Seventh Copper Mountain Conference on Iterative Methods

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?

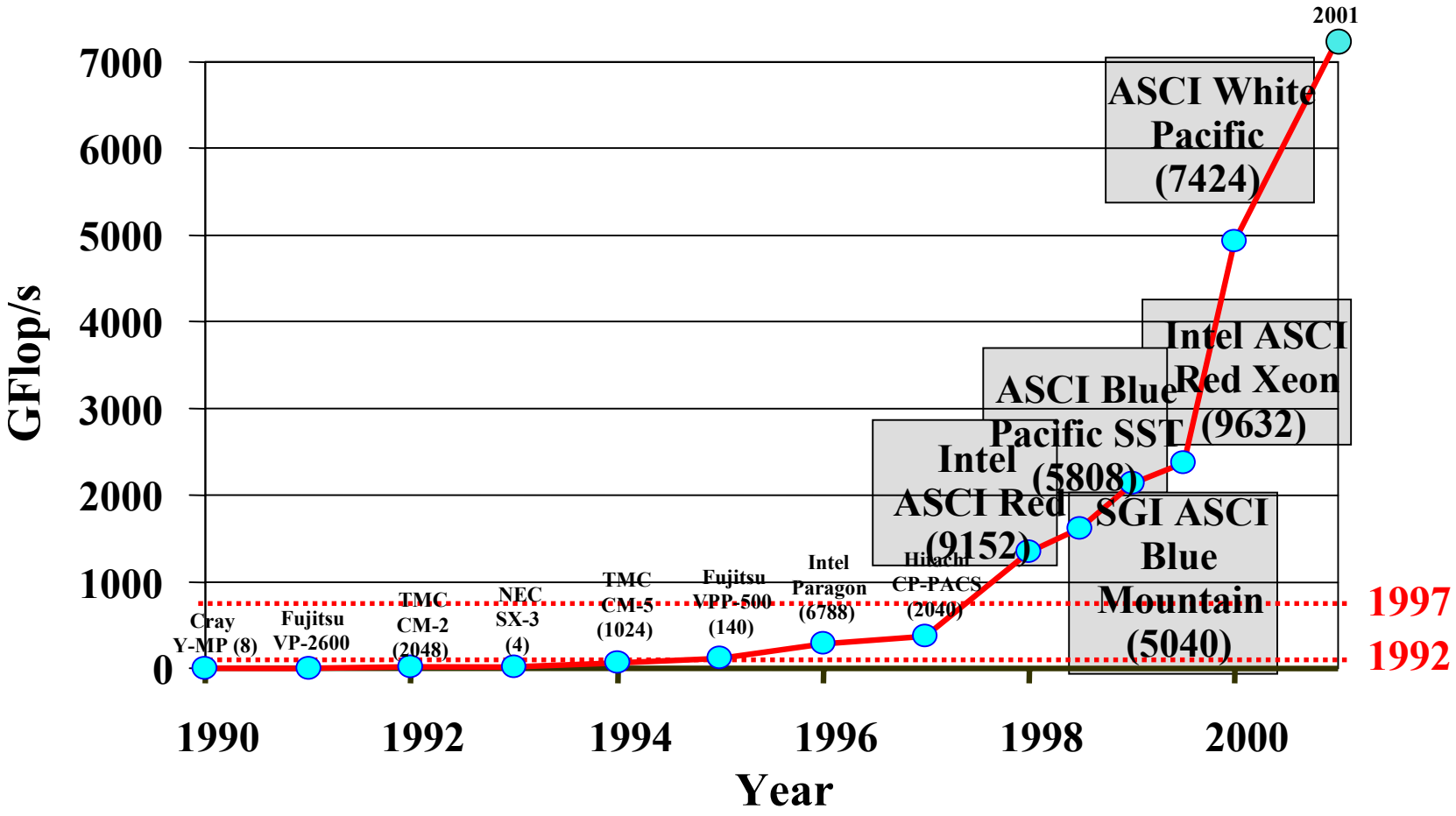
- Solving

  $Ax = b$

  $Ax = \lambda x$

  $A = U\Sigma V^T$
- 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 algorithm 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*

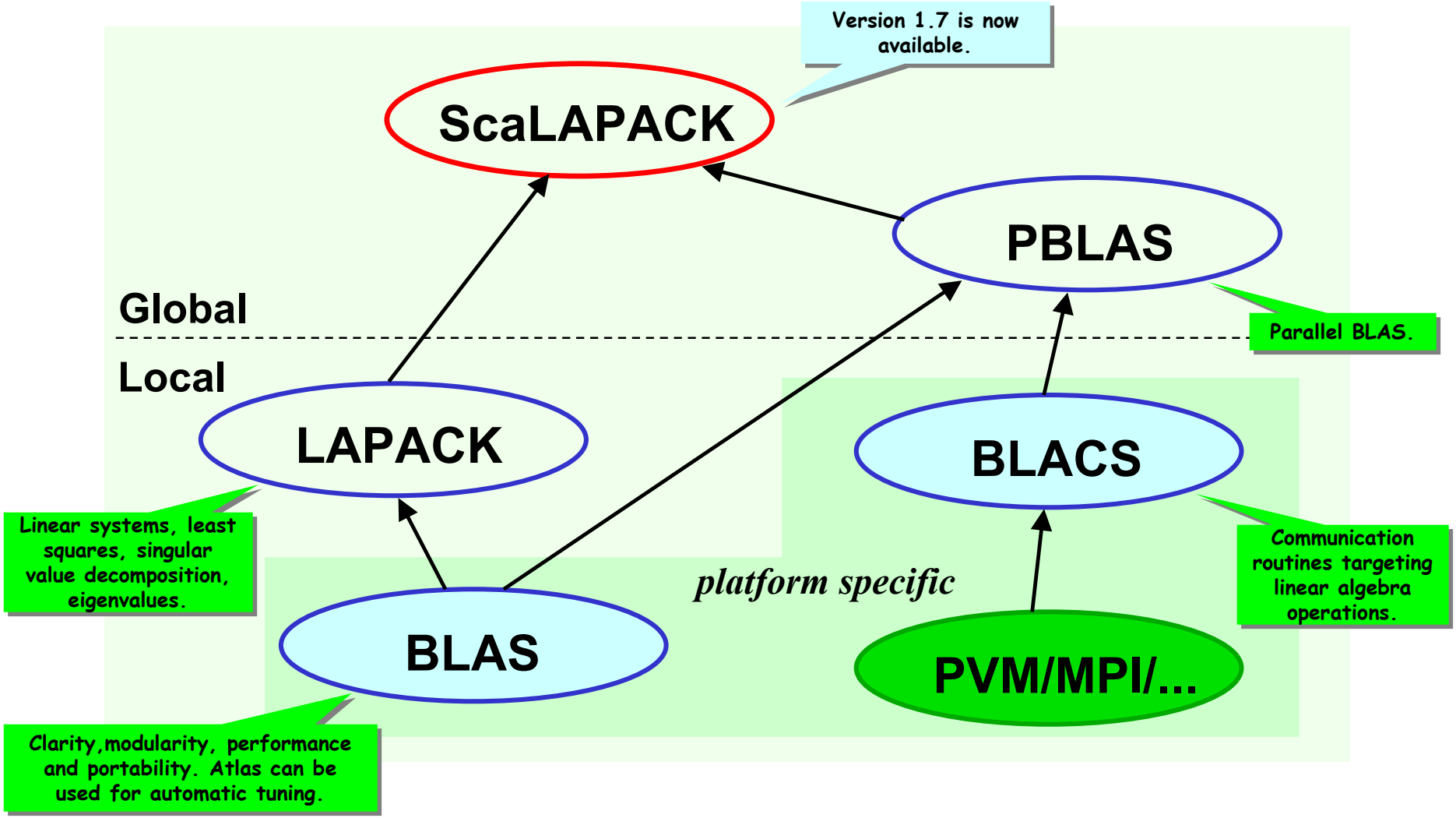
- Basic problems:
  - Linear systems: $Ax = b$
  - Least squares: $\min \|Ax - b\|_2$
  - Singular value decomposition: $A = U\Sigma V^T$
  - Eigenvalues and eigenvectors: $Az = \lambda z$, $Az = \lambda Bz$
- LAPACK does not provide routines for structured problems or general sparse matrices (i.e. sparse storage formats such as compressed-row, -column, -diagonal, skyline ...).
- LAPACK Users' Guide, Third Edition (1999)

