Using Trilinos Linear Solvers
Outline

- General Introduction to Sparse Solvers.
- Overview of Trilinos Linear Solver Packages.
- Detailed look at Trilinos Data classes.
Sparse Direct Methods

- Construct $L$ and $U$, lower and upper triangular, resp, s.t.

$$LU = A$$

- Solve $Ax = b$:
  1. $Ly = b$
  2. $Ux = y$

- Symmetric versions: $LL^T = A$, $LDL^T$

- When are direct methods effective?
  - 1D: Always, even on many, many processors.
  - 2D: Almost always, except on many, many processors.
  - 2.5D: Most of the time.
  - 3D: Only for “small/medium” problems on “small/medium” processor counts.

- Bottom line: Direct sparse solvers should always be in your toolbox.
Sparse Direct Solver Packages

- HSL: [http://www.hsl.rl.ac.uk](http://www.hsl.rl.ac.uk)
- MUMPS: [http://mumps.enseeiht.fr](http://mumps.enseeiht.fr)
- Pardiso: [http://www.pardiso-project.org](http://www.pardiso-project.org)
- PaStiX: [http://pastix.gforge.inria.fr](http://pastix.gforge.inria.fr)
- SuiteSparse: [http://www.cise.ufl.edu/research/sparse/SuiteSparse](http://www.cise.ufl.edu/research/sparse/SuiteSparse)
- UMFPACK: [http://www.cise.ufl.edu/research/sparse/umfpack/](http://www.cise.ufl.edu/research/sparse/umfpack/)
- Trilinos/Amesos/Amesos2: [http://trilinos.org](http://trilinos.org)

Notes:
- All have threaded parallelism.
- All but SuiteSparse and UMFPACK have distributed memory (MPI) parallelism.
- MUMPS, PaStiX, SuiteSparse, SuperLU, Trilinos, UMFPACK are freely available.
- HSL, Pardiso, WSMP are available freely, with restrictions.
- Some research efforts on GPUs, unaware of any products.

Emerging hybrid packages:
- PDSLin – Sherry Li.
- HIPS – Gaidamour, Henon.
- Trilinos/ShyLU – Rajamanickam, Boman, Heroux.
Other Sparse Direct Solver Packages

- “Legacy” packages that are open source but not under active development today.
  - TAUCS: http://www.tau.ac.il/~stoledo/taucs/
  - PSPASES: http://www-users.cs.umn.edu/~mjoshi/pspases/
  - BCSLib: http://www.boeing.com/phantom/bcslib/

- Eigen http://eigen.tuxfamily.org
  - Newer, active, but sequential only (for sparse solvers).
  - Sparse Cholesky (including LDL^T), Sparse LU, Sparse QR.
  - Wrappers to quite a few third-party sparse direct solvers.
Emerging Trend in Sparse Direct

- New work in low-rank approximations to off-diagonal blocks.
- Typically:
  - Off-diagonal blocks in the factorization stored as dense matrices.
- New:
  - These blocks have low rank (up to the accuracy needed for solution).
  - Can be represented by approximate SVD.
- Still uncertain how broad the impact will be.
  - Will rank-$k$ SVD continue to have low rank for hard problems?
- Potential: Could be breakthrough for extending sparse direct method to much larger 3D problems.
Iterative Methods

- Given an initial guess for $x$, called $x^{(0)}$, ($x^{(0)} = 0$ is acceptable) compute a sequence $x^{(k)}$, $k = 1, 2, ...$ such that each $x^{(k)}$ is “closer” to $x$.

- Definition of “close”:
  - Suppose $x^{(k)} = x$ exactly for some value of $k$.
  - Then $r^{(k)} = b - Ax^{(k)} = 0$ (the vector of all zeros).
  - And $\text{norm}(r^{(k)}) = \sqrt{<r^{(k)}, r^{(k)}>} = 0$ (a number).
  - For any $x^{(k)}$, let $r^{(k)} = b - Ax^{(k)}$
  - If $\text{norm}(r^{(k)}) = \sqrt{<r^{(k)}, r^{(k)}>}$ is small ($< 1.0\text{E-6}$ say) then we say that $x^{(k)}$ is close to $x$.
  - The vector $r$ is called the residual vector.
Sparse Iterative Solver Packages

