CS 267 Applications of Parallel Computers

Lecture 19:

Dense Linear Algebra - I

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

Outline

- Motivation for Dense Linear Algebra (Lectures 10-11)
  - Ax=b: Computational Electromagnetics
  - Ax = λx: 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 problems
- Open Problems

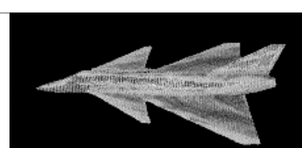
Computational Electromagnetics

- •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)

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/

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)

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

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

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.)

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)

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

Computational Chemistry

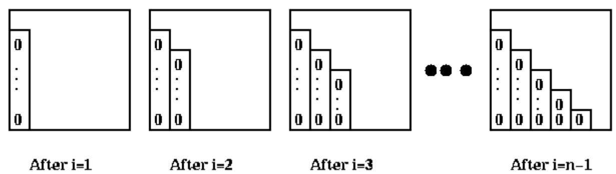
- Seek energy levels of a molecule, crystal, etc.
  - Solve Schroedinger's Equation for energy levels = eigenvalues
  - Discretize to get $Ax = \lambda Bx$, solve for eigenvalues λ and eigenvectors x
  - A and B large, symmetric or Hermitian matrices (B positive definite)
  - May want some or all eigenvalues/eigenvectors
- 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 Glops peak),
  - Eigensolver ran at 684 Gflops
  - www.cs.berkeley.edu/~stanley/gbell/index.html
  - Runner-up for Gordon Bell Prize at Supercomputing 98

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



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)
```

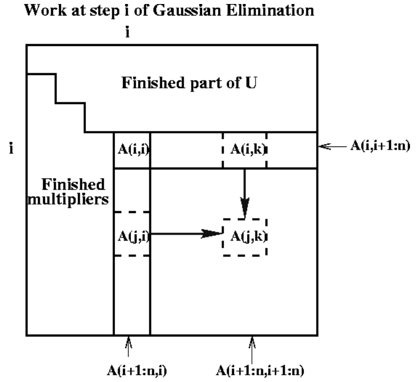
Refine GE Algorithm (4)

- **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)**

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)
```


What GE really computes

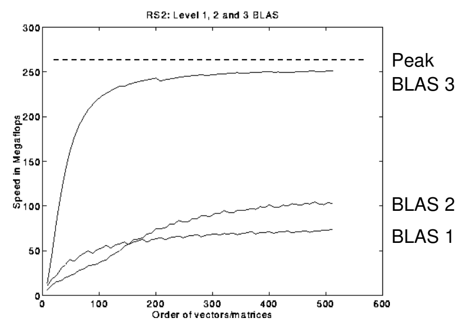
```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n ) - A(i+1:n , i) * A(i , i+1:n)
```

- **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)
```



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

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

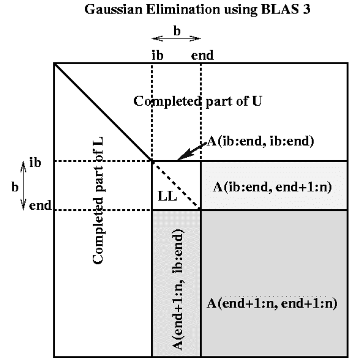
° **Need to choose a block size b**
- Algorithm will save and apply b updates
- b must be small enough so that active submatrix consisting of b columns of A fits in cache
- b must be large enough to make BLAS3 fast

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

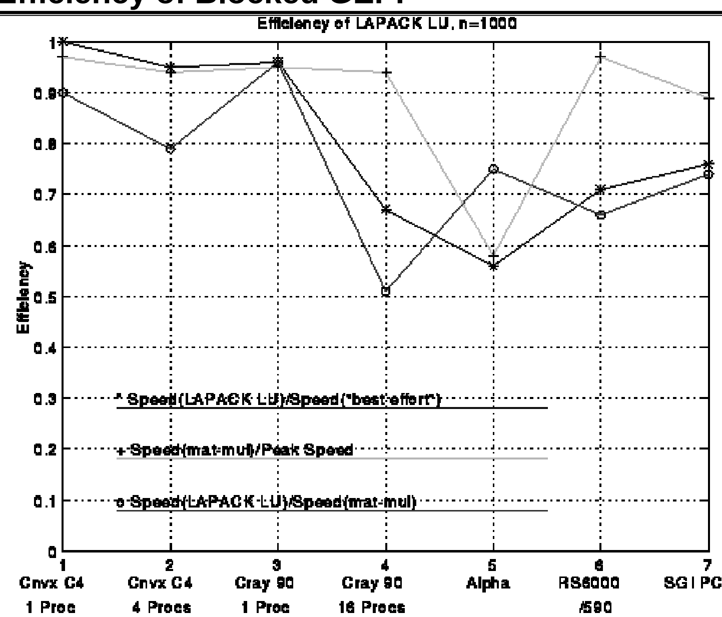
```
for   ib = 1 to n-1 step b      … Process matrix b columns at a time
     end = ib + b-1                  … Point to end of block of b columns
     apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P' * L' * U'
     … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
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

