Designing High Performance Linear Algebra Software for Parallel Computers

James Demmel
CS Division and Math Dept.
UC Berkeley
demmel@cs.berkeley.edu

December 9, 1994

Participants

Zhaojun Bai (U Kentucky)
Michael Berry (U Tenn)
Adam Bojanczyk (Cornell U)
Tony Chan (UCLA)
Inderjit Dhillon (UC Berkeley)
Jack Dongarra (U Tenn, ORNL)
Jeremy Du Croz (NAG)
Vince Fernando (NAG)
Sven Hammarling (NAG)
Steve Huss-Lederman (SRC)
Velvel Kahan (UC Berkeley)
Rencang Li (UC Berkeley)
Alan McKenney (NYU)
Beresford Parlett (UC Berkeley)
Chris Puscasiu (UC Berkeley)
Roldan Pozo (U Tenn)
Huan Ren (UC Berkeley)
Charles Romine (ORNL)
Eunice Santos (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)
Richard Barrett (U. Tenn)
Chris Bischof (ANL)
Soumen Chakrabarti (UC Berkeley)
Jaeyoung Choi (U Tenn)
June Donato (ORNL)
Zlatko Drmač (U Hagen)
Victor Eijkhout (U Tenn)
Ming Gu (UC Berkeley, LBL)
Mike Heath (U Illinois)
Bo Kågström (U Umeå)
Youngbae Kim (U Tenn)
Xiaoye Li (UC Berkeley)
Susan Ostrouchov (U Tenn)
Antoine Petitet (U Tenn)
Peter Poromaa (U Umeå)
Padma Raghavan (U Illinois)
Howard Robinson (UC Berkeley)
Jeff Rutter (UC Berkeley)
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)

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

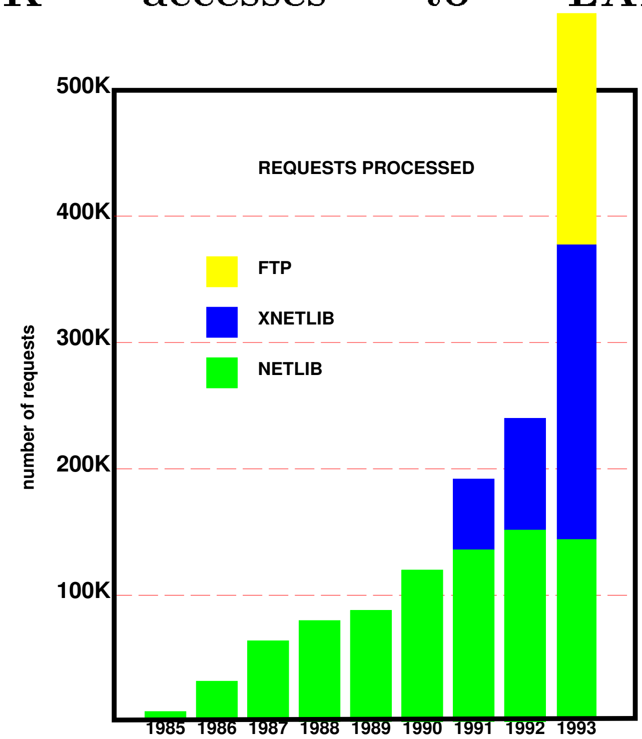
Supported by ARPA, NSF, DOE

Outline

- **Who uses Numerical Libraries?**
- **Motivation for LAPACK/ScaLAPACK**
- **LAPACK - a linear algebra library for single address space machines**
- **ScaLAPACK - a linear algebra library for multiple address space machines**
- **Castle - a unified parallel programming environment at Berkeley**
- **Where to get software**

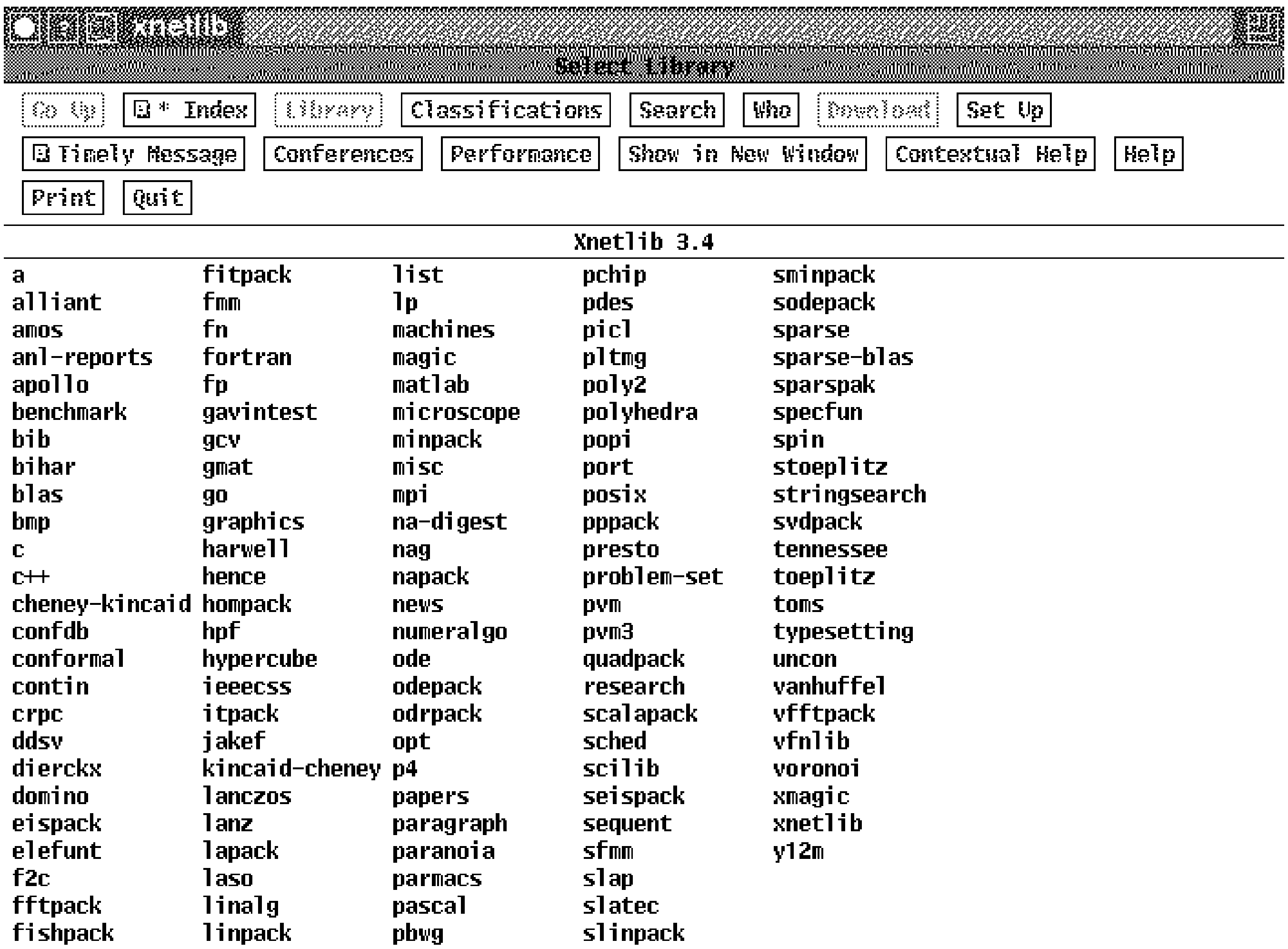
Who uses Numerical Libraries?

- Netlib - on-line library server
- Includes LAPACK, ScaLAPACK, much more
- Accessible via e-mail, ftp, Xnetlib, gopher, Mosaic: http://www.netlib.org{/lapack,/scalapack}
- Over 222K accesses to LAPACK



- 1.8 Million requests in 1993 from 5 sites in US, Europe, Asia

Xnetlib Interface



Different Users with Different Needs

- “Black boxes” for average users
  - Traditional library users
  - Reliability and ease-of-use come before speed
- “Template” or “toolbox” for expert users
  - Requires user intervention
  - Supply building blocks from which user tunes his implementation
  - High performance possible, at cost of reliability for general problems
  - Interface of choice for HPCC applications?
- Many more average than expert users
- Tradeoff between speed and stability

Motivation for LAPACK

- **LINPACK, EISPACK - standard Fortran libraries for linear algebra**
- **Example: Cray Y-MP with 1 and 8 Processors (speeds in Mflops)**

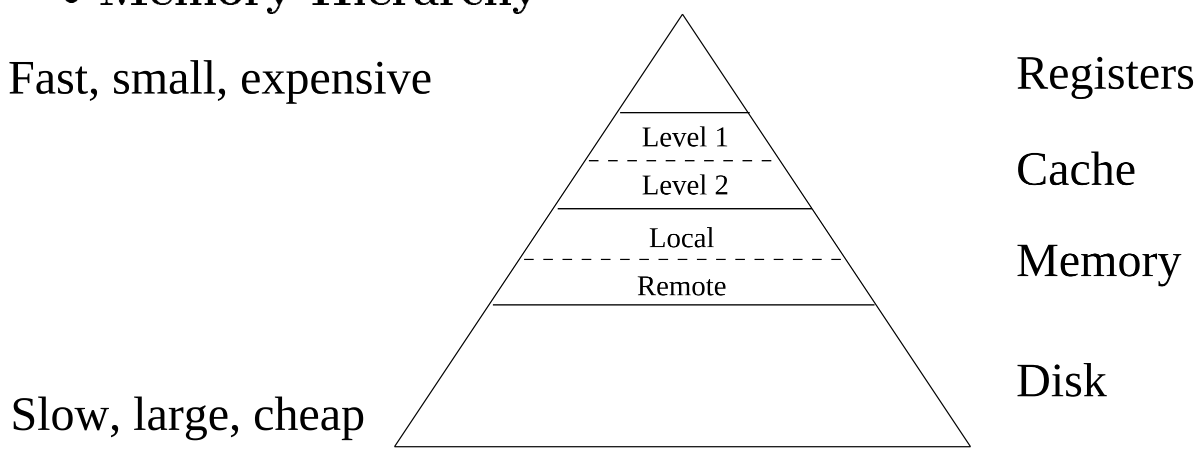
	1 PE	**8 PEs**
Maximum Speed	**330**	**2640**
LINPACK (Cholesky, $n = 500$**)**	**72**	**72**
Matrix-vector multiply	**311**	**2285**
Matrix-matrix multiply	**312**	**2425**
Solve $Tx = b$	**272**	**584**
Solve $TX = B$	**309**	**2398**

- **LINPACK flops lost to data movement in memory hierarchy**
- **Cure: Use BLAS (Basic Linear Algebra Subroutines)**

	1 PE	**8 PEs**
LAPACK (Cholesky, $n = 500$**)**	**290**	**1414**
LAPACK (Cholesky, $n = 1024$**)**	**301**	**2115**

Where do the Flops go?