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



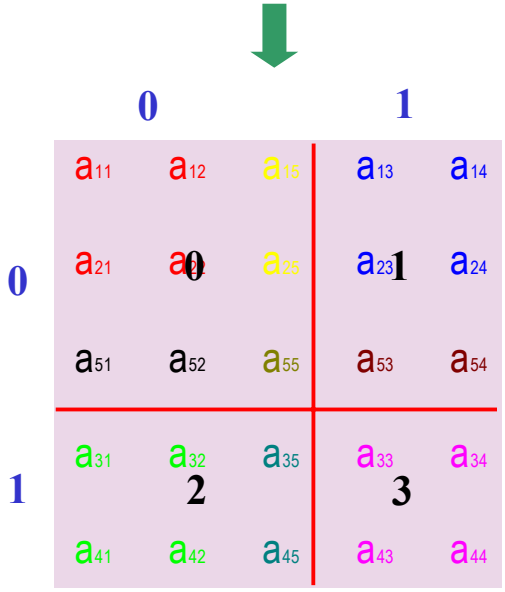
ScaLAPACK: *Goals*

- 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: *Two-Dimensional Block-Cyclic Distribution*

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

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




```
  ⋮
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)

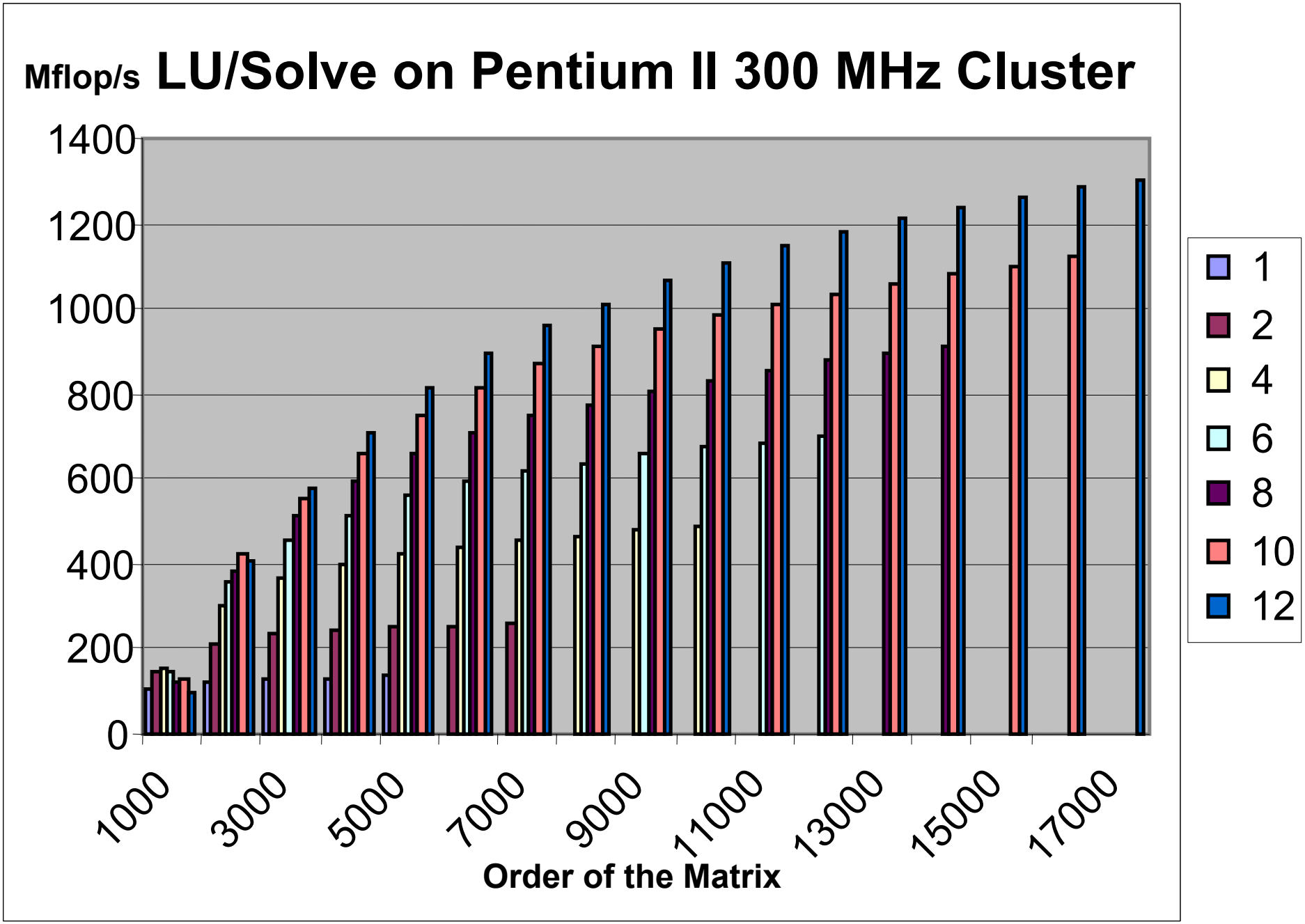
ScaLAPACK: *Functionality*

Ax = b	SDrv	EDrv	Factor	Solve	Inv	Cond. Est.	Iter. Refin.
Triangular				X	X	X	X
SPD	X	X	X	X	X	X	X
SPD Banded	X		X	X			
SPD Tridiagonal	X		X	X			
General	X	X	X	X	X	X	X
General Banded	X		X	X			
General Tridiagonal	X		X	X			
Least squares	X		X	X			
GQR			X				
GRQ			X				
Ax = λx or Ax = λBx	SDrv	Edrv	Reduction	Solution			
Symmetric	X	X	X	X			
General	X	X	X	X			
Generalized BSPD	X		X	X			
SVD			X	X			

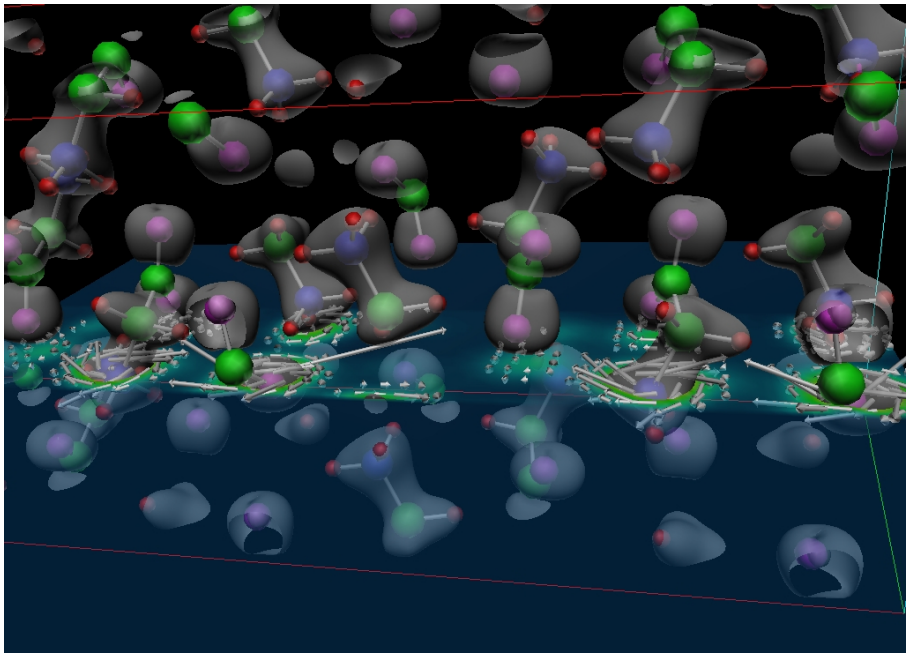
ScaLAPACK: *Achieving High Performance*

Distributed-Memory Computer

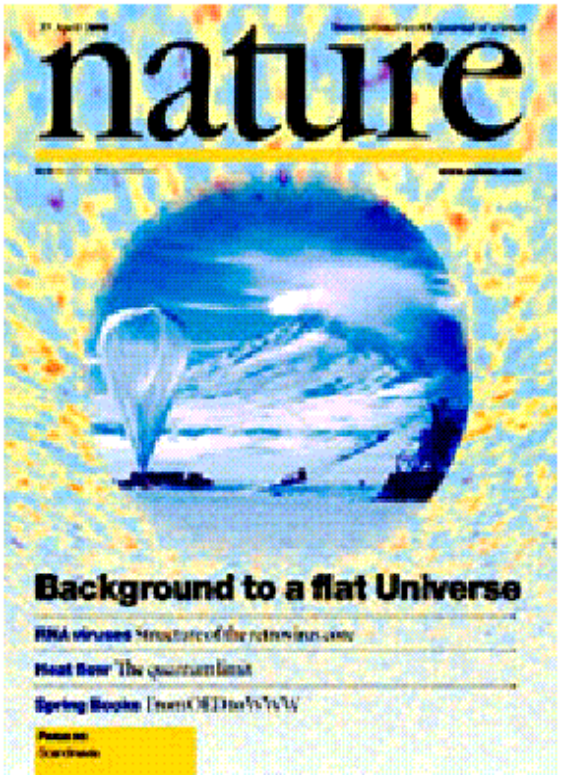
- Use the right number of processors
  - Rule of thumb: P=MxN/10^6 for an MxN matrix. This provides a local matrix of size approximately 1000-by-1000
  - Do not try to solve a small problem on too many processors
  - Do not exceed physical memory
- Use an efficient data distribution.
  - Block size (i.e.,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


Mflop/s
LU/Solve on Pentium II 300 MHz Cluster
1400
1200
1000
800
600
400
200
0
1000
3000
5000
7000
9000
11000
13000
15000
17000
Order of the Matrix
1
2
4
6
8
10
12

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_qref.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 $Ax=b$

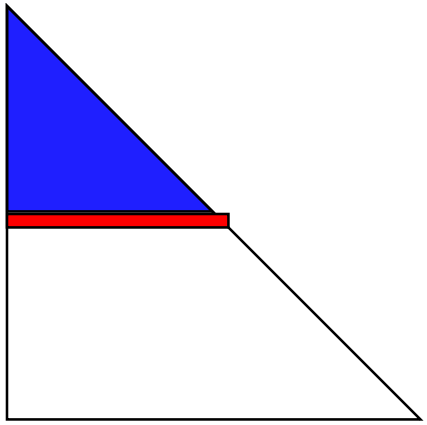
- Use a direct solver if
  - Time and storage space acceptable
  - Iterative methods don't converge
  - Many b's for same A
- Criteria for choosing a direct solver
