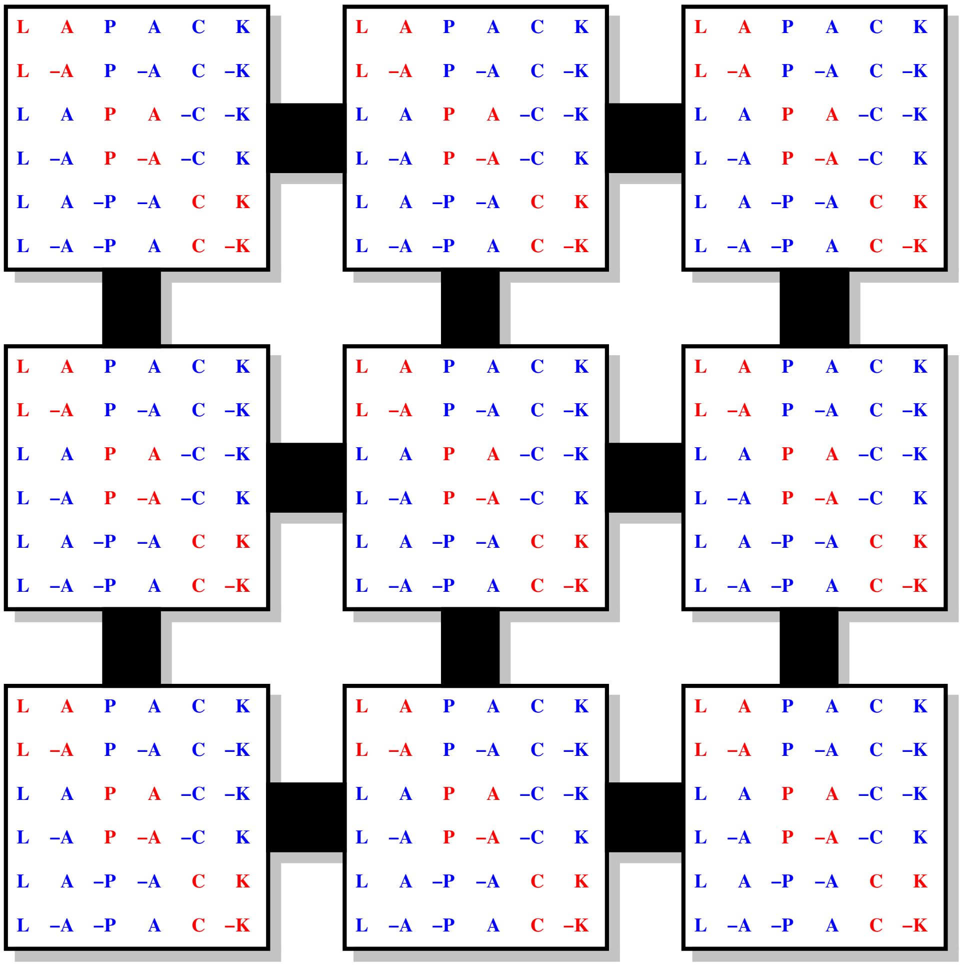
PARA96

WORKSHOP ON APPLIED PARALLEL COMPUTING IN INDUSTRIAL PROBLEMS AND OPTIMIZATION

August 18, 1996

SCALAPACK TUTORIAL



Jack Dongarra, University of Tennessee, Knoxville and Oak Ridge National Laboratory, Oak Ridge, USA
L. Susan Blackford, University of Tennessee, Knoxville, USA

ScaLAPACK Project

The ScaLAPACK project is a collaborative effort between the following institutions and principal investigators:

- Oak Ridge National Laboratory
  David Walker, Co-Principal Investigator
- Rice University
  Dan Sorensen, Co-Principal Investigator
- University of California, Berkeley
  Jim Demmel, Co-Principal Investigator
- University of California, Los Angeles
  Tony Chan, Co-Principal Investigator
- University of Illinois
  Michael Heath, Co-Principal Investigator
  Padma Raghavan, Co-Principal Investigator
- University of Tennessee, Knoxville
  Jack Dongarra, Principal Investigator

The ScaLAPACK project is comprised of 4 components:

- dense matrix software (**ScaLAPACK**)
- large sparse eigenvalue software (**PARPACK**)
- sparse direct systems software (**CAPSS**)
- preconditioners for large sparse iterative solvers (**PARPRE**)

Funding for this effort comes in part from **DARPA**, **DOE**, **NSF**, and **CRPC**.

http://www.netlib.org/scalapack/index.html

SCALAPACK – CONTRIBUTORS

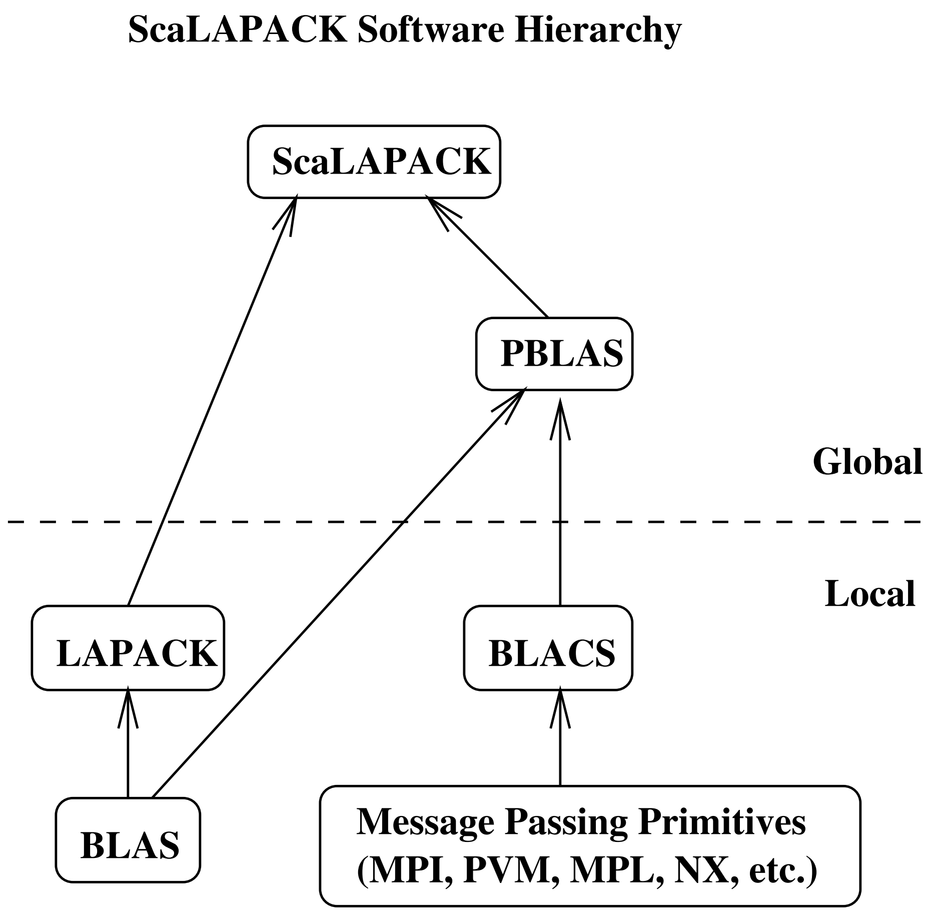
University of Tennessee, Knoxville
Jack Dongarra, Principal Investigator

- Susan Blackford
- Andrew Cleary
- Brett Ellis
- Sven Hammarling
- Greg Henry
- Antoine Petitet
- R. Clint Whaley

University of California, Berkeley
Jim Demmel, Co-Principal Investigator

- Soumen Chakrabarti
- Inderjit Dhillon
- Ray Fellers
- Melody Ivory
- Ren-Cang Li
- Xiaoye Li
- Huan Ren
- Howard Robinson
- Ken Stanley
- Andrei Zege

ScaLAPACK Software Hierarchy



OUTLINE

- **The Basic Linear Algebra Subprograms (BLAS)**
- **The Linear Algebra PACKage (LAPACK)**
- **The Basic Linear Algebra Communication Subprograms (BLACS)**
- **The Parallel Basic Linear Algebra Subprograms (PBLAS)**
- **The Scalable Linear Algebra PACKage (ScaLAPACK)**
- **HPF interface to ScaLAPACK**

The Basic Linear Algebra Subprograms (BLAS)

BLAS – INTRODUCTION

- **Clarity:** code is shorter and easier to read,
- **Modularity:** gives programmer larger building blocks,
- **Performance:** manufacturers will provide tuned machine-specific BLAS,
- **Program portability:** machine dependencies are confined to the BLAS.

BLAS – BASICS

Level 1 BLAS – Vector Operations (Late '70s)

$$
\begin{array}{lll}
y \leftarrow y + \alpha x, & y \leftarrow x, & y \leftarrow \alpha x, \\
\alpha \leftarrow x^T y, & \alpha \leftarrow \|x\|_2, & y \leftrightarrow x.
\end{array}
$$

Example: `SAXPY` operate with vectors:

$$y \leftarrow y + \alpha x$$

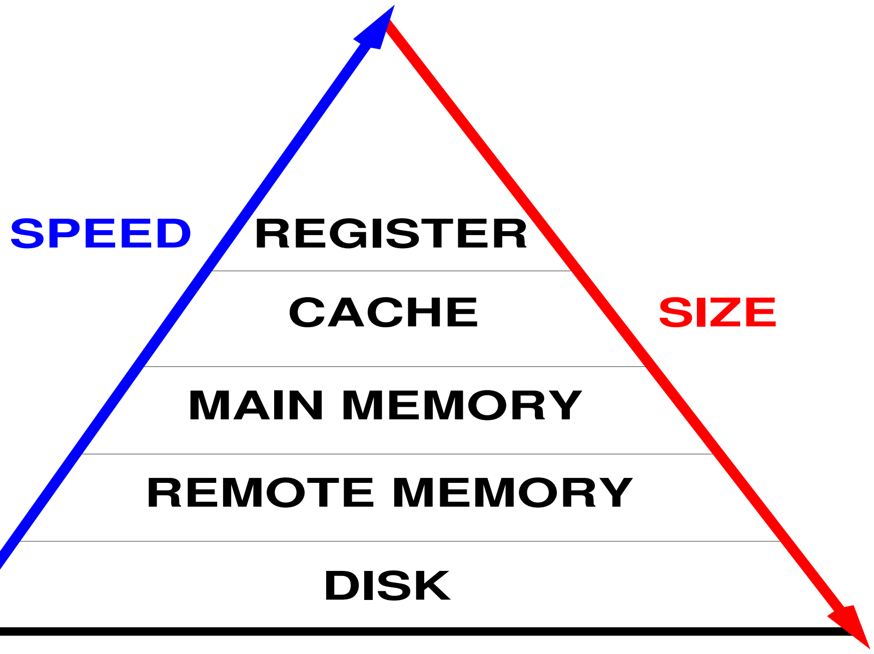
- 2 vector loads
- 2 vector operations
- 1 vector store

Too much memory traffic: little chance to use the memory hierarchy,

- $O(n)$ memory references,
- $O(n)$ floating point operations.