- **Memory Hierarchy**

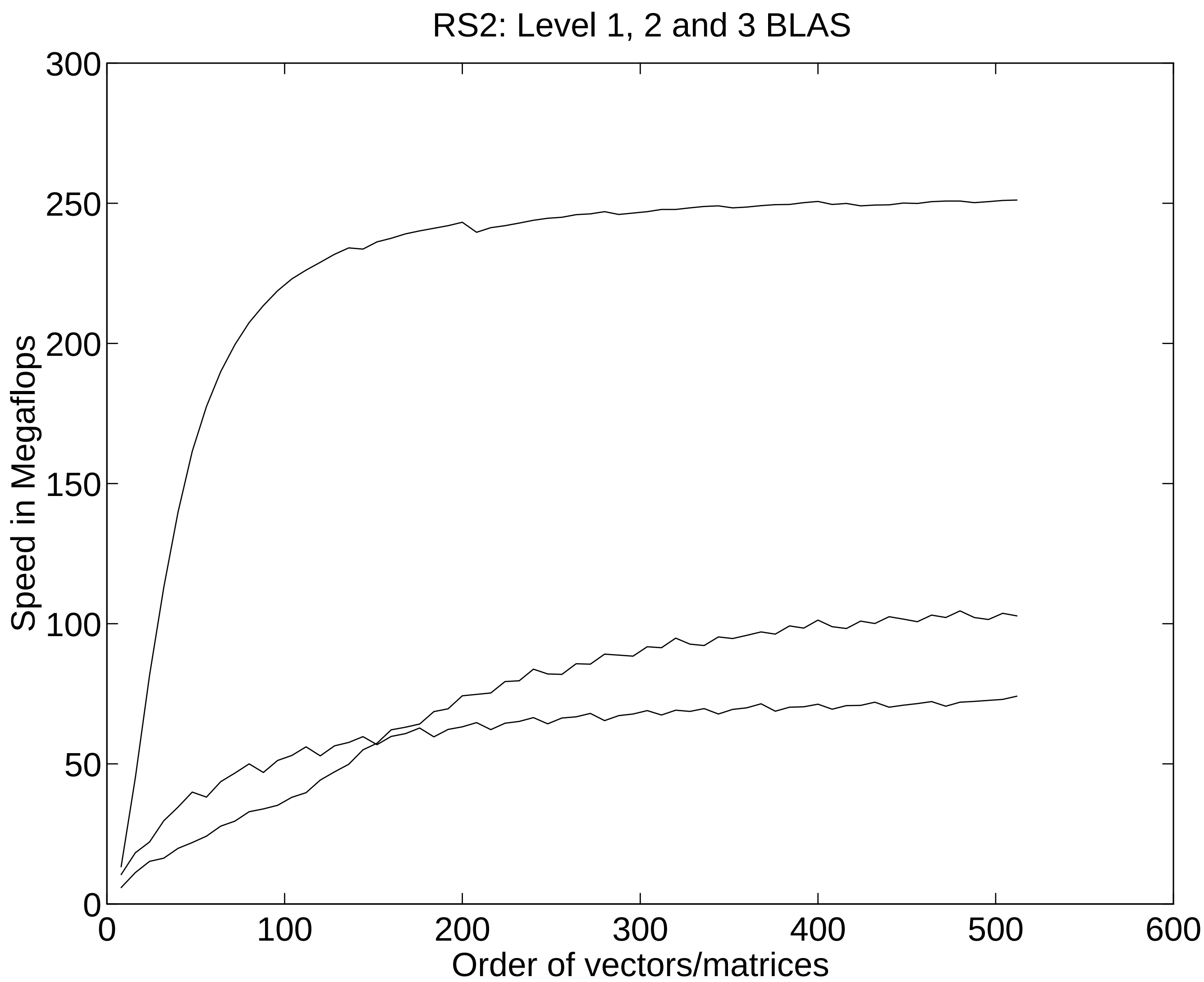


- **Can only do useful work on data at top**
- **Transfer between levels slow, so keep active data as close to top as possible**
- **BLAS:**
  BLAS1: $y \leftarrow \alpha x + y$ **(scalar$*$vector+vector)**
  BLAS2: $y \leftarrow Ax + y$ **(matrix$*$vector+vector)**
  BLAS3: $C \leftarrow AB{+}C$ **(matrix$*$matrix+matrix)**
- **How to tell which BLAS uses memory better**

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

- **BLAS3 best: most flops per memory reference**

BLAS on IBM RS 6000/590



LAPACK and ScaLAPACK

- **LAPACK**
  - **Targets workstations, vector machines, shared memory**
  - **Release 1.0 in Apr 1992, 2.0 in Oct 1994:**
  - **Dense and banded matrices**
  - **Linear systems, least squares, eigenproblems**
  - **Manual available from SIAM (2nd edition soon)**
- **ScaLAPACK**
  - **Targets distributed memory machines**
  - **Release 1.0β in Feb 1994, 2.0 in Nov 1994**
  - **Dense, band, sparse matrices**
  - **Less coverage than LAPACK**
  - **Direct and iterative methods**
  - **Software and templates**

New Features of LAPACK 2.0

- the generalized nonsymmetric eigenproblem $Ax = \lambda Bx$
- the generalized definite banded eigenproblem $Ax = \lambda Bx$
- two generalized linear least squares problems (least squares with linear constraints, and the Gauss-Markov linear model problem)
- the generalized (quotient) singular value decomposition
- faster routines for symmetric eigenproblems, based on divide-and-conquer
- a faster bidiagonal SVD (singular values only)
- reduction of a band matrix to bidiagonal form
- improved linear system drivers, to better reuse existing factorizations, and return the pivot growth factor
- the least squares drivers now return the residual root-sum-of-squares

Unblocked Gaussian Elimination

for   i = 1 to n-1

find k so that |A(k,i)| = $\max_{j >= i}$ |A(j,i)|

swap rows i and k of A

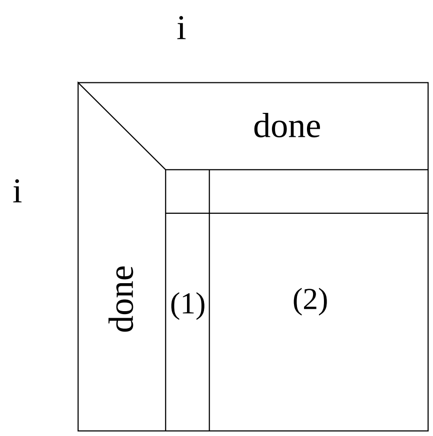
(1)   A(i+1 : n , i) = A(i+1 : n , i) / A(i , i)

... scale column i   (BLAS 1)

(2)   A(i+1 : n , i+1 : n) = A(i+1 : n , i+1 : n) +
A(i+1 : n , i) * A(i , i+1 : n)

... eliminate rows and columns i+1 to n   , rank-1 upddate   (BLAS 2)

end for



Blocked Gaussian Elimination

for i = 1 to n-1 step b

(1) use previous algorithm to factor A(i : n , i : i+b-1) = LU in place

... BLAS 1 and BLAS 2 ,

apply swaps to other columns of A

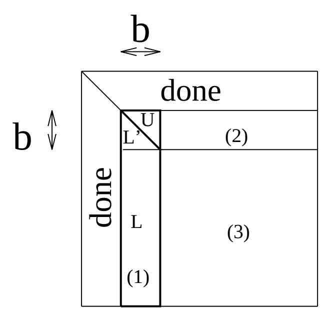
(2) A(i : i+b-1 , i+b : n) = L'^{-1} * A(i : i+b-1 , i+b : n)

