Introduction to Trilinos framework

PRACE Spring School 2012, Krakow, Poland

Michal Merta

17th May 2012

The Trilinos Project

- Trilinos = "a string of pearls"
- Software framework for the solution of large-scale engineering and scientific problems
- Open, package-based architecture – like LEGO
- Developed by Sandia National Laboratories
- Currently in version 10.10

The Trilinos Project

Primary goal

- Develop robust algorithms for scientific and engineering applications on parallel computers, and make these algorithms accessible to application developers in the most efficient way

The Trilinos Project

- Code written in C++ language
- Using modern object-oriented software design
- Focus given not only on numerical and parallel scalability but also "software scalability" – reusability and interoperability
- Allows package developers to focus only on things that are unique to their package
- Use of established software: interfaces to BLAS, LAPACK, SuperLU, Mumps, Umfpack, PETSc, ...

Licensing

- LGPL and BSD licence
- Moving toward BSD licencing for all packages (more permissive than LGPL)
- 30 of 50 packages now under BSD licence

Target platforms

Parallel machines

- Desktop
- Cray XE6, XT5
- Blue Gene
- Clusters

Parallelization techniques

- MPI
- Threads, CUDA, OpenCL

Some applications using Trilinos

- SALINAS – a massively parallel finite element code for structural dynamics and acoustics analysis
- Xyce – parallel electronic circuits simulator
- MPSalsa – reacting flow problems
- Sundance – high-performance parallel finite-element solutions of partial differential equations based on automatic differentiation of symbolic objects

Awards

- 2004 R&D 100 Award
- SC2004 HPC Software Challenge Award

Installation

- Trilinos uses CMake for configuration and installation
- Prerequisites
  - BLAS
  - LAPACK
  - (MPI)
- Not necessary to install all packages
- Current release 10.10:
  - http://trilinos.sandia.gov/download/trilinos-10.10.html

Installation

Configuration script example

```
#!/bin/bash
EXTRA_ARGS=$@

cmake \
-D CMAKE_BUILD_TYPE:STRING=DEBUG \
-D TPL_ENABLE_MPI:BOOL=ON \
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=ON \
-D Trilinos_ENABLE_Fortran:BOOL=OFF \
$EXTRA_ARGS \
../../trilinos-10.10.2-Source
```

- `make install`

Package

- Fundamental atomic unit in Trilinos
- Self-contained SW, developed autonomously
- Can interact with or depend on other package(s)
- Has its own set of requirements, development team and group of users
- Belongs to some capability area
- Provided with services of Trilinos framework

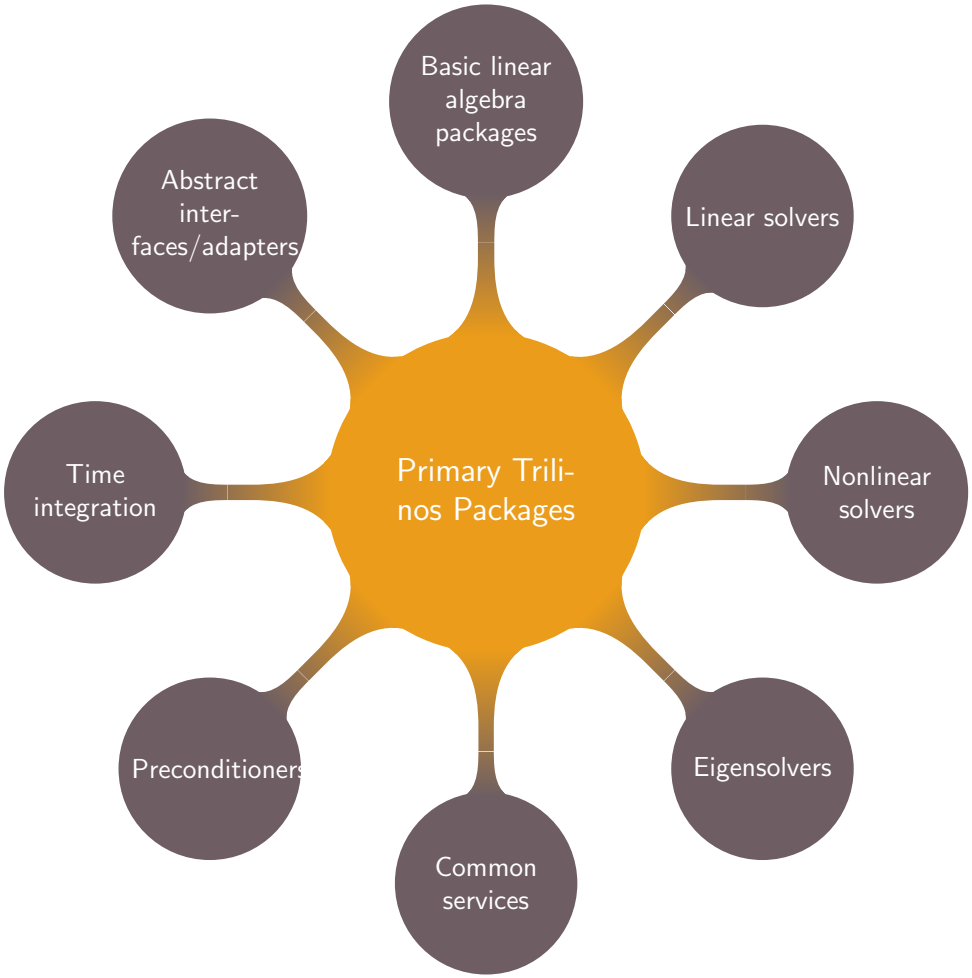
Capability areas

1. Framework & Tools
2. Software Engineering Technologies and Integration
3. I/O Support
4. Discretizations
5. Meshes, Geometry, & Load Balancing
6. Scalable Linear Algebra
7. Linear & Eigen Solvers
8. Embedded Nonlinear Analysis Tools

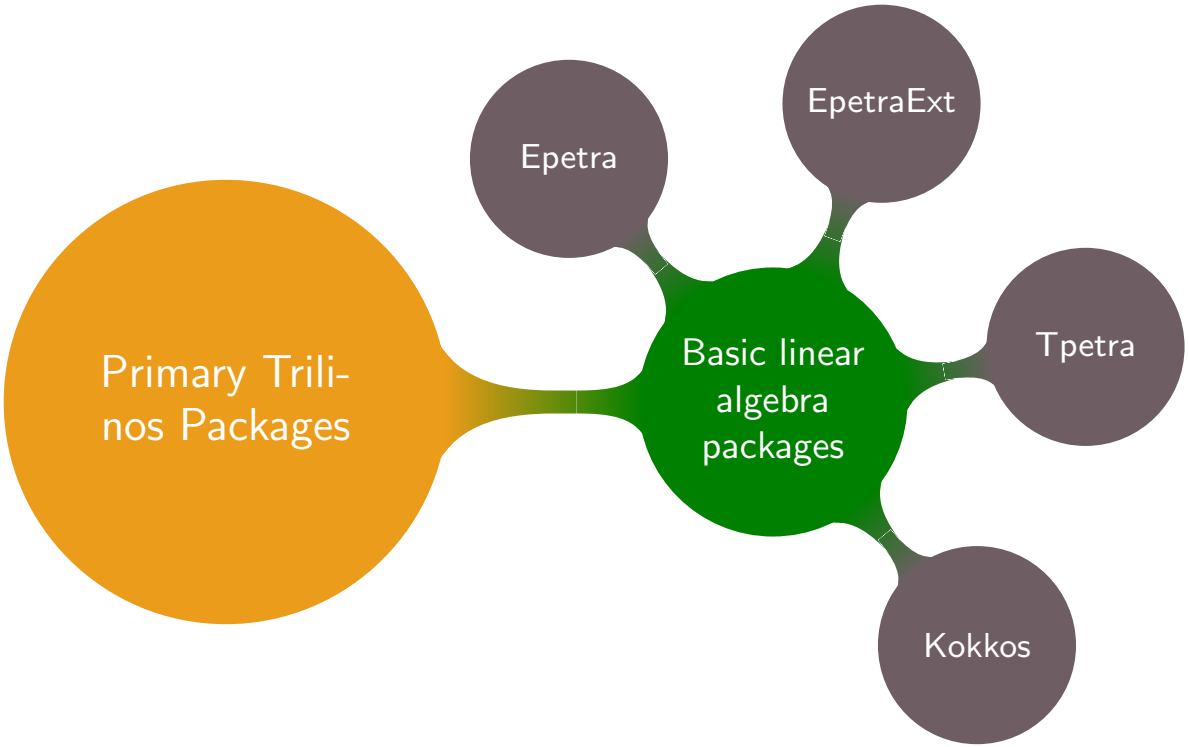
Algorithmic areas

1. Solution of linear systems with successive and simultaneous right-hand-sides
2. Iterative methods (linear and eigen solvers)
3. Incomplete factorizations
4. Multi-level preconditioners
5. Automatic differentiation
6. Data partitioning and load balancing
7. Nonlinear methods
8. Large-scale optimization
9. Time integration methods

The package tree



Basic linear algebra packages



Basic linear algebra packages

Epetra

- Core linear algebra package. Provides classes for manipulation of serial/distributed vectors, matrices, graphs etc.

