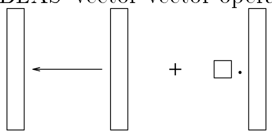
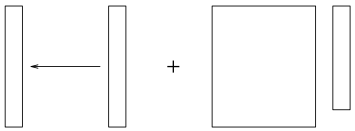
Level 1, 2, and 3 BLAS

- Level 1 BLAS–vector-vector operations



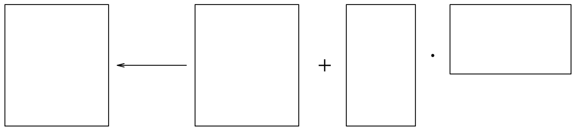
$y \leftarrow y + \alpha x$, also dot product, ...

- Level 2 BLAS–matrix-vector operations



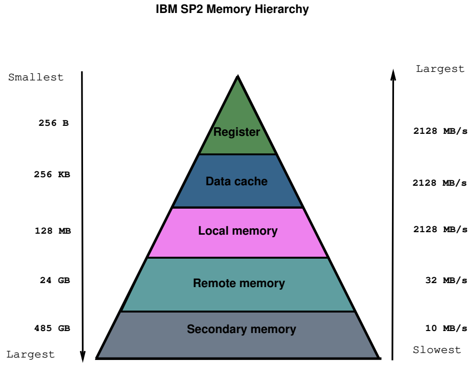
$y \leftarrow y + Ax$, also triangular solve, rank-1 update

- Level 3 BLAS–matrix-matrix operations



$A \leftarrow A + BC$, also block triangular solve, rank-k update

Why Higher Level BLAS?

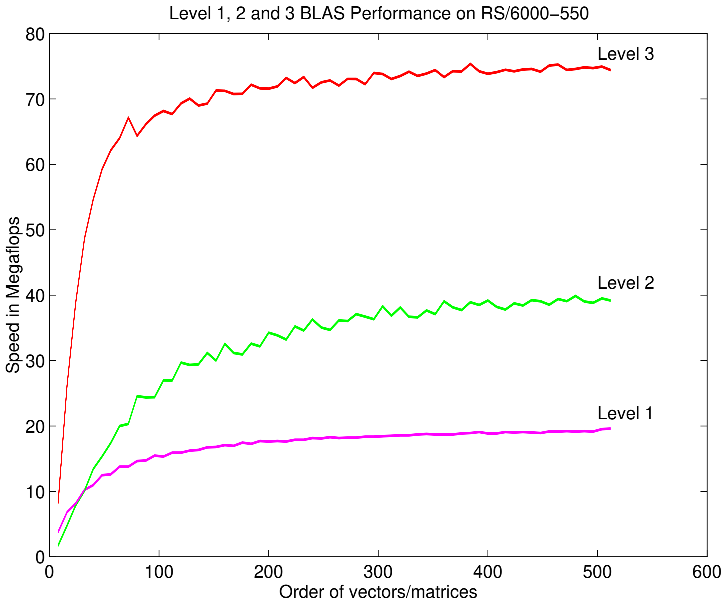


- Can only do arithmetic on data at top
  $\longrightarrow$ keep active data as close to top of hierarchy as possible
- Higher level BLAS lets us do this:

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

- On parallel machines
  higher level BLAS $\longrightarrow$increase granularity $\longrightarrow$ lower synchronization cost

BLAS – BASICS



- Development of blocked 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 `netlib2.cs.utk.edu`:

    `cd blas; get index`
  - email `netlib@ornl.gov` with the message:

    `send index from blas`
- Comments and questions can be addressed at

  `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.

Blocked Algorithms have been developed for:

- LU factorization
- Cholesky factorization
- Factorization of symmetric indefinite matrices
- Matrix inversion
- Banded LU and Cholesky factorization
- QR factorization
- Form Q or $Q^T B$
- Orthogonal reduction to:
  - Hessenberg
  - symmetric tridiagonal
  - bidiagonal
- Block QR iteration for nonsymmetric eigenvalue problems

LAPACK

- Linear algebra library in Fortran 77:
  - solution of systems of equations
  - solution for eigenvalue problems
- Combine algorithms from LINPACK and EISPACK into a single package
- Efficient on a wide range of high-performance computers
  - Single and double precision routines
  - Complex counterparts
- User interface similar to LINPACK
- Built on the Level 1, 2, and 3 BLAS

LAPACK – CONTENTS

- **Basic problems:**

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

- **Factorizations:**

  $A = LU,\ LL^T,\ QR,\ QRP,\ RQ,\ QL,\ LQ$
  $A = V\Lambda V^T$ (SEP and NEP), $A = U\Sigma V^T$ (SVD)
  $A - \lambda B = V(\Lambda_A - \lambda\Lambda_B)V^T, AB = V\Lambda V^{-1}$
  (GSEP, GNEP and GSVD)

- **Dense, banded, symmetric packed, . . .**
  (no general sparse yet)
- **Real and complex, single and double precision**
- **Error bounds, condition estimation** for (almost) everything
- **Easy-to-use drivers plus computational kernels**
- **Uses BLAS for high performance**
  (manufacturers optimize BLAS)
- **(Trans)portable FORTRAN 77 (+BLAS)**, 579K lines
  (248K library, 351K testing and timing).

