High Performance Numerical Libraries for Science and Engineering

Osni Marques, Tony Drummond and Andrew Canning
Lawrence Berkeley National Laboratory (LBNL)
osni@nersc.gov, drummond@nersc.gov, canning@nersc.gov
Outline

• Why are still doing Numerical Linear Algebra?
• Dense linear Algebra
  • LAPACK
  • ScaLAPACK
• Sparse Linear Systems
  • Direct Methods
  • Iterative Methods
• Eigenvalue Problems
  • Templates
  • Applications in Material Sciences (Andrew Canning)
• Automatic Tuning
• The ACTS Toolkit
Why are we still doing Numerical Linear Algebra?

- Solving
  \[ Ax = b \]
  \[ Ax = \lambda x \]
  \[ A = U\Sigma V^T \]

- Many mathematical modeling problems reduce to these problems
- Good libraries available
- Why is it still hard?
  - Problems get bigger
  - Parallel machines are hard to program efficiently
  - Computer memory hierarchies
  - How do we choose the best algorithms among many?
- New applications need new algorithms
Fastest Computer Over Time

A computation that took 1 full year to complete in 1980 can today be done in ~ 27 seconds!
Outline

• Why are still doing Numerical Linear Algebra?
• Dense linear Algebra
  • LAPACK
  • ScaLAPACK
• Sparse Linear Systems
  • Direct Methods
  • Iterative Methods
• Eigenvalue Problems
  • Templates
  • Applications in Material Sciences (Andrew Canning)
• Automatic Tuning
• The ACTS Toolkit
LAPACK

http://www.netlib.org/lapack

- Linear Algebra library written in Fortran 77 (Fortran 90, C and C++ versions also available).
- Combine algorithms from LINPACK and EISPACK into a single package.
- Efficient on a wide range of computers (RISC, Vector, SMPs).
- User interface similar to LINPACK (Single, Double, Complex, Double Complex).
- Built atop level 1, 2, and 3 BLAS for high performance, clarity, modularity and portability.
LAPACK: Features

- Basic problems:
  - Linear systems: $Ax = b$
  - Least squares: $\min \| Ax - b \|_2$
  - Singular value decomposition: $A = U\Sigma V^T$
  - Eigenvalues and eigenvectors: $Az = \lambda z$, $Az = \lambda Bz$

- LAPACK does not provide routines for structured problems or general sparse matrices (i.e. sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).

LAPACK: Highlights of Version 3

- **Speed**
  - Faster routines for the Singular Value Decomposition (SVD)
    - 1000x1000 dense SVD **6.3 times faster** than Version 2 on an Intel Pentium 3
    - 1000x1000 dense SVD **16.8 times faster** than Version 2 on an IBM Power 3
  - Faster routines for Least Squares Problem $\min_x \|Ax-b\|_2$ with rank deficient $A$ (uses SVD)
    - 1000x1000 problem **8.5 faster** than Version 2 on an Alpha EV6
- **Reliability, including error bounds for everything**
  - Now available for generalized eigenvalue problem $Ax = \lambda Bx$
ScaLAPACK: Structure of the Software

http://acts.nersc.gov/scalapack

ScaLAPACK

PBLAS

BLACS

PVM/MPI/...

Global

Local

LAPACK

BLAS

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

Linear systems, least squares, singular value decomposition, eigenvalues.

Communication routines targeting linear algebra operations.

Parallel BLAS.

Version 1.7 is now available.

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

http://acts.nersc.gov/scalapack
ScalAPACK: Goals

- **Efficiency**
  - Optimized computation and communication engines
  - Block-partitioned algorithms (Level 3 BLAS) for good node performance

- **Reliability**
  - Whenever possible, use LAPACK algorithms and error bounds.

- **Scalability**
  - As the problem size and number of processors grow
  - Replace LAPACK algorithm that did not scale (new ones into LAPACK)

- **Portability**
  - Isolate machine dependencies to BLAS and the BLACS

- **Flexibility**
  - Modularity: build rich set of linear algebra tools (BLAS, BLACS, PBLAS)

- **Ease-of-Use**
  - Calling interface similar to LAPACK
ScaLAPACK: \textit{Two-Dimensional Block-Cyclic Distribution}

