PyTrilinos: A Python Interface to Trilinos, a Set of Object-Oriented Solver Packages

Bill Spotz
Sandia National Laboratories

SciPy 2005
Pasadena, CA    22 Sep 2005

With special thanks to
Marzio Sala, Eric Phipps, Alfred Lorber,
Mike Heroux, Jim Willenbring and Mike Phenow
Outline

• An Overview of Trilinos
  – Motivation
  – Philosophy & Infrastructure
  – Packages
• An Overview of PyTrilinos
  – Packages
  – Performance
• Summary
Trilinos Motivation

- Sandia does LOTS of solver work
- Challenges
  - Code reuse
  - Leverage development across projects
  - Consistent APIs
  - ASCI SQA/SQE requirements
- Bringing object-oriented tools to scientific computing
  - Frameworks, inheritance, operator overloading...
Trilinos Motivation

PDEs and Circuits
Trilinos\textsuperscript{1} is an evolving framework to address these challenges:

- Fundamental atomic unit is a package.
- Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages).
- Provides a common abstract solver API (Thyra package).
- Provides a ready-made package infrastructure (new\_package package):
  - Source code management (cvs, bonsai, bugzilla).
  - Build tools (autotools).
  - Automated regression testing (~20 builds, 5+ platforms, >3000 tests).
  - Communication tools (mailman mail lists).
- Specifies requirements and suggested practices for package 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 Development Team

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Packages/Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ross Bartlett</td>
<td>Lead Developer of Thyra</td>
<td>Thyra, Rythmos</td>
</tr>
<tr>
<td>Paul Boggs</td>
<td>Developer of Thyra</td>
<td></td>
</tr>
<tr>
<td>Todd Coffey</td>
<td>Lead Developer of Rythmos</td>
<td></td>
</tr>
<tr>
<td>Jason Cross</td>
<td>Developer of Jpetra</td>
<td>Jpetra</td>
</tr>
<tr>
<td>David Day</td>
<td>Developer of Komplex</td>
<td></td>
</tr>
<tr>
<td>Clark Dohrmann</td>
<td>Developer of CLAPS</td>
<td></td>
</tr>
<tr>
<td>Michael Gee</td>
<td>Developer of ML, NOX</td>
<td></td>
</tr>
<tr>
<td>Bob Heaphy</td>
<td>Lead developer of Trilinos SQA</td>
<td></td>
</tr>
<tr>
<td>Mike Heroux</td>
<td>Trilinos Project Leader</td>
<td>Epetra, AztecOO, Kokkos, Komplex, IFPACK, Thyra, Tpetra, Amoses, Belos, EpetraExt, Jpetra</td>
</tr>
<tr>
<td>Ulrich Hetmaniuk</td>
<td>Developer of Anasazi</td>
<td></td>
</tr>
<tr>
<td>Robert Hoekstra</td>
<td>Lead Developer of EpetraExt</td>
<td>Epetra, Thyra, Tpetra</td>
</tr>
<tr>
<td>Russell Hooper</td>
<td>Developer of NOX</td>
<td></td>
</tr>
<tr>
<td>Vicki Howle</td>
<td>Lead Developer of Meros</td>
<td>Belos and Thyra</td>
</tr>
<tr>
<td>Jonathan Hu</td>
<td>Developer of ML</td>
<td></td>
</tr>
<tr>
<td>Sarah Knepper</td>
<td>Developer of Komplex</td>
<td></td>
</tr>
<tr>
<td>Tammy Kolda</td>
<td>Lead Developer of NOX</td>
<td></td>
</tr>
<tr>
<td>Joe Kotulski</td>
<td>Lead Developer of Pliris</td>
<td></td>
</tr>
<tr>
<td>Rich Lehoucq</td>
<td>Developer of Anasazi and Belos</td>
<td></td>
</tr>
<tr>
<td>Kevin Long</td>
<td>Lead Developer of Thyra</td>
<td>Belos and Teuchos</td>
</tr>
<tr>
<td>Roger Pawlowski</td>
<td>Lead Developer of NOX</td>
<td></td>
</tr>
<tr>
<td>Michael Phenow</td>
<td>Trilinos Webmaster</td>
<td>New_Package</td>
</tr>
<tr>
<td>Eric Phipps</td>
<td>Developer of LOCA and NOX</td>
<td></td>
</tr>
<tr>
<td>Marzio Sala</td>
<td>Lead Developer of Didasko and IFPACK</td>
<td>Amoses</td>
</tr>
<tr>
<td>Andrew Salinger</td>
<td>Lead Developer of LOCA</td>
<td></td>
</tr>
<tr>
<td>Paul Sexton</td>
<td>Developer of Epetra and Tpetra</td>
<td></td>
</tr>
<tr>
<td>Bill Spotz</td>
<td>Lead Developer of PyTrilinos</td>
<td>Epetra, New_Package</td>
</tr>
<tr>
<td>Ken Stanley</td>
<td>Lead Developer of Amoses and New_Package</td>
<td></td>
</tr>
<tr>
<td>Heidi Thornquist</td>
<td>Lead Developer of Anasazi, Belos and Teuchos</td>
<td></td>
</tr>
<tr>
<td>Ray Tuminaro</td>
<td>Lead Developer of ML and Meros</td>
<td></td>
</tr>
<tr>
<td>Jim Willenbring</td>
<td>Developer of Epetra and New_Package, Trilinos library manager</td>
<td></td>
</tr>
<tr>
<td>Alan Williams</td>
<td>Developer of Epetra, EpetraExt, AztecOO, Tpetra</td>
<td></td>
</tr>
<tr>
<td>Linear Algebra Services</td>
<td>Epetra</td>
<td>Kokkos</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Linear Solvers</td>
<td>AztecOO</td>
<td>Amesos</td>
</tr>
<tr>
<td>Preconditioners</td>
<td>IFPACK</td>
<td>ML</td>
</tr>
<tr>
<td>Eigen solvers</td>
<td>Anasazi</td>
<td></td>
</tr>
<tr>
<td>Nonlinear Solvers</td>
<td>NOX</td>
<td></td>
</tr>
<tr>
<td>Continuation Algorithms</td>
<td>LOCA</td>
<td></td>
</tr>
<tr>
<td>Abstract Interfaces</td>
<td>Thyra</td>
<td>TSFCore</td>
</tr>
<tr>
<td>Utilities</td>
<td>Teuchos</td>
<td>EpetraExt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PyTrilinos - Next-Generation
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 configure script is vehicle for:

