Manycore Computing with Trilinos

Michael A. Heroux
Sandia National Laboratories
Collaborators:
Chris Baker, ORNL
Mark Hoemmen, H. Carter Edwards, SNL
Additional contributions:
Barry Smith, ANL, Matt Knepley, U Chicago

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

- What can Trilinos do for you?
- Trilinos’ software organization
- Whirlwind tour of Trilinos packages
- Getting started: “How do I…?”
- Preparation for hands-on tutorial
What can Trilinos do for you?
What is Trilinos?

- Object-oriented software framework for...
- Solving big complex science & engineering problems
- More like LEGO™ bricks than Matlab™
Trilinos Contributors

Chris Baker
Ross Bartlett
Pavel Bochev
Paul Boggs
Erik Boman
Lee Buermann
Cedric Chevalier
Todd Coffey
Eric Cyr
David Day
Karen Devine
Clark Dohrmann
Kelly Fermoyle
David Gay
Mike Heroux
Ulrich Hetmaniuk
Mark Hoemmen
Russell Hooper
Jonathan Hu
Joe Kotulski
Rich Lehoucq
Kevin Long
Karla Morris
Kurtis Nusbaum
Roger Pawlowski
Brent Perschbacher
Eric Phipps
Siva Rajamanickam
Lee Ann Riesen
Marzio Sala
Andrew Salinger
Chris Siefert
Bill Spotz
Dan Sunderland
Heidi Thornquist
Ray Tuminaro
Jim Willenbring
Alan Williams
Past Contributors

Jason Cross
Michael Gee
Esteban Guillen
Bob Heaphy
Robert Hoekstra
Vicki Howle
Kris Kampshoff
Ian Karlin
Sarah Knepper
Tammy Kolda
Joe Outzen
Mike Phenow
Paul Sexton
Bob Shuttleworth
Ken Stanley
Michael Wolf
Background/Motivation
Trilinos

Optimal Kernels to Optimal Solutions:
- Geometry, Meshing
- Discretizations, Load Balancing.
- Scalable Linear, Nonlinear, Eigen, Transient, Optimization, UQ solvers.
- Scalable I/O, GPU, Manycore

- R&D 100 Winner
- 6300 Registered Users.
- 21,000 Downloads.
- Open Source.

Laptops to Leadership systems

- 60 Packages.
- Binary distributions:
  - Cray LIBSCI
  - Debian, Ubuntu
  - Intel (in process)

Transforming Computational Analysis To Support High Consequence Decisions

Systems of systems
Optimization under Uncertainty
Quantify Uncertainties/Systems Margins
Optimization of Design/System
Robust Analysis with Parameter Sensitivities
Accurate & Efficient Forward Analysis
Forward Analysis

Each stage requires greater performance and error control of prior stages: Always will need: more accurate and scalable methods, more sophisticated tools.
Applications

- All kinds of physical simulations:
  - Structural mechanics (statics and dynamics)
  - Circuit simulations (physical models)
  - Electromagnetics, plasmas, and superconductors
  - Combustion and fluid flow (at macro- and nanoscales)

- Coupled / multiphysics models

- Data and graph analysis
  - Even gaming!
Trilinos Strategic Goals

- Scalable Computations: As problem size and processor counts increase, the cost of the computation will remain nearly fixed.

- Hardened Computations: Never fail unless problem essentially intractable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.

- Full Vertical Coverage: Provide leading edge enabling technologies through the entire technical application software stack: from problem construction, solution, analysis and optimization.

- *Grand* Universal Interoperability: All Trilinos packages, and important external packages, will be interoperable, so that any combination of packages and external software (e.g., PETSc, Hypre) that makes sense algorithmically will be possible within Trilinos.

- Universal Accessibility: All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python and the Web, and from the desktop to the latest scalable systems.

- Universal Solver RAS: Trilinos will be:
  - Reliable: Leading edge hardened, scalable solutions for each of these applications
  - Available: Integrated into every major application at Sandia
  - Serviceable: “Self-sustaining”.

Algorithmic Goals

Software Goals
Capability Leaders: Layer of Proactive Leadership

- Areas:
  - Framework, Tools & Interfaces (J. Willenbring).
  - Software Engineering Technologies and Integration (R. Bartlett).
  - Discretizations (P. Bochev).
  - Scalable Linear Algebra (M. Heroux).
  - Linear & Eigen Solvers (J. Hu).
  - Nonlinear, Transient & Optimization Solvers (A. Salinger).
  - Scalable I/O: (R. Oldfield)

- Each leader provides strategic direction across all Trilinos packages within area.
Unique features of Trilinos

- Huge library of algorithms
  - Linear and nonlinear solvers, preconditioners, …
  - Optimization, transients, sensitivities, uncertainty, …

- Growing support for multicore & hybrid CPU/GPU
  - Built into the new Tpetra linear algebra objects
    - Therefore into iterative solvers with zero effort!
  - Unified intranode programming model
  - Spreading into the whole stack:
    - Multigrid, sparse factorizations, element assembly…

- Growing support for mixed and arbitrary precisions
  - Don’t have to rebuild Trilinos to use it!

- Growing support for huge (> 2B unknowns) problems
Trilinos’ software organization
Trilinos is made of packages

- Not a monolithic piece of software
  - Like LEGO™ bricks, not Matlab™

- Each package:
  - Has its own development team and management
  - Makes its own decisions about algorithms, coding style, etc.
  - May or may not depend on other Trilinos packages

- Trilinos is not “indivisible”
  - You don’t need all of Trilinos to get things done
  - Any subset of packages can be combined and distributed
  - Current public release contains ~50 of the 55+ Trilinos packages

- Trilinos top layer framework
  - Not a large amount of source code: ~1.5%
  - Manages package dependencies
    - Like a GNU/Linux package manager
  - Runs packages’ tests nightly, and on every check-in

- Package model supports multifrontal development
# Trilinos Package Summary

<table>
<thead>
<tr>
<th>Objective</th>
<th>Package(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Discretizations</strong></td>
<td></td>
</tr>
<tr>
<td>Meshing &amp; Discretizations</td>
<td>STK, Intrepid, Pamgen, Sundance, ITAPS, Mesquite</td>
</tr>
<tr>
<td>Time Integration</td>
<td>Rythmos</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td>Automatic Differentiation</td>
<td>Sacado</td>
</tr>
<tr>
<td>Mortar Methods</td>
<td>Moertel</td>
</tr>
<tr>
<td><strong>Services</strong></td>
<td></td>
</tr>
<tr>
<td>Linear algebra objects</td>
<td>Epetra, Jpetra, Tpetra, Kokkos</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Thyra, Stratimikos, RTOp, FEI, Shards</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Zoltan, Isorropia, Zoltan2</td>
</tr>
<tr>
<td>“Skins”</td>
<td>PyTrilinos, WebTrilinos, ForTrilinos, Ctrlilinos, Optika</td>
</tr>
<tr>
<td>C++ utilities, I/O, thread API</td>
<td>Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx, Trios</td>
</tr>
<tr>
<td><strong>Solvers</strong></td>
<td></td>
</tr>
<tr>
<td>Iterative linear solvers</td>
<td>AztecOO, Belos, Komplex</td>
</tr>
<tr>
<td>Direct sparse linear solvers</td>
<td>Amesos, Amesos2</td>
</tr>
<tr>
<td>Direct dense linear solvers</td>
<td>Epetra, Teuchos, Pliris</td>
</tr>
<tr>
<td>Iterative eigenvalue solvers</td>
<td>Anasazi, Rbgen</td>
</tr>
<tr>
<td>ILU-type preconditioners</td>
<td>AztecOO, IFPACK, Ifpack2</td>
</tr>
<tr>
<td>Multilevel preconditioners</td>
<td>ML, CLAPS, Muelu</td>
</tr>
<tr>
<td>Block preconditioners</td>
<td>Meros, Teko</td>
</tr>
<tr>
<td>Nonlinear system solvers</td>
<td>NOX, LOCA, Piro</td>
</tr>
<tr>
<td>Optimization (SAND)</td>
<td>MOOCHC, Aristos, TriKota, Globipack, Optipack</td>
</tr>
<tr>
<td>Stochastic PDEs</td>
<td>Stokhos</td>
</tr>
</tbody>
</table>
Interoperability vs. Dependence
(“Can Use”) (”Depends On”)

- Although most Trilinos packages have no explicit dependence, often packages must interact with *some* other packages:
  - NOX needs operator, vector and linear solver objects.
  - AztecOO needs preconditioner, matrix, operator and vector objects.
  - Interoperability is enabled at configure time.
  - Trilinos *cmake* system is vehicle for:
    - Establishing interoperability of Trilinos components…
    - Without compromising individual package autonomy.
    - Trilinos\_ENABLE\_ALL\_OPTIONAL\_PACKAGES option

- Architecture supports simultaneous development on many fronts.
Software Development and Delivery
“Are C++ templates safe? No, but they are good.”

