formulas (*equal* and *atom* style variables) accurately whenever the evaluation is performed. Depending on what is included in the formula, this may require invoking a compute, either directly or indirectly via a thermo keyword, or accessing a value previously calculated by a compute, or accessing a value calculated and stored by a fix. If the compute is one that calculates the pressure or energy of the system, then these quantities need to be tallied during the evaluation of the interatomic potentials (pair, bond, etc) on timesteps that the variable will need the values.

LAMMPS keeps track of all of this during a run or energy minimization. An error will be generated if you attempt to evaluate a variable on timesteps when it cannot produce accurate values. For example, if a thermo_style custom command prints a variable which accesses values stored by a fix ave/time command and

the timesteps on which thermo output is generated are not multiples of the averaging frequency used in the fix command, then an error will occur.

An input script can also request variables be evaluated before or after or in between runs, e.g. by including them in a print command. In this case, if a compute is needed to evaluate a variable (either directly or indirectly), LAMMPS will not invoke the compute, but it will use a value previously calculated by the compute, and can do this only if it is current. Fixes will always provide a quantity needed by a variable, but the quantity may or may not be current. This leads to one of three kinds of behavior:

(1) The variable may be evaluated accurately. If it contains references to a compute or fix, and these values were calculated on the last timestep of a preceeding run, then they will be accessed and used by the variable and the result will be accurate.

(2) LAMMPS may not be able to evaluate the variable and will generate an error message stating so. For example, if the variable requires a quantity from a compute that is not current, LAMMPS will generate an error. This means, for example, that such a variable cannot be evaluated before the first run has occurred. Likewise, in between runs, such a variable cannot be accessed unless it was evaluated on the last timestep of the preceding run, e.g. by thermodynamic output.

One way to get around this problem is to perform a 0-timestep run before using the variable. For example, these commands

```
variable t equal temp
print "Initial temperature = $t"
run 1000
```

will generate an error if the run is the first run specified in the input script, because generating a value for the "t" variable requires a compute for calculating the temperature to be invoked.

However, this sequence of commands would be fine:

```
run 0
variable t equal temp
print "Initial temperature = $t"
run 1000
```

The 0-timestep run initializes and invokes various computes, including the one for temperature, so that the value it stores is current and can be accessed by the variable "t" after the run has completed. Note that a 0-timestep run does not alter the state of the system, so it does not change the input state for the 1000-timestep run that follows. Also note that the 0-timestep run must actually use and invoke the compute in question (e.g. via thermo or dump output) in order for it to enable the compute to be used in a variable after the run. Thus if you are trying to print a variable that uses a compute you have defined, you could insure it was invoked on the last timestep of the preceding run by including it in thermodynamic output.

Unlike computes, fixes will never generate an error if their values are accessed by a variable in between runs. They always return some value to the variable. However, the value may not be what you expect if the fix has not yet calculated the quantity of interest or it is not current. For example, the fix indent command stores the force on the indenter. But this is not computed until a run is performed. Thus if a variable attempts to print this value before the first run, zeroes will be output. Again, performing a 0-timestep run before printing the variable has the desired effect.

(3) The variable may be evaluated incorrectly and LAMMPS may have no way to detect this has occurred. Consider the following sequence of commands:

```
pair_coeff 1 1 1.0 1.0
run 1000
```

```
pair_coeff 1 1 1.5 1.0
variable e equal pe
print "Final potential energy = $e"
```

The first run is performed using one setting for the pairwise potential defined by the [pair_style](#) and [pair_coeff](#) commands. The potential energy is evaluated on the final timestep and stored by the [compute pe](#) compute (this is done by the [thermo_style](#) command). Then a pair coefficient is changed, altering the potential energy of the system. When the potential energy is printed via the "e" variable, LAMMPS will use the potential energy value stored by the [compute pe](#) compute, thinking it is current. There are many other commands which could alter the state of the system between runs, causing a variable to evaluate incorrectly.