EpetraExt

- Extension to Epetra. Input/output functions, interfaces to PETSc, Matlab.

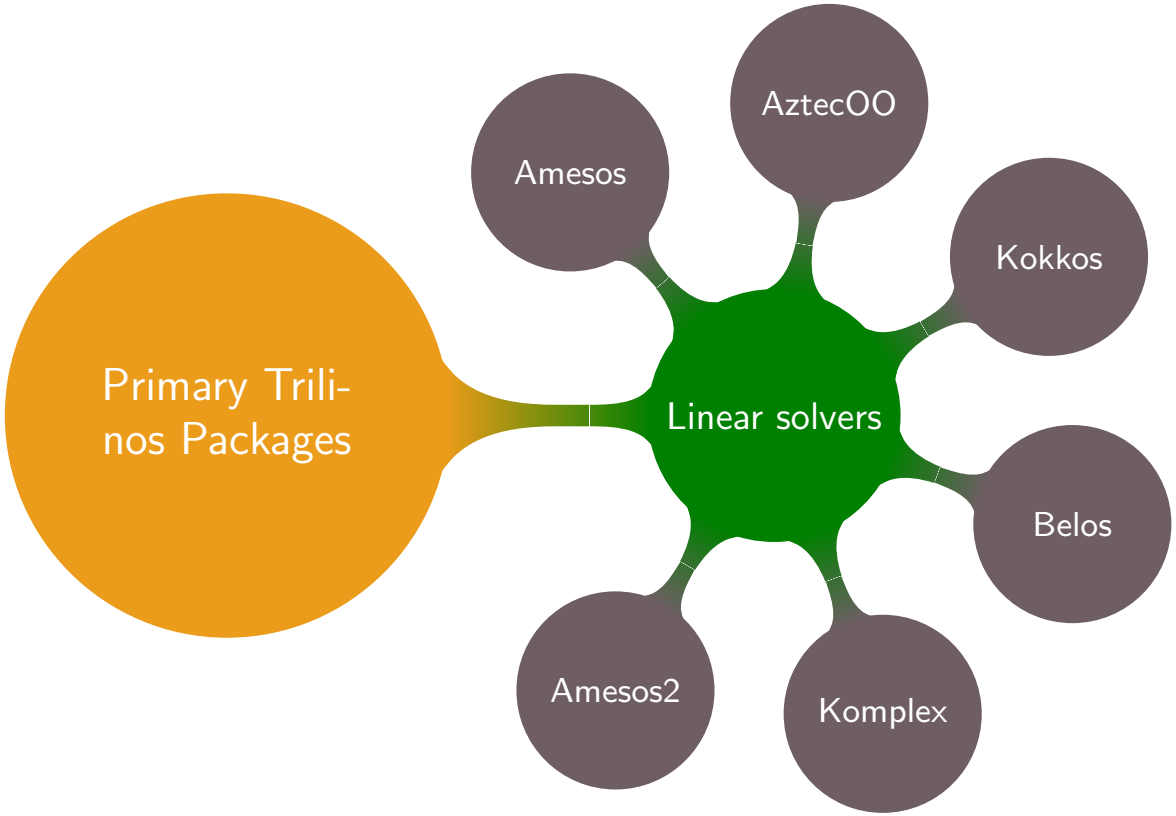
Tpetra

- Modern, templated version of Epetra.

Kokkos

- Core kernel package.

Linear solvers



Linear solvers

Epetra/Teuchos

- Provides wrappers for BLAS and LAPACK routines (solvers for dense matrices).

Amesos

- Direct solver classes. Supports use of third party direct solvers, including DSCPACK, SuperLU, SuperLUDist and UMFPACK. Compatible with Epetra.

AztecOO

- Preconditioned Krylov solver package. Solves linear systems of equations via preconditioned Krylov methods. A follow-on to the Aztec solver package.

Linear solvers

Belos

- A Greek term for "Arrow". Next-generation iterative solvers written using generic programming techniques. Has no explicit dependence on any concrete linear algebra package. Can be used with any LA library implementing the Thyra abstract interfaces.

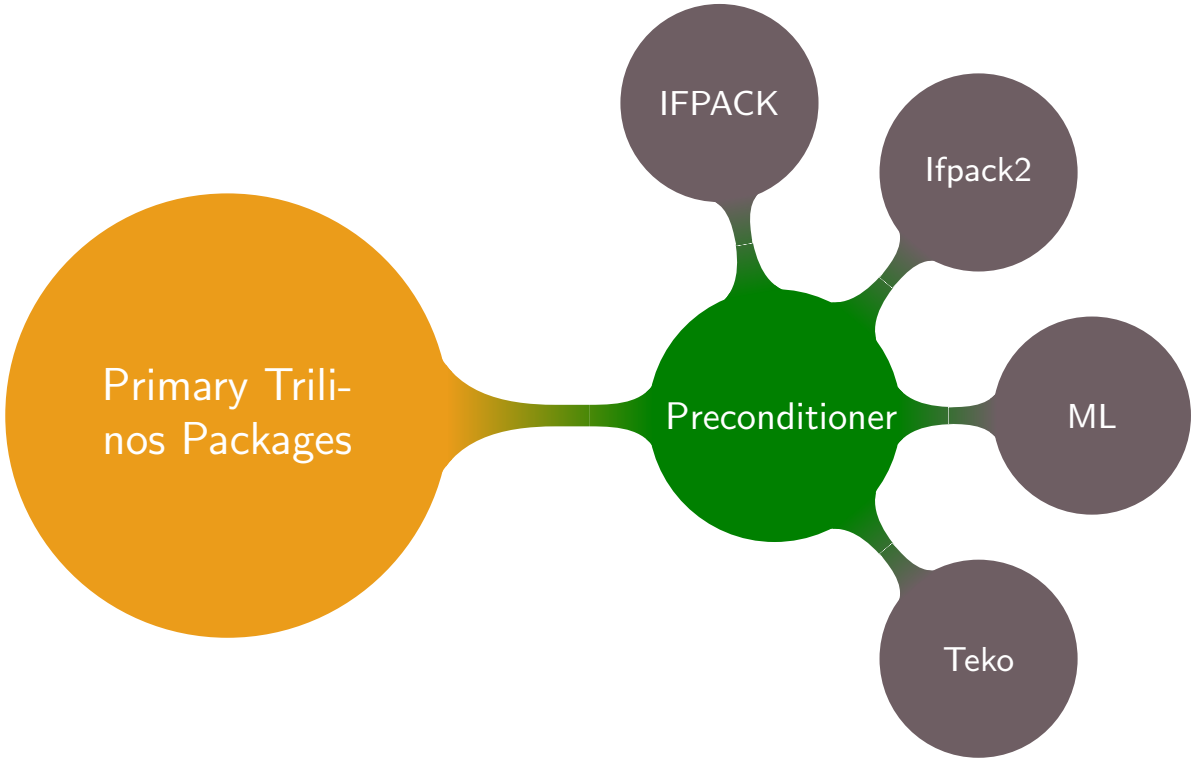
Komplex

- Complex linear solver package. Solves complex-valued linear systems via equivalent real formulations. Uses Epetra and AztecOO.

Amesos2

- One of the newest packages. Modern templated version of Amesos compatible with Tpetra.

Preconditioners



Preconditioners

IFPACK

- Distributed algebraic preconditioner package. Includes incomplete factorizations and relaxation-based preconditioners. Compatible with AztecOO and Epetra.

Ifpack2

- Templated version of Ifpack. New package.

