

A Trilinos Tutorial



Michael A. Heroux
Sandia National Laboratories



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.



Outline of Talk

- Background/Motivation
- Trilinos Package Concepts.
- Overview of Current Release Packages.
- Glimpse at Release 5.0 Packages.
- Example codes.
- Availability and support.
- Concluding remarks.

Trilinos Development Team

Ross Bartlett

Lead Developer of TSFCore

Paul Boggs

Developer of TSF

Jason Cross

Developer of Jpetra

David Day

Developer of Komplex

Bob Heaphy

Lead developer of Trilinos SQA

Mike Heroux

Trilinos Project Leader

Lead Developer of Epetra, AztecOO, Kokkos, Komplex and IFPACK, TSF

Developer of Amesos, Belos

Robert Hoekstra

Developer of Epetra

Russell Hooper

Developer of NOX

Vicki Howle

Developer of Belos and TSF

Jonathan Hu

Developer of ML

Tammy Kolda

Lead Developer of NOX

Rich Lehoucq

Developer of Anasazi and Belos

Kevin Long

Lead Developer of TSF,

Developer of Belos

Roger Pawlowski

Lead Developer of NOX

Michael Phenow

Trilinos Webmaster

Eric Phipps

Developer of LOCA and NOX

Andrew Rothfuss

Developer of TSF

Andrew Salinger

Lead Developer of LOCA

Marzio Sala

Lead author of Trilinos Tutorial

Developer of ML and Amesos

Paul Sexton

Developer of Epetra and Tpetra

Ken Stanley

Lead Developer of Amesos

Heidi Thornquist

Lead Developer of Anasazi and Belos

Ray Tuminaro

Lead Developer of ML

Jim Willenbring

Developer of Epetra and Kokkos.

Trilinos library manager

Alan Williams

Developer of Epetra

Motivation For Trilinos

- Sandia does LOTS of solver work.
- When I started at Sandia in May 1998:
 - ◆ Aztec was a mature package. Used in many codes.
 - ◆ FETI, PETSc, DSCPack, Spooles, ARPACK, DASPCK, and many other codes were (and are) in use.
 - ◆ New projects were underway or planned in multi-level preconditioners, eigensolvers, non-linear solvers, etc...
- The challenges:
 - ◆ Little or no coordination was in place to:
 - Efficiently reuse existing solver technology.
 - Leverage new development across various projects.
 - Support solver software processes.
 - Provide consistent solver APIs for applications.
 - ◆ ASCI was forming software quality assurance/engineering (SQA/SQE) requirements:
 - Daunting requirements for any single solver effort to address alone.

Evolving Trilinos Solution

- Trilinos¹ is an evolving framework to address these challenges:
 - ◆ Includes common core set of vector, graph and matrix classes (Epetra).
 - ◆ Provides a common abstract solver API (TSF).
 - ◆ Provides a ready-made package infrastructure:
 - Source code management (cvs, bonsai).
 - Build tools (autotools).
 - Automated regression testing (queue directories within repository).
 - Communication tools (mailman mail lists).
 - ◆ Specifies requirements and suggested practices for SQA.
- In general allows us to categorize efforts:
 - ◆ Efforts best done at the Trilinos level (useful to most or all packages).
 - ◆ Efforts best done at a package level (peculiar or important to a package).
 - ◆ **Allows package developers to focus only on things that are unique to their package.**

1. Trilinos loose translation: "A string of pearls"

Trilinos Strategic Goals

- **Scalable Solvers:** As problem size and processor counts increase, the cost of the solver will remain a nearly fixed percentage of the total solution time.
- **Hardened Solvers:** Never fail unless problem essentially unsolvable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
- **Universal Interoperability:** All Trilinos packages will be interoperable, so that any combination of solver packages that makes sense algorithmically will be possible within Trilinos.
- **Universal Solver RAS:** Trilinos will be:
 - ◆ Integrated into every major application at Sandia (**Availability**).
 - ◆ The leading edge hardened, scalable solutions for each of these applications (**Reliability**).
 - ◆ Easy to maintain and upgrade within the application environment (**Serviceability**).

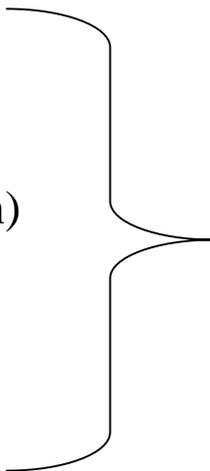
Algorithmic
Goals

Software
Goals

Trilinos: The Three “I”s

■ Infrastructure:

- ◆ Repository (cvs)
- ◆ Issue Tracking (Bugzilla)
- ◆ Communication (Mailman)
- ◆ Debugging (Bonsai)
- ◆ Jumpstart (new_package)
- ◆ SQA Tools and Policies.



Provided at Trilinos level. Packages get for “free”.

■ Interfaces:

- ◆ Interoperability: Between Trilinos Packages, with external SW.
- ◆ Extensible.
- ◆ Without Interdependence.

■ Implementations:

- ◆ Real, working code that implements all interfaces.
- ◆ Solid default implementations, but not required to use.

Trilinos Package Concepts

Trilinos Packages

- Trilinos is a collection of *Packages*.
- Each package is:
 - ◆ Focused on important, state-of-the-art algorithms in its problem regime.
 - ◆ Developed by a small team of domain experts.
 - ◆ Self-contained: No explicit dependencies on any other software packages (with some special exceptions).
 - ◆ Configurable/buildable/documented on its own.
- Sample packages: NOX, AztecOO, ML, IFPACK, Meros.
- Special package collections:
 - ◆ Petra (Epetra, Tpetra, Jpetra): Concrete Data Objects
 - ◆ TSF (TSFCore, TSFExtended): Abstract Conceptual Interfaces
 - ◆ Teuchos: Common Tools.
 - ◆ New_package: Jumpstart prototype.

