Survey of Parallel Numerical Libraries

James Demmel
UC Berkeley
demmel@cs.berkeley.edu
http://www.cs.berkeley.edu/~demmel

Participants

Krste Asanovic (UC Berkeley)
Richard Barrett (U. Tenn)
Jeff Bilmes (UC Berkeley)
Susan Blackford (ORNL)
Tony Chan (UCLA)
Jaeyoung Choi (LBNL)
Ed D'Azevedo (ORNL)
Inderjit Dhillon (UC Berkeley)
Jack Dongarra (U Tenn, ORNL)
Jeremy Du Croz (NAG)
Stan Eisenstat (Yale)
John Gilbert (Xerox PARC)
Sven Hammarling (NAG)
Greg Henry (Intel)
Steve Huss-Lederman (SRC)
W. Kahan (UC Berkeley)
Rencang Li (UC Berkeley)
Joseph Liu (York)
Antoine Petitet (U Tenn)
Roldan Pozo (U Tenn)
Huan Ren (UC Berkeley)
Charles Romine (ORNL)
Ivan Slapničar (U Split)
Ken Stanley (UC Berkeley)
Bernard Tourancheau (U Tenn)
Robert van de Geijn (U Texas)
Paul Van Dooren (U Illinois)
David Walker (ORNL)
Kathy Yelick (UC Berkeley)
Zhaojun Bai (U Kentucky)
Michael Berry (U Tenn)
Chris Bischof (ANL)
Soumen Chakrabarti (UC Berkeley)
Chee-Whye Chin (UC Berkeley)
Andy Cleary (LLNL)
Jim Demmel (UC Berkeley)
June Donato (ORNL)
Zlatko Drmač (U Hagen)
Victor Eijkhout (UCLA)
Vince Fernando (NAG)
Ming Gu (UC Berkeley, LBL)
Mike Heath (U Illinois)
Dominic Lam (UC Berkeley)
Bo Kågström (U Umeå)
Youngbae Kim (U Tenn)
Xiaoye Li (UC Berkeley)
Beresford Parlett (UC Berkeley)
Peter Poromaa (U Umeå)
Padma Raghavan (U Illinois)
Howard Robinson (UC Berkeley)
Jeff Rutter (UC Berkeley)
Dan Sorensen (Rice U)
Xiaobai Sun (ANL)
Anna Tsao (SRC)
Henk van der Vorst (Utrecht U)
Krešimir Veselić (U Hagen)
Clint Whaley (U Tenn)

With the cooperation of
Cray, IBM, Convex, DEC, Fujitsu, NEC, NAG, IMSL

Supported by ARPA, NSF, DOE

Outline

- Kinds of problems, algorithms, parallelism
- Exploiting mathematical structure
- Memory Hierarchies: BLAS
- Dense and Band Problems: (Sca)LAPACK
- Computing on Heterogeneous Networks
- Sparse direct solvers for $Ax = b$
- Krylov Subspace Methods
- Graph Partitioning for $y = Ax$
- Where to go for further information

Kinds of Problems, Algorithms, Parallelism

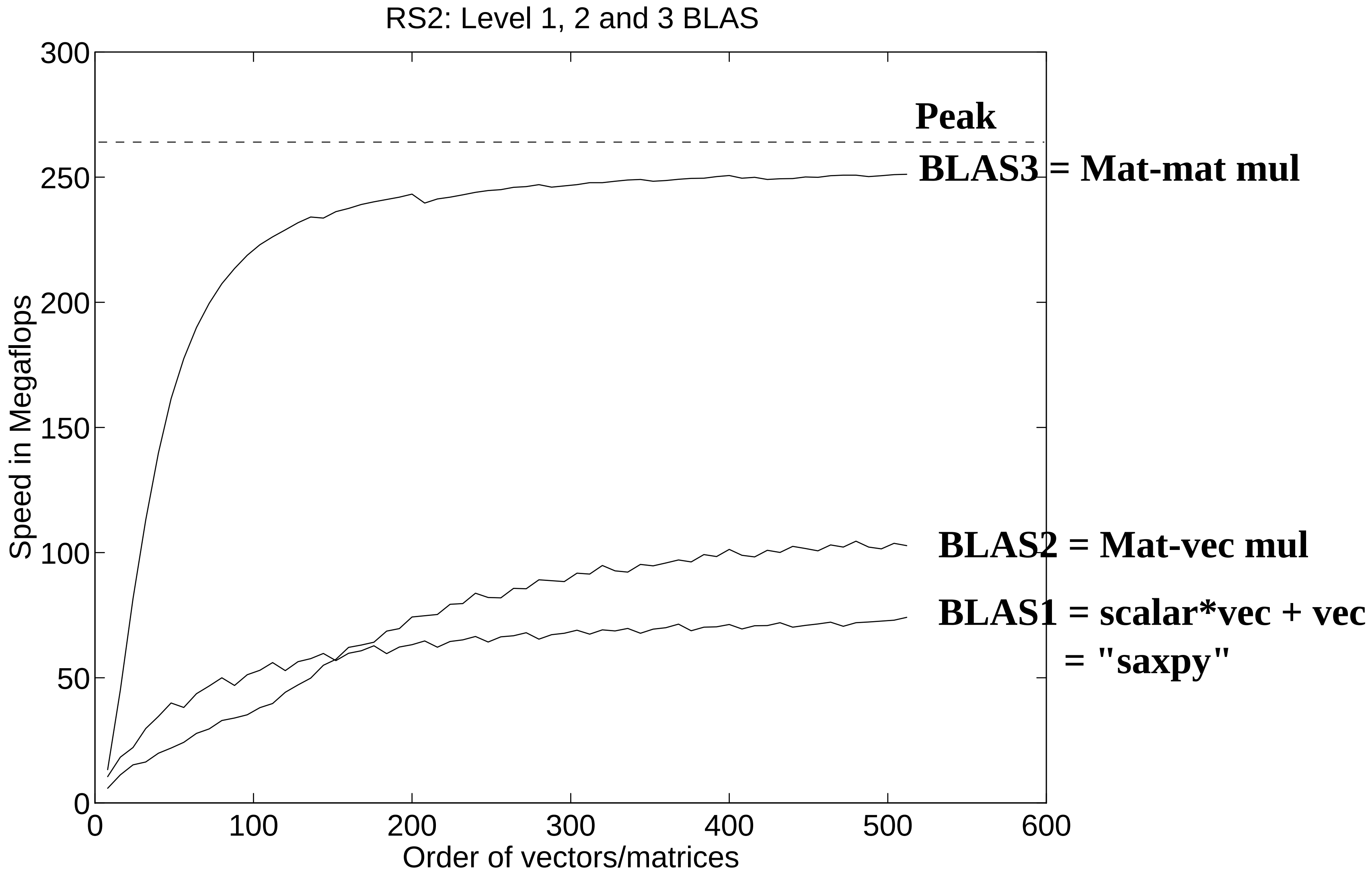
- **Problems**
  - **Linear Systems:** $Ax = b$
  - **Least Squares:** $min_x \|Ax - b\|_2$
  - **Eigenvalue problems:** $Ax = \lambda x$
- **Algorithms**
  - **Direct**
    * **"right" answer after fixed number of steps**
    * **manipulate matrix entries directly**
  - **Iterative**
    * **error decreases after each step**
    * **stop iterating after error small enough**
    * **often access matrix A only via subroutines for $y = Ax$ or $y = A^T x$**
- **Parallelism**
  - **Message passing**
  - **Shared memory with threads**

The importance of exploiting mathematical structure

- **How do I solve the n-by-n system $Ax = b$?**
- **No structure: $\frac{2}{3}n^3$ using Gaussian elimination**
- **Symmetric Positive Definite: $\frac{1}{3}n^3$ using Cholesky**
- **Bandwidth $\sqrt{n}$: $O(n^2)$ using band Cholesky**
- **$O(n)$ nonzeros, $O(n)$ condition number: $O(n^{3/2})$ using Conjugate Gradients (CG)**
- **5 point Poisson equation on square: $O(n)$ using Multigrid (MG)**

- **For more on recognizing such structures:**
  - `http://www.cs.berkeley.edu/~demmel/ma221`
  - `http://www.netlib.org/templates/Templates.html`

Linear Algebra Kernels on IBM 6000/590

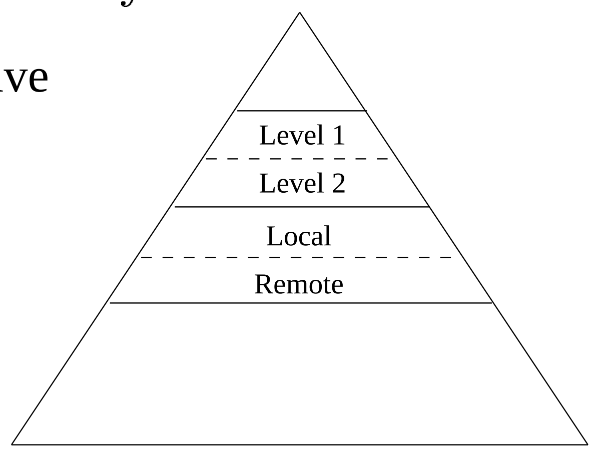


BLAS = Basic Linear Algebra Subroutines

Where do the Flops go?

- **Memory Hierarchy**

Fast, small, expensive



Registers

Cache

Memory

Disk

Slow, large, cheap

- **Most expensive operation is moving data**
- **Can only do useful work on data at top**
- **Transfer between levels slow, so design algorithms to move data infrequently**

BLAS = Basic Linear Algebra Subroutines

- **Three levels**

  BLAS1: $y \leftarrow \alpha x + y$ **(scalar$*$vector+vector)**

  BLAS2: $y \leftarrow Ax + y$ **(matrix$*$vector+vector)**

  BLAS3: $C \leftarrow AB + C$ **(matrix$*$matrix+matrix)**

- **Mathematically equivalent but not cost equivalent**

- **How to tell which BLAS uses memory better**

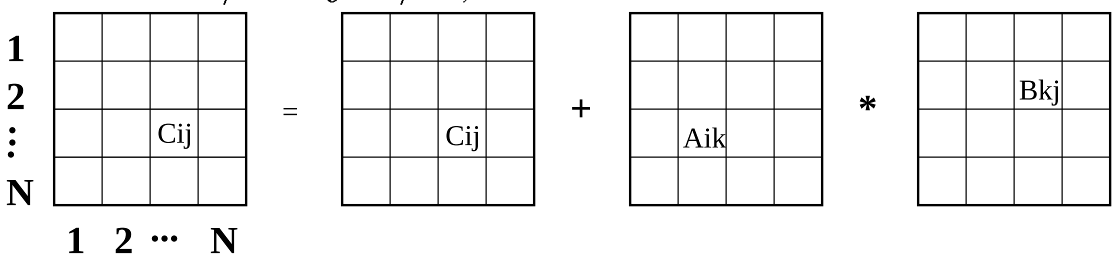
BLAS	# Memory refs	# Flops	$Q = \frac{\#\text{Flops}}{\#\text{Memoryrefs}}$
$y \leftarrow \alpha x + y$	$3n$	$2n$	$2/3$
$y \leftarrow Ax + y$	n^2	$2n^2$	2
$C \leftarrow AB + C$	$4n^2$	$2n^3$	$n/2$

- **Large Q $\Rightarrow$ can mask data movement by flops**

- **BLAS3 best: largest Q**

A "blocked" n-by-n Matrix-Matrix Multiply $C = A \cdot B$

- C^{ij} is n/N-by-n/N, as are A^{ik} and B^{kj}



$$\boxed{\text{Cij}} = \boxed{\text{Cij}} + \sum_{k=1}^{n} \boxed{\text{Aik}} * \boxed{\text{Bkj}}$$

for $i = 1$ **to** N
    for $j = 1$ **to** N
        Read C^{ij} ***into fast memory***
        for $k = 1$ **to** N
            Read A^{ik} ***into fast memory***
            Read B^{kj} ***into fast memory***
            $C^{ij} = C^{ij} + A^{ik} \cdot B^{kj}$
        end for
        Write C^{ij} ***back to slow memory***
    end for
end for

- **Optimal choice of** n/N = **"block size" yields** Q = **# flops / # mem refs** = $\sqrt{\text{(fast memory size)}/3}$
  - Q **large (speed high) for large matrices**
  - Q **increases (speed rises) as fast memory grows**
- **"Obvious" mat-mul has** $Q = 2$**; slower!**