5x5 matrix partitioned into 2x2 blocks and distributed into a 2x2 grid

\[
\begin{array}{cc|cc|c}
  1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\
  -2.1 & 2.2 & 2.3 & 2.4 & 2.5 \\
  -3.1 & -3.2 & 3.3 & 3.4 & 3.5 \\
  -4.1 & -4.2 & -4.3 & 4.4 & 4.5 \\
  -5.1 & -5.2 & -5.3 & -5.4 & 5.5 \\
\end{array}
\]

```
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

IF ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
  A(1) = 1.1; A(2) = -2.1; A(3) = -5.1;
  A(1+LDA) = 1.2; A(2+LDA) = 2.2; A(3+LDA) = -5.2;
  A(1+2*LDA) = 1.5; A(2+3*LDA) = 2.5; A(3+4*LDA) = -5.5;
ELSE IF ( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
  A(1) = 1.3; A(2) = 2.3; A(3) = -5.3;
  A(1+LDA) = 1.4; A(2+LDA) = 2.4; A(3+LDA) = -5.4;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
  A(1) = -3.1; A(2) = -4.1;
  A(1+LDA) = -3.2; A(2+LDA) = -4.2;
  A(1+2*LDA) = 3.5; A(2+3*LDA) = 4.5;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
  A(1) = 3.3; A(2) = -4.3;
  A(1+LDA) = 3.4; A(2+LDA) = 4.4;
END IF
```

LDA is the leading dimension of the local array, used in the array descriptor (an array that simulates an object in Fortran)
### ScaLAPACK: Functionality

<table>
<thead>
<tr>
<th>Ax = b</th>
<th>SDrv</th>
<th>EDrv</th>
<th>Factor</th>
<th>Solve</th>
<th>Inv</th>
<th>Cond. Est.</th>
<th>Iter. Refin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPD Banded</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SPD Tridiagonal</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>General</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>General Banded</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>General Tridiagonal</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Least squares</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GQR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ax = \lambda x or Ax = \lambda Bx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symmetric</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>General</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Generalized BSPD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SVD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ScaLAPACK: Achieving High Performance

Distributed-Memory Computer

- Use the right number of processors
  - Rule of thumb: \( P = \frac{M \times N}{10^6} \) for an \( M \times N \) matrix. This provides a local matrix of size approximately 1000-by-1000
  - Do not try to solve a small problem on too many processors
  - Do not exceed physical memory
- Use an efficient data distribution.
  - Block size (i.e., \( MB,NB \)) = 64
  - Square processor grid: \( P_{row} = P_{column} \)
- Use efficient machine-specific BLAS (not the Fortran77 reference implementation from \texttt{www.netlib.org}) and BLACS (nondebug, \texttt{BLACSDBGLVL=0} in \texttt{Bmake.inc})
**ScaLAPACK: Two Applications**

*Induced current (white arrows) and charge density (colored plane and gray surface) in crystallized glycine due to an external field (Louie, Yoon, Pfrommer and Canning).*

*Cosmic Microwave Background Analysis, BOOMERanG collaboration, MADCAP code (Apr. 27, 2000).*
On line tutorial:  http://acts.nersc.gov/scalapack/hands-on/main.html

Hands-On Exercises for ScaLAPACK

ScaLAPACK Team
March 2001

Introduction

These exercises provide basic and more advanced programming instruction for writing parallel programs calling the BLACS, PBLAS, and ScaLAPACK. A basic knowledge of Fortran, parallel programming with message-passing, and MPI are assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

Detailed information on the BLACS, PBLAS, and ScaLAPACK may be found at the respective URLs:

- http://www.netlib.org/blacs
- http://www.netlib.org/scalapack
- http://www.netlib.org/scalapack/pblas_cref.html

Exercises 1 and 2 give an introduction to parallel programming with the Basic Linear Algebra Communication Subprograms (BLACS). Exercises 3, 4, and 5 provide a range of simplistic to more complex parallel programs calling ScaLAPACK and PBLAS. More example programs for ScaLAPACK can be found at http://www.netlib.org/scalapack/examples

The instructions for the exercises assume that the underlying system is an IBM SP or a Cray T3E, using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. The version of MPI used in these examples is the version 3.0, and we assume the user has this version installed.

These hands-on exercises were prepared in collaboration with the Joint Institute for Computational Science at the University of Tennessee, based on contributions from A. YarKhan, C. Hastings, S. Blackford, C. Whaley, A. Petitet and O. Marques.

