

ScaLAPACK

Osni Marques

Lawrence Berkeley National Laboratory (LBNL)

OAMarques@lbl.gov

ScaLAPACK: Functionalities

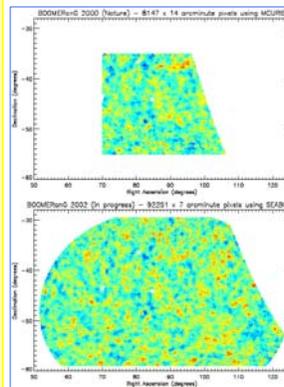
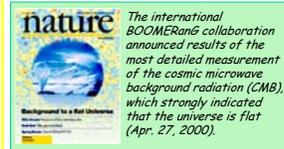
$Ax = b$	Simple Driver	Expert Driver	Factor	Solve	Inversion	Conditioning Estimator	Iterative Refinement
Triangular				x	x	x	x
SPD	x	x	x	x	x	x	x
SPD Banded	x		x	x			
SPD Tridiagonal	x		x	x			
General	x	x	x	x	x	x	x
General Banded	x		x	x			
General Tridiagonal	x		x	x			
Least Squares	x		x	x			
GQR			x	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

Outline

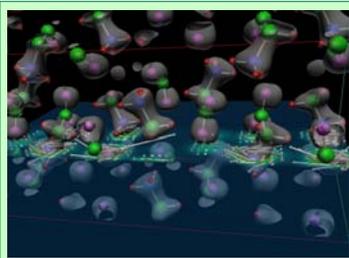
- Motivation
- ScaLAPACK: *software structure*
 - Basic Linear Algebra Subprograms (BLAS)
 - Linear Algebra PACKage (LAPACK)
 - Basic Linear Algebra Communication Subprograms (BLACS)
 - Parallel BLAS (PBLAS)
- ScaLAPACK: *details*
 - Data layout
 - Array descriptors
 - Error handling
 - Performance
- Hands-on

Application: 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.

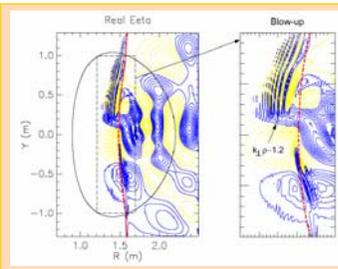
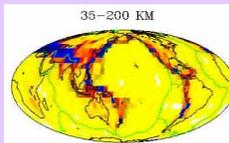


ScaLAPACK: Applications



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)

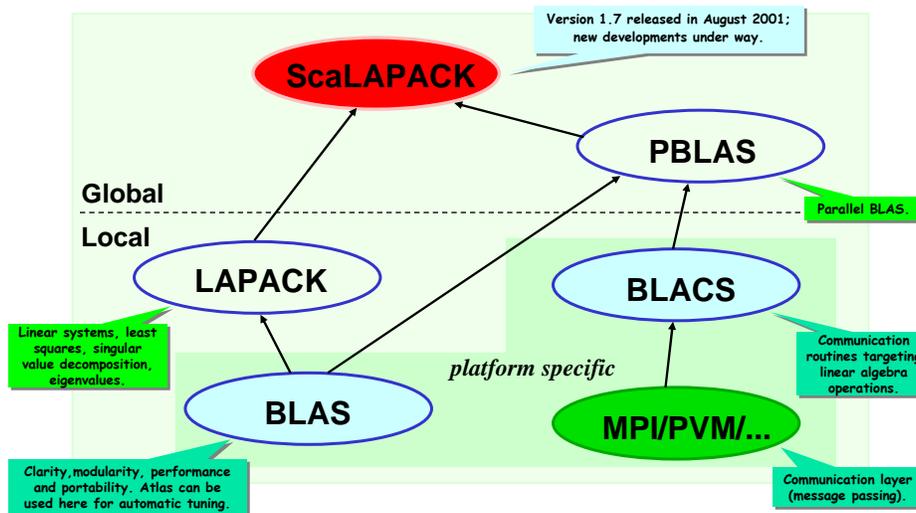


Two ScaLAPACK routines, PZGTRF and PZGTRS, are used for solution of linear systems in the spectral algorithms based AORSA code (Batchelor et al.), which is intended for the study of electromagnetic wave-plasma interactions. The code reaches 68% of peak performance on 1936 processors of an IBM SP.

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 and afterwards actively used ACTS tools, in particular ScaLAPACK. Dario has worked 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 has reported that this work could not be carried out in sequential machines and that ScaLAPACK was fundamental for the parallelization of these codes.

ScaLAPACK: software structure

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



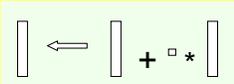
BLAS

(*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).