  - Symmetric positive definite (SPD)
  - Symmetric
  - Symmetric-pattern
  - Unsymmetric
- Row/column ordering schemes available
  - MMD,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

Steps Required by a Typical Sparse Solver

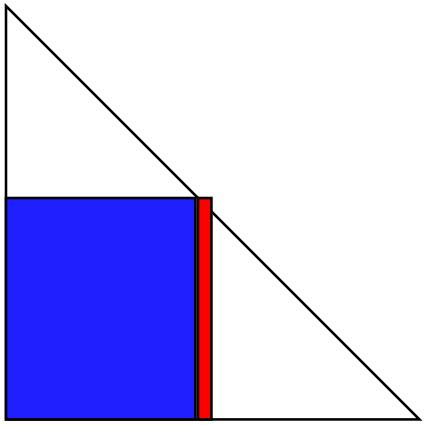
1. An ordering step that reorders the rows and columns of the matrix (to reduce fill-in the factors or to produce a special structure, such as a block tridiagonal form).
2. An analysis step or symbolic factorization to determine the nonzero structures and create suitable data structures for the L and 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

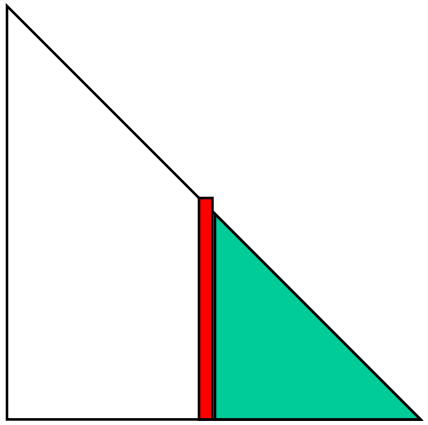
blue: computed and accessed, **red**: current column, **green**: updated



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

$$a_{ij} \Leftarrow a_{ij} - a_{ip}(a_{pp})^{-1} a_{pj}$$

- 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*

- 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

Direct Methods: *Serial Implementations*

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

HSL*: Harwell Subroutine Library, http://www.cse.circ.ac.uk/Activity/HSL*

Direct Methods: *Shared Memory Implementations*

Code	Technique	Scope	Contact
Cholesky	Left-looking	SPD	Rothberg
DMF	Multifrontal	Sym	Lucas
MA41	Multifrontal	Sym-pat	HSL
PanelLLT	Left-looking	SPD	Ng
PARASPAR	Right-looking	Unsym	Zlatev
PARDISO	Left-right looking	Sym-pat	Schenk
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft
SuperLU_MT	Left-looking	Unsym	Li

Direct Methods: *Distributed Memory Implementations*

Code	Technique	Scope	Contact
CAPSS	Multifrontal	SPD	Raghavan
DMF	Multifrontal	Sym	Lucas
MUMPS	Multifrontal	Sym and Sym-pat	Amestoy
PaStiX	Left-right looking	SPD	CEA
PSPASES	Multifrontal	SPD	Gupta
SPOOLES	Left-looking	Sym and Sym-pat	Ashcraft
SuperLU_DIST	Right-looking	Unsym	Li