Exercise 1: BLACS - Hello World Example
Exercise 2: BLACS - Pi Example
Exercise 3: ScaLAPACK - Example Program 1
Exercise 4: ScaLAPACK - Example Program 2
Exercise 5: PBLAS Example

Help: useful calling sequences

Download all exercises
Outline

• Why are still doing Numerical Linear Algebra?
• Dense linear Algebra
  • LAPACK
  • ScaLAPACK
• Sparse Linear Systems
  • Direct Methods
  • Iterative Methods
• Eigenvalue Problems
  • Templates
  • Applications in Material Sciences (Andrew Canning)
• Automatic Tuning
• The ACTS Toolkit
Selecting a Direct Solver for $Ax=b$

- Use a direct solver if
  - Time and storage space acceptable
  - Iterative methods don’t converge
  - Many $b$’s for same $A$
- Criteria for choosing a direct solver
  - Symmetric positive definite (SPD)
  - Symmetric
  - Symmetric-pattern
  - Unsymmetric
- Row/column ordering schemes available
  - MMD, AMD, ND, graph partitioning
- Hardware
Steps Required by a Typical Sparse Solver

1. An ordering step that reorders the rows and columns of the matrix (to reduce fill-in the factors or to produce a special structure, such as a block tridiagonal form).
2. An analysis step or symbolic factorization to determine the nonzero structures and create suitable data structures for the $L$ and $U$ factors.
3. A numerical factorization to compute the $L$ and $U$ factors.
4. A solve step that performs forward and back substitution using the $L$ and $U$ factors.
Three Forms of Cholesky Factorization

- **blue**: computed and accessed
- **red**: current column
- **green**: updated

row Cholesky  |  column Cholesky left-looking  |  submatrix Cholesky right-looking
Frontal Methods

- Origins in finite-element problems
  \[ A = \sum_{l=1}^{m} A^{(l)} , \quad a_{ij} \leftarrow a_{ij} + a_{ij}^{(l)} \]
- Gaussian elimination
  \[ a_{ij} \leftarrow a_{ij} - a_{ip} (a_{pp})^{-1} a_{pj} \]
- For nonelemental problems, the rows of \( A \) are added into the frontal matrix one at a time
- Separate fronts can be handled simultaneously
  - Sparsity preserving ordering
  - Multifrontal Methods

- The elimination can be performed when all these terms are fully summed
- The assembly and elimination can be interleaved
- The intermediate work is performed in a dense matrix, the frontal matrix
## Direct Methods: Serial Implementations

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>Sym-pat</td>
<td>HSL</td>
</tr>
<tr>
<td>MA42</td>
<td>Frontal</td>
<td>Unsym</td>
<td>HSL</td>
</tr>
<tr>
<td>MA48</td>
<td>Right-looking</td>
<td>Unsym</td>
<td>HSL</td>
</tr>
<tr>
<td>MA57</td>
<td>Multifrontal</td>
<td>Sym</td>
<td>HSL</td>
</tr>
<tr>
<td>MA67</td>
<td></td>
<td>$\begin{pmatrix} H &amp; A^T \ A &amp; 0 \end{pmatrix}, \quad H = H^T$</td>
<td>HSL</td>
</tr>
<tr>
<td>SPARSE</td>
<td>Right-looking</td>
<td>Unsym</td>
<td>Kundert</td>
</tr>
<tr>
<td>SPARSPAK</td>
<td>Left-looking</td>
<td>SPD</td>
<td>George</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym and Sym-pat</td>
<td>Ashcraft</td>
</tr>
<tr>
<td>SuperLLT</td>
<td>Left-looking</td>
<td>SPD</td>
<td>Ng</td>
</tr>
<tr>
<td>SuperLU</td>
<td>Left-looking</td>
<td>Unsym</td>
<td>Li</td>
</tr>
<tr>
<td>UMFPACK</td>
<td>Multifrontal</td>
<td>Unsym</td>
<td>Davis</td>
</tr>
</tbody>
</table>

**HSL:** Harwell Subroutine Library, [http://www.cse.circ.ac.uk/Activity/HSL](http://www.cse.circ.ac.uk/Activity/HSL)
# Direct Methods: *Shared Memory Implementations*

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholesky</td>
<td>Left-looking</td>
<td>SPD</td>
<td>Rothberg</td>
</tr>
<tr>
<td>DMF</td>
<td>Multifrontal</td>
<td>Sym</td>
<td>Lucas</td>
</tr>
<tr>
<td>MA41</td>
<td>Multifrontal</td>
<td>Sym-pat</td>
<td>HSL</td>
</tr>
<tr>
<td>PanelLLT</td>
<td>Left-looking</td>
<td>SPD</td>
<td>Ng</td>
</tr>
<tr>
<td>PARASPAR</td>
<td>Right-looking</td>
<td>Unsym</td>
<td>Zlatev</td>
</tr>
<tr>
<td>PARDISO</td>
<td>Left-right looking</td>
<td>Sym-pat</td>
<td>Schenk</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym and Sym-pat</td>
<td>Ashcraft</td>
</tr>
<tr>
<td>SuperLU_MT</td>
<td>Left-looking</td>
<td>Unsym</td>
<td>Li</td>
</tr>
</tbody>
</table>
## Direct Methods: Distributed Memory Implementations

