Building a Reliable Software Infrastructure for Scientific Computing

Osni Marques
Lawrence Berkeley National Laboratory (LBNL)
oamarques@lbl.gov
Outline

- Keeping the pace with the software and hardware
  - Hardware evolution
  - Performance tuning
  - Software selection
  - What is missing?
- The DOE ACTS Collection Project
  - Goals
  - Related activities
  - Current features
  - Lessons learned
High Performance Computers (Sustainable Performance)

- ~ 20 years ago → $1 \times 10^6$ Floating Point Ops/sec (Mflop/s)
  - Scalar based
- ~ 10 years ago → $1 \times 10^9$ Floating Point Ops/sec (Gflop/s)
  - Vector & Shared memory computing, bandwidth aware
  - Block partitioned, latency tolerant
- ~ Today → $1 \times 10^{12}$ Floating Point Ops/sec (Tflop/s)
  - Highly parallel, distributed processing, message passing, network based
  - data decomposition, communication/computation
- ~ 10 years away → $1 \times 10^{15}$ Floating Point Ops/sec (Pflop/s)
  - Many more levels of memory hierarchy, combination of grids&HPC
  - More adaptive, latency and bandwidth aware, fault tolerant, extended precision, attention to SMP nodes
Automatic Tuning

• 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

• 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!

Example: $PHiPAC \Rightarrow C = A \ast B$
What About Software Selection?

Example: \( Ax = b \)

- Use a direct solver \( (A=LU) \) 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

Build 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, etc). See *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.*
Bugs…

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people. The problem was an inaccurate calculation of the time since boot due to computer arithmetic errors.

On June 4, 1996, an Ariane 5 rocket launched by the European Space Agency exploded just forty seconds after its lift-off from Kourou, French Guiana. The rocket was on its first voyage, after a decade of development costing $7 billion. The problem was a software error in the inertial reference system. Specifically a 64 bit floating point number relating to the horizontal velocity of the rocket with respect to the platform was converted to a 16 bit signed integer.

On August 23, 1991, the first concrete base structure for the Sleipner A platform sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway. The post accident investigation traced the error to inaccurate finite element approximation of the linear elastic model of the tricell (using the popular finite element program NASTRAN). The shear stresses were underestimated by 47% leading to insufficient design. In particular, certain concrete walls were not thick enough.

http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html
Challenges in the Development of Scientific Codes

- **Productivity**
  - Time to the first solution (prototype)
  - Time to solution (production)
  - Other requirements

- **Complexity**
  - Increasingly sophisticated models
  - Model coupling
  - Interdisciplinarity

- **Performance**
  - Increasingly complex algorithms
  - Increasingly complex architectures
  - Increasingly demanding applications

- Libraries written in different languages.
- Discussions about standardizing interfaces are often sidetracked into implementation issues.
- Difficulties managing multiple libraries developed by third-parties.
- Need to use more than one language in one application.
- The code is long-lived and different pieces evolve at different rates.
- Swapping competing implementations of the same idea and testing without modifying the code.
- Need to compose an application with some other(s) that were not originally designed to be combined.
Components: *simple example*

**Numerical integration: midpoint**
\[ \int_a^b f(x)\,dx \approx \frac{b-a}{n} \sum_{j=1}^{n} f\left(\frac{x_{j-1} + x_j}{2}\right) \]

**Numerical integration: Monte Carlo**
\[ \int_a^b f(x)\,dx \approx \frac{1}{b-a} \left( \frac{1}{N} \sum_{i=1}^{N} f(x_i) \right) \]

\[ f_1(x) = x^2 \]
\[ f_2(x) = 2x \]
\[ f_3(x) = \frac{4}{1 + x^2} \]
The development of complex simulation codes on parallel computers is not a trivial task.

Usually, a significant percentage of the efforts focus on the development of the codes and their optimization.

There is a need for a collaboration framework for ongoing development and deployment of computational tools.

In 1999, the PITAC Report recommended the creation of a national library of certified domain-specific software in order to reduce the labor required for software development, testing and evolution.