LAPACK timing results for a cray Y/MP (Mflops)

Name		Matrix size N					
		32	64	128	256	512	1024
LU							
SGETRF	(1 proc)	40	108	195	260	290	304
	(2 proc)	32	91	229	408	532	588
	(4 proc)	32	90	260	588	914	1097
	(8 proc)	32	90	205	375	1039	1974
Cholesky							
SPOTRF	(1 proc)	34	95	188	259	289	301
	(2 proc)	29	84	221	410	539	594
	(4 proc)	29	84	252	598	952	1129
	(8 proc)	29	84	273	779	1592	2115
QR Factorization							
SGEQRF	(1 proc)	54	139	225	275	294	301
	(2 proc)	50	134	256	391	505	562
	(4 proc)	50	136	292	612	891	1060
	(8 proc)	50	133	328	807	1476	1937

The BLAS as the Key to Portability

Speed in megaflops of Level 2 and Level 3 BLAS operations on a CRAY C90

(all matrices are of order 1000; U is upper triangular)

Number of processors:	1	2	4	8	16
Level 2: $y \leftarrow \alpha Ax + \beta y$	899	1780	3491	6783	11207
Level 3: $C \leftarrow \alpha AB + \beta C$	900	1800	3600	7199	14282
Level 2: $x \leftarrow Ux$	852	1620	3063	5554	6953
Level 3: $B \leftarrow UB$	900	1800	3574	7147	13281
Level 2: $x \leftarrow U^{-1}x$	802	1065	1452	1697	1558
Level 3: $B \leftarrow U^{-1}B$	896	1792	3578	7155	14009

Level 1 BLAS : for vector operations, such as $y \leftarrow \alpha x + y$

Level 2 BLAS : for matrix-vector operations, such as $y \leftarrow \alpha Ax + \beta y$

Level 3 BLAS : for matrix-matrix operations, such as $C \leftarrow \alpha AB + \beta C$

Here, A, B and C are matrices, x and y are vectors, and α and β are scalars.

Block Algorithms and their Derivation

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

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

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

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

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

Here is the body of the code 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
```



```
   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
```

- change by itself is sufficient to make big gains in 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

To derive a block form of Cholesky factorization, we write the defining equation in partitioned form thus:

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

Equating submatrices in the second block of columns, we obtain:

$$\begin{aligned} A_{12} &= U_{11}^T U_{12} \\ A_{22} &= U_{12}^T U_{12} + U_{22}^T U_{22}. \end{aligned}$$

Hence, if U_{11} has already been computed, we can compute U_{12} as the solution to the equation

$$U_{11}^T U_{12} = A_{12}$$

by a call to the Level 3 BLAS routine STRSM; and then we can compute U_{22} from

$$U_{22}^T U_{22} = A_{22} - U_{12}^T U_{12}.$$

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

Machine:	CRAY Y-MP	CRAY Y-MP
Number of processors:	1	8
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

```
   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.

Clock Speeds of Computers in Timing Results

	Clock Speed	
CONVEX C-4640	135 MHz	7.41 ns
CRAY C90	240 MHz	4.167 ns
DEC 3000-500X Alpha	200 MHz	5.0 ns
IBM POWER2 model 590	66 MHz	15.15 ns
IBM RISC Sys/6000-550	42 MHz	23.81 ns
SGI POWER CHALLENGE	75 MHz	13.33 ns

Speed in megaflops of SGETRF/DGETRF for square matrices of order n

	No. of processors	Block size	Values of n	
			100	1000
CONVEX C-4640	1	64	274	711
CONVEX C-4640	4	64	379	2588
CRAY C90	1	128	375	863
CRAY C90	16	128	386	7412
DEC 3000-500X Alpha	1	32	53	91
IBM POWER2 model 590	1	32	110	168
IBM RISC Sys/6000-550	1	32	33	56
SGI POWER CHALLENGE	1	64	81	201
SGI POWER CHALLENGE	4	64	79	353

Speed in megaflops of SPOTRF/DPOTRF for matrices of order n with UPLO = 'U'

	No. of processors	Block size	Values of n	
			100	1000
CONVEX C-4640	1	64	120	546
CONVEX C-4640	4	64	150	1521
CRAY C90	1	128	324	859
CRAY C90	16	128	453	9902
DEC 3000-500X Alpha	1	32	37	83
IBM POWER2 model 590	1	32	102	247
IBM RISC Sys/6000-550	1	32	40	72
SGI POWER CHALLENGE	1	64	74	199
SGI POWER CHALLENGE	4	64	69	424

Speed in megaflops of DSYTRF for matrices of order n with UPLO = 'U' on an IBM POWER2 model 590

Block size	Values of n	
	100	1000
1	62	86
64	68	165

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/c++/lapack++`,
  - (anonymous) ftp `netlib2.cs.utk.edu`:
    - cd lapack; get index
    - cd lapack/lawns; get index
    - cd linalg; get index
    - cd c++; get index
  - email `netlib@ornl.gov` with the message:

    `send index from lapack`
- Comments and questions can be addressed at

  `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 Situation:

Parallel scientific applications are typically

- written from scratch, or manually adapted from sequential programs
- using simple SPMD model
- in Fortran or C
- with explicit message passing primitives

Which means

- similar components are coded over and over again,
- codes are difficult to develop and maintain
- debugging particularly unpleasant...
- difficult to reuse components
- not really designed for long-term software solution

What should math software look like?

- *Many* **more possibilities than shared memory**
- **Possible approaches**
  - **Minimum change to status quo**
    - **Message passing and object oriented**
  - **Reusable templates – requires sophisticated user**
  - **Problem solving environments**
    - **Integrated systems à la Matlab, Mathematica use with heterogeneous platform**
  - **Computational server, like netlib/xnetlib f2c**
- **Some users want high perf., access to details Others sacrifice speed to hide details**
- **Not enough penetration of libraries into hardest applications on vector machines**
- **Need to look at applications and rethink mathematical software**

The NetSolve Client

Virtual Software Library
Multiplicity of interfaces → Ease of use

Programming interfaces

- C interface
- FORTRAN interface

Interactive interfaces

- Non-graphic interfaces
  - MATLAB interface
  - UNIX-Shell interface
- Graphic interfaces
  - TK/TCL interface
  - HotJava Browser interface (WWW)

The NetSolve Agent

Intelligent Agent → Optimum use of Resource

The agent is permanently aware of :

- The configuration of the NetSolve system
- The characteristics of each NetSolve resource
- The load of each resource
- The functionalities of each resource

Each client requests informs the agent about :

- The type of problem to solve
- The size of the problem

The NetSolve Agent **chooses** the NetSolve resource :