<table>
<thead>
<tr>
<th>Code</th>
<th>Technique</th>
<th>Scope</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAPSS</td>
<td>Multifrontal</td>
<td>SPD</td>
<td>Raghavan</td>
</tr>
<tr>
<td>DMF</td>
<td>Multifrontal</td>
<td>Sym</td>
<td>Lucas</td>
</tr>
<tr>
<td><strong>MUMPS</strong></td>
<td>Multifrontal</td>
<td>Sym and Sym-pat</td>
<td>Amestoy</td>
</tr>
<tr>
<td>PaStiX</td>
<td>Left-right looking</td>
<td>SPD</td>
<td>CEA</td>
</tr>
<tr>
<td>PSPASES</td>
<td>Multifrontal</td>
<td>SPD</td>
<td>Gupta</td>
</tr>
<tr>
<td>SPOOLES</td>
<td>Left-looking</td>
<td>Sym and Sym-pat</td>
<td>Ashcraft</td>
</tr>
<tr>
<td><strong>SuperLU_DIST</strong></td>
<td>Right-looking</td>
<td>Unsym</td>
<td>Li</td>
</tr>
<tr>
<td>S+</td>
<td>Right-looking</td>
<td>Unsym</td>
<td>Yang</td>
</tr>
</tbody>
</table>

MUMPS

Multifrontal Massively Parallel Sparse Direct Solver: http://www.enseeiht.fr/lima/apo/mumps

- Distributed Multifrontal Solver
- F90 and MPI based
- Stability based on partial pivoting
- Dynamic Distributed Scheduling to accommodate both numerical fill-in and multi-user environment
- Use of BLAS, LAPACK, ScaLAPACK
MUMPS: Main Features

- Solution of linear systems with
  - symmetric positive definite matrices
  - general symmetric matrices
  - general unsymmetric matrices
- Parallel factorization and solve phases
  (uniprocessor version also available)
- Iterative refinement and backward error analysis
- Input matrix in
  - assembled format
  - distributed assembled format
  - elemental format
- Null space functionalities (rank detection and null space basis)
- Schur complement matrix
**SuperLU**

http://acts.nersc.gov/superlu

- Solves $Ax=b$ on by sparse Gaussian elimination
- Sequential, SMP and distributed memory (MPI) implementations
- Suitable for general sparse $A$, nonsymmetric, real or complex
- Performance depends strongly on
  - Sparsity structure, good if (number of flops) / (number of nonzeros) is large
  - Ordering of equations and unknowns (controls fill-in, parallelism)
SuperLU: Unsymmetric Supernode

- Exploit dense submatrices in the $L$ and $U$ factors of $PA=LU$
- Permit use of level 3 BLAS
- Reduce inefficient indirect addressing
- Reduce symbolic time by traversing a coarser grid
- Supernode panel factorization (left-looking)
SuperLU: Performance Highlights

- **Sequential:** up to 40% of machine peak Mflop rate on IBM Power 2 and MIPS R8K for matrices
  - EX11 (3D CFD, $n=16614$, nonzeros/row = 66)
  - RAEFSKY4 (buckling of a container, $n=19779$ nonzeros/row = 67)
  - VAVASIS3, 2D PDE, $n=41092$, nonzeros/row = 41
- **Shared Memory**
  - 8-10 times speedup on 18 PE SGI Origin for the above 3 matrices
  - 17% to 33% of machine peak Mflop rate for EX11 on AlphaServer 8400 (8 PEs), Origin 2000 (20 PEs), Power Challenge (12 PEs), Cray J90 (16 PEs)
Distributed SuperLU: *Performance Highlights*