Compile-time Polymorphism
Templates and Sanity upon a shifting foundation

Software delivery:
• Essential Activity

How can we:
• Implement mixed precision algorithms?
• Implement generic fine-grain parallelism?
• Support hybrid CPU/GPU computations?
• Support extended precision?
• Explore redundant computations?
• Prepare for both exascale “swim lanes”?

C++ templates only sane way:
• Moving to completely templated Trilinos libraries.
• Other important benefits.
• A usable stack exists now in Trilinos.

Template Benefits:
– Compile time polymorphism.
– True generic programming.
– No runtime performance hit.
– Strong typing for mixed precision.
– Support for extended precision.
– Many more…

Template Drawbacks:
– Huge compile-time performance hit:
  • But good use of multicore :)  
  • Eliminated for common data types.
– Complex notation:
  • Esp. for Fortran & C programmers).
  • Can insulate to some extent.
### Solver Software Stack

| Optimization | Find $u \in \mathbb{R}^n$ that minimizes $g(u)$  
| Constrained: | Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$  
| Bifurcation Analysis | Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$  
| For $F(x, u) = 0$ find space $u \in U$ if $\partial F/\partial x$  
| Transient Problems | Solve $f(\dot{x}(t), x(t), t) = 0$  
| DAEs/ODEs: | $t \in [0, T], x(0) = x_0, \dot{x}(0) = x_0'$  
| | for $x(t) \in \mathbb{R}^n, t \in [0, T]$  
| Nonlinear Problems | Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}$  
| Solve $F(x) = 0$ $x \in \mathbb{R}^n$  
| Linear Problems | Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$  
| Linear Equations: | Solve $Ax = b$ for $x \in \mathbb{R}^n$  
| Eigen Problems: | Solve $Av = \lambda Bv$ for (all) $v \in \mathbb{R}^n, \lambda \in \mathbb{R}$  
| Distributed Linear Algebra | Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m \times n}, G \in \mathbb{S}^{m \times n}$  
| Matrix/Graph Equations: | Compute $y = \alpha x + \beta w; \alpha = (x, y); x, y \in \mathbb{R}^n$  
| Vector Problems: |  

### Sensitivities

- MOOCHO
- LOCA
- Rythmos
- NOX
- Anasazi
- Ifpack, ML, etc...
- AztecOO
- Epetra
- Teuchos
### Solver Software Stack

<table>
<thead>
<tr>
<th>Phase I packages</th>
<th>Phase II packages</th>
<th>Phase III packages: Manycore*, templated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unconstrained:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constrained:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bifurcation Analysis</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transient Problems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAEs/ODEs:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nonlinear Problems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Linear Problems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear Equations:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eigen Problems:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Distributed Linear Algebra</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix/Graph Equations:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector Problems:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Sensitivities

**MOOCHO**

**LOCA**

**T-LOCA**

**Rythmos**

**NOX**

**T-NOX**

**Anasazi**

**AztecOO**

**Belos**

**Belos**, **Ifpack**, **ML**, etc...

**T-Ifpack**

**T-ML**, etc...

**T-NOX**

**Epetra**

**Tpetra**

**Tpetra**

**Kokkos**

**Kokkos**

**Teuchos**

---

**Solver Software Stack**

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unconstrained:</td>
<td>Find $u \in \mathbb{R}^n$ that minimizes $g(u)$</td>
<td></td>
</tr>
<tr>
<td>Constrained:</td>
<td>Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$</td>
<td></td>
</tr>
<tr>
<td><strong>Bifurcation Analysis</strong></td>
<td>Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$</td>
<td></td>
</tr>
<tr>
<td><strong>Transient Problems</strong></td>
<td>Solve $f(\dot{x}(t), x(t), t) = 0$</td>
<td></td>
</tr>
<tr>
<td>DAEs/ODEs:</td>
<td>$t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for $x(t) \in \mathbb{R}^n, t \in [0, T]$</td>
<td></td>
</tr>
<tr>
<td><strong>Nonlinear Problems</strong></td>
<td>Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Solve $F(x) = 0$ $x \in \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td><strong>Linear Problems</strong></td>
<td>Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m\times n}$</td>
<td></td>
</tr>
<tr>
<td>Linear Equations:</td>
<td>Solve $Ax = b$ for $x \in \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td>Eigen Problems:</td>
<td>Solve $Av = \lambda Bv$ for (all) $v \in \mathbb{R}^n, \lambda \in \mathbb{C}$</td>
<td></td>
</tr>
<tr>
<td><strong>Distributed Linear Algebra</strong></td>
<td>Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m\times n}, G \in \mathbb{S}^{m\times n}$</td>
<td></td>
</tr>
<tr>
<td>Matrix/Graph Equations:</td>
<td>Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td>Vector Problems:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Solver Software Stack**

19
Whirlwind Tour of Packages

Core Utilities
Discretizations
Methods
Solvers
Intrepid offers an **innovative software design** for compatible discretizations:

- allows access to FEM, FV and FD methods using a common API
- supports **hybrid discretizations** (FEM, FV and FD) on unstructured grids
- supports a variety of cell shapes:
  - standard shapes (e.g. tets, hexes): high-order finite element methods
  - arbitrary (polyhedral) shapes: low-order mimetic finite difference methods
- enables optimization, error estimation, V&V, and UQ using fast invasive techniques (direct support for cell-based derivative computations or via automatic differentiation)

Developers: Pavel Bochev and Denis Ridzal
Rythmos

- Suite of time integration (discretization) methods
  - Includes: backward Euler, forward Euler, explicit Runge-Kutta, and implicit BDF at this time.
  - Native support for operator split methods.
  - Highly modular.
  - Forward sensitivity computations will be included in the first release with adjoint sensitivities coming in near future.

Developers: Todd Coffey, Roscoe Bartlett
Whirlwind Tour of Packages

Discretizations    Methods    Core    Solvers
Sacado: Automatic Differentiation

- Efficient OO based AD tools optimized for element-level computations

- Applies AD at “element”-level computation
  - “Element” means finite element, finite volume, network device,…

- Template application’s element-computation code
  - Developers only need to maintain one templated code base

- Provides three forms of AD
  - **Forward Mode:** \((x, V) \rightarrow (f, \frac{\partial f}{\partial x}V)\)
    - Propagate derivatives of intermediate variables w.r.t. independent variables forward
    - Directional derivatives, tangent vectors, square Jacobians, \(\frac{\partial f}{\partial x}\) when \(m \geq n\).
  - **Reverse Mode:** \((x, W) \rightarrow (f, W^T\frac{\partial f}{\partial x})\)
    - Propagate derivatives of dependent variables w.r.t. intermediate variables backwards
    - Gradients, Jacobian-transpose products (adjoints), \(\frac{\partial f}{\partial x}\) when \(n > m\).
  - **Taylor polynomial mode:**
    \[x(t) = \sum_{k=0}^{d} x_k t^k \quad \rightarrow \quad \sum_{k=0}^{d} f_k t^k = f(x(t)) + O(t^{d+1}), \quad f_k = \frac{1}{k!} \frac{d^k}{dt^k} f(x(t))\]
  - Basic modes combined for higher derivatives.

