Parallel Public-domain Numerical Libraries

CSCS Tutorial Series, 11.05.2010
Dr. William Sawyer
Overview

- Introduction and library overview
  - Typical = numerical and semi-numerical problems
  - Survey of parallel libraries
- Case study 1: dense linear algebra
- Case study 2: sparse eigenvalue problems
- Case study 3: non-linear optimization
Poll

- What programming languages do you work with?
- What parallel programming paradigms?
- What problems do you want to solve?
- What algorithms do you want to use?
- What libraries, if any, do you use?
Assumptions and prerequisites

- Basic background in numerical analysis
- C/C++ and Fortran programming experience
- Some parallel programming experience
- Basic UNIX user background
- Account of Cray XT5 gele (for exercises)
Problems you might like to solve

- Partial differential equations (PETSc)
- Dense systems of linear equations
- Sparse systems of linear equations (PETSc)
- Preconditioning of large systems (PETSc)
- Eigenvalue / singular value decompositions of sparse/dense matrices
- Partitioning large graphs
- Non-linear systems and optimization
- ...

Objectives of this tutorial

 Awareness of the available libraries
 Realization that it is not necessary to “recreate the wheel”
 Bonus: some hands-on experiences
Typical linear algebra operations

- **Cholesky factorization:**
  \[ A = A^T = LL^T \quad i < j \Rightarrow L_{i,j} = 0 \]

- **QR factorization:**
  \[ A = QR \quad Q^T Q = I \quad i > j \Rightarrow R_{i,j} = 0 \]

- **LU factorization:**
  \[ A = P^T LU \quad P^T P = I \]

- **Forward/back-substitution:**
  \[ Ax = y \Rightarrow LUX = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w \]

- **Eigenvalue decomposition:**
  \[ Ax = \lambda x \Rightarrow A = QDQ^T \]

- **Generalized eigen-problem:**
  \[ Ax = \lambda Bx \]

- **Singular value decomposition:**
  \[ A = U\Sigma V^T \quad U^T U = I \quad V^T V = I \]

- **Preconditioning:**
  \[ Ax = b \Rightarrow M^{-1}Ax = M^{-1}b \quad M \approx A \]
Parallel libraries for dense linear algebra

- **ScalAPACK**: Fortran Scalable Linear Algebra PACKage (LAPACK)  
  [http://www.netlib.org/scalapack](http://www.netlib.org/scalapack)

- **PLAPACK**: object-oriented Parallel Linear Algebra PACKage  
  [http://userweb.cs.utexas.edu/users/plapack](http://userweb.cs.utexas.edu/users/plapack)

- **PLASMA**: Parallel Linear Algebra for Scalable Multicore Architectures  
  [http://icl.cs.utk.edu/plasma](http://icl.cs.utk.edu/plasma)

- **MAGMA**: Matrix Algebra on GPU and Multicore Architectures  
  [http://icl.cs.utk.edu/magma](http://icl.cs.utk.edu/magma)
PLAPACK vs. ScaLAPACK

(a) Cholesky factorization
(b) LU factorization
(c) QR factorization
(d) LU factorization on different configurations

5/11/10 From: PLAPACK User’s Manual
PLASMA Performance

Figure 1: Performance comparison on a large number of cores (Gflop/s).

MAGMA Performance

MAGMA on GTX280 vs. Xeon quad core Left: QR decomp. SP/DP  Right: LU decomp. SP/DP

5/11/10

From: HIPS 2010, Tomov et al.
Trilinos: a ‘pearl necklace’ of packages

Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems

Trilinos: problems covered

- Constructing and using sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- PDE-constrained optimization problem
- Partitioning and load balancing of distributed data structures
- Automatic differentiation
- PDE discretizations
- More …
Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt* (C++), *Tpetra* (C++ templates)
- Preconditioners: *AztecOO*, *Ifpack/Tifpack*, *ML*, *Meros*
- Iterative linear solvers: *AztecOO*, *Belos*
- Direct linear solvers: *Amesos* (*SuperLU*, *UMFPACK*, *MUMPS*, *ScaLAPACK*, …)
- Non-linear / optimization solvers: *NOX*, *MOOCHO*
- Eigensolvers: *Anasazi*
- Mesh generation / adaptivity: *Mesquite*, *PAMGEN*
- Domain decomposition: *Claps*
- Partitioning / load balance: *Isorropia*, *Zoltan*
# Trilinos solver overview

