

# Lecture 10: Linear Algebra Algorithms

Jack Dongarra, U of Tennessee

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

## Outline

- Motivation for Dense Linear Algebra
  - $Ax=b$ : Computational Electromagnetics
  - $Ax = \lambda x$ : Quantum Chemistry
- Review Gaussian Elimination (GE) for solving  $Ax=b$
- Optimizing GE for caches on sequential machines
  - using matrix-matrix multiplication (BLAS)
- LAPACK library overview and performance
- Data layouts on parallel machines
- Parallel matrix-matrix multiplication
- Parallel Gaussian Elimination
- ScaLAPACK library overview
- Eigenvalue problem

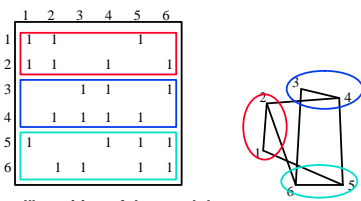
2

## Parallelism in Sparse Matrix-vector multiplication

- $y = A^*x$ , where  $A$  is sparse and  $n \times n$
- Questions
  - which processors store
    - $y[i]$ ,  $x[i]$ , and  $A[i,:]$
  - which processors compute
    - $y[i] = \text{sum}(\text{from } 1 \text{ to } n) A[i,:]*x$
    - = (row  $i$  of  $A$ ) .  $x$  ... a sparse dot product
- Partitioning
  - Partition index set  $\{1, \dots, n\} = N_1 \cup N_2 \cup \dots \cup N_p$
  - For all  $i$  in  $N_k$ , Processor  $k$  stores  $y[i]$ ,  $x[i]$ , and row  $i$  of  $A$
  - For all  $i$  in  $N_k$ , Processor  $k$  computes  $y[i] = (\text{row } i \text{ of } A) \cdot x$ 
    - "owner computes" rule: Processor  $k$  compute the  $y[i]$ s it owns
- Goals of partitioning
  - balance load (how is load measured?)
  - balance storage (how much does each processor store?)
  - minimize communication (how much is communicated?)

3

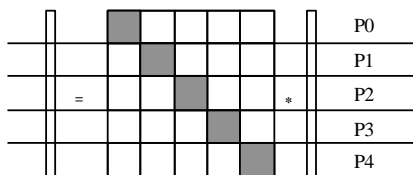
## Graph Partitioning and Sparse Matrices

- Relationship between matrix and graph
- 
- A "good" partition of the graph has
    - equal (weighted) number of nodes in each part (load and storage balance)
    - minimum number of edges crossing between (minimize communication)
  - Can reorder the rows/columns of the matrix by putting all the nodes in one partition together

4

## More on Matrix Reordering via Graph Partitioning

- "Ideal" matrix structure for parallelism: (nearly) block diagonal
- $p$  (number of processors) blocks
- few non-zeros outside these blocks, since these require communication



5

## What about implicit methods and eigenproblems?

- Direct methods (Gaussian elimination)
  - Called LU Decomposition, because we factor  $A = L^*U$
  - Future lectures will consider both dense and sparse cases
  - More complicated than sparse-matrix vector multiplication
- Iterative solvers
  - Will discuss several of these in future
    - Jacobi, Successive overrelaxation (SOR), Conjugate Gradients (CG), Multigrid,...
  - Most have sparse-matrix-vector multiplication in kernel
- Eigenproblems
  - Future lectures will discuss dense and sparse cases
  - Also depend on sparse-matrix-vector multiplication, direct methods
- Graph partitioning

6

## Partial Differential Equations PDEs

7

### Continuous Variables, Continuous Parameters

Examples of such systems include

- Heat flow: Temperature(position, time)
- Diffusion: Concentration(position, time)
- Electrostatic or Gravitational Potential: Potential(position)
- Fluid flow: Velocity, Pressure, Density(position, time)
- Quantum mechanics: Wave function(position, time)
- Elasticity: Stress, Strain(position, time)

8

### Example: Deriving the Heat Equation



Consider a simple problem

- A bar of uniform material, insulated except at ends
- Let  $u(x,t)$  be the temperature at position  $x$  at time  $t$
- Heat travels from  $x-h$  to  $x+h$  at rate proportional to:

$$\frac{d u(x,t)}{dt} = C * \frac{(u(x-h,t)-u(x,t))/h - (u(x,t) - u(x+h,t))/h}{h}$$

