Trilinos Data Services: Then, Now, Tomorrow

Michael Heroux
Trilinos Common Language: Petra

- Petra provides a “common language” for distributed linear algebra objects (operator, matrix, vector)

- Petra\(^1\) provides distributed matrix and vector services.
- Exists in basic form as an object model:
  - Describes basic user and support classes in UML, independent of language/implementation.
  - Describes objects and relationships to build and use matrices, vectors and graphs.
  - Has 2 implementations under development.

\(^1\)Petra is Greek for “foundation”.
Petra Implementations

- **Epetra (Essential Petra):**
  - Current production version.
  - Restricted to real, double precision arithmetic.
  - Uses stable core subset of C++ (circa 2000).
  - Interfaces accessible to C and Fortran users.

- **Tpetra (Templated Petra):**
  - Next generation C++ version.
  - Templated scalar and ordinal fields.
  - Uses namespaces, and STL: Improved usability/efficiency.
  - Builds on top of Kokkos manycore node library.
Perform redistribution of distributed objects:
- Parallel permutations.
- "Ghosting" of values for local computations.
- Collection of partial results from remote processors.

Base Class for All Distributed Objects:
- Performs all communication.
- Requires Check, Pack, Unpack methods from derived class.

Describes layout of distributed objects:
- Vectors: Number of vector entries on each processor and global ID.

Dense Distributed Vector and Matrices:
- Simple local data structure.
- BLAS-able, LAPACK-able.
- "Ghostable", redistributable.
- RTOp-able.

Graph class for structure-only computations:
- Reusable matrix structure.
- Pattern-based preconditioners.
- Pattern-based load balancing tools.
- Redistribution of matrices, vectors, etc...

Supports construction of pre-recorded "plan" for data-driven communications.
Examples:
- Gathering/scatter of off-processor x/y values when computing y = Ax.
- Gathering overlap rows for Overlapping Schwarz.
- Redistribution of matrices, vectors, etc.

Describes layout of distributed objects:
- Vectors: Number of vector entries on each processor and global ID.

Peta Object Model

Abstract Interface to Parallel Machine
- Shameless mimic of MPI interface.
- Keeps MPI dependence to a single class (through all of Trilinos!).
- Allows trivial serial implementation.
- Opens door to novel parallel libraries (shmem, UPC, etc...)

Abstract Interface for Sparse All-to-All Communication
- Supports construction of pre-recorded "plan" for data-driven communications.
Examples:
- Gathering/scatter of off-processor x/y values when computing y = Ax.
- Gathering overlap rows for Overlapping Schwarz.
- Redistribution of matrices, vectors, etc.

Graph class for structure-only computations:
- Reusable matrix structure.
- Pattern-based preconditioners.
- Pattern-based load balancing tools.
- Redistribution of matrices, vectors, etc...

Basic sparse matrix class:
- Flexible construction process.
- Arbitrary entry placement on parallel machine.

Dense Distributed Vector and Matrices:
- Simple local data structure.
- BLAS-able, LAPACK-able.
- "Ghostable", redistributable.
- RTOp-able.
Kokkos: Node-level Data Classes

- Manycore/Accelerator data structures & kernels
- Epetra is MPI-only, or MPI+OMP, Tpetra is MPI+X.
- Kokkos Arrays.
  - Simple multi-dimensional arrays.
  - User specifies dimensions and size. Library handles all else.
  - Very good general performance.
- Pretty-good-kernel (PGK) library:
  - Node-level threaded (X) and vector (Y) sparse and dense kernels.
  - Plug replaceable with vendor-optimized libraries.
- Implement Petra Object Model at Node level:
  - Comm, Map/Perm, Vector/Multivector, RowMatrix, Operator.
Epetra Package

Linear Algebra Package

http://trilinos.sandia.gov/packages/epetra/
Typical Flow of Epetra Object Construction

- Construct Comm
  - Any number of Comm objects can exist.
  - Comms can be nested (e.g., serial within MPI).