- hypre: [https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html](https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html)
- Paralution: [http://www.paralution.com](http://www.paralution.com) (Manycore; GPL/Commercial license)
- HSL: [http://www.hsl.rl.ac.uk](http://www.hsl.rl.ac.uk) (Academic/Commercial License)
- Eigen [http://eigen.tuxfamily.org](http://eigen.tuxfamily.org) (Sequential CG, BiCGSTAB, ILUT/Sparskit)
- Sparskit: [http://www-users.cs.umn.edu/~saad/software](http://www-users.cs.umn.edu/~saad/software)

Notes:

- There are many other efforts, but I am unaware of any that have a broad user base like hypre, PETSc and Trilinos.
- Sparskit, and other software by Yousef Saad, is not a product with a large official user base, but these codes appear as embedded (serial) source code in many applications.
- PETSc and Trilinos support threading, distributed memory (MPI) and growing functionality for accelerators.
- Many of the direct solver packages support some kind of iteration, if only iterative refinement.
# Which Type of Solver to Use?

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D</td>
<td>Direct</td>
<td>Often tridiagonal (Thomas alg, periodic version).</td>
</tr>
<tr>
<td>2D very easy</td>
<td>Iterative</td>
<td>If you have a good initial guess, e.g., transient simulation.</td>
</tr>
<tr>
<td>2D otherwise</td>
<td>Direct</td>
<td>Almost always better than iterative.</td>
</tr>
<tr>
<td>2.5D</td>
<td>Direct</td>
<td>Example: shell problems. Good ordering can keep fill low.</td>
</tr>
<tr>
<td>3D easy</td>
<td>Iterative</td>
<td>Simple preconditioners: diagonal scaling. CG or BiCGSTAB.</td>
</tr>
<tr>
<td>3D harder</td>
<td>Iterative</td>
<td>Swap Prec: IC, ILU (with domain decomposition if parallel).</td>
</tr>
<tr>
<td>3D hard</td>
<td>Iterative</td>
<td>Swap Iterative Method: GMRES (without restart if possible).</td>
</tr>
<tr>
<td>3D + large</td>
<td>Iterative</td>
<td>Add multigrid, geometric or algebraic.</td>
</tr>
</tbody>
</table>
# Trilinos Package Summary

<table>
<thead>
<tr>
<th>Objective</th>
<th>Package(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discretizations</td>
<td>Meshing &amp; Discretizations: STKMesh, Intrepid, Pamgen, Sundance, Mesquite</td>
</tr>
<tr>
<td></td>
<td>Time Integration: Rythmos</td>
</tr>
<tr>
<td>Methods</td>
<td>Automatic Differentiation: Sacado</td>
</tr>
<tr>
<td></td>
<td>Mortar Methods: Moertel</td>
</tr>
<tr>
<td>Services</td>
<td>Linear algebra objects: Epetra, Tpetra</td>
</tr>
<tr>
<td></td>
<td>Interfaces: Xpetra, Thyra, Stratimikos, RTOp, FEI, Shards</td>
</tr>
<tr>
<td></td>
<td>Load Balancing: Zoltan, Isorropia, Zoltan2</td>
</tr>
<tr>
<td></td>
<td>“Skins”: PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika</td>
</tr>
<tr>
<td></td>
<td>Utilities, I/O, thread API: Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx</td>
</tr>
<tr>
<td>Solvers</td>
<td>Iterative linear solvers: AztecOO, Belos, Komplex</td>
</tr>
<tr>
<td></td>
<td>Direct sparse linear solvers: Amesos, Amesos2, ShyLU</td>
</tr>
<tr>
<td></td>
<td>Incomplete factorizations: AztecOO, IFPACK, Ifpack2</td>
</tr>
<tr>
<td></td>
<td>Multilevel preconditioners: ML, CLAPS, MueLu</td>
</tr>
<tr>
<td></td>
<td>Direct dense linear solvers: Epetra, Teuchos, Pliris</td>
</tr>
<tr>
<td></td>
<td>Iterative eigenvalue solvers: Anasazi</td>
</tr>
<tr>
<td></td>
<td>Block preconditioners: Meros, Teko</td>
</tr>
<tr>
<td></td>
<td>Nonlinear solvers: NOX, LOCA</td>
</tr>
<tr>
<td></td>
<td>Optimization: MOOCHO, Aristos, TriKota, Globipack, Optipack</td>
</tr>
<tr>
<td></td>
<td>Stochastic PDEs: Stokhos</td>
</tr>
</tbody>
</table>
ShyLU (Scalable Hybrid LU) is hybrid

- In the mathematical sense (direct + iterative) for robustness.
- In the parallel programming sense (MPI + Threads) for scalability.