- As  $h \rightarrow 0$ , we get the heat equation:

$$\frac{d u(x,t)}{dt} = C * \frac{d^2 u(x,t)}{dx^2}$$

9

### Explicit Solution of the Heat Equation

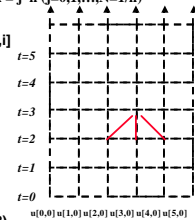
- For simplicity, assume  $C=l$
- Discretize both time and position
- Use finite differences with  $u[j,i]$  as the heat at
  - time  $t = i * dt$  ( $i = 0, 1, 2, \dots$ ) and position  $x = j * h$  ( $j = 0, 1, \dots, N = l/h$ )
  - initial conditions on  $u[j,0]$
  - boundary conditions on  $u[0,i]$  and  $u[N,i]$
- At each timestep  $i = 0, 1, 2, \dots$

For  $j=0$  to  $N$

$$u[j,i+1] = z * u[j-1,i] + (1-2*z) * u[j,i] + z * u[j+1,i]$$

where  $z = dt/h^2$

- This corresponds to
  - matrix vector multiply (what is matrix?)
  - nearest neighbors on grid



10

### Parallelism in Explicit Method for PDEs

- Partitioning the space ( $x$ ) into  $p$  largest chunks
  - good load balance (assuming large number of points relative to  $p$ )
  - minimized communication (only  $p$  chunks)



Generalizes to

- multiple dimensions
- arbitrary graphs (= sparse matrices)

Problem with explicit approach

- numerical instability
- solution blows up eventually if  $z = dt/h^2 > .5$
- need to make the timesteps very small when  $h$  is small:  $dt < .5 * h^2$

11

### Discretization Error

- How accurate will the approximate solution be?
- Beyond the scope of this course.
- The discretization error is

$$e = O(\Delta t) + O[(\Delta x)^2]$$

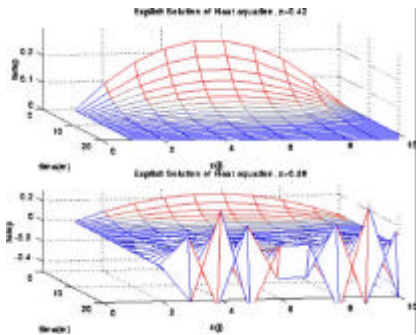
- The fact that  $\Delta t$  appears to the first power and  $\Delta x$  to the second power is usually described as the discretization is first-order accurate in time and second-order accurate in space.

- For the discretization to be stable  $\Delta t$  and  $\Delta x$  must satisfy the relationship

$$\Delta t \leq \frac{1}{2} (\Delta x)^2$$

12

### Instability in solving the heat equation explicitly



13

### Implicit Methods

- The previous method was called explicit because the value of  $u[j,i+1]$  at the next time level are obtained by an explicit formula in terms of the values at the previous time level.

$$u[j,i+1] = z^*u[j-1,i] + (1-2*z)*u[j,i] + z^*u[j+1,i]$$

- Consider the difference approximation

$$u[j,i+1] - u[j,i] = z^*(u[j+1,i+1] - 2^*u[j,i+1] + u[j-1,i+1])$$

- Similar in form but has the important difference that the values of  $u_j$  on the right are now evaluated at the  $i+1^{th}$  time level rather than at the  $i^{th}$ .
- Must solve equations to advance to the next time level.

