Lecture 13:
Linear Algebra Algorithms

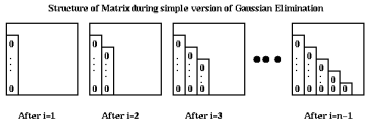
Jack Dongarra, U of Tennessee

Slides are adapted from Jim Demmel, UCB's Lecture on Linear Algebra Algorithms

Review of Gaussian Elimination (GE) for solving Ax=b

- Add multiples of each row to later rows to make A upper triangular
- Solve resulting triangular system Ux = c by substitution

```
... for each column i
... zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    ... for each row j below row i
    for j = i+1 to n
        ... add a multiple of row i to row j
        for k = i to n
            A(j,k) = A(j,k) - (A(j,i)/A(i,i)) * A(i,k)
```



Refine GE Algorithm (1)

- Initial Version

```
... for each column i
... zero it out below the diagonal by adding multiples of row i to later rows
for i = 1 to n-1
    ... for each row j below row i
    for j = i+1 to n
        ... add a multiple of row i to row j
        for k = i to n
            A(j,k) = A(j,k) - (A(j,i)/A(i,i)) * A(i,k)
```

- Remove computation of constant A(j,i)/A(i,i) from inner loop

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```


Refine GE Algorithm (2)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Don't compute what we already know: zeros below diagonal in column i

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```


Refine GE Algorithm (3)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        m = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - m * A(i,k)
```

- Store multipliers m below diagonal in zeroed entries for later use

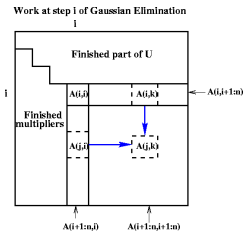
```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```


Refine GE Algorithm (4)

- Last version

```
for i = 1 to n-1
    for j = i+1 to n
        A(j,i) = A(j,i)/A(i,i)
        for k = i+1 to n
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- Express using matrix operations (BLAS)




```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
        - A(i+1:n , i) * A(i , i+1:n)
```

What GE really computes

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

° Call the strictly lower triangular matrix of multipliers M, and let L = I+M

° Call the upper triangle of the final matrix U

° *Lemma (LU Factorization):* If the above algorithm terminates (does not divide by zero) then A = L*U

° Solving A*x=b using GE
- Factorize A = L*U using GE (cost = 2/3 n^3 flops)
- Solve L*y = b for y, using substitution (cost = n^2 flops)
- Solve U*x = y for x, using substitution (cost = n^2 flops)

° Thus A*x = (L*U)*x = L*(U*x) = L*y = b as desired

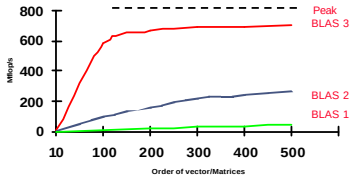
Problems with basic GE algorithm

° What if some A(i,i) is zero? Or very small?
- Result may not exist, or be "unstable", so need to pivot

° Current computation all BLAS 1 or BLAS 2, but we know that BLAS 3 (matrix multiply) is fastest (Lecture 2)

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          ... BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )  ... BLAS 2 (rank-1 update)
        - A(i+1:n , i) * A(i , i+1:n)
```

IBM RS/6000 Power 3 (200 MHz, 800 Mflop/s Peak)



Pivoting in Gaussian Elimination

° A = [0 1] fails completely, even though A is "easy"
[1 0]

° Illustrate problems in 3-decimal digit arithmetic:

A = [1e-4 1] and b = [1], correct answer to 3 places is x = [1]
[1 1] [2] [1]

° Result of LU decomposition is

L = [1 0] = [1 0] ... No roundoff error yet
[fl(1/1e-4) 1] [1e4 1]

U = [1e-4 1] = [1e-4 1] ... Error in 4th decimal place
[0 fl(1-1e4*1)] [0 -1e4]

Check if A = L*U = [1e-4 1] ... (2,2) entry entirely wrong
[1 0]