(some of) What can be done with Trilinos

| Objective | Package(s) |
|---|-------------------------|
| Distributed linear algebra objects | Epetra, Jpetra, Tpetra |
| Krylov solvers | AztecOO, Belos |
| ILU-type preconditioners | AztecOO, IFPACK |
| Smoothed aggregation, multilevel prec's | ML |
| Eigenvalue problems | Anasazi |
| Block preconditioners | Meros |
| Direct sparse linear solvers | Amesos |
| Direct dense solvers | Epetra, Teuchos, Pliris |
| Abstract interfaces | TSF |
| Nonlinear system solvers | NOX, LOCA |
| Complex linear systems | Komplex |
| C++ utilities, (some) I/O | Teuchos, EpetraExt |

| Package | Description | Release | | | |
|-------------------|---|----------------|----------------|--------------|--------------|
| | | 3.1 (9/2003) | | 4 (5/2004) | |
| | | 3.1 General | 3.1 Limited | 4 General | 4 Limited |
| Amesos | 3 rd Party Direct Solver Suite | | X | X | X |
| Anasazi | Eigensolver package | | | | X |
| AztecOO | Linear Iterative Methods | X | X | X | X |
| Belos | Block Linear Solvers | | | | X |
| Epetra | Basic Linear Algebra | X | X | X | X |
| EpetraExt | Extensions to Epetra | | X | X | X |
| Ifpack | Algebraic Preconditioners | X | X | X | X |
| Jpetra | Java Petra Implementation | | | | X |
| Kokkos | Sparse Kernels | | | X | X |
| Komplex | Complex Linear Methods | X | X | X | X |
| LOCA | Bifurcation Analysis Tools | X | X | X | X |
| Meros | Segregated Preconditioners | | X | | X |
| ML | Multi-level Preconditioners | X | X | X | X |
| NewPackage | Working Package Prototype | X | X | X | X |
| NOX | Nonlinear solvers | X | X | X | X |
| Pliris | Dense direct Solvers | | | | X |
| Teuchos | Common Utilities | | | X | X |
| TSFCore | Abstract Solver API | | | X | X |
| TSFExt | Extensions to TSFCore | | | X | X |
| Tpetra | Templated Petra | | | | X |
| Totals | | 8 | 11 | 14 | 20 |

Dependence vs. Interoperability

- Although most Trilinos packages have no explicit dependence, each package must interact with *some* other packages:
 - ◆ NOX needs operator, vector and solver objects.
 - ◆ AztecOO needs preconditioner, matrix, operator and vector objects.
 - ◆ Interoperability is enabled at configure time. For example, NOX:
 - enable-nox-lapack compile NOX lapack interface libraries
 - enable-nox-epetra compile NOX epetra interface libraries
 - enable-nox-petsc compile NOX petsc interface libraries
- Trilinos is a vehicle for:
 - ◆ Establishing interoperability of Trilinos components...
 - ◆ Without compromising individual package autonomy.
- Trilinos offers five basic interoperability mechanisms.

Trilinos Interoperability Mechanisms

Package accepts user data as Epetra or TSF objects

⇒

Applications using Epetra/TSF can use *package*

Package accepts parameters from Teuchos ParameterLists

⇒

Applications using Teuchos ParameterLists can drive *package*

Package can be used via TSF abstract solver classes

⇒

Applications or other packages using TSF can use *package*

Package can use Epetra for private data.

⇒

Package can then use other packages that understand Epetra

Package accesses solver services via TSF interfaces

⇒

Package can then use other packages that implement TSF interfaces

Package builds under Trilinos configure scripts.

⇒

Package can be built as part of a suite of packages; cross-package dependencies can be handled automatically



Interoperability Example: ML

- ML: Multi-level Preconditioner Package.
- Primary Developers: Ray Tuminaro, Jonathan Hu, Marzio Sala.
- No *explicit, essential* dependence on other Trilinos packages.
 - ◆ Uses abstract interfaces to matrix/operator objects.
 - ◆ Has independent configure/build process (but can be invoked at Trilinos level).
- *Interoperable* with other Trilinos packages and other libraries:
 - ◆ Accepts user data as Epetra matrices/vectors.
 - ◆ Can use Epetra for internal matrices/vectors.
 - ◆ Can be used via TSF abstract interfaces.
 - ◆ Can be built via Trilinos configure/build process.
 - ◆ Available as preconditioner to all other Trilinos packages.
 - ◆ Can use IFPACK, Amesos, AztecOO objects as smoothers, coarse solvers.
 - ◆ Can be driven via Teuchos ParameterLists.
 - ◆ Available to PETSc users *without dependence on any other Trilinos packages*.

What Trilinos is not

- Trilinos is not a single monolithic piece of software. Each package:
 - ◆ Can be built independent of Trilinos.
 - ◆ Has its own self-contained CVS structure.
 - ◆ Has its own Bugzilla product and mail lists.
 - ◆ Development team is free to make its own decisions about algorithms, coding style, release contents, testing process, etc.

- Trilinos top layer is not a large amount of source code:
 - ◆ Trilinos repository contains 452,187 source lines of code (SLOC).
 - ◆ Sum of the packages SLOC counts : 445,937.
 - ◆ Trilinos top layer SLOC count: 6, 250 (1.4%).

- Trilinos is not “indivisible”:
 - ◆ You don’t need all of Trilinos to get things done.
 - ◆ Any collection of packages can be combined and distributed.
 - ◆ Current public release contains only 14 of the 20+ Trilinos packages.

Overview of Trilinos Packages

Trilinos Common Language: Petra

- Petra provides a “common language” for distributed linear algebra objects (operator, matrix, vector)
- Petra¹ 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 3 implementations under development.

¹Petra is Greek for “foundation”.

Petra Implementations

- Three version under development:
- 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.
- Jpetra (Java Petra):
 - ◆ Pure Java. Portable to any JVM.
 - ◆ Interfaces to Java versions of MPI, LAPACK and BLAS via interfaces.