Robust than simple preconditioners and scalable than direct solvers.

ShyLU is a subdomain solver where a subdomain is not limited to one MPI process.

Will be part of Trilinos. In precopyright Trilinos for Sandia users.

Results: Over 19x improvement in the simulation time for large Xyce circuits.
Amesos2

- Direct Solver interface for the Tpetra Stack.
- Typical Usage:
  - `preOrder()`,
  - `symbolicFactorization()`,
  - `numericFactorization()`,
  - `solve()`.
- Easy to support new solvers (Current support for all the SuperLU variants).
- Easy to support new multivectors and sparse matrices.
- Can support third party solver specific parameters with little changes.
- Available in the current release of Trilinos.
AztecOO

- Iterative linear solvers: CG, GMRES, BiCGSTAB,…
- Incomplete factorization preconditioners

- Aztec was Sandia’s workhorse solver:
  - Extracted from the MPSalsa reacting flow code
  - Installed in dozens of Sandia apps
  - 1900+ external licenses

- AztecOO improves on Aztec by:
  - Using Epetra objects for defining matrix and vectors
  - Providing more preconditioners & scalings
  - Using C++ class design to enable more sophisticated use

- AztecOO interface allows:
  - Continued use of Aztec for functionality
  - Introduction of new solver capabilities outside of Aztec

Developers: Mike Heroux, Alan Williams, Ray Tuminaro
Belos

- Next-generation linear iterative solvers

- Decouples algorithms from linear algebra objects
  - Linear algebra library has full control over data layout and kernels
  - Improvement over AztecOO, which controlled vector & matrix layout
  - Essential for hybrid (MPI+X) parallelism

- Solves problems that apps really want to solve, faster:
  - Multiple right-hand sides: \( AX = B \)
  - Sequences of related systems: \( (A + \Delta A_k) X_k = B + \Delta B_k \)

- Many advanced methods for these types of systems
  - Block & pseudoblock solvers: GMRES & CG
  - Recycling solvers: GCRODR (GMRES) & CG
  - “Seed” solvers (hybrid GMRES)
  - Block orthogonalizations (TSQR)

- Supports arbitrary & mixed precision, complex, …
- If you have a choice, pick Belos over AztecOO

Developers: Heidi Thornquist, Mike Heroux, Chris Baker, Mark Hoemmen
Ifpack(2): Algebraic preconditioners

- Preconditioners:
  - Overlapping domain decomposition
  - Incomplete factorizations (within an MPI process)
  - (Block) relaxations & Chebyshev
- Accepts user matrix via abstract matrix interface
- Use \{E,T\}petra for basic matrix / vector calculations
- Perturbation stabilizations & condition estimation
- Can be used by all other Trilinos solver packages
- Ifpack2: Tpetra version of Ifpack
  - Supports arbitrary precision & complex arithmetic
  - Path forward to hybrid-parallel factorizations

Developers: Mike Heroux, Mark Hoemmen, Siva Rajamanickam, Marzio Sala, Alan Williams, etc.
Multi-level Preconditioners

- Smoothed aggregation, multigrid, & domain decomposition
- Critical technology for scalable performance of many apps
- ML compatible with other Trilinos packages:
  - Accepts Epetra sparse matrices & dense vectors
  - ML preconditioners can be used by AztecOO, Belos, & Anasazi
- Can also be used independent of other Trilinos packages
- Next-generation version of ML: MueLu
  - Works with Epetra or Tpetra objects (via Xpetra interface)

Developers: Ray Tuminaro, Jeremie Gaidamour, Jonathan Hu, Marzio Sala, Chris Siefert
MueLu: Next-gen algebraic multigrid

- Motivation for replacing ML
  - Improve maintainability & ease development of new algorithms
  - Decouple computational kernels from algorithms
    - ML mostly monolithic (& 50K lines of code)
    - MueLu relies more on other Trilinos packages
  - Exploit Tpetra features
    - MPI+X (Kokkos programming model mitigates risk)
    - 64-bit global indices (to solve problems with >2B unknowns)
    - Arbitrary Scalar types (Tramonto runs MueLu w/ double-double)