S+	Right-looking	Unsym	Yang

Analysis and Comparison of Two General Sparse Solvers for Distributed Memory Computers, Amestoy, Duff, L'Excellent and Li, ACM TOMS, 27:388-421, 2001.

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 accomodate 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

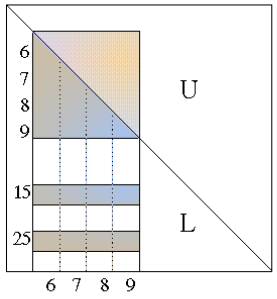
SuperLU

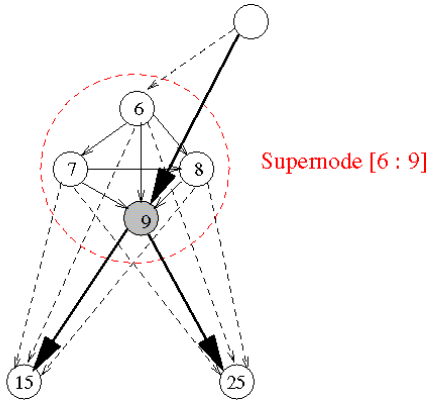
http://acts.nersc.gov/superlu

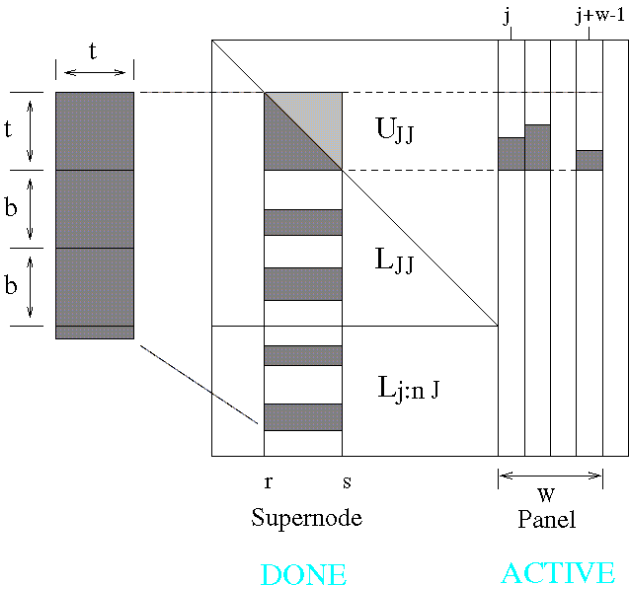
- Solves $Ax=b$ on by sparse Gaussian elimination
- Sequential, SMP and distributed memory (MPI) implementations
- Suitable for general sparse A, nonsymmetric, real or complex
- Performance depends strongly on
  - Sparsity structure, good if (number of flops) / (number of nonzeros) is large
  - Ordering 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 ineficient indirect addressing
- Reduce symbolic time by traversing a coarser grid
- Supernode panel factorization (left-looking)





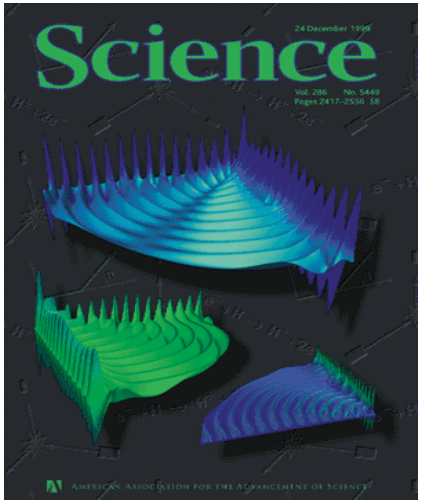


SuperLU: *Performance Highlights*

- 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

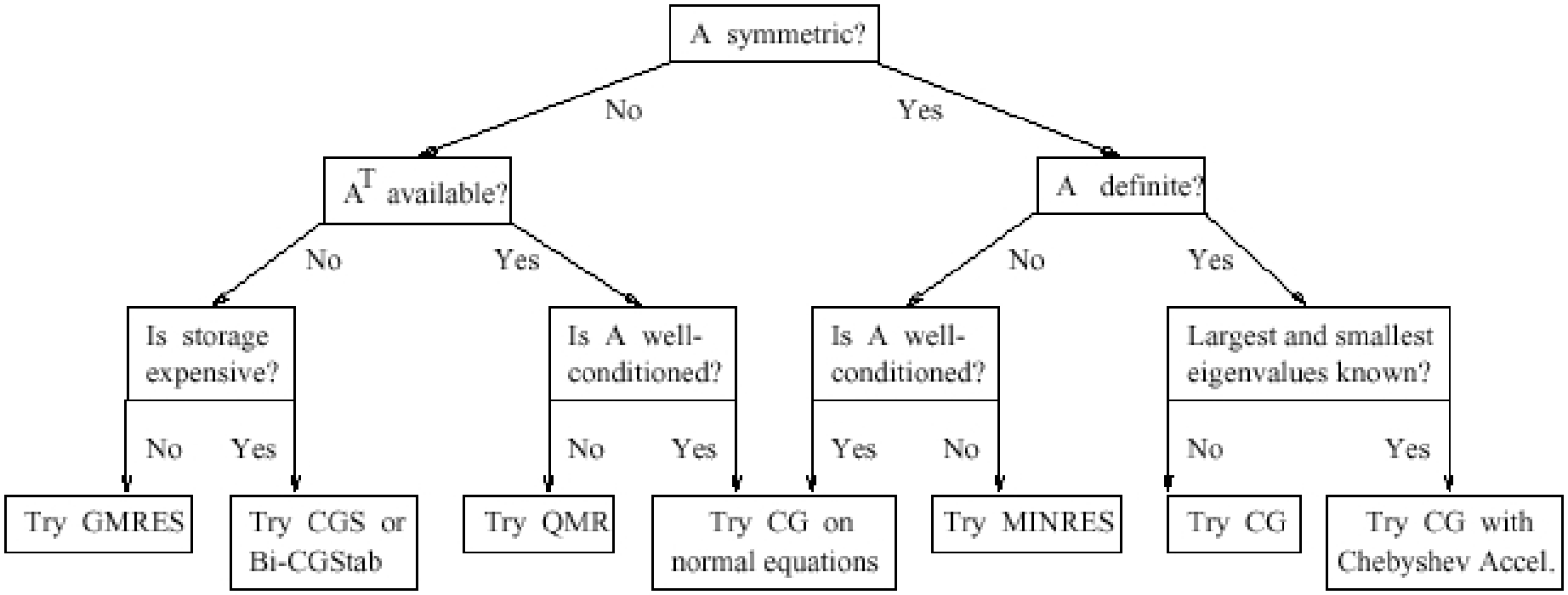


Outline

- 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
http://www.netlib.org/linalg/html_templates/Templates.html



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.

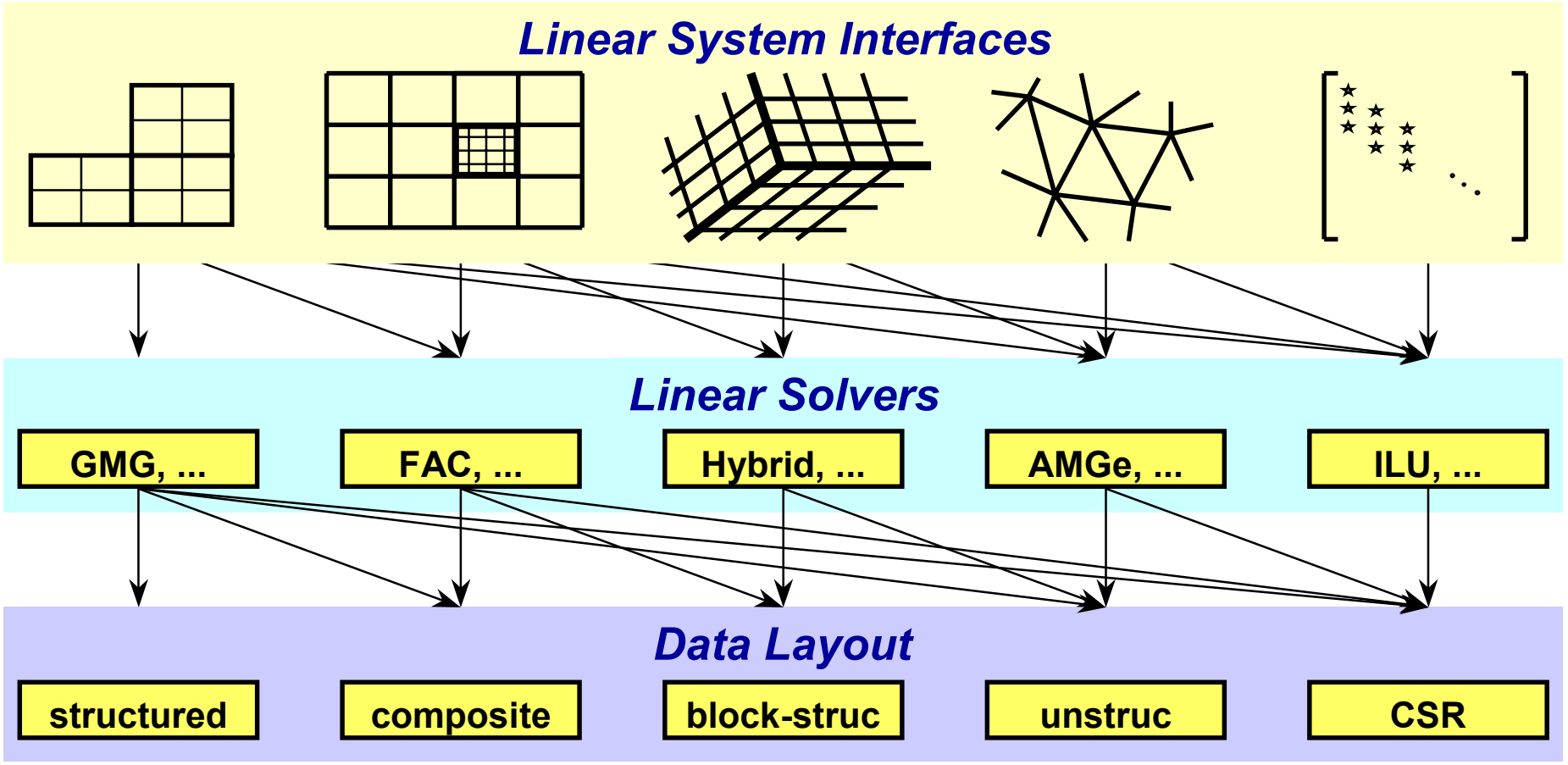
Hypre

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

Hypre: *Interfaces*

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

Solvers	System Interfaces			
	Struct	SStruct	FEI	IJ
Jacobi	X			
SMG	X			
PFMG	X			
BoomerAMG	X	X	X	X
ParaSails	X	X	X	X
PILUT	X	X	X	X
PCG	X	X	X	X
GMRES	X	X	X	X

Hypre: *Setup and Use of Solvers*

- Create the solver

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- Set parameters

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- Prepare to solve the system