The solution to this issue is the same as for case (2) above, namely perform a 0-timestep run before the variable is evaluated to insure the system is up-to-date. For example, this sequence of commands would print a potential energy that reflected the changed pairwise coefficient:

```
pair_coeff 1 1 1.0 1.0
run 1000
pair_coeff 1 1 1.5 1.0
run 0
variable e equal pe
print "Final potential energy = $e"
```

Restrictions:

Indexing any formula element by global atom ID, such as an atom value, requires the atom style to use a global mapping in order to look up the vector indices. By default, only atom styles with molecular information create global maps. The [atom_modify map](#) command can override the default.

All *universe-* and *uloop*-style variables defined in an input script must have the same number of values.

Related commands:

[next](#), [jump](#), [include](#), [temper](#), [fix print](#), [print](#)

Default: none

velocity command

Syntax:

```
velocity group-ID style args keyword value ...
```

- group-ID = ID of group of atoms whose velocity will be changed
- style = *create* or *set* or *scale* or *ramp* or *zero*

```
  create args = temp seed
      temp = temperature value (temperature units)
      seed = random # seed (positive integer)
    set args = vx vy vz
      vx,vy,vz = velocity value or NULL (velocity units)
      any of vx,vy,vz van be a variable (see below)
    scale arg = temp
      temp = temperature value (temperature units)
    ramp args = vdim vlo vhi dim clo chi
      vdim = vx or vy or vz
      vlo,vhi = lower and upper velocity value (velocity units)
      dim = x or y or z
      clo,chi = lower and upper coordinate bound (distance units)
    zero arg = linear or angular
      linear = zero the linear momentum
      angular = zero the angular momentum
```

- zero or more keyword/value pairs may be appended
- keyword = *dist* or *sum* or *mom* or *rot* or *temp* or *loop* or *units*

```
  dist value = uniform or gaussian
    sum value = no or yes
    mom value = no or yes
    rot value = no or yes
    temp value = temperature ID
    loop value = all or local or geom
    rigid value = fix-ID
      fix-ID = ID of rigid body fix
    units value = box or lattice
```

Examples:

```
velocity all create 300.0 4928459 rot yes dist gaussian
velocity border set NULL 4.0 v_vz sum yes units box
velocity flow scale 300.0
velocity flow ramp vx 0.0 5.0 y 5 25 temp mytemp
velocity all zero linear
```

Description:

Set or change the velocities of a group of atoms in one of several styles. For each style, there are required arguments and optional keyword/value parameters. Not all options are used by each style. Each option has a default as listed below.

The *create* style generates an ensemble of velocities using a random number generator with the specified seed as the specified temperature.

The *set* style sets the velocities of all atoms in the group to the specified values. If any component is specified as NULL, then it is not set. Any of the vx,vy,vz velocity components can be specified as an equal-style or atom-style [variable](#). If the value is a variable, it should be specified as v_name, where name is the variable

name. In this case, the variable will be evaluated, and its value used to determine the velocity component.

Equal-style variables can specify formulas with various mathematical functions, and include [thermo_style](#) command keywords for the simulation box parameters or other parameters.

Atom-style variables can specify the same formulas as equal-style variables but can also include per-atom values, such as atom coordinates. Thus it is easy to specify a spatially-dependent velocity field.

The *scale* style computes the current temperature of the group of atoms and then rescales the velocities to the specified temperature.

The *ramp* style is similar to that used by the [compute temp/ramp](#) command. Velocities ramped uniformly from vlo to vhi are applied to dimension vx, or vy, or vz. The value assigned to a particular atom depends on its relative coordinate value (in dim) from clo to chi. For the example above, an atom with y-coordinate of 10 (1/4 of the way from 5 to 25), would be assigned a x-velocity of 1.25 (1/4 of the way from 0.0 to 5.0). Atoms outside the coordinate bounds (less than 5 or greater than 25 in this case), are assigned velocities equal to vlo or vhi (0.0 or 5.0 in this case).

The *zero* style adjusts the velocities of the group of atoms so that the aggregate linear or angular momentum is zero. No other changes are made to the velocities of the atoms. If the *rigid* option is specified (see below), then the zeroing is performed on individual rigid bodies, as defined by the [fix rigid or fix rigid/small](#) commands. In other words, zero linear will set the linear momentum of each rigid body to zero, and zero angular will set the angular momentum of each rigid body to zero. This is done by adjusting the velocities of the atoms in each rigid body.