BLAS: 3 levels

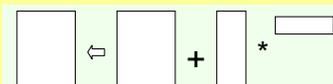
- Level 1 BLAS: vector-vector



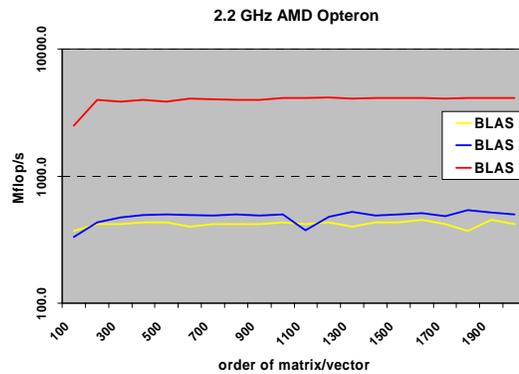
- Level 2 BLAS: matrix-vector



- Level 3 BLAS: matrix-matrix



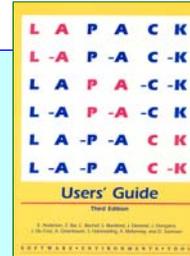
- Clarity
- Portability
- Performance: development of blocked algorithms is important for performance!



LAPACK: main features

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

- Linear Algebra library written in Fortran 77 (Fortran 90)
- Combine algorithms from LINPACK and EISPACK into a single package.
- Efficient on a wide range of computers (RISC, Vector, SMPs).
- 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 ...).



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*

- 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	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 )
```

xx (Data type)	xx (Matrix type)
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 )
```

SCOPE	TOP
'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 )
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.

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, ...)

BLAS

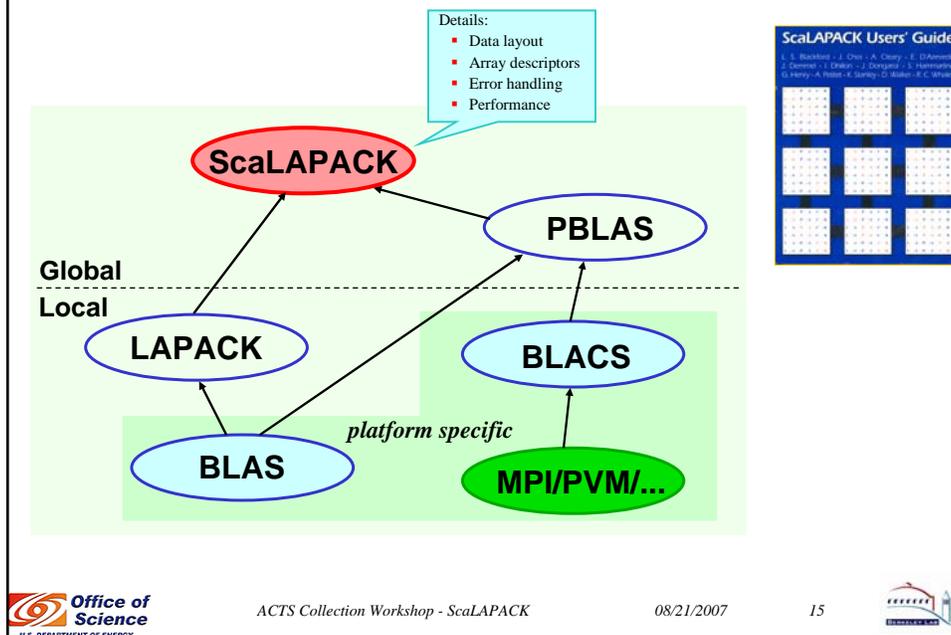
CALL PDGEXXX(M, N, A, IA, JA, DESCA, ...)

PBLAS



Array descriptor
(see next slides)

ScaLAPACK: structure of the software

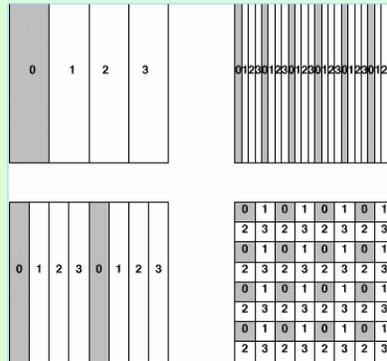


ScaLAPACK: goals

- 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

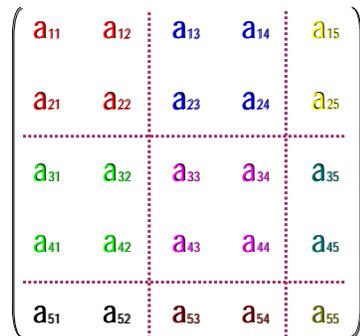
- 1D block and cyclic column distributions



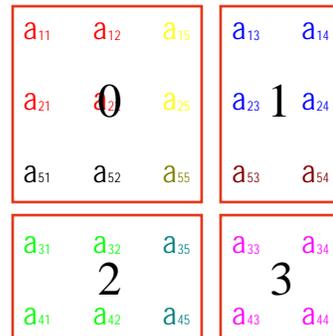
- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