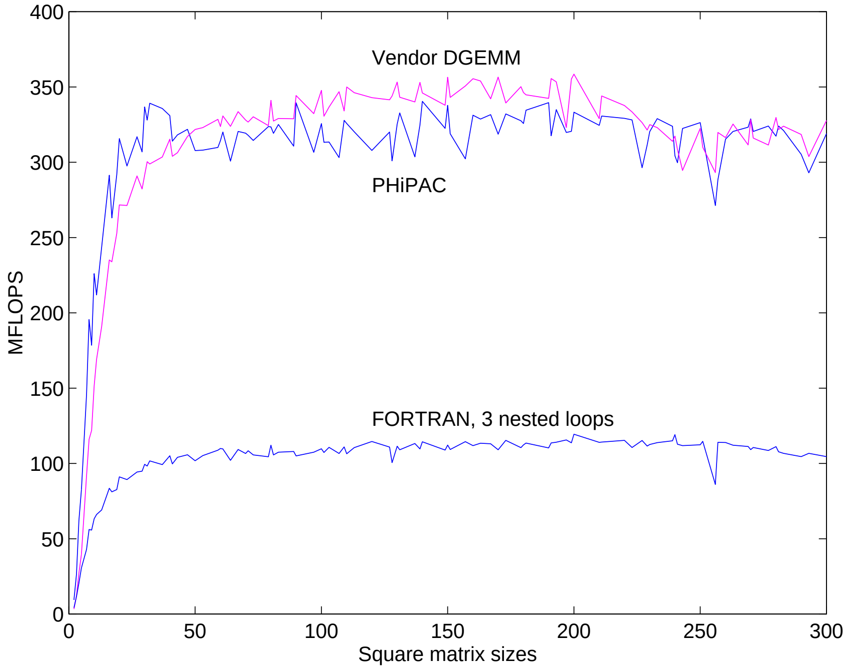
BLAS3 as a Library Building Block

- All higher level libraries use BLAS to maximize single node performance
  - Algorithms in libraries reorganized to call matrix-multiply in inner loops, if possible
- On-going work to extend BLAS `http://www.netlib.org/utk/papers/blast-forum.html`
- Unoptimized Fortran and C BLAS exist, but not good enough for high performance
  - `http://www.netlib.org/blas`
  - Compilers not yet good enough to reliably produce "blocked" matrix multiply from naive version
  - Even less reliable on more complicated algorithms, so we need optimized libraries
  - Ongoing compiler research
- (Good) BLAS are tuned for each architecture, often by manufacturer or software vendor

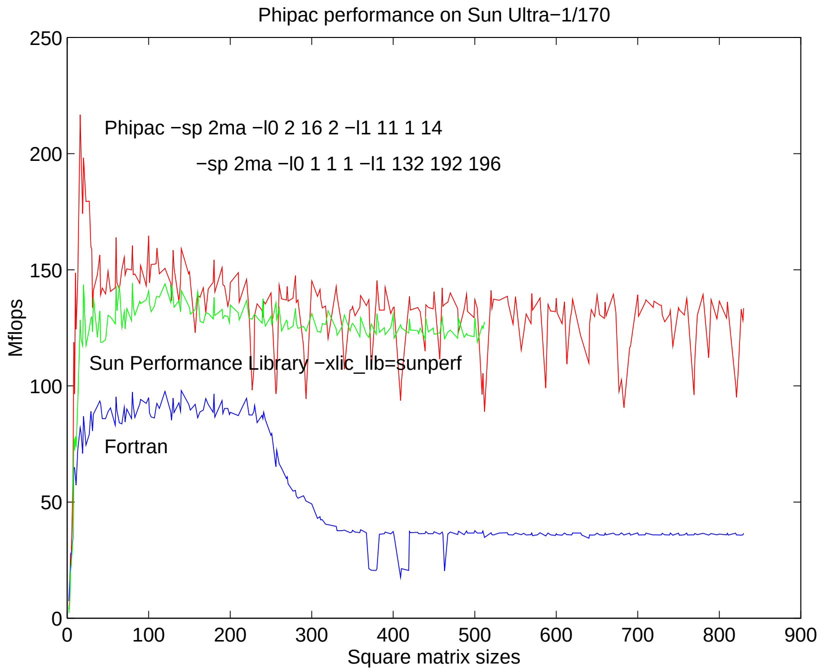
PHIPAC

- Portable High Performance ANSI C
- Automatic BLAS "tuner" for RISC architectures
- Searches through large set of BLAS implementations to find best one
- Often beats manufacturer's BLAS (and cheaper!)
- `http://www.icsi.berkeley.edu/~bilmes/phipac`

PHIPAC on SGI R10000



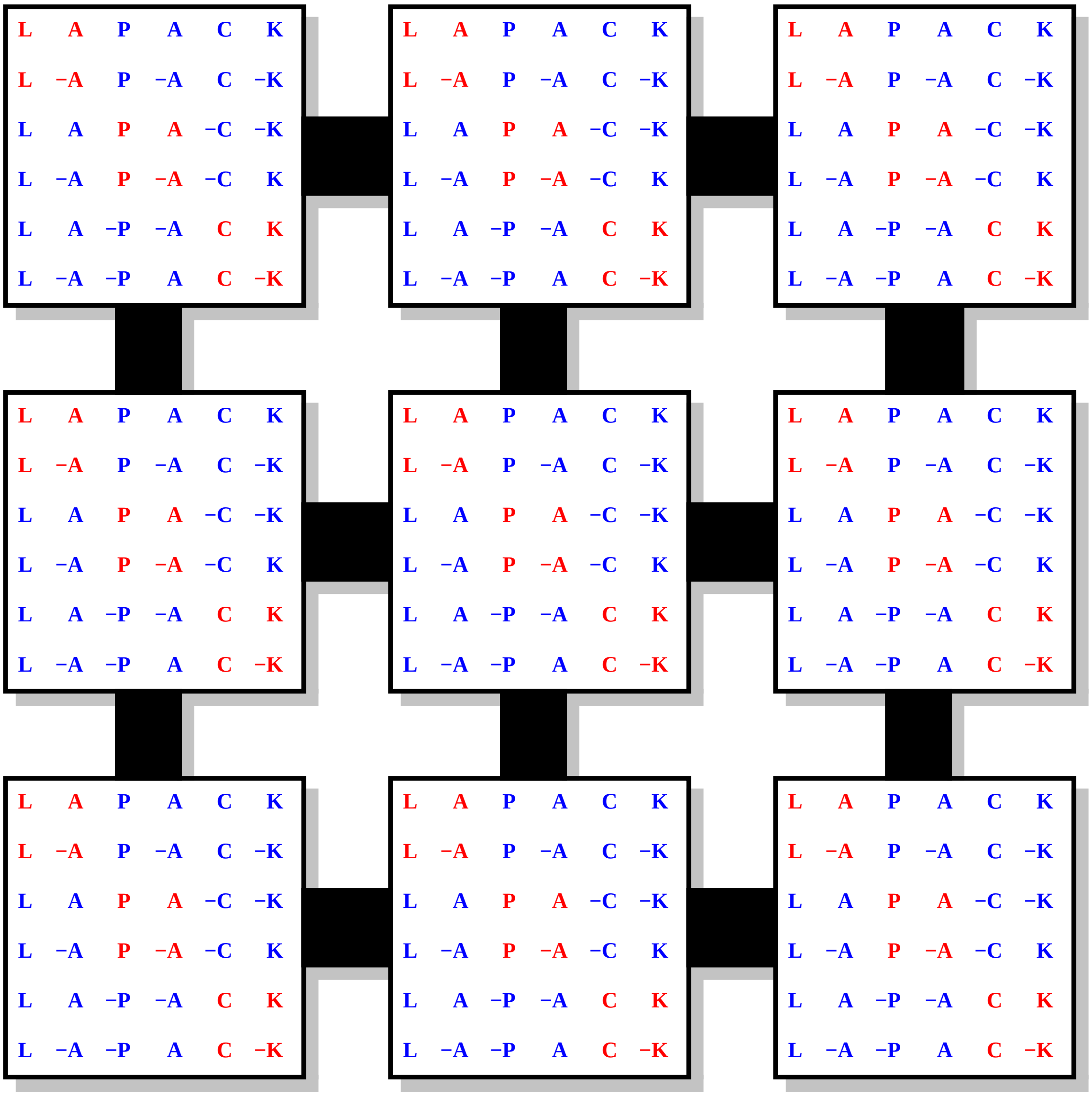
PHIPAC on Ultrasparc



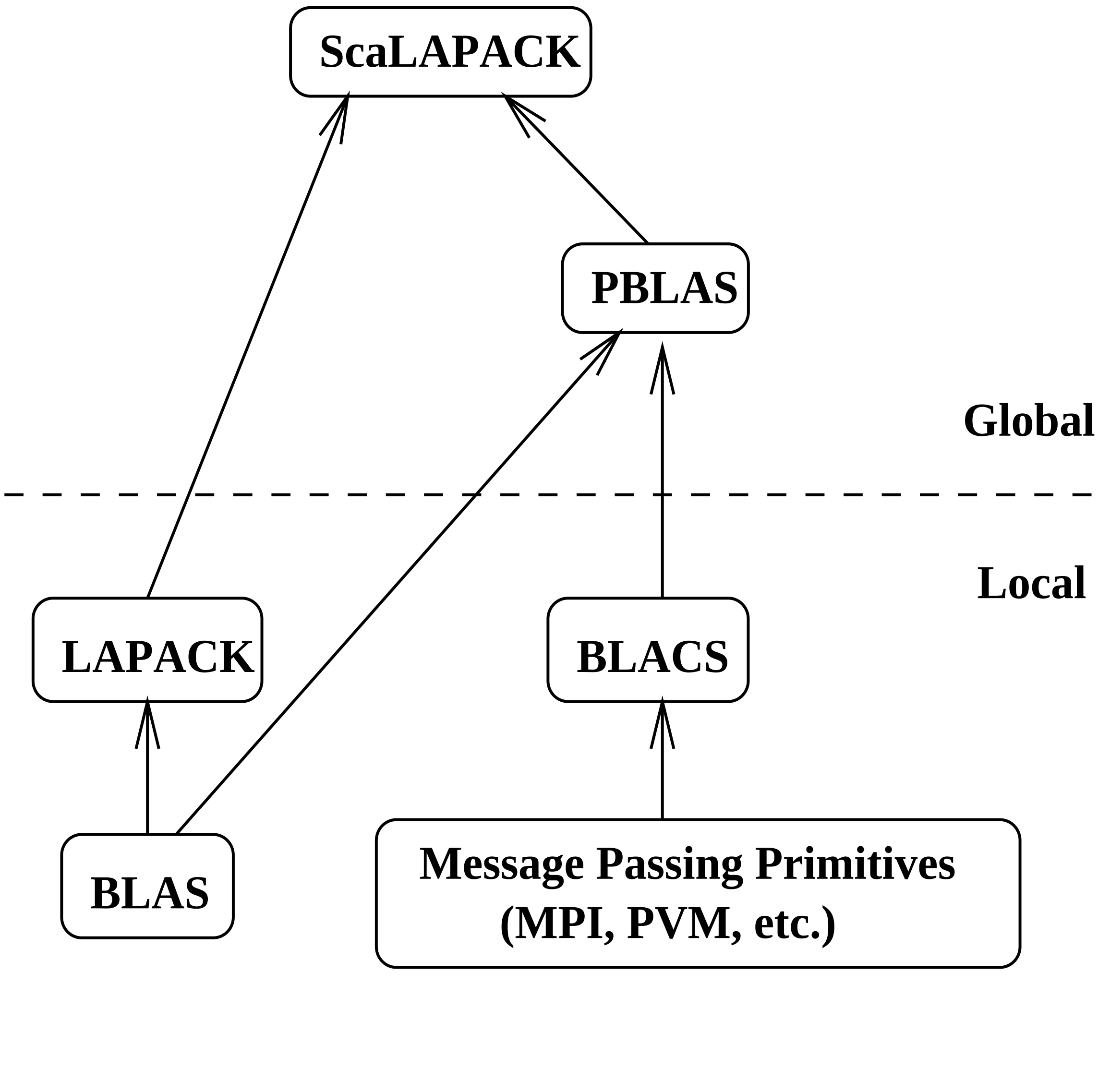
LAPACK and ScaLAPACK

	LAPACK	ScaLAPACK
Machines	Workstations, Vector, SMP	Distributed Memory, DSM
Based on	BLAS	BLAS, BLACS
Functionality	Linear Systems Least Squares Eigenproblems	Linear Systems Least Squares Eigenproblems (less than LAPACK)
Matrix types	Dense, band	Dense, band, out-of-core
Error Bounds	Complete	A few
Languages	F77 or C	F77 and C
Interfaces to	C++, F90	HPF
Manual?	Yes	Yes
Where?	www.netlib.org/ lapack	www.netlib.org/ scalapack