All temperatures specified in the velocity command are in temperature units; see the [units](#) command. The units of velocities and coordinates depend on whether the *units* keyword is set to *box* or *lattice*, as discussed below.

For all styles, no atoms are assigned z-component velocities if the simulation is 2d; see the [dimension](#) command.

The keyword/value option pairs are used in the following ways by the various styles.

The *dist* option is used by *create*. The ensemble of generated velocities can be a *uniform* distribution from some minimum to maximum value, scaled to produce the requested temperature. Or it can be a *gaussian* distribution with a mean of 0.0 and a sigma scaled to produce the requested temperature.

The *sum* option is used by all styles, except *zero*. The new velocities will be added to the existing ones if sum = yes, or will replace them if sum = no.

The *mom* and *rot* options are used by *create*. If mom = yes, the linear momentum of the newly created ensemble of velocities is zeroed; if rot = yes, the angular momentum is zeroed.

The *temp* option is used by *create* and *scale* to specify a [compute](#) that calculates temperature in a desired way. If this option is not specified, *create* and *scale* calculate temperature using a compute that is defined as follows:

```
compute velocity_temp group-ID temp
```

where group-ID is the same ID used in the velocity command. i.e. the group of atoms whose velocity is being altered. This compute is deleted when the velocity command is finished. See the [compute temp](#) command for details. If the computed temperature should have degrees-of-freedom removed due to fix constraints (e.g. SHAKE or rigid-body constraints), then the appropriate fix command must be specified before the velocity command is issued.

velocity command

The *loop* option is used by *create* in the following ways.

If loop = all, then each processor loops over all atoms in the simulation to create velocities, but only stores velocities for atoms it owns. This can be a slow loop for a large simulation. If atoms were read from a data file, the velocity assigned to a particular atom will be the same, independent of how many processors are being used. This will not be the case if atoms were created using the [create_atoms](#) command, since atom IDs will likely be assigned to atoms differently.

If loop = local, then each processor loops over only its atoms to produce velocities. The random number seed is adjusted to give a different set of velocities on each processor. This is a fast loop, but the velocity assigned to a particular atom will depend on which processor owns it. Thus the results will always be different when a simulation is run on a different number of processors.

If loop = geom, then each processor loops over only its atoms. For each atom a unique random number seed is created, based on the atom's xyz coordinates. A velocity is generated using that seed. This is a fast loop and the velocity assigned to a particular atom will be the same, independent of how many processors are used. However, the set of generated velocities may be more correlated than if the *all* or *local* options are used.

Note that the *loop geom* option will not necessarily assign identical velocities for two simulations run on different machines. This is because the computations based on xyz coordinates are sensitive to tiny differences in the double-precision value for a coordinate as stored on a particular machine.

The *rigid* option only has meaning when used with the *zero* style. It allows specification of a fix-ID for one of the [rigid-body fix](#) variants which defines a set of rigid bodies. The zeroing of linear or angular momentum is then performed for each rigid body defined by the fix, as described above.

The *units* option is used by *set* and *ramp*. If units = box, the velocities and coordinates specified in the velocity command are in the standard units described by the [units](#) command (e.g. Angstroms/fmsec for real units). If units = lattice, velocities are in units of lattice spacings per time (e.g. spacings/fmsec) and coordinates are in lattice spacings. The [lattice](#) command must have been previously used to define the lattice spacing.

Restrictions: none

Related commands:

[fix shake](#), [lattice](#)

Default:

The option defaults are dist = uniform, sum = no, mom = yes, rot = no, temp = full style on group-ID, loop = all, and units = lattice. The rigid option is not defined by default.

write_data command

Syntax:

```
write_data file keyword value ...
```

- file = name of data file to write out
- zero or more keyword/value pairs may be appended
- keyword = *pair*

```
  pair value = ii or ij
      ii = write one line of pair coefficient info per atom type
      ij = write one line of pair coefficient info per IJ atom type pair
```

Examples:

```
write_data data.polymer
write_data data.*
```

Description:

Write a data file in text format of the current state of the simulation. Data files can be read by the [read data](#) command to begin a simulation. The [read_data](#) command also describes their format.