- Uses static instead of dynamic pivoting to be as scalable as Cholesky
- Performance on a 512 processor Cray T3E
  - 10.2 Gflops for MIXING-TANK, fluid flow, $n = 29957$, nonzeros/row = 67
  - 8.4 Gflops for ECL32, device simulation, $n = 51993$, nonzeros/row = 7.3
  - 2.5 Gflops for BBMAT, fluid flow, $n = 38744$, nonzeros/row = 46 (20% parallel efficiency)
- Used to solve open Quantum Mechanics problem
  (Science, 24 Dec 1999)
  - $n = 736 K$ on 64 PEs, Cray T3E in 5.7 minutes
  - $n = 1.8 M$ on 24 PEs, ASCI Blue Pacific in 24 minutes
Outline

- Why are still doing Numerical Linear Algebra?
- Dense linear Algebra
  - LAPACK
  - ScaLAPACK
- Sparse Linear Systems
  - Direct Methods
  - Iterative Methods
- Eigenvalue Problems
  - Templates
  - Applications in Material Sciences (Andrew Canning)
- Automatic Tuning
- The ACTS Toolkit
Selecting an Iterative Solver for $Ax=b$

Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods
Iterative Solvers: Preconditioners

Construct a preconditioning matrix $K$ such that $Kx=b$ is much easier to solve than $Ax=b$ and $K$ is somehow “close” to $A$

- Incomplete $LU$ decompositions
- Sparse approximate inverses
- Polynomial preconditioners
- Preconditioning by blocks or domains
- Element-by-element preconditioners
Trilinos

http://www.cs.sandia.gov/Trilinos

- Trilinos is a multifaceted solver project
- Encompasses efforts in:
  - Linear solvers
  - Eigensolvers
  - Nonlinear and time-dependent solvers
  - Others
- Provides a common framework for current and future solver projects
- Specifically provides:
  - A common set of concrete linear algebra objects for solver development and application interfaces
  - A consistent set of solver interfaces via abstract classes (API)
Trilinos: Concrete Solver Components

- Linear systems:
  - Multi-level preconditioners (ML)
  - Robust algebraic preconditioners (IFPACK)
  - Complex solvers (Komplex)
  - Block iterative methods (BGMRES, BLCG)
  - Object-oriented C++ Aztec (AztecOO)

- Eigensystems:
  - Scalable generalized eigensolver (ANASAZI)

- Nonlinear systems:
  - Suite of nonlinear methods (NLS)
Trilinos: *AztecOO*

- Aztec is the workhorse solver at SNL:
  - Extracted from the MPSalsa reacting flow code.
  - Installed in dozens of SNL applications.
  - 800+ external licenses.
- AztecOO leverages the investment in Aztec:
  - Uses Aztec iterative methods and preconditioners.
- AztecOO improves on Aztec by:
  - Using 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.
**Hypre**

http://acts.nersc.gov/hypre

- **Before writing your code:**
  - choose a conceptual interface
  - choose a solver / preconditioner
  - choose a matrix type that is compatible with your solver / preconditioner and conceptual interface

- **Now write your code:**
  - build auxiliary structures (e.g., grids, stencils)
  - build matrix/vector through conceptual interface
  - build solver/preconditioner
  - solve the system
  - get desired information from the solver
Multiple interfaces are necessary to provide “best” solvers and data layouts.
Hypre: Why multiple interfaces?

- Provides natural “views” of the linear system
- Eases some of the coding burden for users by eliminating the need to map to rows/columns
- Provides for more efficient (scalable) linear solvers
- Provides for more effective data storage schemes and more efficient computational kernels
Hypre: Conceptual Interfaces

- Structured-Grid Interface (Struct)
  - applications with logically rectangular grids
- Semi-Structured-Grid Interface (SStruct)
  - applications with grids that are mostly structured, but with some unstructured features (e.g., block-structured, AMR, overset)
- Finite Element Interface (FEI)
  - unstructured-grid, finite element applications
- Linear-Algebraic Interface (IJ)
  - applications with sparse linear systems

<table>
<thead>
<tr>
<th>Solvers</th>
<th>System Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Struct</td>
</tr>
<tr>
<td>Jacobi</td>
<td>X</td>
</tr>
<tr>
<td>SMG</td>
<td>X</td>
</tr>
<tr>
<td>PFMG</td>
<td>X</td>
</tr>
<tr>
<td>BoomerAMG</td>
<td>X</td>
</tr>
<tr>
<td>ParaSails</td>
<td>X</td>
</tr>
<tr>
<td>PILUT</td>
<td>X</td>
</tr>
<tr>
<td>PCG</td>
<td>X</td>
</tr>
<tr>
<td>GMRES</td>
<td>X</td>
</tr>
</tbody>
</table>
Hypre: Setup and Use of Solvers

