PyTrilinos: High-Performance Distributed-Memory Solvers for Python

Marzio Sala, ETH/D-INFK
W. Spotz (SNL), M. Heroux (SNL), E. Phipps (SNL)

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Outline

- Background: An overview of Trilinos
  - Motivation
  - Philosophy & infrastructure
- PyTrilinos: Python for scientific computing
  - Design goals
  - MPI support
  - Examples of usage
- Performances
  - PyTrilinos vs. MATLAB
  - PyTrilinos vs. Trilinos
- Summary

The Trilinos project

- Trilinos is a major software project (mostly) developed at Sandia National Labs (USA)
  – Project leader: M. Heroux (SNL)
  – Interoperable, independent, OO, parallel
  – Focus on sparse linear and nonlinear solvers
  – Current release: Sep-05, next release: Sep-06
- Goals:
  – Bringing object-oriented tools to scientific computing
  – Code reuse
  – Consistent APIs
  – Leverage development across projects

Trilinos Development Team

Ross Bartlett
Lead Developer of Thyra
Developer of Rythmos

Paul Boggs
Developer of Thyra

Todd Coffey
Lead Developer of Rythmos

Jason Cross
Developer of Jpetra

David Day
Developer of Komplex

Clark Dohrmann
Developer of CLAPS

Michael Gee
Developer of ML, NOX

Bob Heaphy
Lead developer of Trilinos SQA

Mike Heroux
Trilinos Project Leader
Lead Developer of Epetra, AztecOO, Kokkos, Komplex, IFPACK, Thyra, Tpetra
Developer of Amesos, Belos, EpetraExt, Jpetra

Ulrich Hetmaniuk
Developer of Anasazi

Robert Hoekstra
Lead Developer of EpetraExt
Developer of Epetra, Thyra, Tpetra

Russell Hooper
Developer of NOX

Vicki Howle
Lead Developer of Meros
Developer of Belos and Thyra

Jonathan Hu
Developer of ML

Sarah Knepper
Developer of Komplex

Tammy Kolda
Lead Developer of NOX

Joe Kotulski
Lead Developer of Pliris

Rich Lehoucq
Developer of Anasazi and Belos

Kevin Long
Lead Developer of Thyra,
Developer of Belos and Teuchos

Roger Pawlowski
Lead Developer of NOX

Michael Phenow
Trilinos Webmaster
Lead Developer of New_Package

Eric Phipps
Developer of LOCA and NOX

Marzio Sala
Lead Developer of Didasko and IFPACK
Developer of ML, Amesos

Andrew Salinger
Lead Developer of LOCA

Paul Sexton
Developer of Epetra and Tpetra

Bill Spotz
Lead Developer of PyTrilinos
Developer of Epetra, New_Package

Ken Stanley
Lead Developer of Amesos and New_Package

Heidi Thornquist
Lead Developer of Anasazi, Belos and Teuchos

Ray Tuminaro
Lead Developer of ML and Meros

Jim Willenbring
Developer of Epetra and New_Package.
Trilinos library manager

Alan Williams
Developer of Epetra, EpetraExt, AztecOO, Tpetra

The Trilinos project (2)

- Trilinos means "string of pearls":
  - Fundamental atomic unit is a package.
- Two-level structure 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
- Source code management (cvs, bonsai, bugzilla), build tools (autotools), automated testing, communication tools (mailing lists)

Some Packages

Linear Algebra Services	Epetra	Kokkos	Komplex	Galeri
Linear Solvers	AztecOO	Amesos	Pliris	Belos
Preconditioners	IFPACK	ML	Claps	Meros
Eigensolvers	Anasazi			
Nonlinear Solvers	NOX			
Continuation Algorithms	LOCA			
APIs	Thyra	TSFCore	TSFCoreUtils	TSFExtended
Utilities	Teuchos	EpetraExt	Triutils	Didasko

(hatched box) = Next-Generation

Why PyTrilinos?

- Trilinos is mostly in C++
  - Some "core" computations in C or FORTRAN
  - BLAS and LAPACK are used whenever possible
  - Serial/Parallel through MPI
- C++/C/FORTRAN are compiled languages
- Very efficient and powerful, however:
  - Classical compile-link-run cycle
  - No interactive usage
  - Sometimes difficult to experiment: poor flexibility, fundamental for rapid prototyping

Why PyTrilinos? (2)

Can we use interpreted languages for scientific computing?

Yes! However:

1. Which interactive language should be used?
2. Develop everything in one language ("pure" approach) or interface different languages?