<table>
<thead>
<tr>
<th>Optimization</th>
<th>MOOCHO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained:</td>
<td></td>
</tr>
<tr>
<td>Constrained:</td>
<td></td>
</tr>
<tr>
<td>Find $u \in \mathbb{R}^n$ that minimizes $g(u)$</td>
<td></td>
</tr>
<tr>
<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>Bifurcation Analysis</td>
<td>LOCA</td>
</tr>
<tr>
<td>Given nonlinear operator $F(x,u) \in \mathbb{R}^{n+m}$</td>
<td></td>
</tr>
<tr>
<td>For $F(x,u) = 0$ find space $u \in U \ni \partial F / \partial x$</td>
<td></td>
</tr>
<tr>
<td>Transient Problems</td>
<td>Rythmos</td>
</tr>
<tr>
<td>DAEs/ODEs:</td>
<td></td>
</tr>
<tr>
<td>Solve $f(\dot{x}(t), x(t), t) = 0$</td>
<td></td>
</tr>
<tr>
<td>$t \in [0, T]$; $x(0) = x_0$, $\dot{x}(0) = v_0$</td>
<td></td>
</tr>
<tr>
<td>for $x(t) \in \mathbb{R}^n$, $t \in [0, T]$</td>
<td></td>
</tr>
<tr>
<td>Nonlinear Problems</td>
<td>NOX</td>
</tr>
<tr>
<td>Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td>Solve $F(x) = 0$ $x \in \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td>Linear Problems</td>
<td></td>
</tr>
<tr>
<td>Linear Equations:</td>
<td>AztecOO, Belos, Ifpack, ML, etc... Anasazi</td>
</tr>
<tr>
<td>Eigen Problems:</td>
<td></td>
</tr>
<tr>
<td>Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$</td>
<td></td>
</tr>
<tr>
<td>Solve $Ax = b$ for $x \in \mathbb{R}^n$</td>
<td></td>
</tr>
<tr>
<td>Solve $Av = \lambda Bu$ for (all) $v \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$</td>
<td></td>
</tr>
<tr>
<td>Distributed Linear Algebra</td>
<td></td>
</tr>
<tr>
<td>Matrix/Graph Equations</td>
<td></td>
</tr>
<tr>
<td>Vector Problems:</td>
<td>Epetra, Tpetra</td>
</tr>
<tr>
<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>Compute $y = \alpha x + \beta w$; $\alpha = \langle x, y \rangle$; $x, y \in \mathbb{R}^n$</td>
<td></td>
</tr>
</tbody>
</table>

5/11/10
Sawyer / CSCS / 2010-05-11
Trilinos / PETSc Interoperability

- Epetra_PETScAIJMatrix class
- Allows PETSc to call Trilinos preconditioners
- ML (Multilevel precond.) can call PETSc solvers as smoothers

More about Trilinos later
Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$
Linear System Solution: \[ Ax = b \]

Two basic techniques:
- Direct methods, i.e. factorize matrix
  - good for multiple right hand sides \( AX = B \)
  - tend to be more robust
- Iterative methods
  - good if matrix known via operators \( Ax, A^T x \)
  - possibilities for approximate solutions
LU decomposition

- Formulations:  $A = LU \quad A = P^T LU \quad A = P^T L U Q^T$
- Permutation Matrices, $P$, $Q$:
- Fill-in

$
\begin{pmatrix}
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0
\end{pmatrix}
$
Parallel direct linear system solvers

- SuperLU_Dist (Super-nodal LU decomposition, C)
  - http://acts.nersc.gov/superlu/
- MUMPS (Multifrontal Massively Parallel, F90)
  - http://graal.ens-lyon.fr/MUMPS/
- PSPASES (Par. SPArse Symm. dirEct, C)
  - http://www-users.cs.umn.edu/~mjoshi/pspases/
- PaStiX (Parallel Sparse matriX package, C)
- PARDISO (SMP), SPOOLES, PSPIKE, others…
Performance comparison

- Each technique has strengths/weaknesses
  - Gould, Scott, RAL-TR-2005-005
- MUMPS and SuperLU possibly the two most used

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Ordering</th>
<th>Solver</th>
<th>Number of processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>bbbmat</td>
<td>AMD</td>
<td>MUMPS</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SuperLU</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ND(metis)</td>
<td>MUMPS</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SuperLU</td>
<td>-</td>
</tr>
<tr>
<td>ecl32</td>
<td>AMD</td>
<td>MUMPS</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SuperLU</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ND(metis)</td>
<td>MUMPS</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SuperLU</td>
<td>-</td>
</tr>
</tbody>
</table>
Direct solvers: PETSc interfaces

- MUMPS
- SuperLU_Dist
- PaStiX
- SPOOLES
- UMFPACK (serial)
Iterative linear system solvers

▪ Older methods:
  – Jacobi iteration
  – Gauss/Seidel
  – SOR/SSOR: Successive Over Relaxation
  – Method of steepest descent