- Construct Map
  - Maps describe parallel layout.
  - Maps typically associated with more than one comp object.
  - Two maps (source and target) define an export/import object.

- Construct x
- Construct b
- Construct A

- Computational objects.
- Compatibility assured via common map.
int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv); // Initialize MPI, MpiComm
    Epetra_MpiComm Comm(MPI_COMM_WORLD);

    // ***** Create x and b vectors *****
    Epetra_Vector x(Map); Epetra_Vector b(Map);
    b.Random(); // Fill RHS with random #s
    // ***** Create Linear Problem *****
    Epetra_LinearProblem problem(&A, &x, &b);
    // ***** Create/define AztecOO instance, solve *****
    AztecOO solver(problem);
    solver.SetAztecOption(AZ_precond, AZ_Jacobi);
    solver.Iterate(1000, 1.0E-8);

    // ***** Report results, finish ****************************
    cout << "Solver performed " << solver.NumIters() << " iterations."
         << " Norm of true residual = "
         << solver.TrueResidual() << endl;
    MPI_Finalize();
    return 0;
}
Details about Epetra Maps

- Note: Focus on Maps (not BlockMaps).
- Getting beyond standard use case…

- Note: All of the concepts presented here for Epetra carry over to Tpetra!
1-to-1 Maps

- **1-to-1 map** (defn): A map is 1-to-1 if each GID appears only once in the map (and is therefore associated with only a single processor).

- Certain operations in parallel data repartitioning require 1-to-1 maps. Specifically:
  - The source map of an import must be 1-to-1.
  - The target map of an export must be 1-to-1.
  - The domain map of a 2D object must be 1-to-1.
  - The range map of a 2D object must be 1-to-1.
2D Objects: Four Maps

- **Epetra 2D objects:**
  - CrsMatrix, FECrsMatrix
  - CrsGraph
  - VbrMatrix, FEVbrMatrix
- **Have four maps:**
  - **RowMap:** On each processor, the GIDs of the **rows** that processor will “manage”.
  - **ColMap:** On each processor, the GIDs of the **columns** that processor will “manage”.
  - **DomainMap:** The layout of domain objects (the $x$ vector/multivector in $y=Ax$).
  - **RangeMap:** The layout of range objects (the $y$ vector/multivector in $y=Ax$).

**Typically a 1-to-1 map**

**Typically NOT a 1-to-1 map**

**Must be 1-to-1 maps!!!**
Sample Problem

\[
\begin{bmatrix}
y \\
y_1 \\
y_2 \\
y_3 \\
\end{bmatrix} =
\begin{bmatrix}
A \\
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2 \\
\end{bmatrix}
\begin{bmatrix}
x \\
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix}
\]
Case 1: Standard Approach

- First 2 rows of $A$, elements of $y$ and elements of $x$, kept on PE 0.
- Last row of $A$, element of $y$ and element of $x$, kept on PE 1.

<table>
<thead>
<tr>
<th>PE 0 Contents</th>
<th>PE 1 Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = \begin{bmatrix} y_1 \ y_2 \end{bmatrix}$, $A = \begin{bmatrix} 2 &amp; -1 &amp; 0 \ -1 &amp; 2 &amp; -1 \end{bmatrix}$, $x = \begin{bmatrix} x_1 \ x_2 \end{bmatrix}$</td>
<td>$y = [y_3]$, $A = [0 &amp; -1 &amp; 2]$, $x = [x_3]$</td>
</tr>
</tbody>
</table>

- RowMap = \{0, 1\}
- ColMap = \{0, 1, 2\}
- DomainMap = \{0, 1\}
- RangeMap = \{0, 1\}

- RowMap = \{2\}
- ColMap = \{1, 2\}
- DomainMap = \{2\}
- RangeMap = \{2\}

Notes:
- Rows are wholly owned.
- RowMap=DomainMap=RangeMap (all 1-to-1).
- ColMap is NOT 1-to-1.
- Call to FillComplete: $A$.FillComplete(); // Assumes
Case 2: Twist 1

- First 2 rows of $A$, first element of $y$ and last 2 elements of $x$, kept on PE 0.
- Last row of $A$, last 2 element of $y$ and first element of $x$, kept on PE 1.