ScaLAPACK: 2D Block-Cyclic Distribution

5x5 matrix partitioned in 2x2 blocks



2x2 process grid point of view



2D Block-Cyclic Distribution

0	1	:
a ₁₁	a ₁₂	a ₁₃
a ₂₁	a ₂₂	a ₂₃
a ₃₁	a ₃₂	a ₃₃
a ₄₁	a ₄₂	a ₄₃

```

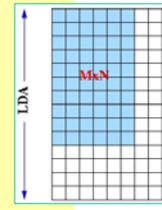
CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

IF ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
  A(1) = a11; A(2) = a21; A(3) = a31;
  A(1+LDA) = a12; A(2+LDA) = a22; A(3+LDA) = a32;
  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) = a33;
  A(1+LDA) = a14; A(2+LDA) = a24; A(3+LDA) = a34;
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( JOB, JOEVT, 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 slides 23-26)

Array descriptor for A (see slides 23-26)



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>

Block Cyclic Data Distribution

The block cyclic data distribution used by ScaLAPACK is useful for the solution of many algorithms intended for linear algebra calculations and by the goal of achieving good load balancing. See The Science of Linear Algebra Calculations for High Performance Computing, by J. Demmel and D. Walker, and Scientific Computing: Linear Algebra, by J. Demmel, M. Heath, and H. van der Vorst.

Use the data below to specify the dimensions of a matrix, the desired blocking, and the process grid configuration. This click on "Generate Data" a new window will pop up with the corresponding block cyclic distribution. The acceptable values are indicated following parentheses, the defaults correspond to those of the matrix A used in Example 2 in the [Introduction](#).

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 (0, > 0)
 number of columns (0, > 0)

matrix blocking: rows per block (0)
 columns per block (0)

process grid: rows (0)
 columns (0)

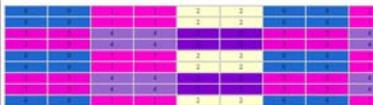
➔

Block Cyclic Data Distribution

The table lists the data on each process. A = global array, B = local array, asterisk means not at 1.

Process (row/col)	Array Values
0 (0,0)	B(1:10,1:10) B(11:20,1:10) B(21:30,1:10) B(31:40,1:10) B(41:50,1:10) B(51:60,1:10) B(61:70,1:10) B(71:80,1:10) B(81:90,1:10) B(91:100,1:10)
1 (0,1)	B(1:10,11:20) B(11:20,11:20) B(21:30,11:20) B(31:40,11:20) B(41:50,11:20) B(51:60,11:20) B(61:70,11:20) B(71:80,11:20) B(81:90,11:20) B(91:100,11:20)
2 (1,0)	B(1:10,61:70) B(11:20,61:70) B(21:30,61:70) B(31:40,61:70) B(41:50,61:70) B(51:60,61:70) B(61:70,61:70) B(71:80,61:70) B(81:90,61:70) B(91:100,61:70)
3 (1,1)	B(1:10,71:80) B(11:20,71:80) B(21:30,71:80) B(31:40,71:80) B(41:50,71:80) B(51:60,71:80) B(61:70,71:80) B(71:80,71:80) B(81:90,71:80) B(91:100,71:80)

The color column shows the distribution of the global array on the computational grid. For the sake of resolution, colors are only displayed for less than 10 processes.