ScaLAPACK

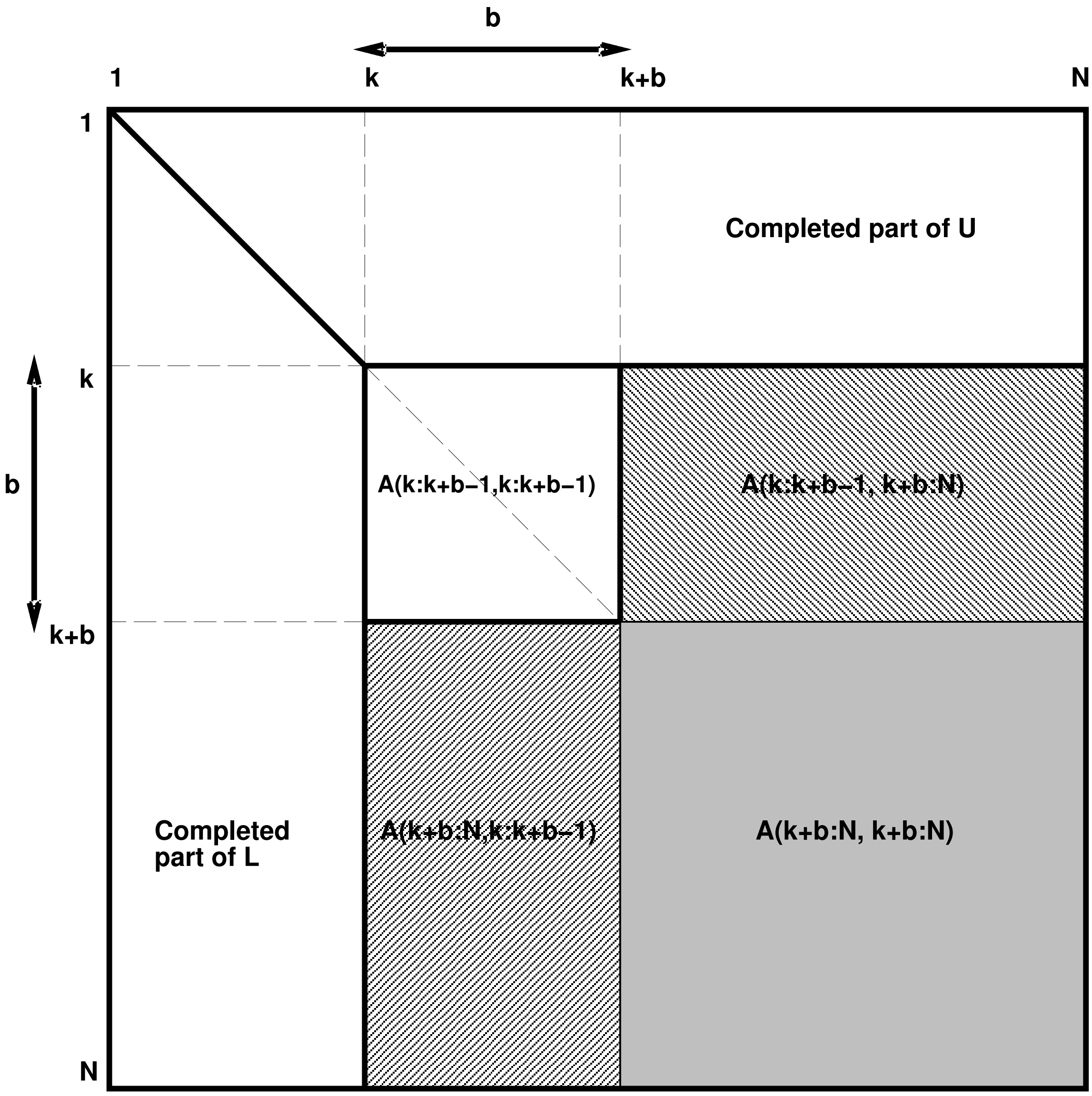


ScaLAPACK SOFTWARE HIERARCHY



Motivation for Data Distributions

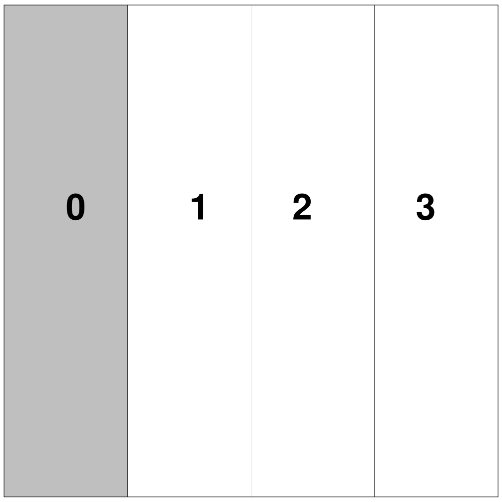
Snapshot of Gaussian Elimination $A = LU$

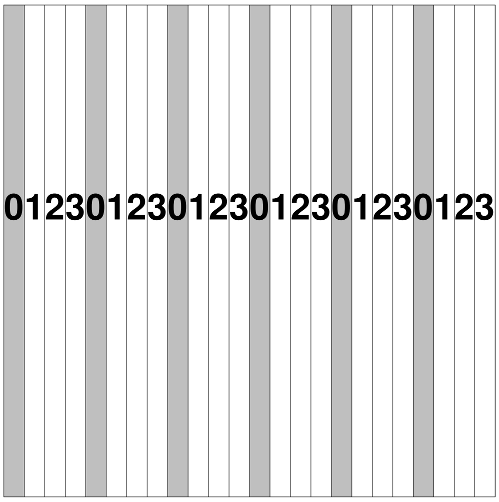


Possible Data Distributions

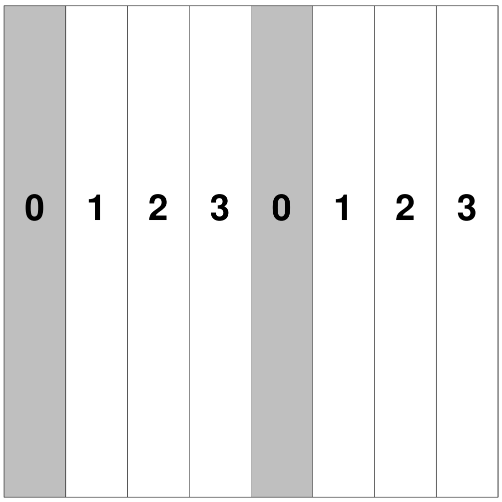
- Desiderata of a good layout:
  - Load Balance
  - Use of BLAS3

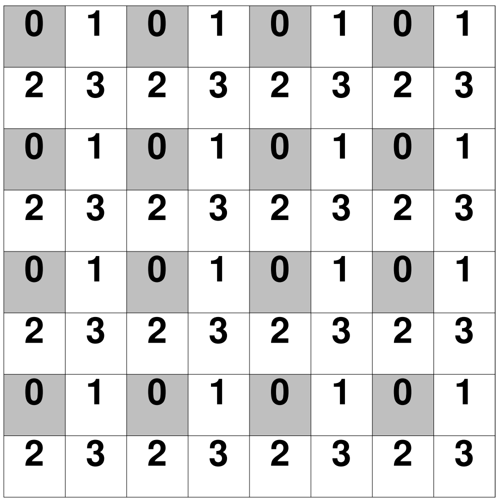
1D block and cyclic column distributions





1D block-cyclic column- and $\boxed{\text{2D block-cyclic}}$ distributions





More on 2D Block Cyclic Distributions

- **1D Block cyclic layout (for each subscript)**
  - **Given P procs and blocksize NB, where is $A(i)$ stored?**
  - **Write $i = b \cdot (P \cdot NB) + p \cdot NB + l$**
  - $b = \lfloor \frac{i}{P \cdot NB} \rfloor$, $p = \lfloor \frac{i \bmod (P \cdot NB)}{B} \rfloor$, $l = i \bmod B$
  - $A(i)$ **stored on proc** p **at** $b \cdot NB + l$
  - **Example for** $P = 2$, $NB = 3$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
p	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1
b	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2
l	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1
$b * NB + l$	1	2	3	1	2	3	4	5	6	4	5	6	7	8	9	7

- **HPF declaration:**

```
      REAL :: X( N )
!HPF$ PROCESSORS PROC( P )
!HPF$ DISTRIBUTE X( CYCLIC( NB ) ) ONTO PROC
```

Array Object Descriptors

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=1 for dense matrices. (Band, tridiagonal, out-of-core available too)
2	CTXT_A	(global)	BLACS context handle, indicating the BLACS process grid over which the global matrix A is distributed. The context itself is global, but the handle (the integer value) may vary.
3	M_A	(global)	Number of rows in the global array A.
4	N_A	(global)	Number of columns in the global array A.
5	MB_A	(global)	Blocking factor used to distribute the rows of the array.
6	NB_A	(global)	Blocking factor used to distribute the columns of the array.
7	RSRC_A	(global)	Process row over which the first row of the array A is distributed.
8	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
9	LLD_A	(local)	Leading dimension of the local array. LLD_A $\geq$ MAX(1,LOC$_r$(M_A)).

Storage Schemes For Band Matrices

Nonsymmetric band matrix A (no pivoting)

Processes

	0			**1**		**2**
*	*	a_{13}	a_{24}	a_{35}	a_{46}	a_{57}
*	a_{12}	a_{23}	a_{34}	a_{45}	a_{56}	a_{67}
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}	a_{77}
a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	a_{76}	*
a_{31}	a_{42}	a_{53}	a_{64}	a_{75}	*	*

Symmetric positive definite band matrix A (UPLO='L')

Processes

	0			**1**		**2**
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}	a_{77}
a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	a_{76}	*
a_{31}	a_{42}	a_{53}	a_{64}	a_{75}	*	*

Symmetric positive definite band matrix A (UPLO='U')

Processes

	0			**1**		**2**
*	*	a_{31}	a_{42}	a_{53}	a_{64}	a_{75}
*	a_{21}	a_{32}	a_{43}	a_{54}	a_{65}	a_{76}
a_{11}	a_{22}	a_{33}	a_{44}	a_{55}	a_{66}	a_{77}

ScaLAPACK Functionality

Problem type	Simple Driver	Expert Driver	Factor	Solve	Invert	Cond. Est.	Iter. Ref.
$Ax = b$							
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 (full rank)	X		X	X			
GQR			X				
GRQ			X				
$Ax = \lambda x$ or $Ax = \lambda Bx$							
Symmetric (2 types)	X	X	X				
General (2 types)			X				
Generalized B SPD		X	X				
SVD			X				

- **Prototype Codes**
  - **HPF interface to ScaLAPACK**
  - **Matrix Sign Function for Eigenproblems**
  - **Out-of-core solvers**

Documentation, Examples, ...

- Documentation
  - ScaLAPACK Users' Guide
  - Installation Guide for ScaLAPACK
  - LAPACK Working Notes