Preconditioners

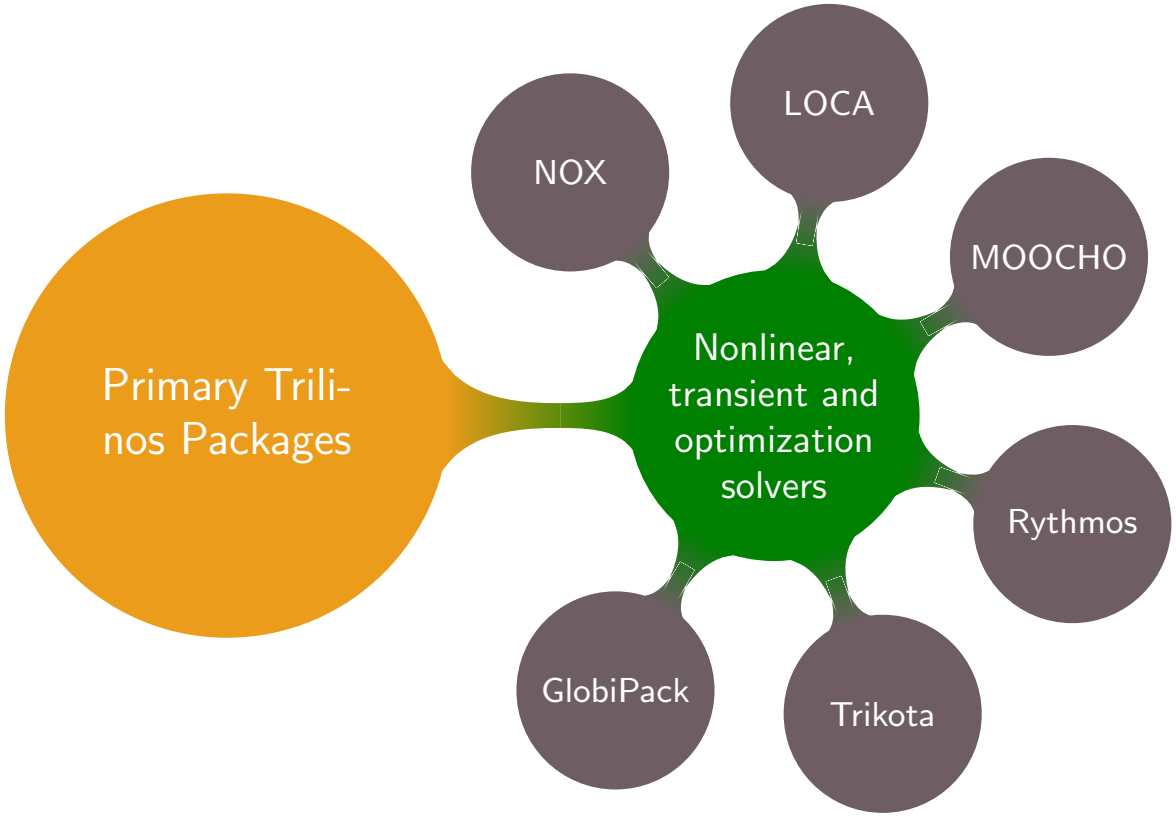
ML

- Multi-level, distributed memory algebraic preconditioners.

Teko

- Blocked and segregated preconditioning package. New package.

Nonlinear, transient and optimization solvers



Nonlinear, Transient, and Optimization Solvers

NOX

- Nonlinear solver package. Abstract and concrete classes for construction and solution of nonlinear problems.

LOCA

- Software library for performing bifurcation analysis of large-scale applications. LOCA is designed to drive application codes that use Newton's method to locate steady-state solutions to nonlinear problems.

MOOCHO

- Package for solving large-scale equality and inequality nonlinearly constrained, non-convex optimization problems using reduced-space successive quadratic programming methods.

Nonlinear, Transient, and Optimization Solvers

Rythmos

- Transient integrator for ordinary differential equations and differential-algebraic equations with support for explicit, implicit, one-step and multi-step algorithms.

Trikota

- Builds the Dakota framework underneath Trilinos as if it were a Trilinos package. Dakota contains a wide array of algorithms for optimization and uncertainity quantification.

GlobiPack

- Contains a set of interfaces and implementations for 1D globalization capabilities for nonlinear and optimization solvers that require globalization methods.

Eigensolver, automatic differentiation, DD

Eigensolvers

Anasazi

- Extensible and interoperable framework for large-scale eigenvalue algorithms. Both matrix and vectors are considered to be opaque objects—only knowledge of the matrix and vectors via elementary operations is necessary. An implementation of Anasazi is accomplished via the use of interfaces.

Automatic differentiation

Sacado

- A set of automatic differentiation tools for C++ applications. Provides templated classes for forward, reverse and Taylor mode automatic differentiation.

Domain decomposition

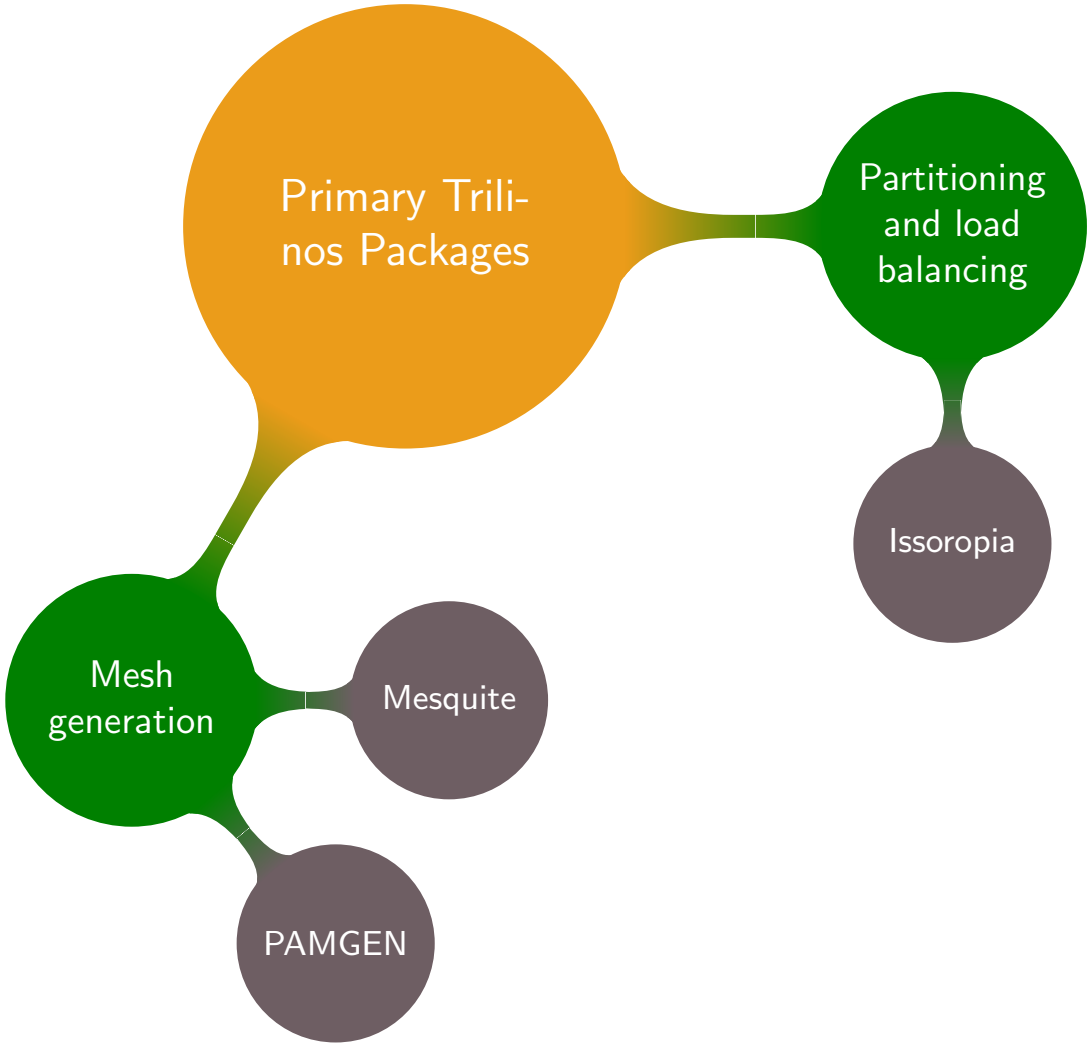
Claps

- A collection of domain decomposition preconditioners and solvers.

