

CS 267 Applications of Parallel Computers

Dense Linear Algebra

Kathy Yelick

<http://www.cs.berkeley.edu/~yelick/cs267>

3/12/2004

1

Outline

- Motivation for Dense Linear Algebra
- Benchmarks
- 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 Gaussian Elimination
- ScaLAPACK library overview
- Eigenvalue problems
- Open Problems

3/12/2004

CS267 Lecture 14

2

Success Stories for ScaLAPACK

- New Science discovered through the solution of dense matrix systems
- ScaLAPACK is a library for dense and banded matrices
 - Nature article on the flat universe used ScaLAPACK
 - Other articles in Physics Review B that also use it
 - 1998 Gordon Bell Prize
 - Joint effort between DOE, DARPA, and NSF



Cosmic Microwave Background Analysis: BOOMERanG collaboration, MADCAP code (Apr. 27, 2000).

ScaLAPACK

3/12/2004

CS267 Lecture 14

3

Motivation (1)

3 Basic Linear Algebra Problems

1. Linear Equations: Solve $Ax=b$ for x
2. Least Squares: Find x that minimizes $\sum r_i^2$ where $r=Ax-b$
 - Statistics: Fitting data with simple functions
- 3a. Eigenvalues: Find λ and x where $Ax = \lambda x$
 - Vibration analysis, e.g., earthquakes, circuits
- 3b. Singular Value Decomposition: $A^T Ax = \sigma^2 x$
 - Information retrieval, web search

Lots of variations depending on structure of A

- A symmetric, positive definite, banded, ...

3/12/2004

CS267 Lecture 14

4

Motivation (2)

- Why dense A , as opposed to sparse A ?
 - Many large matrices are sparse, but ...
 - Dense algorithms easier to understand
 - Some applications yields large dense matrices
 - Benchmarking
 - "How fast is your computer?" =
 - "How fast can you solve dense $Ax=b$?"
 - Large sparse matrix algorithms often yield smaller (but still large) dense problems

3/12/2004

CS267 Lecture 14

5

Winner of TOPS 500 (LINPACK Benchmark)

Year	Machine	Tflops	Factor faster	Peak Tflops	Num Procs	N
2002	Earth System Computer, NEC	35.6	4.9	40.8	5104	1.04M
2001	ASCI White, IBM SP Power 3	7.2	1.5	11.1	7424	.52M
2000	ASCI White, IBM SP Power 3	4.9	2.1	11.1	7424	.43M
1999	ASCI Red, Intel PII Xeon	2.4	1.1	3.2	9632	.36M
1998	ASCI Blue, IBM SP 604E	2.1	1.6	3.9	5808	.43M
1997	ASCI Red, Intel Ppro, 200 MHz	1.3	3.6	1.8	9152	.24M
1996	Hitachi CP-PACS	.37	1.3	.6	2048	.10M
1995	Intel Paragon XP/S MP	.28	1	.3	6768	.13M

3/12/2004

CS267 Lecture 14

6

Source: Jack Dongarra (UTK)

Current Records for Solving Small Dense Systems

www.netlib.org, click on [Performance Database Server](#)

Machine	Megaflops		
	n=100	n=1000	Peak
Intel Pentium 4 (1 proc, 2.53 GHz)	1190	2355	5060
Compaq ES45 (4 proc, 1 GHz) (1 proc, 1 GHz)	824	1542	2000
NEC SX 6 (8 proc, 500 MHz) (1 proc, 500 MHz)	1161	7575	8000

3/12/2004

CS267 Lecture 14

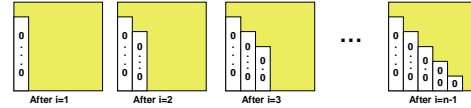
7

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
tmp = A(j,i);
for k = i to n
A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
    
```



3/12/2004

CS267 Lecture 14

8

Refine GE Algorithm (1)

- Initial Version

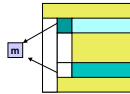
```