... BLAS 3 (multiple triangular solve) to get submatrix of U

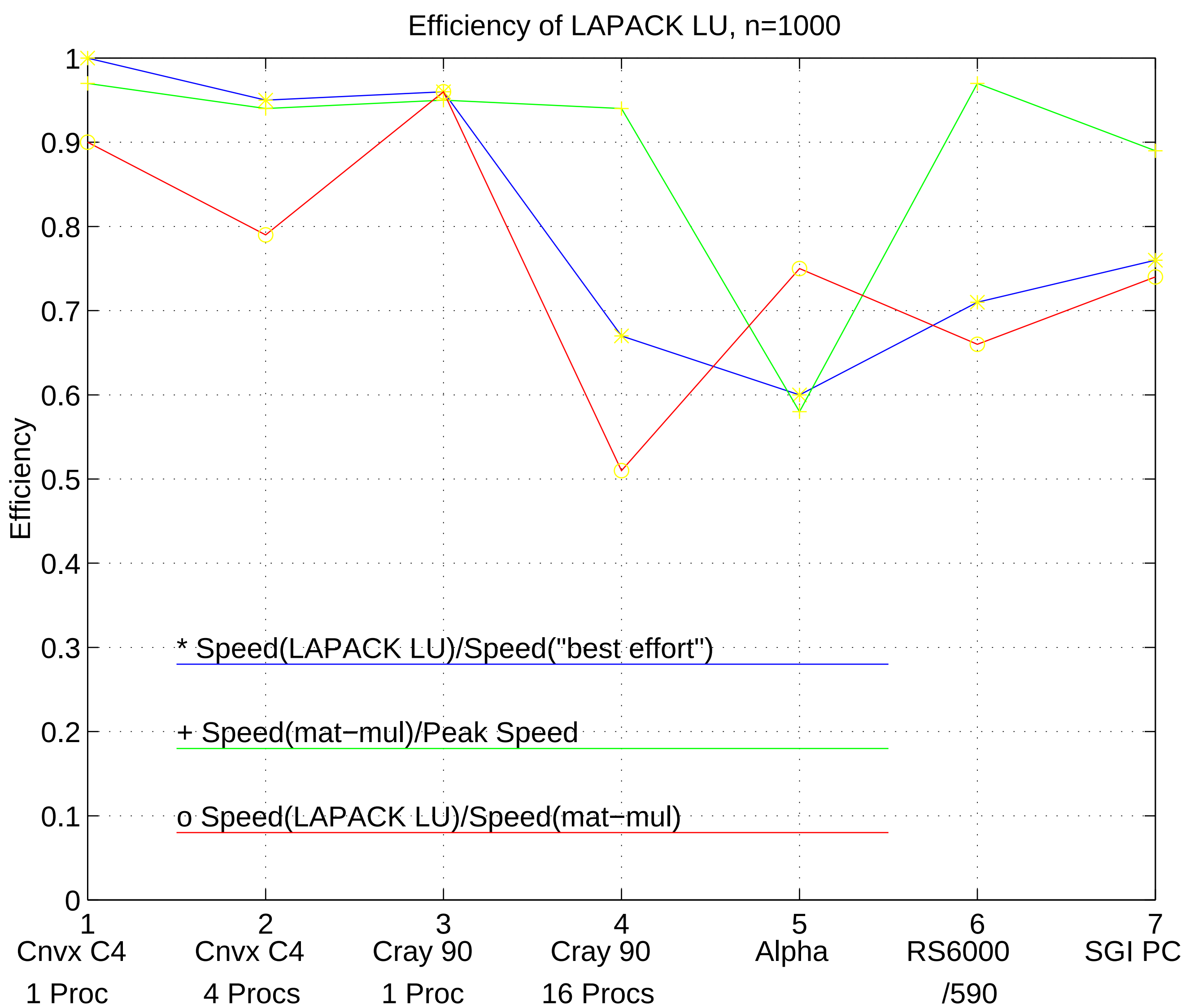
(3) A(i+b : n , i+b : n) = A(i+b : n , i+b : n) -
A(i+b : n , i : i+b-1) * A(i : i+b-1 , i+b : n)

