

**13<sup>th</sup> DOE ACTS Collection Workshop**  
**August 14-17, 2012**

# ScaLAPACK

Osni Marques  
Lawrence Berkeley National Laboratory  
*OAMarques@lbl.gov*

# Outline

---

- Functionalities and applications
- ScaLAPACK: software structure
  - Basic Linear Algebra Subprograms (BLAS)
  - Linear Algebra PACKage (LAPACK)
  - Basic Linear Algebra Communication Subprograms (BLACS)
  - Parallel BLAS (PBLAS)
- ScaLAPACK: details
  - Data layout
  - Array descriptors
  - Error handling
  - Performance
- Examples

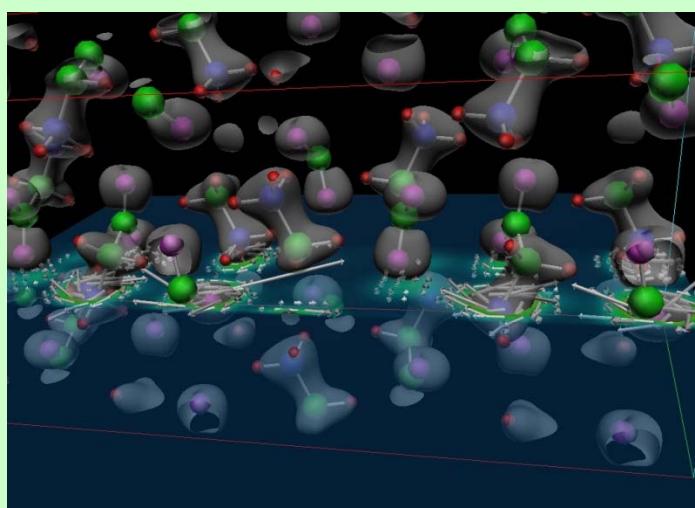
# ScaLAPACK: functionalities

---

- Linear systems:  $Ax = b$
- SVD:  $A = U\Sigma V^T$
- Least squares:  $\min\|Ax - b\|_2$
- Eigenvalues/vectors:  $Az = \lambda z$ ,  $Az = \lambda Bz$

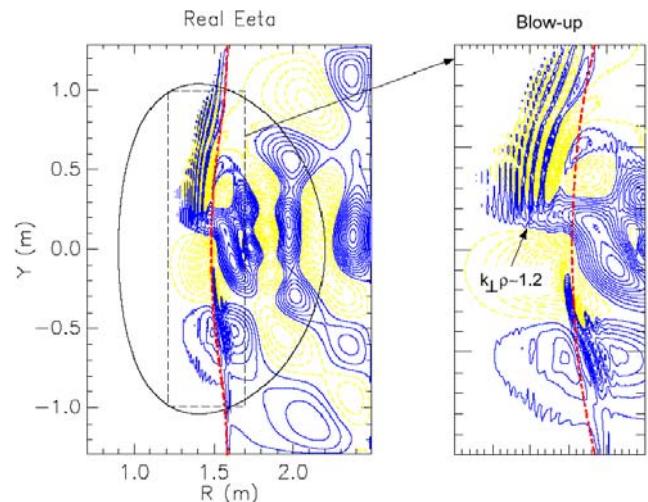
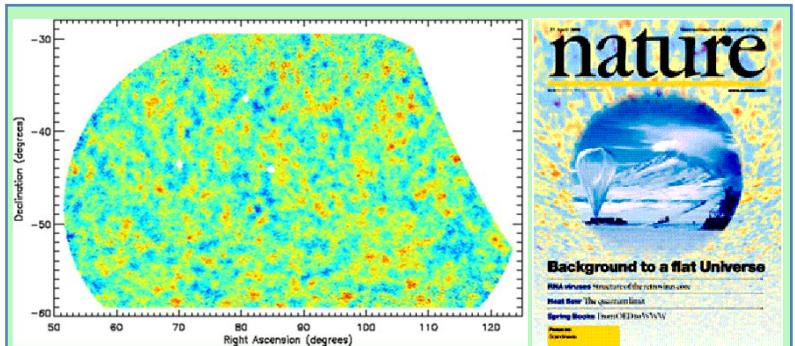
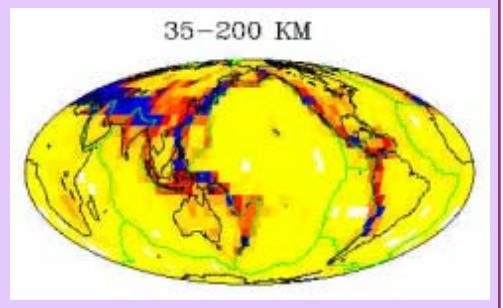
$Ax = b$	Simple Driver	Expert Driver	Factor	Solve	Inversion	Conditioning Estimator	Iterative Refinement
Triangular				X	X	X	X
SPD	X	X	X	X	X	X	X
SPD Banded	X		X	X			
SPD Tridiagonal	X		X	X			
General	X	X	X	X	X	X	X
General Banded	X		X	X			
General Tridiagonal	X		X	X			
Least Squares	X		X	X			
GQR			X				
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			

# Applications



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

*Model for the internal structure of the Earth, resolution matrix (Vasco and Marques)*