BASIC LINEAR ALGEBRA SUBPROGRAMS (BLAS)

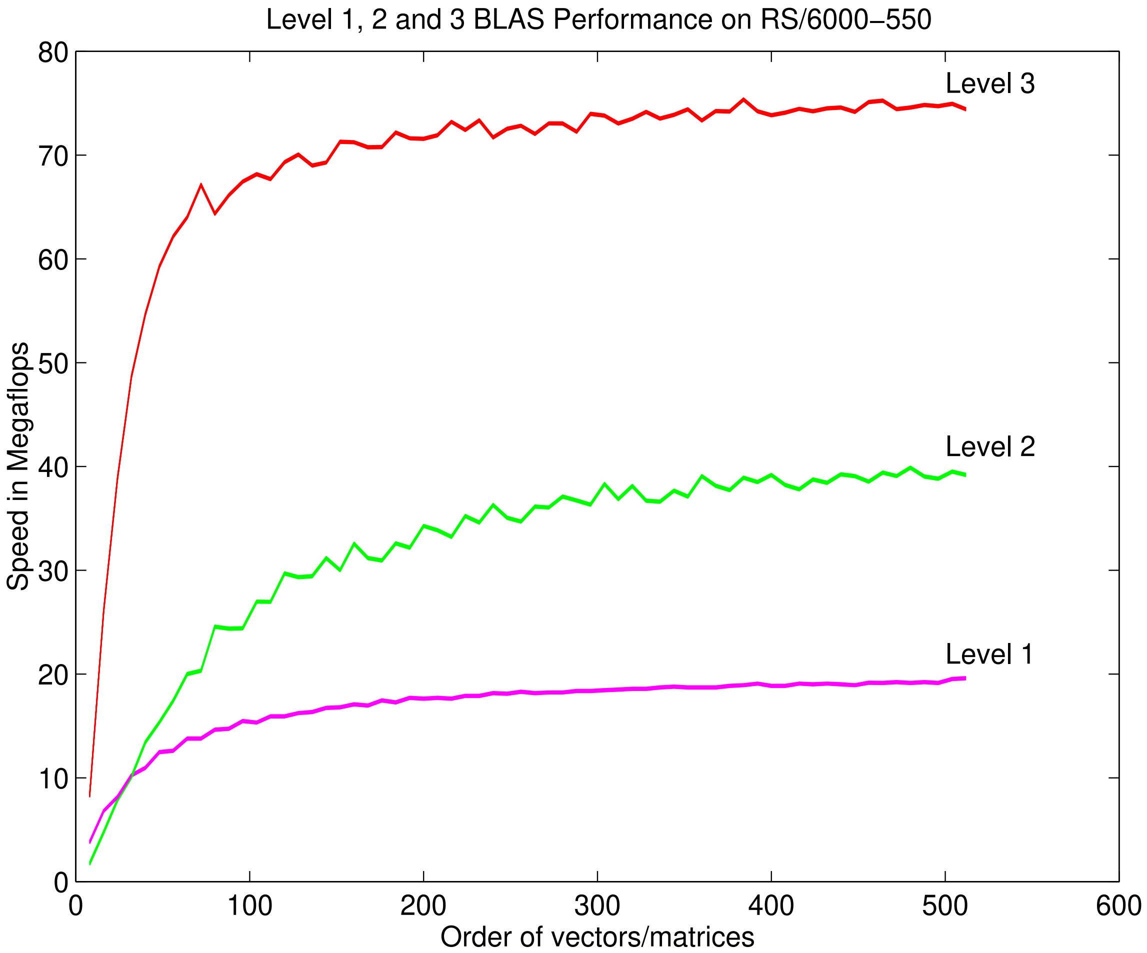
- Arithmetic is only performed at the top of the memory hierarchy, $\Rightarrow$ keep data as close as possible to the top.



- Higher level BLAS $\Rightarrow$ increase granularity $\Rightarrow$ lower synchronization cost,

	flops	mem ref	flops/mem ref
Level 1 BLAS $y \leftarrow y + \alpha x$	$2n$	$3n$	$2/3$
Level 2 BLAS $y \leftarrow y + Ax$	$2n^2$	n^2	2
Level 3 BLAS $C \leftarrow C + AB$	$2n^3$	$4n^2$	$n/2$

BLAS – BASICS



- Development of block-partitioned algorithms using Level 3 BLAS,
- ⇒ LAPACK, high performance, portability.

BLAS – REFERENCES

- BLAS software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/blas`,
  - (anonymous) ftp `ftp.netlib.org`:

    `cd blas; get index`

  - email `netlib@www.netlib.org` with the message:

    `send index from blas`

- Comments and questions can be addressed to

  `lapack@cs.utk.edu`

- C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Transactions on Mathematical Software, 5:308–325, 1979.
- J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, *An Extended Set of Fortran Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14(1):1–32, (1988).
- J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16(1):1–17, 1990.

The Linear Algebra PACKage

(LAPACK)

L	A	P	A	C	K
L	–A	P	–A	C	–K
L	A	P	A	–C	–K
L	–A	P	–A	–C	K
L	A	–P	–A	C	K
L	–A	–P	A	C	–K

LAPACK – MOTIVATIONS

- Example: Cray Y-MP C-90 with 1 and 16 processors (speeds in Gflops)

	1 PE	16 PEs
Maximum Speed	0.95	15.24
LINPACK (Cholesky, n=500)	0.39	0.48

- LINPACK flops lost due to data movement in memory hierarchy:

	1 PE	16 PEs
Matrix-vector multiply	0.895	5.90
Matrix-matrix multiply	0.898	13.00

- **LAPACK uses high level BLAS:**

	1 PE	16 PEs
LAPACK (Cholesky, n=500)	0.79	5.70
LAPACK (Cholesky, n=1000)	0.85	9.80

- **Basic Problems:**

Linear systems:	$Ax = b$
Least squares:	$min_x \|\|Ax - b\|\|_2, A = U\Sigma V^T$
Eigenvalues and vectors:	$Ax = \lambda x, Ax = \lambda Bx$

- Uses (manufacturer optimized) BLAS $\Rightarrow$ high performance
- (Trans)portable FORTRAN 77 (+BLAS), 766K lines (354K library, 412K testing and timing).

LAPACK – BLOCK PARTITIONED ALGORITHMS

Block Partitioned Algorithms have been developed for:

- LU factorization,
- Cholesky factorization,
- Factorization of symmetric indefinite matrices,
- Matrix inversion,

- Banded LU and Cholesky factorizations,

- QR, QL, RQ, LQ factorizations,
- Form Q or Q^TC,

- Orthogonal reduction to:
  - (upper) Hessenberg form,
  - symmetric tridiagonal form,
  - bidiagonal form,

- Block QR iteration for nonsymmetric eigenvalue problems.

LAPACK – DERIVATION OF BLOCK PARTITIONED ALGORITHMS

$$\begin{pmatrix} A_{11} & a_j & A_{13} \\ . & a_{jj} & \alpha_j^T \\ . & . & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ u_j^T & u_{jj} & 0 \\ U_{13}^T & \mu_j & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & u_j & U_{13} \\ 0 & u_{jj} & \mu_j^T \\ 0 & 0 & U_{33} \end{pmatrix}$$

and equating coefficient of the j^{th} column, we obtain

$$\begin{cases} a_j = U_{11}^T u_j \\ \\ a_{jj} = u_j^T + u_{jj}^2 \end{cases}$$

Hence, if U_{11} has already been computed, we can compute u_j and u_{jj} from the equations:

$$\begin{cases} U_{11}^T u_j = a_j \\ \\ u_{jj}^2 = a_{jj} - u_j^T u_j \end{cases}$$

LAPACK – DERIVATION OF BLOCK PARTITIONED ALGORITHMS

Here is the body of the LINPACK routine SPOFA, which implements the above method:

```
      DO 30 J = 1, N
         INFO = J
         S = 0.0E0
         JM1 = J - 1
         IF( JM1.LT.1 ) GO TO 20
         DO 10 K = 1, JM1
            T = A( K, J ) -
     $          SDOT( K-1, A( 1, K ), 1,
     $                     A( 1, J ), 1 )
            T = T / A( K, K )
            A( K, J ) = T
            S = S + T*T
   10    CONTINUE
   20    CONTINUE
         S = A( J, J ) - S
C     ...EXIT
         IF( S.LE.0.0E0 ) GO TO 40
         A( J, J ) = SQRT( S )
   30 CONTINUE
```

LAPACK – DERIVATION OF BLOCK PARTITIONED ALGORITHMS

```
   DO 10 J = 1, N
      CALL STRSV( 'Upper', 'Transpose', 'Non-Unit',
  $               J-1, A, LDA, A( 1, J ), 1 )
      S = A( J, J ) - SDOT( J-1, A( 1, J ), 1,
  $                               A( 1, J ), 1 )
      IF( S.LE.ZERO ) GO TO 20
      A( J, J ) = SQRT( S )
10 CONTINUE
```

- This change by itself is sufficient to significantly improve the performance on a number of machines:

  From 72 to 251 megaflops for a matrix of order 500 on one processor of a CRAY Y-MP (81 % of the peak speed of matrix-matrix multiplication on this processor).

- Only 378 megaflops on 8 PEs of a CRAY Y-MP.

⇒ Re-cast as a **block partitioned algorithm**.

LAPACK – DERIVATION OF BLOCK PARTITIONED ALGORITHMS

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ . & A_{22} & A_{23} \\ . & . & A_{33} \end{pmatrix} = \begin{pmatrix} U_{11}^T & 0 & 0 \\ U_{12}^T & U_{22}^T & 0 \\ U_{13}^T & U_{23}^T & U_{33}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}$$

By equating submatrices in the second block of columns, we obtain:

$$\begin{cases} A_{12} = U_{11}^T U_{12} \\ \\ A_{22} = U_{12}^T U_{12} + U_{22}^T U_{22} \end{cases}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution of the following equation by a call to the Level 3 BLAS routine STRSM:

$$U_{11}^T U_{12} = A_{12} \qquad (\square\text{TRSM})$$

and then compute U_{22} from:

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}. \qquad (\square\text{SYRK})$$

LAPACK – DERIVATION OF BLOCK PARTITIONED ALGORITHMS

```
   DO 10 J = 1, N, NB
      JB = MIN( NB, N-J+1 )
      CALL STRSM( 'Left', 'Upper', 'Transpose',
  $               'Non-Unit', J-1, JB, ONE, A, LDA,
  $               A( 1, J ), LDA )
      CALL SSYRK( 'Upper', 'Transpose', JB, J-1,
  $               -ONE, A( 1, J ), LDA, ONE,
  $               A( J, J ), LDA )
      CALL SPOTF2( 'Upper', JB, A( J, J ), LDA,
  $                INFO )
      IF( INFO.NE.0 ) GO TO 20
10 CONTINUE
```

On a CRAY Y-MP, the use of Level 3 BLAS squeezes a little more performance out of one processor, but makes a large improvement when using all 8 processors:

CRAY-YMP	1 PE	8 PEs
j-variant: LINPACK	72	72
j-variant: using Level 2 BLAS	251	378
j-variant: using Level 3 BLAS	287	1225
i-variant: using Level 3 BLAS	290	1414

Speed in megaflops of Cholesky factorization $A = U^T U$ for $n = 500$

LAPACK – CONTENTS

- Combines algorithms from LINPACK and EISPACK into a single package. User interface similar to LINPACK

- Built on the Level 1, 2 and 3 BLAS, uses BLAS for high performance (manufacturers optimize BLAS)

- Linear Algebra library in FORTRAN 77
  - Solution of systems of equations $Ax = b$ for general dense, symmetric, banded, tridiagonal matrix A,
  - Solution of linear least squares problems $min_x \|Ax - b\|_2$ for general dense matrix A,
  - Singular value decomposition $A = U\Sigma V^T$ for general dense, banded matrix A,
  - Solution for eigenvalue problems $Ax = \lambda x$ for general dense, symmetric, banded matrix A,
  - Solution of general linear least squares, (GSEP), (GNEP), (GSVD),
  - Error bounds, condition estimation for (almost) everything,
  - Real and complex, single and double precision,
  - Easy-to-use drivers plus computational kernels.

- 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 – SUMMARY

- LAPACK success from HP-48G to CRAY C-90
  - (Trans)portable FORTRAN 77 (+BLAS), 766K lines (354K library, 412K testing and timing)
  - Targets workstations, vector machines, and shared memory computers
  - Initial release February 1992
  - 475,000 requests since then (as of 8/96)
  - Linear systems, least squares, eigenproblems
  - Manual available from SIAM
  - being translated into Japanese and Russian
  - Third release and manual in Fall/Winter, 1996

LAPACK – ONGOING WORK

- Add functionality (divide and conquer least squares, updating/downdating, bidiagonal bisection, bidiagonal inverse iteration, band SVD, Jacobi methods...)