Moertel

- Capabilities for nonconforming mesh tying and contact formulations in 2 and 3 dimensions using Mortar methods.

Partitioning and load balancing, mesh generation



Partitioning and load balancing

Isorropia

- Partitioning and load balancing package, intended to assist with redistributing objects such as matrices and matrix-graphs in a parallel execution setting. Isorropia is primarily an interface to the Zoltan library.

Mesh generation

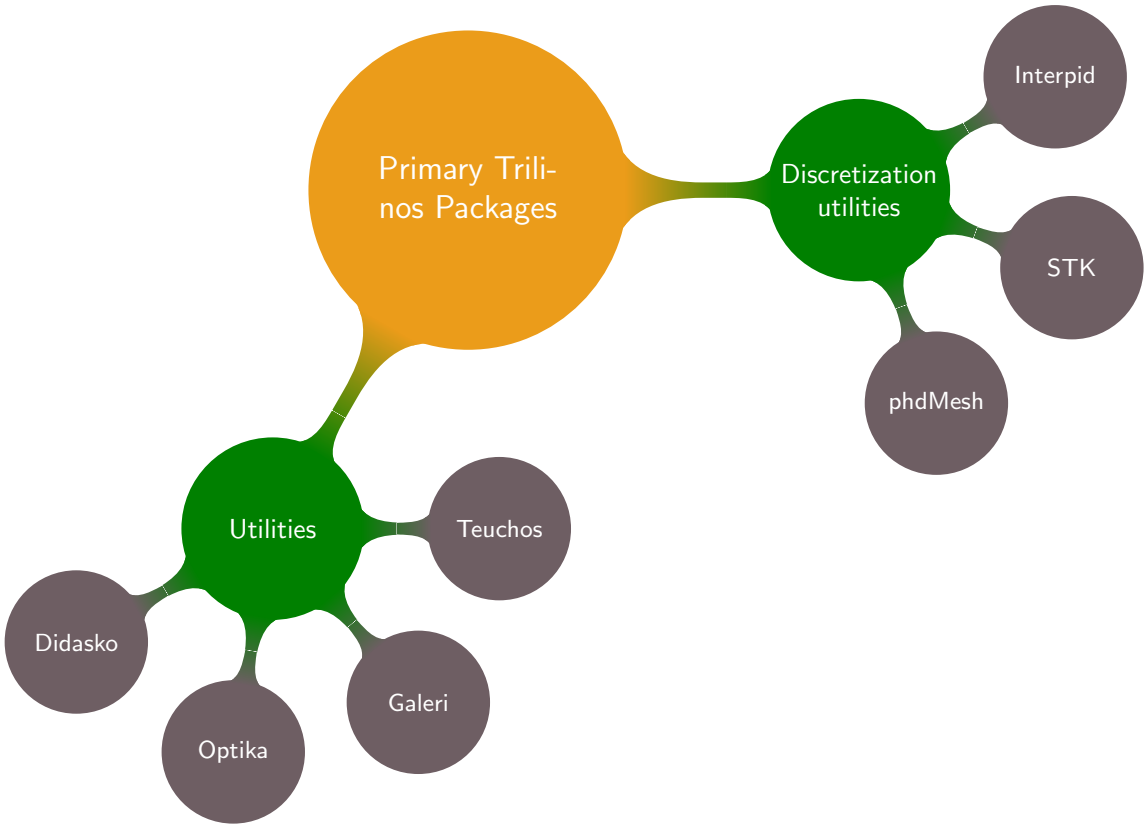
Mesquite

- Improves the quality and adapts a given mesh (using a variety of node-movement algorithms).

PAMGEN

- Creates hexahedral or quadrilateral (in 2D) finite element meshes of simple shapes (cubes and cylinders) in parallel.

Partitioning and load balancing, mesh generation



Discretization utilities

Interpid

- A library of interoperable tools for compatible discretizations of Partial Differential Equations.

STK

- Contains capabilities intended to support massively parallel multi-physics computations on dynamically changing unstructured meshes. The primary capability in the STK package is the mesh database which supports creation and manipulation of mesh entities (nodes, elements etc.).

phdMesh

- Intended to be component used within a finite element or finite volume library or code. Supports arbitrary unstructured mesh connectivity, application-defined groupings of mesh entities.

Utilities

Teuchos

- Suite of common tools for Trilinos developers.

Galeri

- Package for generating linear systems used by many of the Trilinos packages for examples and tests (like MATLAB gallery() function).

Optika

- Provides easy access to GUI input methods.

Didasko

- Instructional package, Trilinos tutorial.

Abstract interfaces and adapters

Thyra

- Abstract linear solver package. A set of interfaces and supporting code that define basic interoperability mechanisms between different types of numerical software.

ForTrilinos

- Fortran interface.

PyTrilinos

- Python interface to selected Trilinos packages.

Two generations

"Old generation packages"

- Mature, slowly becoming obsolete
- Based mainly on Epetra
- Support only 32-bit data types, `int`, `double`, `MPI`
- E.g. Epetra, AztecOO, Amesos, IFPACK

Two generations

"Next generation packages"

- Work in progress, will replace "old packages"
- Based on Tpetra, Thyra or fully generic
- Common code refactored out to Teuchos
- Arbitrary scalar and ordinal types, mixed precision, distributed, shared and hybrid parallelization
- In Trilinos 11, "new packages" will cover all basic functionality
- E.g. Tpetra, Belos, Anasazi, Ifpack2, Amesos2

Teuchos

- Suite of common tools for Trilinos developers
  - Memory management
  - Low-level math
  - Parameter handling
  - MPI support
  - XML parsers, outputting, performance monitoring, exception handling, testing, ...

Teuchos

Parameter list

- `ParameterList` – A templated parameter list mapping string to `ParameterEntry`
- `ParameterEntry` – holder for entry in `ParameterList`
- May be recursive. Can read and write to XML files.

```
Teuchos::ParameterList pl;
pl.set("Solver", "CG");
pl.set("Tolerance", 1e-3);

Teuchos::ParameterList& subPl = pl.sublist("Precond");
subPl.set("Type", "ILU");

double tol = pl.get<double>("Tolerance");
```

Teuchos

Command line processor

- `CommandLineProcessor` - processes and validates command line arguments
- Automatically generates a documented `--help` option

```
Teuchos::CommandLineProcessor CLP;

int maxiter   = 1000;
double tolerance  = 1e-6;

CLP.setOption("maxiter", &maxiter, "Number of iterations");
CLP.setOption("tolerance", &tolerance, "Solver tolerance");

CLP.parse( argc, argv );
```

```
./example --maxiter 100 --tolerance 1e-3
```

Teuchos

Reference counted pointers

- Memory management system for Trilinos packages
- `RCP` – reference-counted smart pointer class for managing dynamically allocated memory to object
- Raw C++ pointer wrapper
- Similar to Java garbage collector but more convenient for large scale parallel computing

```
class A { };

int main {
 Teuchos::RCP<A> a_null_ptr;
 Teuchos::RCP<A> a = Teuchos::rcp(new A);
 Teuchos::RCP<A> b = a;

 a->function();
}
```

Teuchos

BLAS and LAPACK interfaces

- A "thin" layer of C++ code wrapping BLAS and LAPACK routines allowing arbitrary precision arithmetic

```
#include "Teuchos_BLAS.hpp"

int main {
  Teuchos::BLAS<int, double> blas;

  const int n = 10;
  double alpha = 2.0;

  double x[ n ];
  for ( int i=0; i<n; i++ ) { x[i] = i; }

  blas.SCAL( n, alpha, x, 1 );
  int max_idx = blas.IAMAX( n, x, 1 );
}
```

Teuchos

Other low-level math

- `ScalarTraits` – define basic properties and elementary functions for various scalar data types to be used in numerical computation
- `SerialDenseMatrix` – a templated, dense rectangular matrix class providing basic operations like addition, multiplication, scaling, logical comparison, norms, etc.

Teuchos

MPI support

- `MPISession` – has methods to initialize, finalize, and query the global MPI environment.
- `MPIComm` – represents an MPI communicator
  - `gather`, `bcast`, `synchronize`, ...