Research in computational sciences is fundamentally interdisciplinary and addresses, among many others, climate and environment modeling, DNA sequencing, flows in geological structures, etc.
The DOE ACTS Collection

Goals

- Collection of tools for developing parallel applications
- Extended support for experimental software
- Make ACTS tools available on DOE computers
- Provide technical support (acts-support@nersc.gov)
- Maintain ACTS information center (http://acts.nersc.gov)
- Coordinate efforts with other supercomputing centers
- Enable large scale scientific applications
- Educate and train

High Performance Tools
- portable
- library calls
- robust algorithms
- help code optimization
- More code development in less time
- More simulation in less computer time

Levels of Support

- **High**
  - Intermediate level
  - Tool expertise
  - Conduct tutorials

- **Intermediate**
  - Basic level
  - Higher level of support to users of the tool

- **Basic**
  - Help with installation
  - Basic knowledge of the tools
  - Compilation of user’s reports

User Community

- Engineering
- Numerical Simulations
- Mathematics
- Computer Science
- Collaboration
- Bioinformatics

Pool of Software Tools

- Testing and Acceptance Phase
- Interoperability

Scientific Computing Centers

- Workshops and Training
- Computer Vendors

Office of Science

UC Berkeley - CS267 03/31/2004 12
### ACTS Tools Functionalities

<table>
<thead>
<tr>
<th>Category</th>
<th>Tool</th>
<th>Functionalities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Numerical</strong></td>
<td>Aztec</td>
<td>Algorithms for the iterative solution of large sparse linear systems.</td>
</tr>
<tr>
<td></td>
<td>Hypre</td>
<td>Algorithms for the iterative solution of large sparse linear systems, intuitive grid-centric interfaces, and dynamic configuration of parameters.</td>
</tr>
<tr>
<td></td>
<td>PETSc</td>
<td>Tools for the solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations.</td>
</tr>
<tr>
<td>PDEs ODEs M</td>
<td>OPT++</td>
<td>Object-oriented nonlinear optimization package.</td>
</tr>
<tr>
<td></td>
<td>SUNDIALS</td>
<td>Solvers for the solution of systems of ordinary differential equations, nonlinear algebraic equations, and differential-algebraic equations.</td>
</tr>
<tr>
<td></td>
<td>ScaLAPACK</td>
<td>Library of high performance dense linear algebra routines for distributed-memory message-passing.</td>
</tr>
<tr>
<td></td>
<td>SuperLU</td>
<td>General-purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations.</td>
</tr>
<tr>
<td></td>
<td>TAO</td>
<td>Large-scale optimization software, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization.</td>
</tr>
<tr>
<td><strong>Code Development</strong></td>
<td>Global Arrays</td>
<td>Library for writing parallel programs that use large arrays distributed across processing nodes and that offers a shared-memory view of distributed arrays.</td>
</tr>
<tr>
<td></td>
<td>Overture</td>
<td>Object-Oriented tools for solving computational fluid dynamics and combustion problems in complex geometries.</td>
</tr>
<tr>
<td><strong>Code Execution</strong></td>
<td>CUMULVS</td>
<td>Framework that enables programmers to incorporate fault-tolerance, interactive visualization and computational steering into existing parallel programs.</td>
</tr>
<tr>
<td></td>
<td>Globus</td>
<td>Services for the creation of computational Grids and tools with which applications can be developed to access the Grid.</td>
</tr>
<tr>
<td></td>
<td>PAWS</td>
<td>Framework for coupling parallel applications within a component-like model.</td>
</tr>
<tr>
<td></td>
<td>SILOON</td>
<td>Tools and run-time support for building easy-to-use external interfaces to existing numerical codes.</td>
</tr>
<tr>
<td></td>
<td>TAU</td>
<td>Set of tools for analyzing the performance of C, C++, Fortran and Java programs.</td>
</tr>
<tr>
<td><strong>Library Development</strong></td>
<td>ATLAS and PHiPAC</td>
<td>Tools for the automatic generation of optimized numerical software for modern computer architectures and compilers.</td>
</tr>
<tr>
<td></td>
<td>PETE</td>
<td>Extensible implementation of the expression template technique (C++ technique for passing expressions as function arguments).</td>
</tr>
</tbody>
</table>
Use of ACTS Tools

- Finite Differences
- Finite Elements
- Boundary Elements
- Fourier

Continuous problem

Discrete problem

\[ m \frac{d^2 x}{d t^2} + c \frac{dx}{dt} + kx = f(t) \]

\[ \frac{\partial v}{\partial t} = -v \frac{\partial v}{\partial x} + \Gamma \frac{\partial^2 v}{\partial x^2} M \]

(Multiphase flow using PETSc, 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).)