Similar to [dump](#) files, the data filename can contain a "*" wild-card character. The "*" is replaced with the current timestep value.

IMPORTANT NOTE: The write-data command is not yet fully implemented in two respects. First, most pair styles do not yet write their coefficient information into the data file. This means you will need to specify that information in your input script that reads the data file, via the [pair_coeff](#) command. Second, a few of the [atom styles](#) (body, ellipsoid, line, tri) that store auxiliary "bonus" information about aspherical particles, do not yet write the bonus info into the data file. Both these functionalities will be added to the write_data command later.

Because a data file is in text format, if you use a data file written out by this command to restart a simulation, the initial state of the new run will be slightly different than the final state of the old run (when the file was written) which was represented internally by LAMMPS in binary format. A new simulation which reads the data file will thus typically diverge from a simulation that continued in the original input script.

If you want to do more exact restarts, using binary files, see the [restart](#), [write_restart](#), and [read_restart](#) commands. You can also convert binary restart files to text data files, after a simulation has run, using the [-r command-line switch](#).

IMPORTANT NOTE: Only limited information about a simulation is stored in a data file. For example, no information about atom [groups](#) and [fixes](#) are stored. [Binary restart files](#) store more information.

Bond interactions (angle, etc) that have been turned off by the [fix shake](#) or [delete_bonds](#) command will be written to a data file as if they are turned on. This means they will need to be turned off again in a new run after the data file is read.

Bonds that are broken (e.g. by a bond-breaking potential) are not written to the data file. Thus these bonds will not exist when the data file is read.

The *pair* keyword lets you specify in what format the pair coefficient information is written into the data file. If the value is specified as *ii*, then one line per atom type is written, to specify the coefficients for each of the I=J interactions. This means that no cross-interactions for I != J will be specified in the data file and the pair style will apply its mixing rule, as documented on individual pair_style doc pages. Of course this behavior can be overridden in the input script after reading the data file, by specifying additional pair_coeff commands for any desired I,J pairs.

If the value is specified as *ij*, then one line of coefficients is written for all I,J pairs where I <= J. These coefficients will include any specific settings made in the input script up to that point. The presence of these I != J coefficients in the data file will effectively turn off the default mixing rule for the pair style. Again, the coefficient values in the data file can can be overridden in the input script after reading the data file, by specifying additional pair_coeff commands for any desired I,J pairs.

Restrictions:

This command requires inter-processor communication to migrate atoms before the data file is written. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses initialized, etc).

Related commands:

read_data, write_restart

Default:

The option defaults are pair = ii.

write_dump command

Syntax:

```
write_dump group-ID style file dump-args modify dump_modify-args
```

- group-ID = ID of the group of atoms to be dumped
- style = any of the supported [dump styles](#)
- file = name of file to write dump info to
- dump-args = any additional args needed for a particular [dump style](#)
- modify = all args after this keyword are passed to [dump_modify](#) (optional)
- dump-modify-args = args for [dump_modify](#) (optional)

Examples:

```
write_dump all atom dump.atom
write_dump subgroup atom dump.run.bin
write_dump all custom dump.myforce.* id type x y vx fx
write_dump flow custom dump.%.myforce id type c_myF[3] v_ke modify sort id
write_dump all xyz system.xyz modify sort id elements O H
write_dump all image snap*.jpg type type size 960 960 modify backcolor white
write_dump all image snap*.jpg element element &
   bond atom 0.3 shiny 0.1 ssao yes 6345 0.2 size 1600 1600  &
   modify backcolor white element C C O H N C C C O H H S O H
```

Description:

Dump a single snapshot of atom quantities to one or more files for the current state of the system. This is a one-time immediate operation, in contrast to the [dump](#) command which will will set up a dump style to write out snapshots periodically during a running simulation.

The syntax for this command is mostly identical to that of the [dump](#) and [dump_modify](#) commands as if they were concatenated together, with the following exceptions: There is no need for a dump ID or dump frequency and the keyword *modify* is added. The latter is so that the full range of [dump_modify](#) options can be specified for the single snapshot, just as they can be for multiple snapshots. The *modify* keyword separates the arguments that would normally be passed to the *dump* command from those that would be given the *dump_modify*. Both support optional arguments and thus LAMMPS needs to be able to cleanly separate the two sets of args.