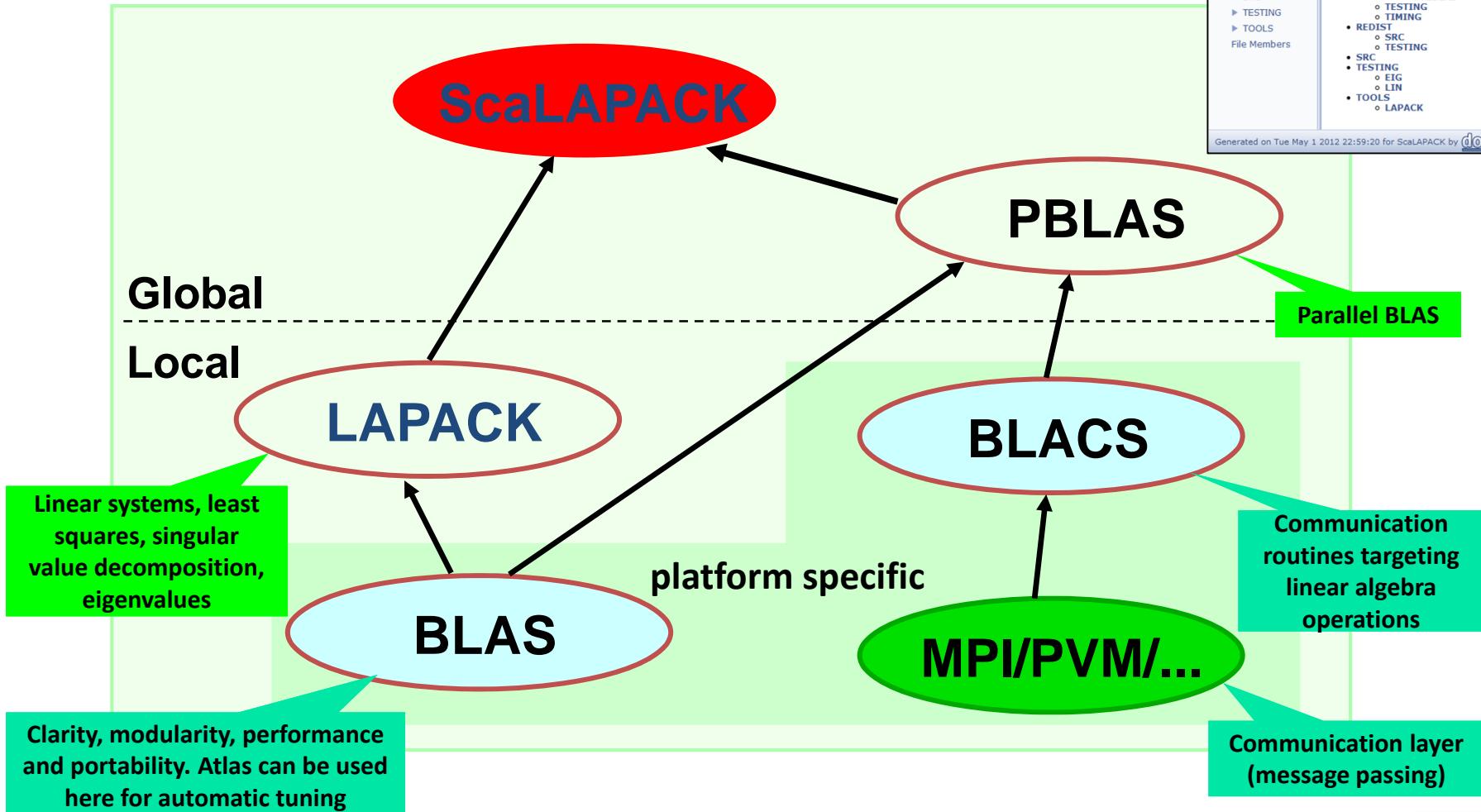


Two **ScaLAPACK** routines, PZGETRF and PZGETRS, are used for solution of linear systems in the spectral algorithms based AORSA code (Batchelor et al.), which is intended for the study of electromagnetic wave-plasma interactions.

The international BOOMERanG performed a detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat. The MADCAP (Microwave Anisotropy Dataset Computational Analysis Package) code makes maps from observations of the CMB and then calculates their angular power spectra. These calculations are dominated by the solution of linear systems, which are solved with **ScaLAPACK**.

# ScaLAPACK: software structure

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



# Basic Linear Algebra Subroutines (BLAS)

- Level 1 BLAS: vector-vector

$$\begin{bmatrix} & \end{bmatrix} \leftarrow \begin{bmatrix} & \end{bmatrix} + \alpha * \begin{bmatrix} & \end{bmatrix}$$