- Model of a "hard" sphere included in a "soft" material, 26 million d.o.f. (Adams and Demmel, Prometheus and PETSc, unstructured meshes in solid mechanics).
- 3D overlapping grid for a submarine produced with Overture's module ogen.

UC Berkeley - CS267

03/31/2004
Use of ACTS Tools

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), eigenvalue problems solved with ScaLAPACK.

Two ScaLAPACK routines, PZGETRF and PZGETRS, are used for solution of linear systems in the spectral algorithms based AORSA code (Batchelor et al.), which is intended for the study of electromagnetic wave-plasma interactions. The code reaches 68% of peak performance on 1936 processors of an IBM SP.

Omega3P is a parallel distributed-memory code intended for the modeling and analysis of accelerator cavities, which requires the solution of generalized eigenvalue problems. A parallel exact shift-invert eigensolver based on PARPACK and SuperLU has allowed for the solution of a problem of order 7.5 million with 304 million nonzeros. Finding 10 eigenvalues requires about 2.5 hours on 24 processors of an IBM SP.

OPT++ is used in protein energy minimization problems (shown here is protein T162 from CASP5, courtesy of Meza, Oliva et al.)
Cosmic Microwave Background (CMB) Analysis

- The statistics of the tiny variations in the CMB (the faint echo of the Big Bang) allows the determination of the fundamental parameters of cosmology to the percent level or better.
- MADCAP (Microwave Anisotropy Dataset Computational Analysis Package)
  - Makes maps from observations of the CMB and then calculates their angular power spectra. (See http://crd.lbl.gov/~borrill).
  - Calculations are dominated by the solution of linear systems of the form $M = A^{-1}B$ for dense $n \times n$ matrices $A$ and $B$ scaling as $O(n^3)$ in flops. MADCAP uses ScaLAPACK for those calculations.
- On the NERSC Cray T3E (original code):
  - Cholesky factorization and triangular solve.
  - Typically reached 70-80% peak performance.
  - Solution of systems with $n \sim 10^4$ using tens of processors.
  - The results demonstrated that the Universe is spatially flat, comprising 70% dark energy, 25% dark matter, and only 5% ordinary matter.
- On the NERSC IBM SP:
  - Porting was trivial but tests showed only 20-30% peak performance.
  - Code rewritten to use triangular matrix inversion and triangular matrix multiplication → one-day work
  - Performance increased to 50-60% peak.
  - Solution of previously intractable systems with $n \sim 10^5$ using hundreds of processors.

The international BOOMERanG collaboration announced results of the most detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat (Apr. 27, 2000).
ScaLAPACK: software structure

http://acts.nersc.gov/scalapack

ScaLAPACK

Global

PBLAS

Local

LAPACK

BLACS

BLAS

MPI/PVM/...

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

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

Communication routines targeting linear algebra operations.

Communication layer (message passing).

Parallel BLAS.
BLAS

*(Basic Linear Algebra Subroutines)*

- **Clarity**: code is shorter and easier to read.
- **Modularity**: gives programmer larger building blocks.
- **Performance**: manufacturers (usually) provide tuned machine-specific BLAS.
- **Portability**: machine dependencies are confined to the BLAS.
- **Key to high performance**: effective use of memory hierarchy (true on all architectures).

Development of blocked algorithms (BLAS 3) is important for performance!

- Level 1 BLAS: vector-vector operations.
- Level 2 BLAS: matrix-vector operations.
- Level 3 BLAS: matrix-matrix operations.
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.

- Basic problems:
  - Linear systems: \( Ax = b \)
  - Least squares: \( \min \| Ax - b \|_2 \)
  - Singular value decomposition: \( A = U \Sigma V^T \)
  - Eigenvalues and eigenvectors: \( Az = \lambda z, \quad Az = \lambda Bz \)
- LAPACK does not provide routines for structured problems or general sparse matrices
BLACS

(Basic Linear Algebra Communication Subroutines)

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations. This improves:
  - program readability
  - self-documenting quality of the code.
- Promote efficiency by identifying frequently occurring operations of linear algebra which can be optimized on various computers.
BLACS: basics

- Processes are embedded in a two-dimensional grid.

```
  0 1 2 3
 0 0 1 2 3
 1 4 5 6 7
 2 8 9 10 11
```

Example: a 3x4 grid

- An operation which involves more than one sender and one receiver is called a **scoped operation**.

