

UC Berkeley - CS267

- ACTS -
**A Reliable Software Infrastructure
for Scientific Computing**

Osni Marques

Lawrence Berkeley National Laboratory (LBNL)

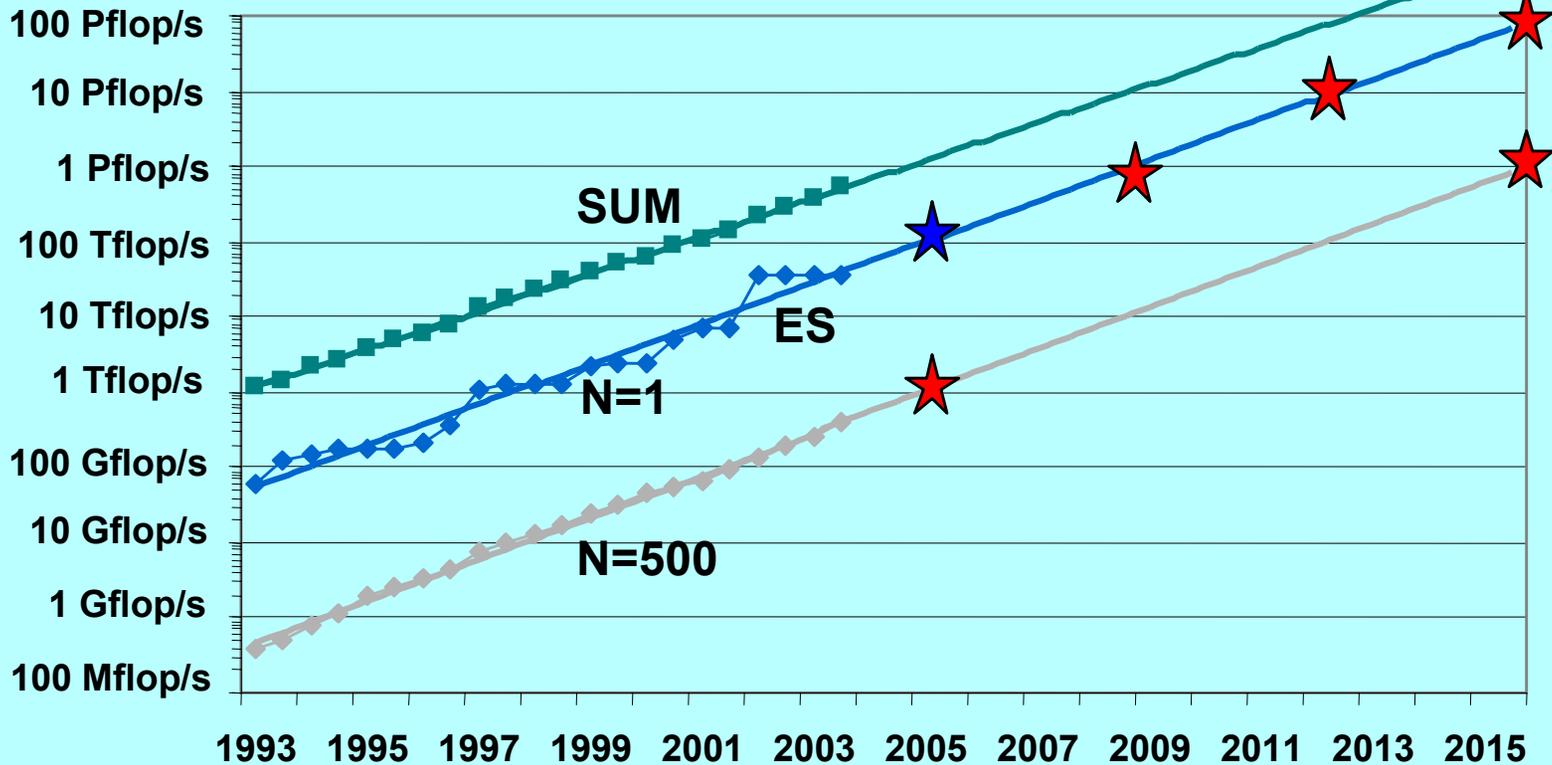
oamarques@lbl.gov

Outline

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



Projected Performance Development



Courtesy of Erich Strohmaier
<http://www.top500.org>



Automatic Tuning

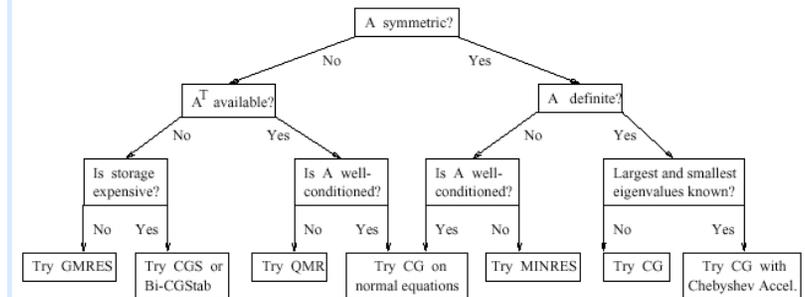
- For each kernel
 1. Identify and generate a space of algorithms
 2. Search for the fastest one, by running them
 - What is a space of algorithms?
 - Depending on kernel and input, may vary
 - instruction mix and order
 - memory access patterns
 - data structures
 - mathematical formulation
 - When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

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.

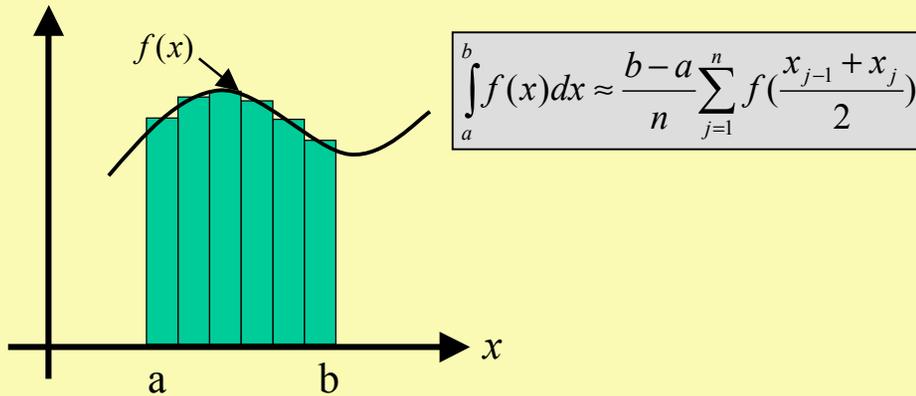


On August 23, 1991, the first concrete base structure for the Sleipner A platform sprang a leak and sank under a controlled ballasting operation during preparation for deck mating in Gandsfjorden outside Stavanger, Norway. The post accident investigation traced the error to inaccurate finite element 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.

