ScaLAPACK
(An Introduction)

Osni Marques

Lawrence Berkeley National Laboratory (LBNL)

National Energy Scientific Computing Center (NERSC)

(*osni@nersc.gov*, *http://www.nersc.gov/~osni*)

Outline

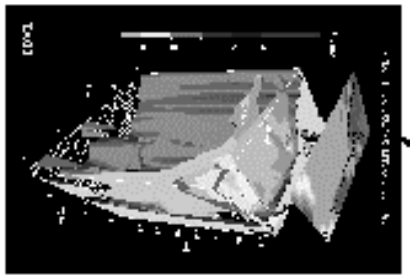
http://acts.nersc.gov/scalapack

- Design of ScaLAPACK
  - Basic Linear Algebra Subprograms (BLAS)
  - Linear Algebra PACKage (LAPACK)
  - Basic Linear Algebra Communication Subprograms (BLACS)
  - Parallel BLAS (PBLAS)
- Contents of ScaLAPACK
- Performance
- Applications
- Hands-on

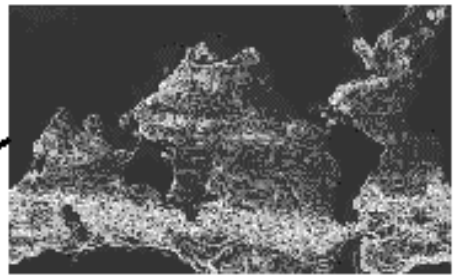
The LAPACK and SCALAPACK Projects

Oak Ridge National Laboratory

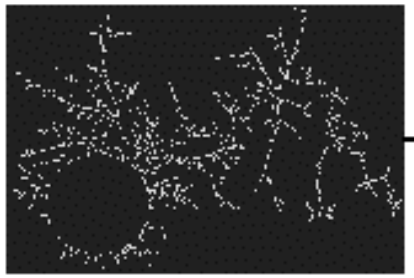
High Quality, High Performance libraries for dense linear algebra