... 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
tmp = A(j,i);
for k = i to n
A(j,k) = A(j,k) - (tmp/A(i,i)) * A(i,k)
    
```

- Remove computation of constant $tmp/A(i,i)$ 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)
    
```



3/12/2004

CS267 Lecture 14

9

Refine GE Algorithm (2)

- Last version

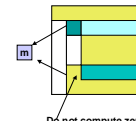
```

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)
    
```



Do not compute zeros

3/12/2004

CS267 Lecture 14

10

Refine GE Algorithm (3)

- Last version

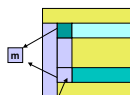
```

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)
    
```



Store m here

3/12/2004

CS267 Lecture 14

11

Refine GE Algorithm (4)

- Last version

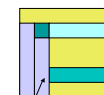
```

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)
    
```

- Split Loop

```

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



Store all m's here before updating rest of matrix

3/12/2004

CS267 Lecture 14

12

Refine GE Algorithm (5)

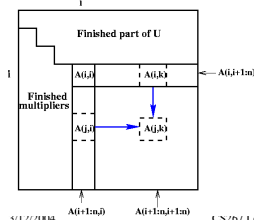
Last version

```

for i = 1 to n-1
  for j = i+1 to n
    A(j,i) = A(j,i)/A(i,i)
  for j = i+1 to n
    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 1 of Gaussian Elimination



```

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

3/12/2004

CS267 Lecture 14

13

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

3/12/2004

CS267 Lecture 14

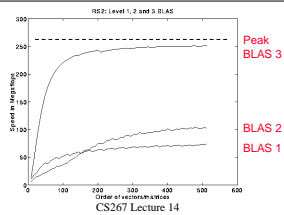
14

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 (earlier lectures...)

```

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)
  
```



3/12/2004

CS267 Lecture 14

15

Pivoting in Gaussian Elimination

$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ fails completely because can't divide by $A(1,1)=0$

- But solving $Ax=b$ should be easy!
- When diagonal $A(i,i)$ is tiny (not just zero), algorithm may terminate but get completely wrong answer
 - Numerical instability**
 - Roundoff error is cause
- Cure:** **Pivot** (swap rows of A) so $A(i,i)$ large

3/12/2004

CS267 Lecture 14

16

Gaussian Elimination with Partial Pivoting (GEPP)

- Partial Pivoting:** swap rows so that $A(i,i)$ is largest in column

```

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.
- This algorithm is numerically stable in practice
- For details see LAPACK code at <http://www.netlib.org/lapack/single/sgeff2.f>

3/12/2004

CS267 Lecture 14

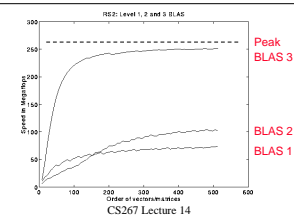
17

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 (earlier lectures...)

```

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)
  
```



3/12/2004

CS267 Lecture 14

18

Parallelizing Gaussian Elimination

- Parallelization steps
 - 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
 - Need to use parallel matrix multiplication
- Assignment
 - Which processors are responsible for which submatrices?

Different Data Layouts for Parallel GE

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

1) 1D Column Blocked Layout

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

2) 1D Column Cyclic Layout

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

3) 1D Column Block Cyclic Layout

Complicated addressing

4) Block Skewed Layout

Bad load balance:
P0 idle after first $n/2$ steps

5) 2D Row and Column Blocked Layout

The winner!