- Test Suites for ScaLAPACK, PBLAS, BLACS

- Example Programs
  - `http://www.netlib.org/scalapack/examples/`

- Pre-built ScaLAPACK libraries on netlib

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

```
      PROGRAM EXAMPLE1
*
*     Example Program solving Ax=b via ScaLAPACK routine PDGESV
*
*     .. Parameters ..
      INTEGER            DLEN_, IA, JA, IB, JB, M, N, MB, NB, RSRC,
     $                   CSRC, MXLLDA, MXLLDB, NRHS, NBRHS, NOUT,
     $                   MXLOCR, MXLOCC, MXRHSC
      PARAMETER          ( DLEN_ = 9, IA = 1, JA = 1, IB = 1, JB = 1,
     $                   M = 9, N = 9, MB = 2, NB = 2, RSRC = 0,
     $                   CSRC = 0, MXLLDA = 5, MXLLDB = 5, NRHS = 1,
     $                   NBRHS = 1, NOUT = 6, MXLOCR = 5, MXLOCC = 4,
     $                   MXRHSC = 1 )
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
*     ..
*     .. Local Scalars ..
      INTEGER            ICTXT, INFO, MYCOL, MYROW, NPCOL, NPROW
      DOUBLE PRECISION   ANORM, BNORM, EPS, RESID, XNORM
*     ..
*     .. Local Arrays ..
      INTEGER            DESCA( DLEN_ ), DESCB( DLEN_ ),
     $                   IPIV( MXLOCR+NB )
      DOUBLE PRECISION   A( MXLLDA, MXLOCC ), A0( MXLLDA, MXLOCC ),
     $                   B( MXLLDB, MXRHSC ), B0( MXLLDB, MXRHSC ),
     $                   WORK( MXLOCR )
*     ..
*     .. External Functions ..
      DOUBLE PRECISION   PDLAMCH, PDLANGE
      EXTERNAL           PDLAMCH, PDLANGE
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLACS_EXIT, BLACS_GRIDEXIT, BLACS_GRIDINFO,
     $                   DESCINIT, MATINIT, PDGEMM, PDGESV, PDLACPY,
     $                   SL_INIT
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          DBLE
*     ..
*     .. Data statements ..
      DATA               NPROW / 2 / , NPCOL / 3 /
*     ..
```

```
*     .. Executable Statements ..
*     INITIALIZE THE PROCESS GRID
*
      CALL SL_INIT( ICTXT, NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*
*     If I'm not in the process grid, go to the end of the program
*
      IF( MYROW.EQ.-1 )
     $   GO TO 10
*
*     DISTRIBUTE THE MATRIX ON THE PROCESS GRID
*     Initialize the array descriptors for the matrices A and B
*
      CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
     $               INFO )
      CALL DESCINIT( DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT,
     $               MXLLDB, INFO )
*
*     Generate matrices A and B and distribute to the process grid
*
      CALL MATINIT( A, DESCA, B, DESCB )
*
*     Make a copy of A and B for checking purposes
*
      CALL PDLACPY( 'All', N, N, A, 1, 1, DESCA, A0, 1, 1, DESCA )
      CALL PDLACPY( 'All', N, NRHS, B, 1, 1, DESCB, B0, 1, 1, DESCB )
*
*     CALL THE SCALAPACK ROUTINE
*     Solve the linear system A * X = B
*
      CALL PDGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB,
     $             INFO )
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         WRITE( NOUT, FMT = 9999 )
         WRITE( NOUT, FMT = 9998 )M, N, NB
         WRITE( NOUT, FMT = 9997 )NPROW*NPCOL, NPROW, NPCOL
         WRITE( NOUT, FMT = 9996 )INFO
      END IF
*
*     Compute residual ||A * X  - B|| / ( ||X|| * ||A|| * eps * N )
*
```

```
      EPS = PDLAMCH( ICTXT, 'Epsilon' )
      ANORM = PDLANGE( 'I', N, N, A, 1, 1, DESCA, WORK )
      BNORM = PDLANGE( 'I', N, NRHS, B, 1, 1, DESCB, WORK )
      CALL PDGEMM( 'N', 'N', N, NRHS, N, ONE, A0, 1, 1, DESCA, B, 1, 1,
     $             DESCB, -ONE, B0, 1, 1, DESCB )
      XNORM = PDLANGE( 'I', N, NRHS, B0, 1, 1, DESCB, WORK )
      RESID = XNORM / ( ANORM*BNORM*EPS*DBLE( N ) )
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         IF( RESID.LT.10.0D+0 ) THEN
            WRITE( NOUT, FMT = 9995 )
            WRITE( NOUT, FMT = 9993 )RESID
         ELSE
            WRITE( NOUT, FMT = 9994 )
            WRITE( NOUT, FMT = 9993 )RESID
         END IF
      END IF
*
*     RELEASE THE PROCESS GRID
*     Free the BLACS context
*
      CALL BLACS_GRIDEXIT( ICTXT )
   10 CONTINUE
*
*     Exit the BLACS
*
      CALL BLACS_EXIT( 0 )
*
 9999 FORMAT( / 'ScaLAPACK Example Program #1 -- May 1, 1997' )
 9998 FORMAT( / 'Solving Ax=b where A is a ', I3, ' by ', I3,
     $      ' matrix with a block size of ', I3 )
 9997 FORMAT( 'Running on ', I3, ' processes, where the process grid',
     $      ' is ', I3, ' by ', I3 )
 9996 FORMAT( / 'INFO code returned by PDGESV = ', I3 )
 9995 FORMAT( /
     $   'According to the normalized residual the solution is correct.'
     $      )
 9994 FORMAT( /
     $ 'According to the normalized residual the solution is incorrect.'
     $      )
 9993 FORMAT( / '||A*x - b|| / ( ||x||*||A||*eps*N ) = ', 1P, E16.8 )
      STOP
      END
```

```
      SUBROUTINE MATINIT( AA, DESCA, B, DESCB )
*
*     MATINIT generates and distributes matrices A and B (depicted in
*     Figures 2.5 and 2.6) to a 2 x 3 process grid
*
*     .. Array Arguments ..
      INTEGER            DESCA( * ), DESCB( * )
      DOUBLE PRECISION   AA( * ), B( * )
*     ..
*     .. Parameters ..
      INTEGER            CTXT_, LLD_
      PARAMETER          ( CTXT_ = 2, LLD_ = 9 )
*     ..
*     .. Local Scalars ..
      INTEGER            ICTXT, MXLLDA, MYCOL, MYROW, NPCOL, NPROW
      DOUBLE PRECISION   A, C, K, L, P, S
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLACS_GRIDINFO
*     ..
*     .. Executable Statements ..
*
      ICTXT = DESCA( CTXT_ )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*
      S = 19.0D0
      C = 3.0D0
      A = 1.0D0
      L = 12.0D0
      P = 16.0D0
      K = 11.0D0
*
      MXLLDA = DESCA( LLD_ )
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         AA( 1 ) = S
         AA( 2 ) = -S
         AA( 3 ) = -S
         AA( 4 ) = -S
         AA( 5 ) = -S
         AA( 1+MXLLDA ) = C
         AA( 2+MXLLDA ) = C
         AA( 3+MXLLDA ) = -C
```

```
      AA( 4+MXLLDA ) = -C
      AA( 5+MXLLDA ) = -C
      AA( 1+2*MXLLDA ) = A
      AA( 2+2*MXLLDA ) = A
      AA( 3+2*MXLLDA ) = A
      AA( 4+2*MXLLDA ) = A
      AA( 5+2*MXLLDA ) = -A
      AA( 1+3*MXLLDA ) = C
      AA( 2+3*MXLLDA ) = C
      AA( 3+3*MXLLDA ) = C
      AA( 4+3*MXLLDA ) = C
      AA( 5+3*MXLLDA ) = -C
      B( 1 ) = 0.0D0
      B( 2 ) = 0.0D0
      B( 3 ) = 0.0D0
      B( 4 ) = 0.0D0
      B( 5 ) = 0.0D0
   ELSE IF( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
      AA( 1 ) = A
      AA( 2 ) = A
      AA( 3 ) = -A
      AA( 4 ) = -A
      AA( 5 ) = -A
      AA( 1+MXLLDA ) = L
      AA( 2+MXLLDA ) = L
      AA( 3+MXLLDA ) = -L
      AA( 4+MXLLDA ) = -L
      AA( 5+MXLLDA ) = -L
      AA( 1+2*MXLLDA ) = K
      AA( 2+2*MXLLDA ) = K
      AA( 3+2*MXLLDA ) = K
      AA( 4+2*MXLLDA ) = K
      AA( 5+2*MXLLDA ) = K
   ELSE IF( MYROW.EQ.0 .AND. MYCOL.EQ.2 ) THEN
      AA( 1 ) = A
      AA( 2 ) = A
      AA( 3 ) = A
      AA( 4 ) = -A
      AA( 5 ) = -A
      AA( 1+MXLLDA ) = P
      AA( 2+MXLLDA ) = P
      AA( 3+MXLLDA ) = P
      AA( 4+MXLLDA ) = P
```

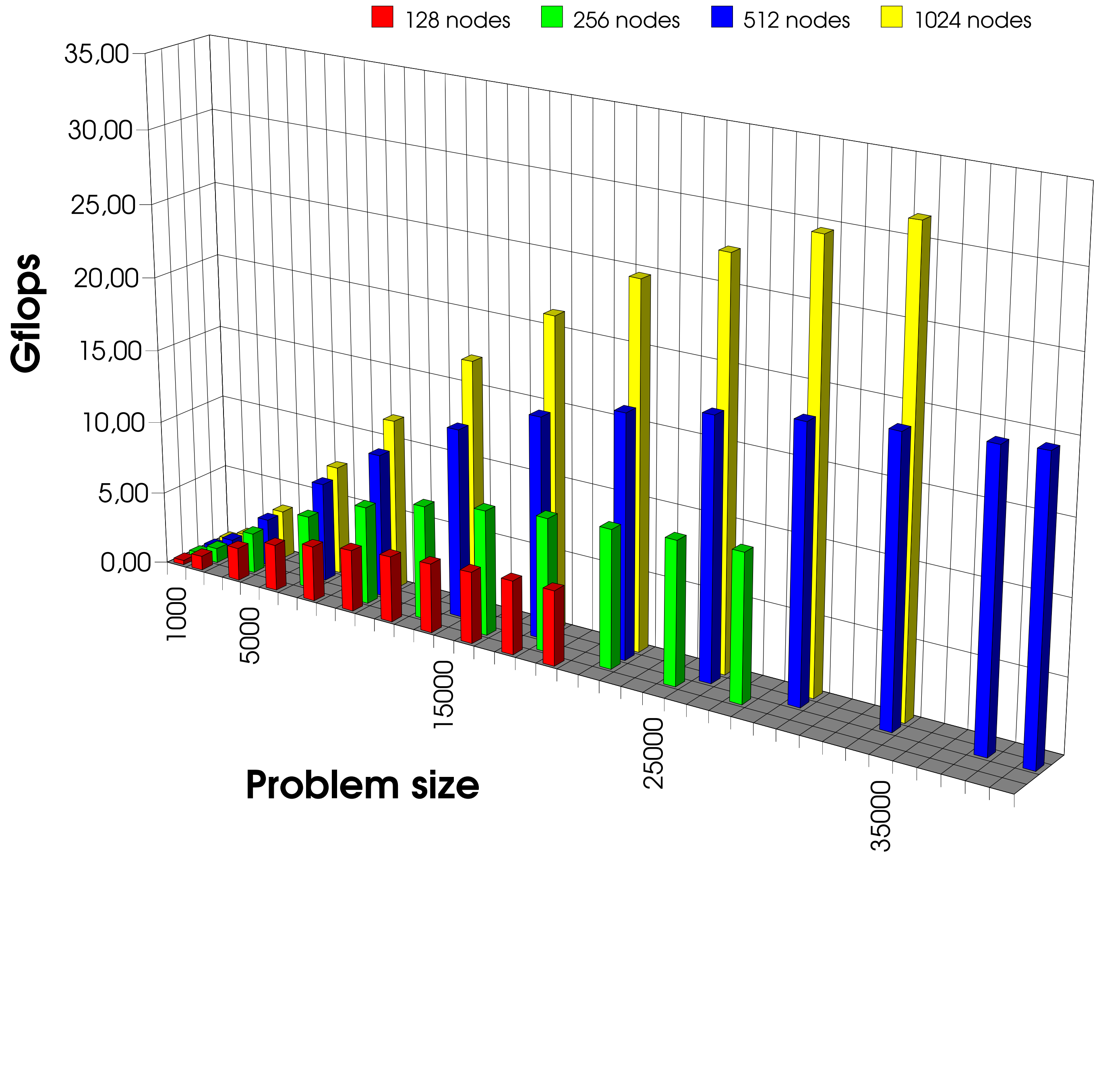
```
   AA( 5+MXLLDA ) = -P
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
   AA( 1 ) = -S
   AA( 2 ) = -S
   AA( 3 ) = -S
   AA( 4 ) = -S
   AA( 1+MXLLDA ) = -C
   AA( 2+MXLLDA ) = -C
   AA( 3+MXLLDA ) = -C
   AA( 4+MXLLDA ) = C
   AA( 1+2*MXLLDA ) = A
   AA( 2+2*MXLLDA ) = A
   AA( 3+2*MXLLDA ) = A
   AA( 4+2*MXLLDA ) = -A
   AA( 1+3*MXLLDA ) = C
   AA( 2+3*MXLLDA ) = C
   AA( 3+3*MXLLDA ) = C
   AA( 4+3*MXLLDA ) = C
   B( 1 ) = 1.0D0
   B( 2 ) = 0.0D0
   B( 3 ) = 0.0D0
   B( 4 ) = 0.0D0
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
   AA( 1 ) = A
   AA( 2 ) = -A
   AA( 3 ) = -A
   AA( 4 ) = -A
   AA( 1+MXLLDA ) = L
   AA( 2+MXLLDA ) = L
   AA( 3+MXLLDA ) = -L
   AA( 4+MXLLDA ) = -L
   AA( 1+2*MXLLDA ) = K
   AA( 2+2*MXLLDA ) = K
   AA( 3+2*MXLLDA ) = K
   AA( 4+2*MXLLDA ) = K
ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.2 ) THEN
   AA( 1 ) = A
   AA( 2 ) = A
   AA( 3 ) = -A
   AA( 4 ) = -A
   AA( 1+MXLLDA ) = P
   AA( 2+MXLLDA ) = P
   AA( 3+MXLLDA ) = -P
```

```
   AA( 4+MXLLDA ) = -P
END IF
RETURN
END
```

```
      SUBROUTINE SL_INIT( ICTXT, NPROW, NPCOL )
*
*     .. Scalar Arguments ..
      INTEGER            ICTXT, NPCOL, NPROW
*     ..
*
*  Purpose
*  =======
*
*  SL_INIT initializes an NPROW x NPCOL process grid using a row-major
*  ordering  of  the  processes. This routine retrieves a default system
*  context  which  will  include all available processes. In addition it
*  spawns the processes if needed.
*
*  Arguments
*  =========
*
*  ICTXT   (global output) INTEGER
*          ICTXT specifies the BLACS context handle identifying the
*          created process grid.  The context itself is global.
*
*  NPROW   (global input) INTEGER
*          NPROW specifies the number of process rows in the grid
*          to be created.
*
*  NPCOL   (global input) INTEGER
*          NPCOL specifies the number of process columns in the grid
*          to be created.
*
*  =====================================================================
*
*     .. Local Scalars ..
      INTEGER            IAM, NPROCS
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLACS_GET, BLACS_GRIDINIT, BLACS_PINFO,
     $                   BLACS_SETUP
*     ..
*     .. Executable Statements ..
*
*     Get starting information
*
      CALL BLACS_PINFO( IAM, NPROCS )
```

```
*
*     If machine needs additional set up, do it now
*
      IF( NPROCS.LT.1 ) THEN
         IF( IAM.EQ.0 )
     $      NPROCS = NPROW*NPCOL
         CALL BLACS_SETUP( IAM, NPROCS )
      END IF
*
*     Define process grid
*
      CALL BLACS_GET( -1, 0, ICTXT )
      CALL BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
*
      RETURN
*
*     End of SL_INIT
*
      END
```

LU Factorization on Intel Paragon



Performance of PBLAS = Parallel BLAS

- DGEMM = uniprocessor matrix-multiply
- PDGEMM = parallel matrix-multiply
- Efficiency(PDGEMM) =
  Speed(PDGEMM) / Peak Machine Speed ?
  No
- Efficiency(PDGEMM) =
  Speed(PDGEMM)/(#Procs*Speed(DGEMM))?
  Yes

Efficiency(PDGEMM)						
Machine	Peak/ proc	DGEMM Mflops	Procs	N		
				2000	4000	10000
Cray T3E	600	360	4	.73	.74	
			16	.63	.70	.75
			64	.58	.62	.73
IBM SP2	266	200	4	.94		
			16	.79	.89	
			64	.48	.68	.84
Intel XP/S MP Paragon	100	90	4	.92		
			16	.86	.89	
			64	.78	.84	.91
Berkeley NOW (Ultrasparcs on fast net)	334	129	4	.90	.91	
			16	.45	.50	
			64	.50	.66	.81

Speed in Mflops of PDGEMM = Parallel Mat-mul					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4=2x2	32	1055	1070	0
	16=4x4		3630	4005	4292
	64=8x8		13456	14287	16755
IBM SP2	4	50	755	0	0
	16		2514	2850	0
	64		6205	8709	10774
Intel XP/S MP Paragon	4	32	330	0	0
	16		1233	1281	0
	64		4496	4864	5257
Berkeley NOW (Ultrasparcs on fast net)	4	32	463	470	0
	16		926	1031	0
	32=4x8		2490	2822	3450
	64		4130	5457	6647

Efficiency = MFlops(PDGEMM)/(Procs*MFlops(DGEMM)) (DGEMM = uniprocessor Mat-mul)						
Machine	Peak/ proc	DGEMM Mflops	Procs	N		
				2000	4000	10000
Cray T3E	600	360	4	.73	.74	
			16	.63	.70	.75
			64	.58	.62	.73
IBM SP2	266	200	4	.94		
			16	.79	.89	
			64	.48	.68	.84
Intel XP/S MP Paragon	100	90	4	.92		
			16	.86	.89	
			64	.78	.84	.91
Berkeley NOW	334	129	4	.90	.91	
			32	.60	.68	.84
			64	.50	.66	.81

Performance of ScaLAPACK LU (Gauss. Elim.)