Global Climate Modeling

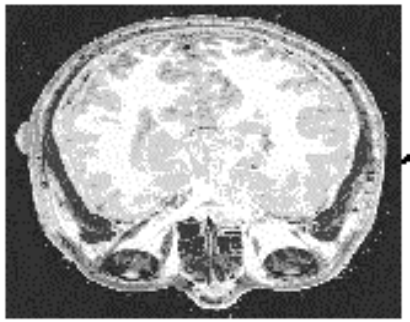


Ocean Modeling

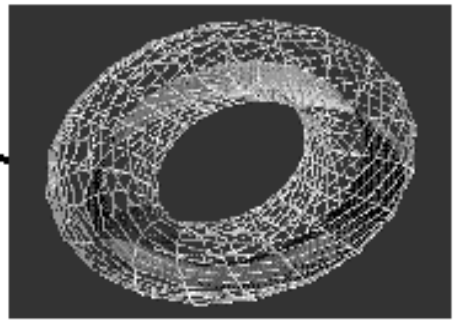


Computational Biology

Computational Fluid Dynamics



Biomedical Imaging



Numerical Tokamak

LAPACK

For High-Performance serial and shared-memory parallel computers

BLAS / ***BLACS***

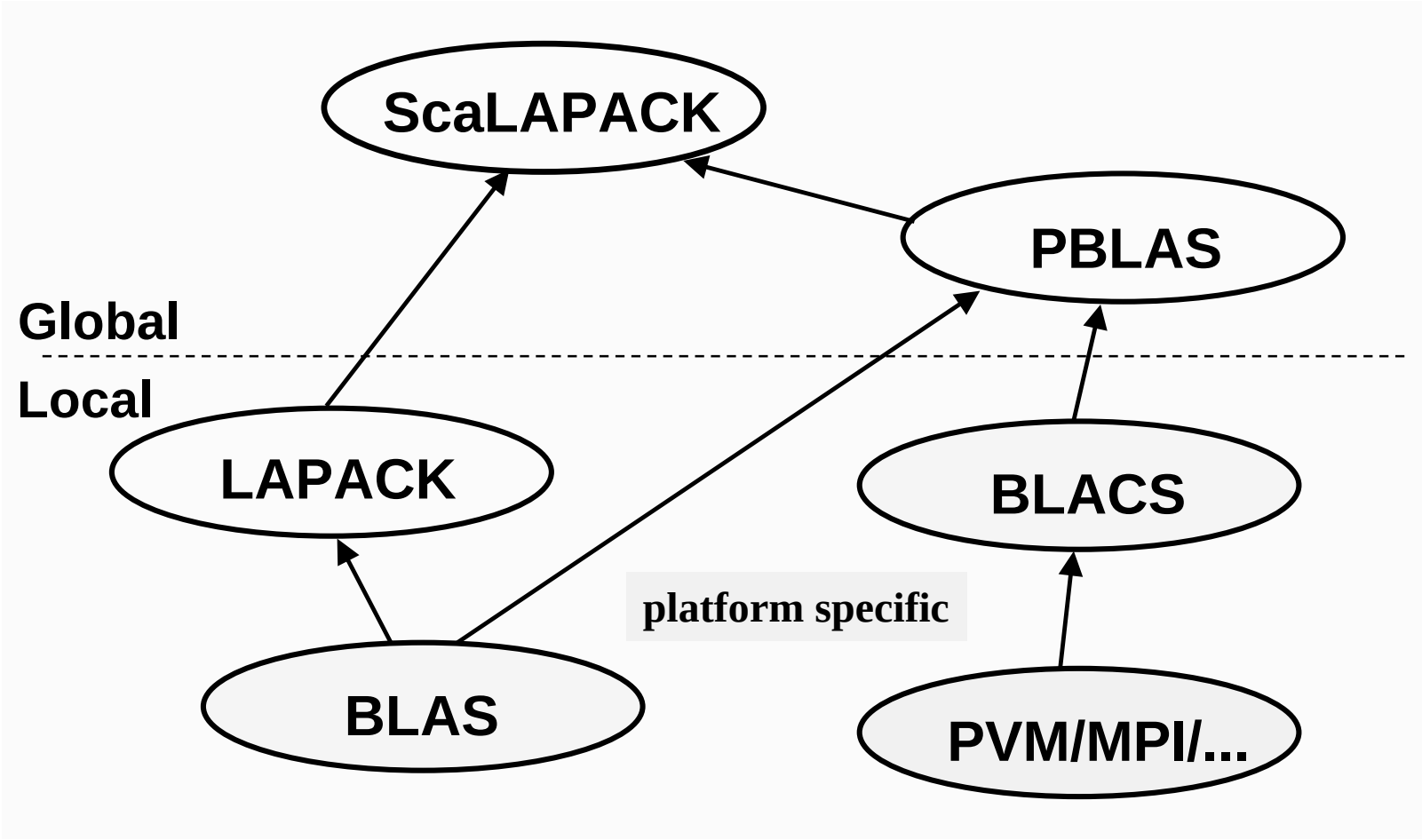
SCALAPACK

For distributed-memory parallel computers

Images from http://www.hpcc.gov/blue96

ScaLAPACK: *structure of the software*

http://acts.nersc.gov/scalapack



BLAS

http://acts.nersc.gov/scalapack

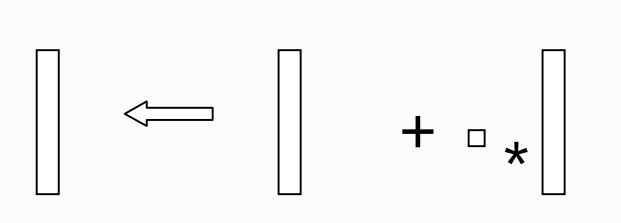
(*Basic Linear Algebra Subroutines*)

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

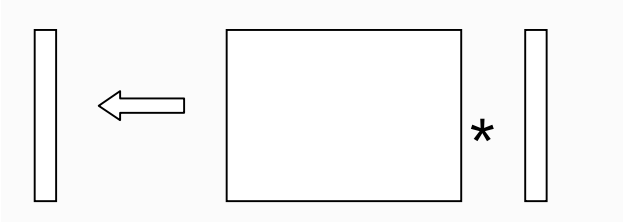
BLAS: *3 levels*

http://acts.nersc.gov/scalapack

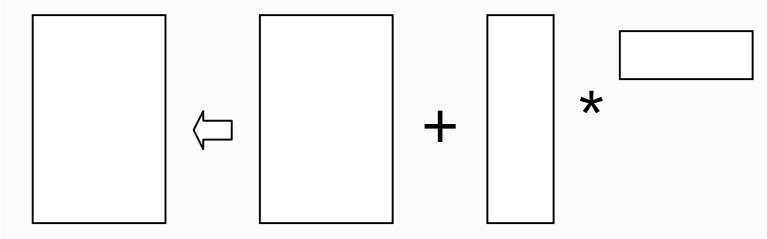
- Level 1 BLAS:
  vector-vector operations.



- Level 2 BLAS:
  matrix-vector operations.