```
HYPRE_SolverSetup(solver, A, b, x);
```

- Solve the system

```
HYPRE_SolverSolve(solver, A, b, x);
```

- Get solution info out via conceptual interface

```
HYPRE_StructVectorGetValues(struct_x, index, values);
```

- Destroy the solver

```
HYPRE_SolverDestroy(solver);
```

PETSc

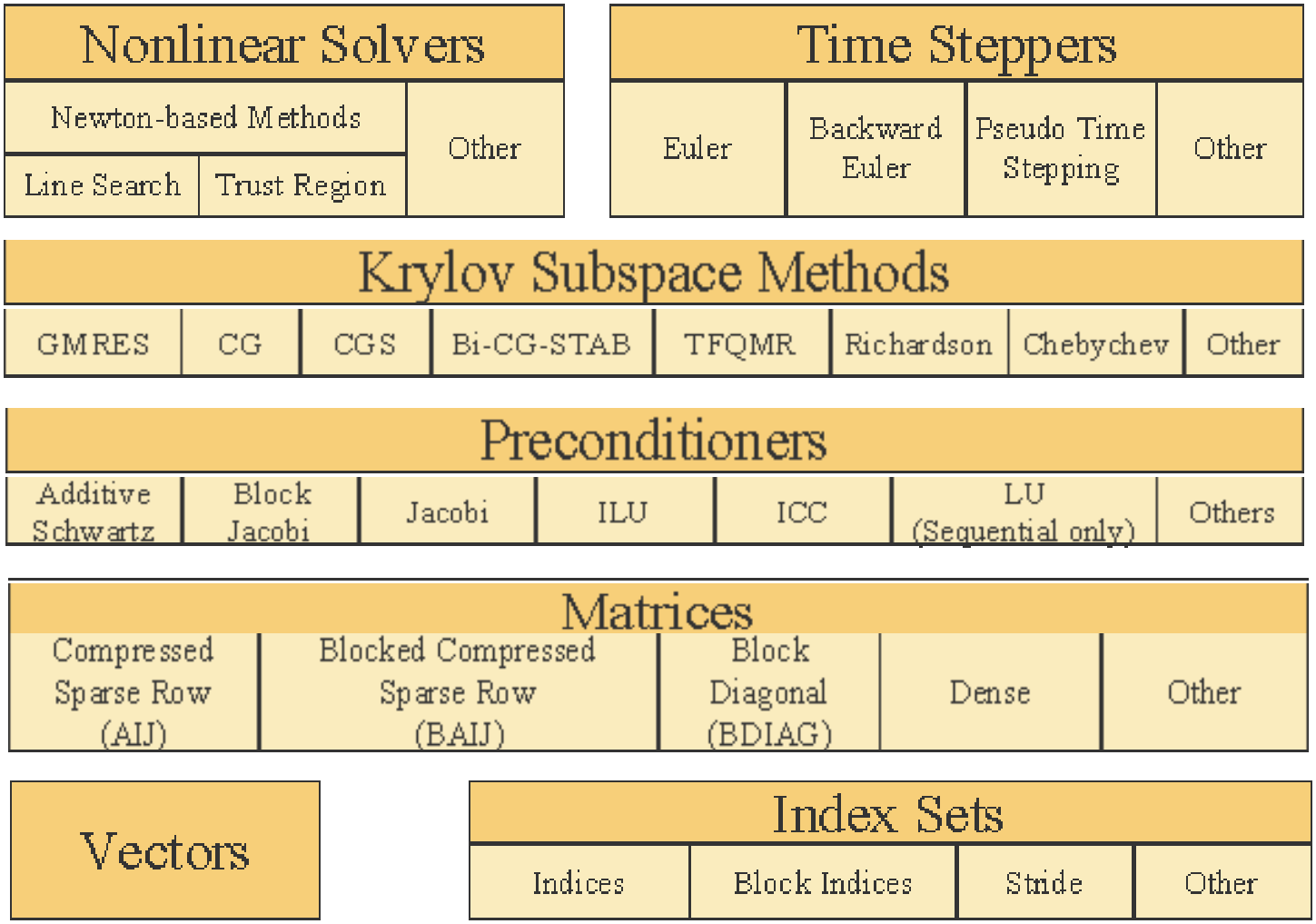
http://acts.nersc.gov/petsc

- 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*



PETSc: *simple example*

```
/*
See http://www-fp.mcs.anl.gov/petsc/docs/tutorials
*/
/* Program usage:  mpirun ex1 [-help] [all PETSc options] */
static char help[] = "This is an introductory PETSc example that illustrates
   printing.\n\n";
/*
   Concepts: Introduction to PETSc;
   Routines: PetscInitialize(); PetscPrintf(); PetscFinalize();
   Processors: n
*/
#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.)*

```
{
  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 = 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_SELF,"[%d] Jumbled Hello
          World\n",rank);CHKERRA(ierr);
  ierr = PetscFinalize();CHKERRA(ierr);
  return 0;
}
```

Every PETSc program should begin with the PetscInitialize routine.