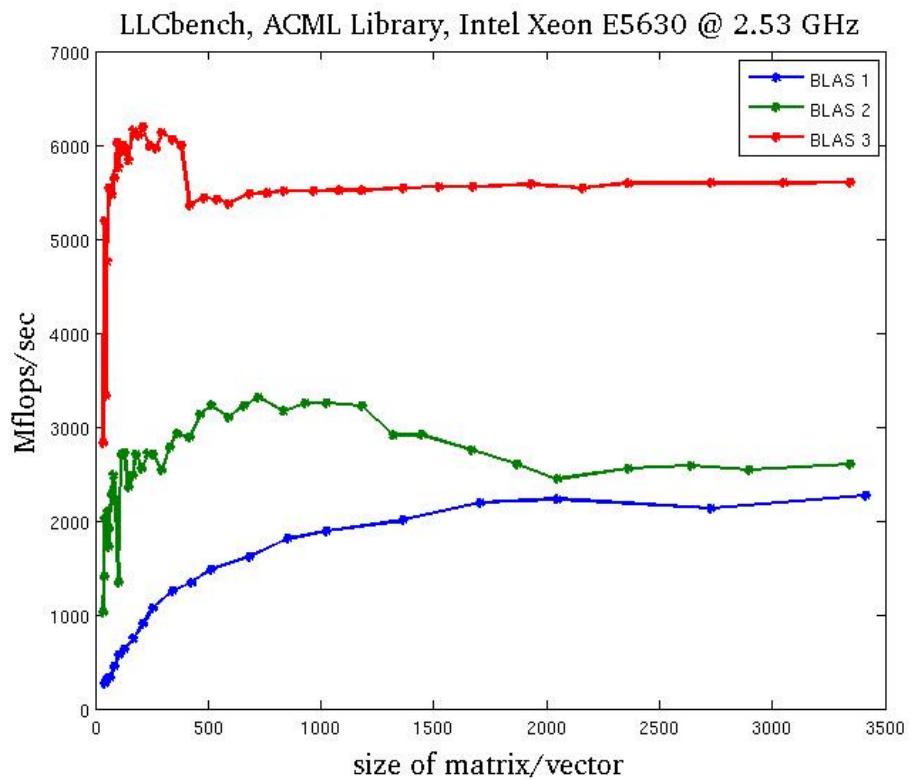
- Level 2 BLAS: matrix-vector

$$\begin{bmatrix} & \end{bmatrix} \leftarrow \begin{bmatrix} & \end{bmatrix} * \begin{bmatrix} & \end{bmatrix}$$

- Level 3 BLAS: matrix-matrix

$$\begin{bmatrix} & \end{bmatrix} \leftarrow \begin{bmatrix} & \end{bmatrix} + \begin{bmatrix} & \end{bmatrix} * \begin{bmatrix} & \end{bmatrix}$$

- Clarity
- Portability
- Performance: development of blocked algorithms is vital for performance!

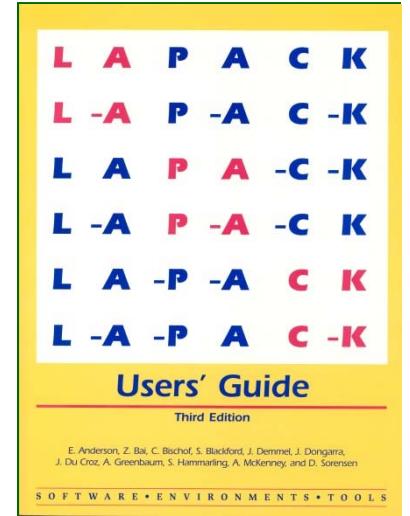


See <http://acts.nersc.gov/atlas>

# LAPACK: main features

(<http://www.netlib.org/lapack>)

- Linear Algebra library written in Fortran
- Combine algorithms from LINPACK and EISPACK into a single package
- Efficient on a wide range of computers
- Built atop level 1, 2, and 3 BLAS 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 ...)



# Basic Linear Algebra Communication Subroutines (BLACS)

---

- 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

---

- Promote efficiency by identifying common operations of linear algebra that can be optimized on various computers
- Processes are embedded in a two-dimensional grid

Example: a 3x4 grid

		0	1	2	3	
		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: communication routines

## Send/Receive:

```
_xxSD2D( ICTXT , [ UPLO , DIAG ] , M , N , A , LDA , RDEST , CDEST )  
_xxRV2D( ICTXT , [ UPLO , DIAG ] , M , N , A , LDA , RSRC , CSRC )
```

<b>_ (Data type)</b>	<b>xx (Matrix type)</b>
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix TR: Trapezoidal matrix

## Broadcast:

```
_xxBS2D( ICTXT , SCOPE , TOP , [ UPLO , DIAG ] , M , N , A , LDA )  
_xxBR2D( ICTXT , SCOPE , TOP , [ UPLO , DIAG ] , M , N , A , LDA , RSRC , CSRC )
```

<b>SCOPE</b>	<b>TOP</b>
'Row'	' ' (default)
'Column'	'Increasing Ring'
'All'	'1-tree' ...

# 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 ) ← leave context
  END IF
:
  CALL BLACS_EXIT( 0 ) ← exit from the BLACS
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)

- 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

# Parallel Basic Linear Algebra Subroutines (PBLAS)

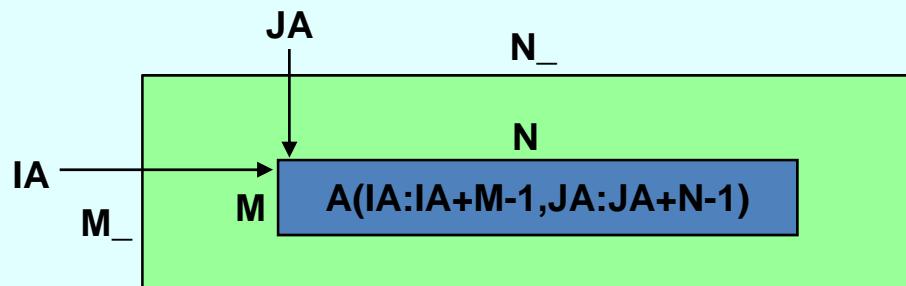
- Similar to the BLAS in portability, functionality and naming
- Built atop the BLAS and BLACS
- Provide global view of matrix