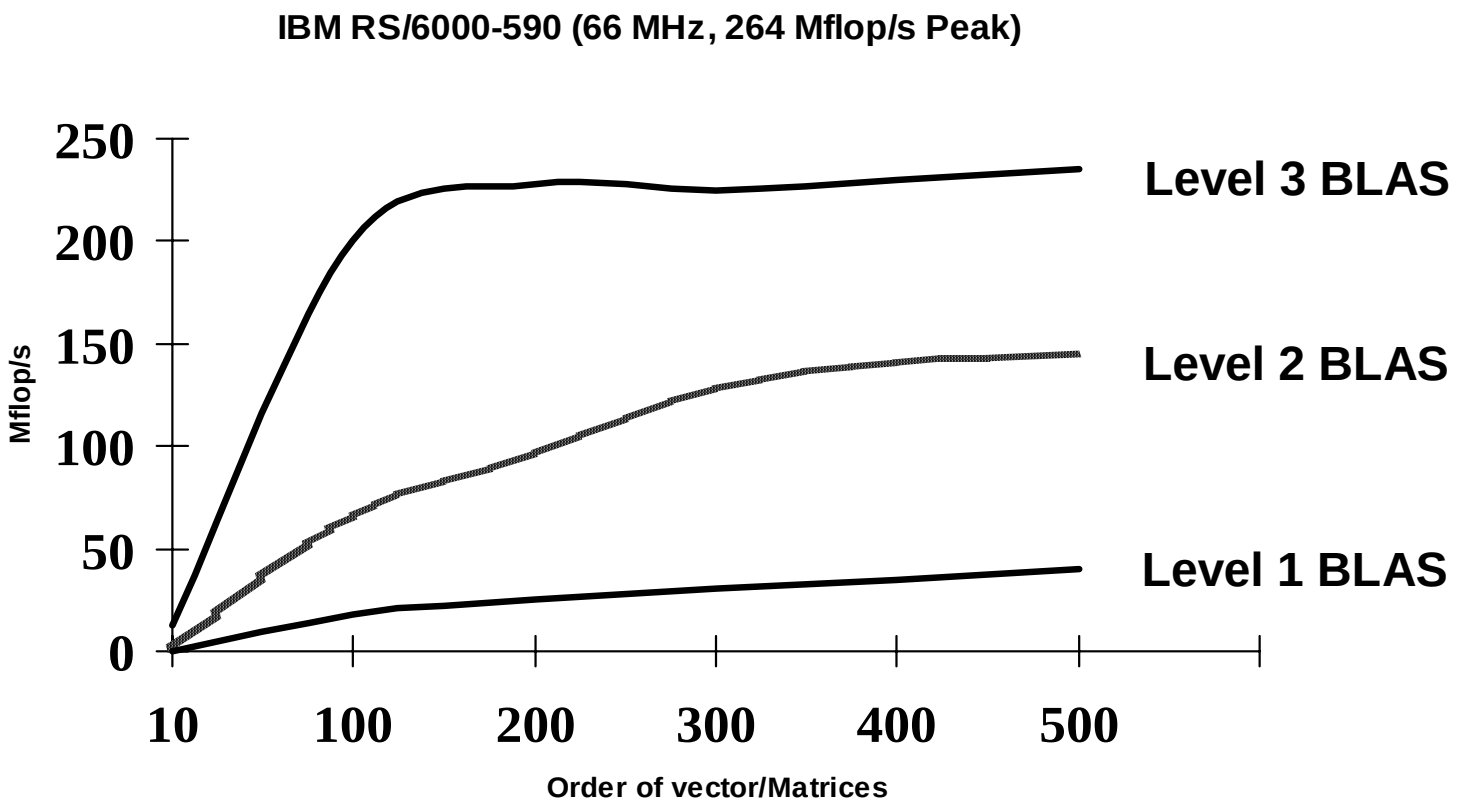


- Level 3 BLAS:
  matrix-matrix operations.



BLAS: *performance*

http://acts.nersc.gov/scalapack



Development of blocked algorithms is important for performance!

LAPACK

http://acts.nersc.gov/scalapack

- Linear Algebra library written in Fortran 77 (C and C++ versions also available).
- Combine algorithms from LINPACK and EISPACK into a single package.
- Efficient on a wide range of computers (RISC, Vector, SMPs).
- User interface similar to LINPACK (Single, Double, Complex, Double Complex).
- Built atop level 1, 2, and 3 BLAS for high performance, clarity, modularity and portability.

LAPACK: *contents*

http://acts.nersc.gov/scalapack

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

BLACS

http://acts.nersc.gov/scalapack

(*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*

http://acts.nersc.gov/scalapack

Processes are embedded in a two-dimensional grid, example: 3x4 grid

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

BLACS: *scopes*

http://acts.nersc.gov/scalapack

An operation which involves more than one sender and one receiver is called a *scoped operation*. Using a 2D-grid, there are 3 natural scopes:

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*

http://acts.nersc.gov/scalapack

Send/Receive

Send (sub)matrix from one process to another:

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

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

BLACS: *communication routines*

http://acts.nersc.gov/scalapack

Broadcast

Send (sub)matrix to all processes or subsection of processes in SCOPE, using various distribution patterns (TOP):

`_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’ ‘Column’ ‘All’	‘ ‘ (default) ‘Increasing Ring’ ‘1-tree’ …

BLACS: *combine operations*

http://acts.nersc.gov/scalapack

Global combine operations

- Perform element-wise SUM, |MAX|, |MIN|, operations on triangular matrices:

```
_GSUM2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
_GAMX2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
_GAMN2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
```

- RDEST = –1 indicates that the result of the operation should be left on all processes selected by SCOPE.
- For |MAX|, |MIN|, when RCFLAG = –1, RA and CA are not referenced; otherwise RA and CA are set on output with the coordinates of the process owning the corresponding maximum (or minimum) element in absolute value of A.

BLACS: *example*

http://acts.nersc.gov/scalapack

BLACS_SETUP

```
*  Get system information
   CALL BLACS_PINFO( IAM, NPROCS )
*  If underlying system needs additional setup, do it now
   IF( NPROCS.LT.1 ) THEN
     IF( IAM.EQ.0 ) NPROCS = 4
       CALL BLACS_SETUP( IAM, NPROCS )
     END IF
*  Get default system context
   CALL BLACS_GET( 0, 0, ICTXT )
   ⋮
```

(out) uniquely identifies each process
(out) number of processes available
(in) uniquely identifies each process
(in) number of processes available
(out) BLACS context (see slide 21)
(in) use (default) system context
(in) integer handle indicating the context

BLACS: *example*

http://acts.nersc.gov/scalapack

BLACS_GRIDINFO

```
*  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.EQ.-1 ) GO TO 10
      ⋮
   CALL BLACS_GRIDEXIT( ICTXT )
10 CONTINUE
   CALL BLACS_EXIT( 0 )
   END
```

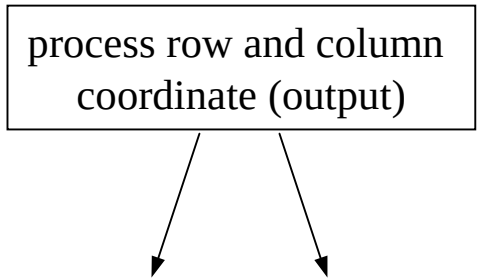
BLACS context (see slide 21)

process row and column coordinate (output)

BLACS: e*xample*

http://acts.nersc.gov/scalapack

SEND/RECEIVE




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

send X to process (1,0)

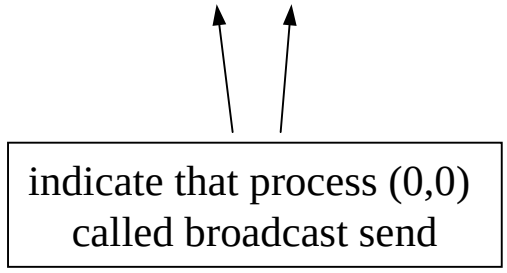
receive X from process (0,0)

BLACS: *example*

http://acts.nersc.gov/scalapack

BROADCAST

```
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*     Broadcast A to processes in row 0
      IF( MYROW.EQ.0 ) THEN
        IF( MYCOL.EQ.0 ) THEN
          CALL DGEBS2D( ICTXT, 'Row', ' ', 2, 2, A, 3 )
        ELSE
          CALL DGEBR2D( ICTXT, 'Row', ' ', 2, 2, A, 3, 0, 0 )
        END IF
      END IF
      ⋮
```