Prints a single message

Prints a multiple message

A program must always end with PetscFinalize

PETSc SLES: *basic steps*

- Define the linear system ($Ax=b$)
  - **`MatCreate, MatSetValue, VecCreate`**
- Create the Solver
  - **`SLESCreate, SLESSetOperators`**
- Solve System of Equations
  - **`SLESSolve`**
- Clean up
  - **`SLESDestroy`**

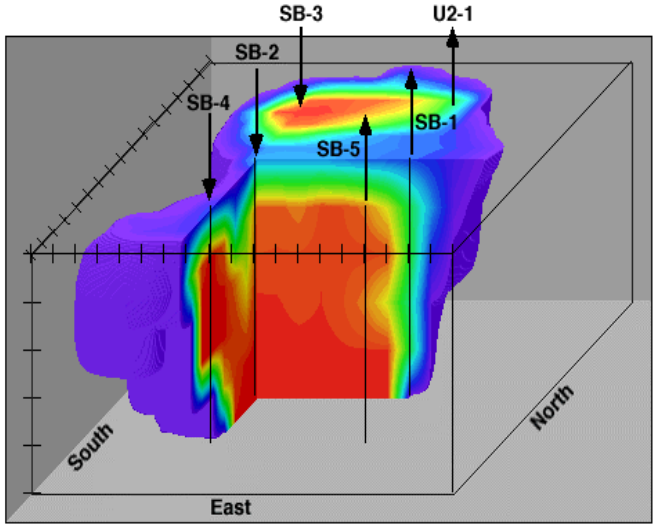
PETSc SNES: *basic steps*

- Non-linear equations of the form:
  - $F(x) = 0$
- Unconstrained Minimization problems of the form:
  - $Min\{f(x)\}$
- 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: **`SNESSolve`**
- Clean up

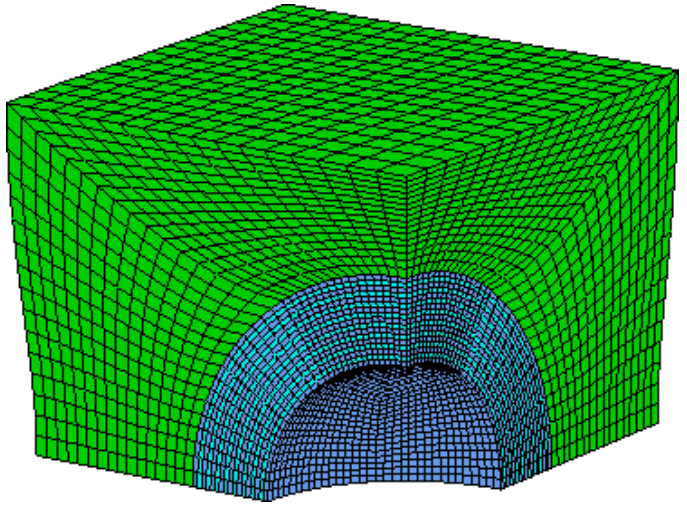
PETSc TS: *basic steps*

- 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 $Ax=\lambda x$

Bai, Demmel, Dongarra, Ruhe and van der Vorst, Editors, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

- 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

Method	Application	Orthogonalization	Eigenvectors			Number of vectors	Factorization
			a few at the end	several at the end	in the middle		
Power method	direct shift-invert	- -	yes -	very slow yes	no yes	2 2	- LU
Subspace iteration	direct shift invert	full full	yes -	slow yes	no yes	moderate moderate	- LU
Lanczos	direct direct shift invert	local selective full	yes yes -	no slow yes	no no yes	3 many moderate	- - LU
Lanczos with IR	direct shift invert	full full	yes -	slow yes	no yes	few fewer	- LU
Block Lanczos	direct shift invert	full full	yes -	yes yes	no yes	many moderate	- LU
Jacobi-Davidson	direct preconditioner shift invert	full full full	slow yes -	slow yes yes	no slow yes	few few few	- ILU LU

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 run-time.
    - Not all algorithms semantically or mathematically equivalent