Why PyTrilinos? (3)

- We use python
  - Mature, well-respected, portable
  - OO, very flexible
  - combines remarkable power with a very clean syntax
- "Pure" Python approach not feasible:
  - Scientific computing projects are based on pre-existing libraries, written in F77/F90/C/C++
  - Trilinos contains about 300.000 code lines (mostly C/C++), without considering BLAS, LAPACK, ScaLAPACK, and other libraries like METIS, ParMETIS, MPI, direct solvers, eigensolvers, ...
  - No interests in rewriting them

Python + Trilinos = PyTrilinos

- We develop interfaces to Trilinos:
  - Python has well-defined APIs to C
  - Tools like SWIG (www.swig.org) almost automatically create the bindings to/from C++ libraries and Python
- SWIG is easy-to-use, but not everything can be (or should be) wrapped
- PyTrilinos is not the full Trilinos in Python
- Only selected capabilities of selected packages

Trilinos vs. PyTrilinios

Linear Algebra Services	Epetra	Kokkos	Komplex	Galeri
Linear Solvers	AztecOO	Amesos	Pliris	Belos
Preconditioners	IFPACK	ML	Claps	Meros
Eigensolvers	Anasazi			
Nonlinear Solvers	NOX			
Continuation Algorithms	LOCA			
Abstract Interfaces	Thyra	TSFCore	TSFCoreUtils	TSFExtended
Utilities	Teuchos	EpetraExt	Triutils	Didasko

PyTrilinos | = Next-Generation

Python + Trilinos = PyTrilinos (2)

- PyTrilinos contains:
  - Sparse linear algebra (maps, vectors, graphs, matrices)
  - Matrix generation tools (like MATLAB's `gallery`)
  - Krylov solvers (CG, GMRES, ...)
  - Preconditioners (ILU-type, smoothed aggregation, ...)
  - Nonlinear solvers
  - Continuation methods
  - Various utilities (matrix generation, I/O, ...)
  - Much more
- PyTrilinos vectors inherit from NumArray vectors
  - Leverage of codes based on NumArray

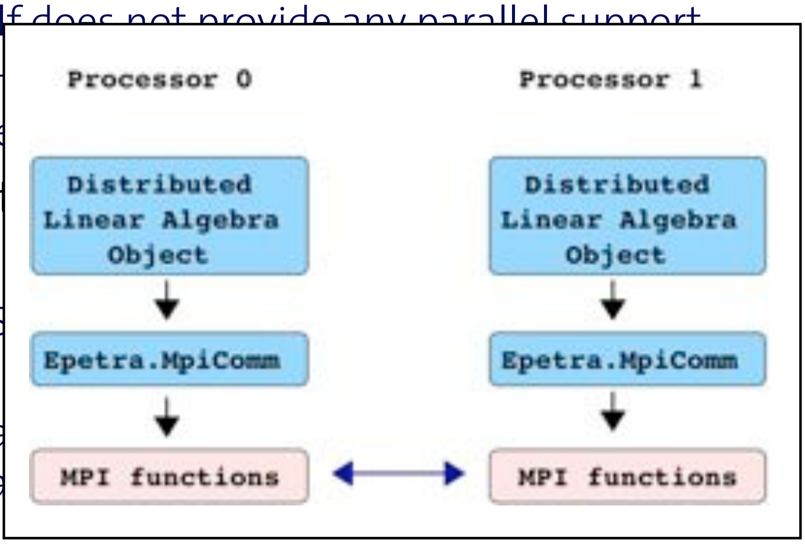
Virtual classes in PyTrilinos

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

- SWIG allows cross-language class derivation
  - The pure virtual class is defined in C++, the concrete implementation is in Python, the Solver interface is in C++, and calls the Python code to query the matrix

MPI support

- Parallel environments still constitute the most important field of application for most Trilinos algorithms:
  - Python itself does not provide any parallel support
  - MPI suppor
    ...), but none
- We decided t
  Python inter
  - wrap with S
    MPI_Finaliz
  - MPI comma
    communica



MPI support (contd.)

- Serial and parallel PyTrilinos scripts are virtually identical:

  >>> from PyTrilinos import Epetra

  >>> comm = Epetra.PyComm()

  >>> print comm.MyPID(), comm.NumProc()

  >>> comm.Barrier()
- To run in parallel:

  mpirun -np 4 python ./my-script.py
- Parallel runs are not interactive

PyTrilinos.Epetra

```
from PyTrilinos import Epetra  # MPI_Init, MPI_Finalize (if needed)
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/NumArray 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.]]
```

PyTrilinos.Epetra (cont.)

```
Comm                = Epetra.PyComm()
NumGlobalElements = 4 * Comm.NumProc()
Map                 = Epetra.Map(NumGlobalElements, 0, Comm)
Matrix              = Epetra.CrsMatrix(Epetra.Copy, Map, 0)
NumMyElements       = Map.NumMyElements()
MyGlobalElements    = Map.MyGlobalElements()

for i in MyGlobalElements:
    if i > 0:
      Matrix[i, i - 1] = -1
    if i < NumGlobalElements - 1:
      Matrix[i, i + 1] = -1
    Matrix[i, i] = 2.
Matrix.FillComplete()

for i in MyGlobalElements:
    print "PE%d: A(%d, %d) = %e" %(Comm.MyPID(), i, i, Matrix[i, i])
```