Developers:

- Mike Heroux, Rob Hoekstra, Alan Williams, Paul Sexton

Epetra

- Basic Stuff: What you would expect.
- Variable block matrix data structures.
- Multivectors.
- Arbitrary index labeling.
- Flexible, versatile parallel data redistribution.
- Language support for inheritance, polymorphism and extensions.
- View vs. Copy.

RowMatrix is a *Operator*

Epetra User Class Categories

- ◆ Sparse Matrices: *RowMatrix*, (CrsMatrix, VbrMatrix, FECrsMatrix, FEVbrMatrix)
- ◆ Linear Operator: *Operator*: (AztecOO, ML, Ifpack)
- ◆ Dense Matrices: DenseMatrix, DenseVector, BLAS, LAPACK, SerialDenseSolver
- ◆ Vectors: Vector, MultiVector
- ◆ Graphs: CrsGraph
- ◆ Data Layout: Map, BlockMap, LocalMap
- ◆ Redistribution: Import, Export, LbGraph, LbMatrix
- ◆ Aggregates: LinearProblem
- ◆ Parallel Machine: *Comm*, (SerialComm, MpiComm, MpiSmpComm)
- ◆ Utilities: Time, Flops

Most codes and packages require row access to matrices. Pure abstract class

Insulates users from MPI calls

AztecOO

- Krylov subspace solvers: CG, GMRES, Bi-CGSTAB,...
- Incomplete factorization preconditioners

- Aztec is the workhorse solver at Sandia:
 - ◆ 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 RHS.
 - ◆ Providing more preconditioners/scalings.
 - ◆ Using C++ class design to enable more sophisticated use.
- AztecOO interfaces allows:
 - ◆ Continued use of Aztec for functionality.
 - ◆ Introduction of new solver capabilities outside of Aztec.

Developers:

- Mike Heroux, Ray Tuminaro, John Shadid

IFPACK: Algebraic Preconditioners

- Overlapping Schwarz preconditioners with incomplete factorizations
- Accept user matrix via abstract matrix interface (Epetra versions).
- Uses Epetra for basic matrix/vector calculations.
- Supports simple perturbation stabilizations and condition estimation.
- Separates graph construction from factorization, improves performance substantially.
- Compatible with AztecOO and TSF. Can be used by NOX and ML.

Developer:

▪ Mike Heroux

Amesos

- Interface to direct solver for distributed sparse linear systems
- Challenges:
 - ◆ No single solver dominates
 - ◆ Different interfaces and data formats, serial and parallel
 - ◆ Interface often change between revisions
- Amesos offers:
 - ◆ A single, clear, consistent interface, to various packages
 - ◆ Common look-and-feel for all classes
 - ◆ Separation from specific solver details
 - ◆ Use serial and distributed solvers; Amesos takes care of data redistribution

Developers:

- Ken Stanley, Marzio Sala

Amesos: Supported Libraries

| Library Name | Language | communicator | # procs for solution ¹ |
|------------------|----------|--------------|-----------------------------------|
| KLU | C | Serial | 1 |
| UMFPACK | C | Serial | 1 |
| SuperLU 3.0 | C | Serial | 1 |
| SuperLU_DIST 2.0 | C | MPI | any |
| MUMPS 4.3.1 | F90 | MPI | any |
| ScaLAPACK | F77 | MPI | any |

Also working on: DSCPACk, SPOOLES, Ksparse,...

¹Matrices can be distributed over any number of processes

ML: Multi-level Preconditioners

- Smoothed aggregation, multigrid and domain decomposition preconditioning package
- Critical technology for scalable performance of some key apps.
- ML compatible with other Trilinos packages:
 - ◆ Accepts user data as Epetra_RowMatrix object (abstract interface). Any implementation of Epetra_RowMatrix works.
 - ◆ Implements the Epetra_Operator interface. Allows ML preconditioners to be used with AztecOO and TSF.
- Can also be used completely independent of other Trilinos packages.

Developers:

- Ray Tuminaro, Jonathan Hu, Marzio Sala

NOX: Nonlinear Solvers

- Suite of nonlinear solution methods
- NOX uses abstract vector and “group” interfaces:
 - ♦ Allows flexible selection and tuning of various strategies:
 - Directions.
 - Line searches.
 - ♦ Epetra/AztecOO/ML, LAPACK, PETSc implementations of abstract vector/group interfaces.
- Designed to be easily integrated into existing applications.

Developers:

- Tammy Kolda, Roger Pawlowski

LOCA

- Library of continuation algorithms
- Provides
 - ◆ Zero order continuation
 - ◆ First order continuation
 - ◆ Arc length continuation
 - ◆ Multi-parameter continuation (via Henderson's MF Library)
 - ◆ Turning point continuation
 - ◆ Pitchfork bifurcation continuation
 - ◆ Hopf bifurcation continuation
 - ◆ Phase transition continuation
 - ◆ Eigenvalue approximation (via ARPACK or Anasazi)

Developers:

- Andy Salinger, Eric Phipps

EpetraExt: Extensions to Epetra

- Library of useful classes not needed by everyone
- Most classes are types of “transforms”.
- Examples:
 - ♦ Graph/matrix view extraction.
 - ♦ Epetra/Zoltan interface.
 - ♦ Explicit sparse transpose.
 - ♦ Singleton removal filter, static condensation filter.
 - ♦ Overlapped graph constructor, graph colorings.
 - ♦ Permutations.
 - ♦ Sparse matrix-matrix multiply.
 - ♦ Matlab, MatrixMarket I/O functions.
 - ♦ ...
- Most classes are small, useful, but non-trivial to write.

Developer:

- Robert Hoekstra, Alan Williams, Mike Heroux

Teuchos

- Utility package of commonly useful tools:
 - ♦ ParameterList class: key/value pair database, recursive capabilities.
 - ♦ LAPACK, BLAS wrappers (templated on ordinal and scalar type).
 - ♦ Dense matrix and vector classes (compatible with BLAS/LAPACK).
 - ♦ FLOP counters, Timers.
 - ♦ Ordinal, Scalar Traits support: Definition of 'zero', 'one', etc.
 - ♦ Reference counted pointers, and more...
- Takes advantage of advanced features of C++:
 - ♦ Templates
 - ♦ Standard Template Library (STL)
- ParameterList:
 - ♦ Allows easy control of solver parameters.