° Algorithm "forgets" (2,2) entry, gets same L and U for all |A(2 ,2)|<5
- ° Numerical instability
- ° Computed solution x totally inaccurate

° Cure: Pivot (swap rows of A) so entries of L and U bounded

Gaussian Elimination with Partial Pivoting (GEPP)

° Partial Pivoting: swap rows so that each multiplier
|L(i,j)| = |A(j,i)/A(i,i)| <= 1

```
for i = 1 to n-1
    find and record k where |A(k,i)| = max{i <= j <= n} |A(j,i)|
        ... i.e. largest entry in rest of column i
    if |A(k,i)| = 0
        exit with a warning that A is singular, or nearly so
    elseif k != i
        swap rows i and k of A
    end if
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          ... each quotient lies in [ -1,1]
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

° *Lemma*: This algorithm computes A = P*L*U, where P is a permutation matrix

° Since each entry of |L(i,j)| <= 1, this algorithm is considered numerically stable

° For details see LAPACK code at www.netlib.org/lapack/single/sgetf2.f

History of Block Partitioned Algorithms

° Early algorithms involved use of small main memory using tapes as secondary storage.

° Recent work centers on use of vector registers, level 1 and 2 cache, main memory, and "out of core" memory.

Blocked Partitioned Algorithms

° LU Factorization

° Cholesky factorization

° Symmetric indefinite factorization

° Matrix inversion

° QR, QL, RQ, LQ factorizations

° Form Q or Q^TC

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

° Block QR iteration for nonsymmetric eigenvalue problems

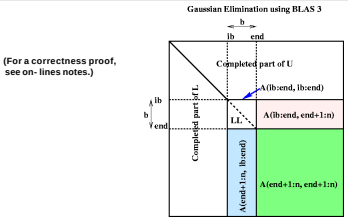
Converting BLAS2 to BLAS3 in GEPP

- **Blocking**
  - Used to optimize matrix-multiplication
  - Harder here because of data dependencies in GEPP
- **Delayed Updates**
  - Save updates to "trailing matrix" from several consecutive BLAS2 updates
  - Apply many saved updates simultaneously in one BLAS3 operation
- **Same idea works for much of dense linear algebra**
  - Open questions remain
- **Need to choose a block size b**
  - Algorithm will save and apply b updates
  - b must be small enough so that active submatrix consisting of b columns of A fits in cache
  - b must be large enough to make BLAS3 fast

Blocked GEPP (www.netlib.org/lapack/single/sgetrf.f)

```
for   ib = 1 to n-1 step b     ... Process matrix b columns at a time
      end = ib + b-1           ... Point to end of block of b columns
      apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P' * L' * U'
      ... let LL denote the strict lower triangular part of  A(ib:end , ib:end) + I
      A(ib:end , end+1:n) = LL-1 * A(ib:end , end+1:n)        ... update next b rows of U
      A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
                - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                              ... apply delayed updates with single matrix-multiply
                              ... with inner dimension b
```

(For a correctness proof, see on- lines notes.)



LAPACK

- **Linear Algebra library in Fortran 77**
  - Solution of systems of equations
  - Solution of eigenvalue problems
- **Combine algorithms from LINPACK and EISPACK into a single package**
- **Efficient on a wide range of computers**
  - RISC, Vector, SMPs
- **User interface similar to LINPACK**
  - Single, Double, Complex, Double Complex
- **Built on the Level 1, 2, and 3 BLAS**

Derivation of Blocked Algorithms
Cholesky Factorization $A = U^T U$

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

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

$$a_j = U_{11}^T u_j$$
$$a_{jj} = u_j^T u_j + u_{jj}^2$$

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

$$U_{11}^T u_j = a_j$$
$$u_{jj}^2 = a_{jj} - u_j^T u_j$$

LINPACK Implementation

- **Here is the body of the LINPACK routine SPOFA which implements the method:**

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


LAPACK Implementation

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

- This change by itself is sufficient to significantly improve the performance on a number of machines.
- From 238 to 312 Mflop/s for a matrix of order 500 on a Pentium 4-1.7 GHz.
- However on peak is 1,700 Mflop/s.
- Suggest further work needed.

Derivation of Blocked Algorithms

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

Equating coefficient of second block of columns, we obtain

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

$$A_{22} = U_{12}^T U_{12} + U_{22}^T U_{22}$$

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

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

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

LAPACK Blocked Algorithms

