Xpetra

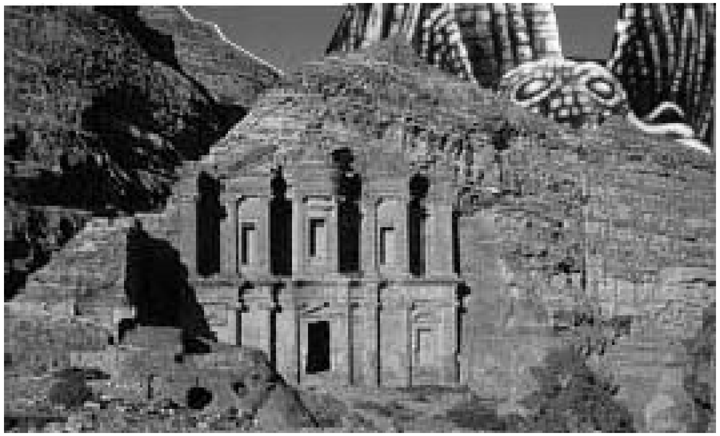
Trilinos Spring Developer Meeting

May 21st - 23rd, 2012

SAND2012-4756 C

Jeremie Gaidamour

Sandia National Laboratories



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

What is Xpetra?

- Xpetra provides a single lightweight interface for applications that wish to use Epetra or Tpetra as the underlying linear algebra

- Xpetra API = Tpetra API
- Xpetra translates Tpetra-like function calls to Epetra calls
- Which library is used is specified at run time

- Initially developed for new packages
- Can also be used to slowly transition to Tpetra

Outline

- Designing linear algebra adapters
- Using Xpetra
  - Capabilities
  - Differences with Tpetra
  - Additional functionalities
  - Limitations
- Mitigating maintainability issues using code generation
- Future developments

Some background

- In Trilinos, 2 generations of packages co-exist:
  - Epetra, Amesos, Aztecoo, Ifpack, ML, Zoltan,...
  - Tpetra, Amesos2, Belos, Ifpack2, MueLu, Zoltan2,...

- No general policy concerning linear algebra adapters

- Some new packages are directly written against Tpetra
  - Application cannot switch to new package until rewritten for Tpetra
  - New packages cannot be validated by existing Epetra applications
  - Long term maintainability issues

- Some packages are compatible with both Epetra and Tpetra via specific Operator and MultiVector adapters
  => Packages define their own lin. alg. interface
  - Packages becomes totally independents of linear alg. Softwares
  - Easy to write new adapters (PETSc...), minimalistic interface
  - Complicated interactions between packages?

Adapters for MueLu

MueLu specificity:

- We are using a lot more than just Operator and MultiVector (individual matrix entries, extract diagonals, import/export,...)
- Difficult to list all the linear algebra functionality we need

Development strategy:

- Develop MueLu using Tpetra interface
- Use a Tpetra->Epetra adapters to support Epetra applications

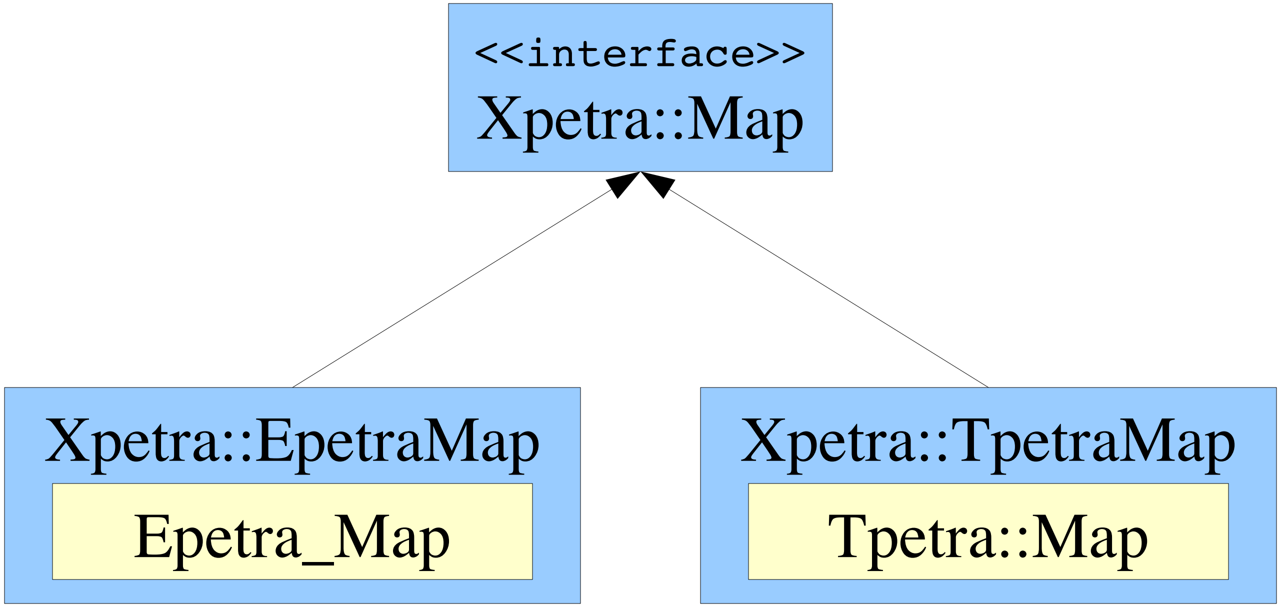
Lin Alg. Interface: Inheritance or Traits?