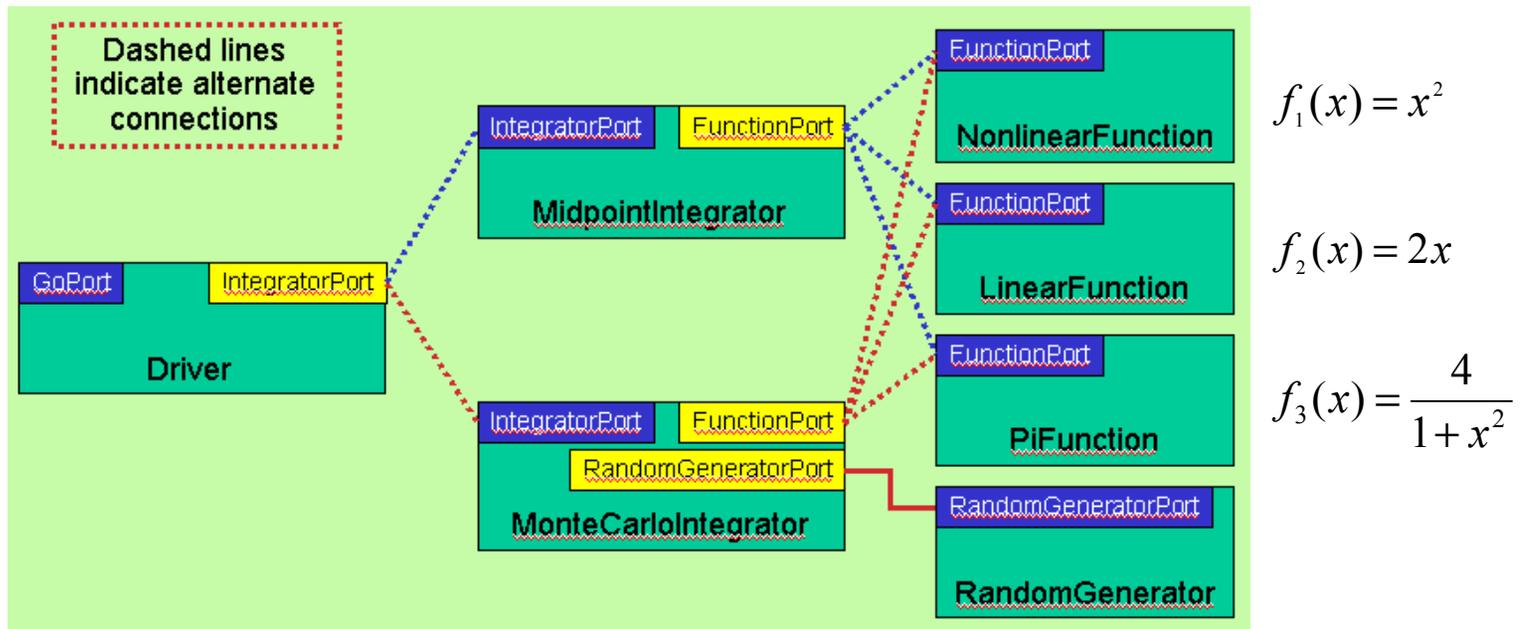
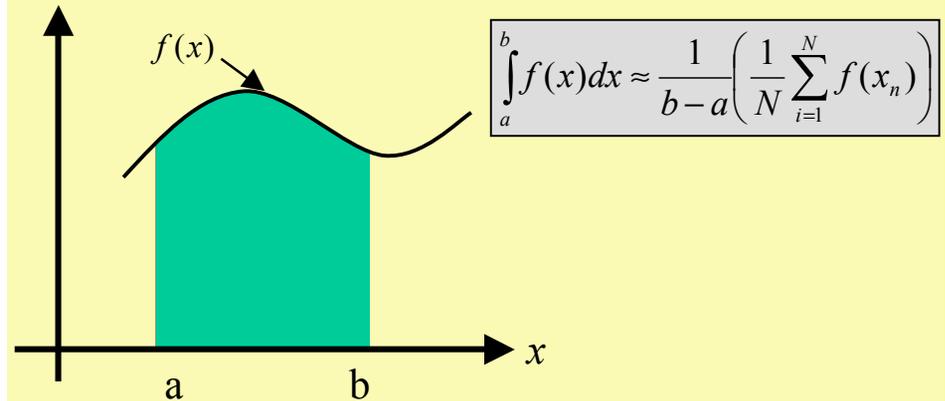
<http://www.zenger.informatik.tu-muenchen.de/persons/huckle/bugse.html>

Components: *simple example*

Numerical integration: midpoint



Numerical integration: Monte Carlo



The DOE ACTS Collection

<http://acts.nersc.gov>

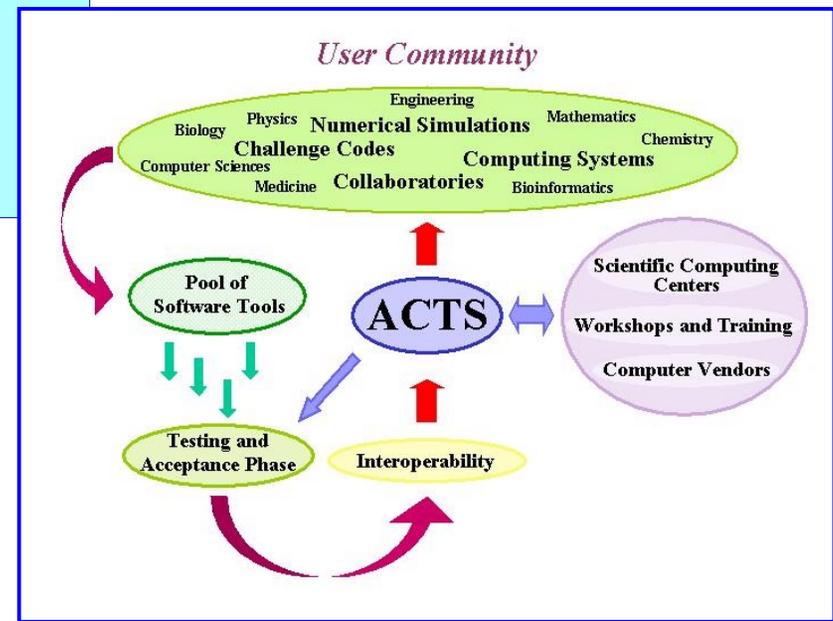
Goals

- ❑ 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

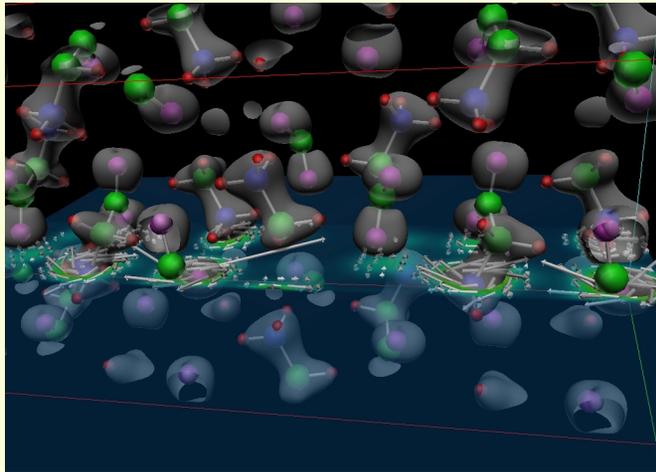
- **High**
 - Intermediate level
 - Tool expertise
 - Conduct tutorials
- **Intermediate**
 - Basic level
 - Higher level of support to users of the tool
- **Basic**
 - Help with installation
 - Basic knowledge of the tools
 - Compilation of user's reports