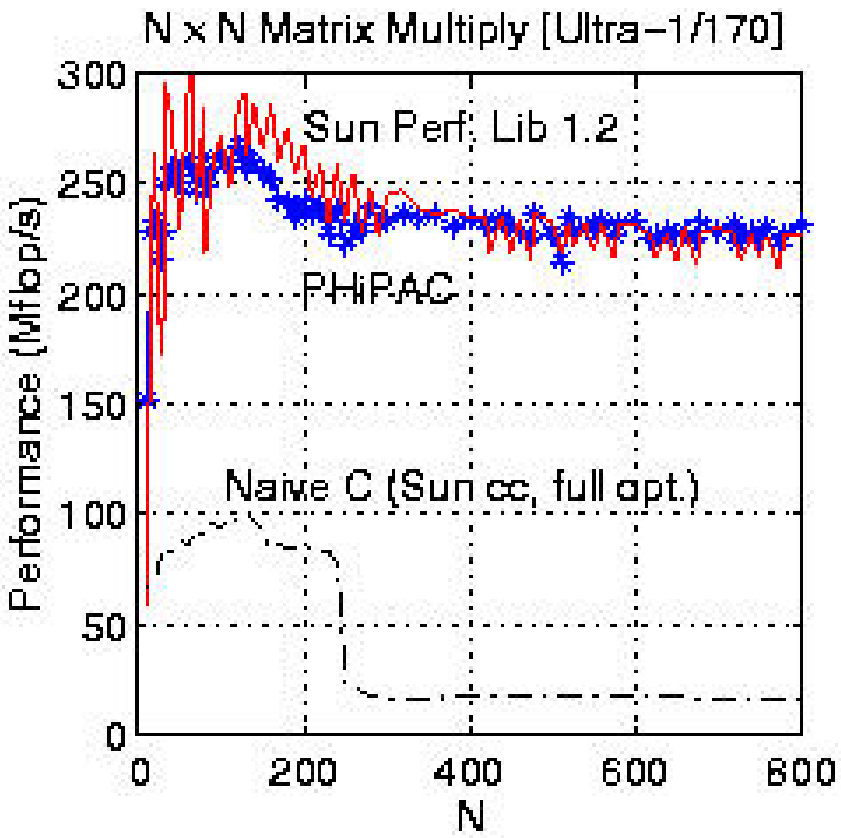
Automatic Performance Tuning

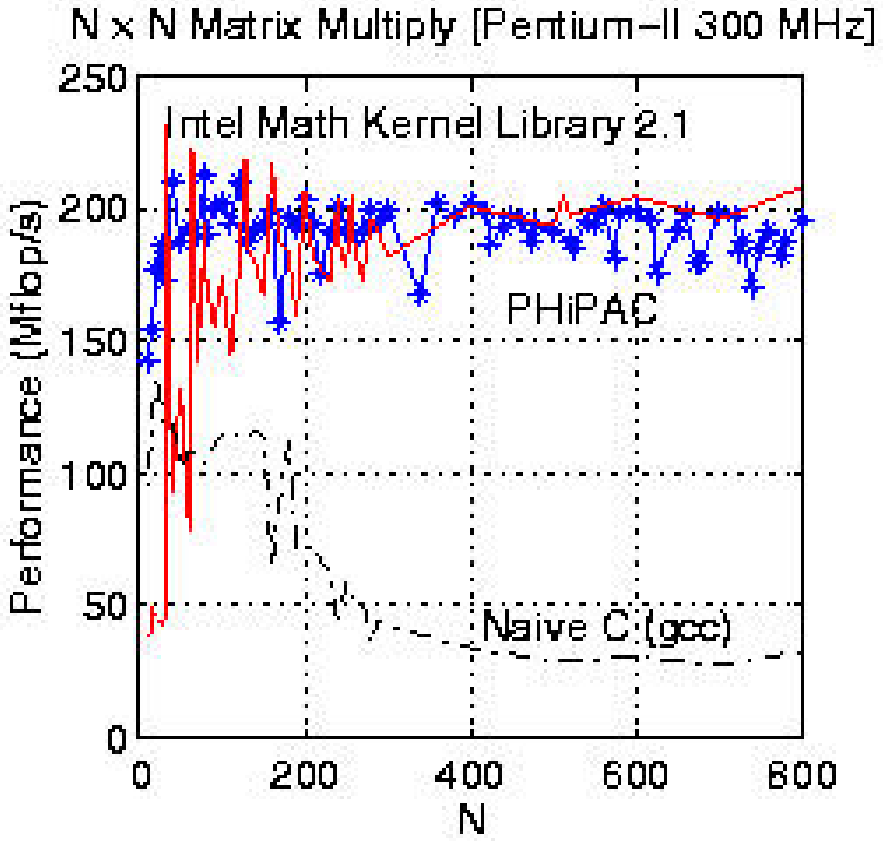
- Approach: for each kernel
  1. Identify and generate a space of algorithms
  2. Search for the fastest one, by running them
- What is a space of algorithms?
  - Depending on kernel and input, may vary
    - instruction mix and order
    - memory access patterns
    - data structures
    - mathematical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

Some Automatic Tuning Projects

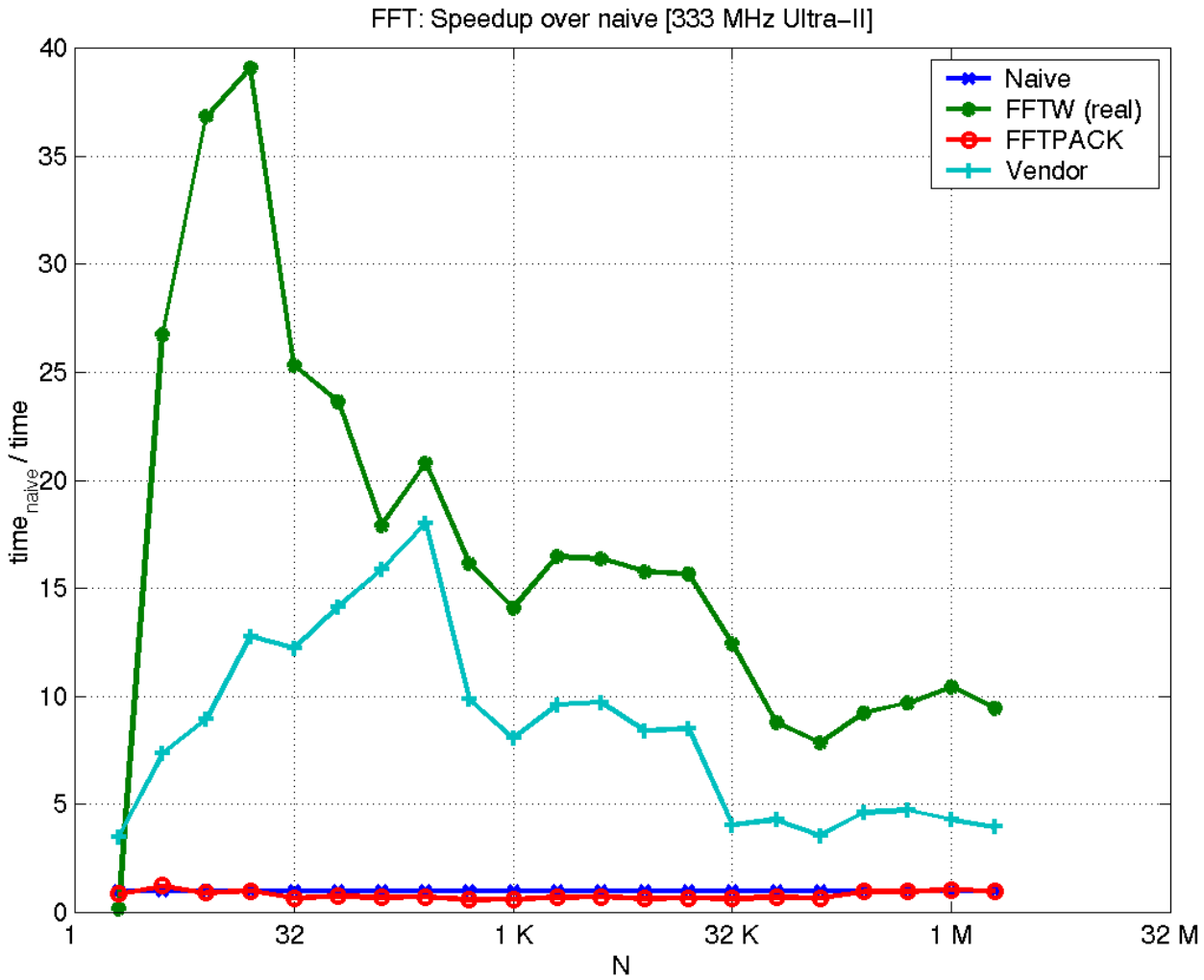
- PHiPAC: *www.icsi.berkeley.edu/~bilmes/phipac*
- ATLAS: *www.netlib.org/atlas*
- XBLAS: *www.nersc.gov/~xiaoye/XBLAS*
- Sparsity: *www.cs.berkeley.edu/~yelick/sparsity*
- FFTs and Signal Processing
  - FFTW: *www.fftw.org*
    - Won 1999 Wilkinson Prize for Numerical Software
  - SPIRAL: *www.ece.cmu.edu/~spiral*
    - 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