- Works with Epetra or Tpetra (via Xpetra common interface)
- Facilitate algorithm development
  - Energy minimization methods
  - Geometric or classic algebraic multigrid; mix methods together
- Better support for preconditioner reuse
  - Explore options between “blow it away” & reuse without change

Developers: Andrey Prokopenko, Jonathan Hu, Chris Siefert, Ray Tuminaro, Tobias Wiesner
Petra Distributed Object Model
Solving $Ax = b$: Typical Petra Object Construction Sequence

- **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 computational object.
  - Two maps (source and target) define an export/import object.

- **Construct $x$**
- **Construct $b$**
- **Construct $A$**
  - Computational objects.
  - Compatibility assured via common map.
Petra Implementations

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

- **Tpetra (Templated Petra):**
  - Next-generation version
  - C++ compiler can’t be too ancient (no need for C++11 but good to have)
  - Supports arbitrary scalar and index types via templates
    - Arbitrary- and mixed-precision arithmetic
    - 64-bit indices for solving problems with >2 billion unknowns
  - Hybrid MPI / shared-memory parallel
    - Supports multicore CPU and hybrid CPU/GPU
    - Built on Kokkos manycore node library

Package leads: Mike Heroux, Mark Hoemmen (many developers)
A Simple Epetra/AztecOO Program

```c
// Header files omitted...
int main(int argc, char *argv[]) {
    Epetra_SerialComm Comm();

    // ***** Create x and b vectors *****
    Epetra_Vector x(Map);
    Epetra_Vector b(Map);
    b.Random(); // Fill RHS with random #s

    // ***** Create an Epetra_Matrix tridiag(-1,2,-1) *****
    Epetra_CrsMatrix A(Copy, Map, 3);
    double negOne = -1.0; double posTwo = 2.0;
    for (int i=0; i<NumMyElements; i++) {
        int GlobalRow = A.GRID(i);
        int RowLess1 = GlobalRow - 1;
        int RowPlus1 = GlobalRow + 1;
        if (RowLess1!=-1)
            A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowLess1);
        if (RowPlus1!=NumGlobalElements)
            A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowPlus1);
        A.InsertGlobalValues(GlobalRow, 1, &posTwo, &GlobalRow);
    }
    A.FillComplete(); // Transform from GIDs to LIDs

    // ***** 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;
}
```
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.

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

Describes layout of distributed objects:
• Vectors: Number of vector entries on each processor and global ID
• Matrices/graphs: Rows/Columns managed by a processor.
• Called "Maps" in Epetra.

Dense Distributed Vector and Matrices:
• Simple local data structure.
• BLAS-able, LAPACK-able.
• Ghostable, redistributable.
• RTOp-able.
Details about Epetra & Tpetra Maps

- Getting beyond standard use case…
1-to-1 Maps

- A map is 1-to-1 if...
  - Each global ID appears only once in the map
  - (and is thus associated with only a single process)

- Certain operations in parallel data repartitioning require 1-to-1 maps:
  - Source map of an import must be 1-to-1.
  - Target map of an export must be 1-to-1.
  - Domain map of a 2D object must be 1-to-1.
  - 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:**
  - **Row Map:** On each processor, the global IDs of the *rows* that process will “manage.”
  - **Column Map:** On each processor, the global IDs of the *columns* that process will “manage.”
  - **Domain Map:** The layout of domain objects (the $x$ (multi)vector in $y = Ax$).
  - **Range Map:** The layout of range objects (the $y$ (multi)vector in $y = Ax$).

Typically a 1-to-1 map

Typically NOT a 1-to-1 map

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

\[
\begin{bmatrix}
y_1 \\ y_2 \\ y_3
\end{bmatrix} =
\begin{bmatrix}
2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2
\end{bmatrix}
\begin{bmatrix}
\color{red}x_1 \\ \color{red}x_2 \\ \color{red}x_3
\end{bmatrix}
\]
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.