BLACS: *example*

http://acts.nersc.gov/scalapack

COMBINE OPERATIONS

```
   CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
   IF( MYROW.EQ.0 ) THEN
     K00 = MYCOL
     CALL IGSUM2D( ICTXT, 'Row', ' ', 1, 1, K00, 1, 0, 0 )
     KALL = MYCOL
     CALL IGSUM2D( ICTXT, 'Row', ' ', 1, 1, KALL, 1, -1, 0 )
   END IF
   :
   EPS = 1.0D+0
10 CONTINUE
   EPS = EPS / 2.0D+0
   IF( ( 1.0D+0 + EPS ) .GT. 1.0D+0 ) GO TO 10
   CALL DGAMX2D( ICTXT, 'All', ' ', 1, 1, EPS, 1, IRSRC, ICSRC, -1, -1, 0 )
   :
```

BLACS: *context*

http://acts.nersc.gov/scalapack

- 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

BLACS: *repeatability*

http://acts.nersc.gov/scalapack

- A routine is *repeatable* if it is guaranteed to give the same answer if called multiple times with the same parallel configuration and input.
- A routine is *coherent* if all processes selected to receive an answer get identical results.
  - Homogeneous coherency: all processes selected to possess the result receive the exact same answer if communication does not change the value of the data or all processes perform floating point arithmetic exactly the same.
  - Heterogeneous coherency: all processes will receive the exact same answer if communication does not change the value of the communicated data.
- Repeatability and coherence do not affect correctness. A routine may be both incoherent and non-repeatable, and still give the correct output.

PBLAS

http://acts.nersc.gov/scalapack

(*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

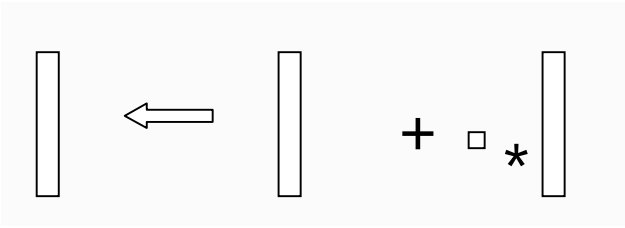
```
CALL PDGEXXX( M, N, A, IA, JA, DESCA, ... )
```

PBLAS

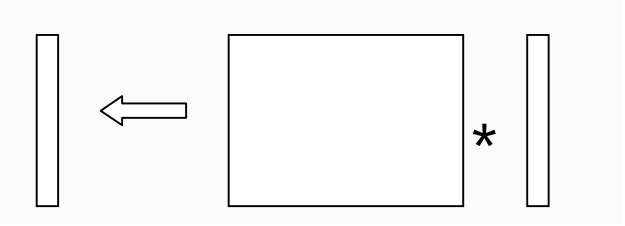
PBLAS: *3 levels*

http://acts.nersc.gov/scalapack

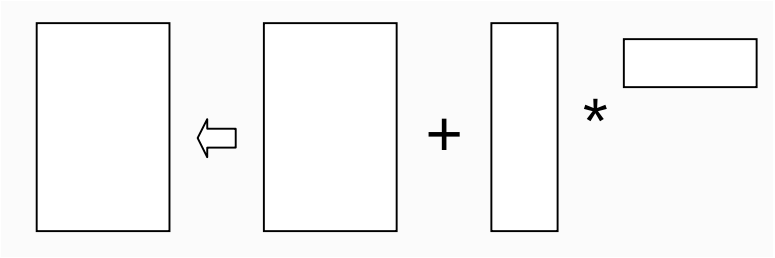
- Level 1 PBLAS:
  vector-vector operations.



- Level 2 PBLAS:
  matrix-vector operations.



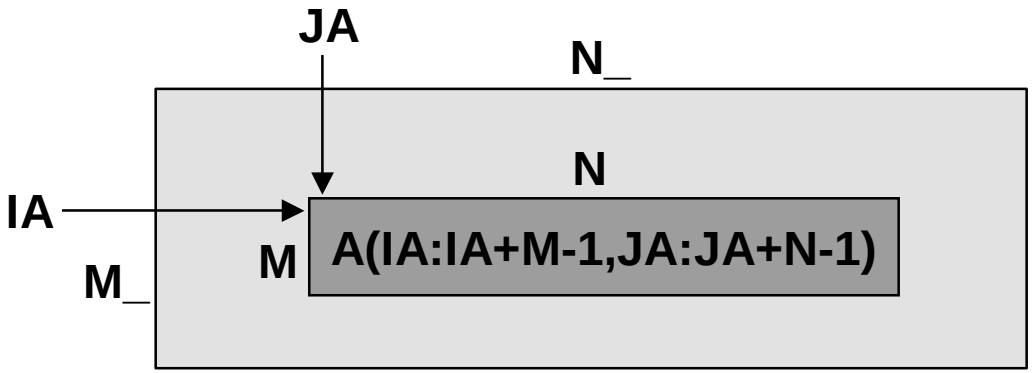
- Level 3 PBLAS:
  matrix-matrix operations.