PE 0 Contents

$$y = [y_1], \ldots A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix}, \ldots x = \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}$$

- RowMap = \{0, 1\}
- ColMap = \{0, 1, 2\}
- DomainMap = \{1, 2\}
- RangeMap = \{0\}

PE 1 Contents

$$y = \begin{bmatrix} y_2 \\ y_3 \end{bmatrix}, \ldots A = [0 -1 2], \ldots x = [x_1]$$

- RowMap = \{2\}
- ColMap = \{1, 2\}
- DomainMap = \{0\}
- RangeMap = \{1, 2\}

Notes:
- Rows are wholly owned.
- RowMap is NOT = DomainMap is NOT = RangeMap (all 1-to-1).
- ColMap is NOT 1-to-1.
- Call to FillComplete:
  \texttt{A.FillComplete(DomainMap, RangeMap);}
Case 2: Twist 2

- First row of $A$, part of second row of $A$, first element of $y$ and last 2 elements of $x$, kept on PE 0.
- Last row, part of second row of $A$, last 2 element of $y$ and first element of $x$, kept on PE 1.

**PE 0 Contents**

\[
y = \begin{bmatrix} y_1 \end{bmatrix}, \ldots A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}, \ldots x = \begin{bmatrix} x_2 \\ x_3 \end{bmatrix}
\]

- RowMap = \{0, 1\}
- ColMap = \{0, 1\}
- DomainMap = \{1, 2\}
- RangeMap = \{0\}

**PE 1 Contents**

\[
y = \begin{bmatrix} y_2 \\ y_3 \end{bmatrix}, \ldots A = \begin{bmatrix} 0 & 1 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \ldots x = \begin{bmatrix} x_1 \end{bmatrix}
\]

- RowMap = \{1, 2\}
- ColMap = \{1, 2\}
- DomainMap = \{0\}
- RangeMap = \{1, 2\}

**Notes:**

- Rows are NOT wholly owned.
- RowMap is NOT = DomainMap is NOT = RangeMap (all 1-to-1).
- RowMap and ColMap are NOT 1-to-1.
- Call to FillComplete:
  
  A.FillComplete(DomainMap, RangeMap);
What does FillComplete Do?

- A bunch of stuff.
- One task is to create (if needed) import/export objects to support distributed matrix-vector multiplication:
  - If ColMap $\neq$ DomainMap, create Import object.
  - If RowMap $\neq$ RangeMap, create Export object.
- A few rules:
  - Rectangular matrices will *always* require:
    A.FillComplete(DomainMap,RangeMap);
  - DomainMap and RangeMap *must be 1-to-1.*
Typical Flow of Tpetra Object Construction

- Construct Comm
- Construct Node
- Construct Map
- Construct $x$
- Construct $b$
- Construct $A$

- Like Epetra “Comm”
- Composed with Kokkos Node

- Kokkos node. Options are:
  - pthread-based, OpenMP, CUDA, Serial, …
## Third Option: Xpetra

### Data Classes Stacks

<table>
<thead>
<tr>
<th>Epetra</th>
<th>Xpetra</th>
<th>Tpetra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Array Types</td>
<td>Manycore BLAS</td>
<td>Kokkos POM Layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Node sparse structures</td>
</tr>
<tr>
<td></td>
<td>Kokkos Array</td>
<td>User Extensions</td>
</tr>
</tbody>
</table>