Load Balancing Strategy

The NetSolve System

The system is a set of computational servers

- Heterogeneity supported (XDR)
- Dynamic addition of resource
- Fault Tolerance

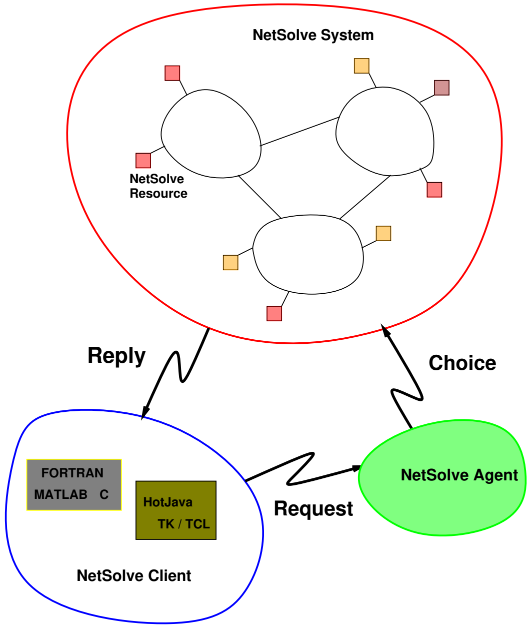
Each server uses the installed scientific packages

Example of packages :

- BLAS
- LAPACK
- ScaLAPACK
- Ellpack

Connection to other systems

- NEOS



NetSolve : The Solution Engine

LAPACK / ScaLapack
A Scalable Library for
Distributed Memory Machines

- LAPACK success (from HP-48G to CRAY C-90)
  - Targets workstations, vector machines, shared memory
  - 449,015 requests since then (as of 10/95)
  - Dense and banded matrices
  - Linear systems, least squares, eigenproblems
  - Manual available from SIAM, html, Japanese translation
  - Second release and manual in 9/94
  - Used by Cray, IBM, Convex, NAG, IMSL, ...
- Provide useful scientific program libraries for computational scientists working on new parallel architectures
  - Move to new generation high performance machines - SP-2, T3D, Paragon, Clusters of WS, ...
  - Add functionality (generalized eigenproblem, sparse matrices, ...)

- New challenges
  - No standard software
    Fortran 90, HP Fortran, PVM, MPIbut ...
  - Many flavors of message passing
    Need standard - BLACS
  - Highly varying ratio of

$$r = \frac{\text{computation speed}}{\text{communication speed}}$$

  - Many ways to layout data
  - Fastest parallel algorithm sometimes less stable numerically

Needs

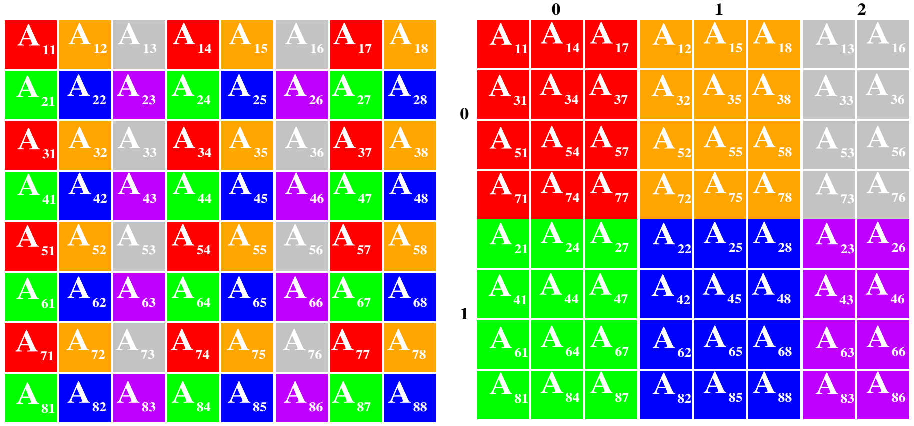
- Standards
  - Portable
  - Vendor-supported
  - Scalable
- High efficiency
- Protect programming investment, if possible
- Avoid low-level details
- Scalability

ScaLAPACK – OVERVIEW

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 (PVM, MPI, ...)
- **Simple calling interface**
  - Stay as close 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 (but not all) data distribution schemes,
- Need redistribution routines to go from distribution to the other,
- Not intuitive.

Communication Library for Linear Algebra

Basic Linear Algebra Communication Subroutines

Purpose of the BLACS:

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations, improve program readability and the self-documenting quality of 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 giving up efficiency since optimized versions 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

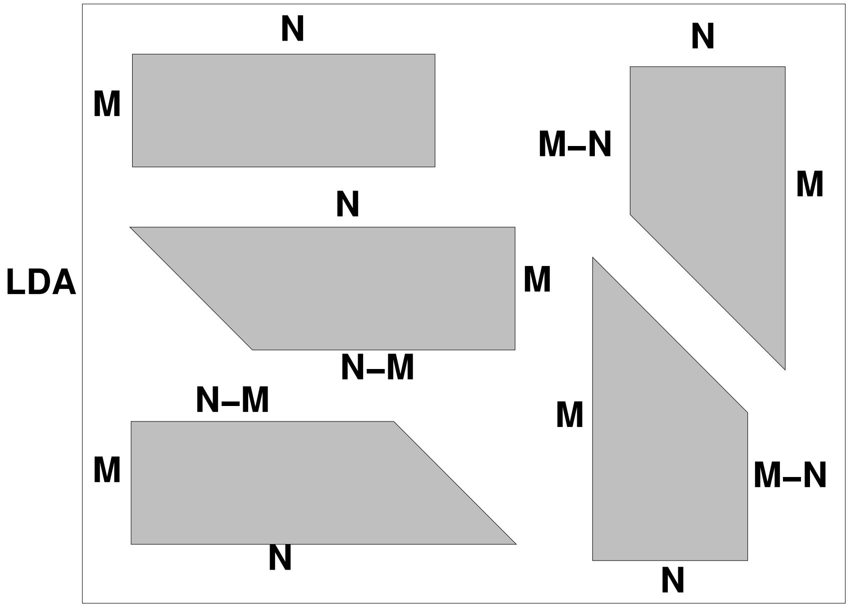
- An operation which involves more than one sender and one receiver is called a **"scoped operation"**. Using a 2D-grid, there are 3 natural scopes:

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

- **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, M, N, A, LDA, RDEST, CDEST )`

   `□xxRV2D( ICONTXT, 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, M, N, A, LDA)

□xxBR2D(ICONTXT, SCOPE, TOP, 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 – 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.