- Move to new generation of high performance machines such as IBM SPs, CRAY T3D, Intel Paragon, clusters of workstations

- New challenges
  - New languages: FORTRAN 90, HP FORTRAN, ...
  - (CMMD, MPL, NX, PVM, MPI) many flavors of message passing:

$\rightarrow$ **BLACS**

  - Highly varying ratio $\frac{\text{computation speed}}{\text{communication speed}}$,
  - Many ways to layout data,
  - Fastest parallel algorithm sometimes less stable numerically.

$\rightarrow$ **ScaLAPACK**

LAPACK – REFERENCES

- LAPACK software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/lapack`,
  - WWW: `http://www.netlib.org/lapack/lawns`,
  - WWW: `http://www.netlib.org/clapack`,
  - WWW: `http://www.netlib.org/c++/lapack++`,
  - WWW: `http://www.netlib.org/lapack90`,
  - (anonymous) ftp `ftp.netlib.org`:
    * cd lapack; get index
    * cd lapack/lawns; get index
    * cd linalg; get index
    * cd clapack; get index
    * cd c++/lapack++; get index
    * cd lapack90; get index
  - email `netlib@www.netlib.org` with the message:

    `send index from lapack`

- Comments and questions can be addressed to

  `lapack@cs.utk.edu`

- E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide, Second Edition*, SIAM, Philadelphia, PA, 1995.

The Basic Linear Algebra Communication Subprograms

(BLACS)

BLACS – INTRODUCTION

- **A design tool**, they are a conceptual aid in design and coding.

- Associate widely recognized mnemonic names with communication operations, improve

  1. **program readability**,
  2. the **self-documenting quality** of the code.

- **Promote efficiency** by identifying frequently occurring operations of linear algebra which can be optimized on various computers.

- **Program portability** is improved through

  standardization of kernels

  without sacrificing efficiency since **optimized versions** are or will be **available**.

BLACS – BASICS

- Processes are embedded in a **2-dimensional 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"**. 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.

$\rightarrow$ use of `SCOPE` parameter.

BLACS – BASICS

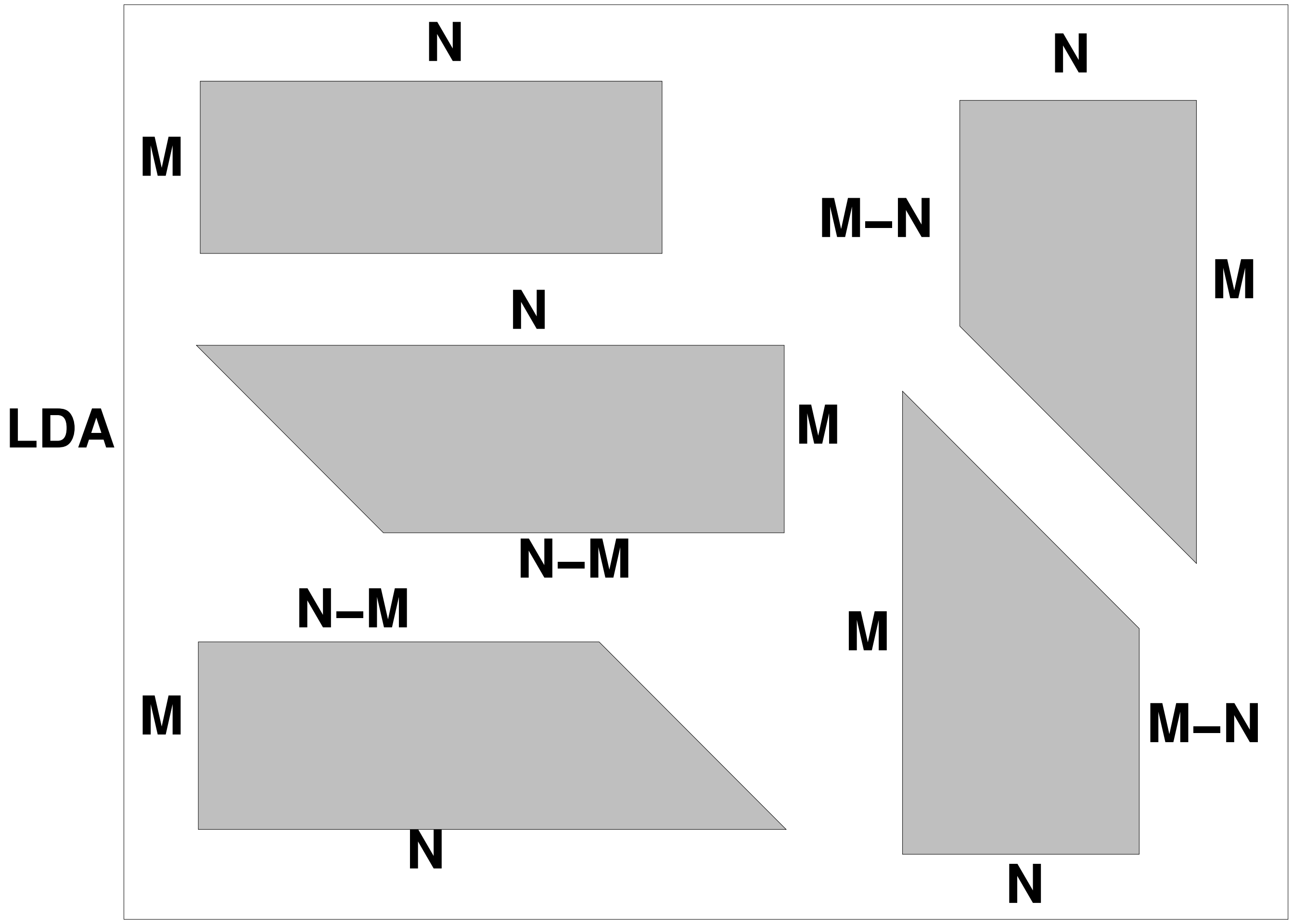
```
      CALL BLACS_PINFO( IAM, NPROCS )
*
*     If underlying system needs additional setup, do it now
*
      IF( NPROCS.LT.1 ) THEN
         IF( IAM.EQ.0 ) NPROCS = 4
         CALL BLACS_SETUP( IAM, NPROCS )
      END IF
*
*     Get default system context, define 1 x (NPROCS/2+1) process grid
*
      CALL BLACS_GET( 0, 0, ICONTXT )
*
      NPROW = 1
      NPCOL = NPROCS / 2 + 1
      CALL BLACS_GRIDINIT( ICONTXT, 'Row-major', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )
*
*     If I'm not in grid, go to end of program
*
      IF( MYROW.EQ.-1 )  GO TO 10

      ...

      CALL BLACS_GRIDEXIT( ICONTXT )
*
   10 CONTINUE
*
      CALL BLACS_EXIT( 0 )
*
      END
```

BLACS – BASICS

- **Operations are matrix based and ID-less**



- **4 Types of BLACS routines: point-to-point communication, broadcast, combine operations and support routines.**

BLACS – COMMUNICATION ROUTINES

1. Send/Recv

Send (sub)matrix from one process to another:

```
□xxSD2D( ICONTXT, [UPLO, DIAG,] M, N, A, LDA, RDEST, CDEST )
□xxRV2D( ICONTXT, [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.

```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
   CALL DGESD2D( ICONTXT, 5, 1, X, 5, 1, 0 )
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
   CALL DGERV2D( ICONTXT, 5, 1, Y, 5, 0, 0 )
END IF
```

BLACS – COMMUNICATION ROUTINES

2. Broadcast

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

```
□xxBS2D( ICONTXT, SCOPE, TOP, [UPLO, DIAG,] M, N, A, LDA )

□xxBR2D( ICONTXT, SCOPE, TOP, [UPLO, DIAG,] M, N, A, LDA,
         RSRC, CSRC )
```

SCOPE	TOP
'Row'	' ' (default)
'Column'	'Increasing Ring'
'All'	'1-tree' ...

```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 ) THEN
   IF( MYCOL.EQ.0 ) THEN
      CALL DGEBS2D( ICONTXT, 'Row', ' ', 2, 2, A, 3 )
   ELSE
      CALL DGEBR2D( ICONTXT, 'Row', ' ', 2, 2, A, 3, 0, 0 )
   END IF
END IF
```

BLACS – COMBINE OPERATIONS

3. Global combine operations

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

```
□GSUM2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,
         RDEST, CDEST )

□GAMX2D( ICONTXT, SCOPE, TOP, M, N, A, LDA,
         RA, CA, RCFLAG, RDEST, CDEST )

□GAMN2D( ICONTXT, 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, these arrays are set on output with the coordinates of the process owning the corresponding maximum (resp. minimum) element in absolute value of `A`. Both arrays must be at least of size $M \times N$ and their leading dimension is specified by `RCFLAG`.

BLACS – COMBINE OPERATIONS

```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 ) THEN
   K00 = MYCOL
   CALL IGSUM2D( ICONTXT, 'Row', ' ', 1, 1, K00, 1, 0, 0 )

   KALL = MYCOL
   CALL IGSUM2D( ICONTXT, '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( ICONTXT, 'All', ' ', 1, 1, EPS, 1,
 $              IRSRC, ICSRC, -1, -1, 0 )
```

BLACS – ADVANCED TOPICS

The **BLACS context** is the BLACS mechanism for partitioning communication space. A defining property of a context is that a message in a context cannot be sent or received in another context. The BLACS context includes the definition of a grid, and each process coordinates in it.

It 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.

$\rightarrow$ the BLACS context is compatible with the MPI communicator

BLACS – ADVANCED TOPICS

The BLACS provide "globally-blocking" point-to-point receive, broadcasts and combines. The BLACS point-to-point send is "locally-blocking":

- **Globally blocking**: The return from the operation implies that the buffer is available for re-use. The operation may not complete unless the complement of the operation is called.
- **Locally blocking**: The return from the send implies that the buffer is available for re-use. The send will complete regardless of whether the corresponding receive is posted.

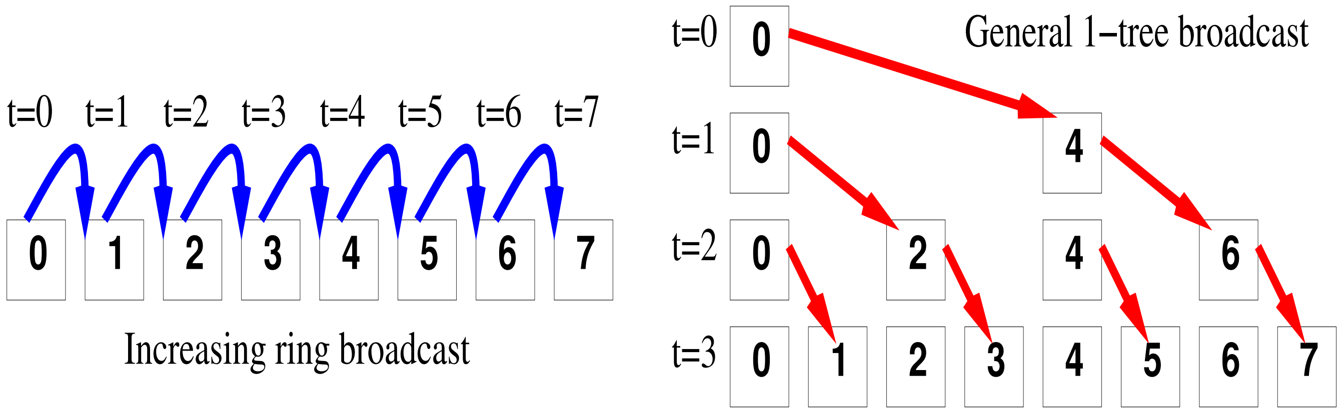
```
CALL BLACS_GRIDINFO( ICONTXT, NPROW, NPCOL, MYROW, MYCOL )

IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
   CALL DGESD2D( ICONTXT, 5, 1, X, 5, 1, 0 )
   CALL DGERV2D( ICONTXT, 5, 1, X, 5, 1, 0 )
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
   CALL DGESD2D( ICONTXT, 5, 1, X, 5, 0, 0 )
   CALL DGERV2D( ICONTXT, 5, 1, X, 5, 0, 0 )
END IF
```

BLACS – ADVANCED TOPICS

The **topology parameter** allows the user to vary the cost per process of the messages involved in a distributed (broadcast or combine) operation. There are two main classes of topologies:

- Pipelining topologies (ring-based),
- Non-pipelining topologies (tree-based).