PBLAS: *syntax*

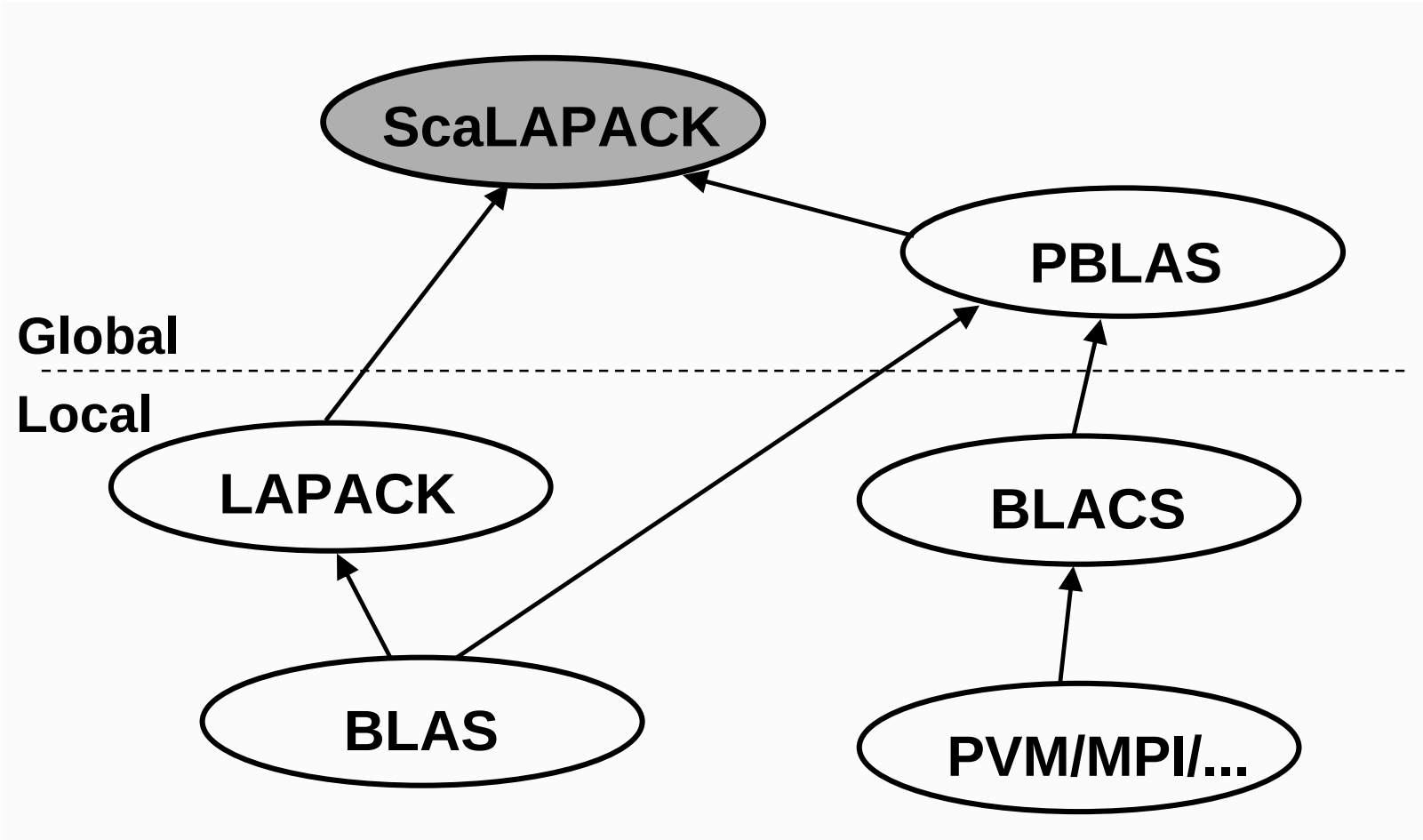
http://acts.nersc.gov/scalapack

Global view of the matrix operands, allowing global addressing of distributed matrices (hiding complex local indexing)