→ the BLACS context is compatible with the MPI communicator

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.

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 dependency are confined to the PBLAS (BLAS and BLACS).

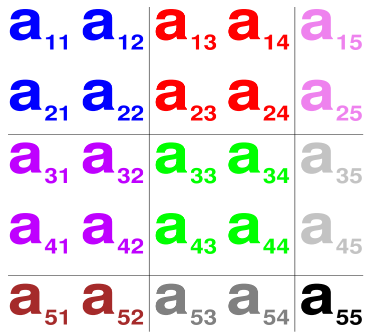
PBLAS – STORAGE CONVENTIONS

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

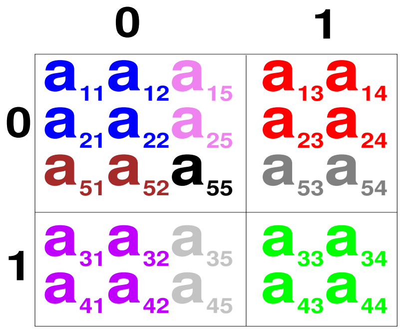
  2-dimensional block-cyclic scheme
  ⇒ **load balanced computations, scalability**,

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

  ⇒ **re-use of the BLAS** (leading dimension `LLD_`).



5 x 5 matrix partitioned in 2 x 2 blocks



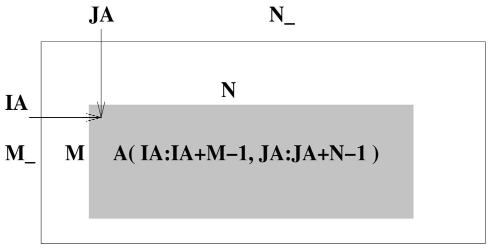
2 x 2 process grid point of view

Descriptor `DESC_`: 8-Integer array describing the matrix layout, containing `M_`, `N_`, `MB_`, `NB_`, `RSRC_`, `CSRC_`, `CTXT_` 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_=5`, `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 )
```

↕

```
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(1)` are supported.

BLAS	PBLAS
`INTEGER LDA`	`INTEGER IA, JA, DESCA( 8 )`
`INTEGER INCX`	`INTEGER INCX, IX, JX, DESCX( 8 )`
`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. Ouput Scalar correct in operand scope.

>> **SPMD PROGRAMMING MODEL** <<

PBLAS – REFERENCES

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

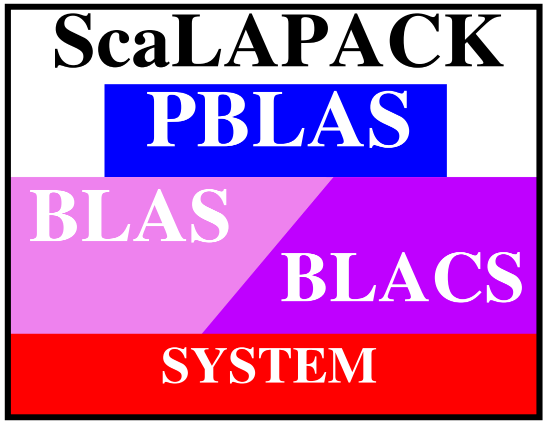
    `cd scalapack; get index`
  - email `netlib@www.netlib.org` with the message:

    `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.

ScaLAPACK IMPLEMENTATION



BLAS (Performance and Portability)

- Blocked data access (level 3 BLAS) yields good local performance

BLACS (Performance and Portability)

- Dedicated machine dependent communication kernel,
- 2D array-based and ID-less communication,
- Code re-entrance, multiple overlapping process grids,
- Safe co-existence with other parallel libraries.

PERFORMANCE MODELING

- **Validation of the data distribution scheme,**
- **Estimation of the efficiency of the library,**
- **Estimation of the library overhead,**
- **Debugging tool,**
- **Precise measure of the impact of block size, grid shape, machine characteristics on the execution time.**

Parallel Level 2 and 3 BLAS: PBLAS

Goal

- Port sequential library for shared memory machine that use the BLAS to distributed memory machines with little effort.
- Reuse the existing software by hiding the message passing in a set of BLAS routines.
- Parallel implementation of the BLAS that understands how the matrix is layed out and when called can perform not only the operation but the required data transfer.

LAPACK $\rightarrow$ ScaLAPACK

BLAS $\rightarrow$ PBLAS (BLAS, BLACS)

- Quality (maintenance)
- Portability (F77 - C)
- Efficiency - Reuseability (BLAS, BLACS)