6) 2D Row and Column Block Cyclic Layout

Review of Parallel MatMul

- Want Large Problem Size Per Processor
- PDGEMM = PBLAS matrix multiply
- Observations:
 - For fixed N , as P increases Mflops increases, but less than 100% efficiency
 - For fixed P , as N increases, Mflops (efficiency) rises

Performance of PBLAS

Machine	Procs	Block Size	Speed in Mflops of PDGEMM		
			2000	4000	10000
Cray T3E	4=2x2	32	1055	1070	0
	16=4x4	360	3630	4005	4292
	64=8x8	13456	14287	16755	
IBM SP2	4	50	755	0	0
	16	2514	2850	0	
	64	6205	8709	10774	
Intel XP/S MP	4	32	330	0	0
	16	1233	1284	0	
	64	4496	4864	5257	
Berkeley NOW	4	32	463	470	0
	16	2490	2822	3450	
	64	4130	5457	6647	

- Efficiency = $Mflops(PDGEMM) / (Procs * Mflops(DGEMM))$
- DGEMM = BLAS routine for matrix multiply
- Maximum speed for PDGEMM = # Procs * speed of DGEMM

- Observations:
 - Efficiency always at least 48%
 - For fixed N , as P increases, efficiency drops
 - For fixed P , as N increases, efficiency increases

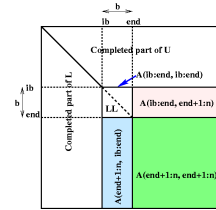
Review: BLAS 3 (Blocked) GEPP

- 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 $A(ib:n, ib:end) = P^{-1} * L^{-1} * U$
 - let LL denote the strict lower triangular part of $A(ib:end, ib:end) + I$
 - update next b rows of U

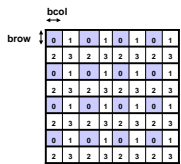
```

A(ib:end, end+1:n) = LL^{-1} * A(ib:end, end+1:n)
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)
    
```

Gaussian Elimination using BLAS 3



Row and Column Block Cyclic Layout



- processors and matrix blocks are distributed in a 2d array
- pcol-fold parallelism in any column, and calls to the BLAS2 and BLAS3 on matrices of size $brow$ -by- $bcol$
- serial bottleneck is eased
- need not be symmetric in rows and columns

Distributed GE with a 2D Block Cyclic Layout

- block size b in the algorithm and the block sizes $brow$ and $bcol$ in the layout satisfy $b=brow=bcol$.
- shaded regions indicate processors busy with computation or communication.
- unnecessary to have a barrier between each step of the algorithm, e.g., step 9, 10, and 11 can be pipelined

Distributed Gaussian Elimination with a 2D Block Cyclic Layout

for $ib = 1$ to $n-1$ step b

$end = \min(ib+b-1, n)$

for $i = ib$ to end

- (1) find pivot row k , column broadcast
- (2) swap rows k and i in block column, broadcast row k
- (3) $A(i+1:n, i) = A(i+1:n, i) / A(i, i)$
- (4) $A(i+1:n, i+1:end) = A(i+1:n, i) * A(i, i+1:end)$

end for

- (5) broadcast all swap information right and left
- (6) apply all rows swaps to other columns

- (7) Broadcast LL right
- (8) $A(ib:ib, end+1:n) = LL \setminus A(ib:ib, end+1:n)$
- (9) Broadcast $A(ib:ib, end+1:n)$ down
- (10) Broadcast $A(end+1:n, ib:ib)$ right
- (11) Eliminate $A(end+1:n, end+1:n)$

Matrix multiply of
green = green * blue * pink

ScaLAPACK Performance Models (1)

ScaLAPACK Operation Counts

$$T(N, P) = \frac{C_2 N^3}{P} + \frac{C_3 N^2}{\sqrt{P}} + \frac{C_4 N}{\sqrt{P}} + \dots \quad T_{comm}(N, P) = C_5 N^2 \sqrt{P}$$

$$B(N, P) = \left(1 + \frac{1}{\sqrt{P}} \frac{C_6 N}{C_7 \sqrt{P}} + \frac{C_8 N}{C_9 \sqrt{P}} \right)^{-1}$$