```
DO 10 J = 1, N, NB
   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 Pentium 4, L3 BLAS squeezes a lot more out of 1 proc

Intel Pentium 4 1.7 GHz N = 500	Rate of Execution
Linpack variant (L1B)	238 Mflop/s
Level 2 BLAS Variant	312 Mflop/s
Level 3 BLAS Variant	1262 Mflop/s

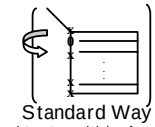
LAPACK Contents

- **Combines algorithms from LINPACK and EISPACK into a single package. User interface similar to LINPACK.**
- **Built on the Level 1, 2 and 3 BLAS, for high performance (manufacturers optimize BLAS)**
- **LAPACK does not provide routines for structured problems or general sparse matrices (i.e sparse storage formats such as compressed -row, -column, -diagonal, skyline ...).**

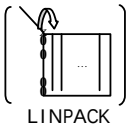
LAPACK Ongoing Work

- **Add functionality**
  - updating/downdating , divide and conquer least squares,bidiagonal bisection, bidiago nal inverse iteration, band SVD, Jacobi methods, ...
- **Move to new generation of high performance machines**
  - IBM SPs , CRAY T3E, SGI Origin, clusters of workstations
- **New challenges**
  - New languages: FORTRAN 90, HP FORTRAN, ...
  - (CMMD, MPL, NX ...)
    - many flavors of message passing, need standard (PVM, MPI): BLACS
- **Highly varying ratio** $\frac{\text{Computational speed}}{\text{Communication speed}}$
- **Many ways to layout data,**
- **Fastest parallel algorithm sometimes less stable numerically.**

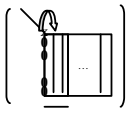
Gaussian Elimination



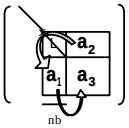
Standard Way
subtract a multiple of a row



LINPACK
apply sequence to a column



nb



nb

$a_2 = L^{-1} a_2$

$a_3 = a_3 - a_1 * a_2$

LAPACK
apply sequence to nb then apply nb to rest of matrix

Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

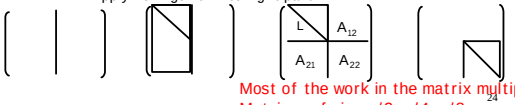
LU Algorithm:

1: Split matrix into two rectangles (m x n/2)
if only 1 column, scale by reciprocal of pivot & return

2: Apply LU Algorithm to the left part

3: Apply transformations to right part
(triangular solve $A_{12} = L^{-1}A_{12}$ and
matrix multiplication $A_{22} = A_{22} - A_{21} * A_{12}$)

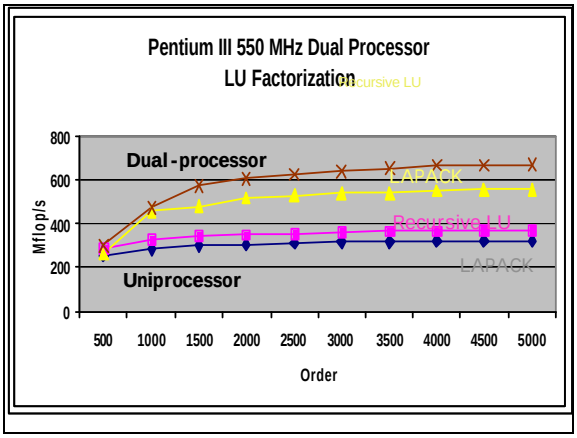
4: Apply LU Algorithm to right part



Most of the work in the matrix multiply
Matrices of size n/2, n/4, n/8, ...