▪ Newer methods:
  – Krylov subspace methods, e.g.,
  – Conjugate Gradients (CG)
Hestenes/Stiefel, 1952: Conjugate Gradient

\[ k = 0; \quad x_0 = 0; \quad r_0 = 0 \]

while \( r_k \neq 0 \) {

\[ k = k + 1 \]

if \( (k = 0) \) \( \Rightarrow \) \( p_1 = r_0 \)

if \( (k > 0) \) \( \Rightarrow \) \( \beta_k = r_k^T r_{k-1} / r_{k-2}^T r_{k-2} \); \( p_k = r_{k-1} + \beta_k p_{k-1} \)

\[ \alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k \]

\[ x_k = x_{k-1} + \alpha_k p_k \]

\[ r_k = r_{k-1} - \alpha_k A p_k \]

}
Some Krylov subspace methods

- Symmetric matrices (requires $Ax$)
  - CG (Conjugate Gradient): convergence ‘assured’

- Non-symmetric matrices (require $Ax, A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge

- Non-symmetric matrices (require $Ax$)
  - GMRES (Generalized Minimal Residuals): ‘converges’
  - BiCGstab (Bi-CG stabilized): may not converge
  - CGS (CG squared): may not converge
  - TFQMR (Transpose-free QMR): may not converge
Parallel libraries with KSM

- **PETSc**: all of the above and more
- **AztecOO**: Trilinos package, numerous methods
  - Object oriented, C++, requires Epetra / EpetraExt
- **BELOS**: Trilinos package, reaching maturity
  - Operators and vectors as opaque objects
- **pARMS** (parallel Algebraic Recursive Multilevel Solvers)
  - Currently only FGMRES supported
Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue

- Easier problem: preconditioner
  \[ Ax = b \Rightarrow M^{-1}Ax = M^{-1}b \quad M \approx A \]
  - Find an approximation for A with \( M^{-1}x \) ‘easily’ calculated
  - Possibilities:
    - Approximate inverse known through physical description
    - Incomplete LU decomposition
    - Sparse approximative inverse (assume inverse also sparse)
    - Multilevel (multigrid) preconditioners
    - More…
Example: Sparse Approximative Inverse

(a) Original matrix.  
(b) Approximate inverse.
Parallel preconditioner libraries

- **SPAI/MSPAI (Modified Sparse Approximative Inverse)**
  - Numerical technique is inherently parallel
  - [http://www5.in.tum.de/wiki/index.php/MSPAI](http://www5.in.tum.de/wiki/index.php/MSPAI)

- **Trilinos**
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)

- **Hypre (high-perf preconditioners)**
  - High level interface for grid-based problems
  - ILU (Euclid/pilut), multigrid (PFMG/BoomerAMG), sparse (parasails)
  - [https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html](https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html)
PETSc preconditioners

- **Internal**
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)

- **External**
  - Hypre
  - Prometheus ([http://www.columbia.edu/~ma2325/prom_intro.html](http://www.columbia.edu/~ma2325/prom_intro.html))
  - SPAI
  - ML
Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real: \( Ax = \lambda x \Rightarrow Q^T AQ = \text{diag}(\lambda_1, \ldots, \lambda_n) \)
  - Symmetric, real: \( Ax = \lambda x \Rightarrow X^{-1}AX = \text{diag}(\lambda_1, \ldots, \lambda_n) \)
  - Non-symmetric, non-defective: \( Ax = \lambda x \Rightarrow X^{-1}AX = \text{diag}(\lambda_1, \ldots, \lambda_n) \)
  - Generalized, symm: \( Ax = \lambda Bx \Rightarrow Q^T AQ = \text{diag}(a_1, \ldots, a_n) \quad Q^T BQ = \text{diag}(b_1, \ldots, b_n) \)
  - Singular values: \( U^T AV = \text{diag}(\sigma_1, \ldots, \sigma_n) \)

- Interpretation: eigenvectors of A when multiplied by A are parallel to themselves

- Applications in numerous fields
Eigen- and singular value solvers, dense matrices

- **ScaLAPACK**
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form
  - PSGEBRD /PDGEBRD: Reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition:
  - PSGEHRD /PDGEHRD: Reduction of a nonsymmetric matrix to Hessenberg form to solve a nonsymmetric eigenvalue problem

- **MAGMA / PLASMA**
  - Under development! Please let us know your needs
Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)
  
  \[ r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0 \]
  
  while \( \beta_j \neq 0 \) 
  
  \[ q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j \]
  
  \[ r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \| r_j \|_2 \]
  
  
- Lanczos vectors: \( q_j \)

- Form tridiagonal matrix T: diagonal \( \alpha_j \) subdiagonal \( \beta_j \)