Teuchos example

```
int main(int argc, char *argv[]) {
  typedef std::complex<double>                SC;
  typedef Teuchos::ScalarTraits<ST>          SCT;
  typedef SCT::magnitudeType                  MT;

  MT tol = 1.0e-5;
```

Teuchos example

```
int main(int argc, char *argv[]) {
  typedef std::complex<double>                SC;
  typedef Teuchos::ScalarTraits<ST>          SCT;
  typedef SCT::magnitudeType                  MT;

  MT tol = 1.0e-5;

  RCP<CommandLineProcessor> cmdp
      = rcp( new CommandLineProcessor() );
  cmdp->setOption("tol",
      &tol,
      "Relative residual convergence tolerance.");

  cmdp.parse(argc,argv);
```

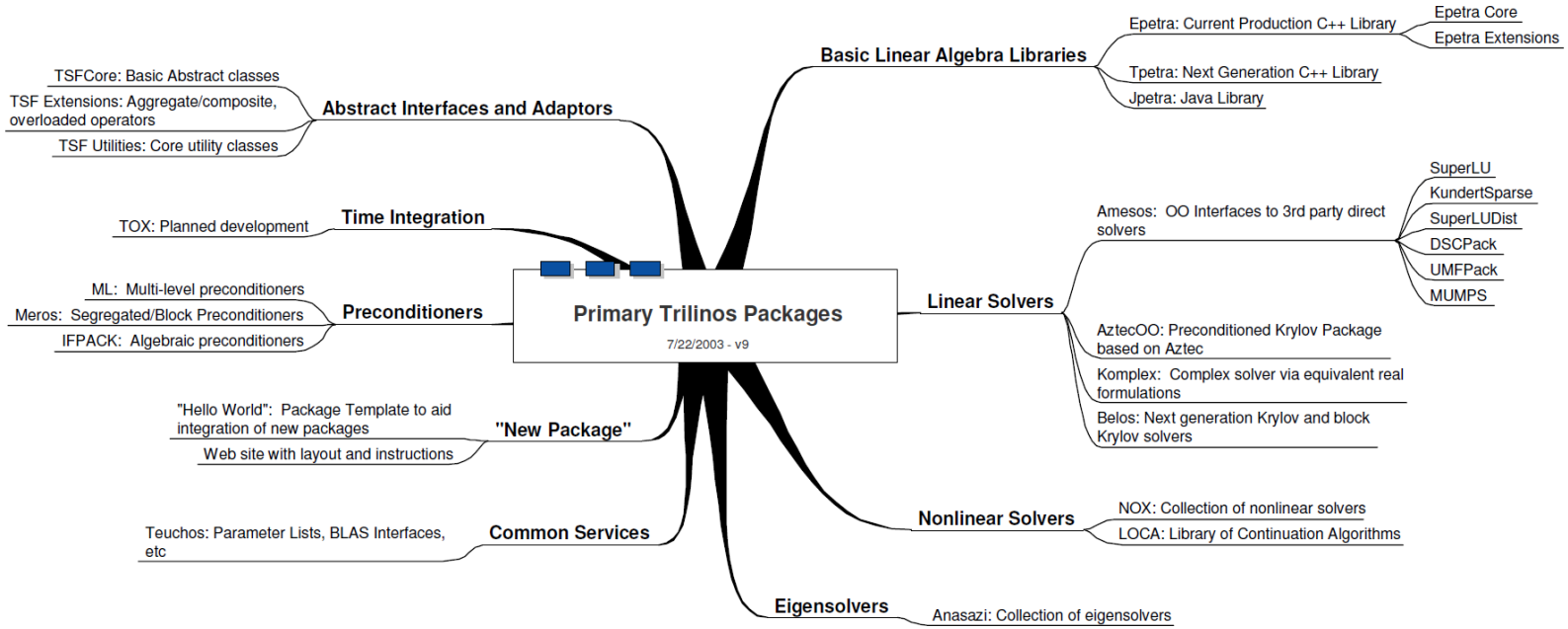
Teuchos example

```
int main(int argc, char *argv[]) {
  typedef std::complex<double>                SC;
  typedef Teuchos::ScalarTraits<ST>          SCT;
  typedef SCT::magnitudeType                  MT;

  MT tol = 1.0e-5;

  RCP<CommandLineProcessor> cmdp
      = rcp( new CommandLineProcessor() );
  cmdp->setOption("tol",
      &tol,
      "Relative residual convergence tolerance.");

  cmdp.parse(argc,argv);

  RCP<ParameterList> myPL = rcp( new ParameterList() );
  myPL->set( "Maximum Iterations", 100 );
  myPL->set( "Convergence Tolerance", tol );
```

Teuchos example

```
int main(int argc, char *argv[]) {
  typedef std::complex<double>                SC;
  typedef Teuchos::ScalarTraits<ST>          SCT;
  typedef SCT::magnitudeType                  MT;

  MT tol = 1.0e-5;

  RCP<CommandLineProcessor> cmdp
      = rcp( new CommandLineProcessor() );
  cmdp->setOption("tol",
      &tol,
      "Relative residual convergence tolerance.");

  cmdp.parse(argc,argv);

  RCP<ParameterList> myPL = rcp( new ParameterList() );
  myPL->set( "Maximum Iterations", 100 );
  myPL->set( "Convergence Tolerance", tol );

  //myPL accepted by solver
  //cmpd and myPL will be deleted automatically
  return 0;
}
```

Trilinos project overview



Petra

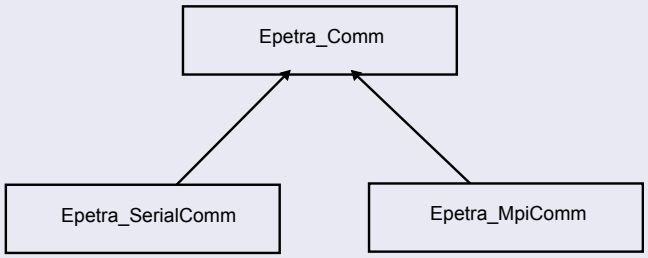
- A Greek term for 'foundation'
- Construction and manipulation with vectors, matrices and graphs
- Three implementations
  - Epetra
    - 'Essential' Petra
    - Uses only integer or real, double-precision floating point data
    - Avoids using advanced features of C++ to guarantee portability
  - Tpetra
    - Next generation of Petra
    - Templated Petra
  - Jpetra

Epetra

- Linear algebra services package
- Supports both serial and parallel computation
- Provides
  - Communicators
  - Maps
  - Vectors & multivectors
  - Operators & matrices
  - Linear problems
  - Exporters, importers
  - Serial dense matrix services (factorization, solvers)
  - Utility classes (timers, FLOPs counters, BLAS and LAPACK wrappers)
- http://trilinos.sandia.gov/packages/epetra/

Epetra Communicator

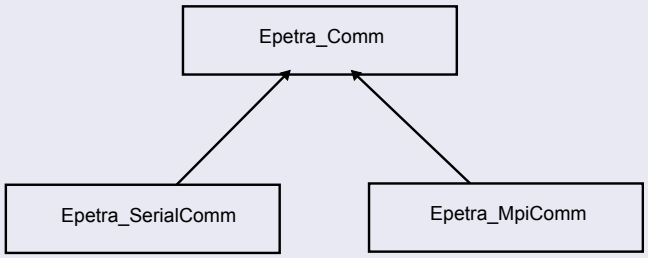
- Interface to parallel machine
- Provides user with a number of available processors, a rank of the current processor, barrier methods, basic collective operations etc.