$\rightarrow$ **topologies enhance performance.**

BLACS – EXAMPLES

- **HELLO** is one of the simplest BLACS programs. This program takes the available processes, forms them into a process grid, and then has each process check in with the process at (0,0) in the process grid.

- **PI** calculates the value of pi, using numerical integration with parallel processing, and clocks the solution time.

- **PROCMAP** illustrates how to map processes to a grid using BLACS_GRIDMAP. This is a fundamental example for whoever is interested in using simultaneously multiple (possibly overlapping) grids.

BLACS – REFERENCES

- BLACS software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/blacs`,
  - (anonymous) ftp `ftp.netlib.org`: `cd blacs; get index`
  - email `netlib@www.netlib.org` with the message:

    `send index from blacs`

- Comments and questions can be addressed to

  `blacs@cs.utk.edu`

- J. Dongarra and R. van de Geijn, *Two dimensional Basic Linear Algebra Communication Subprograms*, Technical Report UT CS-91-138, LAPACK Working Note #37, University of Tennessee, 1991.
- R. C. Whaley, *Basic Linear Algebra Communication Subprograms: Analysis and Implementation Across Multiple Parallel Architectures*, Technical Report UT CS-94-234, LAPACK Working Note #73, University of Tennessee, 1994.
- J. Dongarra and R. C. Whaley, *A User's Guide to the BLACS v1.0*, Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennessee, 1995.
- Message Passing Interface Forum, *MPI: A Message Passing Interface Standard*, *International Journal of Supercomputer Applications and High Performance Computing*, 8(3–4), 1994.
- A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Cambridge, Massachusetts, 1994.

The Parallel Basic Linear Algebra Subprograms

(PBLAS)

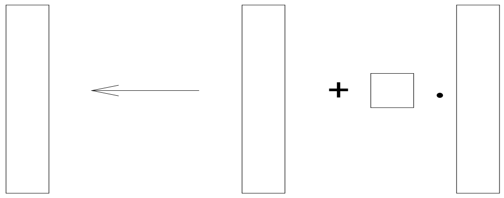
PBLAS – INTRODUCTION

Parallel Basic Linear Algebra Subprograms
for distributed memory MIMD computers.

- **Similar functionality** as the BLAS: distributed vector-vector, matrix-vector and matrix-matrix operations,
- **Simplification of the parallelization** of dense linear algebra codes: especially when implemented on top of the BLAS,
- **Clarity:** code is shorter and easier to read,
- **Modularity:** gives programmer larger building blocks,
- **Program portability:** machine dependencies are confined to the PBLAS (BLAS and BLACS).

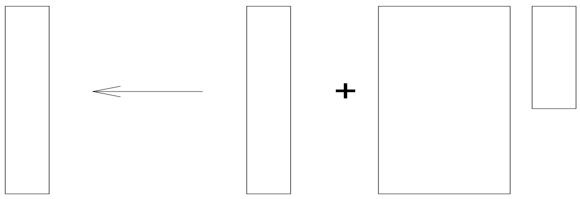
SCOPE OF THE PBLAS

Level 1 - Vector-vector operations



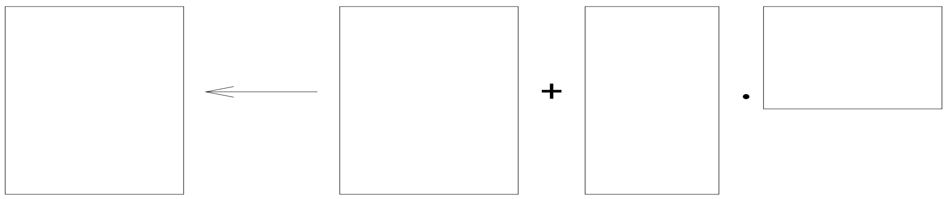
$$y \leftarrow y + \alpha x,\ dot \leftarrow x^T y,\ \ldots$$

Level 2 - Matrix-vector operations



$y \leftarrow y + Ax$, also triangular solve, rank-1 update

Level 3 - Matrix-matrix operations



$C \leftarrow C + AB$, block triangular solve, rank-k update

- No vector rotations, no dedicated subprograms for banded or packed matrices,
- Matrix transposition available.

PBLAS – NAMING CONVENTIONS

The PBLAS naming scheme follows the conventions of the BLAS, with necessary adaptations:

P□XXYYY

□ (Data type)	XX (Type of matrix operands)
S: REAL, D: DOUBLE PRECISION, C: COMPLEX, Z: DOUBLE COMPLEX.	GE: All matrix operands are GEneral rectangular, HE: One of the matrix operands is HErmitian, SY: One of the matrix operands is SYmmetric, TR: One of the matrix operands is TRiangular.

- The Level 1 routines use meaningful abbreviations, for example PSCOPY, PSDOT . . .
- The matrix transposition routine is called P□TRAN□.

YYY	(Operation)
MM	Matrix-matrix product;
MV	Matrix-vector product;
R	Rank-1 update of a matrix;
R2	Rank-2 update of a matrix;
RK	Rank-k update of a symmetric or Hermitian matrix;
R2K	Rank-$2k$ update of a symmetric or Hermitian matrix;
SM	Solves a system of linear equations for a matrix of right-hand sides;
SV	Solves a system of linear equations for a right-hand side vector.

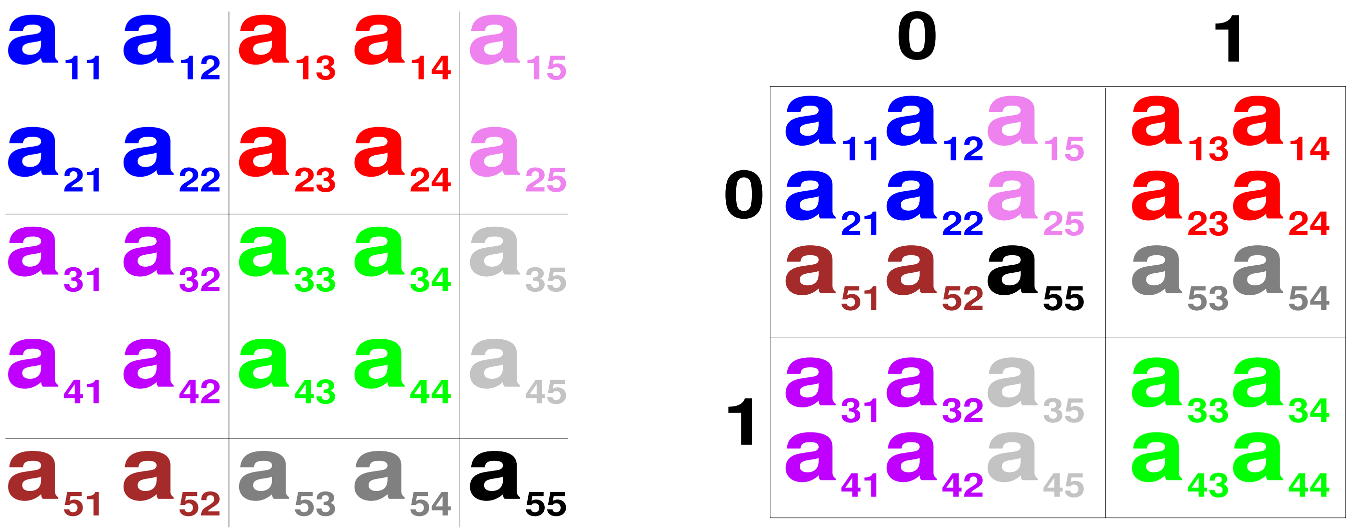
PBLAS – STORAGE CONVENTIONS

- An `M_`-by-`N_` **matrix is block partitioned** and these `MB_`-by-`NB_` blocks are distributed according to the

2-dimensional block-cyclic scheme
$\Rightarrow$ **load balanced computations, scalability**

- Locally, the scattered **columns are stored contiguously** (FORTRAN "Column-major")

$\Rightarrow$ **re-use of the BLAS** (leading dimension `LLD_`).



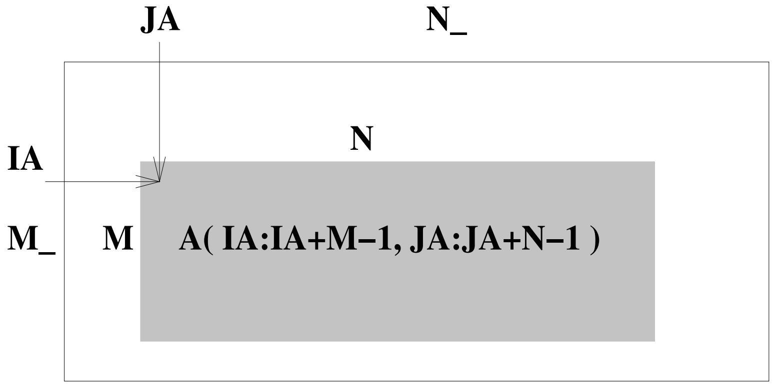
5 x 5 matrix partitioned in 2 x 2 blocks   2 x 2 process grid point of view

Descriptor `DESC_`**:** 9-Integer array describing the matrix layout, containing `DTYPE_`, `CTXT_`, `M_`, `N_`, `MB_`, `NB_`, `RSRC_`, `CSRC_`, and `LLD_`, where (`RSRC_`, `CSRC_`) are the coordinates of the process owning the first matrix entry in the grid specified by `CTXT_`.

Ex: `M_=N_=5`, `MB_=NB_=2`, `RSRC_=CSRC_= 0`, `LLD_` $\geq$ `3` (in process row 0), and `2` (in process row 1).

PBLAS – ARGUMENT CONVENTIONS

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



- **Code reusability**, interface very close to sequential BLAS:

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

 CALL  DGEMM( 'No Transpose', 'No Transpose',
$    M-J-JB+1, N-J-JB+1, JB, -ONE, A(J+JB,J),
$    LDA, A(J,J+JB), LDA, ONE, A(J+JB,J+JB),
$    LDA )
```

↓

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

 CALL PDGEMM( 'No Transpose', 'No Transpose',
$    M-J-JB+1, N-J-JB+1, JB, -ONE, A, J+JB,
$    J, DESCA, A, J, J+JB, DESCA, ONE, A,
$    J+JB, J+JB, DESCA )
```

PBLAS – SPECIFICATIONS

```
DGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX,
       BETA, Y, INCY )
```

$$\updownarrow$$

```
PDGEMV( TRANS, M, N, ALPHA, A, IA, JA, DESCA,
        X, IX, JX, DESCX, INCX,
        BETA, Y, IY, JY, DESCY, INCY )
```

- In the PBLAS, the increment specified for vectors is always global. So far only `INCX=1` and `INCX=DESCX(3)` are supported.

BLAS	PBLAS
`INTEGER LDA`	`INTEGER IA, JA, DESCA( 9 )`
`INTEGER INCX`	`INTEGER INCX, IX, JX, DESCX( 9 )`
`A, LDA`	`A, IA, JA, DESCA`
`X, INCX`	`X, IX, JX, DESCX, INCX`

- PBLAS matrix transposition routine (REAL):

```
PDTRAN( M, N, ALPHA, A, IA, JA, DESCA, BETA,
        C, IC, JC, DESCC )
```

- Level 1 BLAS functions have become PBLAS subroutines. Output scalar correct in operand scope.