- Establishing interoperability of Trilinos components…
- Without compromising individual package autonomy.
Trilinos Packages: Epetra

- **Petra**: “foundation” (E for “essential”)
- **Linear Algebra Services**
  - Communicators: encapsulate parallelism
  - Maps: describe distribution of LA objects
  - Vectors/multivectors
  - Sparse graphs
  - Sparse matrices
  - Base classes for operators and matrices
  - Views and copies
Trilinos Packages: AztecOO

- Krylov subspace solvers: CG, GMRES, BiCGStab...
- Incomplete factorization preconditioners
- Aztec is the workhorse solver at Sandia
  - Extracted from MPSalsa reacting flow code
  - Dozens of Sandia applications
  - 1900+ external licenses
- AztecOO improves on Aztec by
  - Using Epetra objects
  - Providing more preconditioners/scalings
  - Enabling more sophisticated OO use
- AztecOO interfaces allow:
  - Continued use of Aztec for functionality
  - Introduction of new solver capabilities outside of Aztec
• Algebraic preconditioners
• Overlapping Schwarz preconditioners with incomplete factorizations, block relaxations, block direct solves
• Abstract matrix interface (including Epetra)
• Separates graph construction from factorizations
• Compatible with AztecOO, ML, Amesos
• Can be used by NOX and ML
Trilinos Packages: ML

• Multi-level preconditioners
  – Smoothed aggregation
  – Multi-grid
  – Domain decomposition

• Compatibilities:
  – Accepts any implementation of Epetra_RowMatrix
  – Implements Epetra_Operator interface . . . AztecOO

• Can be used completely independent of other Trilinos packages
Trilinos Packages: Amesos