Recursive Factorizations

- Just as accurate as conventional method
- Same number of operations
- Automatic variable blocking
  - Level 1 and 3 BLAS only !
- Extreme clarity and simplicity of expression
- Highly efficient
- The recursive formulation is just a rearrangement of the point-wise LINPACK algorithm
- The standard error analysis applies (assuming the matrix operations are computed the "conventional" way).
- OK for LU, LL^T, & QR
  - Open question on 2-sided algs. eg eigenvalue reduction



Dense recursive factorization

- The algorithm:

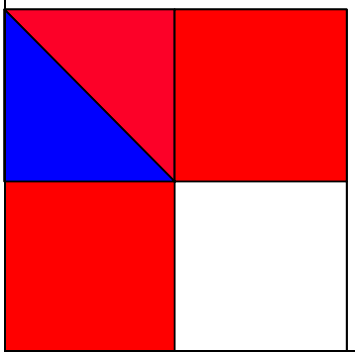
function rlu(A)
begin

$rlu(A_{11})$;	recursive call
$A_{21} \leftarrow A_{21} \cdot U^{-1}(A_{11})$;	xTRSM () on upper triangular submatrix
$A_{12} \leftarrow L_1^{-1}(A_{11}) \cdot A_{12}$;	xTRSM () on lower triangular submatrix
$A_{22} \leftarrow A_{22} - A_{21} \cdot A_{12}$;	xGEMM ()
$rlu(A_{22})$;	recursive call

end.

- Replace xTRSM and xGEMM with sparse implementations that are themselves recursive

Recursive LU Factorization



function RLU(A)
begin

$RLU(A_{11})$

$A_{21} := A_{21} \mathbf{U}^{-1}(A_{11})$ DTRSM()

$A_{12} := \mathbf{L}_1^{-1}(A_{11})\, A_{12}$ DTRSM()

$A_{22} := A_{22} - A_{21} A_{12}$ DGEMM()

$RLU(A_{22})$

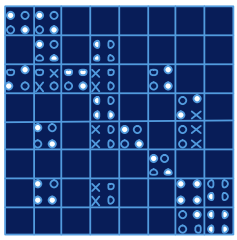
Sparse Factorization Assumptions

- **Sparse recursive LU factorization**
  - Based on recursive formulation of LU factorization
  - No partial pivoting during factorization
    - Diagonal zeros replaced with small elements, eg. $\epsilon\|A\|$
    - Iterative refinement used to regain precision
  - Locate dense blocks, performance comes from the use of BLAS Level 3
  - Aimed at improving time to solution – memory usage may suffer

Sparse Recursive Factorization Algorithm

- **Solutions - continued**
  - fast sparse xGEMM() is two-level algorithm
    - recursive operation on sparse data structures
    - dense xGEMM() call when recursion reaches single block
- **Uses Reverse Cuthill-McKee ordering causing fill-in around the band**
- **No partial pivoting**
  - use iterative improvement or
  - pivot only within blocks

2. Symbolic Factorization
3. Search for Dense blocks



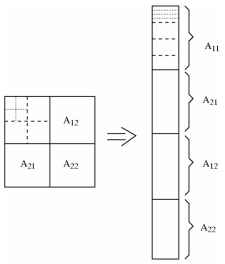
- original nonzero value
  zero value introduced due to fill-in
  zero value introduced due to blocking