14

### Implicit Solution

- As with many (stiff) ODEs, need an implicit method
- This turns into solving the following equation

$$(I + (z/2)*T) * u[:,i+1] = (I - (z/2)*T) * u[:,i]$$

- Here  $I$  is the identity matrix and  $T$  is:

$$T = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}$$



- I.e., essentially solving Poisson's equation in 1D

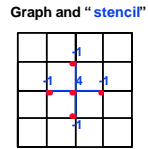
$$u[j,i+1] - u[j,i] = z^*(u[j+1,i+1] - 2^*u[j,i+1] + u[j-1,i+1])$$

15

### 2D Implicit Method

- Similar to the 1D case, but the matrix  $T$  is now

$$T = \begin{pmatrix} 4 & -1 & & & & & \\ -1 & 4 & -1 & & & & \\ & -1 & 4 & -1 & & & \\ -1 & & -1 & 4 & -1 & & \\ & -1 & & -1 & 4 & -1 & \\ & & -1 & & -1 & 4 & -1 \\ & & & -1 & & -1 & 4 \end{pmatrix}$$



- Multiplying by this matrix (as in the explicit case) is simply nearest neighbor computation on 2D grid
- To solve this system, there are several techniques

16

### Algorithms for 2D Poisson Equation with N unknowns

Algorithm	Serial	PRAM	Memory	#Procs
◦ Dense LU	$N^3$	N	$N^2$	$N^2$
◦ Band LU	$N^2$	N	$N^{3/2}$	N
◦ Jacobi	$N^2$	N	N	N
◦ Explicit Inv.	$N^2$	$\log N$	$N^2$	$N^2$
◦ Conj.Grad.	$N^{3/2}$	$N^{1/2} * \log N$	N	N
◦ RB SOR	$N^{3/2}$	$N^{1/2}$	N	N
◦ Sparse LU	$N^{3/2}$	$N^{1/2}$	$N * \log N$	N
◦ FFT	$N * \log N$	$\log N$	N	N
◦ Multigrid	N	$\log^2 N$	N	N
◦ Lower bound	N	$\log N$	N	N

PRAM is an idealized parallel model with zero cost communication (see next slide for explanation)

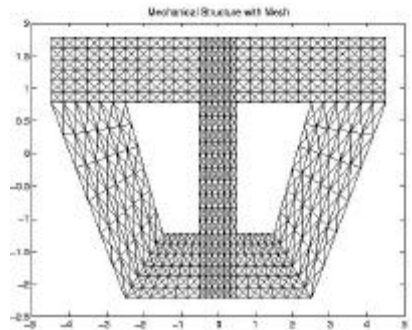
17

### Short explanations of algorithms on previous slide

- Sorted in two orders (roughly):
  - from slowest to fastest on sequential machines
  - from most general (works on any matrix) to most specialized (works on matrices "like" T)
- Dense LU: Gaussian elimination; works on any N-by-N matrix
- Band LU: exploit fact that T is nonzero only on  $\sqrt{N}$  diagonals nearest main diagonal, so faster
- Jacobi: essentially does matrix-vector multiply by T in inner loop of iterative algorithm
- Explicit Inverse: assume we want to solve many systems with T, so we can precompute and store  $\text{inv}(T)$  "for free", and just multiply by it
  - It's still expensive!
- Conjugate Gradients: uses matrix-vector multiplication, like Jacobi, but exploits mathematical properties of T that Jacobi does not
- Red-Black SOR (Successive Overrelaxation): Variation of Jacobi that exploits yet different mathematical properties of T
  - Used in Multigrid
- Sparse LU: Gaussian elimination exploiting particular zero structure of T
- FFT (Fast Fourier Transform): works only on matrices very like T
- Multigrid: also works on matrices like T, that come from elliptic PDEs
- Lower Bound: serial (time to print answer); parallel (time to combine N inputs)