Driver	Options	C_2	C_3	C_4
PdGSEV	1 right hand side	$2/3$	$3 + 1/3 \sqrt{3} \log_2 P$	$5/3 (3 + \log_2 P)$
PdPOSV	1 right hand side	$1/3$	$2 + 1/3 \sqrt{3} \log_2 P$	$4 + \log_2 P$
PdGELS	1 right hand side	$4/3$	$3 + \log_2 P$	$3 (3 \log_2 P + 1)$
PdSYEV1	eigenvalues only	$4/3$	$5/3 \log_2 P$	$17/3 \sqrt{N} + 2$
PdSYEV2	eigenvalues and eigenvectors	$10/3$	$1 \log_2 P$	$17/3 \sqrt{N} + 2$
PdSYEV3	eigenvalues only	$4/3$	$5/3 \log_2 P$	$17/3 \sqrt{N} + 2$
PdSYEV4	eigenvalues and eigenvectors	$12/3$	$1 \log_2 P$	$17/3 \sqrt{N} + 2$
PdGSEVD	singular values only	$26/3$	$10 \log_2 P$	$17 \sqrt{N}$
PdGSEVD	singular values and left and right singular vectors	$35/3$	$14 \log_2 P$	$17 \sqrt{N}$
PdLAHR1	eigenvalues only	1	$8/3 (\sqrt{P} + \log_2 P)$	$8 (2 + \log_2 P) \sqrt{N}$
PdLAHR2	full Schur form	10	$8/3 (\sqrt{P} + \log_2 P)$	$8 (2 + \log_2 P) \sqrt{N}$

3/12/2004 CS267 Lecture 14 33

ScaLAPACK Performance Models (2)

Compare Predictions and Measurements

DRIVER	P	Values of N									
		2000		4000		7000		10000		15000	
		Est.	Obs.	Est.	Obs.	Est.	Obs.	Est.	Obs.	Est.	Obs.
PdGSESV (LU)	4	387	421	632	683						
	16	467	727	1024	1045	2116	1923	3454	2149		
	64	622	824	2432	2017	4236	4208	8765	8886	7887	7087
PdPOSV (Cholesky)	4	680	467	660	678						
	16	1316	1024	2025	1811	2308	2118	3826	2312		
	64	2677	1927	4237	4423	6708	6727	7801	8206	8987	8004

*Our guesses are good for each and our computational PdGSESV and PdPOSV are good.

3/12/2004 CS267 Lecture 14 34

PDGSEV = ScaLAPACK Parallel LU

Efficiency = $\frac{\text{MFlops}(\text{PdGSEV})}{\text{MFlops}(\text{PdGEMM})}$

Since it can run no faster than its inner loop (PDGEMM), we measure:

$$\text{Efficiency} = \frac{\text{Speed}(\text{PdGSEV})}{\text{Speed}(\text{PdGEMM})}$$

Observations:

- Efficiency well above 50% for large enough problems
- For fixed N, as P increases, efficiency decreases (just as for PDGEMM)
- For fixed P, as N increases efficiency increases (just as for PDGEMM)
- From bottom table, cost of solving
 - Ax=b about half of matrix multiply for large enough matrices.
- From the flop counts we would expect it to be $(2 \cdot n^3) / (2/3 \cdot n^3) = 3$ times faster, but communication makes it a little slower.

Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.67	.82	
	16		.44	.65	.84
	64		.18	.47	.75
IBM SP2	4	80	.56		
	16		.29	.52	
	64		.15	.32	.66
Intel XP/S MP	4	32	.64		
	16		.37	.66	
	64		.16	.42	.75
Berkeley NOW	4	32	.76	.82	.71
	16		.38	.62	
	64		.28	.54	.69

Machine	Procs	Block Size	Time(PdGSEV)/Time(PdGEMM)		
			N=2000	N=4000	N=10000
Cray T3E	4	32	.50	.40	
	16		.75	.51	.40
	64		1.86	.72	.45
IBM SP2	4	80	.60		
	16		1.16	.64	
	64		2.24	1.03	.51
Intel XP/S GP	4	32	.52		
	16		.89	.50	
	64		2.08	.79	.44
Berkeley NOW	4	32	.44		
	16		.88	.54	.47
	64		1.18	.62	.49