**Classic Stack**

**New Stack**
#include <Teuchos_RCP.hpp>
#include <Teuchos_DefaultComm.hpp>

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

typedef double Scalar;
typedef int LocalOrdinal;
typedef int GlobalOrdinal;

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

    using Teuchos::RCP;
    using Teuchos::rcp;

    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->getNodesElementList();

    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) {
            A->insertGlobalValues(
                myGlobalElements[i],
                Teuchos::tuple<
                    GlobalOrdinal>(myGlobalElements[i] -1, myGlobalElements[i]),
                Teuchos::tuple<Scalar>(-1.0, 2.0));
        } else {
            A->insertGlobalValues(
                myGlobalElements[i],
                Teuchos::tuple<
                    GlobalOrdinal>(myGlobalElements[i] -1, myGlobalElements[i] +1),
                Teuchos::tuple<Scalar>(-1.0, 2.0, -1.0));
        }
    }

    A->fillComplete();

    return EXIT_SUCCESS;
}
typedef double Scalar;
typedef int LocalOrdinal;
typedef int GlobalOrdinal;

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

   using Teuchos::RCP;
   using Teuchos::rcp;

   Teuchos::GlobalMPI::Session 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->getNumElements();
   Teuchos::ArrayView<const GlobalOrdinal> myGlobalElements = map->getElementList();

   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) {
         A->insertGlobalValues(myGlobalElements[i],
            Teuchos::tuple<GlobalOrdinal>(myGlobalElements[i] -1, myGlobalElements[i]),
            Teuchos::tuple<Scalar> (-1.0, 2.0));
      }
      else {
         A->insertGlobalValues(myGlobalElements[i],
            Teuchos::tuple<GlobalOrdinal>(myGlobalElements[i] -1, myGlobalElements[i] +1),
            Teuchos::tuple<Scalar> (-1.0, 2.0, -1.0));
      }
   }

   A->fillComplete();

   return EXIT_SUCCESS;
}
<#include <Tpetra_Map.hpp>
<#include <Tpetra_CrsMatrix.hpp>
<#include <Tpetra_Vector.hpp>
<#include <Tpetra_MultiVector.hpp>
---
> #include <Xpetra_Map.hpp>
> #include <Xpetra_CrsMatrix.hpp>
> #include <Xpetra_Vector.hpp>
> #include <Xpetra_MultiVector.hpp>
>
> #include <Xpetra_MapFactory.hpp>
> #include <Xpetra_CrsMatrixFactory.hpp>
67c70,72
<   RCP<
< const Tpetra::Map<LO, GO> > map = Tpetra::createUniformContigMap<LO, GO>(numGlobalElements, comm);
---
>   Xpetra::UnderlyingLib lib = Xpetra::UseTpetra;
>
>   RCP<
> const Xpetra::Map<LO, GO> > map = Xpetra::MapFactory<LO, GO>::createUniformContigMap(lib, numGlobalElements, comm);
72c77
<   RCP<Tpetra::CrsMatrix<Scalar, LO, GO> > A = rcp(new Tpetra::CrsMatrix<Scalar, LO, GO>(map, 3));
---
>   RCP<Xpetra::CrsMatrix<Scalar, LO, GO> > A = Xpetra::CrsMatrixFactory<Scalar, LO, GO>::Build(map, 3);
97d101

LO – Local Ordinal
GO – Global Ordinal
Epetra, Tpetra, Xpetra?

- Epetra.
  - Brand newbie: Little or only basic C++, first time Trilinos User.
  - Well-worn path: Software robustness very high: +AztecOO, ML, …
  - Classic workstation, cluster, no GPU: MPI-only or modest OpenMP.

- Tpetra.
  - Forward looking, early adopter: Focus is on future.
  - Templated data types: Only option.
  - MPI+X, more that OpenMP: Only option.

- Xpetra.
  - Stable now, but forward looking: Almost isomorphic to Tpetra.
  - Support users of both Epetra and Tpetra: Single source for both.