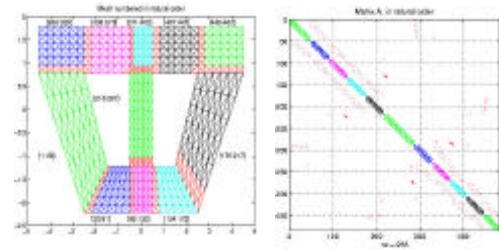
18

### Composite mesh from a mechanical structure



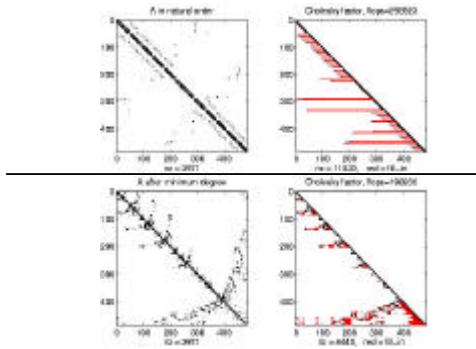
19

### Converting the mesh to a matrix



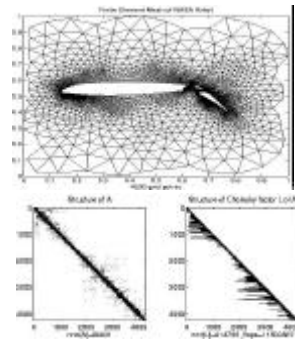
20

### Effects of Ordering Rows and Columns on Gaussian Elimination



21

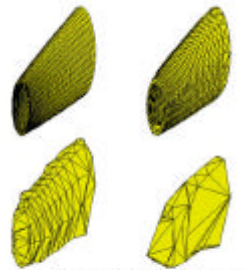
### Irregular mesh: NASA Airfoil in 2D (direct solution)



22

### Irregular mesh: Tapered Tube (multigrid)

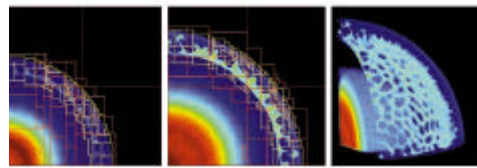
Example of Prometheus meshes



Sample input grid and coarse grids

23

### Adaptive Mesh Refinement (AMR)



- Adaptive mesh around an explosion
- John Bell and Phil Colella at LBL (see class web page for URL)
- Goal of Titanium is to make these algorithms easier to implement in parallel

24

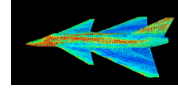
### Computational Electromagnetics

- Developed during 1980s, driven by defense applications
- Determine the RCS (radar cross section) of airplane
- Reduce signature of plane (stealth technology)
- Other applications are antenna design, medical equipment
- Two fundamental numerical approaches:
  - MOM methods of moments ( frequency domain), and
  - Finite differences (time domain)

25

### Computational Electromagnetics

- Discretize surface into triangular facets using standard modeling tools
- Amplitude of currents on surface are unknowns



- Integral equation is discretized into a set of linear equations

image: NW Univ. Comp. Electromagnetics Laboratory <http://nueml.ece.nwu.edu/>

26

### Computational Electromagnetics (MOM)

After discretization the integral equation has the form

$$A x = b$$

where

A is the (dense) impedance matrix,  
 x is the unknown vector of amplitudes, and  
 b is the excitation vector.

(see Cwik, Patterson, and Scott, Electromagnetic Scattering on the Intel Touchstone Delta, IEEE Supercomputing '92, pp 538 - 542)

27

### Computational Electromagnetics (MOM)

The main steps in the solution process are

- Fill: computing the matrix elements of A
- Factor: factoring the dense matrix A
- Solve: solving for one or more excitations b
- Field Calc: computing the fields scattered from the object

28

### Analysis of MOM for Parallel Implementation

Task	Work	Parallelism	Parallel Speed
Fill	$O(n^2)$	embarrassing	low
Factor	$O(n^3)$	moderately diff.	very high
Solve	$O(n^2)$	moderately diff.	high
Field Calc.	$O(n)$	embarrassing	high

29

### Results for Parallel Implementation on Delta

Task	Time (hours)
Fill	9.20
Factor	8.25
Solve	2.17
Field Calc.	0.12

The problem solved was for a matrix of size 48,672. (The world record in 1991.)

30

## Current Records for Solving Dense Systems

Year	System Size	Machine	# Procs	Gflops (Peak)
1950's	O(100)			
1995	128,600	Intel Paragon	6768	281 (338)
1996	215,000	Intel ASCI Red	7264	1068 (1453)
1998	148,000	Cray T3E	1488	1127 (1786)
1998	235,000	Intel ASCI Red	9152	1338 (1830)
1999	374,000	SGI ASCI Blue	5040	1608 (2520)
1999	362,880	Intel ASCI Red	9632	2379 (3207)
2000	430,000	IBM ASCI White	8192	4928 (12000)
2002	1,075,200	NEC Earth Sim	5120	35860 (41000)

source: Alan Edelman <http://www-math.mit.edu/~edelman/records.html>  
 LINPACK Benchmark: <http://www.netlib.org/performance/html/PDSreports.html>