- Diagonalization of T is stable iterative procedure
Nonsymmetric matrices: Arnoldi method, 1951

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in $q_j$, poor choice of $q_i$
- ARPACK (Arnoldi Package, 1992), F77
  - Implicitly restarted algorithm, generalized eigenvalue problem
  - Calculates a few eigenvalues and corresponding eigenvectors
  - Call-back mechanism to request $Ax$ operation, preconditioner
  - [http://www.caam.rice.edu/software/ARPACK/](http://www.caam.rice.edu/software/ARPACK/)
Other eigen-solver libraries, nonsymmetric matrices

- **IETL (Iterative Eigensolver Template Library):**
  - C++ with templates; four methods for hermitian matrices

- **PRIMME (PReconditioned Iterative MultiMethod Eigensolve):**
  - Parallel version for double precision but not complex double
  - [http://www.cs.wm.edu/~andreas/software/](http://www.cs.wm.edu/~andreas/software/)

- **SLEPc (Scalable Library for Eigenvalue Problem computations):**
  - (non-)hermitian matrices; generalized EVP; partial SVD
  - Extension to PETSc
  - [http://www.grycap.upv.es/slepc/](http://www.grycap.upv.es/slepc/)

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges

- Application areas:
  - Partition mesh on p processes
  - Fill reducing orderings for sparse matrix decompositions
  - Some optimization problems can be formulated as a graph
Graph partitioning facets

Dynamic Load Balancing

Graph Coloring

Matrix Ordering

Unstructured Communication

Distributed Data Directories

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Parallel graph (re-)partitioning libraries

- METIS / ParMETIS:
  - http://glaros.dtc.umn.edu/gkhome/metis

- Jostle / PJostle (Networks):
  - http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/

- Zoltan

- SCOTCH / PT-SCOTCH
  - http://www.labri.fr/perso/pelegrin/scotch/
Non-linear and other problems

- Non-linear ordinary differential equations
- Optimization problems
- Differential-algebraic equations
- Non-linear algebraic systems
OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

  \[
  \min_{x \in \mathbb{R}^n} \quad f(x)
  
  \text{subject to} \quad h_i(x) = 0, \quad g_i(x) \geq 0, \quad i = 1, \ldots, p, m.
  \]

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method
  - a parallel direct search method
  - a nonlinear interior point method.


- Parallel OPT++ only tested on Intel Linux, MPICH1.2.2.3
MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization
- Minimize $f(x_D, x_I)$ subject to $c(x_D, x_I) = 0$ \( x_D^{lower} \leq x_D \leq x_D^{upper} \quad x_I^{lower} \leq x_I \leq x_I^{upper} \)
- A priori partitioned into dependent state variables $x_D$ and independent optimization variables $x_I$
- Can utilize parallel direct solvers (Amesos), preconditioners (Ifpack,ML), Krylov methods (AztecOO, Belos)
NOX (Trilinos): non-linear ODE systems

- Solve \( F(x) = 0 \) with \( F(x) = \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{pmatrix} \) and \( J_{i,j} = \frac{\partial F_i}{\partial x_j}(x) \)
- User supplies:
  - Function \( F(x) \) evaluation
  - Optional: Jacobian evaluation, preconditioner
- With good guess, convergence quadratic
- Heuristics used to improve first guess
- PETSc interface available
- \texttt{http://trilinos.sandia.gov/packages/nox/}
SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE \( y' = f(t,y) \)
  - CVODES \( y' = f(t,y,p) \)
  - IDA \( F(t,y,y') = 0 \)
  - IDAS \( F(t,y,y',p) = 0 \)
  - KINSOL: nonlinear algebraic systems (Newton-Krylov)

- Sequential and (reduced) parallel functionality

- [https://computation.llnl.gov/casc/sundials/](https://computation.llnl.gov/casc/sundials/)
Case study 1: dense linear algebra

Objectives:

- Get acquainted with ScaLAPACK
- Example (from ScaLAPACK tutorial):
  - initialize process grid with BLACS
  - initialize array descriptors
  - read a matrix from file
  - perform LU decomposition
  - solve system
ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices
- Developing teams: UT Knoxville, UC Berkeley, ORNL, Rice U., UCLA, UIUC, etc.
- Supported in Commercial Packages
  - NAG Parallel Library, IBM PESSL, CRAY Scilib
  - VNI IMSL, Fujitsu, HP/Convex, Hitachi, NEC
ScaLAPACK Technical Information

- **ScaLAPACK web page**
  - [http://www.netlib.org/scalapack](http://www.netlib.org/scalapack)