• Distributed sparse direct solvers
• Challenge:
  – Many third-party direct solvers available
  – Different APIs, data formats
  – Interface can change with versions
• Amesos offers:
  – Single, consistent interface
  – Common look and feel for all classes
  – Separation from specific solver details
  – Internal data redistribution
• Third-party packages:
  – LAPACK, KLU, UMFPACK, SuperLU, SuperLU_DIST, MUMPS, ScaLAPACK, DSCPACK, PARDISO, WSMP
Trilinos Packages: NOX

- Suite of nonlinear solution methods
- Uses abstract vector and “group” interfaces:
  - Allows flexible selection and tuning of directions and line searches
  - Abstract vector & group interfaces for Epetra, AztecOO, ML, LAPACK and PETSc
- Controlled by flexible parameter list objects
Trilinos Packages: LOCA

• Library of Continuation Algorithms
• Continuation:
  – Zero-order, first-order, arc length
  – Multi-parameter, turning point, phase transition
  – Pitchfork- and Hopf-bifurcation
• Eigenvalue approximation
  – ARPACK or Anasazi
Trilinos Packages: EpetraExt

- Extensions to Epetra . . . useful, but nonessential
- Examples:
  - Graph/matrix view extraction
  - Zoltan interface
  - Sparse transpose
  - Singleton removal, static condensation filters
  - Overlapped graph constructors
  - Graph coloring algorithms
  - Matlab, MatrixMarket I/O functions
  - Etc…
Trilinos Packages: Anasazi

- Eigensolvers written in templated C++
- Generic interface to a collection of algorithms
- Interfaces are derived from vector and operator base classes
Trilinos Packages: Teuchos

- Utility package of useful tools
- Includes
  - LAPLACK, BLAS wrappers
  - Dense matrix & vector classes
  - FLOP counters, timers
  - Reference-counted pointers
  - Parameter lists
- Uses
  - Templates, STL
Trilinos Packages: Triutils

- **Trilinos Utilities** (intended for test harness, but sometimes useful elsewhere)
  - Matrix Galleries
  - Command-line parser
  - Input file reader
PyTrilinos

- Python interface to selected Trilinos packages
  - Epetra, AztecOO, IFPACK, ML, Amesos, NOX, LOCA, EpetraExt, TriUtils (and New_Package)
- Uses SWIG to generate wrappers
- Prerequisites
  - Python 2.3 or higher
  - Swig 1.3.23 or better
  - Numeric
- Python build system integrated into Trilinos configure/make system
  - Building Trilinos is not for the compiler-shy
  - To build PyTrilinos, simply add --enable-python (or --with-python) to the configure invocation
  - Interfaces will be built for enabled packages w/wrappers
  - make calls swig and then setup.py (distutils)
  - My MakefileVariables module
from PyTrilinos import Epetra  # MPI_Init, MPI_Finalize for MPI builds
comm = Epetra.PyComm()  # Epetra.SerialComm or Epetra.MpiComm
size = 4 * comm.NumProc()  # Scaled problem size
map = Epetra.Map(size, 0, comm)  # One of several constructors
v1 = Epetra.Vector(map)  # v1 is also a Numeric array!
print v1
v1.Print()

v1.shape = (2,2)
print v1

[ 0.  0.  0.  0.]
MyPID     GID     Value
  0     0     0
  0     1     0
  0     2     0
  0     3     0

[[ 0.  0.]
 [ 0.  0.]]
from PyTrilinos import Amesos, Triutils, Epetra

comm = Epetra.PyComm()
gallery = Triutils.CrsMatrixGallery("laplace_2d", comm)
gallery.Set("nx", 100)
gallery.Set("ny", 100)

problem = Epetra.LinearProblem(gallery.GetMatrix(),
                               gallery.GetStartingSolution(),
                               gallery.GetRHS())

factory = Amesos.Factory()
solver = factory.Create("SuperLU", problem)

amesosList = {"PrintTiming" : True, "PrintStatus" : True }
solver.SetParameters(amesosList)
solver.SymbolicFactorization()
solver.NumericFactorization()
solver.Solve()

soln = problem.GetLHS()