Developers: Eric Phipps, David Gay
Whirlwind Tour of Packages

Discretizations       Methods       Core       Solvers
Teuchos

- Portable 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 / arrays, and more…
- Takes advantage of advanced features of C++:
  - Templates
  - Standard Template Library (STL)
- Teuchos::ParameterList:
  - Allows easy control of solver parameters.
  - XML format input/output.

Developers: Chris Baker, Roscoe Barlett, Heidi Thornquist, Mike Heroux, Paul Sexton, Kris Kampshoff, Chris Baker, Mark Hoemmen
Petra provides a “common language” for distributed linear algebra objects (operator, matrix, vector).

- Petra\(^1\) 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 2 implementations under development.

\(^1\)Petra is Greek for “foundation”.
Petra Implementations

- **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.
  - Builds on top of Kokkos manycore node library.

Developers: Chris Baker, Mike Heroux, Rob Hoekstra, Alan Williams
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
Data Services for Dynamic Applications

- Dynamic load balancing
- Graph coloring
- Data migration
- Matrix ordering

Partitioners:

Geometric (coordinate-based) methods:
- Recursive Coordinate Bisection (Berger, Bokhari)
- Recursive Inertial Bisection (Taylor, Nour-Omid)
- Space Filling Curves (Peano, Hilbert)
- Refinement-tree Partitioning (Mitchell)

Hypergraph and graph (connectivity-based) methods:
- Hypergraph Repartitioning PaToH (Catalyurek)
- Zoltan Hypergraph Partitioning
- ParMETIS (U. Minnesota)
- Jostle (U. Greenwich)

Developers: Karen Devine, Eric Boman, Siva R., LeAnn Riesen
Thyra

- High-performance, abstract interfaces for linear algebra
- Offers flexibility through abstractions to algorithm developers
- Linear solvers (Direct, Iterative, Preconditioners)
  - Abstraction of basic vector/matrix operations (dot, axpy, mv).
  - Can use any concrete linear algebra library (Epetra, PETSc, BLAS).
- Nonlinear solvers (Newton, etc.)
  - Abstraction of linear solve (solve $Ax=b$).
  - Can use any concrete linear solver library:
    - AztecOO, Belos, ML, PETSc, LAPACK
- Transient/DAE solvers (implicit)
  - Abstraction of nonlinear solve.
  - ... and so on.

Developers: Roscoe Bartlett, Kevin Long
“Skins”

- PyTrilinos provides Python access to Trilinos packages
  - Uses SWIG to generate bindings.
  - Epetra, AztecOO, IFPACK, ML, NOX, LOCA, Amesos and NewPackage are supported.

Developer: Bill Spotz

- CTrilinos: C wrapper (mostly to support ForTrilinos).
- ForTrilinos: OO Fortran interfaces.

Developers: Nicole Lemaster, Damian Rouson

- WebTrilinos: Web interface to Trilinos
  - Generate test problems or read from file.
  - Generate C++ or Python code fragments and click-run.
  - Hand modify code fragments and re-run.
  - Will use during hands-on.

Developers: Ray Tuminaro, Jonathan Hu, Marzio Sala, Jim Willenbring
Whirlwind Tour of Packages

Discretizations       Methods       Core       Solvers
Amesos

- Interface to direct solvers for distributed sparse linear systems (KLU, UMFPACK, SuperLU, MUMPS, ScaLAPACK)

- Challenges:
  - No single solver dominates
  - Different interfaces and data formats, serial and parallel
  - Interface often changes 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
  - Native solvers: KLU and Paraklete

Developers: Ken Stanley, Marzio Sala, Tim Davis
Amesos2/KLU2

- Second-generation sparse direct solvers package

- Unified interface to multiple solvers, just like Amesos
- KLU2: Default direct solver.
- Amesos2 features:
  - Supports matrices of arbitrary scalar and index types
  - Path to multicore CPU and hybrid CPU/GPU solvers
  - Thread safe: multiple solvers can coexist on the same node
    - Supports new intranode hybrid direct / iterative solver ShyLU
  - Abstraction from specific sparse matrix representation
    - Supports Epetra and Tpetra
    - Extensible to other matrix types

- Just released.

Developers: Eric Bavier, Erik Boman, and Siva Rajamanickam
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, Alan Williams, Ray Tuminaro
Belos

- Next-generation linear solver library, written in templated C++.

- Provide a generic framework for developing iterative algorithms for solving large-scale, linear problems.

- Algorithm implementation is accomplished through the use of traits classes and abstract base classes:
  - Operator-vector products: Belos::MultiVecTraits, Belos::OperatorTraits
  - Orthogonalization: Belos::OrthoManager, Belos::MatOrthoManager
  - Status tests: Belos::StatusTest, Belos::StatusTestResNorm
  - Iteration kernels: Belos::Iteration
  - Linear solver managers: Belos::SolverManager

- AztecOO provides solvers for $Ax = b$, what about solvers for:
  - Simultaneously solved systems w/ multiple-RHS: $AX = B$
  - Sequentially solved systems w/ multiple-RHS: $AX_i = B_i, i=1,...,t$
  - Sequences of multiple-RHS systems: $A_iX_i = B_i, i=1,...,t$

- Many advanced methods for these types of linear systems
  - Block methods: block GMRES [Vital], block CG/BICG [O’Leary]
  - “Seed” solvers: hybrid GMRES [Nachtigal, et al.]
  - Restarting techniques, orthogonalization techniques, …

Developers: Heidi Thornquist, Mike Heroux, Mark Hoemmen, Mike Parks, Rich Lehoucq
IFPACK: Algebraic Preconditioners

- Overlapping Schwarz preconditioners with incomplete factorizations, block relaxations, block direct solves.
- 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, ML, Amesos. Can be used by NOX and ML.

Developers: Marzio Sala, Mike Heroux, Siva Rajamanickam, Alan Williams
Ifpack2

- Second-generation IFPACK

- Highly optimized ILUT (60x faster than IFPACK’s!)
- Computed factors fully exploit multicore CPU / GPU
  - Via Tpetra
- Path to hybrid-parallel factorizations
- Arbitrary precision and complex arithmetic support

Developers: Mike Heroux, Siva Rajamanickam, Alan Williams, Michael Wolf
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, Belos, Anasazi.

- Can also be used completely independent of other Trilinos packages.

Developers: Ray Tuminaro, Jeremie Gaidamour, Jonathan Hu, Marzio Sala, Chris Siefert
Anasazi

- Next-generation eigensolver library, written in templated C++.

- Provide a generic framework for developing iterative algorithms for solving large-scale eigenproblems.

- Algorithm implementation is accomplished through the use of traits classes and abstract base classes:
  - Operator-vector products: Anasazi::MultiVecTraits, Anasazi::OperatorTraits
  - Orthogonalization: Anasazi::OrthoManager, Anasazi::MatOrthoManager
  - Status tests: Anasazi::StatusTest, Anasazi::StatusTestResNorm
  - Iteration kernels: Anasazi::EigenSolver
  - Eigensolver managers: Anasazi::SolverManager
  - Eigenproblem: Anasazi::Eigenproblem
  - Sort managers: Anasazi::SortManager

- Currently has solver managers for three eigensolvers:
  - Block Krylov-Schur
  - Block Davidson
  - LOBPCG

- Can solve:
  - standard and generalized eigenproblems
  - Hermitian and non-Hermitian eigenproblems
  - real or complex-valued eigenproblems

Developers: Heidi Thornquist, Mike Heroux, Chris Baker, Rich Lehoucq, Ulrich Hetmaniuk
NOX: Nonlinear Solvers

- Suite of nonlinear solution methods

Broyden’s Method
\[ M_B = f(x_c) + B_c d \]

Newton’s Method
\[ M_N = f(x_c) + J_c d \]

Tensor Method
\[ M_T = f(x_c) + J_c d + \frac{1}{2} T d d \]

Globalizations
- Line Search
  - Interval Halving
  - Quadratic
  - Cubic
  - More’-Thuente

- Trust Region
  - Dogleg
  - Inexact Dogleg

Jacobian Estimation
- Graph Coloring
- Finite Difference
- Jacobian-Free
- Newton-Krylov

http://trilinos.sandia.gov/packages/nox

Developers: Tammy Kolda, Roger Pawlowski

Implementation
- Parallel
- OO-C++
- Independent of the linear algebra package!
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
MOOCHO & Aristos

▪ MOOCHO: Multifunctional Object-Oriented arCHitecture for Optimization

  ▪ Large-scale invasive simultaneous analysis and design (SAND) using reduced space SQP methods.