- **ScaLAPACK User’s Guide**

- **Excellent introductory site**
  - [http://acts.nersc.gov/scalapack](http://acts.nersc.gov/scalapack)
  - [http://acts.nersc.gov/scalapack/hands-on](http://acts.nersc.gov/scalapack/hands-on)

- **LAPACK Working Notes**
  - [http://www.netlib.org/lapack/lawn](http://www.netlib.org/lapack/lawn)
ScaLAPACK: software hierarchy

- **ScaLAPACK**
- **BLAS**
- **LAPACK**
- **PBLAS**
- **BLACS**
- **MPI/PVM/...**

**Global**

**Local**

- **Communication routines targeting linear algebra operations.**
- **Parallel BLAS.**

- **Clarity, modularity, performance and portability. Atlas can be used here for automatic tuning.**

- **Communication layer (message passing).**

**Platform specific**

- **Linear systems, least squares, singular value decomposition, eigenvalues.**
ScaLAPACK: 1D block-cyclic distribution

Block-Cyclic Partitioning
Block size = 2, Cyclic on 3-PE Grid

Global Array

PE 0
11 12 13 14 15 16 17 18 19

PE 1
11 12 13 14 19

PE 2
15 16

Local Arrays

Grid Row 0
0
11 12 17 18
13 14 19
15 16

Grid Row 1
1

Partioned Array often shown by this type of grid map

Indicates sequence of cyclic distribution.
ScalAPACK: 2D block-cyclic distribution

5x5 matrix partitioned in 2x2 blocks

\[
\begin{array}{ccc}
  a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
  a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\
  a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
  a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\
  a_{51} & a_{52} & a_{53} & a_{54} & a_{55}
\end{array}
\]

2x2 process grid point of view

\[
\begin{array}{ccc}
  a_{11} & a_{12} & a_{15} \\
  a_{21} & a_{22} & a_{25} \\
  a_{51} & a_{52} & a_{55} \\
  a_{41} & a_{42} & a_{45} \\
  a_{31} & a_{32} & a_{35} \\
  a_{33} & a_{34} \\
\end{array}
\]

http://acts.nersc.gov/scalapack/hands-on/datadist.html
BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines
- describes the underlying (virtual) parallel platform
- processes embedded in 2D topology
- provides point-to-point, collective communication and support functionality
- collective communication over a row, column or over all processes in the 2D grid
BLACS: controls a virtual process grid

General information:
CALL BLACS_PINFO(iam,nprocs)

Perform setup if needed:
IF (NPROCS < 1) THEN
  IF (IAM == 0) NPROC=12
  CALL BLACS_SETUP(IAM,NPROCS)
ENDIF

Get default system context (communicator):
CALL BLACS_GET(0,0,ICTXT)

Initialize the grid:
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)

If in the grid, perform the computation:
IF (MYROW /= -1) THEN
  ...
  CALL BLACS_GRIDEXIT(ICTXT)
ENDIF
CALL BLACS_EXIT(0)
BLACS: communication routines

<table>
<thead>
<tr>
<th>(Data type)</th>
<th>xx (Matrix type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I: Integer, ( \mathbb{R} ): Real, ( \mathbb{D} ): Double Precision, C: Complex, Z: Double Complex</td>
<td>GE: General rectangular matrix, TR: Trapezoidal matrix</td>
</tr>
</tbody>
</table>

Send/receive submatrix from one process to another:

\[ _{xx}SD2D(ICTXT,\{UPLO,DIAG\},M,N,A,LDA,RDEST,CDEST) \]
\[ _{xx}RV2D(ICTXT,\{UPLO,DIAG\},M,N,A,LDA,RSRC,CSRC) \]

Broadcast (and explicitly receive) submatrix:

\[ _{xx}BS2D(ICTXT,SCOPE,TOP,\{UPLO,DIAG\},M,N,A,LDA) \]
\[ _{xx}BR2D(ICTXT,SCOPE,TOP,\{UPLO,DIAG\},M,N,A,LDA,RSRC,CSRC) \]

Global combine operations, element-wise sum, \(|\max|\), \(|\min|\):

\[ _{GSUM2D}(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST) \]
\[ _{GAMX2D}(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST) \]
\[ _{GAMN2D}(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST) \]
PBLAS: parallel basic LA subroutines

- Level 1 PBLAS
  Vector-Vector operations

- Level 2 PBLAS
  Matrix-Vector operations

- Level 3 PBLAS
  Matrix-Matrix operations
### PBLAS: operations

<table>
<thead>
<tr>
<th><em>(Data type)</em></th>
<th>xx_(Matrix type)_</th>
<th>YYY (Operation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S: Real,</td>
<td>GE: All matrix operands are General rectangular,</td>
<td>MM: Matrix-matrix product;</td>
</tr>
<tr>
<td>D: Double Precision,</td>
<td>HE: One of the matrix operands is Hermitian,</td>
<td>MV: Matrix-vector product;</td>
</tr>
<tr>
<td>C: Complex,</td>
<td>SY: One of the matrix operands is Symmetric,</td>
<td>R: Rank-1 update of a matrix;</td>
</tr>
<tr>
<td>Z: Double Complex,</td>
<td>TR: One of the matrix operands is Triangular,</td>
<td>R2: Rank-2 update of a matrix;</td>
</tr>
</tbody>
</table>