3/12/2004 CS267 Lec

ScaLAPACK Overview

ScaLAPACK SOFTWARE HIERARCHY

3/12/2004 CS267 Lecture 14 36

Parallelism in ScaLAPACK

- Level 3 BLAS block operations
 - All the reduction routines
- Pipelining
 - QR iteration, 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

3/12/2004

CS267 Lecture 14

37

QR (Least Squares)

Performance of ScaLAPACK QR (Least squares)

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

Scales well,
nearly full machine speed

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

3/12/2004

CS267

Performance of Symmetric Eigensolvers

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

Old version,
pre 1998 Gordon Bell Prize
Still have ideas to accelerate

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

Old Algorithm,
plan to abandon

3/12/2004

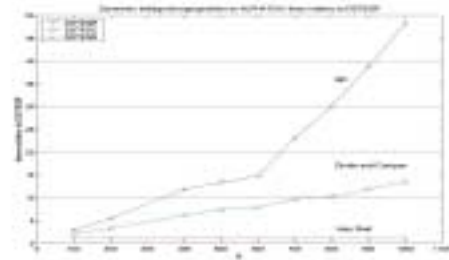
CS267 Lecture 14

39

Scalable Symmetric Eigensolver and SVD

The "Holy Grail" (Parlett, Dhillon, Marques)

Perfect Output complexity ($O(n * \#vectors)$), Embarrassingly parallel, Accurate



To be propagated throughout LAPACK and ScaLAPACK

3/12/2004

CS267 Lecture 14

40

Performance of SVD (Singular Value Decomposition)

Machine	Procs	Block Size	Time(PDGESVD)/Time(PDGEMM)	
			N	2000
Cray T3E	4	32	67	64
	16		66	64
	64		93	70
IBM SP2	4	50	97	90
	16		80	81
	64		81	81
Berkeley NOW	4	32	72	16
	16		38	16
	32		59	26

Have good ideas to speedup
Project available!

Performance of Nonsymmetric Eigensolver (QR iteration)

Machine	Procs	Block Size	Time(PDLAQR)/Time(PDGEMM)	
			N	1000
Intel XP/S MP Paragon	16	50	123	97
	64		123	97

Hardest of all to parallelize

3/12/2004

CS267 Lecture 14

41

Scalable Nonsymmetric Eigensolver

- $Ax_i = \lambda_i x_i$, Schur form $A = QTQ^T$
- Parallel HQR
 - Henry, Watkins, Dongarra, Van de Geijn
 - Now in ScaLAPACK
 - Not as scalable as LU: N times as many messages
 - Block-Hankel data layout better in theory, but not in ScaLAPACK
- Sign Function
 - Beavers, Denman, Lin, Zmijewski, Bai, Demmel, Gu, Godunov, Bulgakov, Malyshev
 - $A_{i+1} = (A_i + A_i^{-1})/2 \rightarrow$ shifted projector onto $\text{Re } \lambda > 0$
 - Repeat on transformed A to divide-and-conquer spectrum
 - Only uses inversion, so scalable
 - Inverse free version exists (uses QRD)
 - Very high flop count compared to HQR, less stable

3/12/2004

CS267 Lecture 14

42

Out of "Core" Algorithms

Out-of-Core Performance Results for Least Squares

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

Out-of-core means matrix lives on disk; too big for main mem

Much harder to hide latency of disk

QR much easier than LU because no pivoting needed for QR

Source: Jack Dongarra

3/12/2004

Recursive Algorithms

- Still uses delayed updates, but organized differently
 - (formulas on board)
- Can exploit recursive data layouts
 - 3x speedups on least squares for tall, thin matrices

- Theoretically optimal memory hierarchy performance
- See references at
 - <http://lawra.uni-c.dk/lawra/index.html>
 - <http://www.cs.umu.se/research/parallel/recursion/>

CS267 Lecture 14

3/12/2004

Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

LU Algorithm:

- 1: Split matrix into two rectangles ($m \times n/2$)
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 $n/2$, $n/4$, $n/8$, ...