Inheritance:
+ Simpler
+ Compilation time, code size
+ More versatile
+ Templated on types (Scalar)

Traits:
+ Compile time polymorphism
+ No wrapped objects
+ Templated on objects (OP, MV)

How Xpetra works

- Use runtime polymorphism
- Epetra / Tpetra objects are wrapped into Xpetra objects
- Xpetra mirrors the Tpetra interface



Classes implemented in Xpetra

- Functionality implemented as needed:
  - Implementing Tpetra adapter is straightforward
  - More work involved in Epetra adapter

- *Partially* implemented:
  - Map
  - Vector
  - MultiVector
  - RowGraph
  - CrsGraph
  - CrsMatrix
  - Import/Export
  - DistObject

- Is there any functionality missing for your use cases?

Converting an application to Xpetra

Transition from Tpetra to Xpetra:

- Using Xpetra *almost* as easy as replacing 'T' by 'X':

```
#include <Tpetra_Map.hpp> -> #include <Xpetra_Map.hpp>
Tpetra::CrsMatrix<SC,LO>  -> Xpetra::CrsMatrix<SC,LO>
```

- But...
  - Xpetra is only a *subset* of Tpetra
  - Minor adjustments are needed

API differences between Tpetra and Xpetra

- Object instantiation:
  - Xpetra::Map, Vector, ... are abstract classes.

```
RCP<Tpetra::Vector<SC, LO, GO, NO> > X =
            rcp(new Tpetra::Vector<SC, LO, GO, NO>(map));
```

vs.

```
#include <Xpetra_VectorFactory.hpp>
RCP<Xpetra::Vector<SC, LO, GO, NO> > X =
            Xpetra::VectorFactory<SC, LO, GO, NO>::Build(map);
```

Tpetra provides some non-member functions to instantiate objects:

- `Tpetra::createContigMap`
- `Tpetra::createCrsMatrix`

... but this has to be extended for all classes and constructors to be used in place of Xpetra “factories”

API differences between Tpetra and Xpetra

- Maps:
  - Xpetra::Map constructors take an extra input argument:

```
Xpetra::UnderlyingLib lib = Xpetra::UseTpetra;
RCP<Xpetra::Map<...> > map =
    Xpetra::MapFactory<...>::Build(lib, numGlobalElements, comm);
```