Developers:

- Roscoe Barlett, Kevin Long, Heidi Thorquist, Mike Heroux, Paul Sexton, Kris Kampshoff

NewPackage Package

- NewPackage provides jump start to develop/integrate a new package
- NewPackage is a “Hello World” program and website:
 - ◆ Simple but it does work with autotools.
 - ◆ Compiles and builds.
- NewPackage directory contains:
 - ◆ Commonly used directory structure: src, test, doc, example, config.
 - ◆ Working autotools files.
 - ◆ Documentation templates (doxygen).
 - ◆ Working regression test setup.
- Substantially cuts down on:
 - ◆ Time to integrate new package.
 - ◆ Variation in package integration details.
 - ◆ Development of website.

Investment in Templates

- 2nd Generation Trilinos packages are templated on:
 - ◆ OrdinalType (think **int**).
 - ◆ ScalarType (think **double**).
- Examples:
Teuchos::SerialDenseMatrix<int, double> A;
Teuchos::SerialDenseMatrix<short, float> B;
- The following packages support templates:

Teuchos (Basic Tools)
TSFCore (Abstract Interfaces) } In current release

Tpetra (including MPI support)
Belos (Krylov and Block Krylov Linear),
IFPACK (algebraic preconditioners, next version),
Anasazi (Eigensolvers),
TSFExt (Abstract interfaces) } In Release 5.0
March 2005

ARPREC

- The ARPREC library uses arrays of 64-bit floating-point numbers to represent high-precision floating-point numbers.
- ARPREC values behave just like any other floating-point datatype, except the maximum working precision (in decimal digits) must be specified before any calculations are done
 - ◆ `mp::mp_init(200);`
- Illustrate the use of ARPREC with an example using Hilbert matrices.

Hilbert Matrices

- A Hilbert matrix H_N is a square N -by- N matrix such that:

- For Example:
$$H_{N_{ij}} = \frac{1}{i + j - 1}$$

$$H_3 = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

Hilbert Matrices

- Notoriously ill-conditioned
 - ◆ $\kappa(H_3) \approx 524$
 - ◆ $\kappa(H_5) \approx 476610$
 - ◆ $\kappa(H_{10}) \approx 1.6025 \times 10^{13}$
 - ◆ $\kappa(H_{20}) \approx 7.8413 \times 10^{17}$
 - ◆ $\kappa(H_{100}) \approx 1.7232 \times 10^{20}$
- Hilbert matrices introduce large amounts of error

Hilbert Matrices and Cholesky Factorization

- With double-precision arithmetic, Cholesky factorization will fail for H_N for all $N > 13$.
- Can we improve on this using arbitrary-precision floating-point numbers?

| Precision | Largest N for which Cholesky Factorization is successful |
|----------------------------------|--|
| Single Precision | 8 |
| Double Precision | 13 |
| Arbitrary Precision (20) | 29 |
| Arbitrary Precision (40) | 40 |
| Arbitrary Precision (200) | 145 |
| Arbitrary Precision (400) | 233 |

Kokkos

- Goal:
 - ◆ Isolate key non-BLAS kernels for the purposes of optimization.
- Kernels:
 - ◆ Dense vector/multivector updates and collective ops (not in BLAS/Teuchos).
 - ◆ Sparse MV, MM, SV, SM.
- Serial-only for now.
- Reference implementation provided (templated).
- Mechanism for improving performance:
 - ◆ Default is aggressive compilation of reference source.
 - ◆ BeBOP: Jim Demmel, Kathy Yelick, Rich Vuduc, UC Berkeley.
 - ◆ Vector version: Cray.

Developer:

■ Mike Heroux

Kokkos Results: Sparse MV/MM

Pentium M 1.6GHz Cygwin/GCC 3.2 (WinXP Laptop)

| Data Set | Dimension | Nonzeros | # RHS | MFLOPS |
|----------|-----------|----------|-------|--------|
| DIE3D | 9873 | 1733371 | 1 | 247.62 |
| | | | 2 | 418.94 |
| | | | 3 | 577.79 |
| | | | 4 | 691.62 |
| | | | 5 | 787.90 |
| dday01 | 21180 | 923006 | 1 | 230.75 |
| | | | 2 | 407.96 |
| | | | 3 | 553.80 |
| | | | 4 | 668.24 |
| | | | 5 | 738.40 |
| FIDAP035 | 19716 | 218308 | 1 | 171.22 |
| | | | 2 | 349.29 |
| | | | 3 | 374.24 |
| | | | 4 | 498.99 |
| | | | 5 | 545.77 |

Examples

A Simple Epetra/AztecOO Program

```
// Header files omitted...
int main(int argc, char *argv[]) {
    MPI_Init(&argc,&argv); // Initialize MPI, MpiComm
    Epetra_MpiComm Comm( MPI_COMM_WORLD );
```

```
// ***** Map puts same number of equations on each pe *****
```

```
int NumMyElements = 1000 ;
Epetra_Map Map(-1, NumMyElements, 0, Comm);
int NumGlobalElements = Map.NumGlobalElements();
```

```
// ***** 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 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." << endl
    << "Norm of true residual = "
    << solver.TrueResidual()
    << endl;
```

```
MPI_Finalize() ;
return 0;
```

```
}
```

Typical Flow of Epetra Object Construction

Construct Comm

- Any number of Comm objects can exist.
- Comms can be nested (ee.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.