... BLAS 3 (mat-mul) to eliminate trailing submatrix of A

end for

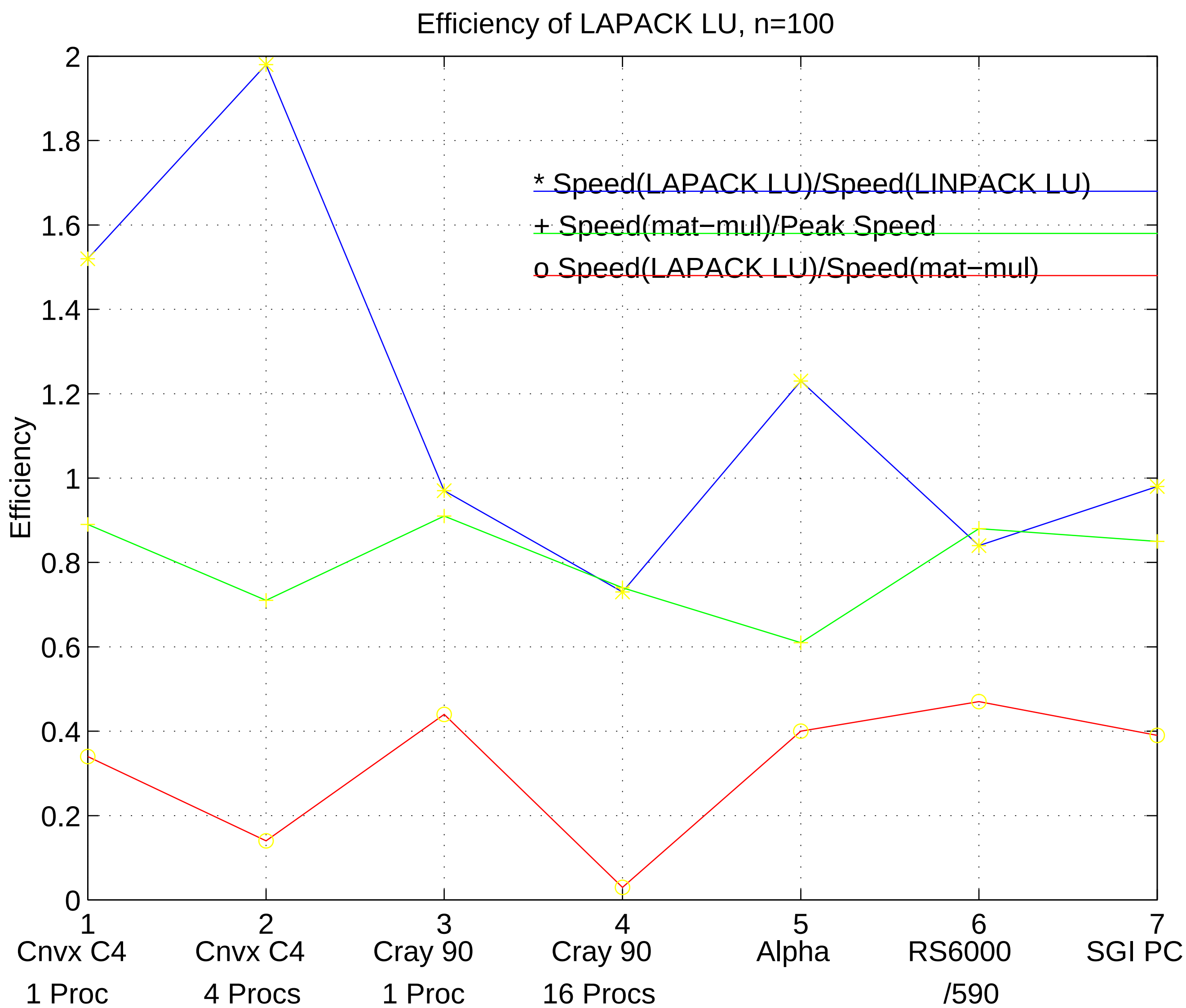


Efficiency of LAPACK LU, N=1000

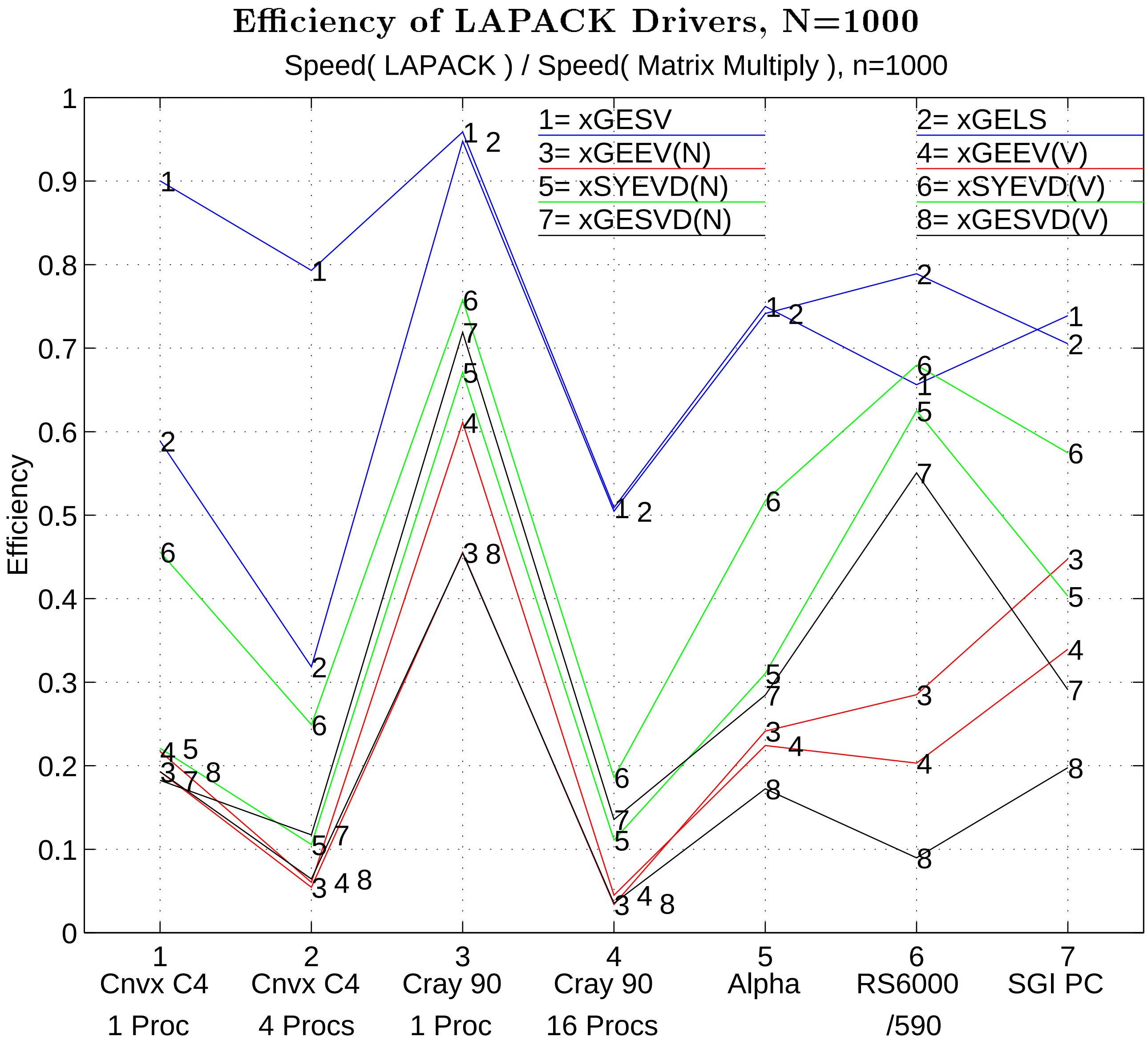


Machine	# Procs	Peak Mflops
Convex C4640	1	810
Convex C4640	4	3240
Cray C90	1	952
Cray C90	16	15238
DEC Alpha 3000-500X	1	200
IBM RS 6000/590	1	264
SGI Power Challenge	1	300

Efficiency of LAPACK LU, N=100

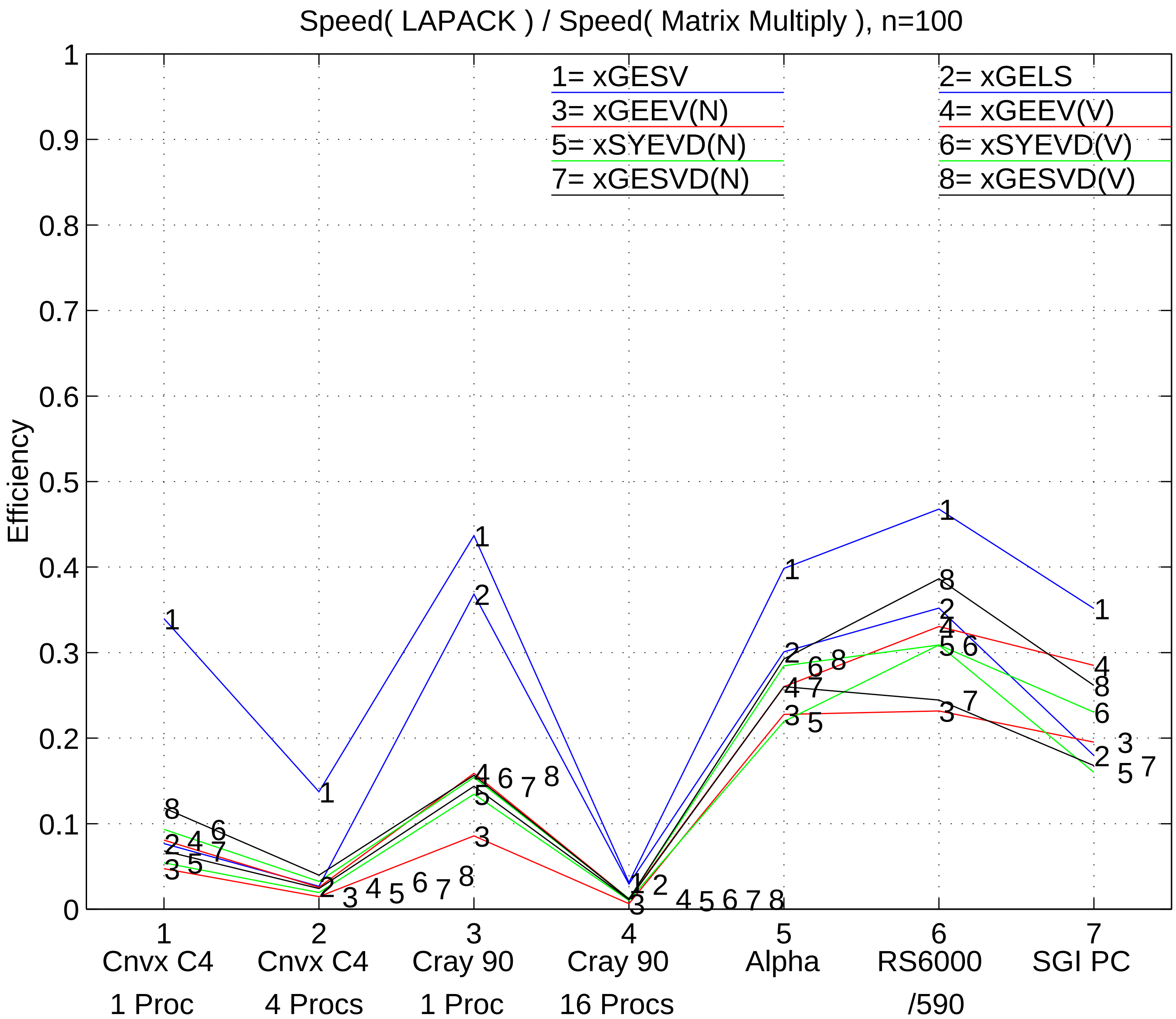


Machine	# Procs	Peak Mflops
Convex C4640	1	810
Convex C4640	4	3240
Cray C90	1	952
Cray C90	16	15238
DEC Alpha 3000-500X	1	200
IBM RS 6000/590	1	264
SGI Power Challenge	1	300



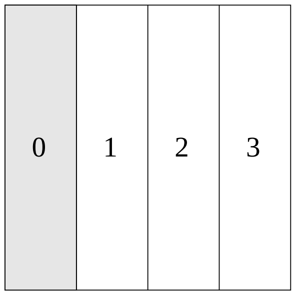
Label	Routine	Purpose
1	xGESV	Solve $Ax = b$
2	xGELS	Least squares $\min_x \|Ax - b\|_2$
3	xGEEV(N)	Eigenvalues of nonsymmetric A
4	xGEEV(V)	Eigenvalues and vectors of nonsymmetric A
5	xSYEVD(N)	Eigenvalues of symmetric A
6	xSYEVD(V)	Eigenvalues and vectors of symmetric A
7	xGESVD(N)	SVD (singular values only) of A
8	xGESVD(V)	SVD (singular values and vectors) of A

Efficiency of LAPACK Drivers, N=100

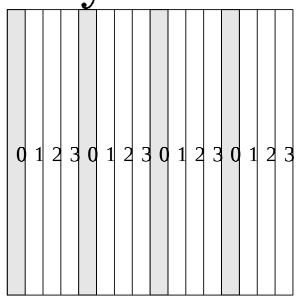


Label	Routine	Purpose
1	**xGESV**	**Solve** $Ax = b$
2	**xGELS**	**Least squares** $\min_x \|Ax - b\|_2$
3	**xGEEV(N)**	**Eigenvalues of nonsymmetric** A
4	**xGEEV(V)**	**Eigenvalues and vectors of nonsymmetric** A
5	**xSYEVD(N)**	**Eigenvalues of symmetric** A
6	**xSYEVD(V)**	**Eigenvalues and vectors of symmetric** A
7	**xGESVD(N)**	**SVD (singular values only) of** A
8	**xGESVD(V)**	**SVD (singular values and vectors) of** A