ScaLAPACK: *structure of the software*

http://acts.nersc.gov/scalapack



ScaLAPACK: *goals*

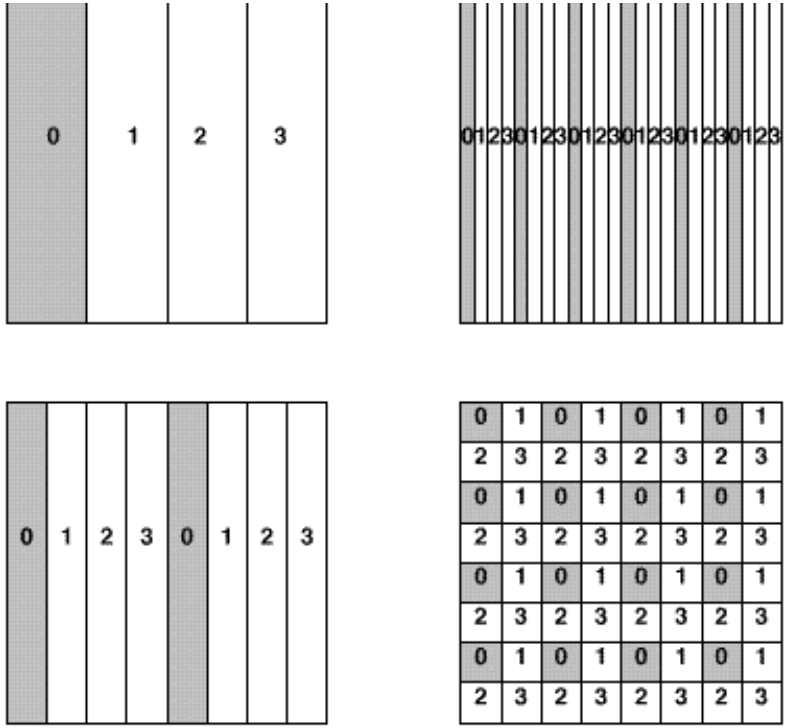
http://acts.nersc.gov/scalapack

- 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: *possible data layouts*

http://acts.nersc.gov/scalapack

- 1D block and cyclic column distributions

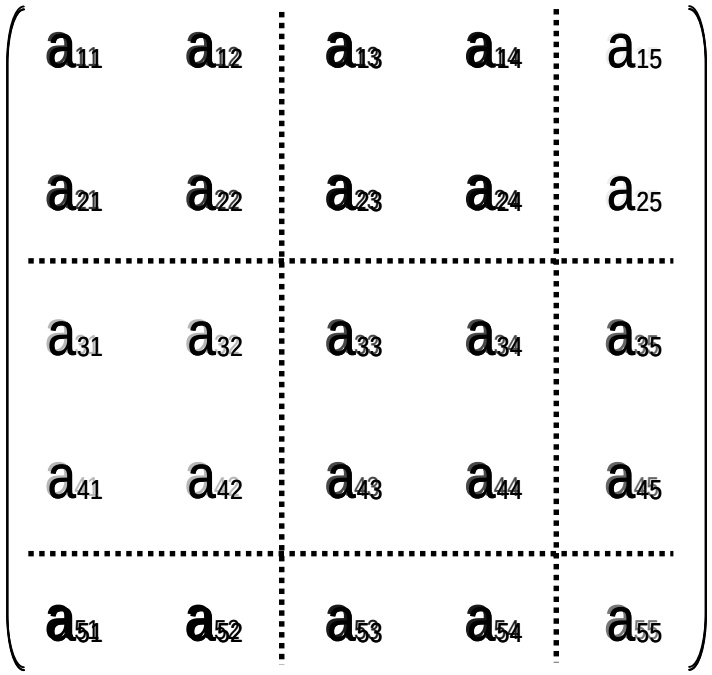


- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

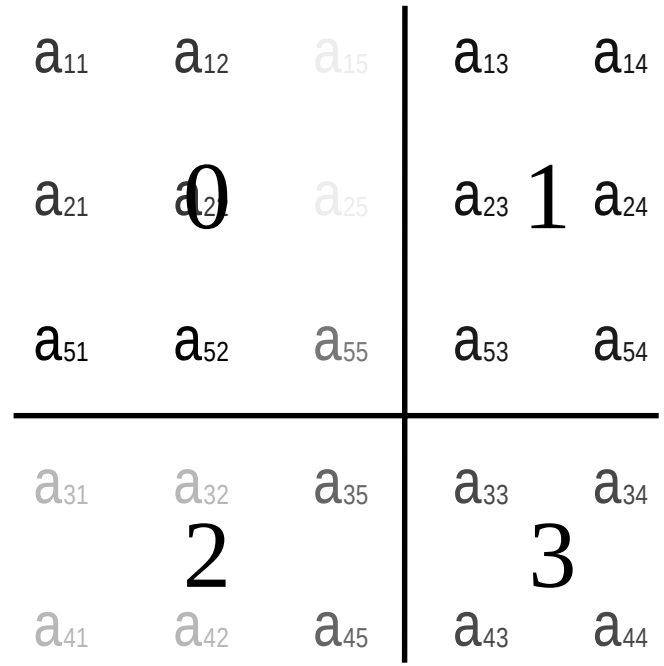
ScaLAPACK: *2D Block-Cyclic Distribution*

http://acts.nersc.gov/scalapack

5x5 matrix partitioned in 2x2 blocks



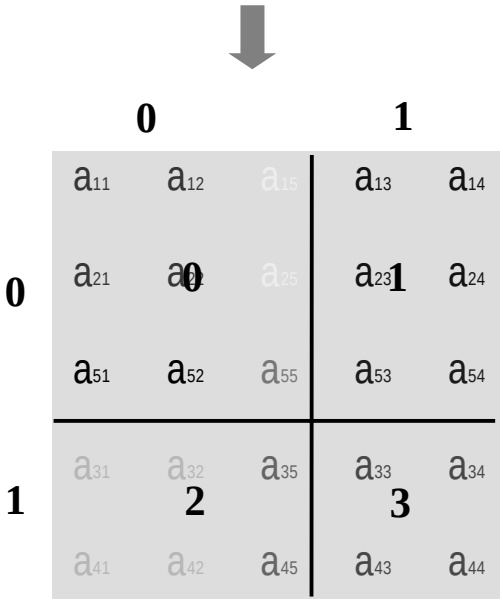
2x2 process grid point of view



Two-Dimensional Block-Cyclic Distribution

http://acts.nersc.gov/scalapack

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




```
⋮

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

IF      ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
        A(1) = 1.1; A(2) = -2.1; A(3) = -5.1;
        A(1+LDA) = 1.2; A(2+LDA) = 2.2; A(3+LDA) = -5.2;
        A(1+2*LDA) = 1.5; A(2+3*LDA) = 2.5; A(3+4*LDA) = -5.5;
ELSE IF ( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
        A(1) = 1.3; A(2) = 2.3; A(3) = -5.3;
        A(1+LDA) = 1.4; A(2+LDA) = 2.4; A(3+LDA) = -5.4;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
        A(1) = -3.1; A(2) = -4.1;
        A(1+LDA) = -3.2; A(2+LDA) = -4.2;
        A(1+2*LDA) = 3.5; A(2+3*LDA) = 4.5;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
        A(1) = 3.3; A(2) = -4.3;
        A(1+LDA) = 3.4; A(2+LDA) = 4.4;
END IF

  ⋮
```

LDA is the leading dimension of the local array, used in the descriptors (see slides 32-35)

Two-Dimensional Block-Cyclic Distribution

http://acts.nersc.gov/scalapack

- Ensure 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).
- Need redistribution routines to go from one distribution to the other.

ScaLAPACK: *array descriptors*

http://acts.nersc.gov/scalapack

- 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

Array descriptor for Narrow Band Matrices

DESC_()	Symbolic Name	Scope	Definition

Array descriptor for Right Hand Sides for Narrow Band Linear Solvers

DESC_()	Symbolic Name	Scope	Definition

ScaLAPACK: *Functionality*

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

ScaLAPACK: *error handling*

http://acts.nersc.gov/scalapack

- 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 is stopped.