Parallel Data Redistribution

- Epetra vectors, multivectors, graphs and matrices are distributed via one of the map objects.
- A map is basically a partitioning of a list of global IDs:
 - ◆ IDs are simply labels, no need to use contiguous values (Directory class handles details for general ID lists).
 - ◆ No *a priori* restriction on replicated IDs.
- If we are given:
 - ◆ A source map and
 - ◆ A set of vectors, multivectors, graphs and matrices (or other distributable objects) based on source map.
- Redistribution is performed by:
 1. Specifying a target map with a new distribution of the global IDs.
 2. Creating Import or Export object using the source and target maps.
 3. Creating vectors, multivectors, graphs and matrices that are redistributed (to target map layout) using the Import/Export object.

Example: epetra/ex9.cpp (p. 25)

```
int main(int argc, char *argv[]) {
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
  int NumGlobalElements = 4; // global dimension of
    the problem
  int NumMyElements; // local nodes
  Epetra_IntSerialDenseVector MyGlobalElements;

  if( Comm.MyPID() == 0 ) {
    NumMyElements = 3;
    MyGlobalElements.Size(NumMyElements);
    MyGlobalElements[0] = 0;
    MyGlobalElements[1] = 1;
    MyGlobalElements[2] = 2;
  } else {
    NumMyElements = 3;
    MyGlobalElements.Size(NumMyElements);
    MyGlobalElements[0] = 1;
    MyGlobalElements[1] = 2;
    MyGlobalElements[2] = 3;
  }
  // create a map
  Epetra_Map Map(-1,MyGlobalElements.Length(),
    MyGlobalElements.Values(),0, Comm);
```

```
// create a vector based on map
  Epetra_Vector xxx(Map);
  for( int i=0 ; i<NumMyElements ; ++i )
    xxx[i] = 10*( Comm.MyPID()+1 );
  if( Comm.MyPID() == 0 ){
    double val = 12;
    int pos = 3;
    xxx.SumIntoGlobalValues(1,0,&val,&pos);
  }
  cout << xxx;
  // create a target map, in which all elements are on proc 0
  int NumMyElements_target;
  if( Comm.MyPID() == 0 )
    NumMyElements_target = NumGlobalElements;
  else
    NumMyElements_target = 0;
  Epetra_Map TargetMap(-1,NumMyElements_target,0,Comm);
  Epetra_Export Exporter(Map,TargetMap);
  // work on vectors
  Epetra_Vector yyy(TargetMap);
  yyy.Export(xxx,Exporter,Add);
  cout << yyy;
  MPI_Finalize();
  return( EXIT_SUCCESS );
}
```

Output: epetra/ex9.cpp

```
> mpirun -np 2 ./ex9.exe
```

```
Epetra::Vector
```

| MyPID | GID | Value |
|-------|-----|-------|
| 0 | 0 | 10 |
| 0 | 1 | 10 |
| 0 | 2 | 10 |

```
Epetra::Vector
```

| | | |
|---|---|----|
| 1 | 1 | 20 |
| 1 | 2 | 20 |
| 1 | 3 | 20 |

```
Epetra::Vector
```

| MyPID | GID | Value |
|-------|-----|-------|
| 0 | 0 | 10 |
| 0 | 1 | 30 |
| 0 | 2 | 30 |
| 0 | 3 | 20 |

```
Epetra::Vector
```

Before Export

PE 0
xxx(0)=10
xxx(1)=10
xxx(2)=10

PE 1
xxx(1)=20
xxx(2)=20
xxx(3)=20

Export/Add

After Export

PE 0
yyy(0)=10
yyy(1)=30
yyy(2)=30
yyy(3)=20

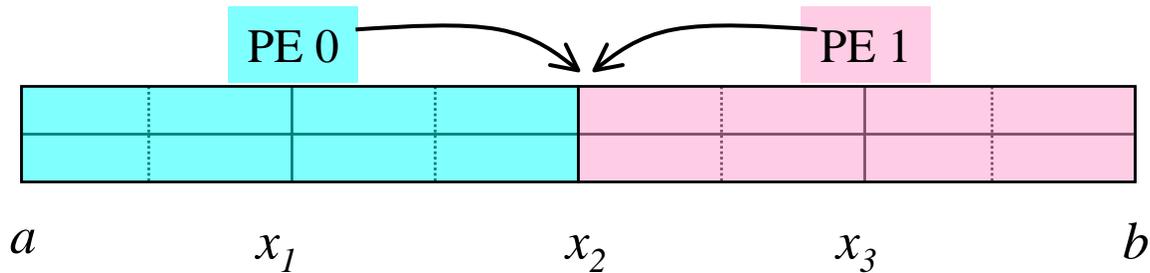
PE 1

Import vs. Export

- Import (Export) means calling processor knows what it wants to receive (send).
- Distinction between Import/Export is important to user, almost identical in implementation.
- Import (Export) objects can be used to do an Export (Import) as a reverse operation.
- When mapping is bijective (1-to-1 and onto), either Import or Export is appropriate.

Example: 1D Matrix Assembly

$$\begin{aligned} -u_{xx} &= f \\ u(a) &= \gamma_0 \\ u(b) &= \gamma_1 \end{aligned}$$



- 3 Equations: Find u at x_1 , x_2 and x_3
- Equation for u at x_2 gets a contribution from PE 0 and PE 1.
- Would like to compute partial contributions independently.
- Then combine partial results.

Two Maps

- We need two maps:
 - ◆ Assembly map:
 - PE 0: { 1, 2 }.
 - PE 1: { 2, 3 }.
 - ◆ Solver map:
 - PE 0: { 1, 2 } (we arbitrate ownership of 2).
 - PE 1: { 3 }.

End of Assembly Phase

- At the end of assembly phase we have AssemblyMatrix:

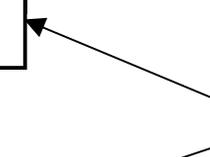
On PE 0:

$$\text{Equation 1: } \begin{bmatrix} 2 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

On PE 1:

$$\text{Equation 2: } \begin{bmatrix} 0 & 1 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Row 2 is shared



- Want to assign all of Equation 2 to PE 0 for use with solver.
- NOTE: For a class of Neumann-Neumann preconditioners, the above layout is exactly what we want.