### PE 0 Contents

\[
y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \quad A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]
- RowMap = \{0, 1\}
- ColMap = \{0, 1, 2\}
- DomainMap = \{0, 1\}
- RangeMap = \{0, 1\}

### PE 1 Contents

\[
y = \begin{bmatrix} y_3 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & -1 & 2 \end{bmatrix}, \quad x = \begin{bmatrix} x_3 \end{bmatrix}
\]
- 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: \( A.\text{FillComplete}(); // \) 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

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

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

Notes:

- Rows are wholly owned.
- RowMap is NOT = DomainMap
- ColMap is NOT 1-to-1.
- Call to FillComplete:
  \[ A.\text{FillComplete(DomainMap, RangeMap)}; \]
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.

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

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);
  ```

Original Problem

\[
\begin{align*}
  y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, & \quad A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, & \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
\end{align*}
\]
What does FillComplete do?

- Signals you’re done defining matrix structure
- Does a bunch of stuff
- Creates communication patterns for distributed sparse matrix-vector multiply:
  - If ColMap \neq \text{DomainMap}, create Import object
  - If RowMap \neq \text{RangeMap}, create Export object
- A few rules:
  - Non-square matrices will always require:
    A.FillComplete(DomainMap,\text{RangeMap});
  - DomainMap and RangeMap must be 1-to-1
## Third Option: Xpetra

### Data Classes Stacks

<table>
<thead>
<tr>
<th>Xpetra</th>
<th>Epetra</th>
<th>Tpetra</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Manycore BLAS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple Array Types</td>
<td></td>
<td>Kokkos Array</td>
</tr>
</tbody>
</table>

- **Classic Stack**
- **New Stack**
Simple 1D Example in Tpetra

```cpp
#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->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) {
      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;
}
```
```plaintext
#include <Teuchos_RCP.hpp>
#include <Teuchos_DefaultComm.hpp>
#include <Teuchos_Map.hpp>
#include <Teuchos_CrsMatrix.hpp>
#include <Teuchos_Vector.hpp>
#include <Teuchos_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::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->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) {
            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], myGlobalElements[i] + 1), Teuchos::tuple<Scalar>(-1.0, 2.0, -1.0));
        }
    }

    A->fillComplete();

    return EXIT_SUCCESS;
}
```
```cpp
#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>

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);

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);
```
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.
  - If you want manycore/accelerator fill.

- **Xpetra.**
  - Stable now, but forward looking: Almost isomorphic to Tpetra.
  - Support users of both Epetra and Tpetra: Single source for both.
    - Example: Muelu.
Abstract solver interfaces
& applications
Stratimikos package

- Greek στρατηγική (strategy) + γραμμικός (linear)
- Uniform run-time interface to many different packages’
  - Linear solvers: Amesos, AztecOO, Belos, …
  - Preconditioners: Ifpack, ML, …
- Defines common interface to create and use linear solvers
- Reads in options through a Teuchos::ParameterList
  - Can change solver and its options at run time
  - Can validate options, & read them from a string or XML file
- Accepts any linear system objects that provide
  - Epetra_Operator / Epetra_RowMatrix view of the matrix
  - Vector views (e.g., Epetra_MultiVector) for right-hand side and initial guess
- Increasing support for Tpetra objects

Developers: Ross Bartlett, Andy Salinger, Eric Phipps
Stratimikos Parameter List and Sublists

<ParameterList name="Stratimikos">
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Amesos">
      <Parameter name="Solver Type" type="string" value="Klu"/>
      <ParameterList name="Amesos Settings">
        <Parameter name="MatrixProperty" type="string" value="general"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="Mumps">
      ...
    </ParameterList>
    <ParameterList name="Superludist">
      ...
    </ParameterList>
  </ParameterList>
  ...
  <ParameterList name="AztecOO">
    <ParameterList name="Forward Solve">
      <Parameter name="Max Iterations" type="int" value="400"/>
      <Parameter name="Tolerance" type="double" value="1e-06"/>
      <ParameterList name="AztecOO Settings">
        <Parameter name="Aztec Solver" type="string" value="GMRES"/>
        ...
      </ParameterList>
    </ParameterList>
    ...
  </ParameterList>
  ...
  <ParameterList name="Belos">
    ...
  </ParameterList>
  <ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="0"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="0"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="ML">
      ...
    </ParameterList>
  </ParameterList>
</ParameterList>

Top level parameters

Linear Solvers

Sublists passed on to package code.

Preconditioners

Every parameter and sublist is handled by Thyra code and is fully validated.
<ParameterList name="Stratimikos">
  <Parameter name="Linear Solver Type" type="string" value="Belos"/>
  <ParameterList name="Preconditioner Type" type="string" value="ML"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Amesos">
      <Parameter name="Solver Type" type="string" value="Klu"/>
      <ParameterList name="Amesos Settings">
        <Parameter name="MatrixProperty" type="string" value="general"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="Mumps">
      ...
    </ParameterList>
    <ParameterList name="Superludist">
      ...
    </ParameterList>
  </ParameterList>
  <ParameterList name="AztecOO">
    <ParameterList name="Forward Solve">
      <Parameter name="Max Iterations" type="int" value="400"/>
      <Parameter name="Tolerance" type="double" value="1e-06"/>
      <ParameterList name="AztecOO Settings">
        <Parameter name="Aztec Solver" type="string" value="GMRES"/>
        ...
      </ParameterList>
    </ParameterList>
    ...
    <ParameterList name="Belos">
      ...
    </ParameterList>
  </ParameterList>
  <ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="0"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="0"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="ML">
      ...
    </ParameterList>
  </ParameterList>
</ParameterList>
Parameter List Validation
Example: User misspells “Aztec Solver” as “ztec Solver”