- Create the solver
  
  ```c
  HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
  ```

- Set parameters
  
  ```c
  HYPRE_SolverSetTol(solver, 1.0e-06);
  ```

- Prepare to solve the system
  
  ```c
  HYPRE_SolverSetup(solver, A, b, x);
  ```

- Solve the system
  
  ```c
  HYPRE_SolverSolve(solver, A, b, x);
  ```

- Get solution info out via conceptual interface
  
  ```c
  HYPRE_StructVectorGetValues(struct_x, index, values);
  ```

- Destroy the solver
  
  ```c
  HYPRE_SolverDestroy(solver);
  ```
PETSc

http://acts.nersc.gov/petsc

- Portable, Extensible Toolkit for Scientific Computing
- What can it do?:
  - Support the development of parallel PDE solvers
  - Implicit or semi-implicit solution methods, finite element, finite difference, or finite volume type discretizations.
  - Specification of the mathematics of the problem
    - Vectors (field variables) and matrices (operators)
  - How to solve the problem?
    - Linear, non-linear, and timestepping (ODE) solvers
PETSc: Features

- Parallelism
  - Uses MPI
  - Data Layout: structure and unstructured meshes
  - Partitioning and coloring
- Viewers
  - Printing Data Object information
  - Visualization of a field and matrix data
- Profiling and performance Tuning
  - -log_summary
  - Profiling by stages of an application
  - User define events
**PETSc: Components**

<table>
<thead>
<tr>
<th>Nonlinear Solvers</th>
<th>Time Steppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-based Methods</td>
<td>Euler</td>
</tr>
<tr>
<td>Line Search</td>
<td>Backward Euler</td>
</tr>
<tr>
<td>Trust Region</td>
<td>Pseudo Time Stepping</td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Krylov Subspace Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMRES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive Schwartz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Sparse Row (AIJ)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vectors</th>
<th>Index Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
<td>Block Indices</td>
</tr>
</tbody>
</table>
PETSc: simple example

#include "petsc.h"
int main(int argc,char **argv)

Argc and Argv are used to pass run time commands to PETSc and MPI
PETSc: simple example (cont.)

Every PETSc program should begin with the PetscInitialize routine.

```c
int ierr, rank, size;
ierr = PetscInitialize(&argc,&argv,(char *)0,help); CHKERRA(ierr);
/*
The following MPI calls return the number of processes
being used and the rank of this process in the group.
*/
ierr = MPI_Comm_size(PETSC_COMM_WORLD,&size); CHKERRA(ierr);
ierr = MPI_Comm_rank(PETSC_COMM_WORLD,&rank); CHKERRA(ierr);
ierr = PetscPrintf(PETSC_COMM_WORLD,"Number of processors = %d, rank = %d\n",size,rank); CHKERRA(ierr);
ierr = PetscPrintf(PETSC_COMM_WORLD," [%d] Jumbled Hello World\n",rank); CHKERRA(ierr);
ierr = PetscFinalize(); CHKERRA(ierr);
return 0;
}
```

A program must always end with PetscFinalize

Prints a single message
Prints a multiple message
PETSc SLES: basic steps

- Define the linear system \((Ax=b)\)
  - \texttt{MatCreate}, \texttt{MatSetValue}, \texttt{VecCreate}
- Create the Solver
  - \texttt{SLESCreate}, \texttt{SLESSetOperators}
- Solve System of Equations
  - \texttt{SLESSolve}
- Clean up
  - \texttt{SLESDestroy}
PETSc SNES: basic steps

- Non-linear equations of the form:
  - $F(x) = 0$
- Unconstrained Minimization problems of the form:
  - $\text{Min}\{f(x)\}$
- Create the Solver
  - `SNESCreate`
- Create Matrices and vectors (like Jacobian matrix)
  - `MatCreate`, `MatSetValue`, `VecCreate`
- Set evaluation routine and linear solver defaults
- Solve non-linear system: `SNESolve`
- Clean up
Consider the ODE $u_t = F(u,t)$, where $u$ is a finite-dimensional vector.