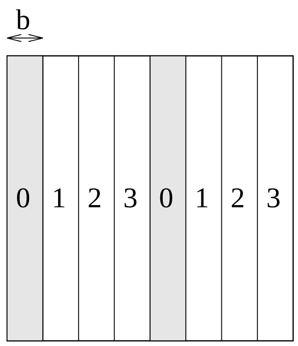
ScaLAPACK Data Layouts



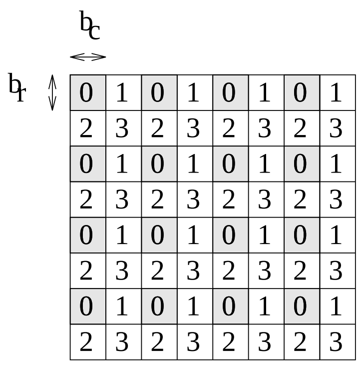
1) Column Blocked Layout



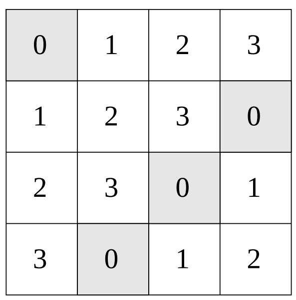
2) Column Cyclic Layout



3) Column Block Cyclic Layout

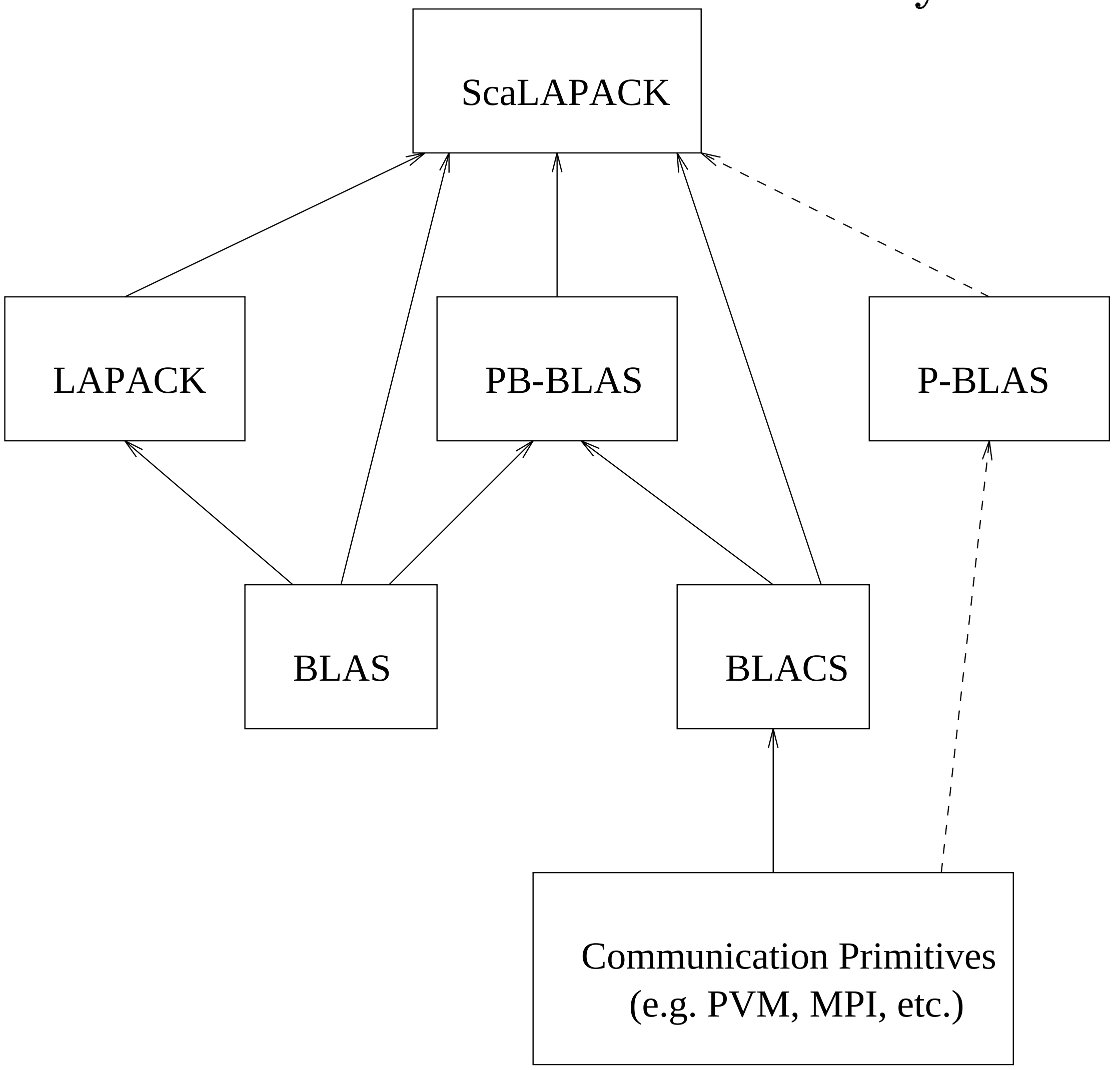


4) Row and Column Block Cyclic Layout

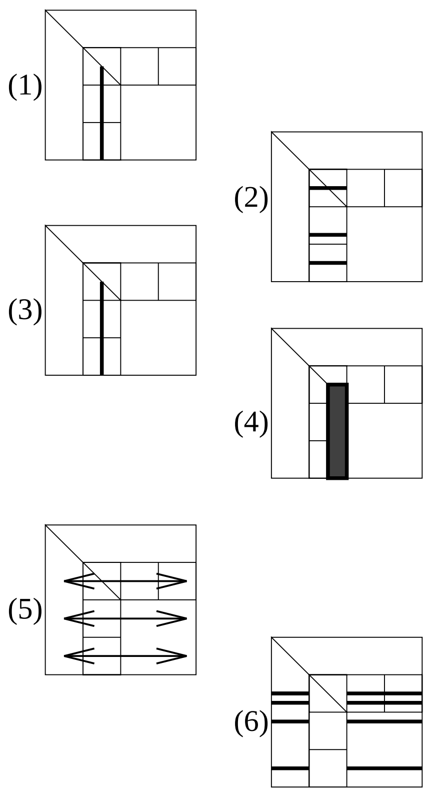


5) Block Skewed Layout

ScaLAPACK Software Hierarchy

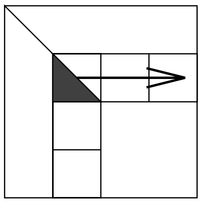


Distributed Gaussian Elimination (1)

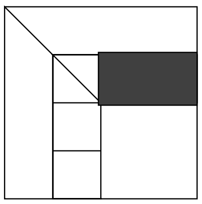



```
for  i  =  1  to  n-1  step  b

      for  j  =  i  to  i+b-1

      (1)     find  pivot  row  k,  column  broadcast

      (2)     swap  rows  k  and  j  in block  column,  broadcast  row  k

      (3)     A(  j+1:n  ,  j )  =  A( j+1:n  ,  j )  /  A( j  ,  j )

      (4)     A( j+1:n  ,  j+1:i+b-1) -= A( j+1:n , j ) * A( j , j+1:i+b-1 )

      end  for

(5)   broadcast  all  swap  information  right  and  left

(6)   apply  all  rows  swaps  to  other  columns
```

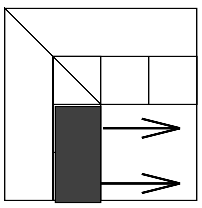
Distributed Gaussian Elimination (2)

(7) 

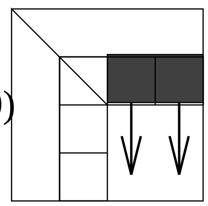
Broadcast L’ right

(8) 

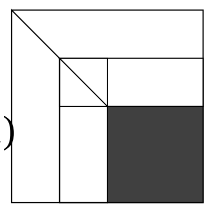
A(i:i+b-1 , i+b:n) = L'^{-1} * A(i:i+b-1 , i+b:n)

(9) 

Broadcast A(i+b:n , i:i+b-1) right

(10) 

Broadcast A(i:i+b-1 , i+b:n) down

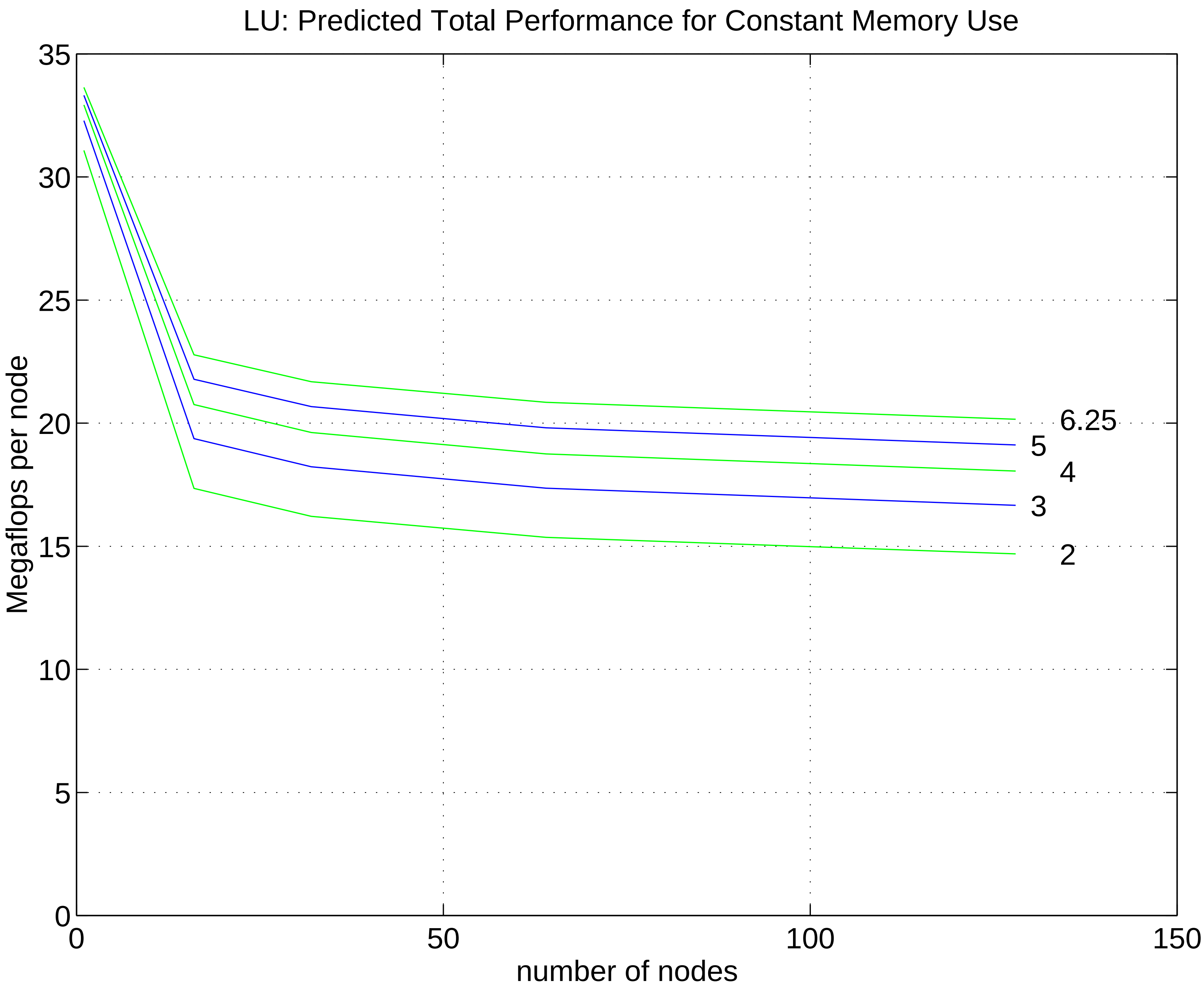
(11) 