<table>
<thead>
<tr>
<th>Scope</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>All processes in a process row participate.</td>
</tr>
<tr>
<td>Column</td>
<td>All processes in a process column participate.</td>
</tr>
<tr>
<td>All</td>
<td>All processes in the process grid participate.</td>
</tr>
</tbody>
</table>
BLACS: example

* Get system information
  CALL BLACS_PINFO( IAM, NPROCS )
* If underlying system needs additional setup, do it now
  IF( NPROCS.LT.1 ) THEN
    IF( IAM.EQ.0 ) NPROCS = 4
    CALL BLACS_SETUP( IAM, NPROCS )
  END IF
* Get default system context
  CALL BLACS_GET( 0, 0, ICTXT )
M
* Define 1 x (NPROCS/2+1) process grid
  NPROW = 1
  NPCOL = NPROCS / 2 + 1
  CALL BLACS_GRIDINIT( ICTXT, 'Row', NPROW, NPCOL )
  CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
* If I’m not in the grid, go to end of program
  IF( MYROW.NE.-1 ) THEN
    IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
      CALL DGESD2D( ICTXT, 5, 1, X, 5, 1, 0 )
    ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
      CALL DGERV2D( ICTXT, 5, 1, Y, 5, 0, 0 )
    END IF
  END IF
M
  CALL BLACS_GRIDEXIT( ICTXT )
END IF
CALL BLACS_EXIT( 0 )
END

See http://www.netlib.org/blacs for more information.

- The BLACS context is the BLACS mechanism for partitioning communication space.
- A message in a context cannot be sent or received in another context.
- The context allows the user to
  - create arbitrary groups of processes
  - create multiple overlapping and/or disjoint grids
  - isolate each process grid so that grids do not interfere with each other
- BLACS context ⇔ MPI communicator
PBLAS

(Parallel Basic Linear Algebra Subroutines)

- Similar to the BLAS in portability, functionality and naming.
- Built atop the BLAS and BLACS
- Provide global view of matrix

CALL DGEXXX( M, N, A( IA, JA ), LDA, ... )  

CALL PDGEXXX( M, N, A, IA, JA, DESCA, ... )  

Array descriptor (see next slides)
PBLAS: *levels and view of the operands*

- **Levels:**
  - Level 1: vector-vector operations.
  - Level 2: matrix-vector operations.
  - Level 3: matrix-matrix operations.

- **Global view of the matrix operands,** allowing global addressing of distributed matrices (hiding complex local indexing)
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: data layouts

- 1D block and cyclic column distributions

- 1D block-cycle column and 2D block-cyclic distribution

- 2D block-cyclic used in ScaLAPACK for dense matrices
ScaLAPACK: 2D Block-Cyclic Distribution

5x5 matrix partitioned in 2x2 blocks

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

2x2 process grid point of view

\[
\begin{pmatrix}
  a_{11} & a_{12} & a_{15} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{25} & a_{23} & 1 & a_{24} \\
  a_{31} & a_{32} & a_{35} & a_{33} & 2 & a_{34} \\
  a_{41} & a_{42} & a_{45} & a_{43} & 3 & a_{44} \\
  a_{51} & a_{52} & a_{55} & a_{53} & a_{54}
\end{pmatrix}
\]
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

CALL PDGESVD( JOBU, JOBVT, M, N, A, IA, JA, DESCA, S, U, IU, 
                JU, DESCU, VT, IVT, JVT, DESCVT, WORK, LWORK, 
                INFO )

M

\[
\begin{bmatrix}
  1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\
  -2.1 & 2.2 & 2.3 & 2.4 & 2.5 \\
  -3.1 & -3.2 & 3.3 & 3.4 & 3.5 \\
  -4.1 & -4.2 & -4.3 & 4.4 & 4.5 \\
  -5.1 & -5.2 & -5.3 & -5.4 & 5.5 \\
\end{bmatrix}
\]
Each global data object is assigned an array descriptor.