Efficiency = MFlops(PDGESV)/MFlops(PDGEMM)					
Machine	Procs	Block	N		
		Size	2000	4000	10000
Cray T3E	4	32	.67	.82	
	16		.44	.65	.84
	64		.18	.47	.75
IBM SP2	4	50	.56		
	16		.29	.52	
	64		.15	.32	.66
Intel XP/S MP Paragon	4	32	.64		
	16		.37	.66	
	64		.16	.42	.75
Berkeley NOW	4	32	.76		
	32		.38	.62	.71
	64		.28	.54	.69

Time(PDGESV)/Time(PDGEMM)					
Machine	Procs	Block	N		
		Size	2000	4000	10000
Cray T3E	4	32	.50	.40	
	16		.75	.51	.40
	64		1.86	.72	.45
IBM SP2	4	50	.60		
	16		1.16	.64	
	64		2.24	1.03	.51
Intel XP/S GP Paragon	4	32	.52		
	16		.89	.50	
	64		2.08	.79	.44
Berkeley NOW	4	32	.44		
	32		.88	.54	.47
	64		1.18	.62	.49

Performance of ScaLAPACK QR (Least squares)

Efficiency = MFlops(PDGELS)/MFlops(PDGEMM)					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.54	.61	
	16		.46	.55	.60
	64		.26	.47	.54
IBM SP2	4	50	.51		
	16		.29	.51	
	64		.19	.36	.54
Intel XP/S GP Paragon	4	32	.61		
	16		.43	.63	
	64		.22	.48	.62
Berkeley NOW	4	32	.51	.77	
	32		.49	.66	.71
	64		.37	.60	.72

Time(PDGELS)/Time(PDGEMM)					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	1.2	1.1	
	16		1.5	1.2	1.1
	64		2.6	1.4	1.2
IBM SP2	4	50	1.3		
	16		2.3	1.3	
	64		3.6	1.8	1.2
Intel XP/S GP Paragon	4	32	1.1		
	16		1.6	1.1	
	64		3.0	1.4	1.1
Berkeley NOW	4	32	1.3	.9	
	32		1.4	1.0	.9
	64		1.8	1.1	.9

Performance of Symmetric Eigensolvers

Time(PDSYEVX)/Time(PDGEMM) (bisection + inverse iteration)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	10	
	16		13	10
	64		29	14
IBB SP2	16	50	24	
	64		40	29
Intel XP/S GP Paragon	16	32	22	
	64		34	20
Berkeley NOW	16	32	20	
	32		24	52

Time(PDSYEV)/Time(PDGEMM) (QR iteration)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	35	
	16		37	35
	64		57	41
IBM SP2	16	50	38	
	64		58	47
Intel XP/S GP Paragon	16	32	99	
	64		193	
Berkeley NOW	16	32	31	
	32		35	55

Performance of SVD
(Singular Value Decomposition)

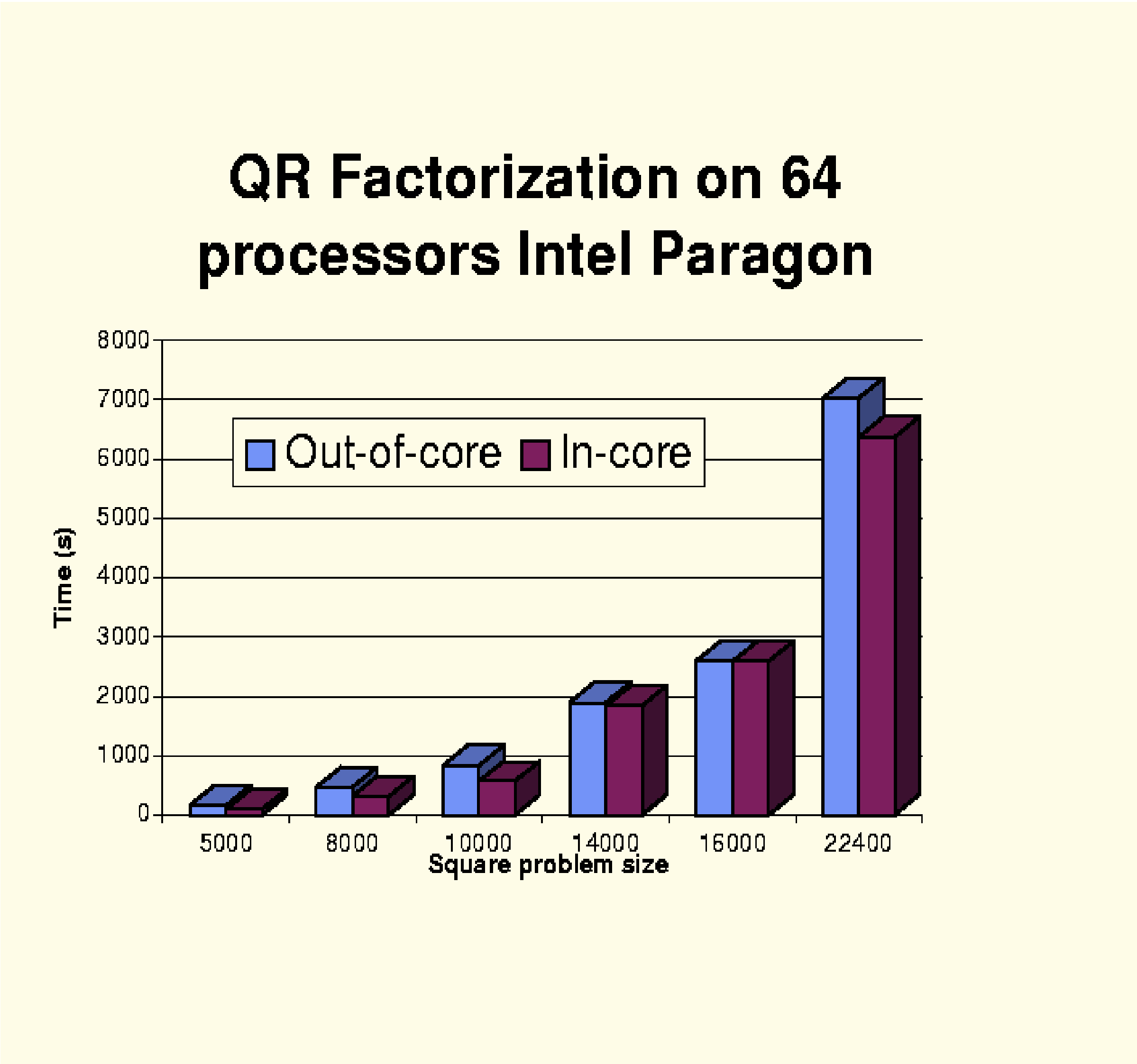
Time(PDGESVD)/Time(PDGEMM)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	67	
	16		66	64
	64		93	70
IBM SP2	4	50	97	
	16		60	
	64		81	
Berkeley NOW	4	32	72	
	16		38	16
	32		59	26

Performance of Nonsymmetric Eigensolver
(QR iteration)

Time(PDLAHQR)/Time(PDGEMM)				
Machine	Procs	Block Size	N 1000	N 1500
Intel XP/S MP Paragon	16	50	123	97

Out-of-Core Performance Results for Least Squares

- Prototype code for Out-of-Core extension
- Linear solvers based on "Left-looking" variants of LU, QR, and Cholesky factorization
- Portable I/O interface for reading/writing ScaLAPACK matrices



Future Work

- **Increased flexibility and usability**
  - **Algorithmic redistribution in PBLAS**
  - **Removal of alignment restrictions**
  - **Algorithmic blocking**
- **Algorithmic Innovations**
  - **“Holy Grail” for Symmetric $Ax = \lambda x$**
  - **Replaces worst case serial $O(n^3)$ algorithm with embarrassingly paralllel $O(n^2/procs)$**
- **HPF interface to all ScaLAPACK driver routines**
- **Parallel Sparse Direct Solvers (SuperLU)**
- **C++ and Java Interface to ScaLAPACK**

Heterogeneous Computing

- Ex: Running ScaLAPACK on network of Ultrasparcs, Alphas, RS6000s, Pentiums...
- Heterogeneity means
  - Floating point results from identical source code may differ on different processors, or
  - Floating point results from identical source code may differ from run to run, or
  - Floating point numbers may change when communicated
- Can arise from different hardware, compilers, compiler flags, communication layer, ...
- Examples of problems, solutions
  - Different results from run to run (eg from nondeterministic sums) make debugging hard; add "repeatability" flag to BLACS, with some performance penalty
  - Sending a denormalized (tiny!) number from Ultrasparc to Alpha can make Alpha "hang"; no fix without big performance penalty

- $s = global\_sum(x)$

  if $s < macheps$ **then**

  return answer A

  else

  communicate some more

  return answer B

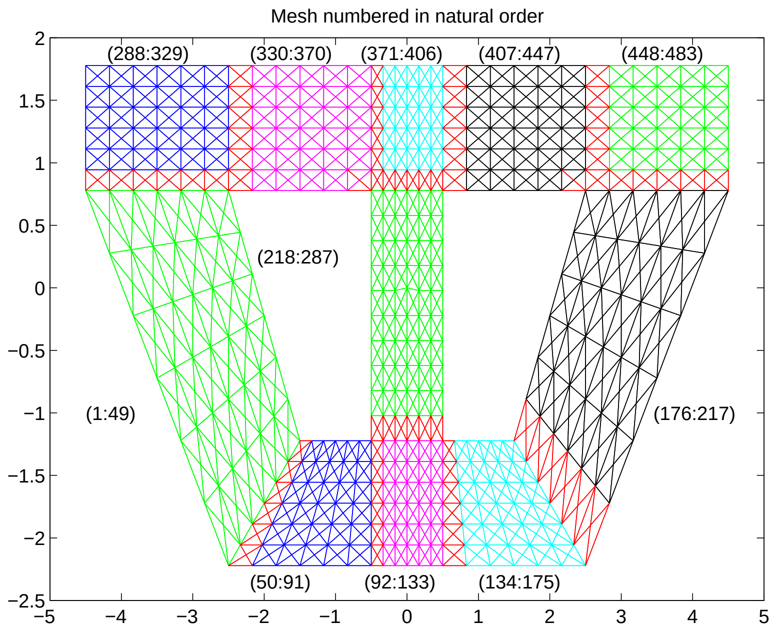
  end

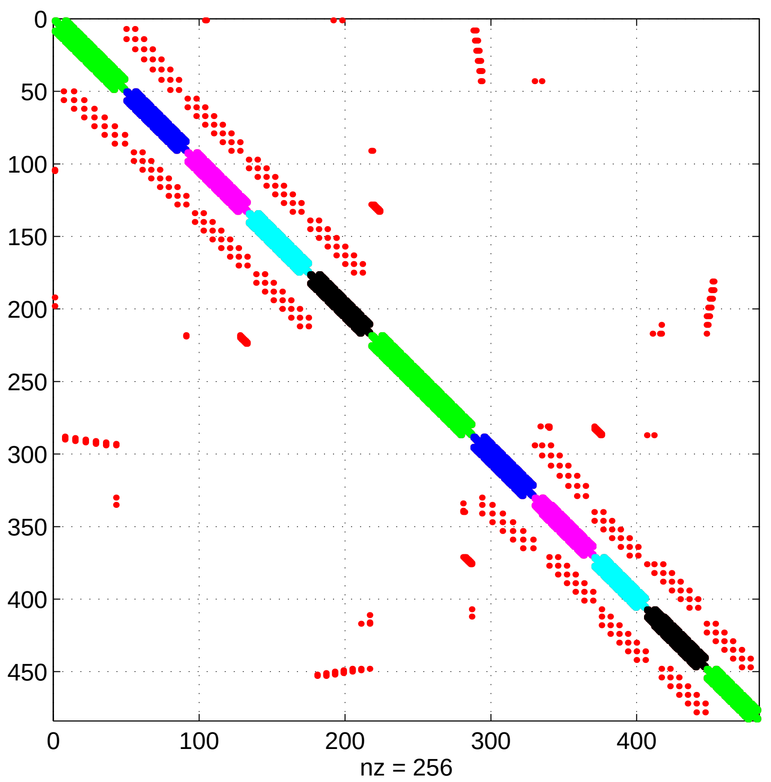
  Code sequence can return A **on some procs,** B **on others, or even deadlock; have one processor compute and broadcast result of** $s < macheps$