31

## Computational Chemistry

- ° Seek energy levels of a molecule, crystal, etc.
  - Solve Schroedinger's Equation for energy levels = eigenvalues
  - Discretize to get  $Ax = \lambda Bx$ , solve for eigenvalues  $\lambda$  and eigenvectors  $x$
  - A and B large, symmetric or Hermitian matrices (B positive definite)
  - May want some or all eigenvalues/eigenvectors
- ° MP-Quest (Sandia NL)
  - Si and sapphire crystals of up to 3072 atoms
  - Local Density Approximation to Schroedinger Equation
  - A and B up to  $n=40000$ , Hermitian
  - Need all eigenvalues and eigenvectors
  - Need to iterate up to 20 times (for self-consistency)
- ° Implemented on Intel ASCI Red
  - 9200 Pentium Pro 200 processors (4600 Duals, a CLUMP)
  - Overall application ran at 605 Gflops (out of 1800 Gflops peak),
  - Eigensolver ran at 684 Gflops
  - [www.cs.berkeley.edu/~stanley/gbell/index.html](http://www.cs.berkeley.edu/~stanley/gbell/index.html)
  - Runner-up for Gordon Bell Prize at Supercomputing 98

32

## EISPACK and LINPACK

- ° EISPACK
  - Design for the algebraic eigenvalue problem,  $Ax = \lambda x$  and  $Ax = \lambda Bx$ .
  - work of J. Wilkinson and colleagues in the 70's.
  - Fortran 77 software based on translation of ALGOL.
- ° LINPACK
  - Design for the solving systems of equations,  $Ax = b$ .
  - Fortran 77 software using the Level 1 BLAS.

33

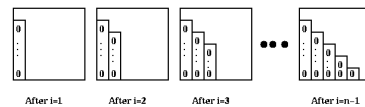
## Review of Gaussian Elimination (GE) for solving $Ax=b$

- ° 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)
  
```

Structure of Matrix during simple version of Gaussian Elimination



34

## 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)
  
```

35

## 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)
  
```

36

### 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)
      
```

37

### 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)
 

Work at step i of Gaussian Elimination

```

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)
  
```

38

### 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

39

### 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)
  A(i+1:n,i+1:n) = A(i+1:n,i+1:n) - A(i+1:n,i) * A(i,i+1:n)
  
```

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

40

### Pivoting in Gaussian Elimination

- $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$  fails completely, even though A is "easy"
- 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 ]
      
```
- Result of LU decomposition is
 

```

L = [ 1 0 ] = [ 1 0 ]
    [ fl(1/1e-4) 1 ] [ 1e4 1 ]
      
```

... No roundoff error yet

```

U = [ 1e-4  1 ] = [ 1e-4  1 ]
    [ 0 fl(1-1e4*1) ] [ 0 -1e4 ]
      
```

... Error in 4th decimal place

Check if  $A = L*U = \begin{bmatrix} 1e-4 & 1 \\ 1 & 0 \end{bmatrix}$  ... (2,2) entry entirely wrong
- 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

41

### Gaussian Elimination with Partial Pivoting (GEPP)

- Partial Pivoting: swap rows so that each multiplier  $|L(i,j)| = |A(j,i)/A(i,i)| \leq 1$ 

```

for i = 1 to n-1
  find and record k where |A(k,i)| = max (1 <= 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
  else k != i
    swap rows i and k of A
  end if
  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)
  
```
- Lemma:** This algorithm computes  $A = P*L*U$ , where P is a permutation matrix
- Since each entry of  $|L(i,j)| \leq 1$ , this algorithm is considered numerically stable
- For details see LAPACK code at [www.netlib.org/lapack/single/sgetf2.f](http://www.netlib.org/lapack/single/sgetf2.f)

42

### 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.

43

### Blocked Partitioned Algorithms

- LU Factorization
- Cholesky factorization
- Symmetric indefinite factorization
- Matrix inversion
- QR, QL, RQ, LQ factorizations
- Form Q or Q<sup>T</sup>C
- Orthogonal reduction to:
  - (upper) Hessenberg form
  - symmetric tridiagonal form
  - bidiagonal form
- Block QR iteration for nonsymmetric eigenvalue problems