The array descriptor:

- Contains information required to establish mapping between a global array entry and its corresponding process and memory location (uses concept of BLACS context).
- Is differentiated by the DTYPE_ (first entry) in the descriptor.
- Provides a flexible framework to easily specify additional data distributions or matrix types.

User must distribute all global arrays prior to the invocation of a ScaLAPACK routine, for example:

- Each process generates its own submatrix.
- One processor reads the matrix from a file and send pieces to other processors (may require message-passing for this).
## Array Descriptor for Dense Matrices

<table>
<thead>
<tr>
<th>DESC_()</th>
<th>Symbolic Name</th>
<th>Scope</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DTYPE_A</td>
<td>(global)</td>
<td>Descriptor type DTYPE_A=1 for dense matrices.</td>
</tr>
<tr>
<td>2</td>
<td>CTXT_A</td>
<td>(global)</td>
<td>BLACS context handle.</td>
</tr>
<tr>
<td>3</td>
<td>M_A</td>
<td>(global)</td>
<td>Number of rows in global array A.</td>
</tr>
<tr>
<td>4</td>
<td>N_A</td>
<td>(global)</td>
<td>Number of columns in global array A.</td>
</tr>
<tr>
<td>5</td>
<td>MB_A</td>
<td>(global)</td>
<td>Blocking factor used to distribute the rows of array A.</td>
</tr>
<tr>
<td>6</td>
<td>NB_A</td>
<td>(global)</td>
<td>Blocking factor used to distribute the columns of array A.</td>
</tr>
<tr>
<td>7</td>
<td>RSRC_A</td>
<td>(global)</td>
<td>Process row over which the first row of the array A is distributed.</td>
</tr>
<tr>
<td>8</td>
<td>CSRC_A</td>
<td>(global)</td>
<td>Process column over which the first column of the array A is distributed.</td>
</tr>
<tr>
<td>9</td>
<td>LLD_A</td>
<td>(local)</td>
<td>Leading dimension of the local array.</td>
</tr>
</tbody>
</table>
## Array Descriptor for Narrow Band Matrices

<table>
<thead>
<tr>
<th>DESC_()</th>
<th>Symbolic Name</th>
<th>Scope</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DTYPE_A</td>
<td>(global)</td>
<td>Descriptor type DTYPE_A=501 for 1 x Pc process grid for band and tridiagonal matrices block-column distributed.</td>
</tr>
<tr>
<td>2</td>
<td>CTXT_A</td>
<td>(global)</td>
<td>BLACS context handle.</td>
</tr>
<tr>
<td>3</td>
<td>N_A</td>
<td>(global)</td>
<td>Number of columns in global array A.</td>
</tr>
<tr>
<td>4</td>
<td>NB_A</td>
<td>(global)</td>
<td>Blocking factor used to distribute the columns of array A.</td>
</tr>
<tr>
<td>5</td>
<td>CSRC_A</td>
<td>(global)</td>
<td>Process column over which the first column of the array A is distributed.</td>
</tr>
<tr>
<td>6</td>
<td>LLD_A</td>
<td>(local)</td>
<td>Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored. Unused, reserved.</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Array Descriptor for Right Hand Sides for Narrow Band Linear Solvers

<table>
<thead>
<tr>
<th>DESC(_)</th>
<th>Symbolic Name</th>
<th>Scope</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DTYPE_B</td>
<td>(global)</td>
<td>Descriptor type DTYPE_B=502 for Pr x 1 process grid for block-row distributed matrices.</td>
</tr>
<tr>
<td>2</td>
<td>CTXT_B</td>
<td>(global)</td>
<td>BLACS context handle.</td>
</tr>
<tr>
<td>3</td>
<td>M_B</td>
<td>(global)</td>
<td>Number of rows in global array B</td>
</tr>
<tr>
<td>4</td>
<td>MB_B</td>
<td>(global)</td>
<td>Blocking factor used to distribute the rows of array B.</td>
</tr>
<tr>
<td>5</td>
<td>RSRC_B</td>
<td>(global)</td>
<td>Process row over which the first row of the array B is distributed.</td>
</tr>
<tr>
<td>6</td>
<td>LLD_B</td>
<td>(local)</td>
<td>Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored. Unused, reserved.</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
## ScaLAPACK: Functionality