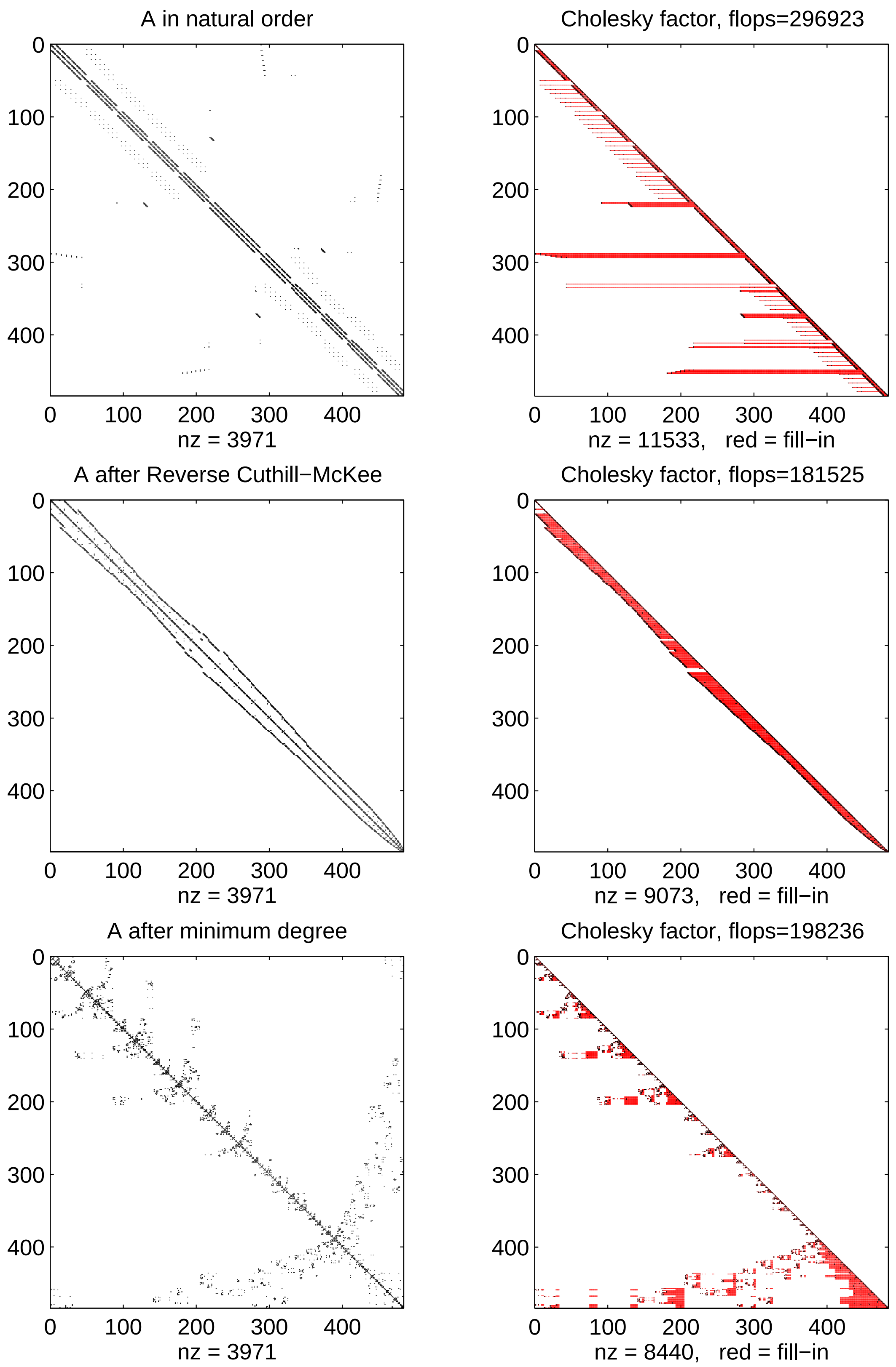
- **Parallel QR algorithm for eigenvalues does** $O(n^2)$ **work redundantly to prepare for** $O(n^3/p)$ **work in parallel; "redundant" results may differ drastically if rounding slightly different; fix requires** $O(n^2)$ **communication, or just warning user if heterogeneity suspected**

Sparse Gaussian Elimination

- Sparse Elimination much more complicated than Dense
  - Only store and compute with nonzero entries in A
  - More nonzeros – "fill-in" – as elimination proceeds
  - Hard to organize algorithm so that most time not spent in traversing data structures, rather than floating point
  - Fill-in, parallelizability, cost depends strongly on order of equations
- Many parallelization strategies, depending on structure


Mesh numbered in natural order
(288:329)
(330:370)
(371:406)
(407:447)
(448:483)
(218:287)
(1:49)
(176:217)
(50:91)
(92:133)
(134:175)


nz = 256


A in natural order
nz = 3971
Cholesky factor, flops=296923
nz = 11533,   red = fill−in
A after Reverse Cuthill−McKee
nz = 3971
Cholesky factor, flops=181525
nz = 9073,   red = fill−in
A after minimum degree
nz = 3971
Cholesky factor, flops=198236
nz = 8440,   red = fill−in

Sparse Direct Solvers

Matrix Type	Name	Algorithm	Status /Source
Serial Algorithms			
nonsym.	SuperLU	LL, partial, BLAS-2.5	Pub/UCB
nonsym.	UMFPACK [3, 4]	MF, Markowitz, BLAS-3	Pub/NETLIB
	MA38 (same as UMFPACK)		Com/HSL
nonsym.	MA48 [6]	Anal: RL, Markowitz Fact: LL, partial, BLAS-1, SD	Com/HSL
nonsym.	SPARSE [14]	RL, Markowitz, Scalar	Pub/NETLIB
sym-pattern	MUPS [1]	MF, threshold, BLAS-3	Com/HSL
sym-pattern	MA42 [8]	Frontal, BLAS-3	Com/HSL
sym.	MA27 [7]/MA47 [5]	MF, LDL^T, BLAS-1	Com/HSL
s.p.d.	Ng & Peyton [16]	LL, BLAS-3	Pub/Author
Shared Memory Algorithms			
nonsym.	SuperLU	LL, partial, BLAS-2.5	Pub/UCB
nonsym.	PARASPAR [19, 20]	RL, Markowitz, BLAS-1, SD	Res/Author
sym-pattern	MUPS [2]	MF, threshold, BLAS-3	Res/Author
nonsym.	George & Ng [9]	RL, partial, BLAS-1	Res/Author
s.p.d.	Gupta, Rothberg, Ng & Peyton [11]	LL, BLAS-3	Com/SGI Pub/Author
s.p.d.	SPLASH [13]	RL, 2-D block, BLAS-3	Pub/Stanford
Distributed Memory Algorithms			
sym.	van der Stappen [18]	RL, Markowitz, Scalar	Res/Author
sym-pattern	Lucas et al. [15]	MF, no pivoting, BLAS-1	Res/Author
s.p.d.	Rothberg et al. [17]	RL, 2-D block, BLAS-3	Res/Author
s.p.d.	Gupta [10]	MF, 2-D block, BLAS-3	Res/Author
s.p.d.	CAPSS [12]	MF, full parallel, BLAS-1 (require coordinates)	Pub/NETLIB

Abbreviations used in the table:

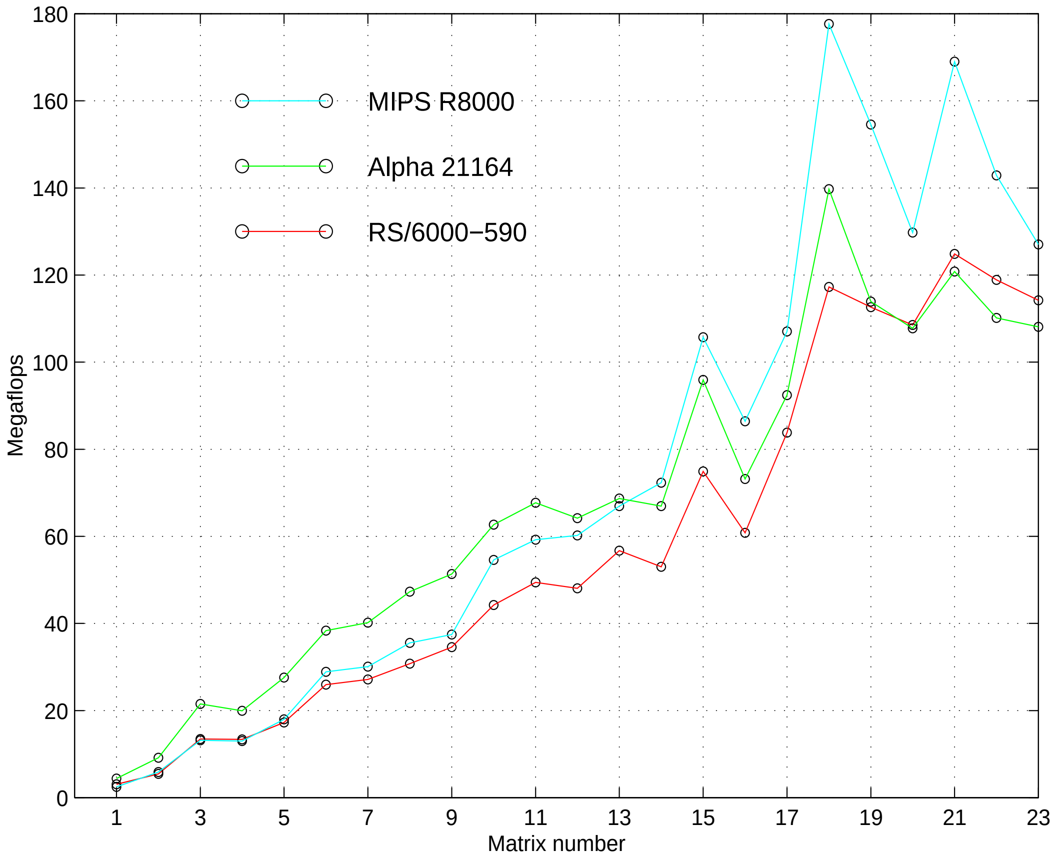
nonsym. – nonsymmetric.
sym-pattern – symmetric nonzero structure, nonsymmetric values.
sym. – symmetric and may be indefinite.
s.p.d – symmetric and positive definite.
MF, LL and RL – multifrontal, left-looking and right-looking.
SD – switches to a dense code on a sufficiently dense trailing submatrix.
Pub – publically available; authors may help use the code.
Res – published in literature but may not be available from the authors.
Com – commercial.
HSL – Harwell Subroutine Library.
http://www.rl.ac.uk/departments/ccd/numerical/hsl/hsl.html.
UCB – http://www.cs.berkeley.edu/~xiaoye/superlu.html.
Stanford – http://www-flash.stanford.edu/apps/SPLASH/.

SuperLU Test Matrices

	Name	N	$nnz(A)$	$\frac{nnz(A)}{N}$	$\frac{nnz(F)}{nnz(A)}$	$\frac{\#\text{flops}}{10^6}$	$Q = \frac{\#flops}{nnz(F)}$
1	Memplus	17758	99147	5.6	1.4	1.8	12.5
2	Gemat11	4929	33108	6.7	2.8	1.5	16.3
3	Rdist1	4134	94408	22.8	3.6	12.9	38.1
4	Orani678	2529	90158	35.6	3.1	14.9	53.3
5	Mcfe	765	24382	31.8	2.8	4.1	59.9
6	Lnsp3937	3937	25407	6.5	16.8	38.9	91.1
7	Lns3937	3937	25407	6.5	17.7	44.8	99.7
8	Sherman5	3312	20793	6.3	12.0	25.2	101.3
9	Jpwh991	991	6027	6.1	23.4	18.0	127.7
10	Sherman3	5005	20033	4.0	21.6	60.6	139.8
11	Orsreg1	2205	14133	6.4	28.5	59.8	148.6
12	Saylr4	3564	22316	6.3	29.3	104.8	160.0
13	Shyy161	76480	329762	4.3	23.2	1571.6	205.8
14	Goodwin	7320	324772	44.4	9.6	665.1	213.9
15	Venkat01	62424	1717792	27.5	7.6	3219.9	247.9
16	Inaccura	16146	1015156	62.9	9.8	4118.7	414.3
17	Af23560	23560	460598	19.6	30.4	6363.7	454.9
18	Dense1000	1000	1000000	1000	1.0	666.2	666.2
19	Raefsky3	21200	1488768	70.2	11.8	12118.7	690.7
20	Ex11	16614	1096948	66.0	23.8	26814.5	1023.1
21	Wang3	26064	177168	6.8	74.9	14557.5	1095.5
22	Raefsky4	19779	1316789	66.6	20.3	31283.4	1172.6
23	Vavasis3	41029	1683902	41.0	29.2	89209.3	1813.5

- Sorted in increasing order of $Q = \frac{\#flops}{nnz(F)}$. $(F = L + U - I)$
- Columns preordered by Matlab's minimum degree