Source: Jack Dongarra

CS267 Lecture 14

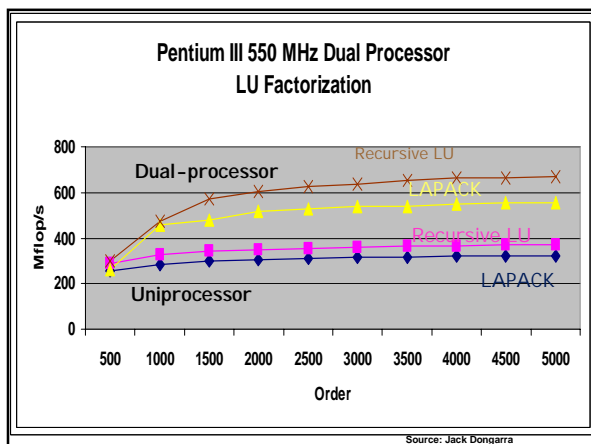
3/12/2004

Recursive Factorizations

- Just as accurate as conventional method
- Same number of operations
- Automatic variable-size 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).

CS267 Lecture 14

3/12/2004



Recursive Algorithms – Limits

- Two kinds of dense matrix compositions
- One Sided
 - Sequence of simple operations applied on left of matrix
 - Gaussian Elimination: $A = L^*U$ or $A = P^*L^*U$
 - Symmetric Gaussian Elimination: $A = L^*D^*L^T$
 - Cholesky: $A = L^*L^T$
 - QR Decomposition for Least Squares: $A = Q^*R$
 - Can be nearly 100% BLAS 3
 - Susceptible to recursive algorithms
- Two Sided
 - Sequence of simple operations applied on both sides, alternating
 - Eigenvalue algorithms, SVD
 - At least ~25% BLAS 2
 - Seem impervious to recursive approach?
 - Some recent progress on SVD (25% vs 50% BLAS2)

CS267 Lecture 14

3/12/2004

ScalAPACK Summary and Conclusions

- “One-sided Problems” are scalable
 - LU (“Linpack Benchmark”)
 - Cholesky, QR
- “Two-sided Problems” are harder
 - Half BLAS2, not all BLAS3
 - Eigenproblems, SVD (Holy Grail coming...)
 - 684 Gflops on 4600 PE ASCI Red (149 Mflops/proc)
 - Henry, Stanley, Sears
 - Hermitian generalized eigenproblem $Ax = \lambda Bx$
 - 2nd Place, Gordon Bell Peak Performance Award, SC98
- Narrow band problems hardest
 - Solving and eigenproblems
 - Galois theory of parallel prefix
- www.netlib.org/scalapack

3/12/2004

CS267 Lecture 14

49

A small software project ...