>> SPMD PROGRAMMING MODEL <<

PBLAS – AUXILIARY SUBPROGRAMS

- Using the BLACS topologies for efficiency and scalability:

```
SUBROUTINE PTOPSET( ICTXT, OP, SCOPE, TOP )
SUBROUTINE PTOPGET( ICTXT, OP, SCOPE, TOP )

INTEGER             ICTXT
CHARACTER*1         OP, SCOPE, TOP
```

  `PTOPSET` assigns the BLACS topology `TOP` to be used in the communication operations `OP` along the scope specified by `SCOPE`. `PTOPGET` retrieves the topology.

  These routines should be called by every process in the grid identified by the BLACS context `ICTXT`.

- Disposing the PBLAS buffer allocated in every process's dynamic memory:

```
SUBROUTINE PBFREEBUF()
```

PBLAS – RATIONALE

- **Software reusability**: BLAS and PBLAS interfaces similar,

- Level 1 PBLAS: Output scalar is only correct in operand scope,

- Level 3 PBLAS uses **"shaped adaptive" procedure** to perform the computations,

- The PBLAS routines follow an **SPMD** programming model (every process in the grid must call the routine).

- The BLACS context provides the ability to have separate "universes" of message passing:

  Safe co-existence with other parallel libraries,

- The PBLAS do not perform "inter-context" operations.

PBLAS – EXAMPLES

```
      INTEGER   IAM, ICTXT, INFO, NMAX, LDA, LDB, LDC, NMAX, NPROCS
      PARAMETER ( NMAX = 3, LDA = NMAX, LDB = NMAX, LDC = NMAX )
      INTEGER   DESCA( 9 ), DESCB( 9 ), DESCC( 9 )
      DOUBLE PRECISION   A( NMAX, NMAX ), B( NMAX, NMAX ),
     $                   C( NMAX, NMAX )

      CALL BLACS_PINFO( IAM, NPROCS )
      IF( NPROCS.LT.1 ) THEN
         NPROCS = 4
         CALL BLACS_SETUP( IAM, NPROCS )
      END IF
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, 'Row-major', 2, 2 )

      (...)
      CALL DESCINIT( DESCA, 5, 5, 2, 2, 0, 0, ICTXT, LDA, INFO )
      CALL DESCINIT( DESCB, 5, 5, 2, 2, 0, 0, ICTXT, LDB, INFO )
      CALL DESCINIT( DESCC, 5, 5, 2, 2, 0, 0, ICTXT, LDC, INFO )
      (...)

      CALL PDGEMM( 'No transpose', 'No transpose', 4, 4, 4,
     $             1.0D+0, A, 1, 1, DESCA, B, 1, 1, DESCB,
     $             0.0D+0, C, 1, 1, DESCC )
      (...)

      CALL PBFREEBUF()
      CALL BLACS_GRIDEXIT( ICTXT )
      CALL BLACS_EXIT( 0 )
```

PBLAS – EXAMPLES

- **PBLAS** shows how to set the matrix descriptors and call the Level 1, 2 and 3 PBLAS routines.

- **SUMMA** illustrates the use of the BLAS and the BLACS to implement an efficient and scalable matrix-matrix multiply routine.

  This implementation was first suggested by R. Agarwal, F. Gustavson, and M. Zubair, in *A High Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication*, IBM J. Res. Develop., Vol. 38, No. 6 (1994), pp. 673-681.

PBLAS – REFERENCES

- PBLAS software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/scalapack`,
  - (anonymous) ftp `ftp.netlib.org`:

    `cd scalapack; get index`
  - email `netlib@www.netlib.org` with the message:

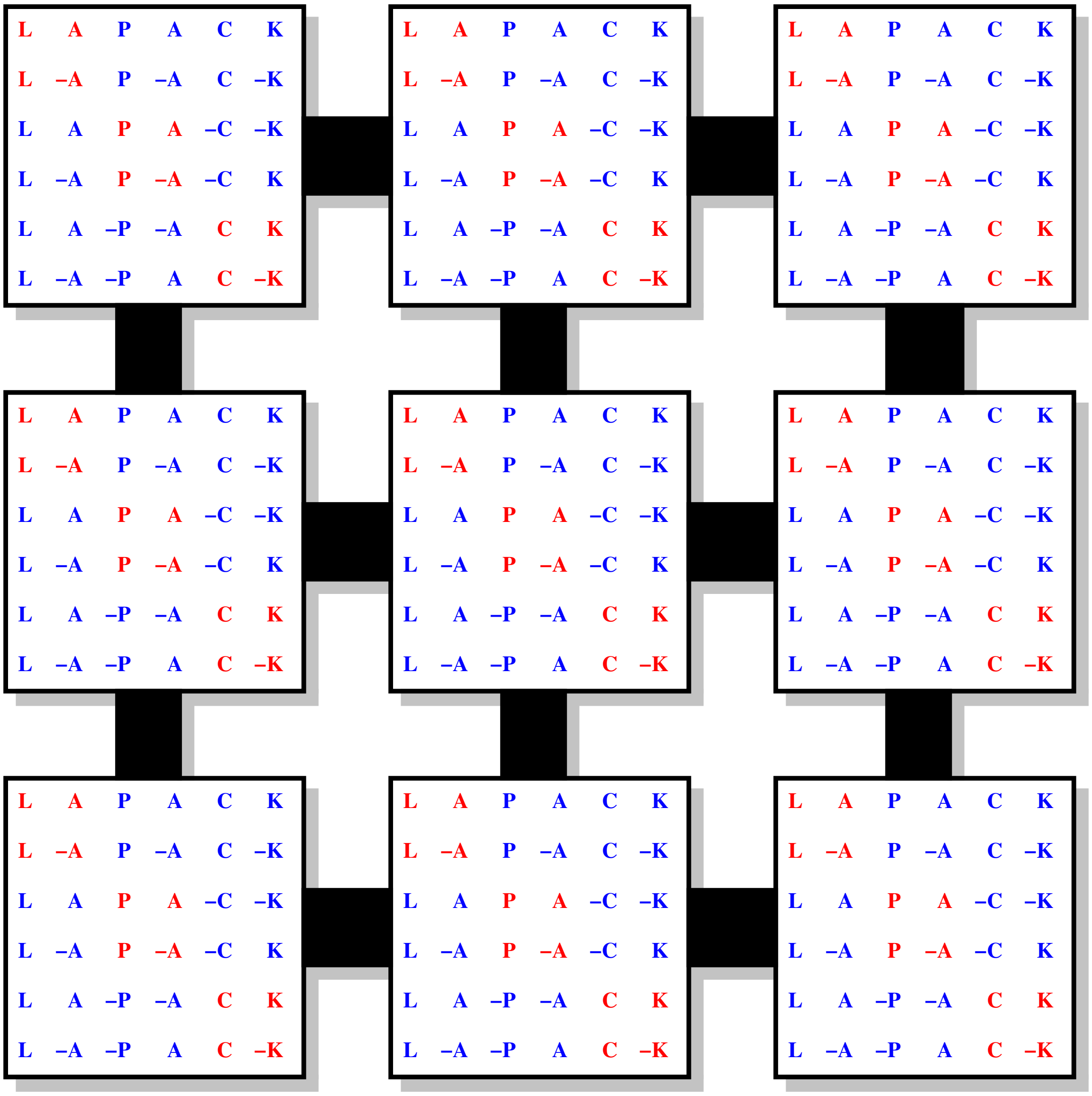
    `send index from scalapack`
- Comments and questions can be addressed to

  `scalapack@cs.utk.edu`

- J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker and R. C. Whaley, *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Technical Report UT CS-95-292, LAPACK Working Note #100, University of Tennessee, 1995.

The Scalable Linear Algebra PACKage

(ScaLAPACK)

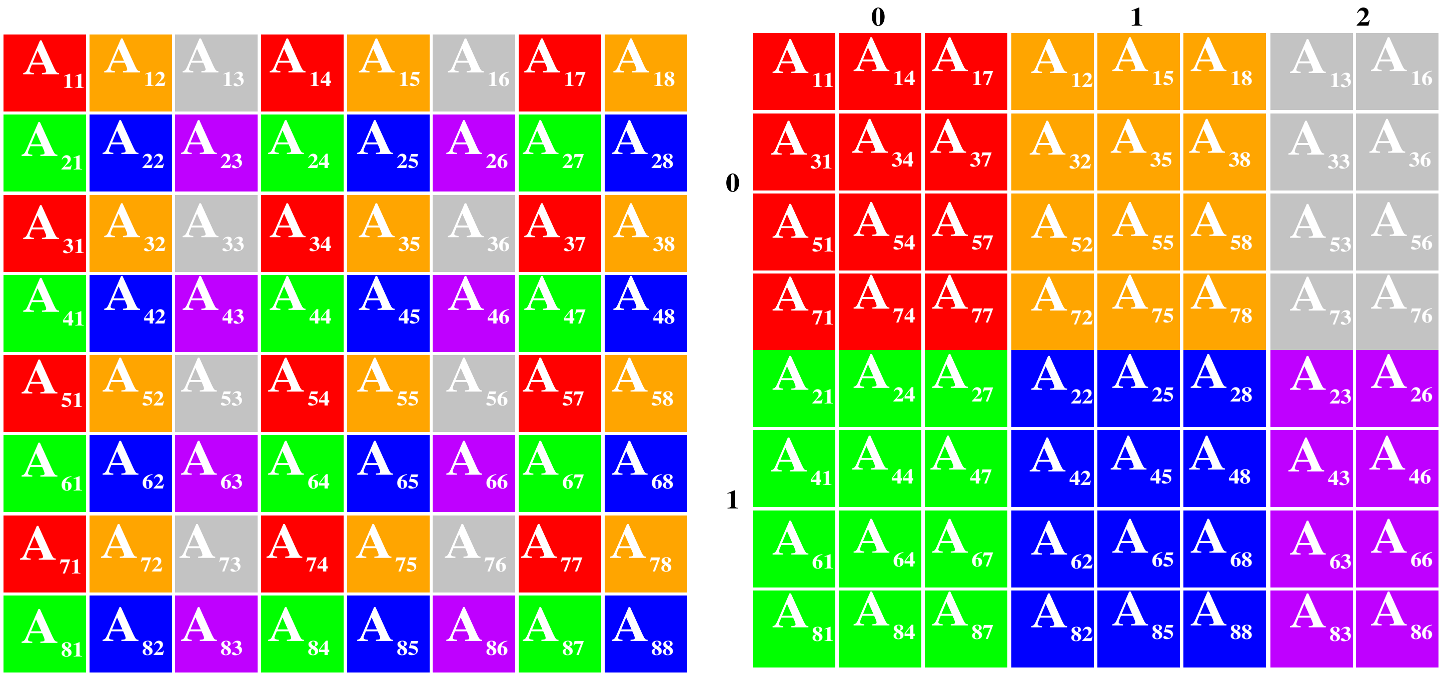


SCALAPACK – INTRODUCTION

Goal: Port Linear Algebra PACKage (LAPACK) to distributed-memory environments.

- **Scalability and Performance**
  - Optimized compute and communication engines
  - Blocked data access (Level 3 BLAS) yields good node performance
  - 2-D Block-cyclic data decomposition yields good load balance
- **Portability**
  - BLAS: optimized compute engine
  - BLACS: communication kernel
- **Simple calling interface**
  - As similar as possible to LAPACK in calling sequence, storage, etc.
- **Ease of Programming / Maintenance**
  - Modularity: Build rich set of linear algebra tools: BLAS, BLACS, PBLAS
  - Whenever possible, use reliable LAPACK algorithms.