- Hide Parallelism in (P)BLAS

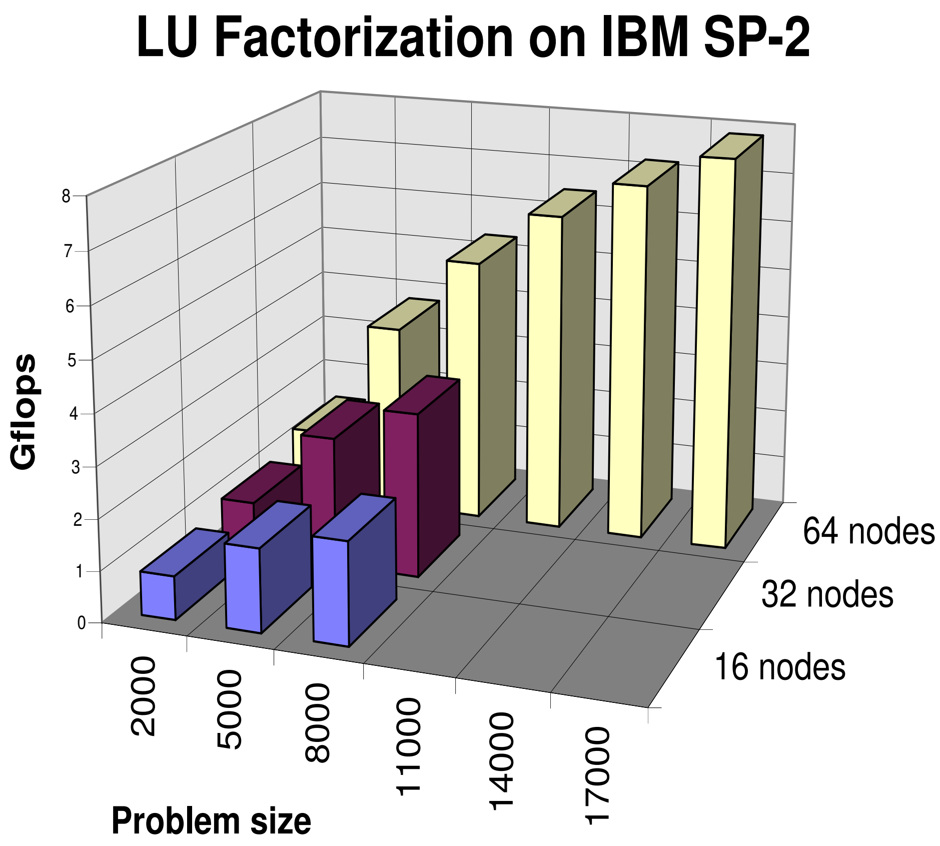
SEQUENTIAL LU FACTORIZATION CODE

```
   DO 20 J = 1, MIN( M, N ), NB
      JB = MIN( MIN( M, N )-J+1, NB )

      Factor diagonal and subdiagonal blocks and test for exact
      singularity.

      CALL DGETF2( M-J+1, JB, A( J, J ), LDA, IPIV( J ),
  $               IINFO )

      Adjust INFO and the pivot indices.

      IF( INFO.EQ.0 .AND. IINFO.GT.0 ) INFO = IINFO + J - 1
      DO 10 I = J, MIN( M, J+JB-1 )
         IPIV( I ) = J - 1 + IPIV( I )
10    CONTINUE

      Apply interchanges to columns 1:J-1.

      CALL DLASWP( J-1, A, LDA, J, J+JB-1, IPIV, 1 )

      IF( J+JB.LE.N ) THEN

         Apply interchanges to columns J+JB:N.

         CALL DLASWP( N-J-JB+1, A( 1, J+JB ), LDA, J, J+JB-1,
  $                   IPIV, 1 )

         Compute block row of U.

         CALL DTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
  $                  JB, N-J-JB+1, ONE, A( J, J ), LDA,
  $                  A( J, J+JB ), LDA )
         IF( J+JB.LE.M ) THEN

            Update trailing submatrix.

            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 )
         END IF
      END IF
20 CONTINUE
```

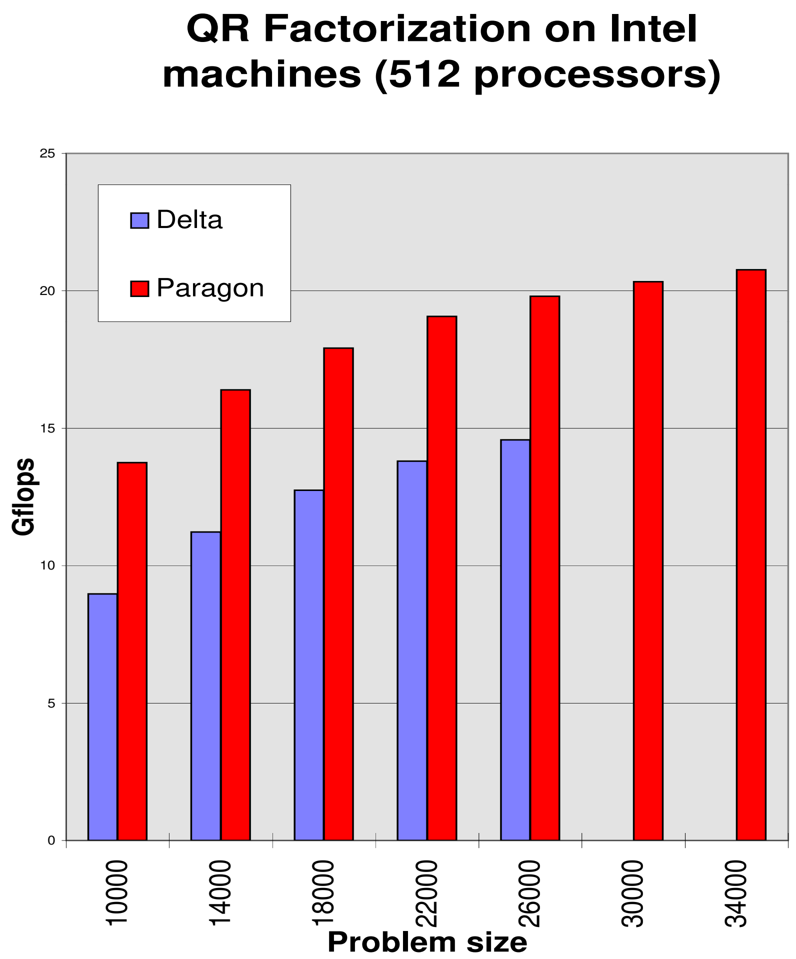
PARALLEL LU FACTORIZATION CODE

```
   DO 10 J = JA, JA+MIN(M,N)-1, DESCA( 4 )
      JB = MIN( MIN(M,N)-J+JA, DESCA( 4 ) )
      I = IA + J - JA

      Factor diagonal and subdiagonal blocks and test for exact
      singularity.

      CALL PDGETF2( M-J+JA, JB, A, I, J, DESCA, IPIV, IINFO )

      Adjust INFO and the pivot indices.

      IF( INFO.EQ.0 .AND. IINFO.GT.0 )
  $      INFO = IINFO + J - JA

      Apply interchanges to columns JA:J-JA.

      CALL PDLASWP( 'Forward', 'Rows', J-JA, A, IA, JA, DESCA,
  $                 J, J+JB-1, IPIV )

      IF( J-JA+JB+1.LE.N ) THEN

         Apply interchanges to columns J+JB:JA+N-1.

         CALL PDLASWP( 'Forward', 'Rows', N-J-JB+JA, A, IA,
  $                    J+JB, DESCA, J, J+JB-1, IPIV )

         Compute block row of U.

         CALL PDTRSM( 'Left', 'Lower', 'No transpose', 'Unit',
  $                   JB, N-J-JB+JA, ONE, A, I, J, DESCA, A, I,
  $                   J+JB, DESCA )
         IF( J-JA+JB+1.LE.M ) THEN

            Update trailing submatrix.

            CALL PDGEMM( 'No transpose', 'No transpose',
  $                      M-J-JB+JA, N-J-JB+JA, JB, -ONE, A,
  $                      I+JB, J, DESCA, A, I, J+JB, DESCA,
  $                      ONE, A, I+JB, J+JB, DESCA )
         END IF
      END IF
10 CONTINUE
```