- Create a TS object
  - `TSCreate`
- Select a solution method (Euler, BEULER, PSEUDO)
- Set initial time and time step
  - `TSSetTimeStep`
- Set the total number of time steps:
  - `TSSetDuration`
- Set the time step context
- Clean up
**PETSc: applications**

*Multiphase flow, 4 million cell blocks, 32 million DOF, over 10.6 Gflops on an IBM SP (128 nodes), entire simulation runs in less than 30 minutes (Pope, Gropp, Morgan, Seperhrnoori, Smith and Wheeler).*

*Prometheus code (unstructured meshes in solid mechanics), 26 million DOF, 640 nodes on NERSC's Cray T3E (Adams and Demmel).*

The parallel version of M3d, a multi-level 3D plasma physics code developed at the Princeton Plasma Physics Laboratory (PPPL), makes extensive use of PETSc for parallelization and solution of an unstructured mesh problem. Recently, one of the members of the PPPL team stated that the parallelization of M3D "would have been very difficult without PETSc, and would have required several physicists to spend a significant amount of time reinventing numerical algorithms instead of doing physics."
Outline

- Why are still doing Numerical Linear Algebra?
- Dense linear Algebra
  - LAPACK
  - ScaLAPACK
- Sparse Linear Systems
  - Direct Methods
  - Iterative Methods
- Eigenvalue Problems
  - Templates
    - Applications in Material Sciences (Andrew Canning)
- Automatic Tuning
- The ACTS Toolkit
Eigenproblems: *a brief tour*

- Hermitian: $Ax = \lambda x$, $A^* = A$
- Generalized Hermitian: $Ax = \lambda Bx$, $A^* = A$, $B^* = B > 0$
- Singular Value Decomposition: $A = U \Sigma V^*$
- Non-Hermitian: $Ax = \lambda x$, $A^* \neq A$
- Generalized Non-Hermitian: $Ax = \lambda Bx$
- Nonlinear: $(\lambda^2 M + \lambda D + K)x = 0$, $P(\lambda) = 0$, ...
Selecting an Eigensolver for $Ax = \lambda x$

- Mathematical properties of the problem
- Desired spectral information
  - Smallest eigenvalue?
  - A few eigenvalues at the end of the spectrum?
  - Eigenvalues somewhere inside spectrum?
  - Most eigenvalues?
  - Eigenvectors, invariant subspace, something else?
  - Accuracy?
- Available operations and their costs
- Store and manipulate full matrix
- Solve $(A-\sigma I)x = b$, or just $Ax = b$
- Multiply $Ax$ and $A^Tx$, or just $Ax$

## Summary of Algorithms for Hermitian Problems

<table>
<thead>
<tr>
<th>Method</th>
<th>Application</th>
<th>Orthogonalization</th>
<th>Eigenvectors</th>
<th>Number of vectors</th>
<th>Factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>a few at the end</td>
<td>several at the end</td>
<td>in the middle</td>
</tr>
<tr>
<td>Power method</td>
<td>direct shift-invert</td>
<td>-</td>
<td>yes</td>
<td>very slow</td>
<td>no yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Subspace iteration</td>
<td>direct shift invert</td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>no yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Lanczos</td>
<td>direct shift invert</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>selective</td>
<td>yes</td>
<td>slow</td>
<td>no yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Lanczos with IR</td>
<td>direct shift invert</td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>no yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Block Lanczos</td>
<td>direct shift invert</td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Jacobi-Davidson</td>
<td>direct precondition shift invert</td>
<td>full</td>
<td>slow</td>
<td>slow</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>full</td>
<td>yes</td>
<td>slow</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Notes:**
- ILU: Incomplete LU
- LU: Lower-upper

**Factorization:**
- Direct
- Shift-invert
Some Eigensolver Implementations

- Subspace Iteration
  - EA12, EB12 (www.cse.clrc.ac.uk/Activity/HSL)
  - PACDYN (pacdyn@cepel.br)
- Symmetric Lanczos
  - laso, lanczos and lanz (www.netlib.org)
  - BLZPACK (www.nersc.gov/~osni)
  - EA15, EA16 (www.cse.clrc.ac.uk/Activity/HSL)
- Nonsymmetric Lanczos
  - ABLE (Bai, Dai, Ye)
  - MSC/Nastran (Komzsik)
- Arnoldi
  - ARPACK (www.caam.rice.edu/software/ARPACK, eigs in Matlab)
  - EB13 (www.cse.clrc.ac.uk/Activity/HSL)
  - ARNCHEB (www.cerfacs.fr/algor/qualitative)
- Jacobi-Davidson:
  - JDQR (www.math.uu.nl/people/vorst)
Outline

• Why are still doing Numerical Linear Algebra?
• Dense linear Algebra
  • LAPACK
  • ScaLAPACK
• Sparse Linear Systems
  • Direct Methods
  • Iterative Methods
• Eigenvalue Problems
  • Templates
  • Applications in Material Sciences (Andrew Canning)
• **Automatic Tuning**
• The ACTS Toolkit
Conventional Performance Tuning