  - Tpetra or Epetra?

```
map->getLib();
```

- Misc:
  - Some methods return `RCP<>&` instead of `RCP<>`

`getComm()`, `getMap()`, `getNode()`,…

Example: Tpetra code

xpetra/example/Simple/

```
#include <Tpetra_Map.hpp>
#include <Tpetra_CrsMatrix.hpp>
#include <Tpetra_Vector.hpp>
#include <Tpetra_MultiVector.hpp>



int main(int argc, char *argv[]) {
  GlobalOrdinal numGlobalElements = 256; // problem size


  Teuchos::GlobalMPISession mpiSession(&argc, &argv, NULL);
  RCP<const Teuchos::Comm<int> > comm = Teuchos::DefaultComm<int>::getComm();



  RCP<const Tpetra::Map<LocalOrdinal, GlobalOrdinal> > map =
             Tpetra::createUniformContigMap<LocalOrdinal, GlobalOrdinal>(numGlobalElements, comm);

  const size_t numMyElements = map->getNodeNumElements();
  Teuchos::ArrayView<const GlobalOrdinal> myGlobalElements = map->getNodeElementList();

  RCP<Tpetra::CrsMatrix<Scalar, LocalOrdinal, GlobalOrdinal> > A =
             rcp(new Tpetra::CrsMatrix<Scalar, LocalOrdinal, GlobalOrdinal>(map, 3));

  for (size_t i = 0; i < numMyElements; i++) {
    if (myGlobalElements[i] == 0) {
      A->insertGlobalValues(myGlobalElements[i],
                            Teuchos::tuple<GlobalOrdinal>(myGlobalElements[i], myGlobalElements[i] +1),
                            Teuchos::tuple<Scalar> (2.0, -1.0));
    }
    else if (myGlobalElements[i] == numGlobalElements - 1) { /* [...] */ }
    else { /* [...] */ }
  }

  A->fillComplete();

  return EXIT_SUCCESS;
}
```

Example: Xpetra code

xpetra/example/Simple/

```
#include <Xpetra_Map.hpp>
#include <Xpetra_CrsMatrix.hpp>
#include <Xpetra_Vector.hpp>
#include <Xpetra_MultiVector.hpp>

#include <Xpetra_MapFactory.hpp>
#include <Xpetra_CrsMatrixFactory.hpp>

int main(int argc, char *argv[]) {
  GlobalOrdinal numGlobalElements = 256; // problem size


  Teuchos::GlobalMPISession mpiSession(&argc, &argv, NULL);
  RCP<const Teuchos::Comm<int> > comm = Teuchos::DefaultComm<int>::getComm();

  Xpetra::UnderlyingLib lib = Xpetra::UseTpetra;

  RCP<const Xpetra::Map<LocalOrdinal, GlobalOrdinal> > map =
             Xpetra::MapFactory<LocalOrdinal, GlobalOrdinal>::createUniformContigMap(lib, numGlobalElements, comm);

  const size_t numMyElements = map->getNodeNumElements();
  Teuchos::ArrayView<const GlobalOrdinal> myGlobalElements = map->getNodeElementList();

  RCP<Xpetra::CrsMatrix<Scalar, LocalOrdinal, GlobalOrdinal> > A =
             Xpetra::CrsMatrixFactory<Scalar, LocalOrdinal, GlobalOrdinal>::Build(map, 3);

  for (size_t i = 0; i < numMyElements; i++) {
    if (myGlobalElements[i] == 0) {
      A->insertGlobalValues(myGlobalElements[i],
                            Teuchos::tuple<GlobalOrdinal>(myGlobalElements[i], myGlobalElements[i] +1),
                            Teuchos::tuple<Scalar> (2.0, -1.0));
    }
    else if (myGlobalElements[i] == numGlobalElements - 1) { /* [...] */ }
    else { /* [...] */ }
  }

  A->fillComplete();

  return EXIT_SUCCESS;
}
```

Xpetra +{E,T}petra

- Wrapping {E,T}petra objects
  - Xpetra constructors:

```
RCP<Tpetra::Map<LO> > tMap;
Xpetra::TpetraMap<LO> xMap(tMap);
```

  - toXpetra() helper functions:

```
RCP<Tpetra::Map<LO> > tMap;
RCP<Xpetra::Map<LO> > xMap = toXpetra(tMap);
```

- Getting underlying {E,T}petra objects
  - toTpetra() / toEpetra() functions:

```
RCP<Tpetra::Map<LO> > tMap = toTpetra(tMap);
```

Helper functions will throw an exception if the conversion fails.