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

BLAS

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

PBLAS



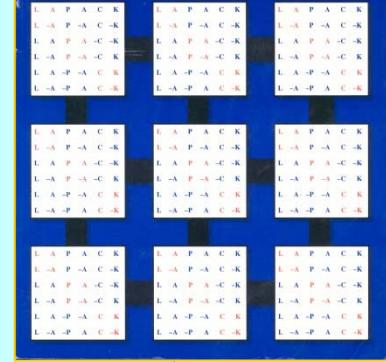
Array descriptor  
(see next slides)

# ScaLAPACK

- Efficiency
  - Optimized computation and communication engines
  - Block-partitioned algorithms (BLAS 3) 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

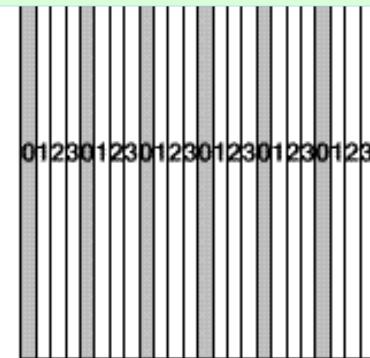
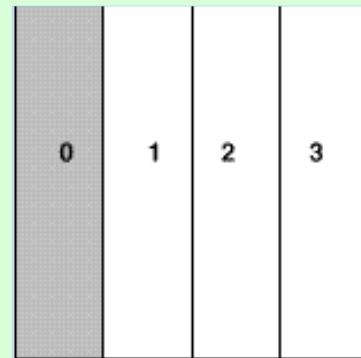
## ScaLAPACK Users' Guide

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



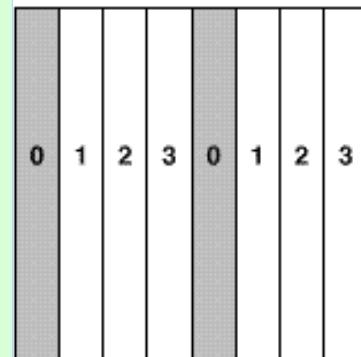
# Parallel Data Distribution

*1D column distribution*



*1D column cyclic distribution*

*1D block column distribution*



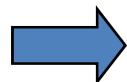
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

*2D block cyclic distribution*

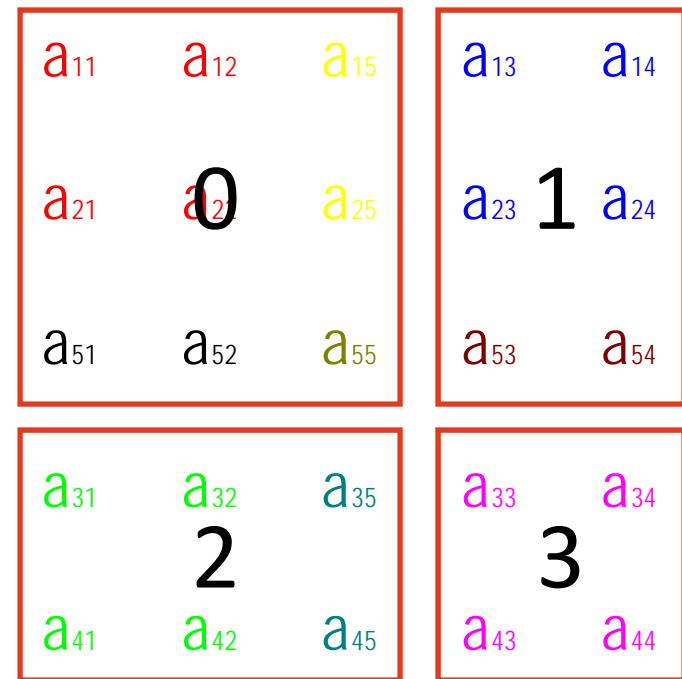
# ScaLAPACK: 2D Block-Cyclic Distribution

5x5 matrix partitioned in 2x2 blocks

$$\left( \begin{array}{cc|cc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array} \right)$$



2x2 process grid point of view



# 2D Block-Cyclic Distribution Snippet

	0	1	
0	$a_{11}$ $a_{12}$ $a_{15}$	$a_{13}$ $a_{14}$	:
1	$a_{21}$ $a_{20}$ $a_{25}$	$a_{23}$ $a_{1}$ $a_{24}$	
	$a_{51}$ $a_{52}$ $a_{55}$	$a_{53}$ $a_{54}$	
1	$a_{31}$ $a_{32}$ $a_{35}$	$a_{33}$ $a_{34}$	
	$a_{41}$ $a_{42}$ $a_{45}$	$a_{43}$ $a_{44}$	