Gaussian Elimination using BLAS 3



Efficiency of Blocked GEPP



Overview of LAPACK

° **Standard library for dense/banded linear algebra**

- **Linear systems: A*x=b**
- **Least squares problems: $\min_x \| A*x-b \|_2$**
- **Eigenvalue problems: $Ax = \lambda x$, $Ax = \lambda Bx$**
- **Singular value decomposition (SVD): $A = U\Sigma V^T$**

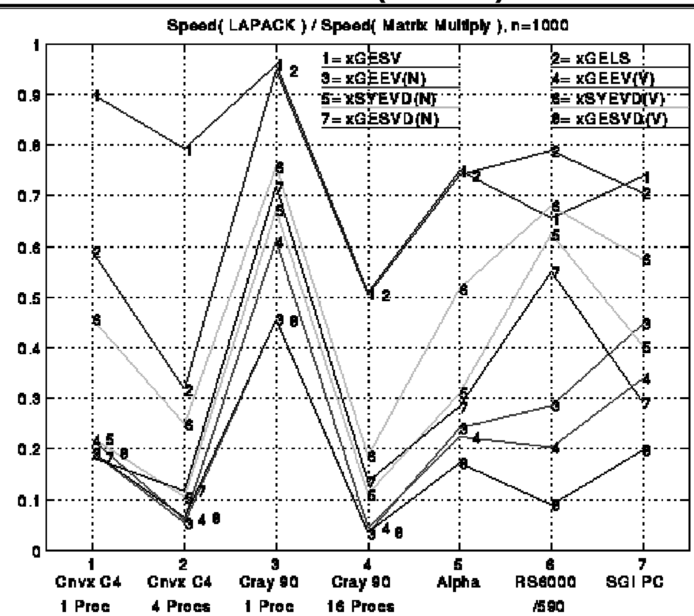
° **Algorithms reorganized to use BLAS3 as much as possible**

° **Basis of math libraries on many computers**

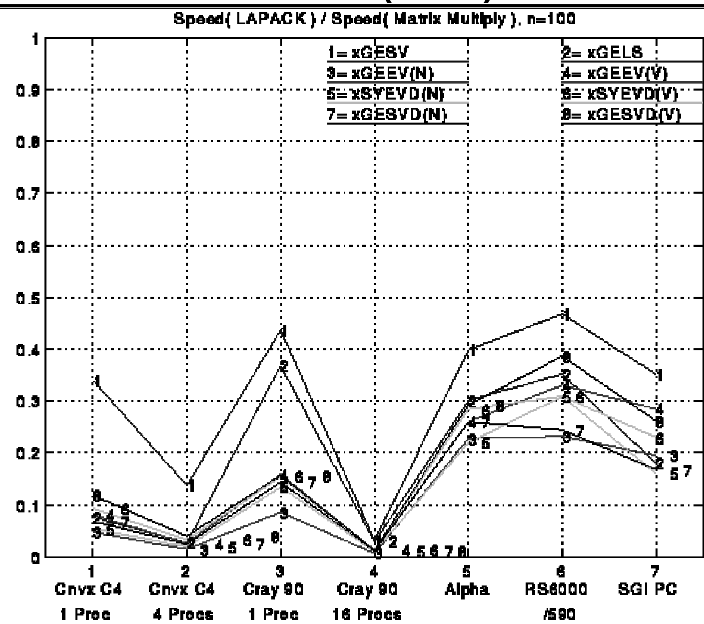
° **Many algorithmic innovations remain**

- **Projects available**

Performance of LAPACK (n=1000)



Performance of LAPACK (n=100)



Parallelizing Gaussian Elimination

° **Recall parallelization steps from Lecture 3**

- Decomposition: identify enough parallel work, but not too much
- Assignment: load balance work among threads
- Orchestrate: communication and synchronization
- Mapping: which processors execute which threads

° **Decomposition**

- In BLAS 2 algorithm nearly each flop in inner loop can be done in parallel, so with n^2 processors, need 3n parallel steps

```
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)
```

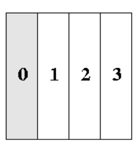
- This is too fine-grained, prefer calls to local matmuls instead

Assignment of parallel work in GE

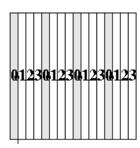
- Think of assigning submatrices to threads, where each thread responsible for updating submatrix it owns
  - "owner computes" rule natural because of locality
- What should submatrices look like to achieve load balance?

Different Data Layouts for Parallel GE (on 4 procs)

Bad load balance: P0 idle after first n/4 steps



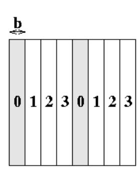
Load balanced, but can't easily use BLAS2 or BLAS3

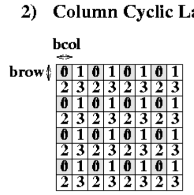


1) Column Blocked Layout

2) Column Cyclic Layout

Can trade load balance and BLAS2/3 performance by choosing b, but factorization of block column is a bottleneck

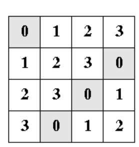




The winner!

3) Column Block Cyclic Layout

4) Row and Column Block Cyclic Layout



Complicated addressing

5) Block Skewed Layout