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_0	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 P _c 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 P _r 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 routine program execution stops.

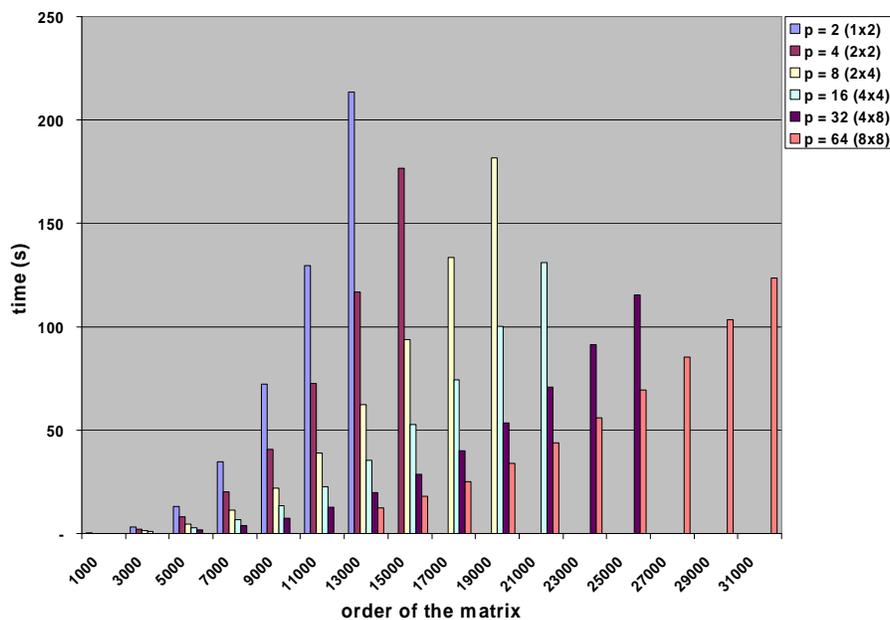
ScaLAPACK: debugging hints

- 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 BLACSDBGVL=1 in Bmake.inc).
- Consult errata files on *netlib*:
<http://www.netlib.org/scalapack/errata.scalapack>
<http://www.netlib.org/blacs/errata.blacs>

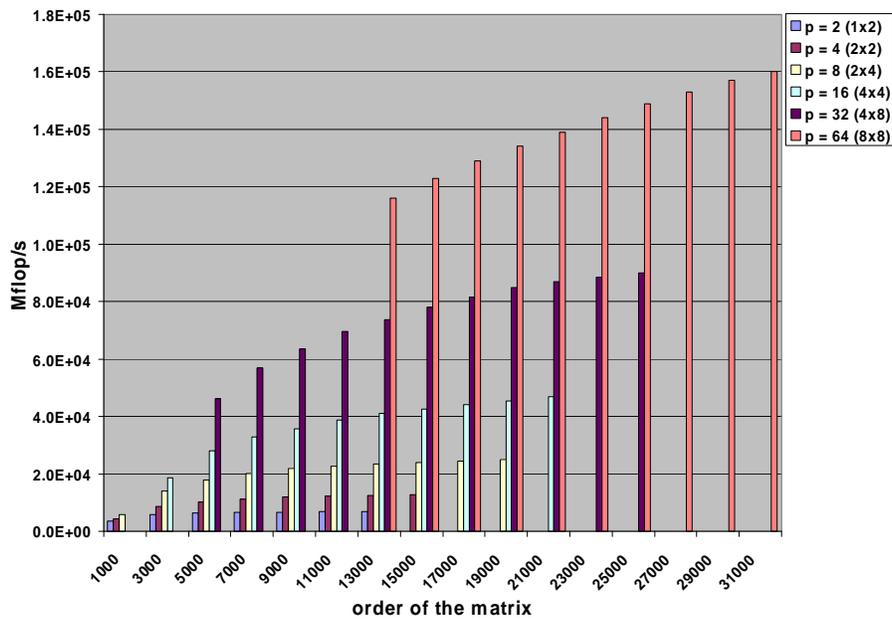
ScaLAPACK: Performance

- 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: $P=M \times N/10^6$ for an $M \times N$ 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: $P_{row} = P_{column}$.

LU on 2.2 GHz AMD Opteron (4.4 GFlop/s peak performance)

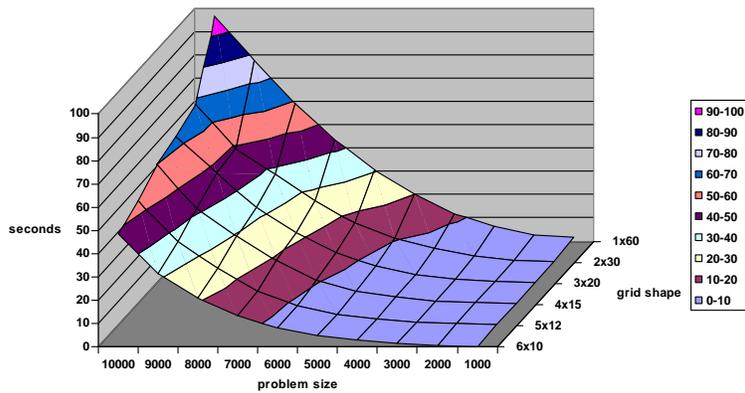


LU+solve on 2.2 GHz AMD Opteron (4.4 GFlop/s peak performance)



ScaLAPACK: Grid Size Effects

Execution time of PDGESV for various grid shape



60 processors, Dual AMD Opteron 1.4GHz Cluster with Myrinet Interconnect, 2GB Memory

ScaLAPACK: Commercial Use

ScaLAPACK has been incorporated in the following commercial 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 Petit, UTK
- Ken Stanley, UCB
- Françoise Tisseur, Manchester
- David Walker, Cardiff
- Clint Whaley, UTK
- Julien Langou, UTK
- ⋮

ScaLAPACK: Summary

- Library of high performance dense linear algebra routines for distributed-memory computing
- Reliability, scalable and portable
- Calling interface similar to LAPACK
- New developments on the way
- LAPACK/ScaLAPACK Forum: <http://icl.cs.utk.edu/lapack-forum>

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

Hands-On Exercises for ScaLAPACK

The ScaLAPACK Team
June 2007

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 - P1 Example](#)

[Exercise 3: PBLAS Example](#)

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

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

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

Additional Resources:

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

[ScaLAPACK](#)

[Tools](#)

[Home](#)

Hands-on: instructions

- Do a “`cp -r /usr/common/acts/SCALAPACK/hands-on hands-on`”.
- There are six subdirectories under *hands-on*:
 - 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
- Examples 1-5 are written in Fortran. For a successful compilation and execution of Example 5, you will have to correct some lines in the code, in particular the lines starting with `***` (commented lines).
- Examples 1-5 can be compiled with “*make*”, which will generate an executable file with “.x”.
- Try also <http://acts.nersc.gov/scalapack/hands-on/datadist.html> with a bigger matrix and different block/grid sizes.

Contents of hands-on/variou