<table>
<thead>
<tr>
<th>( Ax = b )</th>
<th>Simple Driver</th>
<th>Expert Driver</th>
<th>Factor</th>
<th>Solve</th>
<th>Inversion</th>
<th>Conditioning Estimator</th>
<th>Iterative Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td></td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>SPD</td>
<td>( \times )</td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>SPD Banded</td>
<td>( \times )</td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td>( \times )</td>
</tr>
<tr>
<td>SPD Tridiagonal</td>
<td>( \times )</td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td>( \times )</td>
</tr>
<tr>
<td>General</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>General Banded</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td>( \times )</td>
</tr>
<tr>
<td>General Tridiagonal</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Least Squares</td>
<td></td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td>( \times )</td>
</tr>
<tr>
<td>GQR</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRQ</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( Ax = \lambda x ) or ( Ax = Bx )</th>
<th>Simple Driver</th>
<th>Expert Driver</th>
<th>Reduction</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>General</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Generalized BSPD</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>SVD</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
</tbody>
</table>
The algorithms implemented in ScaLAPACK are scalable in the sense that the parallel efficiency is an increasing function of $N^2/P$ (problem size per node).

Maintaining memory use per node constant allows efficiency to be maintained (in practice, a slight degradation is acceptable).

Use efficient machine-specific BLAS (not the Fortran 77 source code available in http://www.netlib.gov) and BLACS (nondebug installation).

On a distributed-memory computer:

- Use the right number of processors
  - Rule of thumb: $P=MxN/1,000,000$ for an $MxN$ matrix, which 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 the physical memory.
- Use an efficient data distribution.
  - Block size (i.e., $MB,NB$) = 64.
  - Square processor grid: $Prow = Pcolumn$. 
What about tuning and performance analysis?

TAU

- Profiling of Java, C++, C, and Fortran codes
- Detailed information (much more than prof/gprof)
- Profiles for each unique template instantiation
- Time spent exclusively and inclusively in each function
- Start/Stop timers
- Profiling data maintained for each thread, context, and node
- Parallel IO Statistics for the number of calls for each profiled function
- Profiling groups for organizing and controlling instrumentation
- Support for using CPU hardware counters (PAPI)
- Graphic display for parallel profiling data
- Graphical display of profiling results (built-in viewers, interface to Vampir)

- COSY: COrpile manager Status display
- FANCY: File ANd Class display
- SPIFFY: Structured Programming Interface and Fancy File display
- CAGEY: CALL Graph Extended display
- CLASSY: CLASS hierarchy browser
- RACY: Routine and data ACcess profile display
- SPEEDY: Speedup and Parallel Execution Extrapolation Display
PROGRAM PSGESVDRIVER
! Example Program solving Ax=b via ScaLAPACK routine PSGESV
! ! Parameters ...

***** a bunch of things omitted for the sake of space *****
! ! Executable Statements ...
! ! INITIALIZE THE PROCESS GRID
!
integer profiler(2)
save profiler

call TAU_PROFILE_INIT()
call TAU_PROFILE_TIMER(profiler,'PSGESVDRIVER')
call TAU_PROFILE_START(profiler)
CALL SL_INIT( ICTXT, NPROW, NPCOL )
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

***** a bunch of things omitted for the sake of space *****
CALL PSGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, & INFO )

***** a bunch of things omitted for the sake of space *****
call TAU_PROFILE_STOP(profiler)
STOP
END

NB. ScaLAPACK routines have not been instrumented and therefore are not shown in the charts.
TAU: Example 2

- **EVH1 (Enhanced Virginia Hydrodynamics #1) benchmark**
- **MPI code developed from VH1, based on the piece-wise parabolic method (PPM) of Colella and Woodward**
- **PPM is a technique used for compressible, non-turbulent hydrodynamics. It has been used in a variety of astrophysical contexts, in addition to some ideal gas computations and studies of convection**

Visualizing TAU traces with Vampir, a commercial trace visualization tool from Pallas, GmbH.

JRACY, exclusive and inclusive level 1 data cache misses for all routines (except PARABOLA), mean over 16 processors.
Why is ACTS unique?