- Naming scheme follows BLAS conventions: \texttt{P}_XXYYY
- No vector rotations, banded matrix routines
- Matrix transposition: \texttt{P}_TRANx
- Level 1 routines have meaningful names: \texttt{PSCOPY, PSDOT}, ..
PBLAS: global view of data

PBLAS provides global view of matrix, allowing global addressing

**BLAS:**
```
CALL DGEXX(M, N, A(IA,JA), LDA,...)
```

**PBLAS:**
```
CALL PDGEXX(M, N, A, IA, JA, DESCA, ...)
```
PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout
  - Type and context: DTYPE_, CTXT_
  - Matrix dimensions: M_, N_, MB_, NB_
  - Coordinates owning first matrix entry: RSRC_, CSRC_
  - Leading local dimension: LLD_
ScaLAPACK: Interfaces

- **SL**: Linear Equations
- **SV**: LU Solver
- **VD**: Singular Value
- **EV**: Eigenvalue
- **GVX**: Generalized Eigenvalue

Data Types:
- **real**
- **double**
- **cmplx**
- **dble cmplx**

Matrix Types:
- **GB**: General Band
- **GE**: GEneral matrices (e.g., unsymmetric, rectangular, etc.)
- **GG**: General matrices, Generalized problem
- **HE**: Complex Hermitian
- **OR**: Orthogonal Real
- **PB**: Positive definite Banded (symmetric or Hermitian)
- **PO**: Positive definite (symmetric or Hermitian)
- **PT**: Positive definite Tridiagonal (symmetric or Hermitian)
- **ST**: Symmetric Tridiagonal Real
- **SY**: SYmmetric
- **TR**: TRiangular (possibly quasi-triangular)
- **TZ**: TrapeZoidal
- **UN**: UNitary complex

Computation Performed:
- **P**: (Not specified)
- **X**: (Not specified)
- **S**: (Not specified)
- **D**: (Not specified)
- **C**: (Not specified)
- **Z**: (Not specified)
ScaLAPACK: example

- Example: initialize process grid, read a matrix and right hand side from file, solve system with LU decomposition, write result to file

- Code:
  - gele:~wsawyer/Examples/ScaLAPACK/Ex5
  - Auxiliaries: pslaread.F90, pslawrite.F90, psscaexinfo.F90
  - Main program: psscaex.F90
  - Configuration: SCAEX.dat
  - Data: SCAEXMAT.dat, SCAEXRHS.dat
  - Makefile
Test data: matrix and right hand side

- **Matrix A**: SCAEXMAT.dat
- **Vector b**: SCAEXRHS.dat

\[
A = \begin{pmatrix}
6 & 0 & -1 & 0 & 0 & 0 \\
3 & -3 & 0 & 0 & 0 & 0 \\
0 & -1 & 11 & 0 & 0 & 0 \\
0 & 1 & 0 & -11 & 2 & 8 \\
3 & 1 & 0 & 0 & -4 & 0 \\
0 & 0 & 10 & 0 & 0 & -10
\end{pmatrix}
\]

\[
b = \begin{pmatrix}
72 \\
0 \\
160 \\
0 \\
0 \\
0
\end{pmatrix}
\]
Case Study 1: assignment

- Work in pairs
- Copy over `gele:~wsawyer/Examples/ScaLAPACK/Ex5/*`
- Read through `psscaex.F90`; try to understand code
- Load ScaLAPACK: module load `xt-libsci`
- Modify Makefile; make; code does not compile!
- Edit `psscaex.F90`: replace “***” with correct code
- Edit `psscaex_run.sh` for your account; run!
- Optional: define a larger problem, with different grid
Case Study 2: Trilinos eigen-solver

- Trilinos, like PETSc, hides parallel complexity through opaque interfaces
- Eigen-solver package: Anasazi (named after the native American tribe), a solver library as well as framework for solver development
- Anasazi: depends on lower level Teuchos (utilities) package and can use Epetra/EpetraExt (matrix/vector/operator package) for linear algebra operations
Lessons from ARPACK / P_ARPACK