Export Assembly Matrix to Solver Matrix

```
Epetra_Export Exporter(AssemblyMap, SolverMap);  
  
Epetra_CrsMatrix SolverMatrix (Copy, SolverMap, 0);  
  
SolverMatrix.Export(AssemblyMatrix, Exporter, Add);  
  
SolverMatrix.FillComplete();
```

Matrix Export

Before Export

PE 0

Equation 1: $\begin{bmatrix} 2 & -1 & 0 \end{bmatrix}$

Equation 2: $\begin{bmatrix} -1 & 1 & 0 \end{bmatrix}$

After Export

PE 0

Equation 1: $\begin{bmatrix} 2 & -1 & 0 \end{bmatrix}$

Equation 2: $\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$

Export/Add

PE 1

Equation 2: $\begin{bmatrix} 0 & 1 & -1 \end{bmatrix}$

Equation 3: $\begin{bmatrix} 0 & -1 & 2 \end{bmatrix}$

PE 1

Equation 3: $\begin{bmatrix} 0 & -1 & 2 \end{bmatrix}$

Example: epetraext/ex2.cpp (p. 111)

```
int main(int argc, char *argv[]) {
    MPI_Init(&argc,&argv);
    Epetra_MpiComm Comm (MPI_COMM_WORLD);

    int MyPID = Comm.MyPID();
    int n=4;

    // Generate Laplacian2d gallery matrix
    Trilinos_Util::CrsMatrixGallery G("laplace_2d", Comm);
    G.Set("problem_size", n*n);
    G.Set("map_type", "linear"); // Linear map initially

    // Get the LinearProblem.
    Epetra_LinearProblem *Prob = G.GetLinearProblem();

    // Get the exact solution.
    Epetra_MultiVector *sol = G.GetExactSolution();
    // Get the rhs (b) and lhs (x)
    Epetra_MultiVector *b = Prob->GetRHS();
    Epetra_MultiVector *x = Prob->GetLHS();
```

```
// Repartition graph using Zoltan
EpetraExt::Zoltan_CrsGraph * ZoltanTrans = new
    EpetraExt::Zoltan_CrsGraph();
EpetraExt::LinearProblem_GraphTrans * ZoltanLPTrans =
    new EpetraExt::LinearProblem_GraphTrans(
        *(dynamic_cast<EpetraExt::StructuralSameTypeTransform<Epetra_CrsGraph>*>(ZoltanTrans)) );
cout << "Creating Load Balanced Linear Problem\n";
Epetra_LinearProblem &BalancedProb =
    (*ZoltanLPTrans)(*Prob);

// Get the rhs (b) and lhs (x)
Epetra_MultiVector *Balancedb = Prob->GetRHS();
Epetra_MultiVector *Balancedx = Prob->GetLHS();
cout << "Balanced b: " << *Balancedb << endl;
cout << "Balanced x: " << *Balancedx << endl;
MPI_Finalize() ;
return 0 ;
}
```

Need for Import/Export

- Solvers for complex engineering applications need expressive, easy-to-use parallel data redistribution:
 - ◆ Allows better scaling for non-uniform overlapping Schwarz.
 - ◆ Necessary for robust solution of multiphysics problems.
- We have found import and export facilities to be a very natural and powerful technique to address these issues.

Other uses for Import/Export

- In addition, import and export facilities provide a variety of other capabilities:
 - ◆ Communication needs of sparse matrix multiplication.
 - ◆ Parallel assembly: Shared nodes receive contributions from multiple processors, reverse operation replicates results back.
 - ◆ Higher order interpolations are easy to implement.
 - ◆ Ghost node distributions.
 - ◆ Changing work loads can be re-balanced.
 - ◆ Sparse matrix transpose become trivial to implement.
 - ◆ Allows gradual MPI-izing of an application.
 - ◆ Cached Overlapped distributed vectors (generalization of distributed sparse MV).
 - ◆ Rendezvous algorithms easy to implement.

Example: ml/ex1.cpp (p. 87)

```
int main(int argc, char *argv[])
{
MPI_Init(&argc,&argv);
Epetra_MpiComm Comm(MPI_COMM_WORLD);
// initialize the command line parser
Trilinos_Util_CommandLineParser CLP(argc,argv);
// initialize an Gallery object
Trilinos_Util_CrsMatrixGallery Gallery("", Comm);
// add default values
if( CLP.Has("-problem_type") == false ) CLP.Add("-
  problem_type", "laplace_2d" );
if( CLP.Has("-problem_size") == false ) CLP.Add("-
  problem_size", "100" );

// initialize the gallery as specified in the command line
Gallery.Set(CLP);

// retrieve pointers to matrix and linear problem
Epetra_CrsMatrix * Matrix = Gallery.GetMatrix();
const Epetra_Map * Map = Gallery.GetMap();

Epetra_LinearProblem * Problem = Gallery.GetLinearProblem();

// Construct a solver object for this problem
AztecOO solver(*Problem);

// solve with CG (change is matrix is not symmetric)
solver.SetAztecOption(AZ_solver, AZ_cg);
```

```
// Create and set an ML multilevel preconditioner
ML *ml_handle;
// Maximum number of levels
int N_levels = 10;
// output level
ML_Set_PrintLevel(3);
ML_Create(&ml_handle,N_levels);
// wrap Epetra Matrix into ML matrix (data is NOT copied)
EpetraMatrix2MLMatrix(ml_handle, 0, Matrix);
// as we are interested in smoothed aggregation, create a
  ML_Aggregate object to store the aggregates
ML_Aggregate *agg_object;
ML_Aggregate_Create(&agg_object);
// specify max coarse size (ML will not coarse further is the matrix at
  a given level is
// smaller than specified here)
ML_Aggregate_Set_MaxCoarseSize(agg_object,1);

// generate the hierarchy
N_levels = ML_Gen_MGHierarchy_UsingAggregation(ml_handle, 0,
  ML_INCREASING, agg_object);
// Set a symmetric Gauss-Seidel smoother for the MG method (change
// if the matrix is not symmetric)
ML_Gen_Smoothing_SymGaussSeidel(ml_handle,
  ML_ALL_LEVELS, ML_BOTH, 1, ML_DEFAULT);
// generate solver
ML_Gen_Solver (ml_handle, ML_MGV, 0, N_levels-1);
// wrap ML_Operator into Epetra_Operator
ML_Epetra::MultiLevelOperator
  MLOp(ml_handle,Comm,*Map,*Map);
```