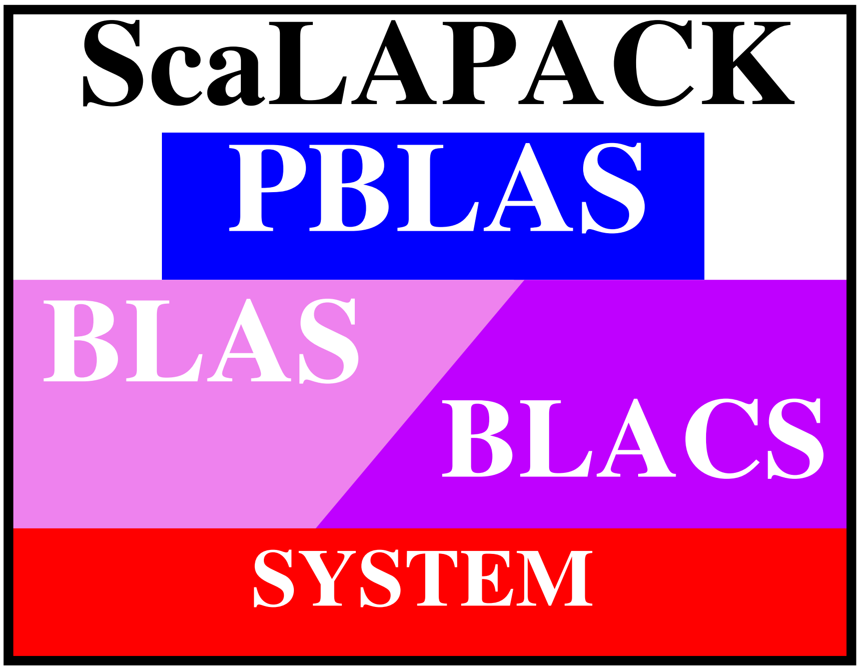
2-DIMENSIONAL BLOCK CYCLIC DISTRIBUTION



Global (left) and distributed (right) views of matrix

- **Ensure good load balance ⇒ Performance and scalability**,
- Encompass a large number of (but not all) data distribution schemes,
- Need redistribution routines to go from one distribution to the other,

ScaLAPACK IMPLEMENTATION



BLAS (Performance and Portability)

- Blocked data access (Level 3 BLAS) yields good local performance,
- **Portability:** standardization of efficient kernels.

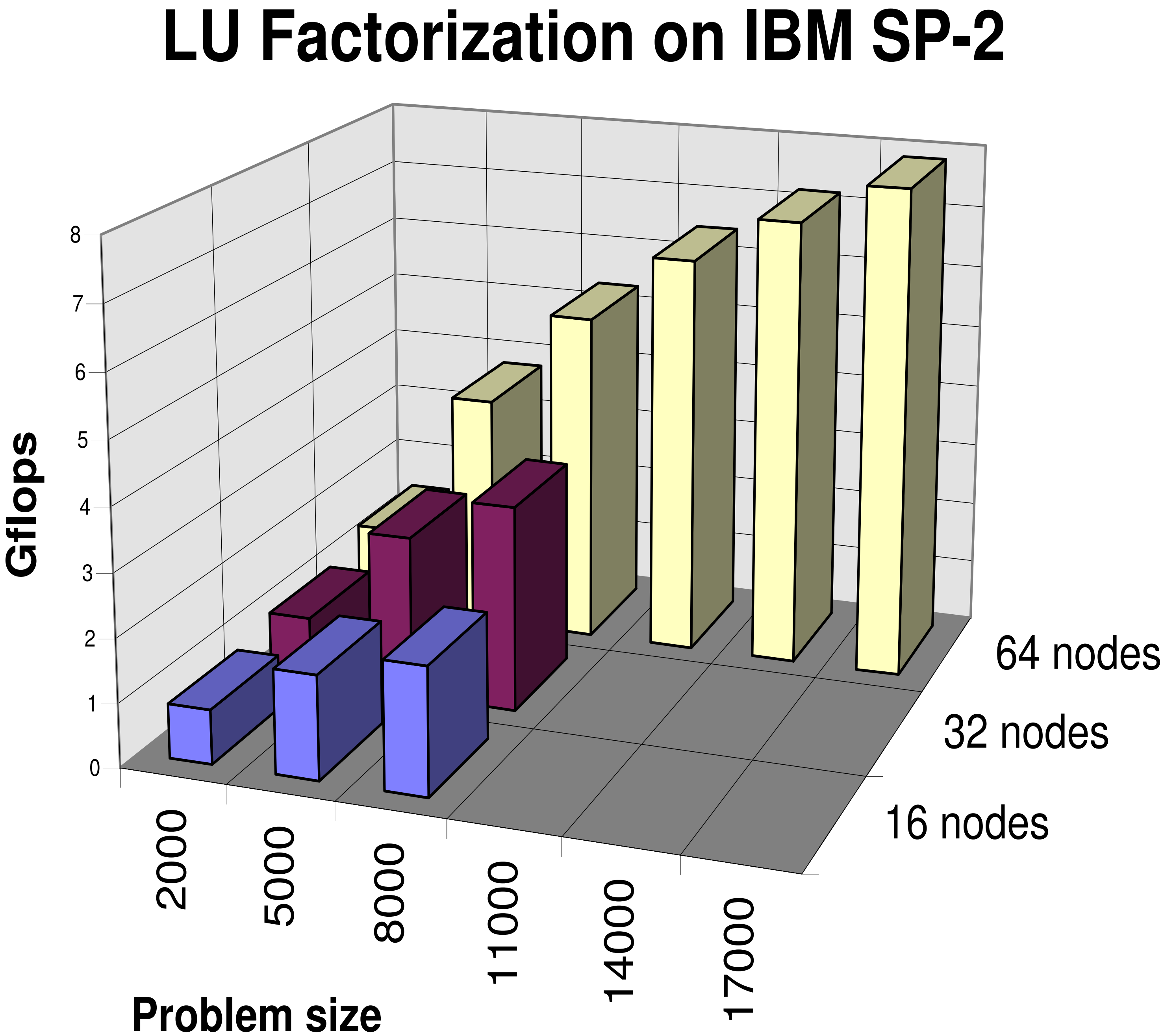
BLACS (Performance and Portability)

- **Correct level of notation:** communication of matrices,
- **Program readability:** simplicity and mnemonic naming scheme → self-documenting quality of code,
- **Efficiency:** identify frequent linear algebra operations which can be optimized on various computers,
- **Portability:** standardization of efficient kernels.

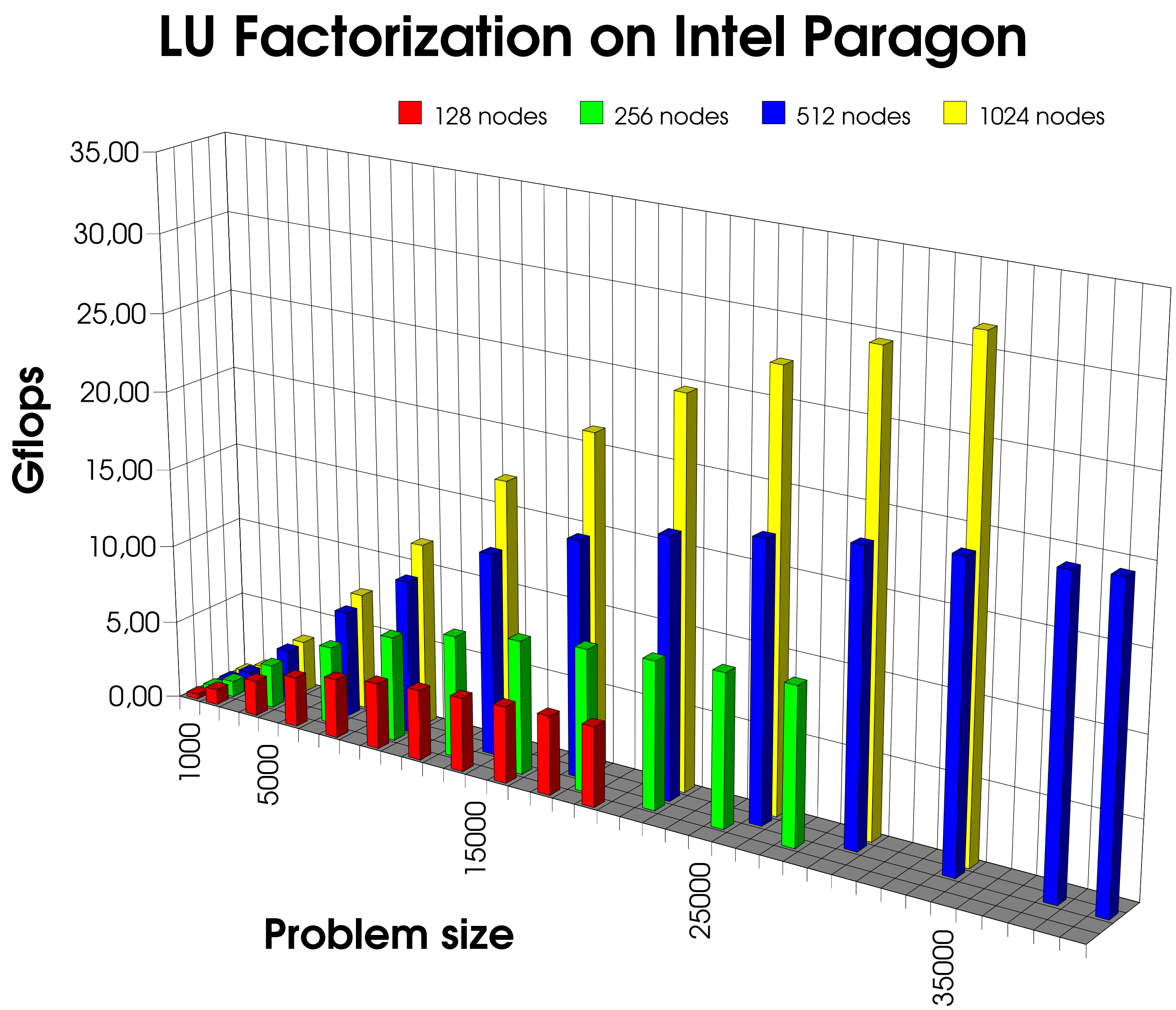
WHAT IS AVAILABLE TO YOU ?

- **Tools**
  - BLACS (CMMD, MPL, NX, PVM, MPI)
  - PBLAS
  - REDISTRIBUTION
- **Factorizations**
  - LU + solve + cond. est. + it. ref.
  - Cholesky + solve + cond. est. + it. ref.
  - Band Cholesky + solve
  - QR, QL, LQ, RQ, QRP
  - Generalized QR and RQ
- **Linear Least Squares solvers (full-rank)**
- **Matrix inversions**
- $Q, QC, Q^T C, CQ, CQ^T$
- **Reductions**
  - to (upper) Hessenberg form
  - to symmetric tridiagonal form
  - to bidiagonal form
  - to standard form
- **Symmetric eigensolver**
- **Generalized Symmetric eigensolver**
- **Nonsymmetric eigensolver**