Participants

Krzysztof Anaszkiewicz (UC Berkeley)	Zhaojun Bai (U Kentucky)
Richard Barrett (U. Texas)	Michael Berry (U. Texas)
Jeff Bilmes (UC Berkeley)	Chris Bischof (ANL)
Sumit Bhattacharya (ORNL)	Sourav Chakrabarti (UC Berkeley)
Tony Chan (UC LA)	Chao-Wei Chin (UC Berkeley)
Jaeoung Choi (LSNL)	Andy Cleary (LENL)
Eli D'Azevedo (ORNL)	Jim Demmel (UC Berkeley)
Deleight Ehillon (UC Berkeley)	Jane Donato (ORNL)
Jack Dongarra (U. Texas, ORNL)	Zlatko Drmac (U. Hagen)
Jeremy Du Crocq (NAG)	Vince Eijkhout (UC LA)
Shan Ekanlati (Yale)	Vinco Ferraro (NAG)
John Gilbert (Xerox PARC)	Ming Gu (UC Berkeley, LLNL)
Sven Hammarling (NAG)	Mike Heath (U. Illinois)
Greg Henry (Intel)	Dominik Lam (UC Berkeley)
Steve Hum-Lee (IBM)	Bo Kågström (U. Umeå)
W. Kahan (UC Berkeley)	Yongqiang Kiri (U. Texas)
Ramcang Li (UC Berkeley)	Xiaoqi Li (UC Berkeley)
Joseph Liu (Yorq)	Bowdoin Padgett (UC Berkeley)
Antoine Pottier (U. Texas)	Peter Ponomarev (U. Umeå)
Hakim Pons (U. Texas)	Padma Raghavan (U. Illinois)
Huan Ren (UC Berkeley)	Howard Robinson (UC Berkeley)
Charles Romine (ORNL)	Jeff Rutter (UC Berkeley)
Yoon Sangjune (U. South)	Earl Steihaug (Hewlett)
Ken Stanley (UC Berkeley)	Xiaohai Sun (ANL)
Bernard Teunissen (U. Texas)	Anita Tang (ORNL)
Robert van de Geijn (U. Texas)	Henk van der Vorst (Utrecht U)
Paul Van Dooren (U. Illinois)	Krestimir Veselic (U. Hagen)
Daniel Walker (ORNL)	Clint Whaley (U. Texas)
Kathy Yalick (UC Berkeley)	

With the cooperation of
Cray, IBM, Comex, EBC, Fujitsu, NRC, NAG, SGI, SMC

3/12/2004

Supported by AFPA, NSF, DOE

50

Extra Slides

3/12/2004

51

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?

3/12/2004

CS267 Lecture 14

52

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

3/12/2004

CS267 Lecture 14

53

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

3/12/2004

CS267 Lecture 14

54

BLAS2 version of GE with Partial Pivoting (GEPP)

```

for i = 1 to n-1
  find and record k where |A(k,i)| = max(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]
  ... BLAS 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)
  ... BLAS 2, most work in this line
    
```

3/12/2004

CS267 Lecture 14

55

Computational Electromagnetics – Solve $Ax=b$

- 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)
 - Large dense matrices
 - Finite differences (time domain)
 - Even larger sparse matrices

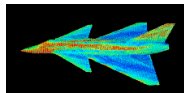
3/12/2004

CS267 Lecture 14

56

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.ecn.nwu.edu/>

3/12/2004

CS267 Lecture 14

57

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)

3/12/2004

CS267 Lecture 14

58

Results for Parallel Implementation on Intel Delta

Task	Time (hours)
Fill (compute n^2 matrix entries) (embarrassingly parallel but slow)	9.20
Factor (Gaussian Elimination, $O(n^3)$) (good parallelism with right algorithm)	8.25
Solve ($O(n^2)$) (reasonable parallelism with right algorithm)	2.17
Field Calc. ($O(n)$) (embarrassingly parallel and fast)	0.12

The problem solved was for a matrix of size 48,672.
2.6 Gflops for Factor - The world record in 1991.

3/12/2004

CS267 Lecture 14

59

Computational Chemistry – $Ax = \lambda x$

- Seek energy levels of a molecule, crystal, etc.
 - Solve Schrodinger's Equation for energy levels = eigenvalues
 - Discretize to get $Ax = \lambda Bx$, solve for eigenvalues λ and eigenvectors x
 - A and B large Hermitian matrices (B positive definite)
- MP-Quest (Sandia NL)
 - Si and sapphire crystals of up to 3072 atoms
 - A and B up to $n=40000$, complex 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
 - Runner-up for Gordon Bell Prize at Supercomputing 98

3/12/2004

CS267 Lecture 14

60

LAPACK and ScaLAPACK

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

3/12/2004

CS267 Lecture 14

61