Xpetra Additions

- Hiding templates arguments (Easy Trilinos?)

```
#include <Xpetra_Map.hpp>
#include <Xpetra_UseShortNames.hpp>// == typedef Xpetra::Map<LocalOrdinal,
                                   //                  GlobalOrdinal, Node> Map;

RCP<Map> map = ... vs. RCP<Map<LO,GO,NO> >
```

- Also works inside of templated classes:

```
#include <Xpetra_Map.hpp>
template <class Scalar, class LocalOrdinal, class GlobalOrdinal,
          class Node,   class LocalMatOps>
class MyClass {
#include "MueLu_UseShortNames.hpp"
  /* [...] */
};
```

Xpetra Additions

- Forward declaration headers (_fwd.hpp)

  Use `#include <Xpetra_Map_fwd.hpp>` in `_decl.hpp`

- Teuchos::CommandLineProcessor runtime option:
  --linAlgebra=Epetra/Tpetra

```
Teuchos::CommandLineProcessor clp(false);
Xpetra::Parameters xpetraParameters(clp);
map = MapFactory::Build(xpetraParameters.getLib(), ...)
```

- Embedded Teuchos::TimeMonitor to compare Epetra/Tpetra (disabled by default)

Adapter overhead costs

Costs of using Xpetra:

- Virtual methods: V-Table lookup
- Methods using {E,T}petra objects as input:
  - require a dynamic cast (unwrap input parameters)
- Methods returning a {E,T}petra object:
  - require an Xpetra constructor call (wrap output parameter)

Remarks:

- Calling Tpetra methods via Xpetra is a simple operation

- Some Epetra calls involves converting array elements (rare)

  Ex: CrsMatrix constructor:

  ```
  const ArrayRCP<const size_t> & numEntriesPerRowToAlloc
  ```

  vs.

  ```
  const int NumEntiresPerRowToAlloc
  ```

Xpetra limitations

- Tpetra/Epetra objects cannot be used together

  Example:

```
RCP<Xpetra::Map<LO> > tMap; // Tpetra map
RCP<Xpetra::Map<LO> > eMap; // Epetra map
xCrsMatrix->fillComplete(tMap, eMap); // will throw an exception
```

- Xpetra does not provide an *Epetra to Tpetra* adapter

- Xpetra is not a minimalistic interface

  ie: adding support to other lin. alg. packages (ie: PETSc) is difficult

Xpetra limitations

- Error handling: getting a consistent behavior between Epetra and Tpetra adapters is arduous
- How to deal with Tpetra / Epetra fundamental differences?

  ex: `fillComplete()` / `resumeFill()`
- How to implement Tpetra templated methods?
- Output of `describe()` methods differs

Minimizing maintenance issues

Problems:

- Xpetra is a lot of code! (4600 lines)
- Tpetra is under active development

Testing:

- Ideally, Tpetra unit tests can be translated to Xpetra. In practice, not that simple:
  - Missing functions in Xpetra
  - Handling correctly Epetra errors is hard

Maintainability:

- Scripts generates most of the Xpetra library
  - 360 lines of python
  - Allows to distinguish easily straightforward code / handwritten code
  - Xpetra evolves with Tpetra
  - Pick up interface documentation from Tpetra

Xpetra scripts

Xpetra is fairly easy to modify:

- Script input:
  - Doxygen's XML output of Tpetra
  - Configuration file (.conf)

    ```
    skip=function1;function2
    fillComplete = FillComplete
    ```

  - Template (.tmpl) with hand coded features

- Script generates headers files:

```
CrsMatrix:
    virtual void fillComplete(OptimizeOption os=DoOptimizeStorage)= 0;

TpetraCrsMatrix:
    void fillComplete(OptimizeOption os=DoOptimizeStorage)
            { mtx_->fillComplete(toTpetra(os)); }

EpetraCrsMatrix:
    void fillComplete(OptimizeOption os=DoOptimizeStorage)
          { mtx_->FillComplete(toEpetra(os)); }
```

Future developments

- XpetraExt
- Functions to convert Tpetra objects to Epetra (Irina Kalashnikova)
- Matrix loader abstraction (Andy Salinger)
- Additional abstraction layer for Crs/Vbr matrices (for MueLu)

Current status

- Awaiting copyright

- In preCopyright/muelu/xpetra

```
-D Trilinos_EXTRA_REPOSITORIES="preCopyrightTrilinos" \
-D Trilinos_ENABLE_MueLu:BOOL=ON \
```

- Will become a separate package when moved to the main repository

- Epetra and Tpetra are only optional dependencies