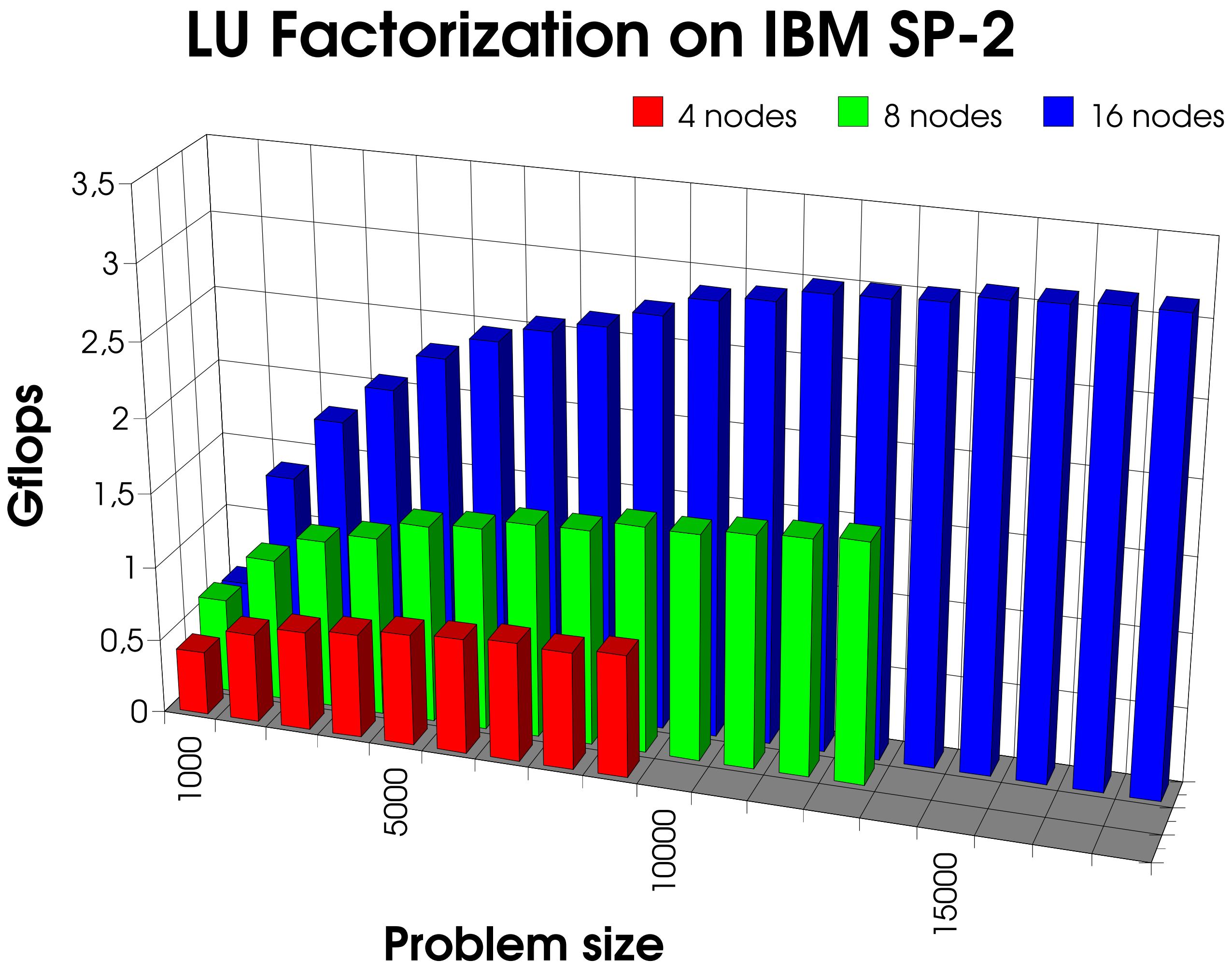
PERFORMANCE RESULTS



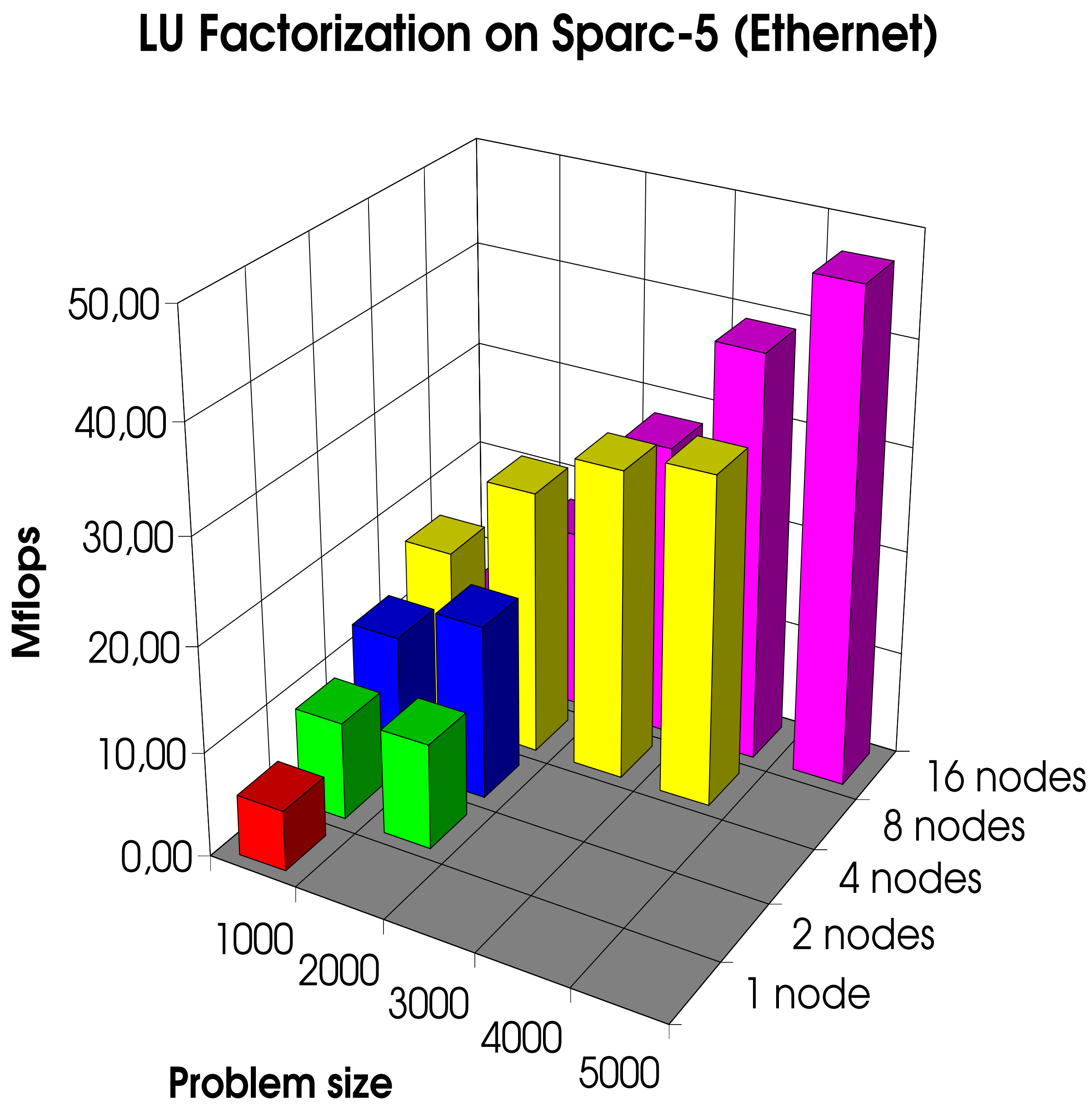
PERFORMANCE RESULTS

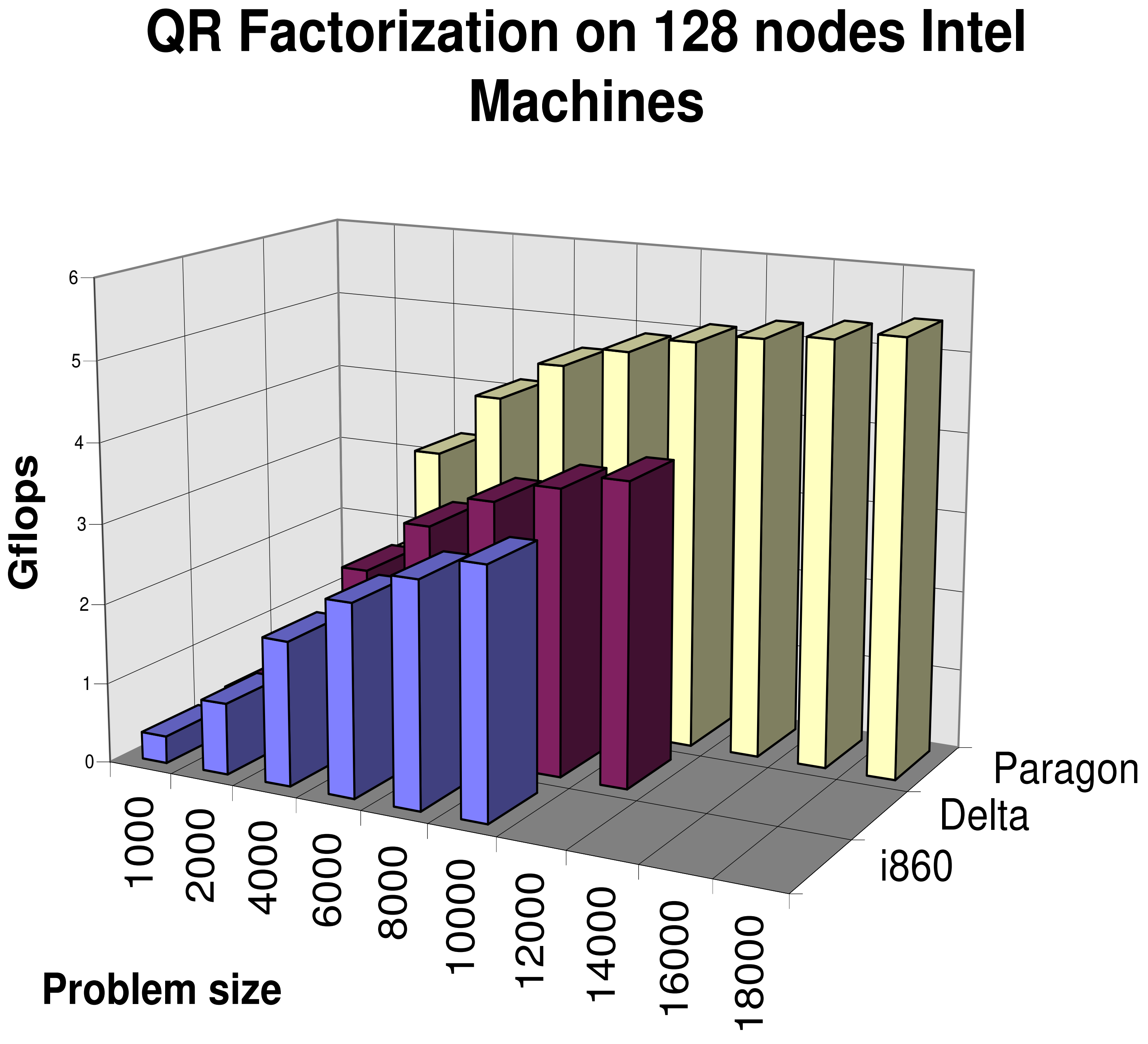


PERFORMANCE RESULTS

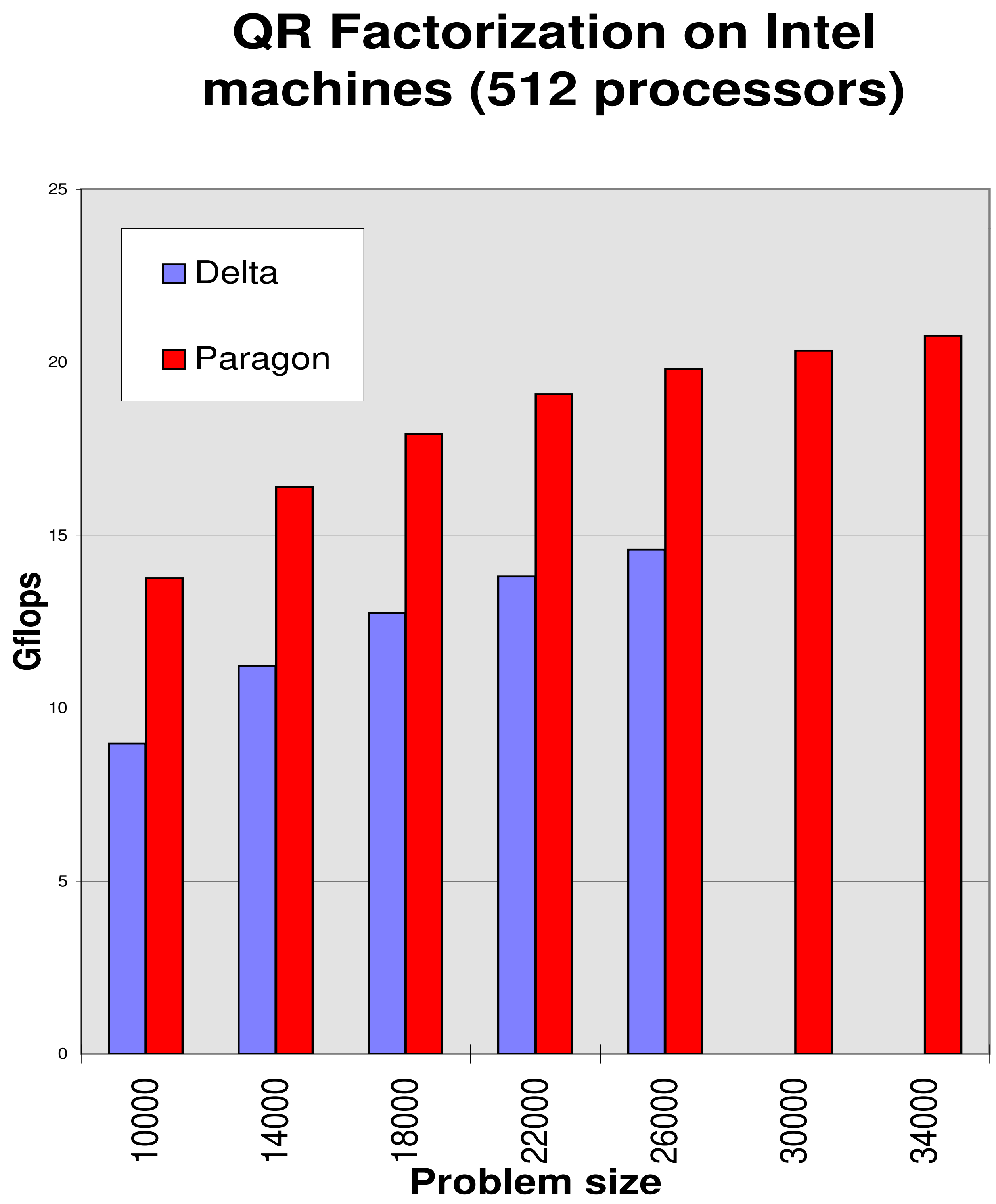


PERFORMANCE RESULTS




QR Factorization on 128 nodes Intel Machines
Gflops
0
1
2
3
4
5
6
Problem size
1000
2000
4000
6000
8000
10000
12000
14000
16000
18000
Paragon
Delta
i860

PERFORMANCE RESULTS



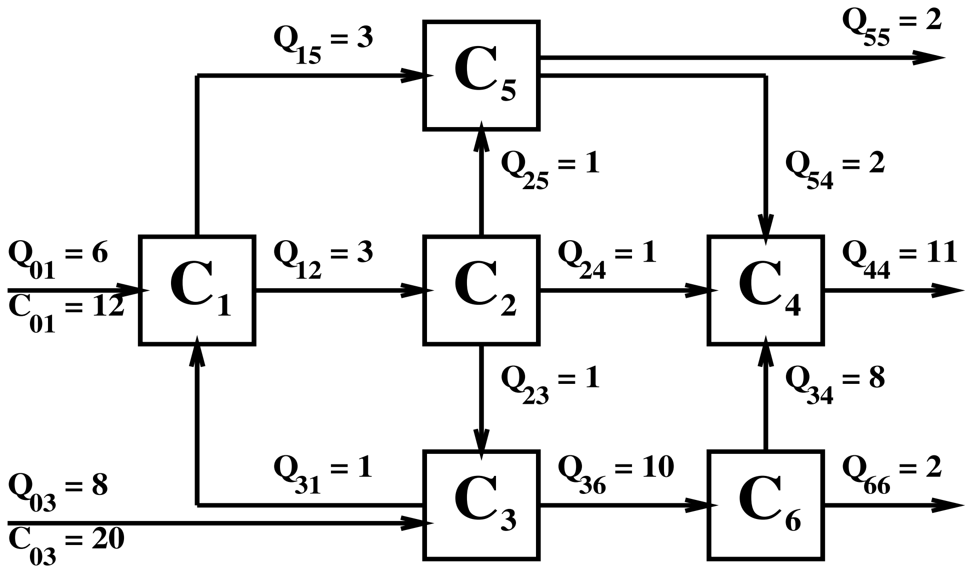
SCALAPACK – APPLICATIONS

- Solution of boundary integral equations, boundary element methods,
- Airplane wing design,
- Radar cross-section studies (Helmholtz equation, method of moments),
- Supercomputer benchmarking,
- Flow around ships and off-shore constructions,
- Diffusion of solid bodies in a liquid,
- Noise reduction,
- Diffusion of light through small particles,
- Reduced sparse problems, …

SCALAPACK – EXAMPLE

SCAEX shows how to call the ScaLAPACK routine `PDGESV` to solve a linear system of equations. The matrix and the right-hand-side are read from a file. The vector solution is written to a file.

Analysis of a Processing Plant Consisting of Interconnected Reactors



Problem: Concentration of the mixture into and out of the reactors ?

Application of conservation of mass to all these reactors:

Q_{ij}: volumetric flow rate, C_i: concentration at section i

For reactor 1: $Q_{31}C_3 + Q_{01}C_{01} = (Q_{12} + Q_{15})C_1$

$$\begin{pmatrix} 6 & 0 & -1 & 0 & 0 & 0 \\ 3 & -3 & 0 & 0 & 0 & 0 \\ 0 & -1 & 11 & 0 & 0 & 0 \\ 0 & 1 & 0 & -11 & 2 & 8 \\ 3 & 1 & 0 & 0 & -4 & 0 \\ 0 & 0 & 10 & 0 & 0 & -10 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \end{pmatrix} = \begin{pmatrix} 72 \\ 0 \\ 160 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

SCALAPACK – ONGOING WORK

- **HPF**
  - HPF supports the 2D-block-cyclic distribution used by ScaLAPACK
  - HPF like calling sequence (Global indexing scheme)
- **More flexibility added to the PBLAS**
- **SVD**
- **Linear Least Square solvers (rank-deficient)**
- **General banded systems**
- **Tridiagonal solvers**
- **QR-based symmetric eigensolvers**
- **Out of core linear system solvers**
- ...

SCALAPACK – REFERENCES

- ScaLAPACK software and documentation can be obtained via:
  - WWW: `http://www.netlib.org/scalapack`,
  - WWW: `http://www.netlib.org/lapack/lawns`,
  - (anonymous) ftp `ftp.netlib.org`:
    - cd scalapack; get index
    - cd lapack/lawns; get index
  - email `netlib@www.netlib.org` with the message:

    `send index from scalapack`

- Comments and questions can be addressed to

  `scalapack@cs.utk.edu`

- LAPACK Working Notes:
  #43, #55, #57, #58, #61, #65, #73, #80, #86, #91, #92, **#93**, **#94**, **#95**, #96, **#100**.
- J. Dongarra and D. Walker, *Software Libraries for Linear Algebra Computations on High Performance Computers*, SIAM Review, Vol. 37, (2), pp. 151 – 180, 1995
- S. Chapara and R. Canale, *Numerical Methods for Engineers*, Mc Graw-Hill, New York, 1988
- J. Wang, *Generalized Moment Methods in Electromagnetics*, John Wiley & Sons, New York, 1991

An HPF Interface for ScaLAPACK

...or ...

Minimal Features Needed To Interface HPF with Message Passing Libraries

HPF INTERFACE – FOREWORD

- **Our Experience:** no presently available compiler supports the full features of HPF.

- **Determining** the minimal features required to efficiently support a HPF library which calls a process-local message passing library.

HPF INTERFACE – OUTLINE

- **ScaLAPACK Overview**
- **Avoiding Redistribution**
- **Determining Operand Distribution**
- **Conclusions**

ScaLAPACK – OVERVIEW

Linear Algebra Software Library for Distributed Memory Concurrent Computers

- **BLACS (Message Passing Layer)**