Epetra Communicator

Epetra Communicator

- Interface to parallel machine
- Provides user with a number of available processors, a rank of the current processor, barrier methods, basic collective operations etc.




```
Epetra_SerialComm *serialComm = new Epetra_SerialComm();
Epetra_MpiComm *mpiComm = new Epetra_MpiComm(MPI_COMM_WORLD);
```

Epetra Map

Epetra Map

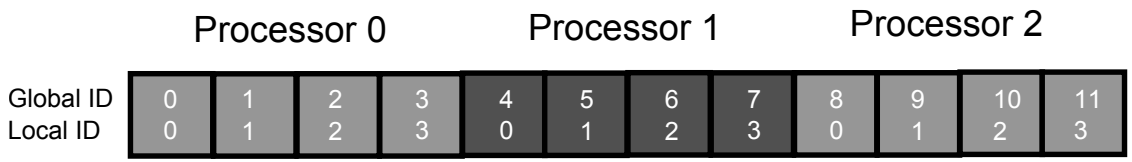
- Defines a distribution of elements of some distributed object (vector or multivector, rows of matrix) over processes
- Several ways to define a distribution
  - fully automatic
  - user-defined number of elements on each node
  - user-defined distribution of elements over nodes

Epetra Map

Epetra Map

- Defines a distribution of elements of some distributed object (vector or multivector, rows of matrix) over processes
- Several ways to define a distribution
  - fully automatic
  - user-defined number of elements on each node
  - user-defined distribution of elements over nodes

```
Epetra_Map *map1 = new Epetra_Map(NumGlobalElements, 0, Comm);
Epetra_Map *map2 = new Epetra_Map(-1, NumMyElements, 0, Comm);
Epetra_Map *map3 = new Epetra_Map(-1, NumMyElems,
                                  MyGlElems, 0, Comm);
```



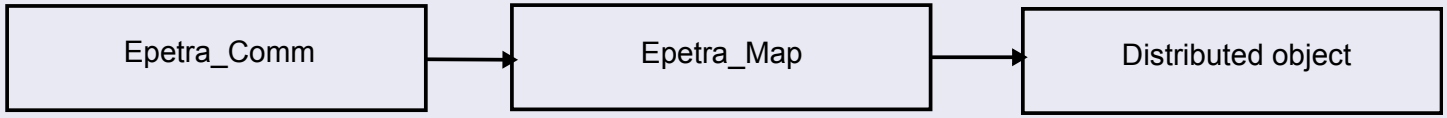
Epetra Map

Information provided by `Epetra_Map` class

- `NumGlobalElements` - total number of elements across all processes
- `NumMyElements` - the number of elements owned by the calling process
- `MyGlobalElements` - a list of length NumMyElements that contains the global element IDs of the elements owned by the calling processor.
- `IndexBase` - the base integer value for indexed array references. Typically this is 0 for C/C++ and 1 for Fortran.
- `Comm` - used communicator.
- `MinMyGID` - minimum global index value on the calling process
- etc.

Epetra Map

Common workflow



Hands-on exercise no. 1

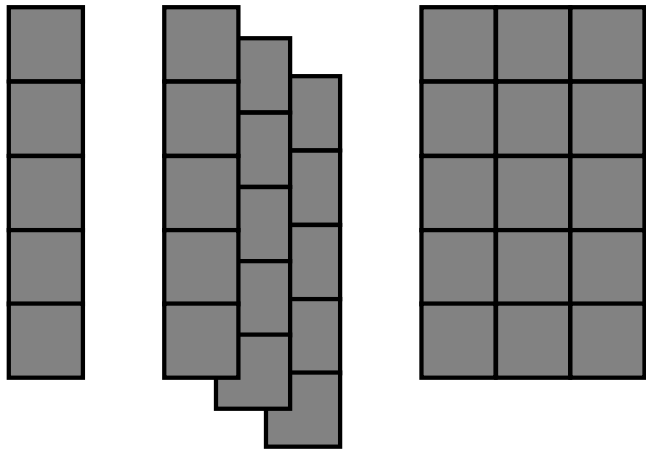
Hello World!

- This example demonstrates the usage of `Epetra_Map` class
  - For instructions see printouts.
  - To compile the example type `make EpetraInit.exe`
  - Submit the example using LoadLeveler
    `llsubmit EpetraInit.ll`
  - Solution of exercises can be found in
    `./Solutions/Exercise1`

Epetra Vectors

Vectors, multivectors and matrices

Vector	A list of double precision numbers (multivector with one vector)
Multivector	A collection of one or more vectors having the same length and distribution
Dense matrix	A special form of multi-vector such that stride in memory between any two consecutive vectors in the multi-vector is the same for all vectors.



Epetra Vectors

Epetra Multivector

- `Epetra_MultiVector` is the class for constructing and using dense multi-vectors, vectors and matrices in parallel.
  - Base class for all other classes representing vectors
  - Dimension and distribution specified by `Epetra_Comm`, `Epetra_Map` and number of vectors passed in

Epetra Vectors

Epetra Multivector

- `Epetra_MultiVector` is the class for constructing and using dense multi-vectors, vectors and matrices in parallel.
  - Base class for all other classes representing vectors
  - Dimension and distribution specified by `Epetra_Comm`, `Epetra_Map` and number of vectors passed in

Functions of `Epetra_MultiVector`

- Dot products
- Vector updates (sum, scaling...)
- Various types of norms.
- Minimum, maximum and average values
- Dense matrix-matrix multiplication (via call to `DGEMM`)
- FLOP counter

Epetra Vectors

Distributed global vs. replicated local vectors

Distributed global Multi-vector partitioned across multiple memory images associated with multiple processors.

Replicated local Some algorithms use vectors that are too small to be distributed across all processors. Replicated local vectors handle these types of situation.

Epetra Vectors

Epetra Vector

- Class representing a single dense parallel vector
- Inherits from `Epetra_MultiVector` class

```
Epetra_MpiComm Comm(MPI_COMM_WORLD);

int NumElements = 1000;

Epetra_Map Map(NumElements, 0, Comm);
Epetra_Vector x(Map);
```

Epetra Vectors

```
Epetra_MpiComm Com(MPI_COMM_WORLD);

int NumElements = 1000;
Epetra_Map Map(NumElements, 0, Comm);
int NumMyElements = Map.NumMyElements();

double *values = new double[NumMyElements];
for (int i=0; i<NumMyElements; i++)
  values[i] = (double) i;

Epetra_Vector x(Copy, Map, values);
```

Epetra Vectors

```
Epetra_MpiComm Com(MPI_COMM_WORLD);

int NumElements = 1000;
Epetra_Map Map(NumElements, 0, Comm);
int NumMyElements = Map.NumMyElements();

double *values = new double[NumMyElements];
for (int i=0; i<NumMyElements; i++)
  values[i] = (double) i;

Epetra_Vector x(Copy, Map, values);
```

Data access mode

Copy Allocates memory and makes a copy of the user-provided data.

View Creates a "view" of the user data. The user data is required to remain intact for the life of the multi-vector.

Epetra Vectors

Other Epetra vectors

Epetra_FEVector Finite element vector. Supports finite element vector assembly, sub-vectors contributions. Sub-vectors need not to be owned by calling processor.

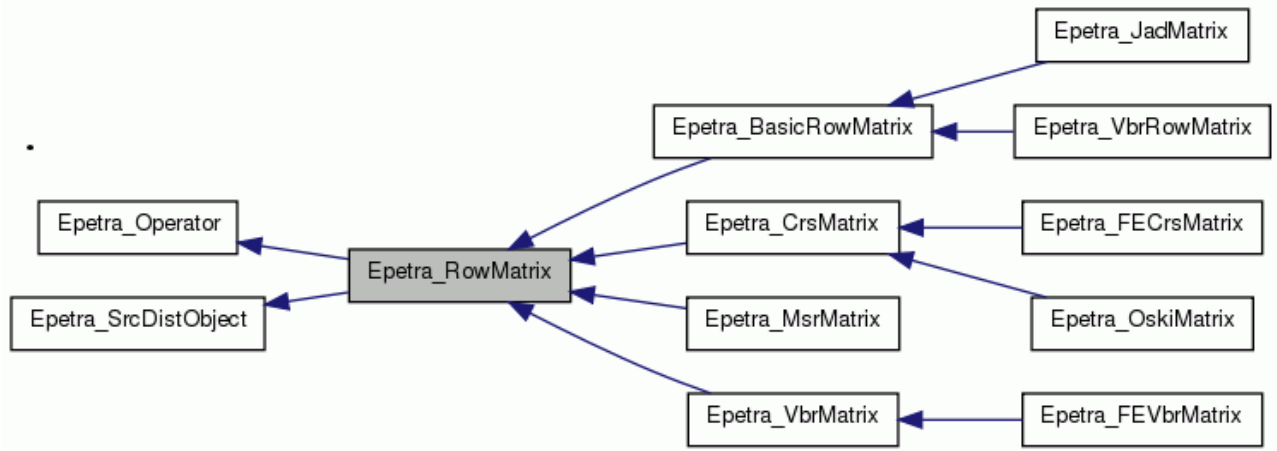
Epetra_OskiVector Class for using dense OSKI (Optimized Sparse Kernel Library) vectors on a single processor or a single core of a multiprocessor.