PERFORMANCE RESULTS

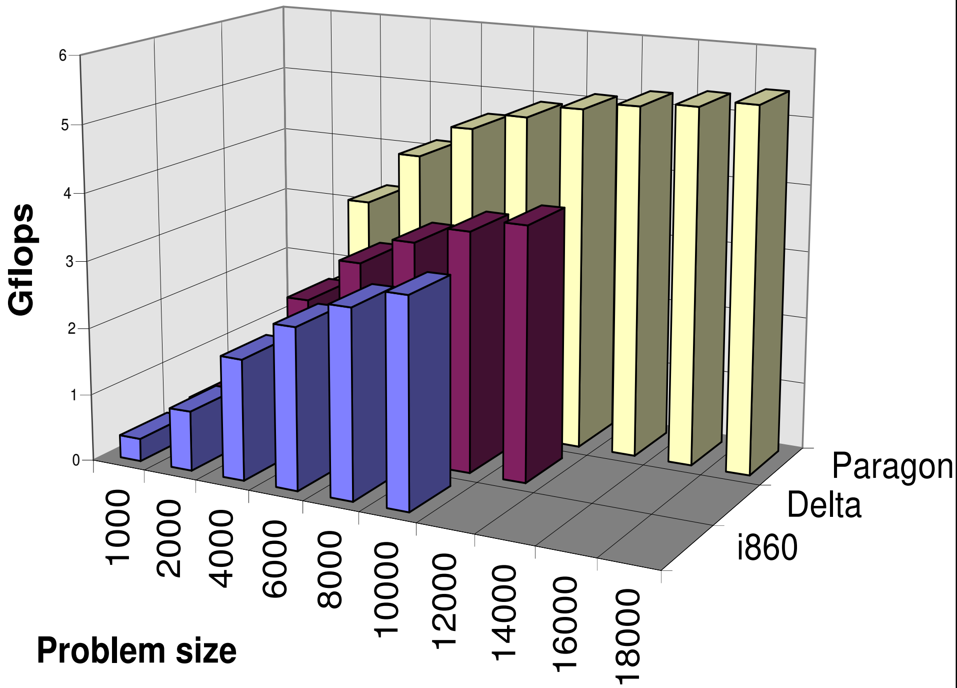


PERFORMANCE RESULTS

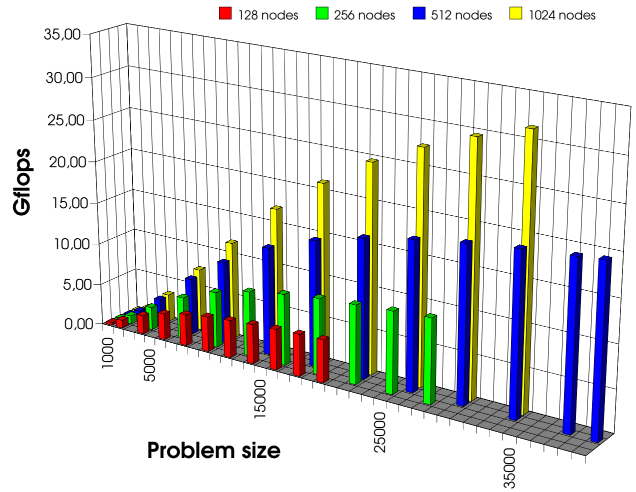


PERFORMANCE RESULTS

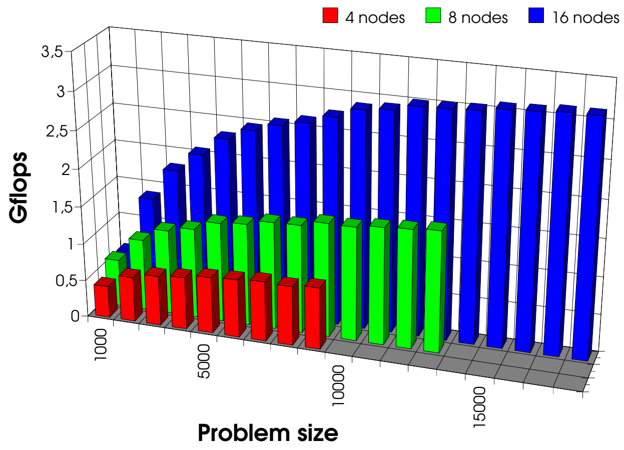
QR Factorization on 128 nodes Intel Machines