Example: ml/ex1.cpp (p. 87)

```
// set this operator as preconditioner for AztecOO
solver.SetPrecOperator(&MLop);

// solve
solver.Iterate(1550, 1e-12);

// verify that residual is really small
double residual, diff;

Gallery.ComputeResidual(residual);
Gallery.ComputeDiffBetweenStartingAndExactSolutions(diff);

if( Comm.MyPID() == 0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
}

#ifdef EPETRA_MPI
    MPI_Finalize() ;
#endif

return 0 ;
}
```

- AztecOO can use Epetra_Operators as preconditioners.
- ML_Epetra::MultiLevelOperator *isa* Epetra_Operator
- Use of IFPACK is very similar to ML:
 - ◆ Also implements Epetra_Operator.
- AztecOO also implements Epetra_Operator.
 - ◆ Allows use as preconditioner.
 - ◆ See example aztecoo/ex2.cpp.

Example: amesos/ex1.cpp (p. 100)

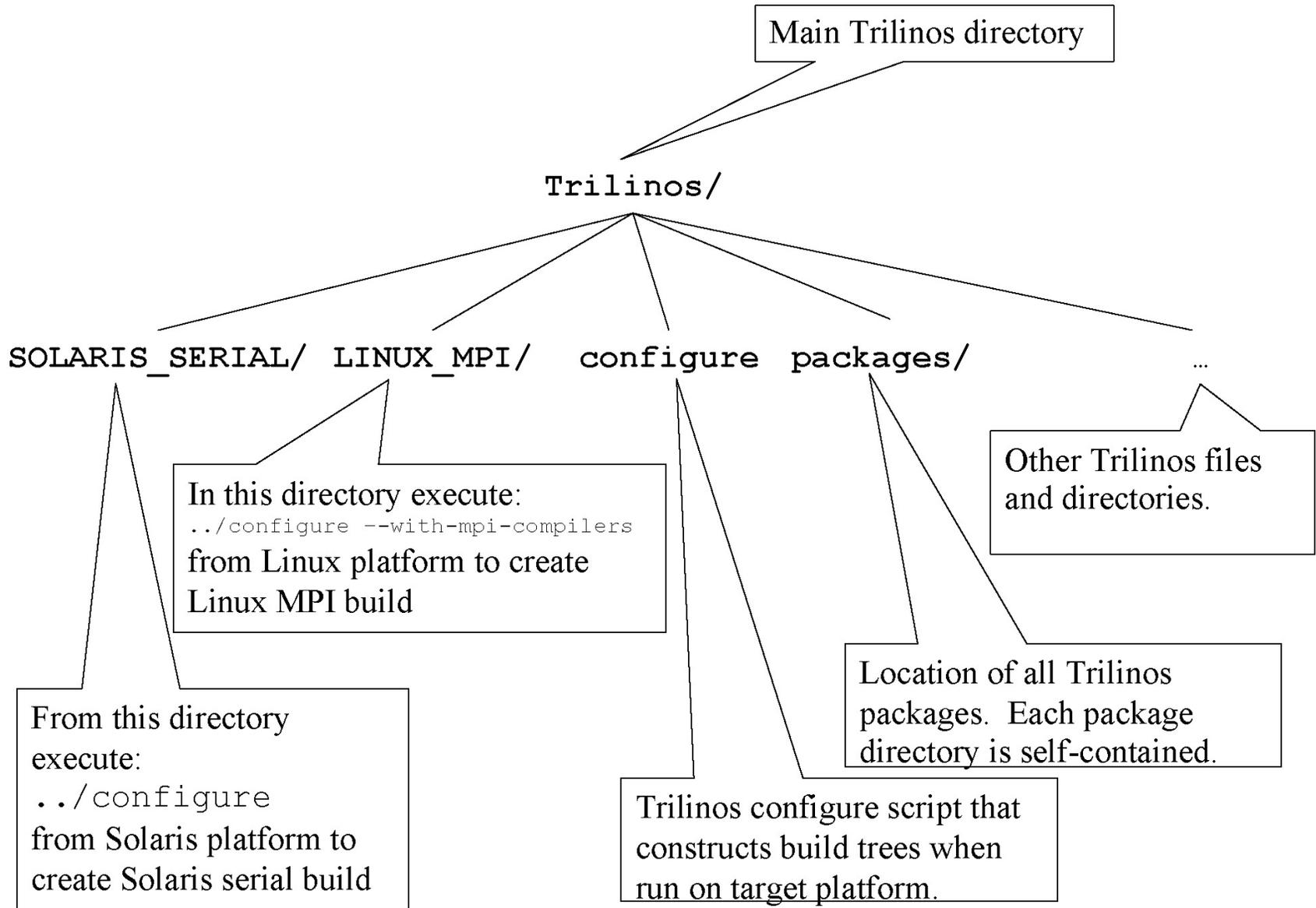
```
int main(int argc, char *argv[]) {
MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
// initialize the command line parser
  Trilinos_Util_CommandLineParser CLP(argc,argv);
// initialize an Gallery object
  Trilinos_Util_CrsMatrixGallery Gallery("", Comm);
// add default values
  if( CLP.Has("-problem_type") == false ) CLP.Add("-
    problem_type", "laplace_2d" );
  if( CLP.Has("-problem_size") == false ) CLP.Add("-
    problem_size", "100" );
// initialize the gallery as specified in the command line
  Gallery.Set(CLP);

Epetra_CrsMatrix * Matrix = Gallery.GetMatrix();
  Epetra_Vector * LHS = Gallery.GetStartingSolution();
  Epetra_Vector * RHS = Gallery.GetRHS();

  Epetra_LinearProblem * Problem = Gallery.GetLinearProblem();
// initialize Amesos solver
  Amesos_BaseSolver * Solver;
  Amesos_Amesos_Factory;
// empty parameter list
  Teuchos::ParameterList List;
```

```
// use KLU (other choices are valid as well)
  int choice = 0;
  switch( choice ) {
  case 0:
    Solver = A_Factory.Create("Amesos_Klu", *Problem);
    break;
  case 1:
    Solver = A_Factory.Create("Amesos_Umfpack", *Problem);
    break;
  }
// start solving
  Solver->SymbolicFactorization();
  Solver->NumericFactorization();
  Solver->Solve();
// verify that residual is really small
  double residual, diff;
  Gallery.ComputeResidual(residual);
  Gallery.ComputeDiffBetweenStartingAndExactSolutions(diff);
  if( Comm.MyPID() == 0 ) {
    cout << "||b-Ax||_2 = " << residual << endl;
    cout << "||x_exact - x||_2 = " << diff << endl;
  }
// delete Solver
  delete Solver;
MPI_Finalize();
  return( EXIT_SUCCESS );
}
```