• Provides pointers and documentation about software tools.
• Accumulates the expertise and user feedback on the use of the software tools and scientific applications that used them:
  • independent software evaluations
  • participation in the developer user groups e-mail list
  • presentation of a gallery of applications
  • leverage between tool developers and tool users
  • workshops and tutorials
  • tool classification
  • support
Related Activities

- Software Repositories:
  - Netlib: http://www.netlib.org
  - HPC-Netlib: http://www.nhse.org/hpc-netlib
  - MGNet: http://www.mgnet.org
  - OO Numerics: http://oonumerics.org/oon

- Portable timing routines, tools for debugging, compiler technologies:
  - Ptools: http://www.ptools.org
  - Center for Programming Models for Scalable Parallel Computing: http://www.pmodels.org

- Education:
  - Computational Science Educational Project: http://csep1.phy.ornl.gov
  - UCB’s Applications of Parallel Computers: http://www.cs.berkeley.edu/~demmel/cs267_Spr99
  - Dictionary of algorithms, data structures and related definitions: http://www.nist.gov/dads
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 Collection should be seen as dynamical 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 making the tools mature.
- 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.
Why do we need these tools?

High Performance Tools
- portable
- library calls
- robust algorithms
- help code optimization

More code development in less time!

More simulation in less computer time!

A computation that took 1 full year to complete in 1980 could be done in ~ 10 hours in 1992, in ~ 16 minutes in 1997 and in ~ 27 seconds in 2001!
Who Benefits from these tools?

<table>
<thead>
<tr>
<th>Application</th>
<th>Computational Problem</th>
<th>Software Tools</th>
<th>Highlights</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADCAP</td>
<td>Matrix factorization and triangular solves</td>
<td>ScaLAPACK</td>
<td>• 50% peak performance on an IBM SP&lt;br&gt;• Nearly perfect scalability on 1024, 2048, 3072 and 4096 processors&lt;br&gt;• Fast implementation of numerical algorithms</td>
</tr>
<tr>
<td>3-Charged Particles</td>
<td>Solution of large, complex unsymmetric linear systems</td>
<td>SuperLU</td>
<td>• Solves systems of equations of order 8.4 million on 64 processors in 1 hour of wall clock time, 30 GFLOPs.</td>
</tr>
<tr>
<td>NWChem</td>
<td>Distribute large data arrays, collective operations</td>
<td>Global Arrays and LAPACK</td>
<td>• Very good scaling for large problems</td>
</tr>
</tbody>
</table>

http://acts.nersc.gov/AppMat

Enabling sciences and discoveries… with high performance and scalability…

... More Applications …
Scientific Computing – Third Pillar of Science

"a new way of doing science"

Many SC programs need dramatic advances in simulation capabilities to meet their mission goals.

- Health Effects, Bioremediation
- Combustion
- Components of Matter
- Subsurface Transport
- Fusion Energy
- Materials
- Global Climate
Challenges in the Development of Scientific Codes

- **Productivity**
  - Time to the first solution (prototype)
  - Time to solution (production)
  - Other requirements

- **Complexity**
  - Increasingly sophisticated models
  - Interdisciplinarity
  - Model coupling

- **Performance**
  - Increasingly complex algorithms
  - Increasingly complex architectures
  - Increasingly demanding applications

- Libraries written in different languages.
- Different pieces of the code evolve at different rates
- Swapping competing implementations of the same idea and testing without modifying the code

Peak performance is skyrocketing
- *In 1990s, peak performance increased 100x; in 2000s, it will increase 1000x*

However
- *Efficiency for many science applications declined from 40-50% on the vector supercomputers of 1990s to as little as 5-10% on parallel supercomputers of today*
The DOE ACTS Collection

High Performance Tools
- portable
- library calls
- robust algorithms
- help code optimization

Scientific Computing Centers like NERSC:
- Reduce user’s code development time that sums up in more production runs and faster and effective scientific research results
- Overall better system utilization
- Facilitate the accumulation and distribution of high performance computing expertise
- Provide better scientific parameters for procurement and characterization of specific user needs

Tool descriptions, installation details, examples, etc
Agenda, accomplishments, conferences, releases, etc
Goals and other relevant information
Points of contact
Search engine