```xml
<ParameterList>
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="AztecOO">
      <ParameterList name="Forward Solve">
        <ParameterList name="AztecOO Settings">
          <Parameter name="ztec Solver" type="string" value="GMRES"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Error message generated from PL::validateParameters(...) with exception:

Error, the parameter {name="ztec Solver",type="string",value="GMRES"}
in the parameter (sub)list "RealLinearSolverBuilder->Linear Solver Types->AztecOO->Forward Solve->AztecOO Settings"
was not found in the list of valid parameters!

The valid parameters and types are:

```plaintext
{
  "Aztec Preconditioner" : string = ilu
  "Aztec Solver" : string = GMRES
  ...
}
```
Error Messages for Improper Parameters/Sublists

Example: User specifies the wrong type for “Aztec Solver”

```xml
<ParameterList>
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="AztecOO">
      <ParameterList name="Forward Solve">
        <ParameterList name="AztecOO Settings">
          <Parameter name="Aztec Solver" type="int" value="GMRES"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Error message generated from PL::validateParameters(...) with exception:

```
Error, the parameter {paramName="Aztec Solver",type="int"}
in the sublist "DefaultRealLinearSolverBuilder->Linear Solver Types->AztecOO->Forward Solve->AztecOO Settings"
has the wrong type. The correct type is "string"!
```
Example: User specifies the wrong value for “Aztec Solver”

```
<ParameterList>
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="AztecOO">
      <ParameterList name="Forward Solve">
        <ParameterList name="AztecOO Settings">
          <Parameter name="Aztec Solver" type="string" value="GMRESS"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Error message generated from PL::validateParameters(...) with exception:

Error, the value "GMRESS" is not recognized for the parameter "Aztec Solver" in the sublist "". Valid selections include: "CG", "GMRES", "CGS", "TFQMR", "BiCGStab", "LU".
Stratimikos Details

- Stratimikos has just one primary class:
  - Stratimikos::DefaultLinearSolverBuilder
  - An instance of this class accepts a parameter list that defines:
    - Linear Solver: Amesos, AztecOO, Belos.
    - Preconditioner: Ifpack, ML, AztecOO.

- Albany, other apps:
  - Access solvers through Stratimikos.
  - Parameter list is standard XML. Can be:
    - Read from command line.
    - Read from a file.
    - Passed in as a string.
    - Defined interactively.
    - Hand coded in source code.
Trilinos provides a rich collection of linear solvers:

• Uniform access to many direct sparse solvers.

• An extensive collection of iterative methods:
  – Classic single RHS: CG, GMRES, etc.
  – Pseudo-block: Multiple independent systems.
  – Recycling: Multiple sequential RHS.
  – Block: Multiple simultaneous RHS.

• A broad set of preconditioners:
  – Domain decomposition.
  – Algebraic smoothers.
  – AMG.

• Composable, extensible framework.
  – RowMatrix and Operator base classes enable user-defined operators.
  – Multi-physic and multi-scale operators composed from Trilinos parts.

• Template features enable:
  – Variable precision, complex values.

• Significant R&D in:
  – Thread-scalable algorithms, kernels.
  – Resilient methods.