ScaLAPACK: *debugging hints*

http://acts.nersc.gov/scalapack

- 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*

http://acts.nersc.gov/scalapack

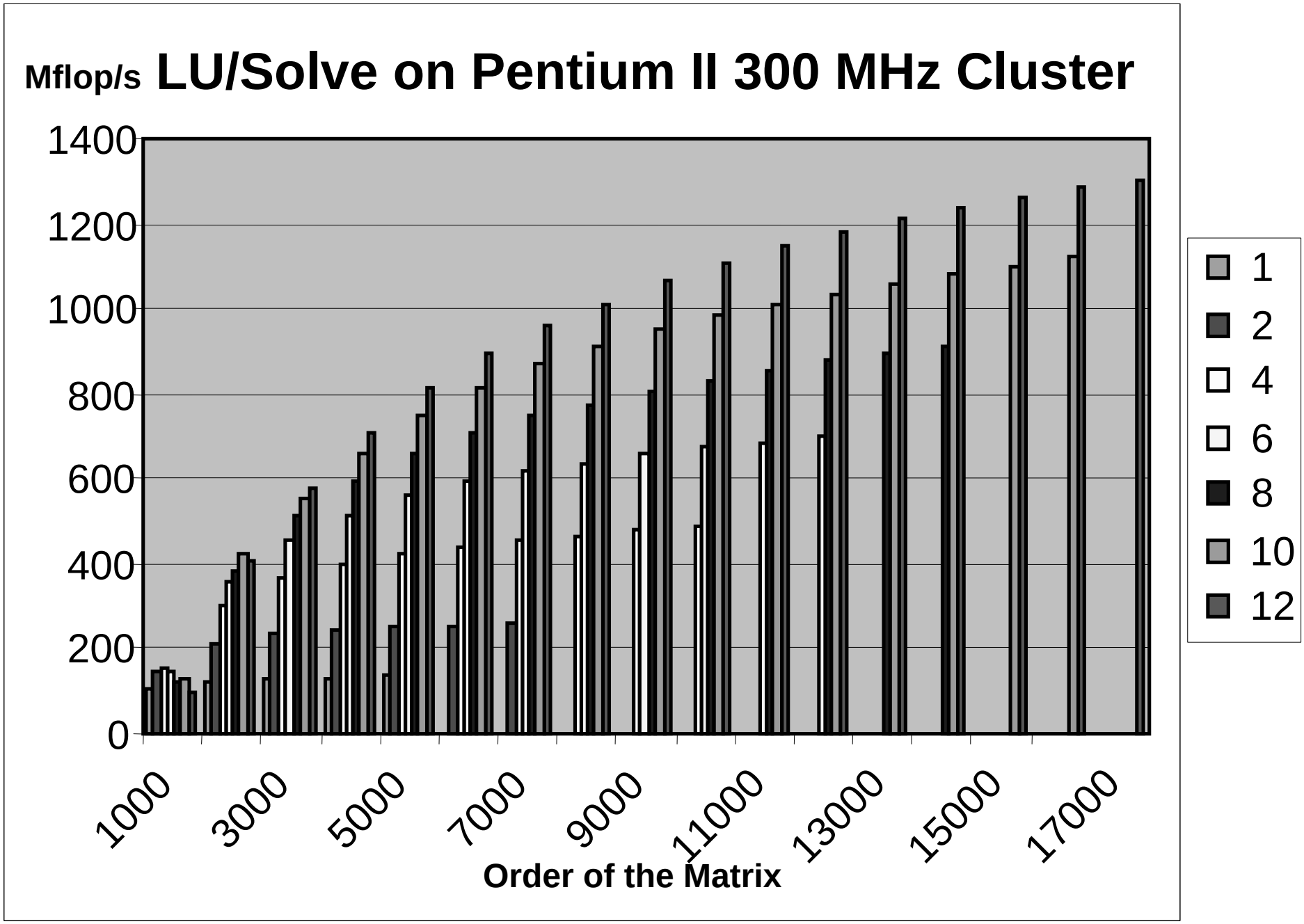
- For dense matrix computations, an implementation is said to be *scalable* if the parallel efficiency is an increasing function of N^2/P, the problem size per node. The algorithms implemented in ScaLAPACK are scalable in this sense.
- Maintaining memory use per node constant allows efficiency to be maintained (in practice, a slight degradation is acceptable).

Distributed-Memory Computer

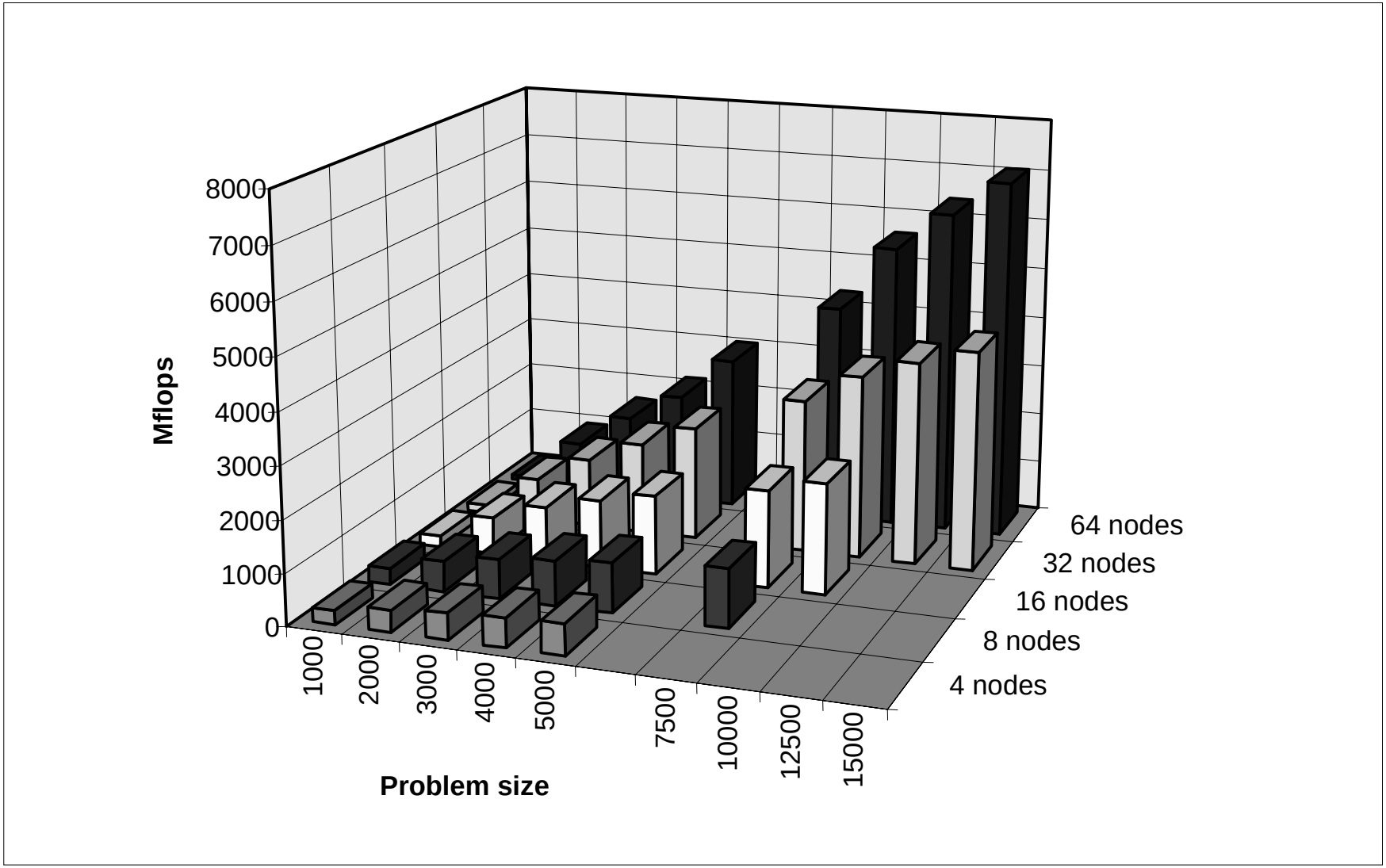
- Use the right number of processors
  - Rule of thumb: P=MxN/1,000,000 for an MxN matrix. This provides a local matrix of size approximately 1000-by-1000.
  - Do not try to solve a small problem on too many processors.
  - Do not exceed physical memory.
- Use an efficient data distribution.
  - Block size (i.e., MB,NB) = 64.
  - Square processor grid: Prow = Pcolumn.
- Use efficient machine-specific BLAS (not the Fortran77 reference implementation from netlib) and BLACS (nondebug, BLACSDBGLVL=0 in Bmake.inc)