print "∥x_computed∥_2 =", soln.Norm2()
from PyTrilinos import IFPACK, AztecOO, Triutils
comm = Epetra.PyComm()
gallery = Triutils.CrsMatrixGallery("laplace_2d", comm)
gallery.Set("nx", 8)
gallery.Set("ny", 8)
matrix = gallery.GetMatrix()
lhs = gallery.GetStartingSolution()
rhs = gallery.GetRHS()
IFPACK.PrintSparsity(matrix, "matrix.ps")
solver = AztecOO.AztecOO(matrix, lhs, rhs)
solver.SetAztecOption(AztecOO.AZ_solver, AztecOO.AZ_cg)
solver.SetAztecOption(AztecOO.AZ_precond, AztecOO.AZ_dom_decomp)
solver.SetAztecOption(AztecOO.AZ_subdomain_solve, AztecOO.AZ_ilu)
solver.SetAztecOption(AztecOO.AZ_graph_fill, 1)
solver.Iterate(50, 1e-5)  # Max iteration = 50, tolerance = 1e-5
PyTrilinos Performance vs MATLAB

- CPU sec to fill \( n \times n \) dense matrix

<table>
<thead>
<tr>
<th>( n )</th>
<th>MATLAB</th>
<th>PyTrilinos</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00001</td>
<td>0.000416</td>
</tr>
<tr>
<td>100</td>
<td>0.0025</td>
<td>0.0357</td>
</tr>
<tr>
<td>1000</td>
<td>0.0478</td>
<td>3.857</td>
</tr>
</tbody>
</table>

- CPU sec to fill \( n \times n \) diagonal matrix

<table>
<thead>
<tr>
<th>( n )</th>
<th>MATLAB</th>
<th>PyTrilinos</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00006</td>
<td>0.000159</td>
</tr>
<tr>
<td>1000</td>
<td>0.00397</td>
<td>0.0059</td>
</tr>
<tr>
<td>10,000</td>
<td>0.449</td>
<td>0.060</td>
</tr>
<tr>
<td>50,000</td>
<td>11.05</td>
<td>0.313</td>
</tr>
<tr>
<td>100,000</td>
<td>50.98</td>
<td>0.603</td>
</tr>
</tbody>
</table>

- CPU sec for 100 MatVecs

<table>
<thead>
<tr>
<th>( n )</th>
<th>MATLAB</th>
<th>PyTrilinos</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.02</td>
<td>0.0053</td>
</tr>
<tr>
<td>100</td>
<td>0.110</td>
<td>0.0288</td>
</tr>
<tr>
<td>500</td>
<td>3.130</td>
<td>1.782</td>
</tr>
<tr>
<td>1000</td>
<td>12.720</td>
<td>7.150</td>
</tr>
</tbody>
</table>
PyTrilinos Performance vs Trilinos

- **Fine-grained script:**

<table>
<thead>
<tr>
<th>n</th>
<th>Trilinos</th>
<th>PyTrilinos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.010</td>
<td>0.15</td>
</tr>
<tr>
<td>10,000</td>
<td>0.113</td>
<td>0.241</td>
</tr>
<tr>
<td>100,000</td>
<td>0.280</td>
<td>1.238</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1.925</td>
<td>11.28</td>
</tr>
</tbody>
</table>

- **Course-grained script:**

![Graph showing CPU time comparison between Trilinos and PyTrilinos across different numbers of processors.](image)
PyTrilinos Performance

• Some Trilinos packages are designed for users to derive classes from pure virtual base classes
  – Epetra_Operator
  – Epetra_RowMatrix
  – NOX::Abstract::Interface . . .

• Numerical kernels (matvecs, nonlinear function evaluations) are therefore written by users.

• Using PyTrilinos, numerical kernels are therefore written in python (fine-grained . . . bad)

• If efficiency is a consideration,
  – Use array slice syntax
  – Use weave
  – Inefficient code is 20-100x slower
Summary

• Trilinos is a major software development project at Sandia National Laboratories
  – Interoperable, independent, object-oriented, parallel, sparse linear and nonlinear solver packages
  – Release 6.0: September, 2005
• PyTrilinos provides python access to selected packages
  – Numeric compatibility (NumArray?)
  – Still in early stages . . . portability, guinea pigs
  – Parallelism
  – Rapid prototyping
  – Unit testing
  – Application development