   Developer: Roscoe Bartlett

▪ Aristos: Optimization of large-scale design spaces

  ▪ Invasive optimization approach based on full-space SQP methods.
  ▪ Efficiently manages inexactness in the inner linear system solves.

   Developer: Denis Ridzal
Solver Collaborations: ANAs, LALs and APPs
An **abstract numerical algorithm** (ANA) is a numerical algorithm that can be expressed solely in terms of vectors, vector spaces, and linear operators.

### Example Linear ANA (LANA) : Linear Conjugate Gradients

**Given:**

- \( A \in \mathcal{X} \to \mathcal{X} \): s.p.d. linear operator
- \( b \in \mathcal{X} \): right hand side vector

Find vector \( x \in \mathcal{X} \) that solves \( Ax = b \)

#### Linear Conjugate Gradient Algorithm

1. Compute \( r^{(0)} = b - Ax^{(0)} \) for the initial guess \( x^{(0)} \).
2. For \( i = 1, 2, \ldots \)
   - \( \rho_{i-1} = \langle r^{(i-1)}, r^{(i-1)} \rangle \)
   - \( \beta_i = \rho_i / \rho_{i-1} \) (\( \beta_0 = 0 \))
   - \( p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)} \) (\( p^{(1)} = r^{(1)} \))
   - \( q^{(i)} = Ap^{(i)} \)
   - \( \gamma_i = \langle p^{(i)}, q^{(i)} \rangle \)
   - \( \alpha_i = \rho_i / \gamma_i \)
   - \( x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)} \)
   - \( r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)} \)

3. Check convergence; continue if necessary

### Types of operations
- Linear operator applications
- Vector-vector operations
- Scalar operations
- Scalar product <x,y> defined by vector space

### Types of objects
- Linear Operators
  - \( A \)
- Vectors
  - \( r, x, p, q \)
- Scalars
  - \( \rho, \beta, \gamma, \alpha \)
- Vector spaces
  - \( \mathcal{X} \)
Introducing Stratimikos

• **Stratimikos** created Greek words "stratigiki“ (strategy) and "grammikos“ (linear)

• Defines class Thyra::DefaultLinearSolverBuilder.

• Provides common access to:
  • Linear Solvers: Amesos, AztecOO, Belos, …
  • Preconditioners: Ifpack, ML, …

  • Reads in options through a parameter list (read from XML?)

  • Accepts any linear system objects that provide
    • Epetra_Operator / Epetra_RowMatrix view of the matrix
    • SPMD vector views for the RHS and LHS (e.g. Epetra_[Multi]Vector objects)

• Provides uniform access to linear solver options that can be leveraged across multiple applications and algorithms

---

**Key Points**

• Stratimikos is an important building block for creating more sophisticated linear solver capabilities!
Stratimikos Parameter List and Sublists

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

Where to start?
http://trilinos.sandia.gov
“How do I…?”

- Build my application with Trilinos?
- Learn about common Trilinos programming idioms?
- Download / find an installation of Trilinos?
- Find documentation and help?
Building your app with Trilinos

If you are using Makefiles:
- Makefile.export system

If you are using CMake:
- CMake FIND_PACKAGE
Using CMake to build with Trilinos

- **CMake**: Cross-platform build system
  - Similar function as the GNU Autotools
- Trilinos uses CMake to build
- You don’t have to use CMake to build with Trilinos
- But if you do:
  - `FIND_PACKAGE(Trilinos ...)`
  - Example CMake script in hands-on demo
- I find this much easier than hand-writing Makefiles
Export Makefile System

Once Trilinos is built, how do you link against the application?

There are a number of issues:

• Library link order:
  • -lnoxepetra -lnox -lepetra -lteuchos -lblas -llapack

• Consistent compilers:
  • g++, mpiCC, icc...

• Consistent build options and package defines:
  • g++ -g -O3 -D HAVE_MPI -D _STL_CHECKED

Answer: Export Makefile system
Why Export Makefiles are Important

- Trilinos has LOTS of packages
- As package dependencies (especially optional ones) are introduced, more maintenance is required by the top-level packages:

```
NOX -> ML
   ^      ^
   |      |  
   Ifpack Epetra
   ^      ^
   |      |  
   EpetraExt

NOX -> Amesos
       ^
       |  
       New Library

NOX -> SuperLU
       ^
       |  
       New Library
```

NOX either must:
- Account for the new libraries in it’s configure script (unscalable)
- Depend on direct dependent packages to supply them through export Makefiles
# A Makefile that your application can use if you want to build with Epetra.
# (Excerpt from $(TRILINOS_INSTALL_DIR)/include/Makefile.client.Epetra)
#
# Include the Trilinos export Makefile from package=Epetra.
include $(TRILINOS_INSTALL_DIR)/include/Makefile.export.Epetra

# Add the Trilinos installation directory to the library and header search paths.
LIB_PATH = $(TRILINOS_INSTALL_DIR)/lib
INCLUDE_PATH = $(TRILINOS_INSTALL_DIR)/include $(CLIENT_EXTRA_INCLUDES)

# Set the C++ compiler and flags to those specified in the export Makefile.
# This ensures your application is built with the same compiler and flags
# with which Trilinos was built.
CXX = $(EPETRA_CXX_COMPILER)
CXXFLAGS = $(EPETRA_CXX_FLAGS)

# Add the Trilinos libraries, search path, and rpath to the
# linker command line arguments
LIBS = $(CLIENT_EXTRA_LIBS) $(SHARED_LIB_RPATH_COMMAND) \  
   $(EPETRA_LIBRARIES) \  
   $(EPETRA_TPL_LIBRARIES) \  
   $(EPETRA_EXTRA_LD_FLAGS)

#
# Rules for building executables and objects.
#
%.exe : %.o $(EXTRA_OBJS)
   $(CXX) -o $@ $(LDFLAGS) $(CXXFLAGS) $< $(EXTRA_OBJS) -L$(LIB_PATH) $(LIBS)

%.o : %.cpp
   $(CXX) -c -o $@ $(CXXFLAGS) -is$(INCLUDE_PATH) $(EPETRA_TPL_INCLUDES) $<
Related Efforts
PETSc on GPUs

Matthew Knepley

Computation Institute
University of Chicago
Department of Molecular Biology and Physiology
Rush University Medical Center

Summary for Release 3.2
September, 2011
PETSc now has support for Krylov solves on the GPU

- `with-cuda=1`  `with-cusp=1`  `with-thrust=1`
  Also possibly `with-precision=single`

New classes VECCUDA and MATAIJCUDA

Just change type on command line, `-vec_type veccuda`

Uses Thrust and Cusp libraries from Nvidia guys

Does not communicate vectors during solve
Strategy: Define a new Vec implementation

- Uses **Thrust** for data storage and operations on GPU
- Supports full PETSc Vec interface
- Inherits PETSc scalar type
- Can be activated at runtime, -vec_type cuda
- PETSc provides memory coherence mechanism
Also define new Mat implementations

- Uses **Cusp** for data storage and operations on GPU
- Supports full PETSc Mat interface, some ops on CPU
- Can be activated at runtime, `-mat_type aijcuda`
- Notice that parallel matvec necessitates off-GPU data transfer
Solvers

Solvers come for Free

**Preliminary Implementation of PETSc Using GPU**, Minden, Smith, Knepley, 2010

- All linear algebra types work with solvers
- Entire solve can take place on the GPU
- Only communicate scalars back to CPU