Current ACTS Tools and their Functionalities

Category	Tool	Functionalities
Numerical	Aztec/Trilinos	Algorithms for the iterative solution of large sparse linear systems.
	Hypre	Algorithms for the iterative solution of large sparse linear systems, intuitive grid-centric interfaces, and dynamic configuration of parameters.
	PETSc	Tools for the solution of PDEs that require solving large-scale, sparse linear and nonlinear systems of equations.
	OPT++	Object-oriented nonlinear optimization package.
	SUNDIALS	Solvers for the solution of systems of ordinary differential equations, nonlinear algebraic equations, and differential-algebraic equations.
	ScaLAPACK	Library of high performance dense linear algebra routines for distributed-memory message-passing.
	SuperLU	General-purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations.
	TAO	Large-scale optimization software, including nonlinear least squares, unconstrained minimization, bound constrained optimization, and general nonlinear optimization.
Code Development	Global Arrays	Library for writing parallel programs that use large arrays distributed across processing nodes and that offers a shared-memory view of distributed arrays.
	Overture	Object-Oriented tools for solving computational fluid dynamics and combustion problems in complex geometries.
Code Execution	CUMULVS	Framework that enables programmers to incorporate fault-tolerance, interactive visualization and computational steering into existing parallel programs
	Globus	Services for the creation of computational Grids and tools with which applications can be developed to access the Grid.
	TAU	Set of tools for analyzing the performance of C, C++, Fortran and Java programs.
Library Development	ATLAS	Tools for the automatic generation of optimized numerical software for modern computer architectures and compilers.

ScaLAPACK: Applications

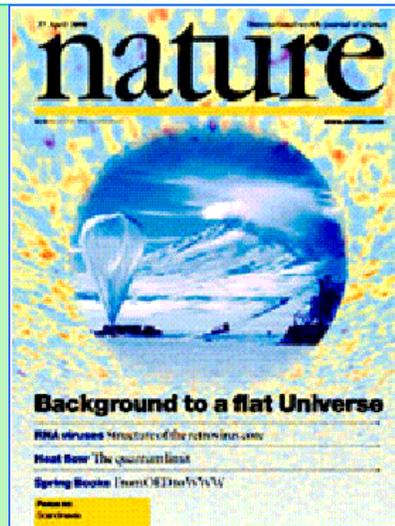


Induced current (white arrows) and charge density (colored plane and gray surface) in crystallized glycine due to an external field (Louie, Yoon, Pfrommer and Canning), eigenvalue problems solved with *ScaLAPACK*.

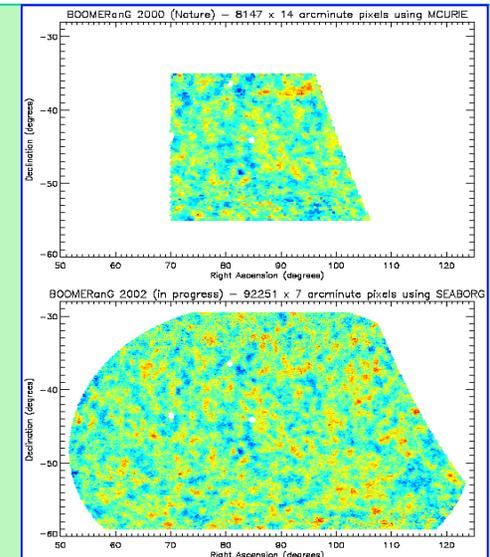
Advanced Computational Research in Fusion (SciDAC Project, PI Mitch Pindzola). Point of contact: Dario Mitnik (Dept. of Physics, Rollins College). Mitnik attended the workshop on the ACTS Collection in September 2000. Since then he has been actively using some of the ACTS tools, in particular *ScaLAPACK*, for which he has provided insightful feedback. Dario is currently working on the development, testing and support of new scientific simulation codes related to the study of atomic dynamics using time-dependent close coupling lattice and time-independent methods. He reports that this work could not be carried out in sequential machines and that *ScaLAPACK* is fundamental for the parallelization of these codes.

Performance of four science-of-scale applications that use *ScaLAPACK* functionalities on an IBM SP

Project	Number of processors	Performance (% of peak)
Electromagnetic Wave-Plasma Interactions	1,936	68%
Cosmic Microwave Background Data Analysis	4,096	50%
Terascale Simulations of Supernovae	2,048	43%
Quantum Chromodynamics at High Temperatures	1,024	13%



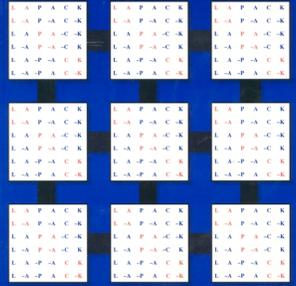
The international BOOMERanG collaboration announced results of the most detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat (Apr. 27, 2000). Likelihood methods implemented in the MADCAP software package, using routines from *ScaLAPACK*, were used to examine the large dataset generated by BOOMERanG.



ScaLAPACK: *software structure*

ScaLAPACK Users' Guide

L. S. Blackford · J. Choi · A. Cleary · E. D'Azevedo
J. Demmel · I. Dhillon · J. Dongarra · S. Hammarling
G. Henry · A. Petitet · K. Stanley · D. Walker · R. C. Whaley



<http://acts.nersc.gov/scalapack>

Version 1.7 released in August 2001; recent NSF funding for further development.

ScaLAPACK

PBLAS

Parallel BLAS.

Global

Local

LAPACK

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

BLACS

Communication routines targeting linear algebra operations.

platform specific

BLAS

MPI/PVM/...

Communication layer (message passing).

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



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

PBLAS

(*Parallel Basic Linear Algebra Subroutines*)

- Similar to the BLAS in portability, functionality and naming:
 - Level 1: vector-vector operations
 - Level 2: matrix-vector operations
 - Level 3: matrix-matrix operations

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

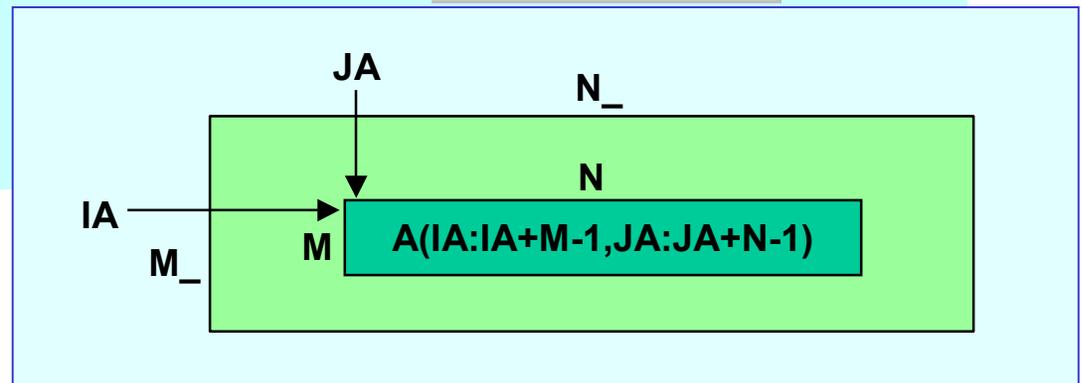
BLAS

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

PBLAS

array descriptor
(see next slides)

- Built atop the BLAS and BLACS
- Provide global view of the matrix operands



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.