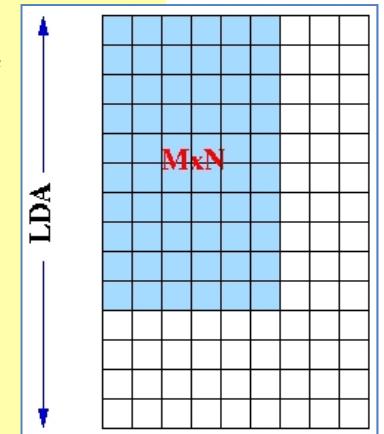
```

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

IF      ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
        A(1) = a11; A(2) = a21; A(3) = a51;
        A(1+LDA) = a12; A(2+LDA) = a22; A(3+LDA) = a52;
        A(1+2*LDA) = a15; A(2+3*LDA) = a25; A(3+4*LDA) = a55;
ELSE IF ( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
        A(1) = a13; A(2) = a23; A(3) = a53;
        A(1+LDA) = a14; A(2+LDA) = a24; A(3+LDA) = a54;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
        A(1) = a31; A(2) = a41;
        A(1+LDA) = a32; A(2+LDA) = a42;
        A(1+2*LDA) = a35; A(2+3*LDA) = a45;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
        A(1) = a33; A(2) = a43;
        A(1+LDA) = a34; A(2+LDA) = a44;
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 array descriptors).



Array descriptor for  
A (see next slides)

# 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 DTTYPE\_ (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 DTTYPE_A=501 for 1 x P <sub>c</sub> 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: error handling

---

- Driver and computational routines perform *global* and *local* input error-checking
  - Global checking → synchronization
  - Local checking → validity
- No input error-checking is performed on the auxiliary routines
- If an error is detected in a PBLAS or BLACS program execution stops

# ScaLAPACK: debugging hints

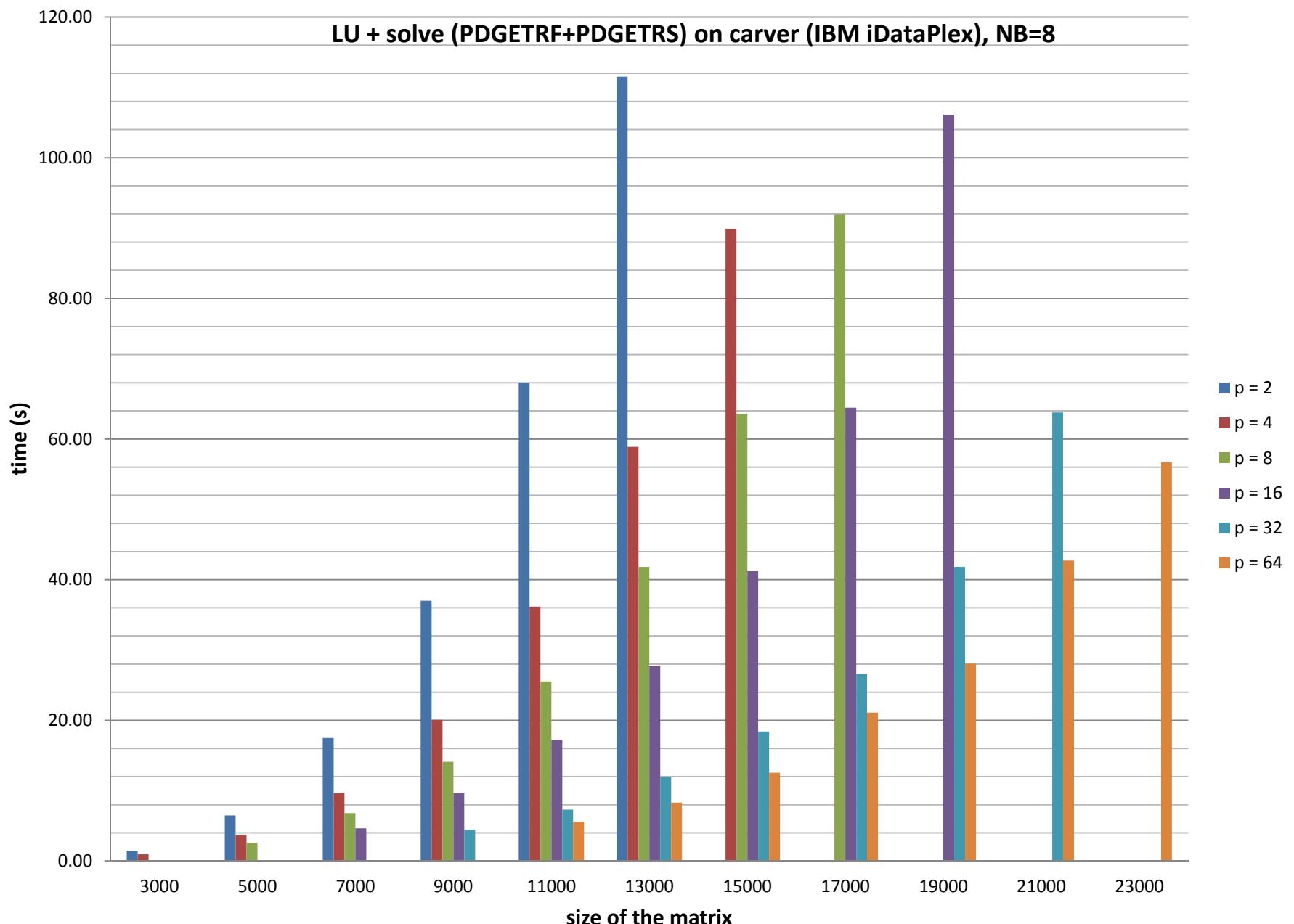
---

- Look at ScaLAPACK example programs.
- Always check the value of INFO on exit from a ScaLAPACK routine.
- Query for size of workspace, LWORK = -1.
- Link to the Debug Level 1 BLACS (specified by BLACSDBGLVL=1 in Bmake.inc).
- Consult errata files on *netlib*:  
<http://www.netlib.org/scalapack/errata.scalapack>  
<http://www.netlib.org/blacs/errata.blacs>

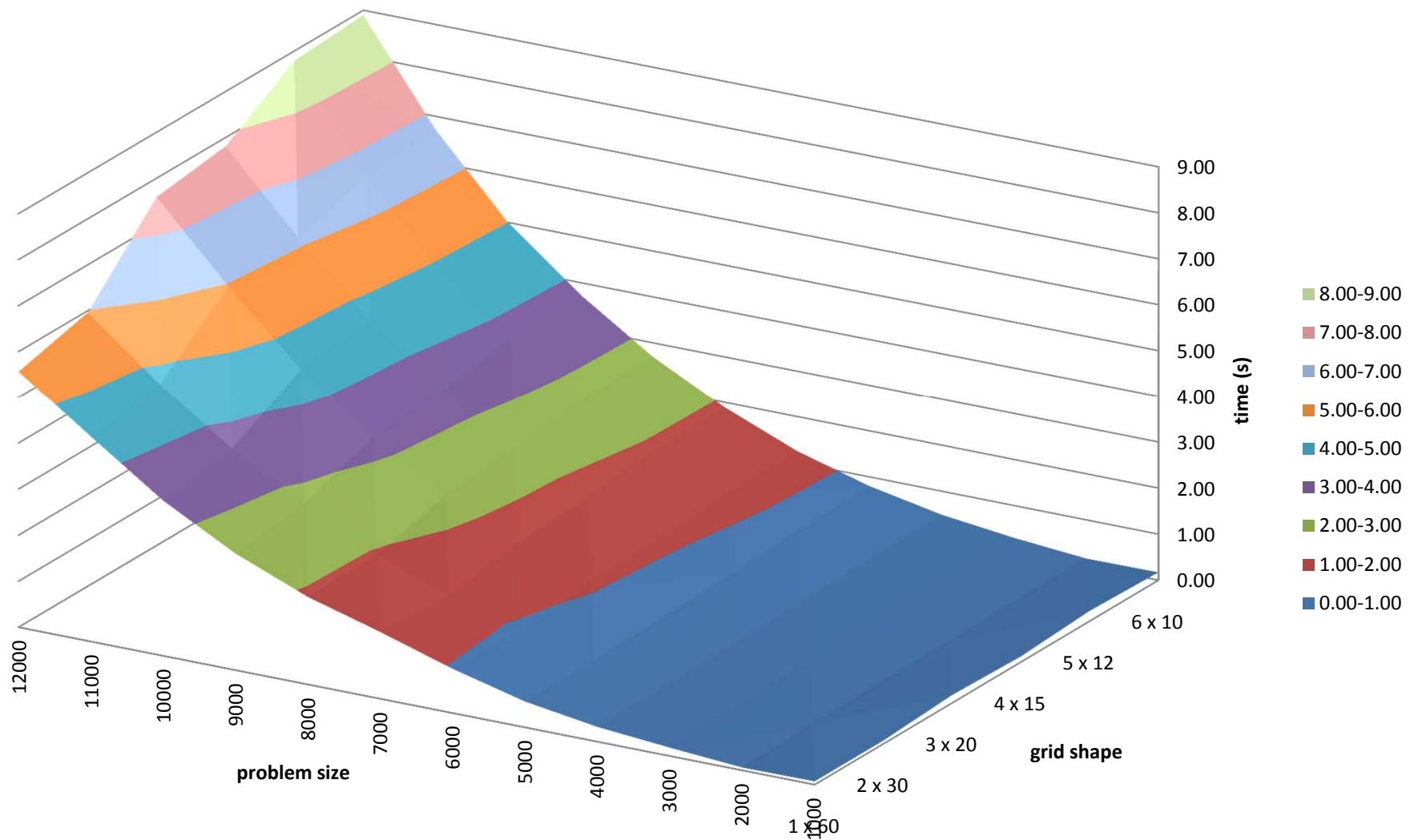
# ScaLAPACK: performance

---

- 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:  $P=MxN/10^6$  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



## LU + solve (PDGETRF+PDGETRS) for different grids on carver, NB=8



# ScaLAPACK: Commercial Use

---

ScaLAPACK has been incorporated in the following packages:

- Fujitsu
- Hewlett-Packard
- Hitachi
- IBM Parallel ESSL
- NAG Numerical Library
- Cray LIBSCI
- NEC Scientific Software Library
- Sun Scientific Software Library
- Visual Numerics (IMSL)

# ScaLAPACK: Development Team

---

- Susan Blackford, UTK
- Jaeyoung Choi, Soongsil University
- Andy Cleary, LLNL
- Ed D'Azevedo, ORNL
- Jim Demmel, UCB
- Inderjit Dhillon, UT Austin
- Jack Dongarra, UTK
- Ray Fellers, LLNL
- Sven Hammarling, NAG
- Greg Henry, Intel
- Sherry Li, LBNL
- Osni Marques, LBNL
- Caroline Papadopoulos, UCSD