```

[~/usr/common/acts/SCALAPACK/hands-on/variou] ls -ls
total 184
3 -rw-r----- 1 osni  acts      2906 Aug 17 16:22 a.SVD
3 -rw-r----- 1 osni  acts      2700 Aug 17 16:22 a.dat
3 -rw-r----- 1 osni  acts      1592 Aug 17 16:22 README
3 -rw-r----- 1 osni  acts      1767 Aug 17 16:22 dcomplex.h
3 -rw-r----- 1 osni  acts       373 Aug 17 16:22 desc.h
3 -rw-r----- 1 osni  acts      4785 Aug 17 16:22 example1.f
3 -rw-r----- 1 osni  acts      6045 Aug 17 16:22 example2.f
10 -rw-r----- 1 osni  acts     10039 Aug 17 16:22 example3.f
3 -rw-r----- 1 osni  acts     3953 Aug 17 16:22 pddttrdrv.c
3 -rw-r----- 1 osni  acts     3705 Aug 17 16:22 pddttrdrv.f
3 -rw-r----- 1 osni  acts       365 Aug 17 16:22 pddttrdrv.ll
3 -rw-r----- 1 osni  acts       13 Aug 17 16:22 pdgesvdrv.dat
3 -rw-r----- 1 osni  acts     6951 Aug 17 16:22 pdgesvdrv.f
3 -rw-r----- 1 osni  acts       369 Aug 17 16:22 pdgesvdrv.ll_3
3 -rw-r----- 1 osni  acts       369 Aug 17 16:22 pdgesvdrv.ll_4
3 -rw-r----- 1 osni  acts     6160 Aug 17 16:22 pdptr_2.c
3 -rw-r----- 1 osni  acts     4234 Aug 17 16:22 pdptr_2.f
3 -rw-r----- 1 osni  acts       361 Aug 17 16:22 pdptr_2.ll
3 -rw-r----- 1 osni  acts     4690 Aug 17 16:22 pzdt_col_major.c
3 -rw-r----- 1 osni  acts       389 Aug 17 16:22 pzdt_col_major.ll
3 -rw-r----- 1 osni  acts     4616 Aug 17 16:22 pzdt_row_major.c
3 -rw-r----- 1 osni  acts       389 Aug 17 16:22 pzdt_row_major.ll
[~/usr/common/acts/SCALAPACK/hands-on/variou]
    