Example:
a 3x4 grid

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

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

Scope	Meaning
Row	All processes in a process row participate.
Column	All processes in a process column participate.
All	All processes in the process grid participate.

BLACS: *example*

```
* Get system information
CALL BLACS_PINFO( IAM, NPROCS )
* Get default system context
CALL BLACS_GET( 0, 0, ICTXT )
:
* 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
  :
  CALL BLACS_GRIDEXIT( ICTXT )
END IF
CALL BLACS_EXIT( 0 )
END
```

(out) uniquely identifies each process
(out) number of processes available

(in) integer handle indicating the context
(in) use (default) system context
(out) BLACS context

(output) process row and column coordinate

send X to process (1,0)

receive X from process (0,0)

leave context

exit from the BLACS

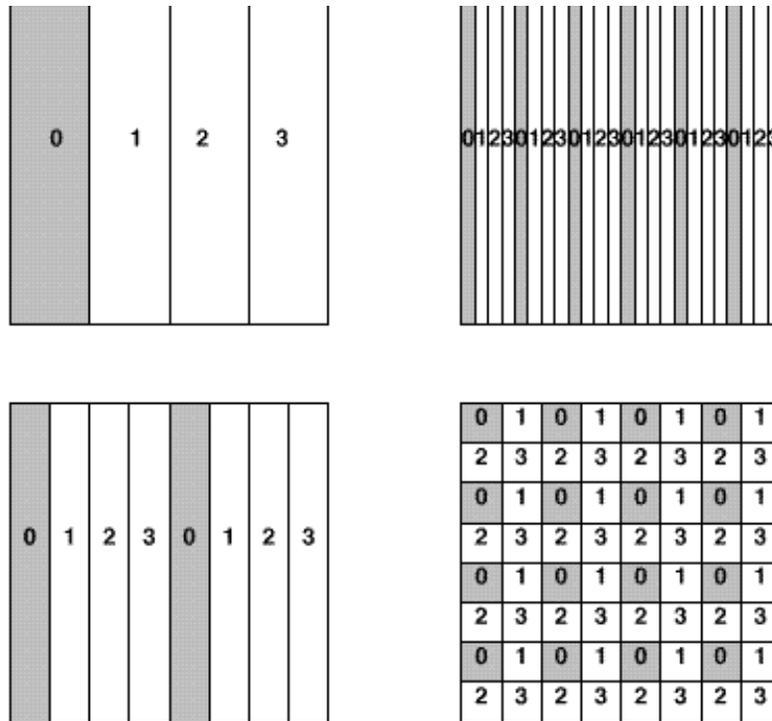
- The BLACS context is the BLACS mechanism for partitioning communication space.
- A message in a context cannot be sent or received in another context.
- The context allows the user to
 - create arbitrary groups of processes
 - create multiple overlapping and/or disjoint grids
 - isolate each process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

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



ScaLAPACK: *data layouts*

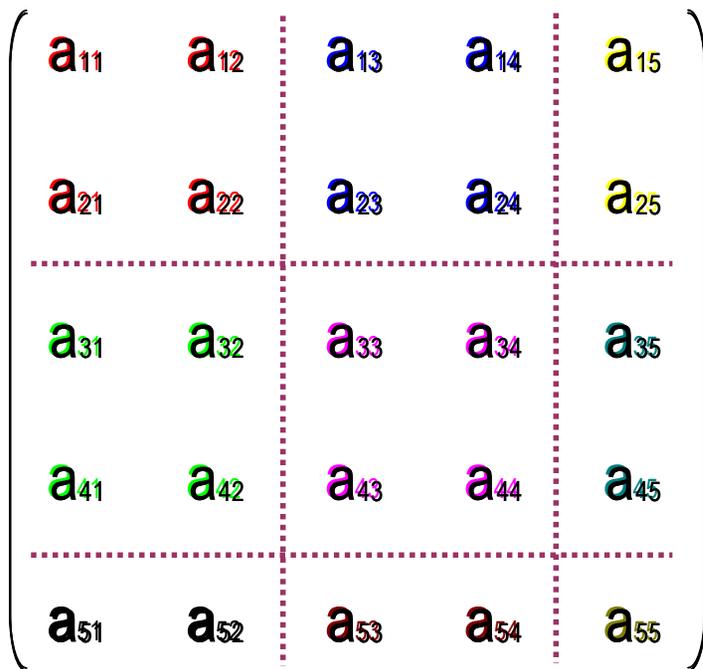
- 1D block and cyclic column distributions



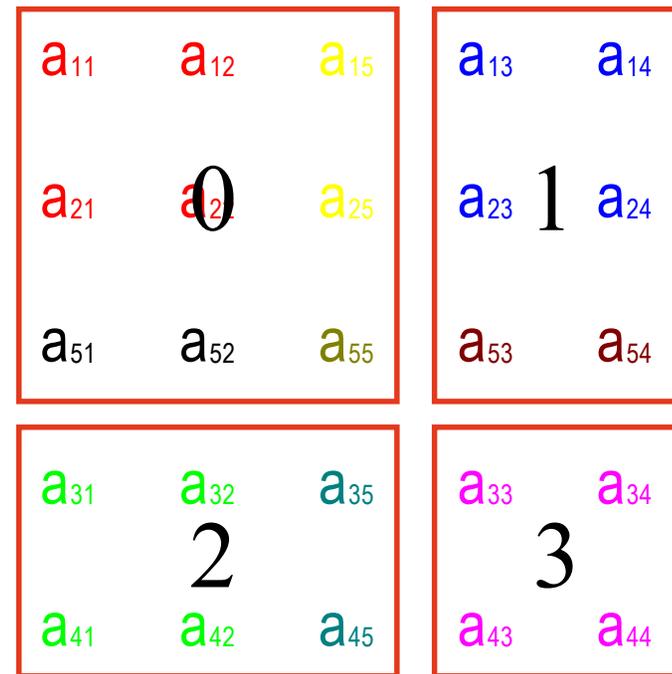
- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

ScaLAPACK: 2D Block-Cyclic Distribution (1/2)