- Antoine Petitet, UTK
- Ken Stanley, UCB
- Francoise Tisseur, Manchester
- David Walker, Cardiff
- Clint Whaley, UTK
- Julien Langou, UTK
- :

## Related Projects

---

- PLAPACK (Parallel Linear Algebra Package)  
*[www.cs.utexas.edu/~plapack](http://www.cs.utexas.edu/~plapack)*
- MAGMA (Matrix Algebra on GPU and Multicore Architectures)  
*<http://icl.eecs.utk.edu/magma>*
- PLASMA (Parallel Linear Algebra for Scalable Multi-core Architectures)  
*<http://icl.cs.utk.edu/plasma>*

# Hands-on: <http://acts.nersc.gov/scalapack/hands-on>

---

## Hands-On Exercises for ScaLAPACK

The ScaLAPACK Team

August 2010

### 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 is assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

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 PBLAS and ScaLAPACK.

The instructions for the exercises assume that the underlying system is an IBM SP; using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. These hands-on exercises were prepared in collaboration with 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: [PBLAS Example](#)

Exercise 4: [ScaLAPACK - Example Program 1](#)

Exercise 5: [ScaLAPACK - Example Program 2](#)

Other: A collection of [additional exercises](#) and examples.

[Download all exercises \(tar file\)](#)

### Addition Resources:

- [Block Cyclic Data Distribution](#)
- Detailed information on the [BLACS](#)
- Detailed information on the [PBLAS](#)
- Detailed information on [ScaLAPACK](#) and more [examples](#)
- [Useful calling sequences](#)

[Scalapack](#)

[Tools](#)

[Home](#)

# ScaLAPACK Hands-on: code samples

---

- Example 1: BLACS, “hello world” example
- Example 2: BLACS, “pi” example
- Example 3: PBLAS example
- Example 4: ScaLAPACK example 1 (PSGESV)
- Example 5: ScaLAPACK example 2 (PSGESV)
- Additional exercises
  - example1.f, example2.f, and example3.f: show how to generate a ScaLAPACK matrix
  - pddttrdrv.c (pddttrdrv.f): shows how to use PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations  $Tx = b$
  - pdpttr\_2.c (pdpttr\_2.f): shows how to use PDPTTRF and PPPTTRS to factor and solve a symmetric positive definite tridiagonal system of linear equations  $Tx = b$ , in two distinct contexts.
  - pdgesvddrv.f: reads a (full) matrix A from a file, distributes A among the available processors and then calls PDGESVD to compute the SVD of A,  $A = U * S * V^T$
  - pzdtt\_col\_major.c: shows how to use PZDTTRF and PZDTTRS to factor and solve a tridiagonal system of linear equations in nprow distinct contexts, column-major ordering.
  - pzdtt\_row\_major.c: shows how to use PZDTTRF and PZDTTRS to factor and solve a tridiagonal system of linear equations in nprow distinct contexts, row-major ordering.

# Contents of hands-on/etc