Hands-on exercise no. 2

Gram-Schmidt orthogonalization

- This example demonstrates the usage of `Epetra_Vector` class
  - For instructions see printouts.
  - To compile the example type `make EpetraVector.exe`
  - Submit the example using LoadLeveler
    `llsubmit EpetraVector.ll`
  - Solution of exercises can be found in
    `./Solutions/Exercise2`

Epetra Sparse Matrices



Epetra_RowMatrix

- Pure virtual class.
- Base for classes representing a sparse matrix stored in a compressed row format

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Creating `Epetra_CrsMatrix`

1. Create appropriate Epetra Communicator

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Creating `Epetra_CrsMatrix`

1. Create appropriate Epetra Communicator
2. Create Epetra Map objects

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Creating `Epetra_CrsMatrix`

1. Create appropriate Epetra Communicator
2. Create Epetra Map objects
3. Create `Epetra_CrsMatrix`

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Creating `Epetra_CrsMatrix`

1. Create appropriate Epetra Communicator
2. Create Epetra Map objects
3. Create `Epetra_CrsMatrix`
4. Fill the matrix

Epetra Sparse Matrices

`Epetra_CrsMatrix`

- Concrete implementation of `Epetra_RowMatrix` class
- Matrix stored in compressed row format
- Both serial and parallel (according to communicator)
- Distribution specified by `Epetra_Map` arguments

Creating `Epetra_CrsMatrix`

1. Create appropriate Epetra Communicator
2. Create Epetra Map objects
3. Create `Epetra_CrsMatrix`
4. Fill the matrix
5. Call `FillComplete()`

Epetra Sparse Matrices

Row/Column maps vs. domain/range maps

RowMap Defines a distribution of rows of the matrix across processes.

ColumnMap Indices of columns that are owned by given processor.

Epetra Sparse Matrices

Row/Column maps vs. domain/range maps

RowMap	Defines a distribution of rows of the matrix across processes.
ColumnMap	Indices of columns that are owned by given processor.
DomainMap	Distribution of domain objects (x vector in $y = Ax$)
RangeMap	Distribution of range objects (y vector in $y = Ax$)

Epetra Sparse Matrices

Example

```
...
Epetra_Map RowMap(NumGlobalElements, 0, Comm);

Epetra_CrsMatrix A(Copy, RowMap, 3);

// insert values to matrix
...
A.FillComplete();

Epetra_Vector x(RowMap);
Epetra_Vector y(RowMap);

A.Multiply(false, x, y);
```

Epetra Sparse Matrices

Example

```
...
Epetra_Map RowMap(NumGlobalElements, 0, Comm);

Epetra_CrsMatrix A(Copy, RowMap, 3);

// insert values to matrix
...
A.FillComplete();

Epetra_Vector x(RowMap);
Epetra_Vector y(RowMap);

A.Multiply(false, x, y);
```

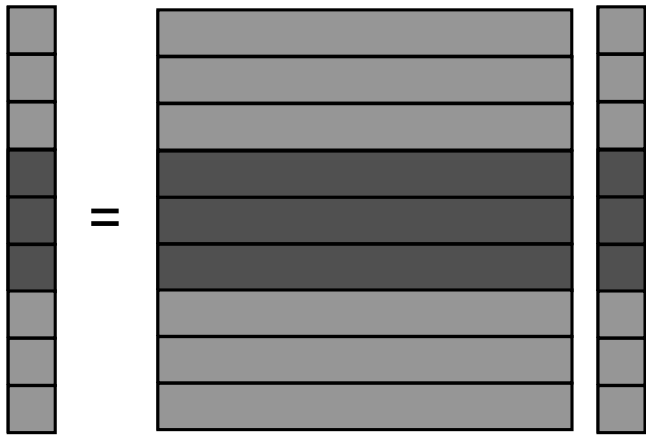
Inserting values

- `InsertGlobalValues(int GlobalRow, int NumEntries, double* Values, int* Indices)`
- `InsertMyValues(int MyRow, int NumEntries, double* Values, int* Indices)`

Epetra Sparse Matrices

`FillComplete()`

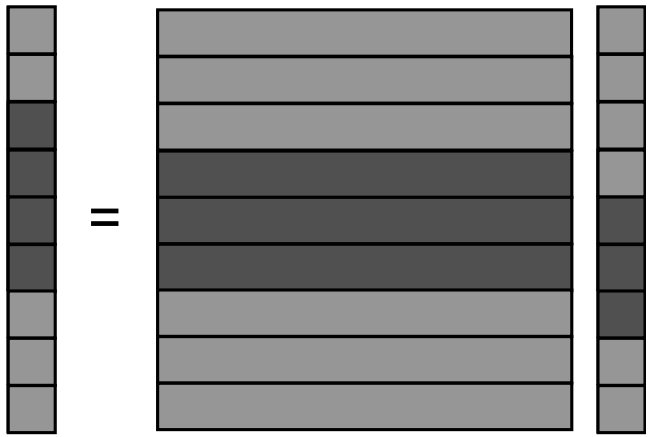
- Performs transformation from global to local indexing.
- Creates import/export objects to support distributed matrix-vector multiplication (if `ColMap`≠`DomainMap` or `RowMap`≠`RangeMap`).
- `A.FillComplete()` - if `RowMap=RangeMap=DomainMap`
- For rectangular matrices always `A.FillComplete(DomainMap, RangeMap)`



Epetra Sparse Matrices

`FillComplete()`

- Performs transformation from global to local indexing.
- Creates import/export objects to support distributed matrix-vector multiplication (if `ColMap`≠`DomainMap` or `RowMap`≠`RangeMap`).
- `A.FillComplete()` - if `RowMap=RangeMap=DomainMap`
- For rectangular matrices always `A.FillComplete(DomainMap, RangeMap)`



Epetra Sparse Matrices

Other Epetra matrix types

- `Epetra_VbrMatrix`
  - Variable block-row sparse matrix.
  - For PDE problems with more than one unknown per grid node.
- `Epetra_FECrsMatrix`
  - Epetra Finite Element matrix, supports including non-local data.

EpetraExt

- Set of extensions to Epetra
- Sparse matrix-matrix multiplication and addition
- Explicit sparse transform
- Trilinos/PETSc interface
- Epetra/Zoltan interface
- Matlab, MatrixMarket, HDF5 I/O functions
- http://trilinos.sandia.gov/packages/epetraext/

Hands-on exercise no. 3

Conjugate gradient method

- This example demonstrates the usage of `Epetra_CrsMatrix` class
  - For instructions see printouts.
  - To compile the example type `make EpetraMatrix.exe`
  - Submit the example using LoadLeveler
    `llsubmit EpetraMatrix.ll`
  - Solution of exercises can be found in
    `./Solutions/Exercise3`

Solving linear systems with Amesos

Amesos

- Object oriented interface to several direct sparse solvers
- Sequential & parallel
- Based on `Epetra_CrsMatrix`, `Epetra_MultiVector`
- http://trilinos.sandia.gov/packages/amesos/

Solving linear systems with Amesos

- Amesos provides interfaces to the following direct solvers
  - Serial
    - KLU - default Amesos direct sparse solver (distributed within Amesos)
    - LAPACK (dense, distributed within Amesos)
    - SuperLU
    - UMFPACK
    - TAUCS