Network of Workstations

- The bandwidth per node, if measured in Megabytes per second per node, should be no less than one tenth the peak floating-point rate as measured in megaflops/second/node.
- The underlying network must allow simultaneous messages, that is, not standard ethernet and not FDDI (Fiber Distributed Data Interface).
- Message latency should be no more than 500 microseconds.
- All processors should be similar in architecture and performance. ScaLAPACK will be limited by the slowest processor. Data format conversion significantly reduces communication performance.
- Dedicated use of processors
- No more than one process should be executed per processor.


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

LU factorization+solve on IBM SP2 thin nodes



ScaLAPACK: *Heterogeneous Computing*

http://acts.nersc.gov/scalapack

- Software intended to be used in cluster computing context
- Difficulties arise with the following issues:
  - Communication of floating point numbers between processors
  - Repeatability and coherency
  - Machine precision and other machine specific parameters
  - Different versions of compilers
  - Checking global floating-point arguments
  - Iterative convergence across clusters of processors

ScaLAPACK: *Commercial Use*

http://acts.nersc.gov/scalapack

ScaLAPACK has been incorporated in the following commercial packages:

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

ScaLAPACK: *Hands-on*

http://acts.nersc.gov/scalapack

- Exercises: *http://acts.nersc.gov/scalapack/hands-on/main.html*
- Information on the Cray T3E:
  - *man intro_scalapack*
  - */usr/local/pkg/acts/SCALAPACK-1.5/examples*
- Further information on netlib (installation, working notes):
  - BLACS: *www.netlib.org/blacs*
  - LAPACK: *www.netlib.org/lapack*
  - ScaLAPACK: *www.netlib.org/scalapack*

ScaLAPACK: *Development Team*

http://acts.nersc.gov/scalapack

- 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
- Osni Marques, LBNL/NERSC
- Caroline Papadopoulos, UCSD
- Antoine Petitet, UTK
- Ken Stanley, UCB
- Francoise Tisseur, Manchester
- David Walker, Cardiff
- Clint Whaley, UTK

URL: ***http://acts.nersc.gov/scalapack/hands-on/main.html***

Copy exercises from: ***/usr/local/pkg/acts/SCALAPACK-1.5/examples/hands-on.tar.gz***

Hands-On Exercises for ScaLAPACK

ScaLAPACK Team
March 2001

Introduction

These exercises provide basic and more advanced programming instruction for writing parallel programs calling the BLACS, PBLAS, and ScaLAPACK. A basic knowledge of Fortran, parallel programming with message-passing, and MPI are assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

Detailed information on the BLACS, PBLAS, and ScaLAPACK may be found at the respective URLs:

- http://www.netlib.org/blacs
- http://www.netlib.org/scalapack
- http://www.netlib.org/scalapack/pblas_qref.html

Exercises 1 and 2 give an introduction to parallel programming with the Basic Linear Algebra Communication Subprograms (BLACS). Exercises 3, 4, and 5 provide a range of simplistic to more complex parallel programs calling ScaLAPACK and PBLAS. More example programs for ScaLAPACK can be found at http://www.netlib.org/scalapack/examples.

The instructions for the exercises assume that the underlying system is an IBM SP or a Cray T3E; using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. The version of MPI used in these examples is the version 3.0, and we assume the user has this version installed.

These hands-on exercises were prepared in collaboration with the Joint Institute for Computational Science at the University of Tennessee, based on contributions from A. YarKhan, C. Hastings, S. Blackford, C. Whaley, A. Petitet and O. Marques.

Exercise 1: **BLACS - Hello World Example**
Exercise 2: **BLACS - Pi Example**
Exercise 3: **ScaLAPACK - Example Program 1**
Exercise 4: **ScaLAPACK - Example Program 2**
Exercise 5: **PBLAS Example**

Help: useful calling sequences

Download all exercises