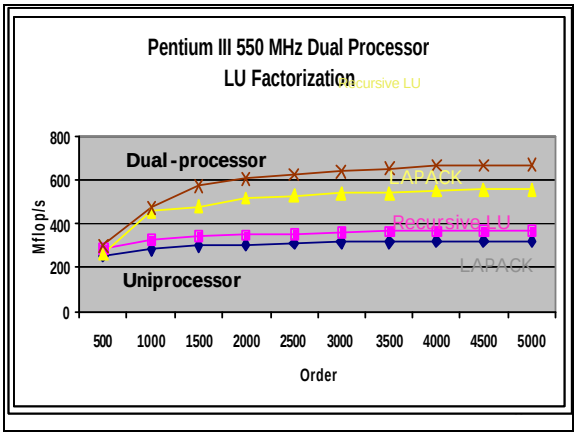
31

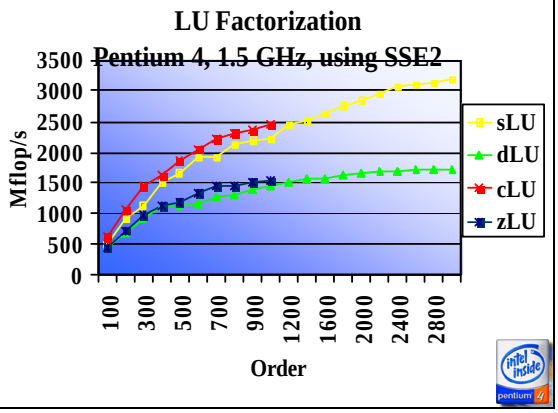
Recursive Factorization Applied to Sparse Direct Methods

1. Symbolic Factorization
2. Search for blocks that contain non-zeros
3. Conversion to sparse recursive storage
4. Search for embedded blocks
5. Numerical factorization

Layout of sparse recursive matrix in storage follows recursion







Challenges in Developing Distributed Memory Libraries

- How to integrate software?
  - Until recently no standards
  - Many parallel languages
  - Various parallel programming models
  - Assumptions about the parallel environment
    - granularity
    - topology
    - overlapping of communication/computation
    - development tools
- Where is the data
  - Who owns it?
  - Opt data distribution
- Who determines data layout
  - Determined by user?
  - Determined by library developer?
  - Allow dynamic data dist.
  - Load balancing

35

ScaLAPACK

- Library of software dealing with dense & banded routines
- Distributed Memory - Message Passing
- MIMD Computers and Networks of Workstations
- Clusters of SMPs

36

Programming Style

- SPMD Fortran 77 with object based design
- Built on various modules
  - PBLAS Interprocessor communication
  - BLACS
    - PVM, MPI, IBM SP, CRI T3, Intel, TMC
    - Provides right level of notation.
  - BLAS
- LAPACK software expertise/quality
  - Software approach
  - Numerical methods

Overall Structure of Software

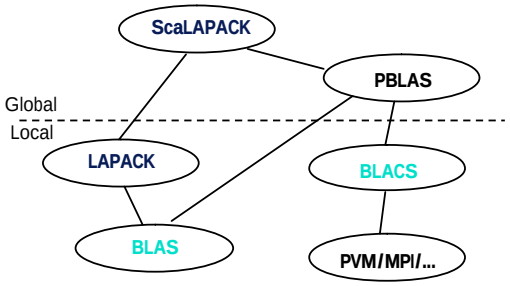
- Object based - Array descriptor
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out -of-core
- Using the concept of context

PBLAS

- Similar to the BLAS in functionality and naming.
- Built on the BLAS and BLACS
- Provide global view of matrix