44

### 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

45

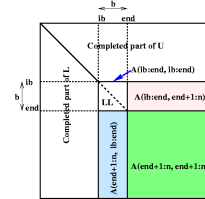
### Blocked GEPP ([www.netlib.org/lapack/single/sgetrf.f](http://www.netlib.org/lapack/single/sgetrf.f))

```

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
    
```

Gaussian Elimination using BLAS 3

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



46

### 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

47

### Derivation of Blocked Algorithms

#### Cholesky Factorization $A = U^T U$

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

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

$$a_j = U_{11}^T u_j$$

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

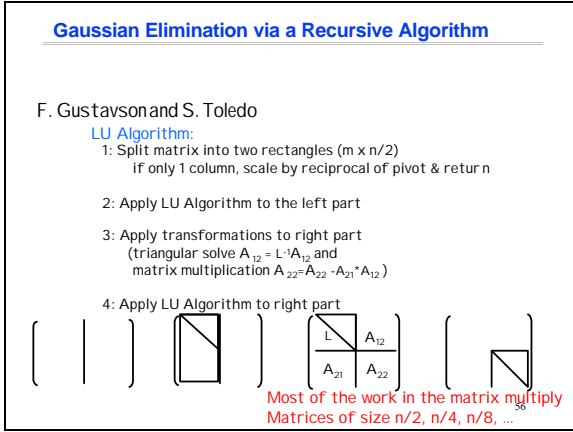
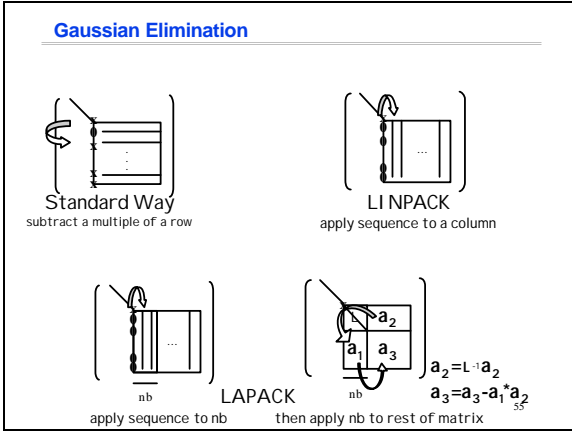
Hence, if  $U_{11}$  has already been computed, we can compute  $u_j$  and  $u_{jj}$  from the equations:

$$U_{11}^T u_j = a_j$$

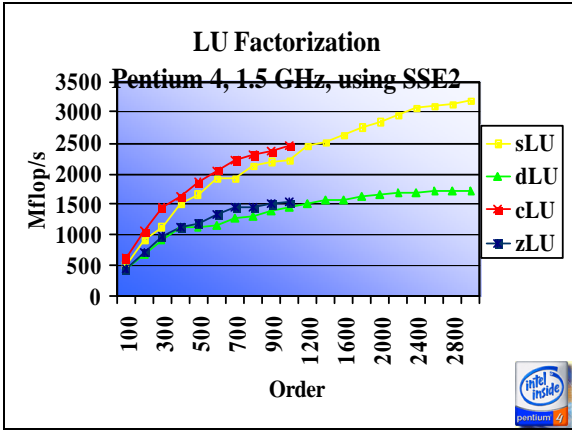
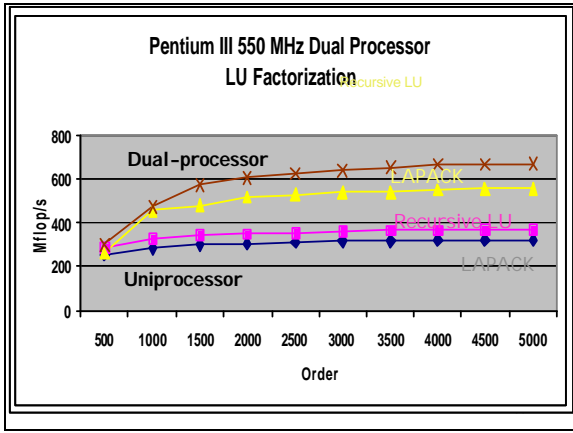
$$u_{jj}^2 = a_{jj} - u_j^T u_j$$