Note that if the specified filename uses wildcard characters "*" or "%", as supported by the [dump](#) commmand, they will operate in the same fashion to create the new filename(s). Normally, [dump image](#) files require a filename with a "*" character for the timestep. That is not the case for the write_dump command; no wildcard "*" character is necessary.

Restrictions:

All restrictions for the [dump](#) and [dump_modify](#) commands apply to this command as well, with the exception of the [dump image](#) filename not requiring a wildcard "*" character, as noted above.

Since dumps are normally written during a [run](#) or [energy minimization](#), the simulation has to be ready to run before this command can be used. Similarly, if the dump requires information from a compute, fix, or variable, the information needs to have been calculated for the current timestep (e.g. by a prior run), else LAMMPS will generate an error message.

For example, it is not possible to dump per-atom energy with this command before a run has been performed, since no energies and forces have yet been calculated. See the [variable](#) doc page sectinn on Variable Accuracy for more information on this topic.

Related commands:

[dump](#), [dump image](#), [dump_modify](#)

Default:

The defaults are listed on the doc pages for the [dump](#) and [dump image](#) and [dump_modify](#) commands.

write_restart command

Syntax:

```
write_restart file keyword value ...
```

- file = name of file to write restart information to
- zero or more keyword/value pairs may be appended
- keyword = *fileper* or *nfile*

```
  fileper arg = Np
      Np = write one file for every this many processors
    nfile arg = Nf
      Nf = write this many files, one from each of Nf processors
```

Examples:

```
write_restart restart.equil
write_restart poly.%.* nfile 10
```

Description:

Write a binary restart file of the current state of the simulation.

During a long simulation, the restart command is typically used to output restart files periodically. The write_restart command is useful after a minimization or whenever you wish to write out a single current restart file.

Similar to dump files, the restart filename can contain two wild-card characters. If a "*" appears in the filename, it is replaced with the current timestep value. If a "%" character appears in the filename, then one file is written by each processor and the "%" character is replaced with the processor ID from 0 to P-1. An additional file with the "%" replaced by "base" is also written, which contains global information. For example, the files written for filename restart.% would be restart.base, restart.0, restart.1, ... restart.P-1. This creates smaller files and can be a fast mode of output and subsequent input on parallel machines that support parallel I/O. The optional *fileper* and *nfile* keywords discussed below can alter the number of files written.

Restart files can be read by a read_restart command to restart a simulation from a particular state. Because the file is binary (to enable exact restarts), it may not be readable on another machine. In this case, you can use the -r command-line switch to convert a restart file to a data file.

IMPORTANT NOTE: Although the purpose of restart files is to enable restarting a simulation from where it left off, not all information about a simulation is stored in the file. For example, the list of fixes that were specified during the initial run is not stored, which means the new input script must specify any fixes you want to use. Even when restart information is stored in the file, as it is for some fixes, commands may need to be re-specified in the new input script, in order to re-use that information. See the read_restart command for information about what is stored in a restart file.

The optional *nfile* or *fileper* keywords can be used in conjunction with the "%" wildcard character in the specified restart file name. As explained above, the "%" character causes the restart file to be written in pieces, one piece for each of P processors. By default P = the number of processors the simulation is running on. The *nfile* or *fileper* keyword can be used to set P to a smaller value, which can be more efficient when running on a large number of processors.

The *nfile* keyword sets P to the specified Nf value. For example, if Nf = 4, and the simulation is running on 100 processors, 4 files will be written, by processors 0,25,50,75. Each will collect information from itself and the next 24 processors and write it to a restart file.

For the *fileper* keyword, the specified value of Np means write one file for every Np processors. For example, if Np = 4, every 4th processor (0,4,8,12,etc) will collect information from itself and the next 3 processors and write it to a restart file.

Restrictions:

This command requires inter-processor communication to migrate atoms before the restart file is written. This means that your system must be ready to perform a simulation before using this command (force fields setup, atom masses initialized, etc).

Related commands:

restart, read_restart, write_data

Default: none