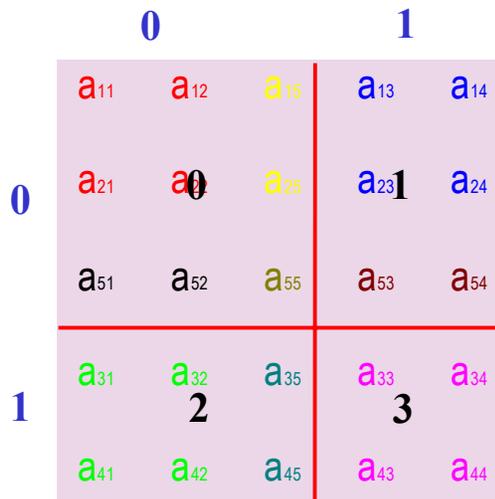
5x5 matrix partitioned in 2x2 blocks



2x2 process grid point of view



ScaLAPACK: 2D Block-Cyclic Distribution (2/2)

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


```

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

```

LDA is the leading dimension of the local array (see next slides)

Array descriptor for A (see next slides)

2D Block-Cyclic Distribution

<http://acts.nersec.gov/scalapack/hands-on/datadist.html>

ScaLAPACK Block Cyclic Data Distribution - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop <http://acts.nersec.gov/scalapack/hands-on/d> Search Print

Block Cyclic Data Distribution

The block cyclic data distribution used by ScaLAPACK is justified by the analysis of many algorithms intended for linear algebra calculations and by the goal of achieving good load balancing.

Use the entries in the form below to specify the matrix dimensions, blocking, and process grid configuration (note the acceptable values between parentheses). Then click on "distribute data": a new window will pop up with the corresponding block cyclic distribution. The defaults correspond to those of the matrix A used in [Exercise 3](#).

Note that although the PBLAS allow for non-square blocking factors, most ScaLAPACK routines do not, because of alignment constraints (which vary from routine to routine).

matrix dimensions	number of rows	<input type="text" value="9"/> (>0, <101)
	number of columns	<input type="text" value="9"/> (>0, <101)
matrix blocking	rows per block	<input type="text" value="2"/> (>0)
	columns per block	<input type="text" value="2"/> (>0)
process grid	rows	<input type="text" value="2"/> (>0)
	columns	<input type="text" value="3"/> (>0)

ScaLAPACK Tools Project Home Search



Block Cyclic Data Distribution

A = global array, B = local array
array indices start at 1

Process (coordinates)	Array Values
0 (0,0)	B[1,1]=A[1,1] B[1,2]=A[1,2] B[2,1]=A[2,1] B[2,2]=A[2,2] B[1,3]=A[1,7] B[1,4]=A[1,8] B[2,3]=A[2,7] B[2,4]=A[2,8] B[3,1]=A[5,1] B[3,2]=A[5,2] B[4,1]=A[6,1] B[4,2]=A[6,2] B[3,3]=A[5,7] B[3,4]=A[5,8] B[4,3]=A[6,7] B[4,4]=A[6,8] B[5,1]=A[9,1] B[5,2]=A[9,2] B[5,3]=A[9,7] B[5,4]=A[9,8]
1 (0,1)	B[1,1]=A[1,3] B[1,2]=A[1,4] B[2,1]=A[2,3] B[2,2]=A[2,4] B[1,3]=A[1,9] B[2,3]=A[2,9] B[3,1]=A[5,3] B[3,2]=A[5,4] B[4,1]=A[6,3] B[4,2]=A[6,4] B[3,3]=A[5,9] B[4,3]=A[6,9] B[5,1]=A[9,3] B[5,2]=A[9,4] B[5,3]=A[9,9]
2 (0,2)	B[1,1]=A[1,5] B[1,2]=A[1,6] B[2,1]=A[2,5] B[2,2]=A[2,6] B[3,1]=A[5,5] B[3,2]=A[5,6] B[4,1]=A[6,5] B[4,2]=A[6,6] B[5,1]=A[9,5] B[5,2]=A[9,6]
3 (1,0)	B[1,1]=A[3,1] B[1,2]=A[3,2] B[2,1]=A[4,1] B[2,2]=A[4,2] B[1,3]=A[3,7] B[1,4]=A[3,8] B[2,3]=A[4,7] B[2,4]=A[4,8] B[3,1]=A[7,1] B[3,2]=A[7,2] B[4,1]=A[8,1] B[4,2]=A[8,2] B[3,3]=A[7,7] B[3,4]=A[7,8] B[4,3]=A[8,7] B[4,4]=A[8,8]
4 (1,1)	B[1,1]=A[3,3] B[1,2]=A[3,4] B[2,1]=A[4,3] B[2,2]=A[4,4] B[1,3]=A[3,9] B[2,3]=A[4,9] B[3,1]=A[7,3] B[3,2]=A[7,4] B[4,1]=A[8,3] B[4,2]=A[8,4] B[3,3]=A[7,9] B[4,3]=A[8,9]
5 (1,2)	B[1,1]=A[3,5] B[1,2]=A[3,6] B[2,1]=A[4,5] B[2,2]=A[4,6] B[3,1]=A[7,5] B[3,2]=A[7,6] B[4,1]=A[8,5] B[4,2]=A[8,6]

2D Block-Cyclic Distribution: *pros and cons*

- Ensures good load balance → performance and scalability (analysis of many algorithms to justify this layout).
- Encompasses a large number of data distribution schemes (but not all).
- Needs redistribution routines to go from one distribution to the other.

ScaLAPACK: *array descriptors*

- 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

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



Array Descriptor for Narrow Band Matrices

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=501 for 1 x Pc process grid for band and tridiagonal matrices block-column distributed.
2	CTXT_A	(global)	BLACS context handle.
3	N_A	(global)	Number of columns in global array A.
4	NB_A	(global)	Blocking factor used to distribute the columns of array A.
5	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
6	LLD_A	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored.
7	—	—	Unused, reserved.



Array Descriptor for Right Hand Sides for Narrow Band Linear Solvers

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_B	(global)	Descriptor type DTYPE_B=502 for Pr x 1 process grid for block -row distributed matrices .
2	CTXT_B	(global)	BLACS context handle.
3	M_B	(global)	Number of rows in global array B
4	MB_B	(global)	Blocking factor used to distribute the rows of array B.
5	RSRC_B	(global)	Process row over which the first row of the array B is distributed.
6	LLD_B	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored.
7	—	—	Unused, reserved.