- ARPACK (F77) is a popular eigen-solver library, with several novel features when released (1992):
  - Reverse communication mechanism allowing user to define mat-vec product, preconditioner, etc.
  - The implicitly-restarted Arnoldi algorithm, efficiently finding a small set of eigenvalues around a given value

- Some limitations:
  - Dependent on one underlying implementation of LA primitives; reverse communication is high maintenance
  - Interfaces were not abstract, revealing underlying complexity
  - Limited to one, albeit successful, method
Anasazi: design objectives

- Opaque objects: hide low level complexity
- Flexibility to allow for various linear algebra primitives; ease of incorporation with other frameworks
- Provide a small set of ‘turn-key’ eigen-solvers for large (sparse) matrices
- Provide a ‘workbench’ framework in which new methods can be implemented
Epetra: linear algebra primitives

- Anasazi can (but does not need to) use the Epetra and EpetraExt packages for linear algebra primitives

- Epetra contents
  - *Serial* and *distributed matrix* and *(multi-)*vector objects (various types)
  - Operator objects to avoid explicit matrix construction
  - Map object of global to local indices for distributed case
  - Communicator object
  - Import/Export objects for off-core communication
  - Time, Flop and other utilities
Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local
- Epetra_LAPACK class: thin layer on top of LAPACK
- Distributed vectors and matrices require a Epetra_map defines the global-to-local mapping
- Multiple ways to define maps; some opaque (best balance algorithm), others ‘hands on’
- Epetra_Import and Epetra_Export transfer data between two maps
Epetra: distributed matrices

- Virtual class: Epetra_RowMatrix; derived classes:
  - Epetra_CrsMatrix: compressed row storage
  - Epetra_VbrMatrix: matrices with block structure, may have differently sized blocks
  - Epetra_FECrsMatrix / Epetra_FEVbrMatrix: matrices arising from finite element discretizations

- Strategies to define a sparse matrix
  - *Suggest* number of non-zeros per row
  - Specify the global indices on each process
Epetra: defining a sparse matrix

```c
int NumGlobalElements=50
Epetra_Map Map(NumGlobal;Elements,0,Comm);
int NumMyElements = Map.NumMyElements();
int * MyGlobalElements = Map.MyGlobalElements();
int * NumNz = new int[NumMyElements];
for( int i=0 ; i<NumMyElements ; i++ )
  if( MyGlobalElements[i]==0 || MyGlobalElements[i] == NumGlobalElements-1)
      NumNz[i] = 2;
  else
      NumNz[i] = 3;

Epetra_CrsMatrix A(Copy,Map,NumNz)
// At this point there are two nonzeros in first and last row
// Otherwise three nonzeros per row
```
Epetra: filling in matrix structure and values

double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
    if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
    else if (MyGlobalElements[i] == NumGlobalElements-1) {
        Indices[0] = NumGlobalElements-2; NumEntries = 1; }
    else {
        Indices[0] = MyGlobalElements[i]-1;
        Indices[1]=MyGlobalElements[i]+1; NumEntries = 2; }
}
A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+1);
Epetra: defining a matrix-vector product operator

- Many applications never construct matrix
- Instead define an operator with Epetra_operator

```cpp
class TriDiagonalOperator : public Epetra_operator{
public: // constructors, methods
private:
    Epetra_Map Map_;  // sub-diagonal value
    double diag_minus_one_;  // sub-diagonal value
    double diag_;  // values on diagonal
    double diag_plus_one_;  // super-diagonal value
}
```
Epetra_operator: applying mat-vec product

```cpp
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
    int Length = X.MyLength();
    for( int vec=0 ; vec<X.NumVectors() ; ++vec ){
        if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
        // first row
        Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
        for( int i=1 ; i<Length-1 ; ++i ){  // intermediate rows
            Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                        diag_minus_one_ * X[vec][i-1];
        }
        // final_row
        Y[vec][Length-1] = diag_ * X[vec][Length-1] +
                           diag_minus_one_ * X[vec][Length-2];
    }
    return true;
}
```
Epetra: pictorial summary
Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS
- Teuchos::LAPACK -- templated wrappers for LAPACK
- Teuchos::ParameterList: container to group parameters
- Teuchos::RCP: smart reference-counted pointer class with garbage collection
- Others…
Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to underlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits
- MultiVecTraits: e.g., MvAddMv, MvDot, MvNorm, …
- OperatorTraits method:

  OperatorTraits<ScalarType, MV, OP>::Apply(const OP &Op, const MV &x, MV &y)
Anasazi: classes for $Ax = \lambda Bx$

- **Anasazi::Eigenproblem**
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec

- **Anasazi::Eigensolution**
  - Manages the solution of the eigen-problem

- **Anasazi::Eigensolver**
  - Defines interface which must be met by any solver
  - Currently implemented solvers: BlockDavidson, BlockKrylovSchur, LOBPCG