48





- ### 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<sup>T</sup>, & QR
    - Open question on 2-sided algs. eg eigenvalue reduction
- 57



- ### 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
- 60

## ScaLAPACK

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

61

## Programming Style

- SPMD Fortran 77 with object based design
- Built on various modules
  - PBLAS Interprocessor communication
  - BLACS
    - PVM, MPI, IBM SP, CR1 T3, Intel, TMC
    - Provides right level of notation.
  - BLAS
- LAPACK software expertise/quality
  - Software approach
  - Numerical methods

62

## 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

63

## PBLAS

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

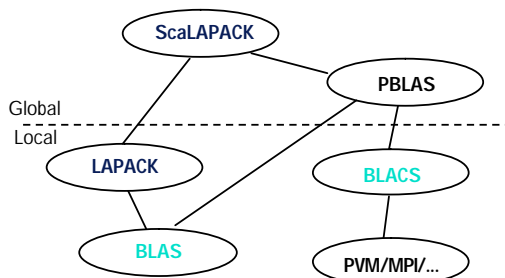
```
CALL DGEXXX ( M, N, A( IA, JA ), LDA,... )
```

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



64

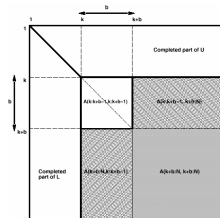
## ScaLAPACK Structure



65

## Choosing a Data Distribution

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



66

### 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

67

### 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.

68

### 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

69

### ScaLAPACK - What's Included

Problem type	SDrv	EDrv	Factor	Solve	Inv	Cond Est	Iter Refit
<b>Ax = b</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		X	X	X			
GQR			X				
GRQ			X				
<b>Ax = l x or Ax = l Bx</b>							
Symmetric (2 types)	X	X	X	X			
General (2 types)			X	X			
Generalized BSPD		X	X	X			
SVD			X	X			

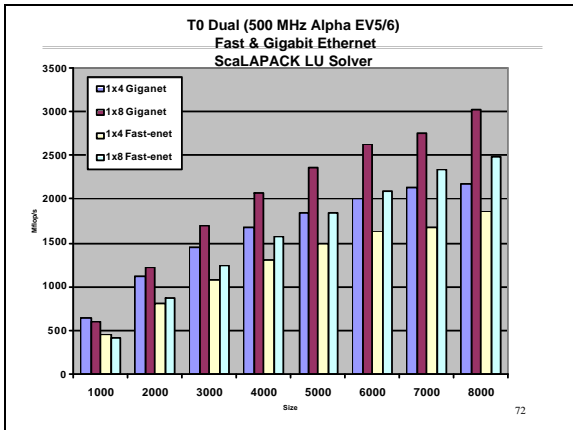
- Timing and Testing routines for almost all
- This is a large component of the package
- Prebuilt libraries available for SP, PGON, HPPA, DEC, Sun, RS6K

70

### 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

71



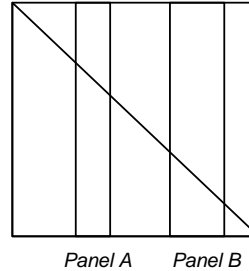
### Out of Core Approach

- 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.

73

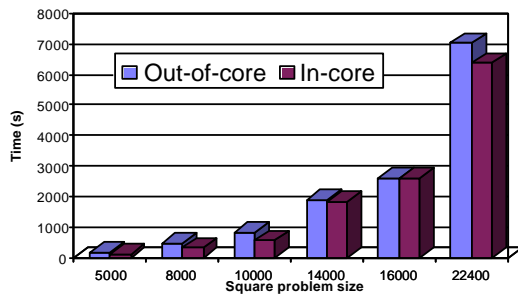
### Out of Core Algorithm

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



74

### 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](mailto:lapack@cs.utk.edu)
- [scalapack@cs.utk.edu](mailto:scalapack@cs.utk.edu)

76