```

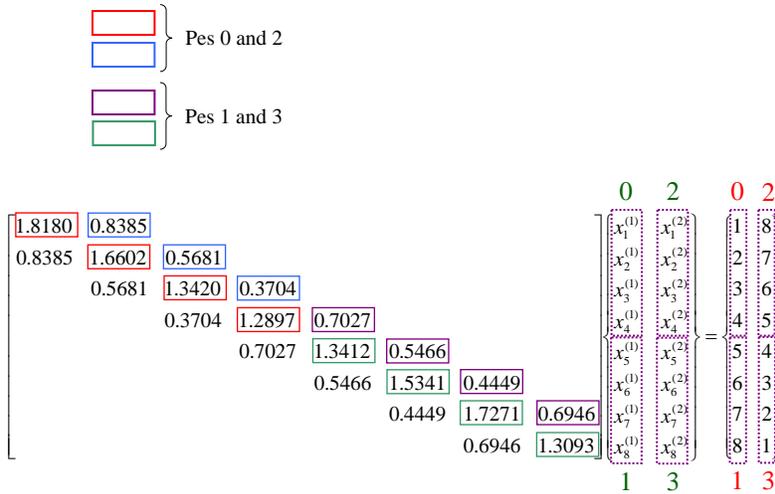
pddttrdrv.c (*pddttrdrv.f*): illustrates the use of the ScaLAPACK routines PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations $Tx = b$. After compilation, it can be executed with `lsubmit pddttrdrv.ll`.

pdptr_2.c (*pdptr_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 `lsubmit pdptr_2.ll`.

pdgesvdrv.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 *pdgesvdrv.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 $m=10, n=7$: `diag(S)=[4.4926 1.4499 0.8547 0.8454 0.6938 0.4332 0.2304]`
- if $m=7, n=10$: `diag(S)=[4.5096 1.1333 1.0569 0.8394 0.8108 0.5405 0.2470]`

Data distribution for *pdptr_2.c* (*pdptr_2.f*)



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

Hands-On Exercises for ScaLAPACK

Exercise 1: BLACS - Hello World Example

Information: This is a simple "Hello World" program using BLACS routines. Each process is taken and formed into a process grid. Then, each processes check in with the process at (0,0) in the process grid.

Exercises: For these exercises, use the file [hello.f](#). You should compile it with [make](#) (you may also need to perform minor modifications in the file [make.inc](#)).

- To execute the program on the IBM SP, type *poe hello.x -procs 4*
- Once you're certain that the code is running properly, make note of the output
- Try to run *hello.x* using a differentt value for *procs*

Questions:

- Do you understand what this program is doing?
- Can you see that BLACS can be used in much more powerful examples?

[Hands-on](#)[ScaLAPACK](#)[Tools](#)[Home](#)

Hands-On Exercises for ScaLAPACK

Exercise 2: BLACS - Pi Example

Information: This is a program that calculates the value of pi using BLACS routines. Using numerical integration combined with parallel processing, this program calculates the value of pi and the time it takes for computation. The user may select the number of processes and points of integration.

Exercises: For these exercises, use the file [pi.f](#). You should compile it with [make](#) (you may also need to perform minor modifications in the file [make.inc](#)).

- To execute the program on the IBM SP, type `poe pi.x -procs 4`
- Once you're certain that the code is running properly, make note of the output.

Questions:

- Do you understand what this program is doing?
- What happens as you increase the number of processes? Decrease?
- What happens as you increase the number of points of integration?

[Hands-on](#)[ScaLAPACK](#)[Tools](#)[Home](#)

Hands-On Exercises for ScaLAPACK

Exercise 3: PBLAS Example

Information: This is a program that tests several PBLAS routines. It generates 3 random 4-by-4 matrices (A, B, and C). Then it tests three PBLAS routine calls (PSNRM2, PSGEMV, and PSGEMM). To learn more about PBLAS and its calls, check the [PBLAS reference guide](#).

Exercises: For this exercise, use the file [pspblasdriver.f](#). You should compile it with [make](#) (you may also need to perform minor modifications in the file [make.inc](#)).

- To execute the program on the IBM SP type *poe* `pspblasdriver.x -procs 4`
- Once you're certain that the code is running properly, make note of the output.

Questions:

- Do you understand what this program is doing?
- What are some of the benefits of using PBLAS?

[Hands-on](#)

[ScaLAPACK](#)

[Tools](#)

[Home](#)

Hands-On Exercises for ScaLAPACK

Exercise 4: ScaLAPACK - Example Program 1

Information: This is a very simple program that solves a linear system by calling the ScaLAPACK routine [PSGESV](#). More complete details of this example program can be found in Chapter 2 of the [ScaLAPACK Users' Guide](#). This example program demonstrates the basic requirements to call a ScaLAPACK routine: initializing the process grid, assigning the matrix to the processes, calling the ScaLAPACK routine, and releasing the process grid.

This example program solves the 9-by-9 system of linear equations given by:

$$\begin{array}{cccccccccc|cccc}
 / & 19 & 3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 & \backslash & / & x1 & \backslash & / & 0 & \backslash \\
 | & -19 & 3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 & | & | & x2 & | & | & 0 & | \\
 | & -19 & -3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 & | & | & x3 & | & | & 1 & | \\
 | & -19 & -3 & -1 & 12 & 1 & 16 & 1 & 3 & 11 & | & | & x4 & | & | & 0 & | \\
 | & -19 & -3 & -1 & -12 & 1 & 16 & 1 & 3 & 11 & | & * & x5 & | & = & 0 & | \\
 | & -19 & -3 & -1 & -12 & -1 & 16 & 1 & 3 & 11 & | & | & x6 & | & | & 0 & | \\