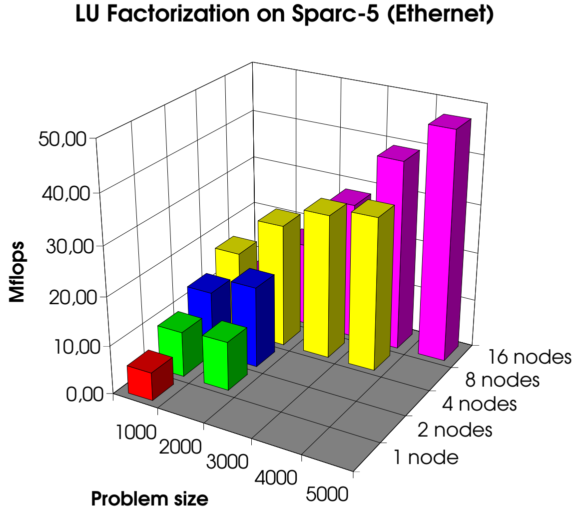


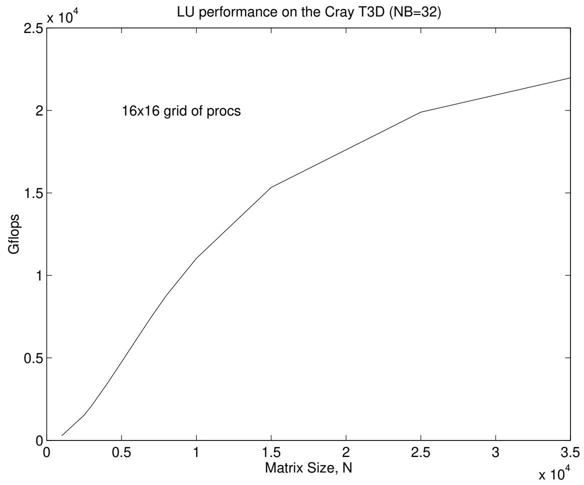
LU Factorization on Intel Paragon



LU Factorization on IBM SP-2







Symmetric Eigenproblem $Ax = \lambda x$, $A = A^T$

- **Two phases:**

  Phase 1: Reduce A to tridiagonal T:

$$A = UTU^T, \quad T = \begin{bmatrix} a_1 & b_1 & & 0 \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ 0 & & b_{n-1} & a_n \end{bmatrix}$$

  Phase 2: Find e-vals, vectors of T: $T = Q\Lambda Q^T$

  So $A = (UQ)\Lambda(UQ)^T \equiv V\Lambda V^T$

- **Phase 2 is bottleneck (3x Phase 1)**
- **LAPACK: Cuppen's divide-and-conquer method**
- **ScaLAPACK:**
  - **Bisection + inverse iteration with limited reorthogonalization: fast but may fail to compute orthogonal eigenvectors if eigenvalues clustered**
  - **QR iteration: slower but reliable**

Outline of Cuppen's Divide and Conquer

1. **Goal: compute** $T = Q\Lambda Q^T$, $\Lambda = \mathbf{diag}(\lambda_i)$, $QQ^T = I$
2.

$$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix} = \left[\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right] + vv^T$$

3.

Solve $T_1 = Q_1 \Lambda_1 Q_1^T$ **and** $T_2 = Q_2 \Lambda_2 Q_2^T$ **recursively**

4.

$$T = \left[\begin{array}{c|c} Q_1\Lambda_1 Q_1^T & \\ \hline & Q_2\Lambda_2 Q_2^T \end{array}\right] + vv^T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot (D + zz^T) \cdot \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix}$$

where

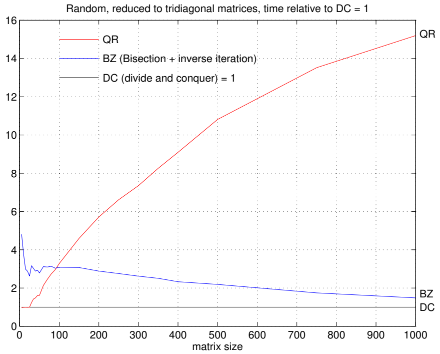
$$D = \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} \quad \textbf{and} \quad z = \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} v$$

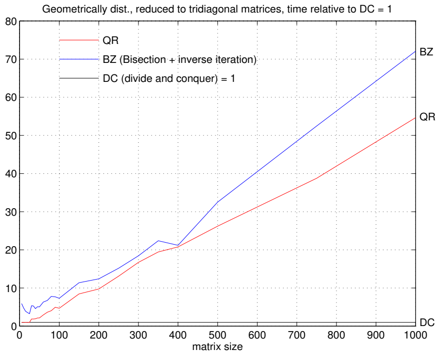
5. **Eigenvalues** λ **of** $D + zz^T$ **are roots of "secular equation"**

$$1 + \sum_i \frac{z_i^2}{\lambda - d_{ii}} = 0$$

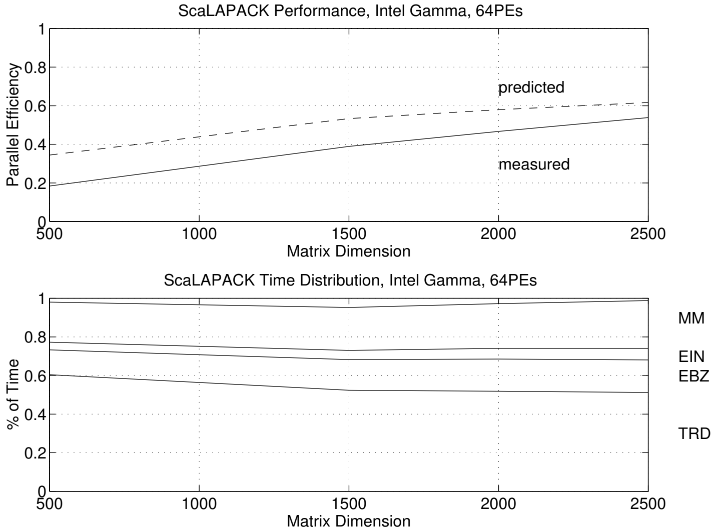
6. **Eigenvectors** v_j **of** $D + zz^T$ **are** $(D - \lambda_j I)^{-1} z$
7. **Eigenvectors of** T **are** $\begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot V$

Speedups with Divide-and-Conquer





ScaLAPACK Performance on 64 Processor Intel Gamma
Expert Driver - Bisection + Inverse iteration