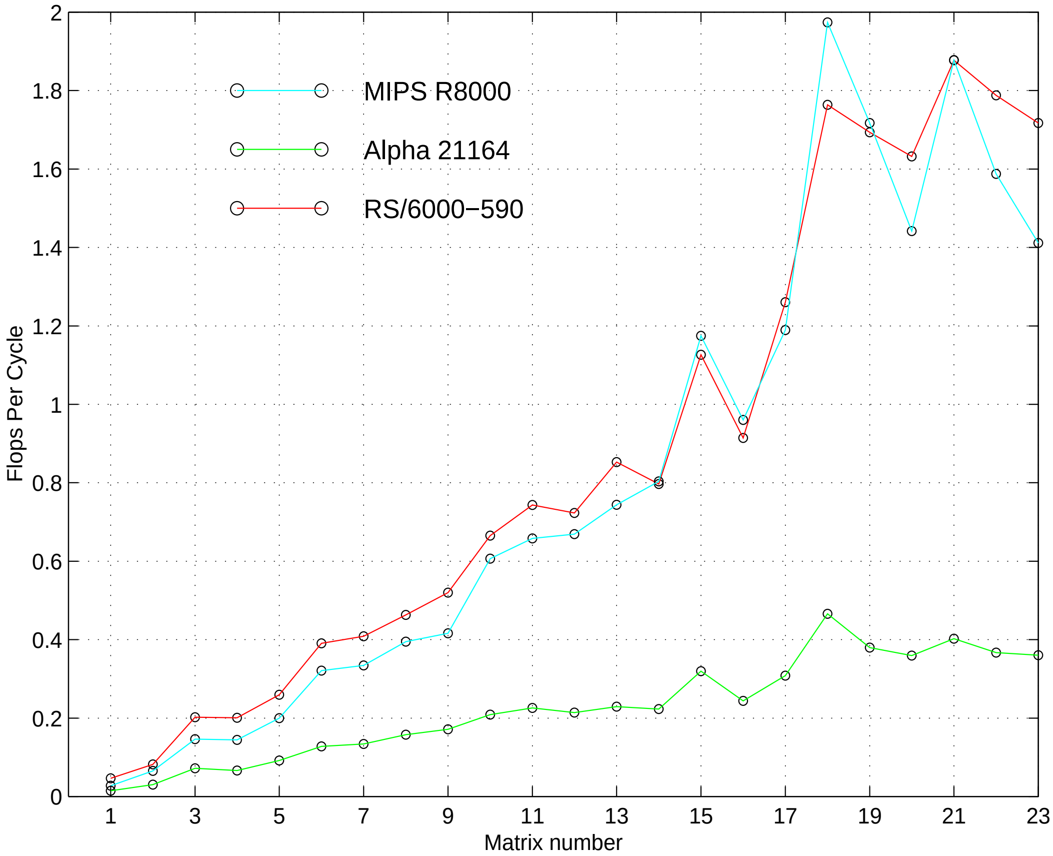
SuperLU Sequential Performance in MFlops



- Examples sorted in increasing order of $Q = \#flops/nnz(F)$
- Matrix 18 is 1000-by-1000 dense

Processor	Clock (MHz)	Peak (Mflops)	DGEMM (Mflops)	LAPACK (Mflops)
MIPS R8000	90	360	340	241
Alpha 21164	300	600	350	250
RS/6000-590	66.5	266	250	168

SuperLU Sequential Performance in #Flops/Cycle

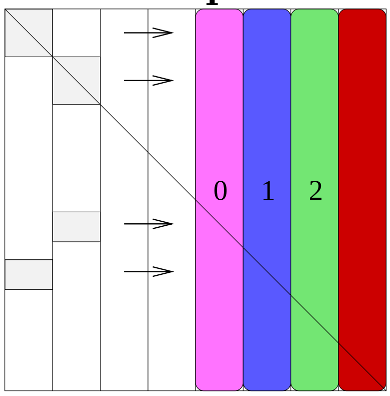


- **Examples sorted in increasing order of** $Q = \#flops/nnz(F)$
- **Matrix 18 is 1000-by-1000 dense**

Processor	Clock (MHz)	Peak (#flops/cycle)
MIPS R8000	90	4
Alpha 21164	300	2
RS/6000-590	66.5	4

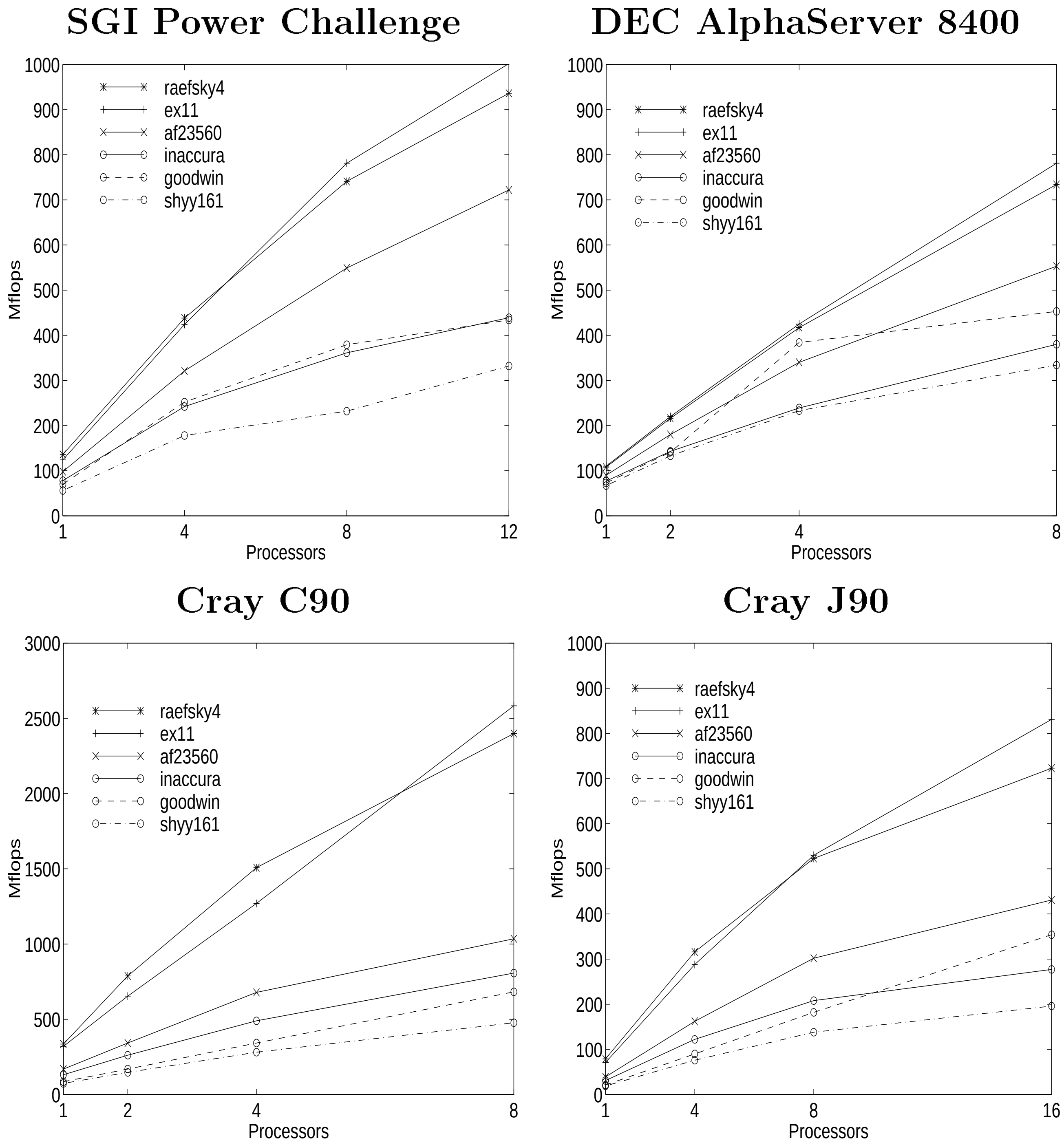
Shared Memory Parallel SuperLU

- 1D data layout across processors



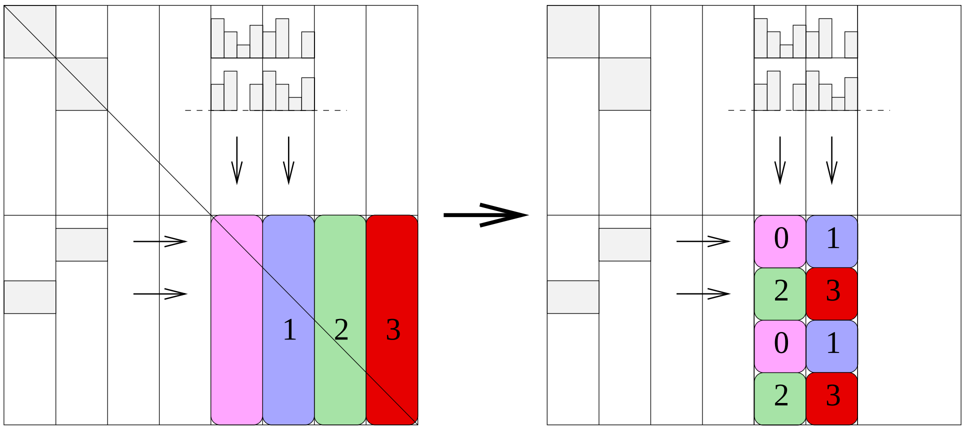
- Central scheduler dynamically assigns tasks to processors, using threads
- Almost identical code for
  - Cray C90 & J90
  - SGI Power Challenge
  - Sun SPARCcenter 2000
  - DEC AlphaServer 8400
  - NEC SX-4
- Manual, Code available at
  `http://www.cs.berkeley.edu/~xiaoye`

Shared Memory Parallel SuperLU MFlops



Future Work

- Study more scalable algorithms for larger parallel machines.



+ higher level of concurrency.

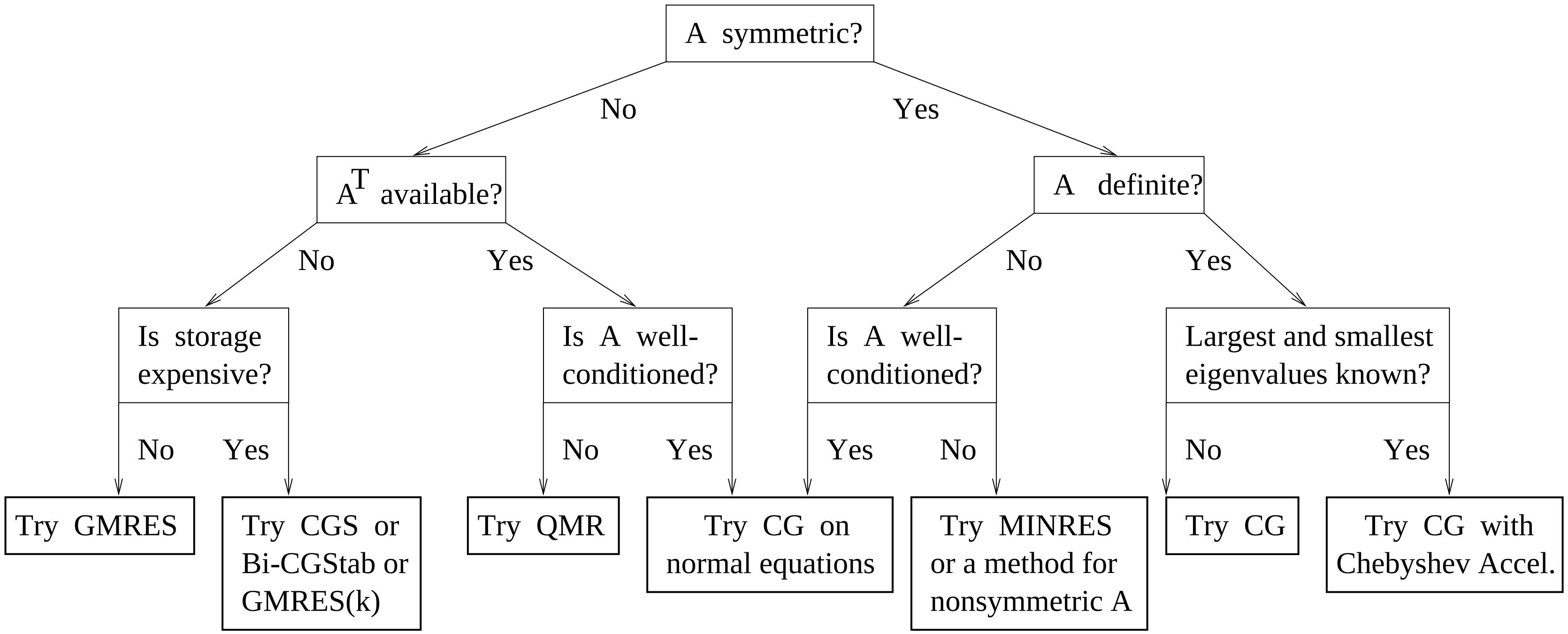
– more synchronizations.

– more complex data structures.

- Incomplete LU factorizations as preconditioners for iterative solvers.