- Motivation: performance of many applications dominated by a few kernels
- Vendor or user hand tunes kernels
- Drawbacks:
  - Very time consuming and tedious work
  - Even with intimate knowledge of architecture and compiler, performance hard to predict
  - Growing list of kernels to tune (example: new BLAS standard)
  - Must be redone for every architecture, compiler
  - Compiler technology often lags architecture
  - Not just a compiler problem:
    - Best algorithm may depend on input, so some tuning at runtime.
    - Not all algorithms semantically or mathematically equivalent
Automatic Performance Tuning

- Approach: for each kernel
  1. Identify and generate a space of algorithms
  2. Search for the fastest one, by running them
- What is a space of algorithms?
  - Depending on kernel and input, may vary
    - instruction mix and order
    - memory access patterns
    - data structures
    - mathematical formulation
- When do we search?
  - Once per kernel and architecture
  - At compile time
  - At run time
  - All of the above
Some Automatic Tuning Projects

- PHiPAC: www.icsi.berkeley.edu/~bilmes/phipac
- ATLAS: www.netlib.org/atlas
- XBLAS: www.nersc.gov/~xiaoye/XBLAS
- Sparsity: www.cs.berkeley.edu/~yelick/sparsity
- FFTs and Signal Processing
  - FFTW: www.fftw.org
    - Won 1999 Wilkinson Prize for Numerical Software
  - SPIRAL: www.ece.cmu.edu/~spiral
    - Extensions to other transforms, DSPs
  - UHFFT
    - Extensions to higher dimension, parallelism
Tuning pays off: PHiPAC
Tuning pays off: **FFTW**
Outline

• Why are still doing Numerical Linear Algebra?
• Dense linear Algebra
  • LAPACK
  • ScaLAPACK
• Sparse Linear Systems
  • Direct Methods
  • Iterative Methods
• Eigenvalue Problems
  • Templates
  • Applications in Material Sciences (Andrew Canning)
• Automatic Tuning
• The ACTS Toolkit
The ACTS Toolkit

Advanced Computational Testing and Simulation

Tools for developing parallel applications
  • developed (primarily) at DOE Labs
  • separate projects originally
  • 20 tools

ACTS is an “umbrella” project
  • leverage numerous independently funded projects
  • collect tools in a toolkit

http://acts.nersc.gov
ACTS: Project Goals

acts-support@nersc.gov    http://acts.nersc.gov

- Bring software packages together into a “toolkit”
- Make the software interoperable
- Provide consistent application interfaces
- Promote general solutions to complex programming needs
- Promote code reusability
- Enable large scale applications
- Long-term support for experimental software
- Outreach and dissemination
What needs to be computed?

\[ Ax = b \]
\[ Az = \lambda z \]
\[ A = U\Sigma V^T \]
\[ \min \left\{ \frac{1}{2} \| r(x) \|^2 : x_l \leq x \leq x_u \right\} \]

PDEs

ODEs

ScaLAPACK

Aztec/Trilinos

SuperLU

PETSc

TAO

Hypre

PVODE

Optimization problems

ODEs

Optimization problems
What codes are being developed?

Parallel programs that use large distributed arrays
Operations with grids for PDE applications
Scripting interface for C++ numerics
Performance analysis and monitoring
Infrastructure for distributed computing
Coupling distributed applications
Interactive visualization
Expression templates for C++

POOMA
SILOON
PAWS
TAU
Overture
Global Arrays
CUMULVS
PETE
ACTS: Lessons Learned

- There is still a gap between tool developers and application developers which leads to duplication of efforts.
- The tools currently included in the ACTS Toolkit should be seen as dynamically configurable toolkits and should be grouped into toolkits upon user/application demand.
- Users demand long-term support of the tools.
- Applications and users play an important role in hardening tools.
- Tools evolve or are superseded by other tools.
- There is a demand for tool interoperability and more uniformity in the documentation and user interfaces.
- There is a need for an intelligent and dynamic catalog/repository of high performance tools.
More Information

- Workshops on the ACTS Toolkit: http://acts.nersc.gov/presentations
  - ScaLAPACK - Parallel Linear Algebra Routines
  - SuperLU - A Parallel Direct Solver
  - Survey of Parallel Direct Solvers
  - Automatic Performance Tuning
  - PETSc - Portable, Extensible Toolkit for Scientific Computation
  - State-of-the-art Iterative Solver Libraries
  - Templates for the Solution of Eigenvalue Problems