- GPU communication cost could be amortized over several solves
- Preconditioners are a problem
- Cusp has a promising AMG
Example
Driven Cavity Velocity-Vorticity with Multigrid

ex50 -da_vec_type seq_cusp
    -da_mat_type aijcusp -mat_no_inode
    -da_grid_x 100 -da_grid_y 100
    -pc_type none -pc_mg_levels 1
    -preload off -cuda_synchronize
    -log_summary

    # Setup types
    # Set grid size
    # Setup solver
    # Setup run
# Example

## PFLOTRAN

### Flow Solver

**32 × 32 × 32 grid**

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time (s)</th>
<th>MFlops</th>
<th>MFlops/s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSPSolve</td>
<td>8.3167</td>
<td>4370</td>
<td>526</td>
</tr>
<tr>
<td>MatMult</td>
<td>1.5031</td>
<td>769</td>
<td>512</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KSPSolve</td>
<td>1.6382</td>
<td>4500</td>
<td>2745</td>
</tr>
<tr>
<td>MatMult</td>
<td>0.3554</td>
<td>830</td>
<td>2337</td>
</tr>
</tbody>
</table>

P. Lichtner, G. Hammond, R. Mills, B. Phillip
Serial Performance
NVIDIA GeForce 9400M

Performance on SNES Example 19

- GPU
- CPU

Time (s) vs. Number of Dof

M. Knepley (UC)  GPU  2011  8/1
Serial Performance
NVIDIA Tesla M2050

Performance on SNES Example 19

- GPU VecMDot
- CPU VecMDot
- GPU VecMAXPY
- CPU VecMAXPY
- GPU MatMult
- CPU MatMult

Time (s) vs. Number of Dof
Trilinos / PETSc Interoperability

- **Epetra_PETScAIJMatrix class**
  - Derives from Epetra_RowMatrix
  - Wrapper for serial/parallel PETSc aij matrices
  - Utilizes callbacks for matrix-vector product, getrow
  - No deep copies

- Enables PETSc application to construct and call virtually any Trilinos preconditioner

- ML accepts fully constructed PETSc KSP solvers as smoothers
  - Fine grid only
  - Assumes fine grid matrix is really PETSc aij matrix

- Complements Epetra_PETScAIJMatrix class
  - For any smoother with getrow kernel, PETSc implementation should be *much* faster than Trilinos
  - For any smoother with matrix-vector product kernel, PETSc and Trilinos implementations should be comparable
Trilinos Availability / Information

- Trilinos and related packages are available via LGPL or BSD.
- Current release (10.8) is “click release”. Unlimited availability.

- Trilinos Awards:
  - 2004 R&D 100 Award.
  - SC2004 HPC Software Challenge Award.
  - Sandia Team Employee Recognition Award.
  - Lockheed-Martin Nova Award Nominee.

- More information:

- Annual Forums:
  - DOE ACTS Tutorial (3rd week in August).
  - Annual Trilinos User Group Meeting in November @ SNL
    - talks and video available for download
Useful Links

**Trilinos website:**  http://trilinos.sandia.gov

**Trilinos tutorial:**  http://trilinos.sandia.gov/Trilinos10.6Tutorial.pdf

**Trilinos mailing lists:**  http://trilinos.sandia.gov/mail_lists.html

**Trilinos User Group (TUG) meetings:**
http://trilinos.sandia.gov/events/trilinos_user_group_2009
http://trilinos.sandia.gov/events/trilinos_user_group_2008
Trilinos Hands-On Tutorial

Teuchos Package

• For many Trilinos packages, this is the only required or “depends on” package.

• Provides basic utilities:
  • Parameter List
  • Memory management/Smart Pointer classes
  • Command Line Parser
  • Templated BLAS/LAPACK interfaces
  • XML Parser
  • MPI Communicator
Parameter List

- A key/value pair database that is recursive
  - Uses an implementation of the boost:\:Any object
  - Can read or output to XML files (internal xml or link to external xml)
  - Recursive: Sublists – nesting of parameter lists within itself
- Primary means of setting parameters in Trilinos packages:

```cpp
Teuchos::ParameterList p;

p.set("Solver", "GMRES");
p.set("Tolerance", 1.0e-4);
p.set("Max Iterations", 100);

Teuchos::ParameterList& lsParams = p.sublist("Solver Options");
lsParams.set("Fill Factor", 1);

double tol = p.get<double>("Tolerance");
int max_iters = p.get<int>("Max Iterations");
int fill = p.sublist("Solver Options").get<int>("Fill Factor");
```
Reference Counted Smart Pointer

- Powerful memory management for Trilinos packages!
- A wrapper for a pointer so that you don’t have to explicitly deallocate the memory.
  - When last RCP to the object is deleted, the underlying memory is deallocated.
- Next C++ standard will have Boost Smart Pointers

```cpp
class A {
};

int main {
    A* a = new A;
    .
    .
    .
    delete a;
}
```

```cpp
class A {
};

int main {
    A* a = new A;
    .
    .
    .
    using namespace Teuchos;
    RCP<A> a = rcp(new A);
    RCP<A> b = a;
}
```
Teuchos::RCP Beginner’s Guide

An Introduction to the Trilinos Smart Reference-Counted Pointer Class for (Almost) Automatic Dynamic Memory Management in C++

http://trilinos.sandia.gov/documentation.html
Trilinos/doc/RCPbeginnersGuide
Time Monitor

- Timers that keep track of:
  - Runtime
  - Number of calls
- Time object associates a string name to the timer.
  ```
  RCP<Time> fill_timer = TimeMonitor::getNewTimer("Fill Time");
  ```
- When TimeMonitor is created, the timer starts:
  ```
  TimeMonitor tm(Time& t);
  ```
- When TimeMonitor is destroyed (usually when you leave scope), the timer stops.
Petra Implementations

- **Epetra** (Essential Petra):
  - Classic production version.
  - Restricted to real, double precision arithmetic.
  - Uses stable core subset of C++ (circa 2000).
  - Interfaces accessible to C and Fortran users.
  - Classical approach to node-level parallel support.

- **Tpetra** (Templated Petra):
  - Emerging production C++ version.
  - Templated scalar and ordinal fields.
  - Uses namespaces, and STL: Improved usability/efficiency.
  - Advanced multi-precision support.
  - Advanced node architecture.
Epetra Package

Linear Algebra Package

http://trilinos.sandia.gov/packages/epetra/
Typical Flow of Epetra Object Construction

Construct Comm
- Any number of Comm objects can exist.
- Comms can be nested (e.g., serial within MPI).

Construct Map
- Maps describe parallel layout.
- Maps typically associated with more than one 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.
A Simple Epetra/Belos Program

http://code.google.com/p/trilinos/wiki/SimpleEpetraBelos
Typical Flow of Tpetra Object Construction

- Construct Platform
  - Options are: Pthreads, OpenMP, Thrust (CUDA), TBB, Serial

- Construct Node
  - Generalization of Epetra “Comm”
  - Composed with Kokkos::Node

- Construct Map

- Construct x
- Construct b
- Construct A
A Simple Tpetra/Belos Program

http://code.google.com/p/trilinos/wiki/SimpleTpetraBelos
Observations and Strategies for Next Generation Parallel Applications
Three Parallel Computing Design Points

• Terascale Laptop: Uninode-Manycore
• Petascale Deskside: Multinode-Manycore
• Exascale Center: Manynode-Manycore

Goal: Make Petascale = Terascale + more
Exascale = Petascale + more

Common Element

New functional.
Bonded systems.
552 lines C code.

How much MPI-specific code?
dft_fill_wjdc.c
MPI-specific code
SPMD Patterns for Domain Decomposition

• Halo Exchange:
  – Conceptual.
  – Needed for any partitioning, halo layers.
  – MPI is simply portability layer.
  – Could be replaced by PGAS, one-sided, …

• Collectives:
  – Dot products, norms.

• All other programming:
  – Sequential!!!
Reasons for MPI/SPMD Success?