Eliminate A(i+b:n , i+b:n)

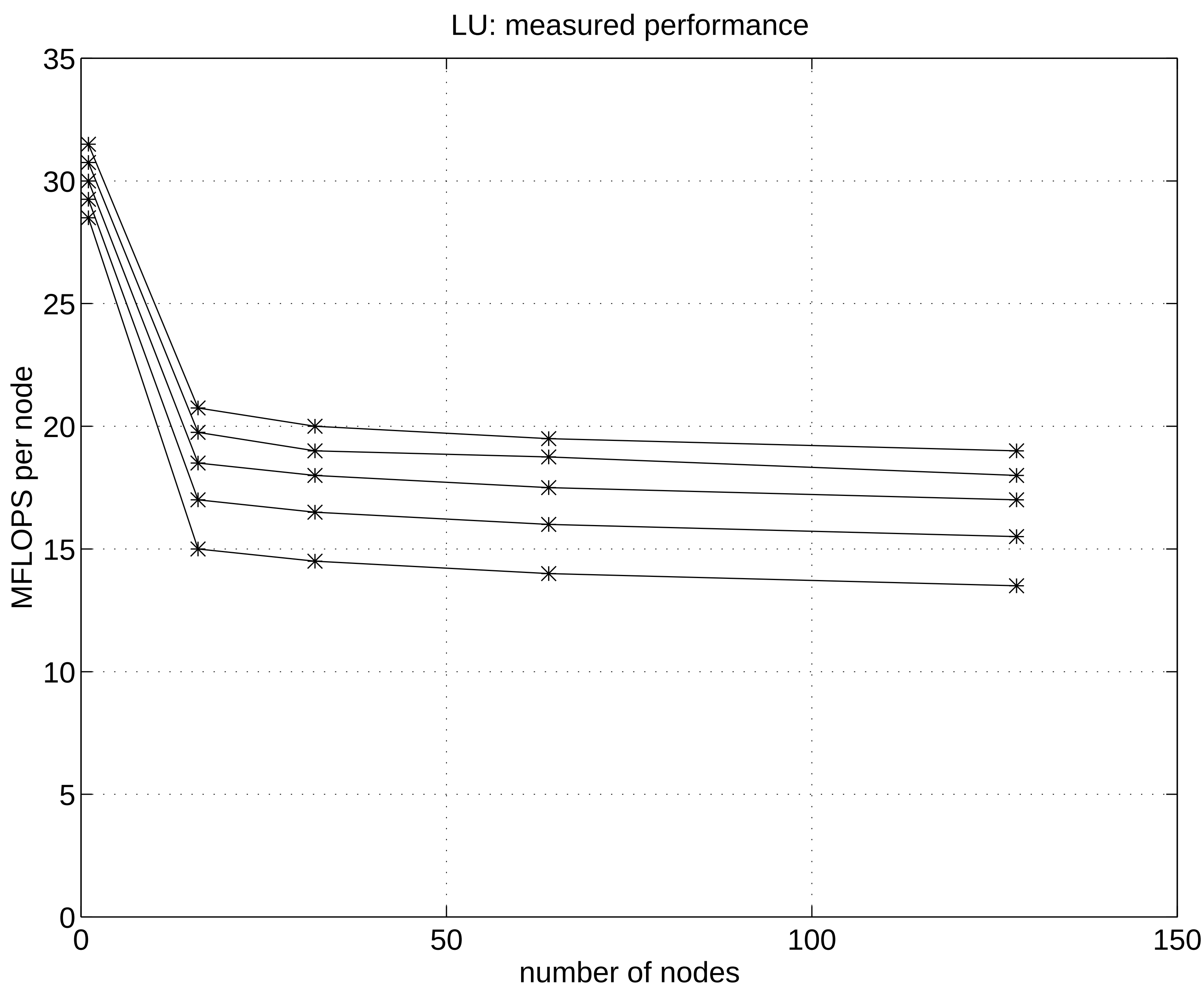
Cost Model of Distributed Gaussian Elimination

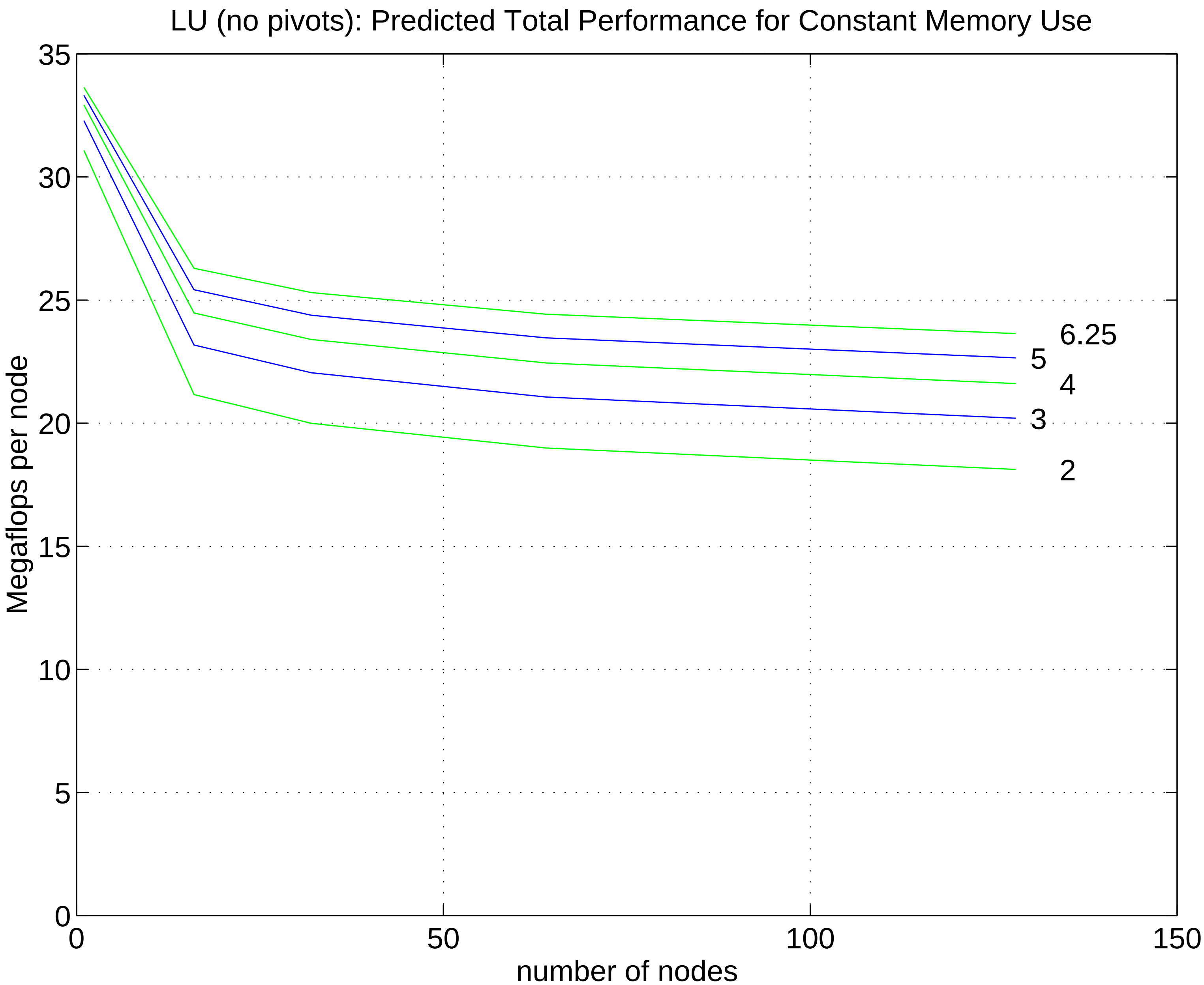
- **Last 3 steps most expensive**
- **Notation**
  - n = **matrix dimension**
  - α = **startup,** β = **per word send time**
  - γ = **time per flop (BLAS 3)**
  - p = **# processors, in** *nprow* **by** *npcol* **grid**
  - b = **blocksize**
- **Serial time =** $(2/3)n^3\gamma$
- **Step (9): Broadcast A(i+b:n , i:i+b-1) right**
  - **Use ring**
  - **time =** $npcol \cdot (\alpha + \beta b(n-i-b+1)/nprow)$
- **Step (10): Broadcast A(i:i+b-1 , i+b:n) down**
  - **Use tree**
  - **time =** $(\log_2 nprow)(\alpha + \beta b(n-i-b+1)/npcol)$
- **Step (11): Eliminate** $A(i+b:n, i+b:n)$
  - $2\gamma b(n-i-b+1)^2/p$
- **Efficiency = serial_time / (**p **· parallel_time)**

$$\approx \frac{1}{1+\frac{3\mathrm{p}\log_2 \mathrm{nprow}}{2\mathrm{n}^2}\cdot\frac{\alpha}{\gamma}+\frac{3}{4\mathrm{n}}(2\cdot\mathrm{npcol}+\mathrm{nprow}\cdot\log_2\mathrm{nprow})\frac{\beta}{\gamma}}$$



LU: Predicted Total Performance for Constant Memory Use
Megaflops per node
number of nodes
0
5
10
15
20
25
30
35
0
50
100
150
6.25
5
4
3
2