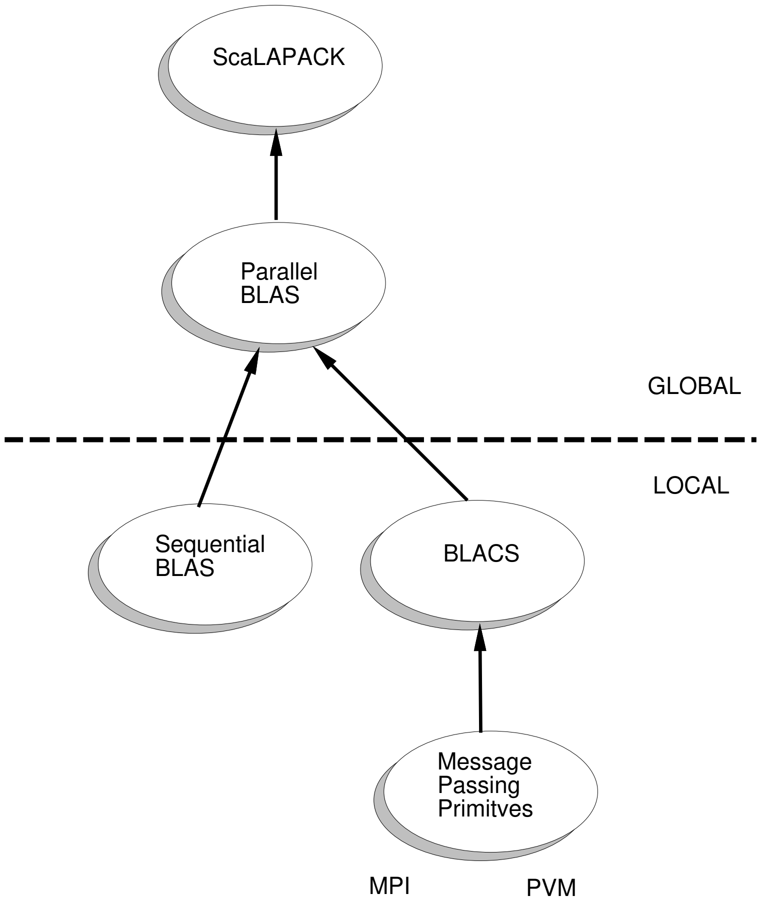
ScaLAPACK Sparse matrix activities

- **ARPACK - Arnoldi-Pack - D. Sorensen**
  - Algorithms for symmetric and nonsymmetric linear systems, eigenproblems, generalized eigenproblems
  - User's matrix-vector product, rest serial
  - Based on clever identity for compressing partial Arnoldi factorization
- **CAPSS - Cartesian Parallel Sparse Solver - P. Raghavan and M. Heath**
  - Parallel sparse Cholesky
  - For finite difference/finite element problems
  - Usese coordinates from underlying mesh
  - Cartesian nested dissection to layout data
- **Sparse LU with Pivoting**
  - Exploits data locality using supernodal approach
  - Up to 120 Mflops on IBM RS6000/590, depending on sparsity pattern
  - Fastest code available on many test cases, along with UMFPACK
  - To be released soon

WHAT IS AVAILABLE TO YOU ?

- **Tools**
  - BLACS (CMMD, MPL, NX, PVM)
  - PBLAS
  - REDISTRIBUTION
  - PUMMA
- **Factorizations**
  - LU + solve
  - Cholesky + solve
  - QR, QL, LQ, RQ, QRP
- **Matrix inversions**
- Q, QC, Q^TC, CQ, CQ^T
- **Reductions**
  - to Hessenberg form
  - to symmetric tridiagonal form
  - to bidiagonal form
- **Symmetric eigensolver**

Software Framework



PVM calls within the BLACS

PVM (Parallel Virtual Machine) is a software package that permits a heterogeneous collection of serial, parallel, and vector computers hooked together by a network to appear as one large computer.

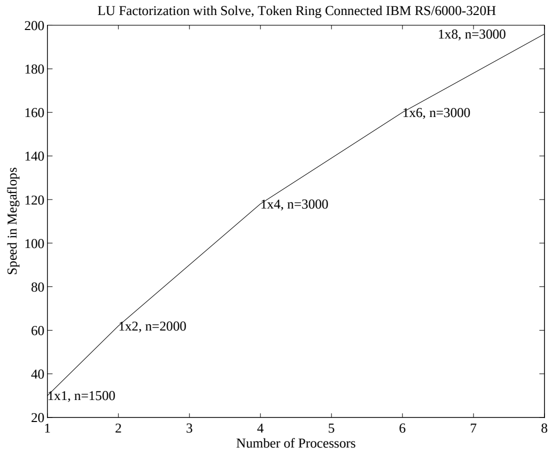
BLACS (Basic Linear Algebra Communication Subroutines) is a software package to perform communication at the same level as the BLAS are for computation.

- **BLACS have been written in terms of calls to PVM.**
- **Allows network of workstations to be used with minimal change.**

MPI – Message Passing Interface

Standard for Message Passing

- **BLACS $\rightarrow$ MPI underway**



LU Results on IBM Cluster

Proc	Time	Speedup	Proc	Mflop/s	Speedup
1	1030		1 (n=1500)	33	
2	403	2.6	2 (n=2000)	64	1.9
4	155	6.6	4 (n=3000)	116	3.5
6	115	8.9	6 (n=3000)	156	4.7
8	93	11.0	8 (n=3000)	194	5.9

SCALAPACK – ONGOING WORK

- **HPF**
  - HPF supports the 2D-block-cyclic distribution used by ScaLAPACK
  - HPF like calling sequence (Global indexing scheme)
- **Portability: MPI implementation of the BLACS**
- **More flexibility added to the PBLAS**
- **More testing and timing programs**
- **Condition estimators**
- **Iterative refinement of linear system solutions**
- **SVD**
- **Linear Least Square solvers**
- **Banded systems**
- **Non symmetric eigensolvers**
- ...