Krylov Subspace Methods

- Iterative methods for $Ax = b$ and $Ax = \lambda x$.
- Ingredients
  - Matrix multiplication Ax, possibly $A^T z$
  - Dot products, saxpys, scaling (scalar*vector)
  - "Preconditioner" M, such that MA is better conditioned than A
- Many methods with different properties
  - For $Ax = b$: CG, GMRES, QMR, ...
  - For $Ax = \lambda x$: Lanczos, Arnoldi, ...
- Many preconditioners: Jacobi, SOR, ILU, Domain Decomposition, Multigrid, ...
- Best choice problem dependent, no easy answer



A symmetric?
No
Yes
A^T available?
A definite?
No
Yes
No
Yes
Is storage expensive?
Is A well-conditioned?
Is A well-conditioned?
Largest and smallest eigenvalues known?
No
Yes
No
Yes
Yes
No
No
Yes
Try GMRES
Try CGS or Bi-CGStab or GMRES(k)
Try QMR
Try CG on normal equations
Try MINRES or a method for nonsymmetric A
Try CG
Try CG with Chebyshev Accel.

Libraries for Krylov Subspace Methods

- "Templates for the Solution of Linear Systems"
  - Book with short description of each method, advice
  - Pseudocode + Matlab, Fortran, C
  - `http://www.netlib.org/templates/Templates.html`
- (P)ARPACK
  - Methods for $Ax = \lambda x$, $Ax = \lambda Bx$
  - Symmetric and nonsymmetric A
  - Fortran + message passing
  - `http://www.netlib.org/scalapack`
- PETSc
  - Portable Extensible Toolkit for Scientific Computation
  - Object oriented collection of solvers, aimed at PDEs
  - Can assemble complex methods from simple building blocks
  - `http://www.mcs.anl.gov/petsc`

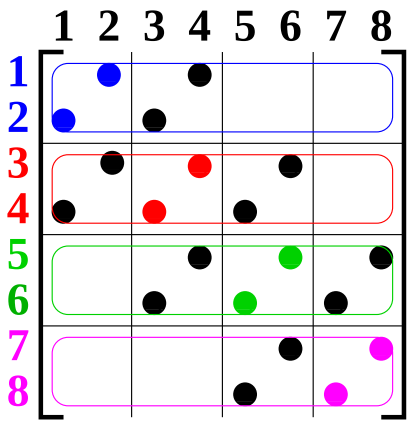
Parallelizing Sparse Matrix-Vector Multiply $y = A \cdot x$

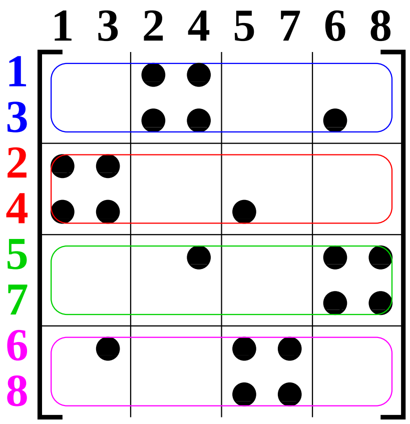
- **Decisions:**
  - **On which procs are entries of A, X, y stored?**
  - **Which proc computes each y_i?**
- **Approach**
  - **Partition $\{1, 2, ..., n\} = N_1 \cup N_2 \cup \cdots \cup N_p$**
  - **Processor k stores x_i, y_i, row i of A for all $i \in N_k$**
  - **Processor k computes all y_i that it stores:**

$$y_i = \sum_j A_{ij} x_j = \sum_{j\,:\,A_{ij} \neq 0} A_{ij} x_j$$

- **Goals of partitioning**
  - **Balance the load: each processor does about equally many flops**
    $\Rightarrow$ each processor should store about equally many nonzeros in A
  - **Minimize communication: if $i \in N_k$, proc k computes**
    $$y_i = \underbrace{\sum_{j \in N_k\,:\,A_{ij} \neq 0} A_{ij} x_j}_{\text{all data local}} + \underbrace{\sum_{j \notin N_k\,:\,A_{ij} \neq 0} A_{ij} x_j}_{\text{need to communicate } x_j}$$
    $\Rightarrow$ minimize number of A_{ij} where i and j are in different N_k

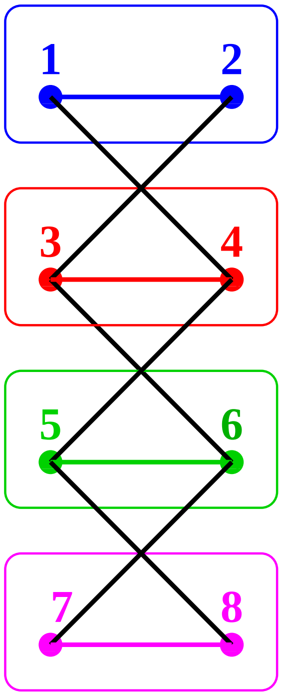
Partitioning a Sparse Symmetric Matrix

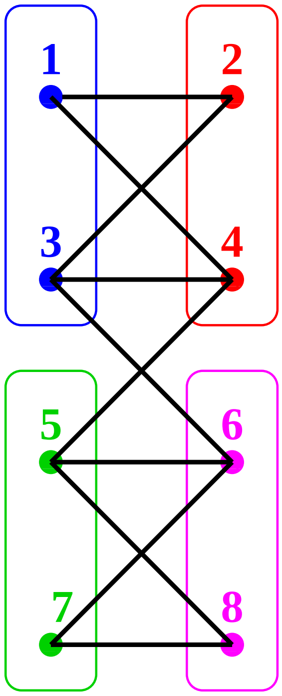




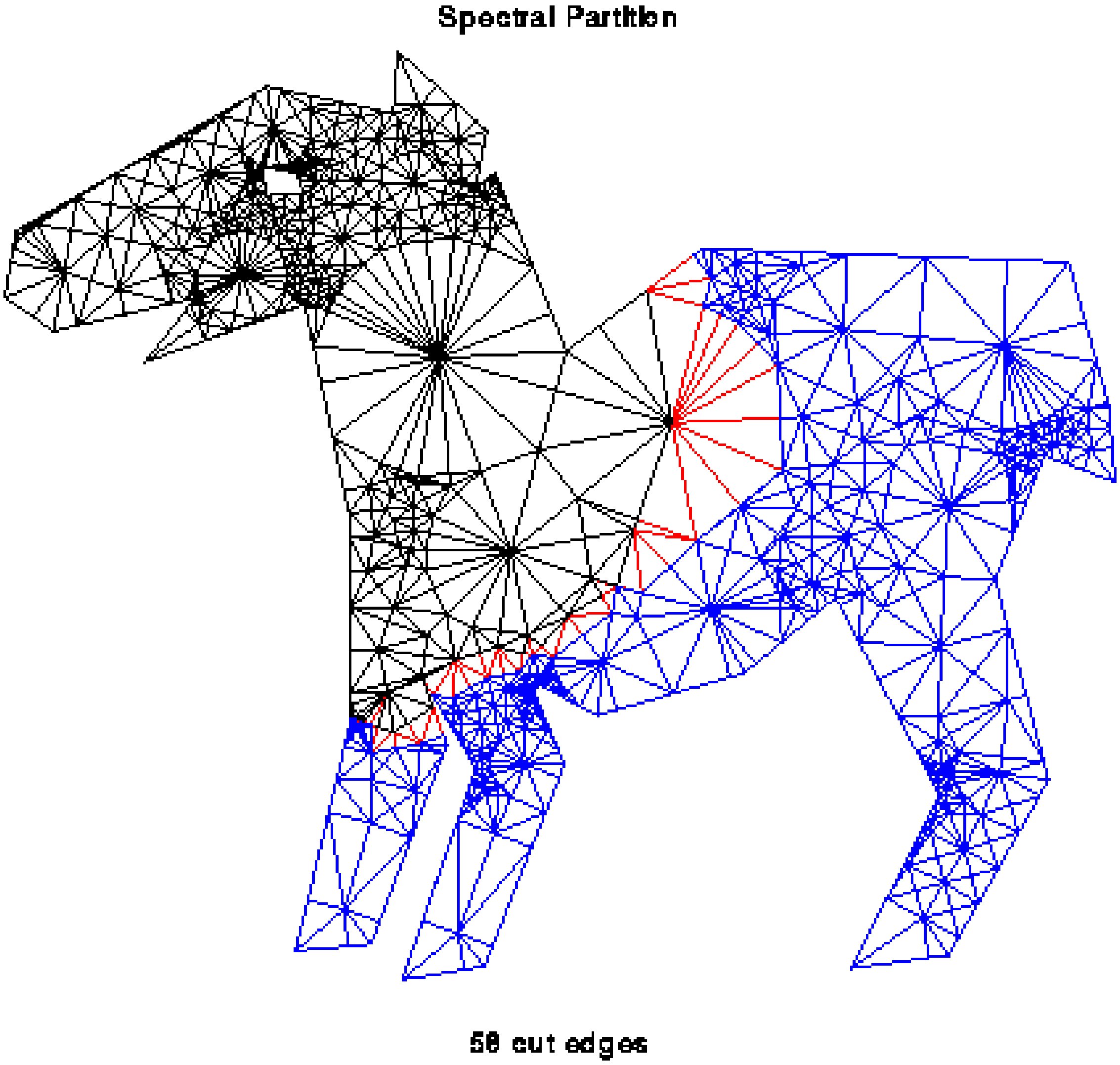
 # Edge Crossings = 6

 # Edge Crossings = 10





More Graph Partitioning



- **Graph Partitioning Library: (Par)METIS:**
  `http://www.cs.umn.edu/~karypis/metis`

For further information ...

- **Netlib:** `www.netlib.org` (≡ `NETLIB` **below)**
- **(Unoptimized) BLAS:** `NETLIB/blas`
- **BLAS Forum:** `NETLIB/utk/papers/blast-forum.html`
- **PHIPAC:** `www.icsi.berkeley.edu/~bilmes/phipac`
- **LAPACK:** `NETLIB/lapack`
- **ScaLAPACK:** `NETLIB/scalapack`
- **SuperLU:** `NETLIB/scalapack/prototype`
- **UMFPACK:** `NETLIB/linalg`
- **CAPSS:** `NETLIB/scalapack`
- **(P)ARPACK:** `NETLIB/scalapack`
- **PETSc:** `www.mcs.anl.gov/petsc`
- **GAMS:** `NETLIB/bib/gams.html`
- **Templates for** $Ax = b$: `NETLIB/templates/Templates.html`
- **(Par)METIS:** `www.cs.umn.edu/~karypis/metis`
- **Parallel Computing Course:** `www.cs.berkeley.edu/~demmel/cs267`
- **Numerical Linear Algebra Course:** `www.cs.berkeley.edu/~demmel/ma221`

References for Sparse Direct Solvers

References

[1] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *The International Journal of Supercomputer Applications*, 3:41–59, 1989.

[2] Patrick R. Amestoy. Factorization of large unsymmetric sparse matrices based on a multifrontal approach in a multiprocessor environment. Technical Report TH/PA/91/2, CERFACS, Toulouse, France, February 1991. Ph.D thesis.

[3] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report RAL 93-036, Rutherford Appleton Laboratory, Chilton, Didcot, Oxfordshire, 1994.

[4] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, Computer and Information Sciences Department, University of Florida, 1995.

[5] I.S Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL-95-001, Rutherford Appleton Laboratory, 1995.

[6] I.S Duff and J. K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 22:187–226, 1996.

[7] I.S Duff and J.K Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, September 1983.

[8] I.S Duff and J. A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Transactions on Mathematical Software*, 22(1):30–45, 1996.

[9] Alan George and Esmond Ng. Parallel sparse Gaussian elimination with partial pivoting. *Annals of Operation Research*, 22:219–240, 1990.

[10] A. Gupta and V. Kumar. Optimally scalable parallel sparse Cholesky factorization. In *Proceedings of the Seventh SIAM Conference on Parallel Proceesing for Scientific Computing*, pages 442–447. SIAM, 1995.

[11] A. Gupta, E. Rothberg, E. Ng, and B. W. Peyton. Parallel sparse Cholesky factorization algorithms for shared-memory multiprocessor systems. In R. Vichnevetsky, D. Knight, and G. Richter, editors, *Advances in Computer Methods for Partial Differential Equations–VII*. IMACS, 1992.

[12] M. T. Heath and P. Raghavan. Performance of a fully parallel sparse solver. In *Proc. Scalable High-Performance Computing Conf.*, pages 334–341, Los Alamitos,CA, 1994. IEEE.

[13] W-D. Webber J.P. Singh and A. Gupta. Splash: Stanford parallel applications for shared-memory. *Computer Architecture News*, 20(1):5–44, 1992.

[14] K. Kundert. Sparse matrix techniques. In Albert Ruehli, editor, *Circuit Analysis, Simulation and Design*. North-Holland, 1986.

[15] R. Lucas, W. Blank, and J. Tieman. A parallel solution method for large sparse systems of equations. *IEEE Trans. Computer Aided Design*, CAD-6:981–991, 1987.

[16] E. G. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Stat. Comp.*, 14:1034–1056, 1993.

[17] Edward Rothberg and Robert Schreiber. Improved load distribution in parallel sparse cholesky factorization. In *Supercomputing*, pages 783–792, November 1994.

[18] A. Frank van der Stappen, Rob H. Bisseling, and Johannes G. G. van der Vorst. Parallel sparse LU decomposition on a mesh network of transputers. *SIAM J. Mat. Anal. Appl.*, 14(3):853–879, July 1993.

[19] Z. Zlatev, J. Waśniewski, P. C. Hansen, and Tz. Ostromsky. PARASPAR: a package for the solution of large linear algebraic equations on parallel computers with shared memory. Technical Report 95-10, Technical University of Denmark, September 1995.

[20] Zahari Zlatev. *Computational methods for general sparse matrices.* Kluwer Academic, Dordrecht; Boston, 1991.