ScaLAPACK: *Functionality*

$Ax = b$	Simple Driver	Expert Driver	Factor	Solve	Inversion	Conditioning Estimator	Iterative Refinement
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				
$Ax = \lambda x$ or $Ax = \lambda Bx$	Simple Driver	Expert Driver	Reduction	Solution			
Symmetric	x	x	x	x			
General	x	x	x	x			
Generalized BSPD	x		x	x			
SVD			x	x			

On line tutorial: <http://acts.nersc.gov/scalapack/hands-on/main.html>

Hands-On Exercises for ScaLAPACK

ScaLAPACK Team
August 2002

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](#)

Addition Resources:

- ♦ [Block Cyclic Data Distribution](#)
- ♦ [Useful calling sequences](#)
- ♦ [Download all exercises](#)

[ScaLAPACK](#)

[Tools](#)

[Project](#)

[Home](#)

[Search](#)

Contents of hands-on/example0

```
seaborg
[/usr/common/homes/o/osni/hands-on/example0] ls -la
total 224
drwxr-xr-x  2 osni  m340      8192 Aug 18 12:39 .
drwxr-xr-x  8 osni  m340      8192 Aug 17 15:51 ..
-rwxr-xr-x  1 osni  m340      2906 Aug 18 12:39 A.SVD
-rwxr-xr-x  1 osni  m340      2700 Aug 18 12:39 A.dat
-rwxr-xr-x  1 osni  m340      3953 Aug 18 12:39 pddttrdrv.c
-rwxr-xr-x  1 osni  m340      3705 Aug 18 12:39 pddttrdrv.f
-rw-----  1 osni  m340        365 Aug 18 12:39 pddttrdrv.job
-rwxr-xr-x  1 osni  m340        13 Aug 18 12:39 pdgesvddrv.dat
-rwxr-xr-x  1 osni  m340      6951 Aug 18 12:39 pdgesvddrv.f
-rw-----  1 osni  m340        369 Aug 18 12:39 pdgesvddrv.job_3
-rw-----  1 osni  m340        369 Aug 18 12:39 pdgesvddrv.job_4
-rwxr-xr-x  1 osni  m340      6160 Aug 18 12:39 pdpitr_2.c
-rwxr-xr-x  1 osni  m340      4284 Aug 18 12:39 pdpitr_2.f
-rw-----  1 osni  m340        361 Aug 18 12:39 pdpitr_2.job
[/usr/common/homes/o/osni/hands-on/example0]
```

pddttrdrv.c (*pddttrdrv.f*): illustrates the use of the ScaLAPACK routines PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations $Tx = b$. After compilation, it can be executed with *llsubmit pddttrdrv.job*.

pdpitr_2.c (*pdpitr_2.f*): illustrates the use of the ScaLAPACK routines PDPTTRF and PPPTTRS to factor and solve a symmetric positive definite tridiagonal system of linear equations $Tx = b$, in two distinct contexts. After compilation, it can be executed with *llsubmit pdpitr_2.job*.

pdgesvddrv.f: reads a (full) matrix A from a file, distributes A among the available processors and then call the ScaLAPACK subroutine PDGESVD to computed the SVD of A , $A=USV^T$. It requires the file *pdgesvddrv.dat*, which should contain: line 1, the name of the file where A will be read from; line 2, the number of rows of A ; line 3: the number of columns of A . Considering the file *A.dat*:

- if $m=n=10$ the results are given in the file *A.SVD*
- if $m=10, n=7$: $\text{diag}(S)=[4.4926 \ 1.4499 \ 0.8547 \ 0.8454 \ 0.6938 \ 0.4332 \ 0.2304]$
- if $m=7, n=10$: $\text{diag}(S)=[4.5096 \ 1.1333 \ 1.0569 \ 0.8394 \ 0.8108 \ 0.5405 \ 0.2470]$

```

/*****
/* This program illustrates the use of the ScalAPACK routines PDPTRF */
/* and PPTTRS to factor and solve a symmetric positive definite */
/* tridiagonal system of linear equations, i.e., T*x = b, with */
/* different data in two distinct contexts. */
/*****

/* a bunch of things omitted for the sake of space */

main()
{
    /* Start BLACS */
    Cblacs_pinfo( &mype, &npe );
    Cblacs_get( 0, 0, &context );
    Cblacs_gridinit( &context, "R", 1, npe );
    /* Processes 0 and 2 contain d(1:4) and e(1:4) */
    /* Processes 1 and 3 contain d(5:8) and e(5:8) */
    if ( mype == 0 || mype == 2 ){
        d[0]=1.8180; d[1]=1.6602; d[2]=1.3420; d[3]=1.2897;
        e[0]=0.8385; e[1]=0.5681; e[2]=0.3704; e[3]=0.7027;
    }
    else if ( mype == 1 || mype == 3 ){
        d[0]=1.3412; d[1]=1.5341; d[2]=1.7271; d[3]=1.3093;
        e[0]=0.5466; e[1]=0.4449; e[2]=0.6946; e[3]=0.0000;
    }
    if ( mype == 0 || mype == 1 ) {
        /* New context for processes 0 and 1 */
        map[0]=0; map[1]=1;
        Cblacs_get( context, 10, &context_1 );
        Cblacs_gridmap( &context_1, map, 1, 1, 2 );
        /* Right-hand side is set to b = [ 1 2 3 4 5 6 7 8 ] */
        if ( mype == 0 ) {
            b[0]=1.0; b[1]=2.0; b[2]=3.0; b[3]=4.0;
        }
        else if ( mype == 1 ) {
            b[0]=5.0; b[1]=6.0; b[2]=7.0; b[3]=8.0;
        }
        /* Array descriptor for A (D and E) */
        desca[0]=501; desca[1]=context_1; desca[2]=n; desca[3]=nb;
        desca[4]=0; desca[5]=lda; desca[6]=0;
        /* Array descriptor for B */
        descb[0]=502; descb[1]=context_1; descb[2]=n; descb[3]=nb;
        descb[4]=0; descb[5]=ldb; descb[6]=0;
        /* Factorization */
        pdpstrf( &n, d, e, &ja, desca, af, &laf,
            work, &lwork, &info );
        /* Solution */
        pdpstrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
            af, &laf, work, &lwork, &info );
        printf( "MYPE=%i: x[:] = %7.4f %7.4f %7.4f %7.4f\n",
            mype, b[0], b[1], b[2], b[3] );
    }
}

```

```

else {
    /* New context for processes 0 and 1 */
    map[0]=2; map[1]=3;
    Cblacs_get( context, 10, &context_2 );
    Cblacs_gridmap( &context_2, map, 1, 1, 2 );
    /* Right-hand side is set to b = [ 8 7 6 5 4 3 2 1 ] */
    if ( mype == 2 ) {
        b[0]=8.0; b[1]=7.0; b[2]=6.0; b[3]=5.0;
    }
    else if ( mype == 3 ) {
        b[0]=4.0; b[1]=3.0; b[2]=2.0; b[3]=1.0;
    }
    /* Array descriptor for A (D and E) */
    desca[0]=501; desca[1]=context_2; desca[2]=n; desca[3]=nb;
    desca[4]=0; desca[5]=lda; desca[6]=0;
    /* Array descriptor for B */
    descb[0]=502; descb[1]=context_2; descb[2]=n; descb[3]=nb;
    descb[4]=0; descb[5]=ldb; descb[6]=0;
    /* Factorization */
    pdpstrf( &n, d, e, &ja, desca, af, &laf,
        work, &lwork, &info );
    /* Solution */
    pdpstrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
        af, &laf, work, &lwork, &info );
    printf( "MYPE=%i: x[:] = %7.4f %7.4f %7.4f %7.4f\n",
        mype, b[0], b[1], b[2], b[3] );
}
Cblacs_gridexit( context );
Cblacs_exit( 0 );
}