- Portability? Yes.
- Standardized? Yes.
- Momentum? Yes.
- Separation of many Parallel & Algorithms concerns? Big Yes.

Once framework in place:
- Sophisticated physics added as serial code.
- Ratio of science experts vs. parallel experts: 10:1.

Key goal for new parallel apps: Preserve this ratio
MFIX Source term for pressure correction

- MPI-callable, OpenMP-enabled.
- 340 Fortran lines.
- No MPI-specific code.
- Ubiquitous OpenMP markup (red regions).

MFIX: Multiphase Flows with Interphase eXchanges (https://www.mfix.org/)
Evolving Parallel Programming Model
Parallel Programming Model: Multi-level/Multi-device

- **Message Passing**
  - Inter-node/inter-device (distributed) parallelism and resource management
  - Node-local control flow (serial)
  - Intra-node (manycore) parallelism and resource management
  - Stateless vectorizable computational kernels run on each core

- **Networking**
  - Network of computational nodes

- **Resources**
  - Computational node with manycore CPUs and/or GPGPU

- **Threading**
  - Thrashing
• Goal: Don’t repeat yourself (DRY).
• Every parallel programming environment supports basic patterns: parallel_for, parallel_reduce.
  – OpenMP:
    #pragma omp parallel for
    for (i=0; i<n; ++i) {y[i] += alpha*x[i];}
  – Intel TBB:
    parallel_for(blocked_range<int>(0, n, 100), loopRangeFn(…));
  – CUDA:
    loopBodyFn<< nBlocks, blockSize >>> (…);
• How can we write code once for all these (and future) environments?
Kokkos Compute Model

- How to make shared-memory programming generic:
  - **Parallel reduction** is the intersection of `dot()` and `norm1()`
  - **Parallel for loop** is the intersection of `axpy()` and `mat-vec`
  - We need a way of fusing kernels with these basic constructs.
- Template meta-programming is the answer.
  - This is the same approach that Intel TBB and Thrust take.
  - Has the effect of requiring that Tpetra objects be templated on Node type.
- Node provides generic parallel constructs, user fills in the rest:

```
template <class WDP>
void Node::parallel_for(
  int beg, int end, WDP workdata);

Work-data pair (WDP) struct provides:
- loop body via WDP::execute(i)

template <class WDP>
WDP::ReductionType Node::parallel_reduce(
  int beg, int end, WDP workdata);

Work-data pair (WDP) struct provides:
- reduction type WDP::ReductionType
- element generation via WDP::generate(i)
- reduction via WDP::reduce(x, y)
```
Example Kernels: `axpy()` and `dot()`

```cpp
template <class WDP>
void Node::parallel_for(int beg, int end,
                         WDP workdata);

template <class WDP>
WDP::ReductionType Node::parallel_reduce(int beg, int end,
                                         WDP workdata);

template <class T>
struct AxpyOp {
  const T * x;
  T * y;
  T alpha, beta;
  void execute(int i)
  { y[i] = alpha*x[i] + beta*y[i]; }
};

AxpyOp<double> op;
op.x = ...;  op.alpha = ...;
op.y = ...;  op.beta = ...;
node.parallel_for<AxpyOp<double>> (0, length, op);

template <class T>
struct DotOp {
  typedef T ReductionType;
  const T * x, * y;
  T identity() { return (T)0; }
  T generate(int i) { return x[i]*y[i]; }
  T reduce(T x, T y) { return x + y; }
};

DotOp<float> op;
op.x = ...;  op.y = ...;
float dot;
dot = node.parallel_reduce<DotOp<float>> (0, length, op);
```
Compile-time Polymorphism

Kokkos functor (e.g., AxpyOp)

+SerialNode

Serial Kernel

+TpiNode

pthread Kernel

+ThrustNode

Thrust Kernel

+FutureNode

Future Kernel
What’s the Big Deal about Vector-Vector Operations?

Examples from OOQP (Gertz, Wright)

\[ y_i \leftarrow y_i + \alpha x_i z_i \quad , i = 1 \ldots n \]
\[ y_i \leftarrow \begin{cases} 
  y_{\min} - y_i & \text{if } y_i < y_{\min} \\
  y^\max - y_i & \text{if } y_i > y^\max \\
  0 & \text{if } y_{\min} \leq y_i \leq y^\max 
\end{cases} , i = 1 \ldots n \]
\[ \alpha \leftarrow \{ \max \alpha : x + \alpha d \geq \beta \} \]

Examples from TRICE (Dennis, Heinkenschloss, Vicente)

\[ d_i \leftarrow \begin{cases} 
  (b - u)_i^{1/2} & \text{if } w_i < 0 \text{ and } b_i < +\infty \\
  1 & \text{if } w_i < 0 \text{ and } b_i = +\infty \\
  (u - a)_i^{1/2} & \text{if } w_i \geq 0 \text{ and } a_i > -\infty \\
  1 & \text{if } w_i \geq 0 \text{ and } a_i = -\infty 
\end{cases} , i = 1 \ldots n \]

Many different and unusual vector operations are needed by interior point methods for optimization!

Example from IPOPT (Waechter)

\[ x_i \leftarrow \begin{cases} 
  \left( x_i^L + \frac{\left( x_i^U - x_i^L \right)}{2} \right) & \text{if } \bar{x}_i^L \geq \bar{x}_i^U \\
  \bar{x}_i^L & \text{if } x_i < \bar{x}_i^L \\
  \bar{x}_i^U & \text{if } x_i > \bar{x}_i^U 
\end{cases} , i = 1 \ldots n \]

Currently in MOOCHO:

\[ > 40 \text{ vector operations!} \]

where:

\[ \bar{x}_i^L = \min \left( x_i^L + \eta \left( x_i^U - x_i^L \right), x_i^L + \delta \right) \]
\[ \bar{x}_i^U = \max \left( x_i^L - \eta \left( x_i^U - x_i^L \right), x_i^U - \delta \right) \]
Defining your own kernels
Tpetra RTI Components

• Set of stand-alone non-member methods:
  - `unary_transform<UOP>(Vector &v, UOP op)`
  - `binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)`
  - `reduce<G>(const Vector &v1, const Vector &v2, G op_glob)`
  - `binary_pre_transform_reduce<G>( Vector &v1, const Vector &v2, G op_glob)`

• These are non-member methods of Tpetra::RTI which are loosely coupled with Tpetra::MultiVector and Tpetra::Vector.

• Tpetra::RTI also provides Operator-wrappers:
  - `class KernelOp<..., Kernel > : Tpetra::Operator<...>`
  - `class BinaryOp<...,BinaryOp> : Tpetra::Operator<...>`
// isn't this nicer than a bunch of typedefs?
auto &platform = Tpetra::DefaultPlatform::getDefaultPlatform();
auto comm = platform.getComm();
auto node = platform.getNode();

// create Map and some Vector objects
Tpetra::global_size_t numGlobalRows = ...;
auto map = createUniformContigMapWithNode<int, int>(numGlobalRows, comm, node);
const size_t numLocalRows = map->getNodeNumElements();
auto x = Tpetra::createVector<float>(map),
    y = Tpetra::createVector<float>(map),
auto z = Tpetra::createVector<double>(map),
    w = Tpetra::createVector<double>(map);

// parallel initialization of x[i] = 1.0 using C++-0x lambda function
Tpetra::RTI::unary_transform(  *x, [](float xi){return 1.0f;} );
// parallel initialization of y[i] = x[i]
Tpetra::RTI::binary_transform(  *y, *x, [](float, float xi) {return xi;} );
// parallel y[i] = x[i] + y[i]
Tpetra::RTI::binary_transform(  *y, *x, std::plus<float>() );
// parallel single precision dot(x,y)
fresult = Tpetra::RTI::reduce(  *x, *y, reductionGlob<ZeroOp<float>>(
    std::multiplies<float>(),
    std::plus<float>() ));
Tool: Tpetra Reduce/Transform

- Set of stand-alone non-member methods:
  - unary_transform<UOP>(Vector &v, UOP op)
  - binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)
  - reduce>G>(const Vector &v1, const Vector &v2, G op_glob)
- This levels provides maximal expressiveness, but convenience wrappers are available as well.

```cpp
// single dot() with double accumulator using custom kernels
result = Tpetra::RTI::reduce( *x, *y, myDotProductKernel<float,double>() );
// ... or an composite adaptor and well known functors
result = Tpetra::RTI::reduce( *x, *y,
    reductionGlob<ZeroOp<double>>(
        std::multiplies<float>(),
        std::plus<double>()) );
// ... or using inline functors via C++ lambdas
result = Tpetra::RTI::reduce( *x, *y,
    reductionGlob<ZeroOp<double>>(
        [](float x, float y) {return x*y;},
        [](double a, double b){return a+b;} );
// ... or using a convenience macro
result = TPETRA_REDUCE2( x, y, x*y, ZeroOp<float>, std::plus<double>() );
```
Future Node API Trends

• TBB provides very rich pattern-based API.
  – It, or something very much like it, will provide environment for sophisticated parallel patterns.