Building Trilinos



Simple Sample Scripts

```
./configure
```

- Builds serial version of default packages.
- All 3rd party libraries (BLAS, LAPACK) must be in lib search path.

```
./configure --prefix=/home/mheroux/Trilinos/EPETRA_OPT_SERIAL \  
CXXFLAGS="-O3" --disable-default-packages --enable-epetra
```

- Builds serial version of Epetra only with some optimization.
- All 3rd party libraries (BLAS, LAPACK) must be in lib search path.
- "make install" will put /include/*.h and /lib/*.a in specified directory.

- Builds MPI version of Epetra only with specified MPI environment.
- All 3rd party libraries (BLAS, LAPACK) must be in lib search path.
- "make install" will put /include/*.h and /lib/*.a in specified directory.

```
./configure --enable-mpi --with-mpi=/usr/MPICH/SDK.gcc \  
--prefix=/home/mheroux/Trilinos/EPETRA_MPI \  
--with-mpi-cxx=g++ --with-mpi-cc=gcc --with-mpi-f77=g77 \  
--with-mpi-libs=-Impich \  
--disable-default-packages --enable-epetra
```

More Complex Sample Scripts

Most found in Trilinos/sampleScripts

```
./configure --host=powerpc-ibm-aix5.1.0.0 \  
--enable-teuchos --enable-teuchos-extended --disable-new-package \  
--enable-aztecoo-azlu \  
--enable-anasazi \  
--enable-epetraext --enable-epetraext-transform \  
--enable-epetraext-transform-tests \  
--enable-amesos \  
--with-ml_teuchos --with-ml_anasazi \  
--with-ml_external_mpi_functions \  
--disable-examples \  
--with-blas="-lessl -L/sierra/Release/lapack/3.0/lib/dbg_dp_ibm -lblas" \  
--with-lapack="-lessl -L/sierra/Release/lapack/3.0/lib/dbg_dp_ibm -llapack" \  
--enable-mpi \  
--with-ldflags="-L/sierra/Release/y12m/1.00/lib/dbg_dp_ibm -ly12m" \  
--prefix=/sierra/Release/Trilinos/3.1.1/install_ibm \  
--with-mpi-incdir=. \  
--with-mpi-libdir=. \  
--with-mpi-libs="-binitfjni:poe_remote_main -lmpi_r -lvtd_r" \  
--with-ar="ar -X64 csv" \  
CXX="mpCC -q64" CC="mpcc -b64 -q64" F77="mpxlf -b64 -q64" \  
CXXFLAGS="-O3 -w -qnofullpath -qlanglvl=ansi" \  
CCFLAGS="-O3 -w" \  
FFLAGS="-O3 -w" \  
CPPFLAGS= LDFLAGS= LIBS=-lm \  
FLIBS="-lxf90 -lxlopt -lxf -lxlomp_ser"
```

- Configure/make on some platforms is complex.
- We maintain a directory of sample scripts to guide installers.

Tools on software.sandia.gov

- CVS: Source management.
- Bugzilla: Bug/feature tracking.
- Mailman: Mail list management.
- Bonsai: Interface to CVS/Bugzilla.
- Webserver: Webpages accessible from anywhere.
- Autotools: Autoconf/automake facilities.

Trilinos Availability/Support

- Trilinos and related packages are available via LGPL.
- Current release (4.0) is “click release”. Unlimited availability.
- Mail lists:
 - ◆ Each Trilinos package, including Trilinos itself, has four mail lists:
 - package-checkins@software.sandia.gov
 - CVS commit emails.
 - package-developers@software.sandia.gov
 - Mailing list for developers.
 - package-users@software.sandia.gov
 - Issues for package users.
 - package-announce@software.sandia.gov
 - Releases and other announcements specific to the package.
 - ◆ Additional list: Trilinos-Leaders@software.sandia.gov
 - ◆ <http://software.sandia.gov/mailman/listinfo/>

Conclusions

- Trilinos services to developers and users:
 - ◆ The 3 I's: Infrastructure, Interfaces, Implementations.
 - ◆ Simplifies installation, support for users of total collection.
 - ◆ Epetra & TSF promote common APIs across all other Trilinos packages.
 - ◆ Each package can be built, used independently, and exists as independent project.
- **Primary goals:**
 - ◆ **Rapid development and installation of robust numerical solvers.**
 - ◆ **High-quality production software for the critical path.**
- More information:
 - ◆ <http://software.sandia.gov>
 - ◆ <http://software.sandia.gov/trilinos>
 - ◆ Additional documentation at my website:
<http://www.cs.sandia.gov/~mheroux>.