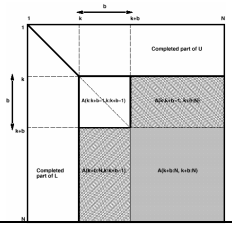
```
CALL DGEXXX   ( M, N, A( IA, JA ), LDA,... )

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

ScaLAPACK Structure



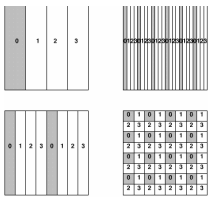
Choosing a Data Distribution

- Main issues are:
  - Load balancing
  - Use of the Level 3 BLAS



Possible Data Layouts

- 1D block and cyclic column distributions



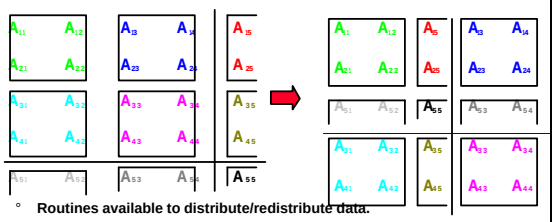
- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

Distribution and Storage

- Matrix is block-partitioned & maps blocks
- Distributed 2-D block- cyclic scheme

5x5 matrix partitioned in 2x2 blocks

2x2 process grid point of view



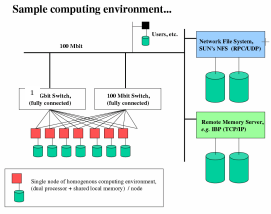
- Routines available to distribute/redistribute data.

Parallelism in ScaLAPACK

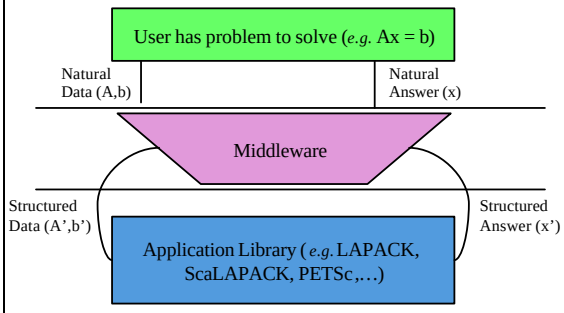
- Level 3 BLAS block operations
  - All the reduction routines
- Pipelining
  - QR Algorithm, Triangular Solvers, classic factorizations
- Redundant computations
  - Condition estimators
- Static work assignment
  - Bisection
- Task parallelism
  - Sign function eigenvalue computations
- Divide and Conquer
  - Tridiagonal and band solvers, symmetric eigenvalue problem and Sign function
- Cyclic reduction
  - Reduced system in the band solver
- Data parallelism
  - Sign function

LAPACK For Clusters

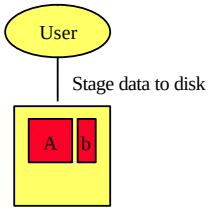
- Developing middleware which couples cluster system information with the specifics of a user problem to launch cluster based applications on the "best" set of resource available.
- Using ScaLAP



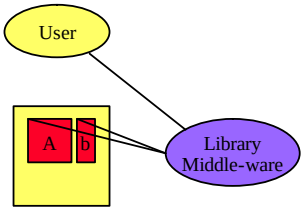
Big Picture…



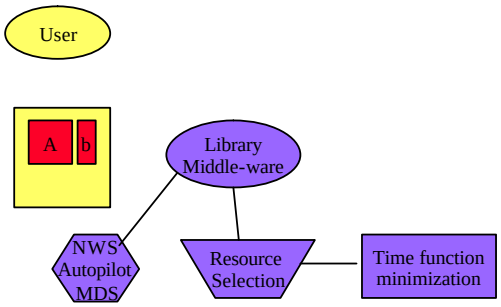
Numerical Libraries for Clusters



Numerical Libraries for Clusters

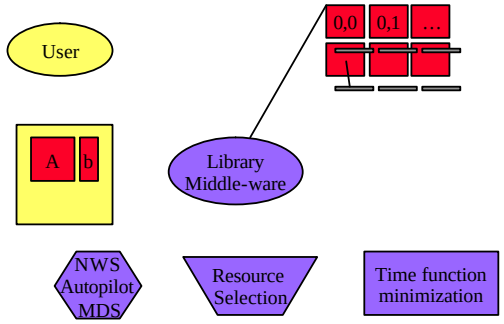


Numerical Libraries for Clusters



49

Numerical Libraries for Clusters



Uses Grid infrastructure, i.e.Globus/NWS, but doesn't have to.

50

LAPACK For Clusters (LFC)

- LFC will automate much of the decisions in the Cluster environment to provide best time to solution.
  - Adaptivity to the dynamic environment.
  - As the complexities of the Clusters and Grid increase need to develop strategies for self adaptability.
  - Handcrafted developed leading to an automated design.