 | & -19 & -3 & -1 & -12 & -1 & -16 & 1 & 3 & 11 & | & | & x7 & | & | & 0 & | \\
 | & -19 & 3 & -1 & -12 & -1 & -16 & -1 & 3 & 11 & | & | & x8 & | & | & 0 & | \\
 \backslash & -19 & -3 & -1 & -12 & -1 & -16 & -1 & -3 & 11 & / & \backslash & x9 & / & \backslash & 0 & /
 \end{array}$$

using the ScaLAPACK driver routine `PSGESV`. The ScaLAPACK routine `PSGESV` solves a system of linear equations $A*X = B$, where the coefficient matrix (denoted by A) and the right-hand-side matrix (denoted by B) are real, general distributed matrices. The coefficient matrix A

is distributed as depicted below and, for simplicity, we shall solve the system for one right-hand side (`NRHS=1`); that is, the matrix B is a vector. The third element of the matrix B is equal to 1, and all other elements are equal to 0. After solving this system of equations, the solution vector X is given by

$$\begin{array}{cccc|cccc}
 / & x1 & \backslash & / & 0 & \backslash \\
 | & x2 & | & | & -1/6 & | \\
 | & x3 & | & | & 1/2 & | \\
 | & x4 & | & | & 0 & | \\
 | & x5 & | & = & 0 & | \\
 | & x6 & | & | & 0 & | \\
 | & x7 & | & | & -1/2 & | \\
 | & x8 & | & | & 1/6 & | \\
 \backslash & x9 & / & \backslash & 0 & /
 \end{array}$$

Let us assume that the matrix A is partitioned and distributed such that we have chosen the row and column block sizes as $MB=NB=2$, and the matrix is distributed on a 2-by-3 process grid ($P_r=2, P_c=3$). The

partitioning and distribution of our example matrix A is represented in the two figures below, where, to aid visualization, we use the notation $s=19$, $c=3$, $a=1$, $l=12$, $p=16$, and $k=11$.

Partitioning of global matrix A :

s	c		a	l		a	p		a	c		k
$-s$	c		a	l		a	p		a	c		k
-----+												
$-s$	$-c$		a	l		a	p		a	c		k
$-s$	$-c$		$-a$	l		a	p		a	c		k
-----+												
$-s$	$-c$		$-a$	$-l$		a	p		a	c		k
$-s$	$-c$		$-a$	$-l$		$-a$	p		a	c		k
-----+												
$-s$	$-c$		$-a$	$-l$		$-a$	$-p$		a	c		k
$-s$	c		$-a$	$-l$		$-a$	$-p$		$-a$	c		k
-----+												
$-s$	$-c$		$-a$	$-l$		$-a$	$-p$		$-a$	$-c$		k

Mapping of matrix A

onto process grid ($P_r=2$, $P_c=3$). Note, for example, that process (0,0) contains a local array of size $A(5,4)$.

	0		1		2									
	s	c		a	c		a	l		k		a	p	
	$-s$	c		a	c		a	l		k		a	p	
	-----+													
	$-s$	$-c$		a	c		$-a$	$-l$		k		a	p	0
	$-s$	$-c$		a	c		$-a$	$-l$		k		$-a$	p	
	-----+													
	$-s$	$-c$		$-a$	$-c$		$-a$	$-l$		k		$-a$	$-p$	
	-----+													
	$-s$	$-c$		a	c		a	l		k		a	p	1
	$-s$	$-c$		a	c		$-a$	l		k		a	p	
	-----+													
	$-s$	$-c$		a	c		$-a$	$-l$		k		$-a$	$-p$	
	$-s$	c		$-a$	c		$-a$	$-l$		k		$-a$	$-p$	

The partitioning and distribution of our example matrix B are demonstrated in the figure below. Note that the matrix B is distributed only in column 0 of the process grid. All other columns in the process grid possess an empty local portion of the matrix B .

Mapping of matrix B onto process grid ($P_r=2$, $P_c=3$):

	0	1	2
b1			
b2			

b5			0
b6			

b9			
-----	+	-----	
b3			
b4			
-----			1
b7			
b8			

On exit from PSGESV, process **(0,0)** contains (in the global view) the global vector X and (in the local view) the local array B given by

$$\begin{array}{l}
 / \ x1 \ \backslash \\
 | \ x2 \ | \\
 | \ x5 \ | \\
 | \ x6 \ | \\
 \backslash \ x9 \ /
 \end{array}
 \begin{array}{l}
 / \ b1 \ \backslash \\
 | \ b2 \ | \\
 | \ b5 \ | \\
 | \ b6 \ | \\
 \backslash \ b9 \ /
 \end{array}
 =
 \begin{array}{l}
 / \ 0 \ \backslash \\
 | \ -1/6 \ | \\
 | \ 0 \ | \\
 | \ 0 \ | \\
 \backslash \ 0 \ /
 \end{array}$$

and process **(1,0)** contains (in the global view) the global vector X and (in the local view) local array B given by

$$\begin{array}{l}
 / \ x3 \ \backslash \\
 | \ x4 \ | \\
 | \ x7 \ | \\
 \backslash \ x8 \ /
 \end{array}
 \begin{array}{l}
 / \ b3 \ \backslash \\
 | \ b4 \ | \\
 | \ b7 \ | \\
 \backslash \ b8 \ /
 \end{array}
 =
 \begin{array}{l}
 / \ 1/2 \ \backslash \\
 | \ 0 \ | \\
 | \ -1/2 \ | \\
 \backslash \ 1/6 \ /
 \end{array}$$

The normalized residual check

$$\frac{\| A*x - b \|}{(\| x \| * \| A \| * \text{eps} * N)}$$

where eps is the machine precision and N is the dimension of the matrix, is performed on the solution to verify the accuracy of the results.

Simplifying Assumptions Used in Example Program. Several simplifying assumptions and/or restrictions have been made in this example program in order to present the most

basic example for the user:

1. We have chosen a small block size, $MB=NB=2$; however, this should not be regarded as a typical choice of block size in a user's application. For best performance, a choice of $MB=NB=32$ or $MB=NB=64$ is more suitable. Refer to Chapter 5 of the [ScaLAPACK Users' Guide](#) for further details.
2. A simplistic subroutine `MATINIT` is used to assign matrices A and B to the process grid. Note that this subroutine hardcodes the local arrays on each process and does not perform communication. It is not a ScaLAPACK routine and is provided only for the purposes of this example program.
3. We assume $RSRC=CSRC=0$, and thus both matrices A and B are distributed across the process grid starting with process $(0,0)$. In general, however, any process in the current process grid can be assigned to receive the first element of the distributed matrix.
4. We have set the local leading dimension of local array A and the local leading dimension of local array B to be the same over all process rows in the process grid. The variable `MXLLDA` is equal to the maximum local leading dimension for array A (denoted `LLD_A`) over all process rows. Likewise, the variable `MXLLDB` is the maximum local leading dimension for array B (denoted `LLD_B`) over all process rows. In general, however, the local leading dimension of the local array can differ from process to process in the process grid.
5. The system is solved by using the entire matrix A , as opposed to a submatrix of A , so the global indices, denoted by IA , JA , IB , and JB , into the matrix are equal to 1.

Exercises: For this exercise, use the files [psgesvdriver.f](#). You should compile it with [make](#) (you may also need to perform minor modifications in the file [make.inc](#)).

- To execute the program on an IBM SP, type `poe psgesvdriver.x -procs 6`
- Once you're certain that the code is running properly, compare the results given above..

Questions:

- Do you understand what this program is doing?
- How can this program be made more efficient?
- How would we need to modify this example program if we changed the size of the process grid to 2-by-2?

[Hands-on](#)

[ScaLAPACK](#)

[Tools](#)

[Home](#)

Hands-On Exercises for ScaLAPACK

Exercise 5: ScaLAPACK - Example Program 2

Information: This is a program that solves a linear system by calling the ScaLAPACK routine [PSGESV](#). In this example, the input matrix is read from a [file](#), distributed to the process grid, and the output is written into a file.

Exercises: For this exercise, use the files [psscaex.f](#). In this file, you will find that some of the code has been replaced by `*****`. You should look over this code and replace the `*****` with the proper code. Also, be sure to delete the three asterisks at the beginning of each of these key lines. After editing `psscaex.f`, you should compile it with [make](#) (you may also need to perform minor modifications in the file [make.inc](#)).

- To execute the program on the IBM SP type `poe psscaex.x -procs 6`
- Once you're certain that the code is running properly, make note of the output.
- Possible fixes for `psscaex.f` are given in the file `.psscaex.f`.

Questions:

- Do you understand what this program is doing?
- Is the ScaLAPACK routine, `PSGESV`, used effectively in this program (hopefully you discovered the correct calling position for it)?

[Hands-on](#)[ScaLAPACK](#)[Tools](#)[Home](#)

Hands-On Exercises for ScaLAPACK

Additional examples: This directory contains additional examples illustrating the use of ScaLAPACK functionalities:

- [example1.f](#), [example2.f](#), and [example3.f](#) show how to generate a ScaLAPACK matrix.
- [pddttrdrv.c](#) ([pddttrdrv.f](#)) shows the use of the ScaLAPACK routines PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations $Tx = b$.
- [pdptr_2.c](#) ([pdptr_2.f](#)) shows 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.
- [pdgesvdrv.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 , i.e. $A=U*S*V^T$. It requires the file [pdgesvdrv.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 . If the first 100 rows of the file [A.dat](#) are used to generate a matrix A of dimension 10-by-10 the corresponding SVD is given in the file [A.SVD](#).
- [pzdt_col_major.c](#): shows the use of the ScaLAPACK routines PZDTTRF and PZDTTRS to factor and solve a tridiagonal system of linear equations in n row distinct contexts, column-major ordering.
- [pzdt_rwo_major.c](#): shows the use of the ScaLAPACK routines PZDTTRF and PZDTTRS to factor and solve a tridiagonal system of linear equations in n row distinct contexts, row-major ordering.

[Hands-on](#)[ScaLAPACK](#)[Tools](#)[Home](#)