Example: Krylov solvers

```
#! /usr/bin/env python
from PyTrilinos import AztecOO, Triutils, Epetra

Comm = Epetra.PyComm()
Map, A, x, b, Exact = Triutils.ReadHB("fidap035.rua", Comm)

Solver = AztecOO.AztecOO(A, x, b)
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_icc)
Solver.SetAztecOption(AztecOO.AZ_graph_fill, 1)

Solver.Iterate(1550, 1e-5)
```

```
$ mpirun -np 4 python my-script.py
```

Example: direct solvers

```
#! /usr/bin/env python
from PyTrilinos import Amesos, Triutils, Epetra

Comm = Epetra.PyComm()
Map, A, x, b, Exact = Triutils.ReadHB("fidap035.rua", Comm)

Problem = Epetra.LinearProblem(A, x, b);
Factory = Amesos.Factory()
SolverType = "MUMPS"
Solver = Factory.Create(SolverType, Problem)
AmesosList = {
  "MaxProcs": 2,
  "PrintStatus": True
}
Solver.SetParameters(AmesosList)
Solver.SymbolicFactorization()
Solver.NumericFactorization()
Solver.Solve()
```

All solvers can be accessed in parallel through a Python script with no effort

```
$ mpirun -np 4 python my-script.py
```


PyTrilinos vs. MATLAB

n	MATLAB	PyTrilinos
10	0.00006	0.000159
1000	0.00397	0.0059
10,000	0.449	0.060
50,000	11.05	0.313
100,000	50.98	0.603

⇐**CPU sec to fill nxn diagonal matrix**

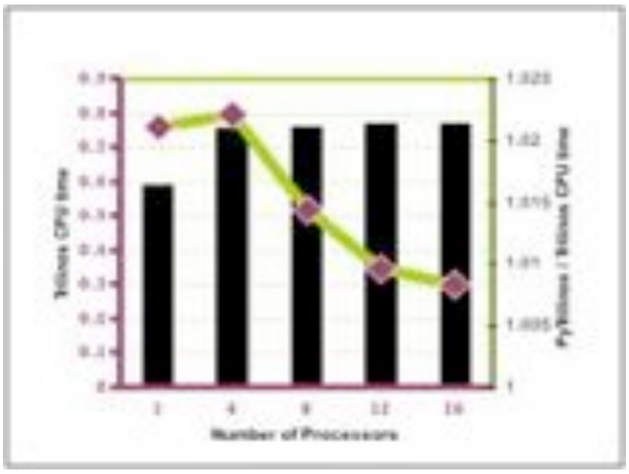
CPU sec for 100 MatVecs ⇒

n	MATLAB	PyTrilinos
50	0.02	0.0053
100	0.110	0.0288
500	3.130	1.782
1000	12.720	7.150

PyTrilinos vs. Trilinos

n	Trilinos	PyTrilinos
1000	0.010	0.15
10,000	0.113	0.241
100,000	0.280	1.238
1,000,000	1.925	11.28

<u>Fine-grained</u> scripts:
Creation of a diagonal sparse matrix



<u>Coarse-grained</u> scripts:
Distributed sparse matrix-vector product

Constant problem size / proc

PyTrilinos Performance

- Numerical kernels (matvecs, nonlinear function evaluations) are therefore written by users
- Using PyTrilinos, numerical kernels are therefore written in python (fine-grained . . . bad)

- Often, during development efficiency is not crucial
- If efficiency is a consideration,
  - Use array slice syntax
  - Use `weave` or other modules
  - Inefficient code is 20-100x slower

Conclusions

- Python interface to selected Trilinos packages:
  - Epetra, AztecOO, IFPACK, ML, Amesos, NOX, LOCA, EpetraExt ,Triutils, Galeri (and New_Package)
- Use SWIG to generate wrappers
- Prerequisites
  - Python 2.4 or higher
  - SWIG 1.3.29 or better
  - Numeric (Trilinos 6.0) or NumArray (Trilinos 7.0)
- Python build system integrated into Trilinos configure/make/make install system
  - Just add --enable-python to your configure script

Documentation

- The project is described in **PyTrilinos: High-Performance Distributed-Memory Solvers for Python**. MS, W. Spotz and M. Heroux. Submitted to ACM-TOMS.
- Web site: http://software.sandia.gov/trilinos/packages/PyTrilinos
- E-mail: marzio@inf.ethz.ch