```

Using Matlab notation:

$$T = \text{diag}(D) + \text{diag}(E, -1) + \text{diag}(E, 1)$$

where

$$D = [1.8180 \ 1.6602 \ 1.3420 \ 1.2897 \ 1.3412 \ 1.5341 \ 1.7271 \ 1.3093]$$

$$E = [0.8385 \ 0.5681 \ 0.3704 \ 0.7027 \ 0.5466 \ 0.4449 \ 0.6946]$$

Then, solving $T*x = b$,

$$\text{if } b = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8]$$

$$x = [0.3002 \ 0.5417 \ 1.4942 \ 1.8546 \ 1.5008 \ 3.0806 \ 1.0197 \ 5.5692]$$

$$\text{if } b = [8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1]$$

$$x = [3.9036 \ 1.0772 \ 3.4122 \ 2.1837 \ 1.3090 \ 1.2988 \ 0.6563 \ 0.4156]$$

Global Arrays (GA) wrappers

<http://www.emsl.pnl.gov/docs/global/ga.html>

- Simpler than message-passing for many applications
- Complete environment for parallel code development
- Data locality control similar to distributed memory/message passing model
- Compatible with MPI
- Scalable

Distributed Data: data is explicitly associated with each processor, accessing data requires specifying the location of the data on the processor and the processor itself.

Shared Memory: data is in a globally accessible address space, any processor can access data by specifying its location using a global index.

GA: distributed dense arrays that can be accessed through a shared memory-like style.

The function

```
Fortran integer function ga_solve(g_a, g_b)
C       int GA_Solve(int g_a, int g_b)
C++    int GA::GlobalArray::solve(const GA::GlobalArray * g_a)
```

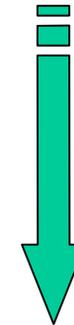
solves a system of linear equations $A * X = B$. It first will call the Cholesky factorization routine and, if successful, will solve the system with the Cholesky solver. If Cholesky is not able to factorize A , then it will call the LU factorization routine and will solve the system with forward/backward substitution. On exit B will contain the solution X .



Who Benefits from these tools?

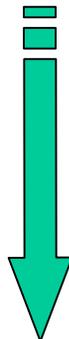
Application	Computational Problem	Software Tools	Highlights
MADCAP	Matrix factorization and triangular solves	ScaLAPACK	<ul style="list-style-type: none"> • 50% peak performance on an IBM SP • Nearly perfect scalability on 1024, 2048, 3072 and 4096 processors • Fast implementation of numerical algorithms
3-Charged Particles	Solution of large, complex unsymmetric linear systems	SuperLU	<ul style="list-style-type: none"> • Solves systems of equations of order 8.4 million on 64 processors in 1 hour of wall clock time • 30 GFLOPs
NWChem	Distribute large data arrays, collective operations	Global Arrays and LAPACK	<ul style="list-style-type: none"> • Very good scaling for large problems

<http://acts.nersc.gov/AppMat>



Enabling sciences and discoveries... with high performance and scalability...

... More Applications ...



FDN3D &	Unstructured grids, compressible and incompressible Euler and Navier-Stokes equations.	PETSc	<ul style="list-style-type: none"> • Parallelization of legacy code • Gordon Bell price, 0.23 Tflop/s on 3072 procs of ASCI Red
P-FLAPW	Eigenvalue problems	ScaLAPACK	<ul style="list-style-type: none"> • Study of systems up to 700 atoms (mat size=35,000) • Runs efficiently • Facilitated the study of new problems in materials science such as impurities and disordered systems.
NIMROD	Quad and triangular high order finite elements, semi-implicit time integration, sparse matrix solvers	SuperLU	<ul style="list-style-type: none"> • Code improvement of 5 fold, equivalent to 3-5 years progress in computing hardware.

TAU - Tuning and Performance Analysis

- Multi-level performance instrumentation
 - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
 - Computer system architectures and operating systems
 - Different programming languages and compilers
- Support for multiple parallel programming paradigms
 - Multi-threading, message passing, mixed-mode, hybrid
- Support for performance mapping
- Support for object-oriented and generic programming
- Integration in complex software systems and applications



Study Case: Electronic Structure Calculations Code

```

xterm
#Makefile for PETot on IBM SP
#Modules required: "lapack" and "tau"
include $(TAUROOTDIR)/rs6000/lib/Makefile.tau-mpi-pdt

FC=mpxlf90_r
#LDR=mpxlf90_r
#FC=$(TAU_F90)
LDR=$(TAU_LINKER)
PDTF90PARSE = $(PDTDIR)/$(PDTARCHDIR)/bin/f95parse
TAUINSTR = $(TAUROOTDIR)/$(CONFIG_ARCH)/bin/tau_instrumentor
#FFLAGS = -g -qfixed=80 -qarch=pwr3 -qmaxmem=4096 \
# $(TAU_INCLUDE) $(TAU_MPI_INCLUDE)
FFLAGS = -g -qfixed=80 -qarch=pwr3 -qmaxmem=4096 $(TAU_MPI_INCLUDE)
LDFLAGS = -brtl -binitfini:poe_remote_main
LIB = $$LAPACK -lessl
TAULIBS = $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_FORTRANLIBS)

COMP_RULE = $(PDTF90PARSE) $< -f $(TAU_MPI_INCLUDE); \
$(TAUINSTR) $*.pdb $< -o $*.inst.f; \
$(FC) $(FFLAGS) -c $*.inst.f -o $@; \
if [ ! -f $@ ]; then \
echo "Error in compiling $*.inst.f: trying without PDT"; \
$(FC) $(FFLAGS) -c $< -o $@; \
fi; \
rm -f $*.pdb ;
    
```

```

#.f.o:
# $(FC) $(FFLAGS) -c $<
.f.o:
$(COMP_RULE)
    
```