Solving linear systems with Amesos

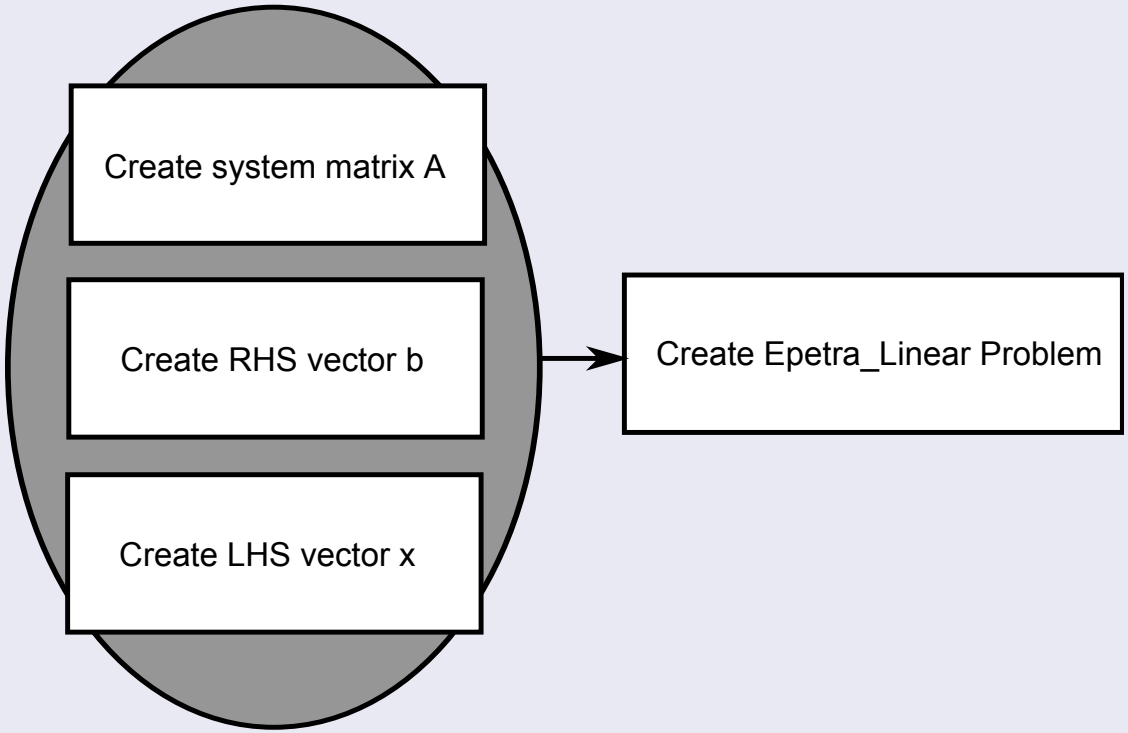
- Amesos provides interfaces to the following direct solvers
  - Serial
    - KLU - default Amesos direct sparse solver (distributed within Amesos)
    - LAPACK (dense, distributed within Amesos)
    - SuperLU
    - UMFPACK
    - TAUCS
  - Paralell
    - ScaLAPACK (dense)
    - PARDISO
    - SuperLU_DIST
    - DSPACK
    - MUMPS

Solving linear systems with Amesos

- Solves $A\mathbf{x} = \mathbf{b}$ where
  - A is squared `Epetra_RowMatrix` matrix
  - LHS **x** is `Epetra_Vector` or `Epetra_MultiVector` object
  - RHS **b** is `Epetra_Vector` or `Epetra_MultiVector` object
- If the chosen solver is serial, the matrix is shipped to process 0, solved there, and the solution distributed

Solving linear systems with Amesos

Step 1



Solving linear systems with Amesos

Spep 1

- Define a linear problem $A\mathbf{x} = \mathbf{b}$

```
Epetra_LinearProblem *Problem = new
    Epetra_LinearProblem;

problem->SetOperator(A);
problem->SetRHS(b);
problem->SetLHS(x);
```

Solving linear systems with Amesos

Step 2

- Create Amesos solver using Factory

```
Amesos_BaseSolver* Solver;
Amesos Factory;
string SolverType = "Klu";

Solver = Factory.Create(SolverType, Problem);
```

Solving linear systems with Amesos

Step 3

- Perform symbolic and numeric factorization

```
Solver->SymbolicFactorization();
Solver->NumericFactorization();
```

Solving linear systems with Amesos

Step 3

- Perform symbolic and numeric factorization

```
Solver->SymbolicFactorization();
Solver->NumericFactorization();
```

Step 4

```
Solver->Solve();
Problem->GetLHS();
```

Solving linear systems with Belos

Belos

- Next-generation iterative solver.
- Modern C++ framework for the numerical solution of large scale linear systems.
- Designed with extensibility and interoperability in mind.
- Allows user to leverage existing linear algebra libraries
- Abstract interfaces to operators and vectors.
  - `Belos::Operator/Vector` abstract interface
  - Currently implemented traits for Epetra, Tpetra and Thyra operators and vectors
- http://trilinos.sandia.gov/packages/belos/

Solving linear systems with Belos

- Belos utilizes abstract interfaces for operators and vectors
- Algorithms in Belos are developed at high level and underlying objects are opaque
- To leverage the existing linear algebra library, the user has to implement trait classes `Belos::MultiVecTraits` and `Belos::OperatorTraits`
  - Necessary to provide implementations of the methods `Clone`, `GetVecLength`, `MvTimesMatAddMv`, `MvAddMv`, `Apply` etc.

Solving linear systems with Belos

- Belos provides solvers for
  - Single RHS: $A\mathbf{x} = \mathbf{b}$
  - Multiple RHS (available simultaneously): $AX = B$
  - Multiple RHS (available sequentially): $A\mathbf{x}_i = \mathbf{b}_i$
  - Sequences of linear systems: $A_i\mathbf{x}_i = \mathbf{b}_i$

Solving linear systems with Belos

Solvers

- Hermitian Systems ($A = A^H$)
  - Conjugate gradients
  - Recycling conjugate gradients
  - Projected conjugate gradients
- Non-Hermitian Systems ($A \neq A^H$)
  - GMRES
  - Recycling GMRES
  - Flexible GMRES
  - TFQMR (Transpose-free quasi-minimal residual method)

Solving linear systems with Belos

Modular structure

- `Belos::LinearProblem`
- `Belos::SolverManager`
  - `Belos::Iteration`
  - `Belos::StatusTest`
  - `Belos::OrthoManager`
  - `Belos::OutputManager`

Solving linear systems with Belos

Step 1

```
typedef double                          SC;
typedef Teuchos::ScalarTraits<SC>       SCT;
typedef SCT::magnitudeType              MT;
typedef Epetra_MultiVector              MV;
typedef Epetra_Operator                 OP;
```

Solving linear systems with Belos

Step 2

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

RCP< LinearProblem<SC, MV, OP> > myProblem =
  rcp(new LinearProblem<SC, MV, OP>(A,X,B) );
myProblem->setHermitian();
myProblem->setProblem();
```

Solving linear systems with Belos

Step 2

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

RCP< LinearProblem<SC, MV, OP> > myProblem =
  rcp(new LinearProblem<SC, MV, OP>(A,X,B) );
myProblem->setHermitian();
myProblem->setProblem();
```

Step 3

```
RCP<Teuchos::ParameterList> myPL;
myPL->set( "Maximum Iterations", 100 );
myPL->set( "Convergence Tolerance", 1.0e-8 );
```

Solving linear systems with Belos

Step 4

```
Belos::BlockCGSolMgr<ST,MV,OP> mySolver( myProblem,
    myPL );
```

Solving linear systems with Belos

Step 4

```
Belos::BlockCGSolMgr<ST,MV,OP> mySolver( myProblem,
    myPL );
```

Step 5

```
Belos::ReturnType solverRet = mySolver.solve();
RCP<MV> sol = myProblem->getLHS();
```

Output

```
Passed.......OR Combination ->
   OK...........Number of Iterations = 21 < 100
   Converged....(2-Norm Imp Res Vec) / (2-Norm Res0)
                residual [ 0 ] = 6.38815e-09 < 1e-08
                residual [ 1 ] = 9.08579e-09 < 1e-08
                residual [ 2 ] = 9.26932e-09 < 1e-08
                residual [ 3 ] = 4.43278e-09 < 1e-08
================================================================================
                                TimeMonitor Results
Timer Name                                          Local time (num calls)
--------------------------------------------------------------------------------
Belos: Operation Op*x                               0.000583 (22)
Belos: Operation Prec*x                             0 (0)
Epetra_CrsMatrix::Multiply(TransA,X,Y)              0.000453 (22)
Belos: Orthogonalization                            0.006996 (23)
Belos: BlockCGSolMgr total solve time               0.01208 (1)
================================================================================
```

Conclusion

- Trilinos is an alternative (complement) to PETSc based on C++
- we've seen just a small (but important) fraction of the Trilinos framework
- especially the 'next-generation' packages are now rapidly developing
- for more information visit the official Trilinos site

Bibliography I

M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams.

An Overview of Trilinos.

Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

M. A. Heroux and J. M. Willenbring.

Trilinos Users Guide.

Technical Report SAND2003-2952, Sandia National Laboratories, 2003.

M. Sala, M. A. Heroux, and D. M. Day.

Trilinos Tutorial.

Technical Report SAND2004-2189, Sandia National Laboratories, 2004.

Bibliography II


Trilinos Project Webpage
http://trilinos.sandia.gov


Trilinos User Group Meetings
http://trilinos.sandia.gov/events/index.html


Trilinos Google Code
http://code.google.com/p/trilinos/

Thank you for your attention!