- Developing a basic infrastructure for computational science applications and software in the Cluster and Grid environment.
  - Lack of tools is hampering development today.
- Plan to do suite: LU, Cholesky, QR, Symmetric eigenvalue, and Nonsymmetric eigenvalue
- Model for more general framework

51

ScaLAPACK - What's Included

Problem type Ax=b	SDrv	EDrv	Factor	Solve	Inv	Cond Est	Iter Refin
Triangular				X	X	X	X
SPD	X	X	X	X	X	X	X
SPD Banded	X		X	X			
SPD Tridiagonal	X		X	X			
General	X	X	X	X	X	X	X
General Banded	X		X	X			
General Tridiagonal	X		X	X			
Least squares	X		X	X			
GQR			X				
GRQ			X				

Ax = lx or Ax = lBx	SDrv	Edrv	Reduct	Solution
Symmetric (2 types)	X	X	X	X
General (2 types)			X	X
Generalized BSPD		X	X	X
SVD			X	X

- Timing and Testing routines for almost all
- This is a large component of the package
- Prebuilt libraries available for SP, PGON, HPPA, DEC, Sun, RS6K

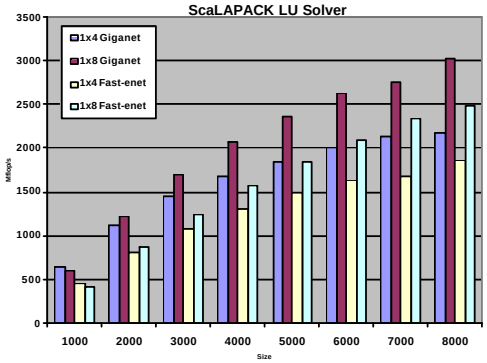
52

Heterogeneous Computing

- Software intended to be used in this context
- Communication of ft. pt. numbers between processors
- Machine precision and other machine specific parameters
- Iterative convergence across clusters of processors
- Defensive programming required

53

T0 Dual (500 MHz Alpha EV5/6)
Fast & Gigabit Ethernet
ScaLAPACK LU Solver



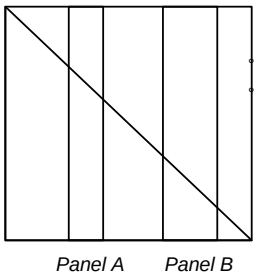
54

Out of Core Approach

- High-level I/O Interface
- ScaLAPACK uses a `Right-looking' variant for LU, QR and Cholesky factorizations.
- A `Left-looking' variant is used for Out-of-core factorization to reduce I/O traffic.
- Requires two in-core column panels.
- Imposes another level in the memory hierarchy.

55

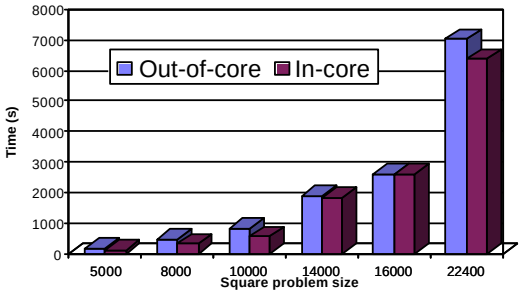
Out of Core Algorithm



- Hybrid approach
- Algorithm is ``Left-Looking'' in nature, but uses ``Right-Looking'' (ScaLAPACK) on Panel B
- Latency Tolerant
- Model for deep memory hierarchy algorithms

56

QR Factorization on 64 processors Intel Paragon



References

- http://www.netlib.org
- http://www.netlib.org/lapack
- http://www.netlib.org/scalapack
- http://www.netlib.org/lapack/lawns
- http://www.netlib.org/atlas
- http://www.netlib.org/papi/
- http://www.netlib.org/netsolve/
- http://www.netlib.org/lapack90
- http://www.nhse.org

- lapack@cs.utk.edu
- scalapack@cs.utk.edu

58