http://acts.nersc.gov

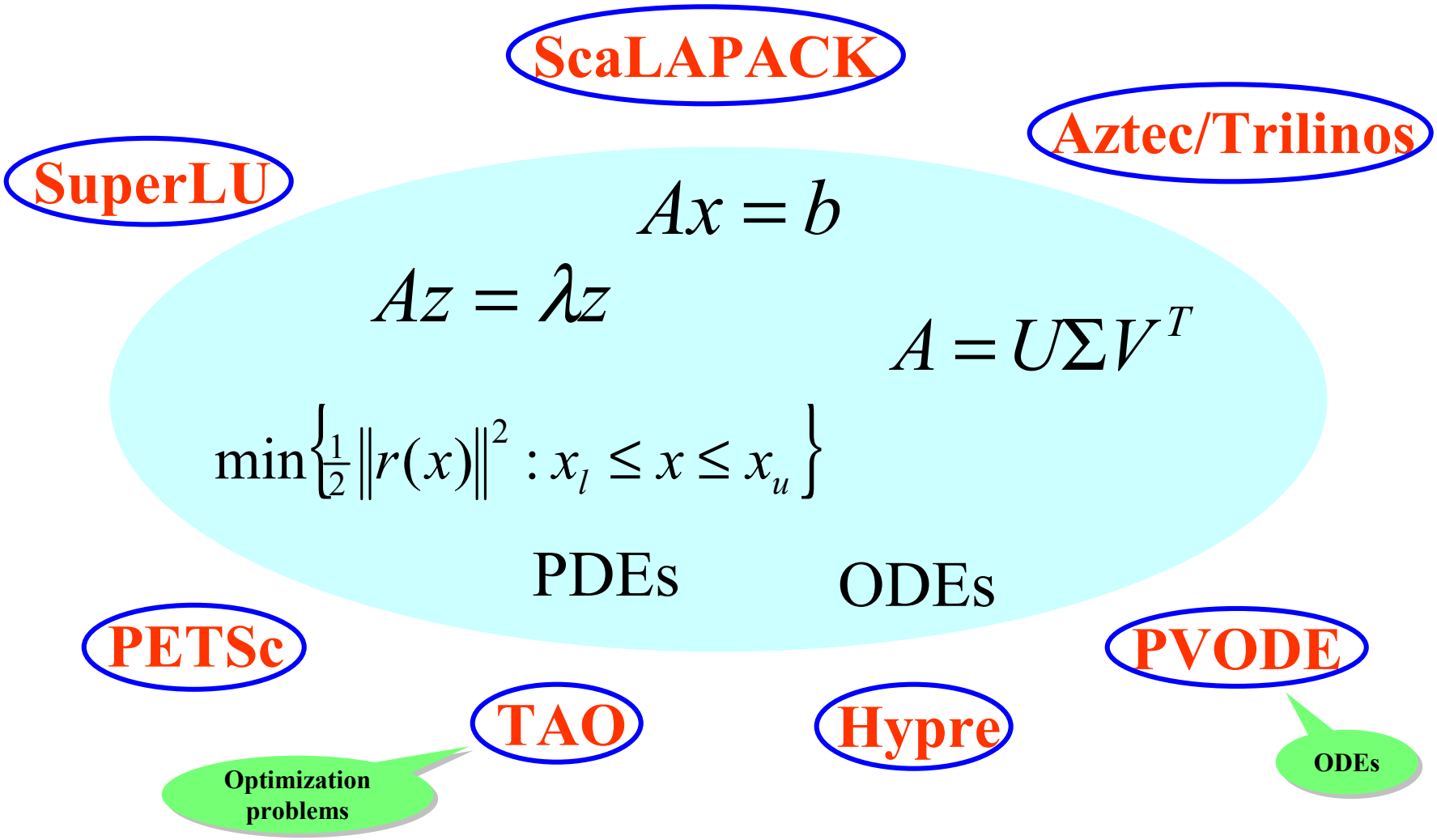
- Advanced Computational Testing and Simulation
- Tools for developing parallel applications
  - developed (primarily) at DOE Labs
  - separate projects originally
  - 20 tools
- ACTS is an "umbrella" project
  - leverage numerous independently funded projects
  - collect tools in a toolkit

ACTS: *Project Goals*

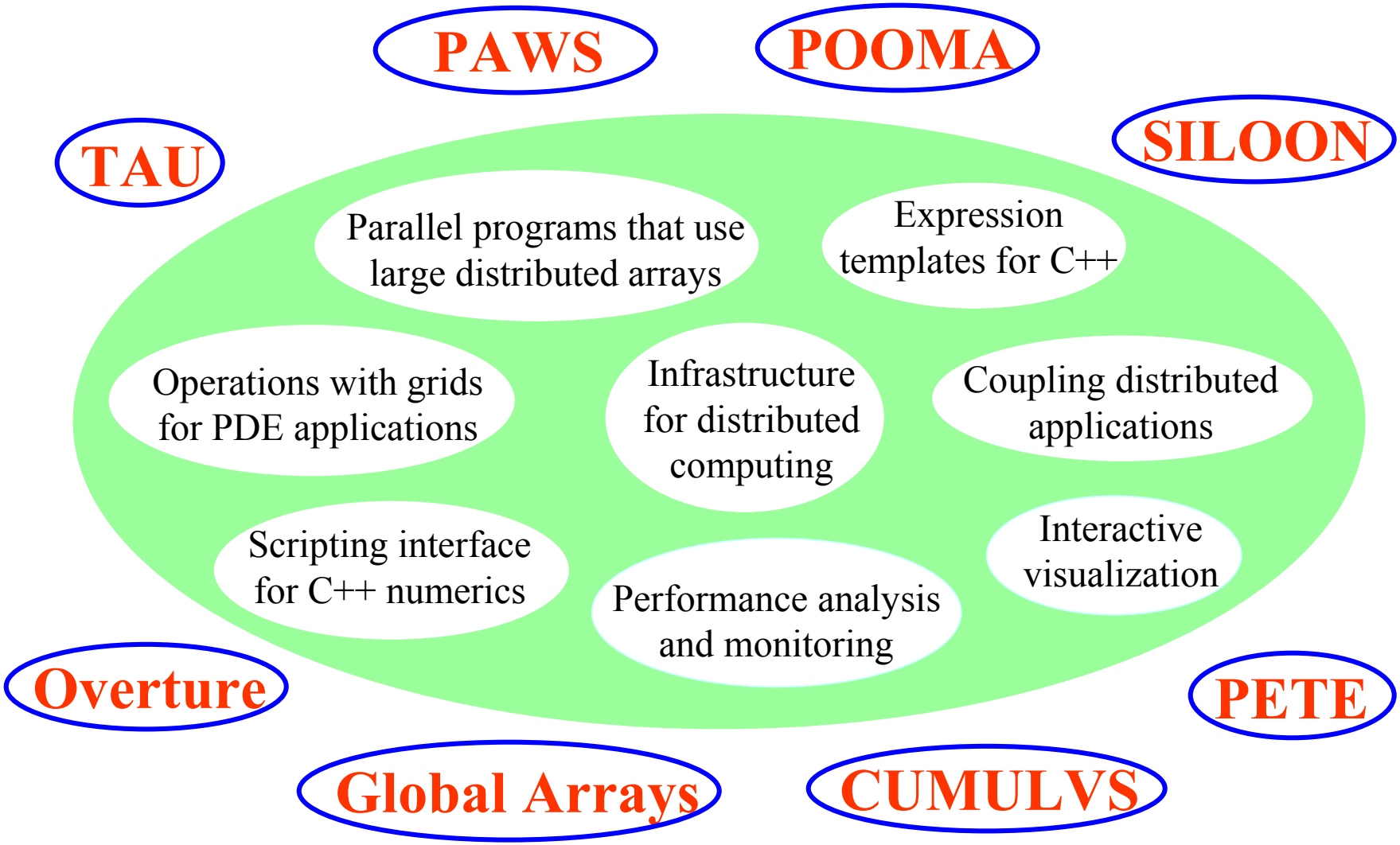
acts-support@nersc.gov ***http://acts.nersc.gov***

- 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?



What codes are being developed?



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*
- NPACI All-Hands Meeting, San Diego, March 6, 2002, by Jim Demmel, Tony Drummond and Osni Marques, *http://acts.nersc.gov/presentations/AH2002*
  - 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
- *Numerical Linear Algebra for High-Performance Computers*, Dongarra, Duff, Sorensen and van der Vorst, SIAM 1998.