LU: measured performance
MFLOPS per node
number of nodes
0
5
10
15
20
25
30
35
0
50
100
150


LU (no pivots): Predicted Total Performance for Constant Memory Use
Megaflops per node
number of nodes
0
5
10
15
20
25
30
35
0
50
100
150
6.25
5
4
3
2

Sequential LU (LAPACK's DGETRF)

```
      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
```

PBLAS Parallel LU

```
      DO 10 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 PDGETF2( M-J+1, JB, A, IA+J-1, JA+J-1, ADESC, IPIV, INFO )

         Apply interchanges to columns 1:J-1.

         CALL PDLASWP( J-1, A, IA, JA, ADESC, IA+J-1, JA+J+JB-2, IPIV )

         IF( J+JB.LE.N ) THEN

            Apply interchanges to columns J+JB:N.

            CALL PDLASWP( N-J-JB+1, A, IA, JA+J+JB-1, ADESC, IA+J-1,
     $                    JA+J+JB-2, IPIV )

            Compute block row of U.

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

               Update trailing submatrix.

               CALL PDGEMM( 'No', 'No', M-J-JB+1, N-J-JB+1, JB, -ONE,
     $                      A, IA+J+JB-1, JA+J-1, ADESC, A, IA+J-1,
     $                      JA+J+JB-1, ADESC, ONE, A, IA+J+JB-1,
     $                      JA+J+JB-1, ADESC )
            END IF
         END IF
   10 CONTINUE
```

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

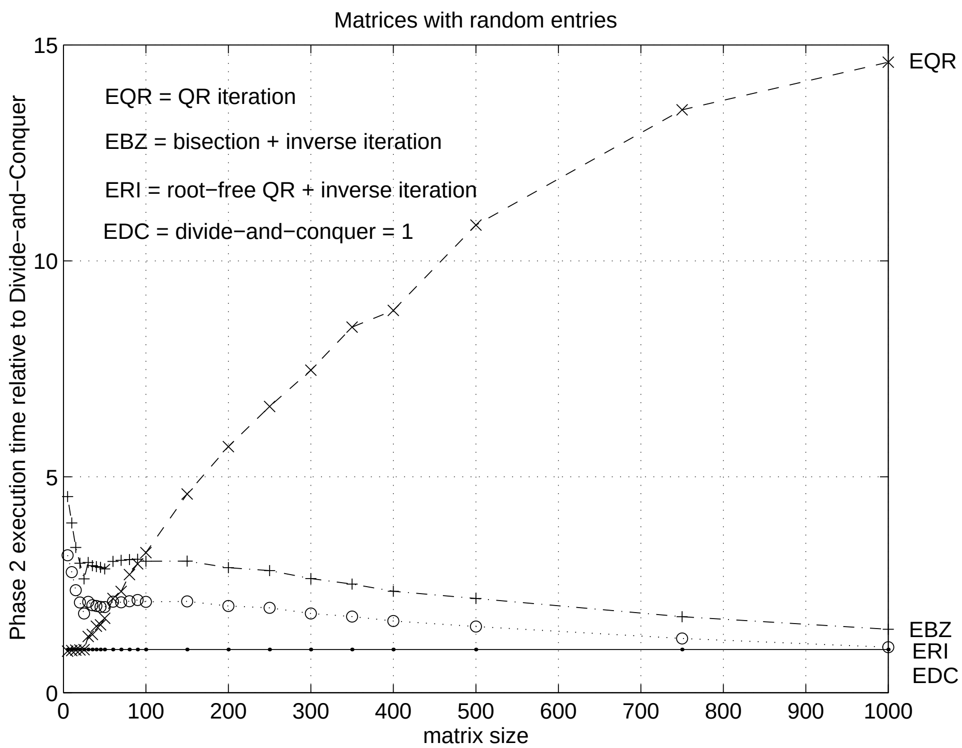
1. **Goal: compute** $T = Q\Lambda Q^T$, $\Lambda = \mathbf{diag}(\lambda_i)$, $QQ^T = I$
2. $$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix} = \left[\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right] + vv^T$$
3. **Solve** $T_1 = Q_1\Lambda_1 Q_1^T$ **and** $T_2 = Q_2\Lambda_2 Q_2^T$ **recursively**
4. $$T = \left[\begin{array}{c|c} Q_1\Lambda_1 Q_1^T & \\ \hline & Q_2\Lambda_2 Q_2^T \end{array}\right] + vv^T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot (D + zz^T) \cdot \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix}$$

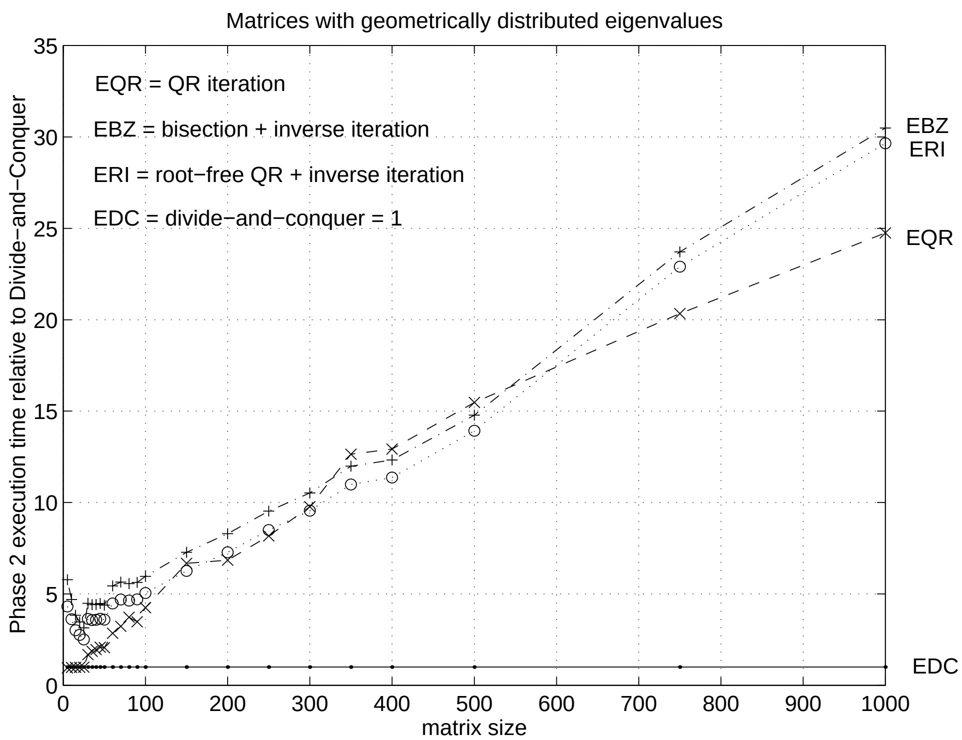
   where

   $$D = \begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} \text{ and } z = \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix} v$$
5. **Eigenvalues** λ **of** $D + zz^T$ **are roots of "secular equation"**

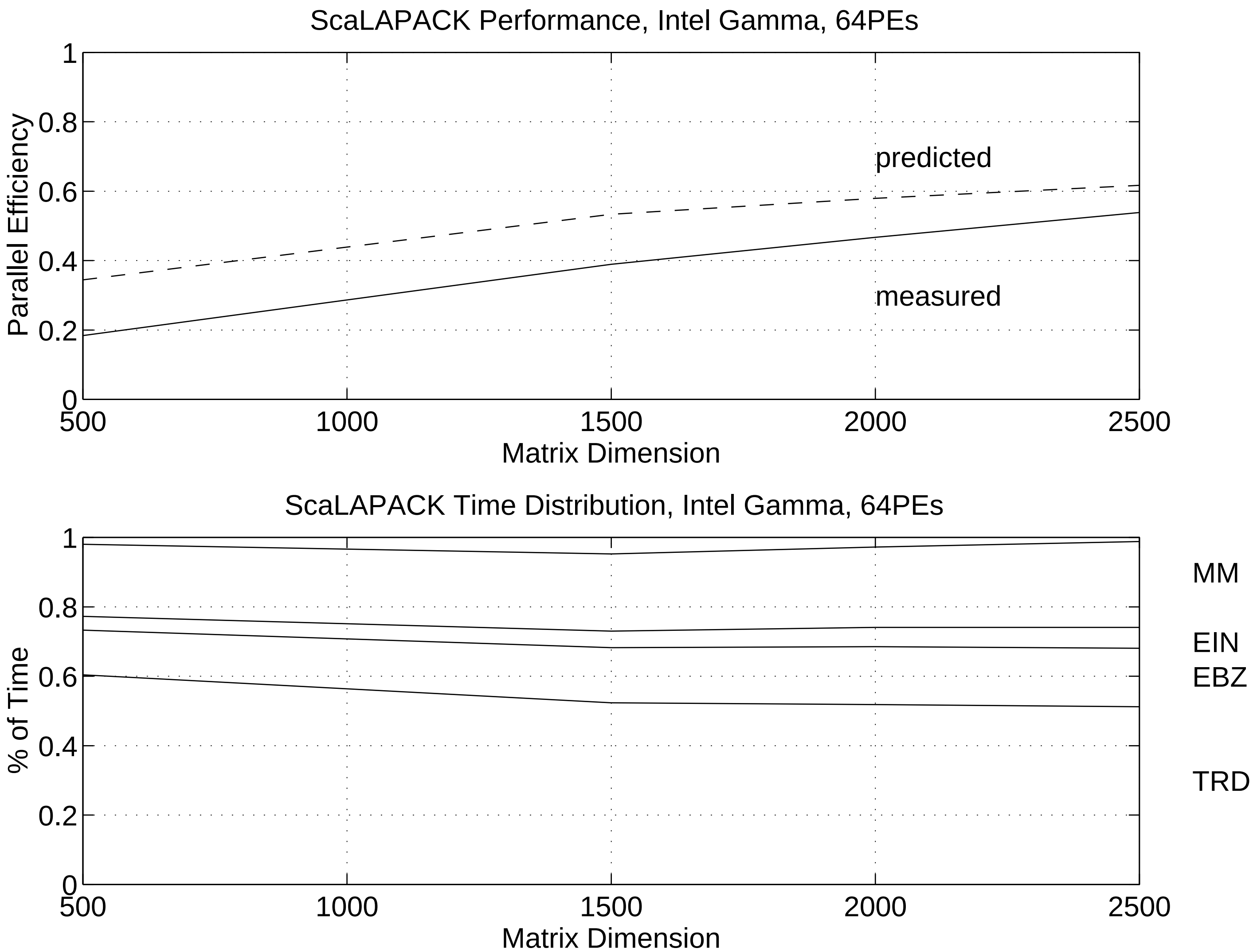
   $$1 + \sum_i \frac{z_i^2}{\lambda - d_{ii}} = 0$$
6. **Eigenvectors** v_j **of** $D + zz^T$ **are** $(D - \lambda_j I)^{-1} z$
7. **Eigenvectors of** T **are** $\begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \cdot V$