• Simple patterns: FutureNode may simply be OpenMP.
  – OpenMP handles parallel_for, parallel_reduce fairly well.
  – Deficiencies being addressed.
  – Some evidence it can beat CUDA.

• OpenCL practically unusable?
  – Functionally portable.
  – Performance not.
  – Breaks the DRY principle.
Additional Benefits of Templates
Multiprecision possibilities

- Tpetra is a templated version of the Petra distributed linear algebra model in Trilinos.
  - Objects are templated on the underlying data types:
    ```
    MultiVector<scalar=double, local_ordinal=int, global_ordinal=local_ordinal> ... 
    CrsMatrix<scalar=double, local_ordinal=int, global_ordinal=local_ordinal> ... 
    ```
  - Examples:
    ```
    MultiVector<double, int, long int> V; 
    CrsMatrix<float> A; 
    ```

  Speedup of float over double in Belos linear solver.

    | Scalar | float | double | double-double | quad-double |
    |--------|-------|--------|---------------|-------------|
    |        | 18 s  | 26 s   | 1.42x         |

    | Scalar | Solve time (s) | Accuracy |
    |--------|---------------|----------|
    |        | 2.6           | 10^-6    |
    |        | 5.3           | 10^-12   |
    |        | 29.9          | 10^-24   |
    |        | 76.5          | 10^-48   |

Arbitrary precision solves using Tpetra and Belos linear solver package.
class FloatShadowDouble {

public:
    FloatShadowDouble( ) {
        f = 0.0f;
        d = 0.0;  }
    FloatShadowDouble( const FloatShadowDouble & fd) {
        f = fd.f;
        d = fd.d;  }

    ...}

    inline FloatShadowDouble operator+=(const FloatShadowDouble & fd) {
        f += fd.f;
        d += fd.d;
        return *this;  }

    ...}

    inline std::ostream& operator<<(std::ostream& os, const FloatShadowDouble& fd) {
        os << fd.f << "f " << fd.d << ", double."; return os;}

    ...}

• Templates enable new analysis capabilities
• Example: Float with “shadow” double.
FloatShadowDouble

Sample usage:
#include “FloatShadowDouble.hpp”
Tpetra::Vector<FloatShadowDouble> x, y;
Tpetra::CrsMatrix<FloatShadowDouble> A;
A.apply(x, y); // Single precision, but double results also computed, available

Initial Residual = 455.194f 455.194d
Iteration = 15 Residual = 5.07328f 5.07618d
Iteration = 30 Residual = 0.00147022f 0.00138466d
Iteration = 45 Residual = 5.14891e-06f 2.09624e-06d
Iteration = 60 Residual = 4.03386e-09f 7.91927e-10d

http://locklessinc.com/articles/interval_arithmetic: Interval arithmetic package
Example: Recursive Multi-Prec CG

```cpp
for (k=0; k<numIters; ++k) {
    T pAp = TPETRA_REDUCE2( p, Ap,
                             p*Ap, ZeroOp<T>, plus<T>() ); // p'*Ap
    const T alpha = zr / pAp;
    TPETRA_BINARY_TRANSFORM( x, p,
                             x + alpha*p ); // x = x + alpha*p
    TPETRA_BINARY_TRANSFORM( rold, r,
                             r ); // rold = r
    T rr = TPETRA_BINARY_PRETRANSFORM_REDUCE( r, Ap,
                                             r - alpha*Ap, // : r - alpha*Ap
                                             r*r, ZeroOp<T>, plus<T>() ); // : sum r'*r
    // recursive call to precondition this iteration
    recursiveFPCG<TS::next,LO,GO,Node>(out,db_T2); // x_T2 = A_T2 \ b_T2
    auto plusTT = make_pair_op<T,T>(plus<T>());
    pair<T,T> both = TPETRA_REDUCE3( z, r, rold,
                                     make_pair(z*r, z*rold),
                                     ZeroPTT, plusTT );
    const T beta = (both.first - both.second) / zr;
    zr = both.first;
    TPETRA_BINARY_TRANSFORM( p, z, z + beta*p ); // p = z + beta*p
}
Example: Recursive Multi-Prec CG

TBBNode initializing with numThreads == 2
TBBNode initializing with numThreads == 2

Running test with Node==Kokkos::TBBNode on rank 0/2

Beginning recursiveFPCG<qd_real>
Beginning recursiveFPCG<dd_real>
|res|/|res_0|: 1.269903e-14
|res|/|res_0|: 3.196573e-24
|res|/|res_0|: 6.208795e-35
Convergence detected!
Leaving recursiveFPCG<dd_real> after 2 iterations.
|res|/|res_0|: 2.704682e-32
Beginning recursiveFPCG<dd_real>
|res|/|res_0|: 4.531185e-09
|res|/|res_0|: 6.341084e-20
|res|/|res_0|: 8.326745e-31
Convergence detected!
Leaving recursiveFPCG<dd_real> after 2 iterations.
Leaving recursiveFPCG<qd_real> after 2 iterations.
Example: Recursive Multi-Prec CG

- Problem: Oberwolfach/gyro
- N=17K, nnz=1M
- qd_real/dd_real/double
- MPI + TBB parallel node
- #threads = #mpi x #tbb
- Solved to over 60 digits
- Around 99.9% of time spent in double precision computation.
- Single codebase.
Additional Benefits of Templates
Current Scientific Library Paradigm

• Library provides a specific capability.
  – A user can grab the data and expand the functionality.

• In an MPI-only scenario, expansion comes via domain-specific serial kernels coded by domain specialists.
  – By “serial”, I mean that they are not writing raw MPI or doing any shared-memory programming (OpenMP, native threads)

• With a single memory pool, data is commonly shared between library and app.
  – The main difficulty here regards data ownership.

• With a single target architecture, compilation is relatively simple.
  – Compile with MPI compiler wrappers or link against MPI library.
Enter the Hybrid Parallel Environment

• Modern supercomputers
  – LANL RoadRunner has Cell BE in addition to multi-core CPUs.
  – Tianhe 1A uses NVIDIA GPUs
  – K Computer simply consists of nodes of 8-core CPUs
  – OLCF Titan will utilize NVIDIA GPUs and 12-way multi-core CPUs
  – The path to exascale apparently requires addressing multi-core.

• Ditch the assumptions of the previous slide/paradigm:
  1. shared-memory programming augments MPI-only
  2. accelerators with their own memory space complicate data passing; even data layout for NUMA archs. requires finesse
  3. heterogeneous execution environment requires compiling our kernels multiple times, for each hardware type
  4. tuning library is more important and intrusive than ever
A Generic Shared Memory API

• One solution is an API or language for programming to a generic shared memory node.

• Should provide two main components:
  – **Generic memory model** addressing data issues
    • Allocation, deallocation and efficient access of memory
    • Use of automatic memory management classes
  – **Generic compute model** addressing work issues
    • Description of kernels for parallel execution on a node
    • Skeletons for efficient execution on multiple architectures
    • User-provided kernels provide the body of these skeletons
Current Approaches to These Problems in Stage 2 Trilinos

• Templated C++ code
  – Templating data allows more efficient use of cache and bandwidth.
  – Templating data expands capability (e.g., integer limit, complex)

• Generic shared memory parallel node
  – Kokkos provides shared memory parallel node API
  – Many abstractions exist for a generic shared-memory parallel node.
  – These use a mix of different approaches (compile-time versus run-time polymorphism)

• Plug-and-play structures and algorithms
  – Expose the SMP node to apps; enable node-optimized kernels.
Tpetra and Kokkos Packages

• **Tpetra** is a distributed linear algebra library.
  – Similar to Trilinos/Epetra:
    • Provides maps, vectors, sparse matrices and abstract linear operators
  – Heavily exploits templated C++
  – Employs hybrid (distributed + shared) parallelism via Kokkos

• **Kokkos** is an API for shared-memory parallel nodes.
  – Provides parallel_for and parallel_reduce skeletons
  – Provides local, shared-memory parallel linear algebra
  – Currently supports multiple shared-memory APIs:
    • ThreadPool Interface (TPI, a Trilinos pthreads package)
    • Intel Threading Building Blocks (TBB)
    • NVIDIA CUDA-capable GPUs (via Thrust)
    • OpenMP (*implemented by Radu Popescu/EPFL, not yet released*)
Tpetra Hybrid Parallelism

• The typical Tpetra computational kernel concerns:
  1) member data structures
  2) calls to Kokkos NodeAPI for shared-memory programming
  3) calls to Tpetra::Comm for distributed programming

<table>
<thead>
<tr>
<th>(1) internal class data</th>
<th>e.g., Tpetra::Vector::norm1()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar *x; int N;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(2) call the Kokkos NodeAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>DotOp&lt;Scalar&gt; op(x);</td>
</tr>
<tr>
<td>lcl = node.parallel_for( 0, N, op );</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(3) call the Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>gbl = comm.reduceAll( lcl, SUM );</td>
</tr>
</tbody>
</table>

• Extending library functionality can be done via external input at these three junctions.
Tool: Tpetra Reduce/Transform

• Set of stand-alone non-member methods:
  - unary_transform<UOP>(Vector &v, UOP op)
  - binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)
  - reduce<G>(const Vector &v1, const Vector &v2, G op_glob)

• This levels provides maximal expressiveness, but convenience wrappers are available as well.