```
NERSC (on cvrsvc04)
[/global/homes/o/osni/work/hands-on/etc] ls -l
total 164
8 drwx--- 2 osni osni 4096 Aug 13 14:45 .
8 drwx--- 8 osni osni 4096 Aug 12 17:23 ..
4 -rwx--- 1 osni osni 2906 Aug 12 16:35 A.SVD
4 -rwx--- 1 osni osni 2700 Aug 12 16:35 A.dat
4 -rwx--- 1 osni osni 3065 Aug 12 16:35 Makefile
4 -rwx--- 1 osni osni 1502 Aug 12 16:35 README
4 -rwx--- 1 osni osni 1767 Aug 12 16:35 dcomplex.h
4 -rwx--- 1 osni osni 373 Aug 12 16:35 desc.h
8 -rwx--- 1 osni osni 4831 Aug 12 16:35 example1.f
8 -rwx--- 1 osni osni 6091 Aug 12 16:35 example2.f
12 -rwx--- 1 osni osni 10093 Aug 12 16:35 example3.f
4 -rwx--- 1 osni osni 3995 Aug 12 16:35 pddttrdrv.c
4 -rwx--- 1 osni osni 3749 Aug 12 16:35 pddttrdrv.f
4 -rwx--- 1 osni osni 365 Aug 12 16:35 pddttrdrv.ll
4 -rwx--- 1 osni osni 172 Aug 12 16:35 pddttrdrv.pbs
4 -rwx--- 1 osni osni 13 Aug 12 16:35 pdgesvddrv.dat
8 -rwx--- 1 osni osni 7119 Aug 12 16:35 pdgesvddrv.f
4 -rwx--- 1 osni osni 369 Aug 12 16:35 pdgesvddrv.ll_2
4 -rwx--- 1 osni osni 369 Aug 12 16:35 pdgesvddrv.ll_4
4 -rwx--- 1 osni osni 180 Aug 12 17:34 pdgesvddrv.pbs_4
4 -rwx--- 1 osni osni 176 Aug 12 17:25 pdgesvddrv.pbs_6
8 -rwx--- 1 osni osni 6379 Aug 12 16:35 pdpttr_2.c
8 -rwx--- 1 osni osni 4398 Aug 12 16:35 pdpttr_2.f
4 -rwx--- 1 osni osni 361 Aug 12 16:35 pdpttr_2.ll
8 -rwx--- 1 osni osni 4819 Aug 12 16:35 pdttt_col_major.c
4 -rwx--- 1 osni osni 389 Aug 12 16:35 pdttt_col_major.ll
4 -rwx--- 1 osni osni 196 Aug 12 17:25 pdttt_col_major.pbs
8 -rwx--- 1 osni osni 4835 Aug 12 16:35 pdttt_row_major.c
4 -rwx--- 1 osni osni 389 Aug 12 16:35 pdttt_row_major.ll
4 -rwx--- 1 osni osni 196 Aug 12 17:25 pdttt_row_major.pbs
[/global/homes/o/osni/work/hands-on/etc]
```

*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 *qsub pddttrdrv.pbs*.

*pdpttr\_2.c (pdpttr\_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 *qsub pdpttr.pbs\_4*.

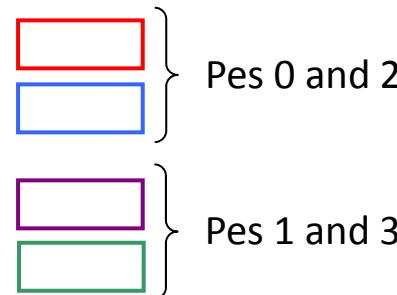
*pdgesvddrv.f*: reads a (full) matrix  $A$  from a file, distributes  $A$  among the available processors and then call the ScaLAPACK subroutine PDGESVD to compute the SVD of  $A$ ,  $A=USV^T$ . It requires the file *pdgesvddrv.dat*, which should contain: line 1, the name of the file where  $A$  will 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  $m=7, n=10$ :  $\text{diag}(S)=[ 4.5096 \ 1.1333 \ 1.0569 \ 0.8394 \ 0.8108 \ 0.5405 \ 0.2470 ]$

# Data distribution for *pdpttr\_2.c* (*pdpttr\_2.f*)

---



1.8180	0.8385
0.8385	1.6602
0.5681	0.5681
0.3704	1.3420
0.3704	0.3704
0.7027	1.2897
0.7027	0.7027
0.5466	1.3412
0.5466	0.5466
0.4449	1.5341
0.4449	0.4449
0.6946	1.7271
0.6946	0.6946
	1.3093

$$\begin{matrix}
 & & 0 & 2 & 0 & 2 \\
 & & \left[ \begin{array}{c} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ x_4^{(1)} \\ x_5^{(1)} \\ x_6^{(1)} \\ x_7^{(1)} \\ x_8^{(1)} \end{array} \right] & \left[ \begin{array}{c} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \\ x_5^{(2)} \\ x_6^{(2)} \\ x_7^{(2)} \\ x_8^{(2)} \end{array} \right] & = & \left[ \begin{array}{cc} 1 & 8 \\ 2 & 7 \\ 3 & 6 \\ 4 & 5 \\ 5 & 4 \\ 6 & 3 \\ 7 & 2 \\ 8 & 1 \end{array} \right] \\
 & & 1 & 3 & 1 & 3
 \end{matrix}$$