- **Anasazi::SolverManager**
  - ‘Turn-key’ class to use existing eigen-solvers
Anasazi: other classes

- **Anasazi::StatusTest**
  - Responsible for stopping the solver iteration

- **Anasazi::SortManager**
  - Sorts the eigenvalues and eigenvectors
  - Increasing/decreasing magnitude, real/imaginary part

- **Anasazi::OrthoManager**
  -Provides methods to perform (re-)orthogonalization

- **Anasazi::OutputManager**
  - Controls verbosity of output
Anasazi: the Epetra adapter

- Anasazi can use any linear algebra functionality satisfying the multi-vector and operator traits
- Implementing the MV and operator classes is tedious
- Epetra/EpetraExt provide the needed classes, e.g.,

```cpp
#include "AnasaziEpetraAdapter.hpp
typedef Epetra_MultiVector MV;
typedef Epetra_Operator OP;
// Multi-vectors of type MV
Teuchos::RefCountPtr<MM> X = Teuchos::rcp( new MV(…) );
// Operators can be any subclass of OP
Teuchos::RefCountPtr<OP> A = Teuchos::rcp( new Epetra_CrsMatrix(…) );
```
Anasazi: example

- Problem: define a block-tridiagonal matrix arising from a 5-point Laplacian finite-difference stencil on a square domain; find some eigen-pairs with Block Davidson solver, using Epetra

- Code:
  - gele:~wsawyer/Examples/Trilinos/BD/*
  - Main program: BlockDavidsonEpetraEx.cpp
  - Batch script: BDEE_run.sh
  - Makefile
Case Study 2: assignment

- Work in pairs
- Copy over `gele:~wsawyer/Examples/Trilinos/Anasazi/BD/*`
- Read through `BlockDavidsonEpetraEx.cpp`
- Load Trilinos: module load trilinos
- Modify Makefile; make; code does not compile!
- Edit `BlockDavidsonEpetraEx.cpp`: replace “***” with correct code
- Edit `BDEE_run.sh` for your account; run!
- Optional: define a larger problem on different numbers of cores
Case Study 3: NOX for non-linear ODE systems

- Solve $F(x) = 0$ with $F(x) = \left( \begin{array}{c} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \end{array} \right)$ and $J_{i,j} = \frac{\partial F_i}{\partial x_j}(x)$

- User supplies:
  - Function $F(x)$ evaluation
  - Optional: Jacobian evaluation, preconditioner

- Not based on a particular linear algebra package, but Epetra is a possible choice (as in Anasazi)

- Users required to derive from abstract classes:
  - NOX::Abstract::Vector – basic vector operations
  - NOX::Abstract::Group – function and Jacobian evaluation
NOX: example

- Non-linear PDE on square domain
- Dirichlet boundary condition
- Point discretization
- 5 point finite difference stencil

\[-\Delta u + \lambda e^u = 0 \quad \text{in} \quad \Omega = (0,1) \times (0,1)\]

\[u = 0 \quad \text{on} \quad \partial \Omega\]

\[-\frac{u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h^2} - \lambda e^{u_{i,j}} = 0\]
NOX: example Jacobian

\[ J = \frac{1}{h^2} \begin{bmatrix} T_1 & -I & & & \\ -I & T_2 & \ddots & & \\ & \ddots & \ddots & -I & \\ & & -I & T_n \end{bmatrix} \]

\[ T_k = \begin{bmatrix} 2 + \lambda h^2 e^{u_{(k-1)n+1,(k-1)n+1}} & -1 & & & \\ -1 & 2 + \lambda h^2 e^{u_{(k-1)n+2,(k-1)n+2}} & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 2 + \lambda h^2 e^{u_{kn,kn}} \end{bmatrix} \]
NOX: non-linear PDE example

- Code:
  - `gele:~wsawyer/Examples/Trilinos/NOX/NL_PDE/*`
  - Main program: `NOXNewton2.cpp`
  - Batch script: `NL_PDE_run.sh`
  - Makefile

- Code requires some minor additions
Case Study 3: assignment

- Work in pairs
- Copy over `~wsawyer/Examples/Trilinos/NOX/NL_PDE/*`
- Read through NOXNewton2.cpp
- Load Trilinos: module load trilinos
- Load PETSc (for solvers): module load petsc
- Modify Makefile; make; code does not compile!
- Edit NOXNewton2.cpp: replace “***” with correct code
- Edit NL_PDE_run.sh for your account; run!
- Optional: define other problem sizes on different numbers of cores; how far will the code scale?
Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples
- Trilinos tutorial
- Tim Stitt, Ladina Gilly: tutorial organization

…and thanks to you for attending!