ParaProf Manager

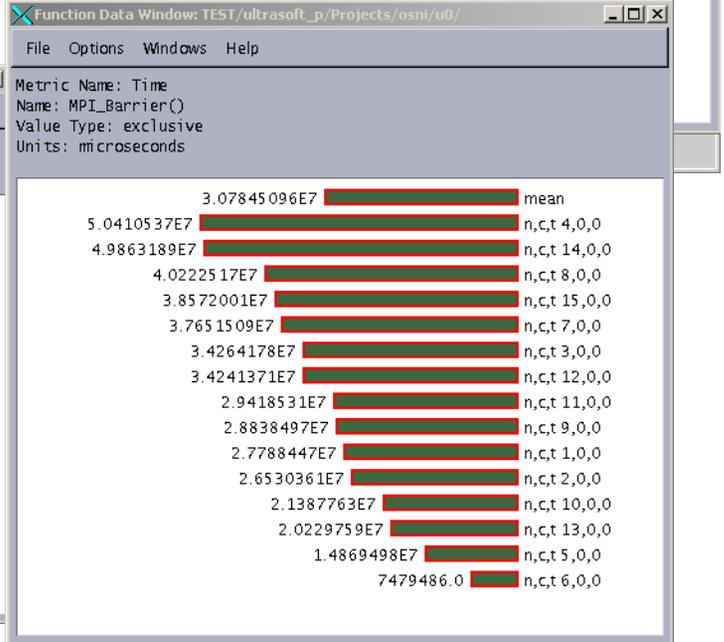
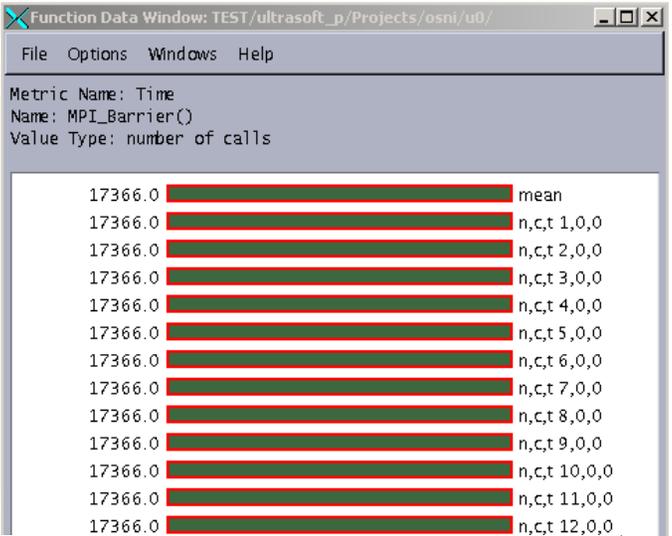
File Options Help

- Applications
 - Standard Applications
 - Default App
 - Default Exp
 - Default Trial
 - Time
- Runtime Applications
- DB Applications

Function Ledger Window: TEST/ultraso...

File Windows Help

- MPLBarrier()
- W_LINE_VWR
- MPLAllreduce()
- GEN_G_COMP
- HPSI_COMP::NONLOCAL_QSP
- CG_COMP
- INVCFFT_COMP
- MPLBcast()



Using TAU_COMPILER in previous example

```
#####
/*
/* This makefile shows how to use TAU to automatically instrument and
/* compile a simple Fortran program that calls the ScalAPACK routine
/* PSGESV to solve a system of linear equations. It requires the
/* modules "tau" and "scalapack", as well as "gmake"
/*
#####

.SUFFIXES : .f90

# The following defines the SCALAPACK macros
# include $(SCALAPACKROOTDIR)/SLmake.inc

# The following defines the TAU macros
# For other options see $(TAUROOTDIR)/rs6000/lib
include $(TAUROOTDIR)/rs6000/lib/Makefile.tau-mpi-pdt-profile-trace

F90          = $(TAU_COMPILER) $(TAU_F90)
F90SUFFIX    = -qsuffix=f-f90
LINKER       = $(TAU_LINKER)
LIBS         = $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_FORTRANLIBS)
LDFLAGS      = -brtl -binitfini:poe_remote_main
STLIBS       = $(SCALAPACK) $(PBLAS) $(BLACS) -lesslp2

TARGET = psgesvdriver.x
OBJS = psgesvdriver.o

$(TARGET): $(OBJS)
    $(LINKER) $(LDFLAGS) $(OBJS) -o $@ $(LIBS) $(STLIBS)

.f90.o:
    $(F90) $(F90SUFFIX) -c $<

clean:
    -@rm -f $(TARGET) *.o *.inst.f90 profile.* \
    *.trc *.edf *.pv *.pprof *.txt
```



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

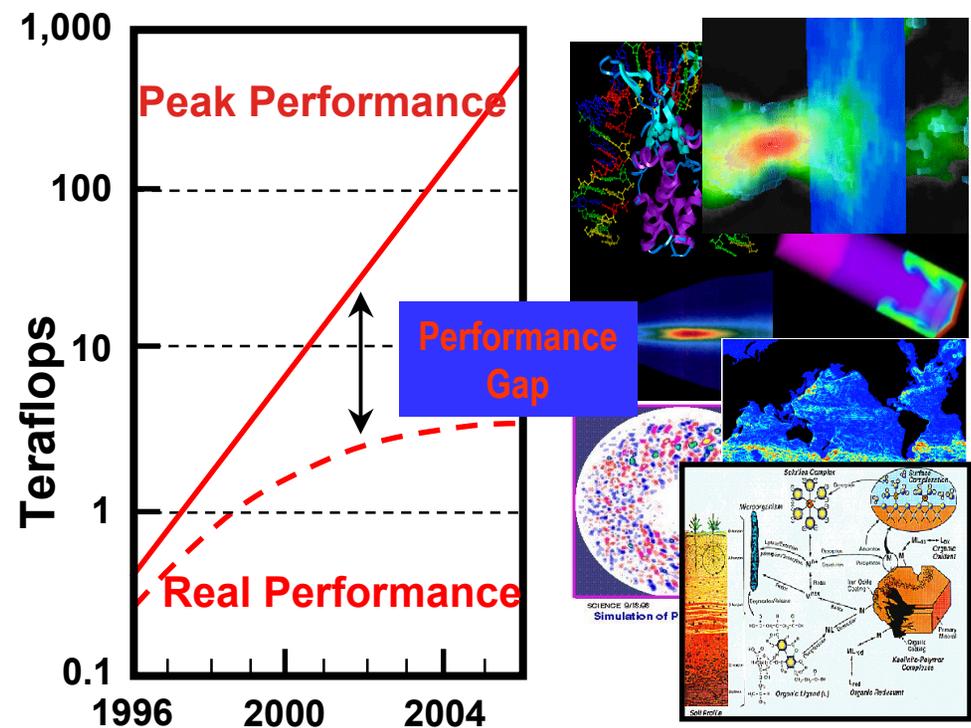
- Libraries written in different languages.
- Different pieces of the code evolve at different rates
- Swapping competing implementations of the same idea and testing without modifying the code

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*



<http://acts.nerisc.gov>

The DOE ACTS Collection

[Tools](#) [News](#) [Project](#) [Center](#) [Search](#)

click on the image below to see other applications that have benefited from ACTS Tools

The DOE ACTS (Advanced Computational Software) Collection is a set of DOE-developed software tools that make it easier for programmers to write high performance scientific applications for parallel computers. This site is the central information center for the ACTS Collection and is brought to you by NERSC and the [Mathematical, Information, and Computational Sciences \(MICS\)](#) Division of DOE. Correspondence regarding the collection (including requests for support) should be directed to acts-support@nerisc.gov.

The image shows pressure and velocity around a moving valve in a diesel engine. The flow here was found as part of a CFD effort to simulate the flow within the complex 3D geometry of a diesel engine. The computation was carried out using the Overture Framework and the PADRE library for parallel data distribution.

[Tools](#) [News](#) [Project](#) [Center](#) [Search](#)

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

- ❖ **Compframe 2005**
- ❖ **ACTS Workshop 2005**