```cpp
// single dot() with double accumulator using custom kernels
result = Tpetra::RTI::reduce( *x, *y, myDotProductKernel<float,double>() );
// ... or an composite adaptor and well known functors
result = Tpetra::RTI::reduce( *x, *y,
    reductionGlob<ZeroOp<double>>(
        std::multiplies<float>(),
        std::plus<double>() ) );

// ... or using inline functors via C++ lambdas
result = Tpetra::RTI::reduce( *x, *y,
    reductionGlob<ZeroOp<double>>(
        [](float x, float y) {return x*y;} ,
        [](double a, double b){return a+b;} ) );

// ... or using a convenience macro
result = TPETRA_REDUCE2( x, y, x*y, ZeroOp<float>, std::plus<double>() );
```
How to Program Hybrid Clusters

• Programming for a single GPU is well studied.
  – So what about more than one GPU?

• Distributed memory ➞ distributed memory model

• **One MPI process per shared-memory pool.**
  – This currently comes at the cost of at least one CPU core per physical node.
  – Have to be even more careful with communication than before.
  – Even then, you may have to pay twice (PCIe + node interconnect).
  – Other MPI processes marshal other shared-memory nodes, such as multi-core CPUs.
Tool: Tpetra HybridPlatform

- **Encapsulate main in a templated class method:**

  ```c++
  template <class Node>
  class myMainRoutine {
      static void run(ParameterList &runParams,
           const RCP<const Comm<int> > &comm,
           const RCP<Node> &node)
      {
        // do something interesting
      }
  };
  ```

- **HybridPlatform** maps the communicator rank to the Node type, instantiates a node and the user routine:

  ```c++
  int main(...) {
    Comm<int> comm = ...;
    ParameterList machine_file = ...;
    // instantiate appropriate node and myMainRoutine
    Tpetra::HybridPlatform platform( comm , machine_file );
    platform.runUserCode< myMainRoutine >();
    return 0;
  }
  ```
<ParameterList>
  <ParameterList name="%2=0">
    <Parameter name="NodeType" type="string" value="Kokkos::ThrustGPUNode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Device Number" type="int" value="0"/>
    <Parameter name="Node Weight" type="int" value="4"/>
  </ParameterList>
  <ParameterList name="%2=1">
    <Parameter name="NodeType" type="string" value="Kokkos::TPINode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Num Threads" type="int" value="15"/>
    <Parameter name="Node Weight" type="int" value="15"/>
  </ParameterList>
</ParameterList>
Example: CG with simple operator

for (k=0; k<numIters; ++k) {
    S pAp = TPETRA_REDUCE2(
        p, Ap,
    const S alpha = rr / pAp; // alpha = r'*r/p'*Ap
    TPETRA_BINARY_TRANSFORM(
        x, p,
        x + alpha*p // x = x + alpha*p
    );
    S rrold = rr;
    rr = TPETRA_BINARY_PRETRANSFORM_REDUCE(
        r, Ap,
        r - alpha*Ap, // r - alpha*Ap
        r*r, ZeroOp<S>, plus<S>()); // sum r'*r
    const S beta = rr / rrold; // beta = r'*r/old(r'*r)
    TPETRA_BINARY_TRANSFORM(
        p, r,
        r + beta*p // p = z + beta*p
    );
}
Example: CG with simple operator

- Problem dimension 5M
- 500 iterations
- **Double precision** arithmetic
- MPI + TBB parallel node
- \#threads = \#mpi x \#tbb

**invocation like:**

```
mpirun -np 4 ./driver.exe --machine-file=tbb4.xml
```
Example: CG with simple operator

- Problem dimension 512K
- 125 iterations
- Quad-double precision
- MPI + TBB parallel node
- \#threads = \#mpi x \#tbb
- *Same codebase*, simply instantiated on qd_real instead of double.
Next Generation Preconditioners
Two Threaded Smoother Approaches

- Bordered block diagonal form:
  - Compute the subdomains via hypergraph partitioner (Zoltan)
  - Compute doubly bordered block diagonal form as part of factorization.
  - P1 … P4 correspond to multiple sockets in a multicore node.

- Multithreaded direct factorization KLU2:
  - Templated version of popular KLU solver.
  - New multithreaded direct factorization.
  - T1…T2: multiple threads within same socket.
  - Incomplete version as preconditioner.

KLU2: Multithreaded Direct Factorization

```
1   X
2   X   X
3   X
X   4   X
X   X   X
X   5   X
X   6   X
X   7   X
X   8   X
X   X   X   9
```

Decomposition by Partitioning

![Diagram showing decomposition by partitioning with subdomains P1, P2, P3, and P4 arranged in a block diagonal form.]

![Diagram illustrating the multithreaded factorization with nodes 1 through 9 and threads T1, T2, and T1/T2 indicated.]
ShyLU Results

- **ShyLU provides:**
  - Hybrid MPI+threads
  - Exact and inexact solves.

- **Results:**
  - Hopper: Cray (24-core node).
    - 2-socket, dual hexacore per socket.
    - 1,2,3,4 nodes.
  - MPI ranks vs. num threads (r x t).
  - Show thread value at large core counts.

![Graph showing performance over number of cores for different configurations (4x6, 6x4, 12x2, 24x1).](Image)
MueLu

- Future package of the Trilinos project (to replace ML)
  - C++ - Object-oriented design
  - Massively parallel
  - Multicore and GPU aware
  - Templated types for mixed precision and complex arithmetic

- Objective is to solve problem with billions of DOF on 100Ks of cores...

- Leverage the Trilinos software stack:

  ![Diagram of software stack](image)

- Currently in development...