- **PBLAS** $(C = \beta C + \alpha AB)$
- **Linear Equations Solvers** $(Ax = b)$
- **Eigensolvers** $(Av = \lambda v)$

- **Written in FORTRAN 77 and C**
- **FORTRAN 77 interface, C callable**

- **2D Block Cyclic Data Distribution:**
  `!HPF$ DISTRIBUTE A( CYCLIC(MB), CYCLIC(NB) )`
- **Include** `BLOCK` **and collapsed storage**

HPF – REDISTRIBUTION

- **Expensive** in both Space (memory) and Time,
- Users often utilize all available memory,
- ScaLAPACK performance is not very sensitive for a large subset of all supported cyclic mappings.

Use !HPF$ INHERIT **for most general purpose libraries:**

- Data will be left in place across subroutine calls,
- Whenever possible data passed to ScaLAPACK without redistribution,
- Otherwise, redistribute to an optimal distribution.

HPF – DETERMINING DISTRIBUTION

1. `HPF_DISTRIBUTE` from `HPF_LIBRARY` returns:
   - Distribution Type (`CYCLIC`, `BLOCK`, ...)
   - Row and column blocking factors (`MB`, `NB`),
   - Size of the process grid,

   of the **ultimate align target**.

   Use then `HPF_ALIGNMENT` from `HPF_LIBRARY` to see how this information affects our matrix operand.

2. Find on what processor particular blocks of the matrix reside:
   - `GLOBAL_TO_LOCAL` from `HPF_LOCAL_LIBRARY` or
   - `!HPF$ ALIGN` and a mapping array.

3. If redistribution is found to be necessary, use

- `!HPF$ PROCESSORS PROC(P,Q)`
- `!HPF$ DISTRIBUTE(CYCLIC(MB),CYCLIC(NB)) ONTO PROC`

CALLING FORTRAN 77 FROM HPF

If FORTRAN 77 is called directly from HPF, one cannot determine the leading dimension of the local arrays,

$\Rightarrow$ Intermediate layer (HPF $\leftrightarrow$ FORTRAN 77).

- **LAYER 1**: User callable wrapper functions, and their HPF tools (written in strict HPF):
  - Determine if input data distribution is acceptable:
    * All matrix operands can be expressed as some variant of `CYCLIC(K)`,
    * All matrix operands are distributed across the same process grid,
    * Process grid is one or two dimensional,
    * Certain routine-dependent alignment restrictions are met.
  - Remap to an acceptable redistribution.

CALLING FORTRAN 77 FROM HPF

- **LAYER 2**: Transition layer
  - Extrinsic with local view of the matrix
    - `HPF_LOCAL` (layer 3 `F77_LOCAL` or `F90_LOCAL`),
    - `F90_LOCAL` (layer 3 `F77_LOCAL` or `F90_LOCAL`),
  - Translate HPF's assumed shape array to FORTRAN 77's assumed size array,
  - Determine leading dimension of assumed size arrays so FORTRAN 77 routines can do index computation.
- **LAYER 3**: ScaLAPACK message passing layer
  - Extrinsic F77_LOCAL or F90_LOCAL

HPF INTERFACE – SUMMARY

- `!HPF$ INHERIT` **to avoid unneeded redistribution across subroutine calls**,

- `HPF_DISTRIBUTION` from `HPF_LIBRARY` **to find out the distribution of the matrix's ultimate align target**,

- `HPF_ALIGNMENT` from `HPF_LIBRARY` **to figure out how the distribution of the ultimate align target affects the actual matrix distribution**,

- `!HPF$ PROCESSORS PROC(P,Q)`, where `P` and `Q` are dummy arguments to a routine, when one has determined redistribution must occur, this command ensures that all matrices passed to the routine are distributed over the **same process grid**,

HPF INTERFACE – SUMMARY

- `!HPF$ DISTRIBUTE A(CYCLIC(MB),CYCLIC(NB)) ONTO PROC`, with `MB` and `NB` parameters to routine and `PROC` from above,

  to redistribute data according to 2D block cyclic scheme,

- `GLOBAL_TO_LOCAL` from `HPF_LOCAL_LIBRARY` or

- `!HPF$ ALIGN M(I,J) WITH A(1+(I-1)*MB, 1+(J-1)*NB)`, with `MB` and `NB` parameters to routine,

  to label the processes allowing for system independent process grid setup,

- `EXTRINSIC(F90_LOCAL)` or,

- `EXTRINSIC(HPF_LOCAL)` + `EXTRINSIC(F77_LOCAL)`

  to call FORTRAN 77 from HPF.