Speedups with Divide-and-Conquer





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



Future Design possibilities

- $O(n \log^p n)$ possible if vectors kept "factored"
  uses Cuppen + Fast Multipole Method
- Realistic goal: $O(n^2)$ embarrassingly parallel algorithm for Phase 2 with guaranteed accuracy
- Design possibilities
  - Use higher precision bisection + inverse iteration
    IEEE arithmetic makes simulating double-double fast
    Fused mul-add (RS 6000) even faster
  - Parallel Cuppen
    Mixed parallelism may not help
  - Hybrid Cuppen/inverse iteration
  - Naive algorithm done right
  - Spectral Divide-and-conquer

Trading off Speed and Stability

- **Many problems have two solutions:**
  slow, reliable
  fast, unstable in rare instances
- **Paradigm to achieve reliability and speed:**
  1. **Run fast, unreliable algorithm**
  2. **Test stability of result; if ok, done**
  3. **Otherwise, rerun using slow, reliable algorithm**
- **Drawback: instability may cause exceptions, abort**
- **Cure: Use exception handling in IEEE arithmetic**
  - **Can compute past exception with ∞, NAN, setting sticky flags to examine later**
  - **Can trap to user-supplied trap handler**

Speedup using IEEE Arithmetic

- Condition estimation using Hager's method
  Fast: use BLAS2 to solve triangular system
  Slow: test, scale in inner loop

Machine	Dimension 100	300	500
DEC 5000	2.0	1.5	1.4
SUN 4/260	2.0	1.9	1.6
DEC Alpha	1.7	1.9	1.7
Cray C90 (non IEEE)	4.2	3.0	2.6

- Barriers to using IEEE Arithmetic
  - No standard way to refer to exceptions
    ISO LIA standard: flawed attempt
  - Not all IEEE implementations equally fast, so can't be sure fast algorithm will be fast

Machine	Time(special arithmetic) /Time(standard arithmetic) denorm	∞	NAN
Sun4/260 (Weitek1164/5)	8	1	10
SuperSPARC (Viking)	1	1	1
PA-RISC	68	1	42
RS 6000	1	1	1
387, 486, Pentium	1	1	1
i860	868	432	411
DEC Alpha	690	343	457
Cray C90 (non IEEE)	N/A	abort	abort

Software Engineering in LAPACK

- **793K lines of Fortran 77**
  - **Source: 303K**
  - **Testing (including BLAS): 313K**
  - **Timing: 147K**
  - **BLAS: 30K (should use optimized BLAS!)**
- **Source code standardized using Toolpack**
- **Double and Complex*16 produced automatically from Single and Complex**
- **"Mail-order portability"**
  - **Source code runs anywhere**
  - **Machine parameters computed at run-time**
  - **Speed from machine dependent BLAS, block sizes**
- **Half of 341 page manual produced automatically from code**
- **1M line C version produced mostly automatically using f2c**

ScaLAPACK Sparse matrix activities

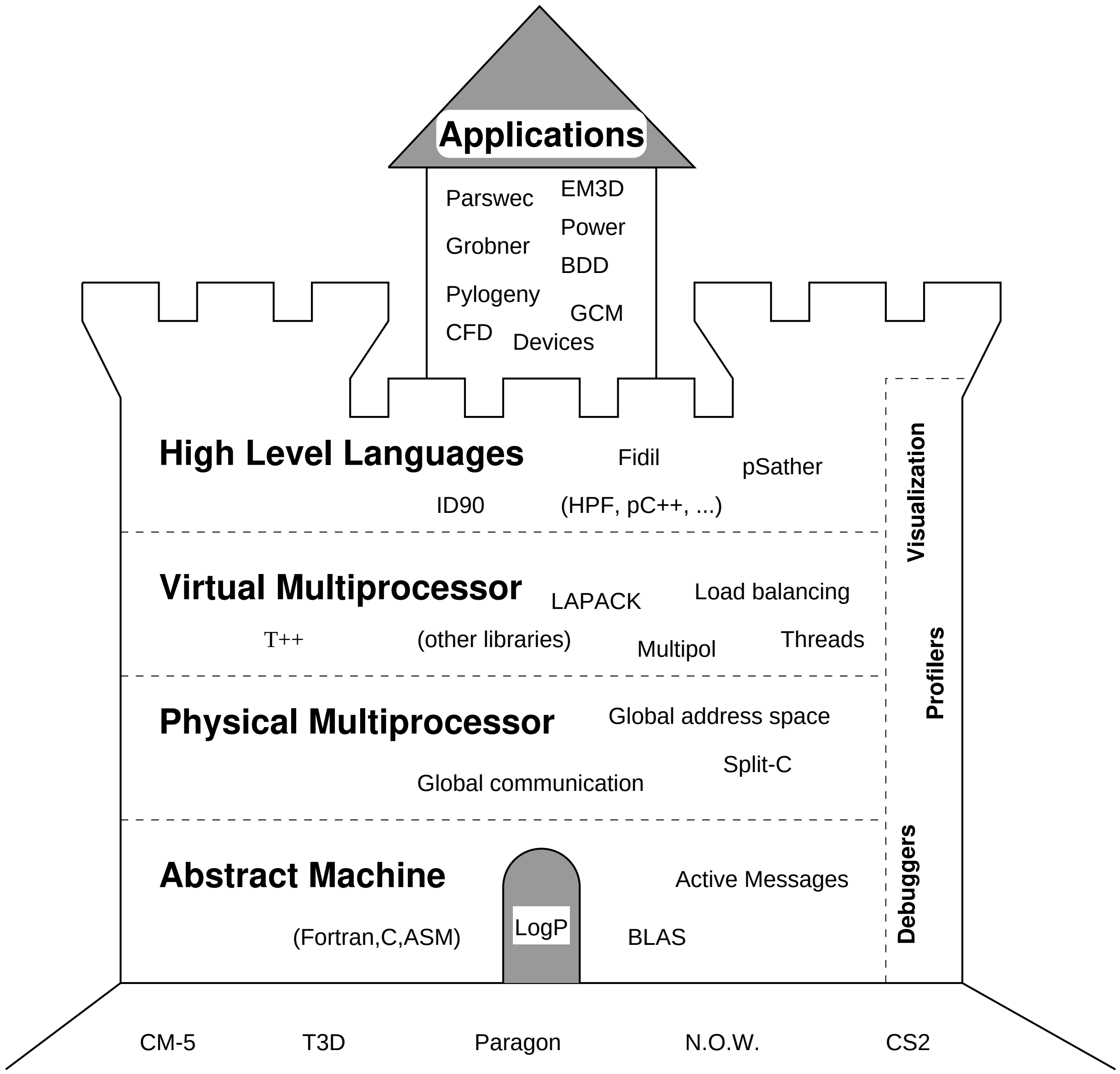
- Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods
  - "Recipes" (done right!) for which iterative scheme to use when
  - Software on-line (matlab, fortran)
  - Book on-line, or from SIAM
  - Templates for eigenproblems planned
- ARPACK - Arnoldi-Pack - D. Sorensen
  - 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
  - Fastest parallel Cholesky available
- Sparse LU with Pivoting

Castle

Faculty:

David Culler
James Demmel
Katherine Yelick

Thomas Anderson
Phillip Colella
Jerome Feldman
Susan Graham
Paul Hilfinger
Richard Karp
Abhiram Ranade
David Patterson
A. Sangiovanni-Vincentelli
James Sethian



Where to get Software and Reports

- **Accessible via Netlib – xnetlib/{file}**
  Mosaic – http://www.netlib.org/{file}
- **LAPACK manual – SIAM or Mosaic:**
  http://www.netlib.org/lapack/lug/lapack_lug.html
- **LAPACK in Fortran – lapack**
- **LAPACK in C – clapack (soon!)**
- **LAPACK in C++ – lapack (soon!)**
- **LAPACK manual – lapack (soon!) or SIAM**
- **Tech reports – lapack/lawns**
- **ScaLAPACK software – scalapack**
- **Templates book – linalg/templates.ps**
- **Templates Matlab – linalg/mltemplates.ps**
- **Templates Fortran – linalg/sltemplates.ps or**
  linalg/dltemplates.ps

- **CSEP - on-line text book for scientific computing (many contributors; we contributed chapter on linear algebra) - available via Mosaic, http://compsci.cas.vanderbilt.edu/csep.html**