# Block Cyclic Distribution

Consider the 12-by-10 matrix:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & a_{1,9} & a_{1,10} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} & a_{2,9} & a_{2,10} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} & a_{3,9} & a_{3,10} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & a_{4,9} & a_{4,10} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & a_{5,9} & a_{5,10} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} & a_{6,9} & a_{6,10} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} & a_{7,9} & a_{7,10} \\ a_{8,1} & a_{8,2} & a_{8,3} & a_{8,4} & a_{8,5} & a_{8,6} & a_{8,7} & a_{8,8} & a_{8,9} & a_{8,10} \\ a_{9,1} & a_{9,2} & a_{9,3} & a_{9,4} & a_{9,5} & a_{9,6} & a_{9,7} & a_{9,8} & a_{9,9} & a_{9,10} \\ a_{10,1} & a_{10,2} & a_{10,3} & a_{10,4} & a_{10,5} & a_{10,6} & a_{10,7} & a_{10,8} & a_{10,9} & a_{10,10} \\ a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} & a_{11,5} & a_{11,6} & a_{11,7} & a_{11,8} & a_{11,9} & a_{11,10} \\ a_{12,1} & a_{12,2} & a_{12,3} & a_{12,4} & a_{12,5} & a_{12,6} & a_{12,7} & a_{12,8} & a_{12,9} & a_{12,10} \end{bmatrix}$$

Do the following block cyclic distributions:

- 3-by-3 blocking on a 3-by-2 process grid
- 4-by-4 blocking on a 2-by-3 process grid

Use <http://acts.nersc.gov/scalapack/hands-on/datadist.html> to compare

# 2D Block-Cyclic Distribution

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

The image shows two screenshots of the ScalAPACK Block Cyclic Data Distribution tool. On the left, the 'Block Cyclic Data Distribution' window displays input fields for matrix dimensions (9x9), matrix blocking (rows per block: 2, columns per block: 2), and process grid (rows: 2, columns: 3). A large blue arrow points to the right, leading to the 'Block Cyclic Data Distribution' results window. This window shows a table of array values for 16 processes (0 to 15) and a corresponding 4x4 color-coded distribution grid. The color scheme indicates the distribution of global array elements across the grid. The grid shows a repeating pattern of colors: blue, magenta, yellow, and purple, representing different global array segments assigned to each process.

Process (coordinates)	Array Values
(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]
(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]
(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]
(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]
(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]
(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]

# Hands-on: compiling and running on carver

---

```
% cp /usr/common/acts/scalapack.hands-on.tgz .
% tar zxvf scalapack.hands-on.tgz
% cd hands-on/exercise4
% module load scalapack mkl
% make
% qsub -l -V -q interactive -l nodes=1:ppn=6 -l walltime=00:05:00
qsub: waiting for job 3098004.cvrsvc09-ib to start
qsub: job 3098004.cvrsvc09-ib ready
% cd $PBS_O_WORKDIR
% mpirun -np 6 ./pdgesvdriver.x
% exit
logout
qsub: job 3098009.cvrsvc09-ib completed
```

[DO ONLY ONCE]

[WAIT FOR OUTPUT]

Alternatively

```
% qsub pdgesvdriver.pbs.carver
```

[LOOK FOR \*.OUT]