

Lecture 7: Linear Algebra Algorithms

Jack Dongarra, U of Tennessee

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

1

Outline

- Motivation, overview for Dense Linear Algebra
- 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

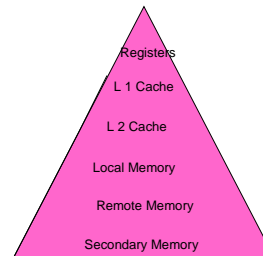
2

BLAS -- Introduction

- **Clarity:** code is shorter and easier to read,
- **Modularity:** gives programmer larger building blocks,
- **Performance:** manufacturers will provide tuned machine-specific BLAS,
- **Program portability:** machine dependencies are confined to the BLAS

3

Memory Hierarchy

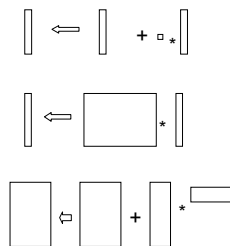


- Key to high performance in effective use of memory hierarchy
- True on all architectures

4

Level 1, 2 and 3 BLAS

- **Level 1 BLAS** Vector-Vector operations
- **Level 2 BLAS** Matrix-Vector operations
- **Level 3 BLAS** Matrix-Matrix operations



5

More on BLAS (Basic Linear Algebra Subroutines)

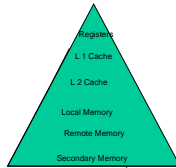
- Industry standard interface (evolving)
- Vendors, others supply optimized implementations
- History
 - **BLAS1 (1970s):**
 - vector operations: dot product, saxpy ($y = \alpha \cdot x + y$), etc
 - $m=2 \cdot n$, $f=2 \cdot n$, $q \sim 1$ or less
 - **BLAS2 (mid 1980s)**
 - matrix-vector operations: matrix vector multiply, etc
 - $m=n^2$, $f=2 \cdot n^2$, $q \sim 2$, less overhead
 - somewhat faster than BLAS1
 - **BLAS3 (late 1980s)**
 - matrix-matrix operations: matrix matrix multiply, etc
 - $m \geq 4n^2$, $f=O(n^3)$, so q can possibly be as large as n , so BLAS3 is potentially much faster than BLAS2
- Good algorithms used BLAS3 when possible (LAPACK)
- www.netlib.org/blas, www.netlib.org/lapack

6

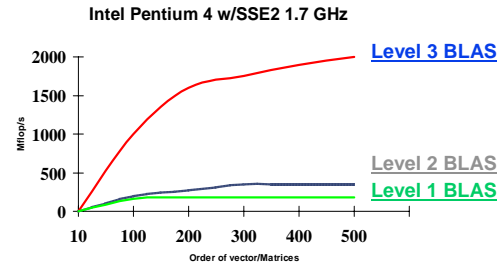
Why Higher Level BLAS?

- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

BLAS	Memory Refs	Flops	Flops/Memory Refs
Level 1 $y = y + \alpha x$	$3n$	$2n$	$2/3$
Level 2 $y = y + Ax$	n^2	$2n^2$	2
Level 3 $C = C + AB$	$4n^2$	$2n^3$	$n/2$



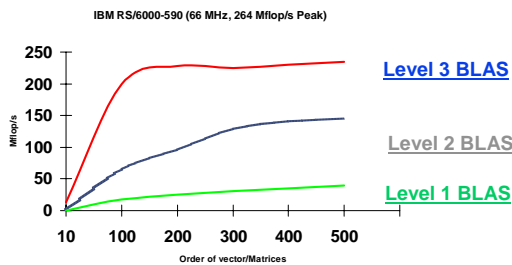
BLAS for Performance



- Development of blocked algorithms important for performance

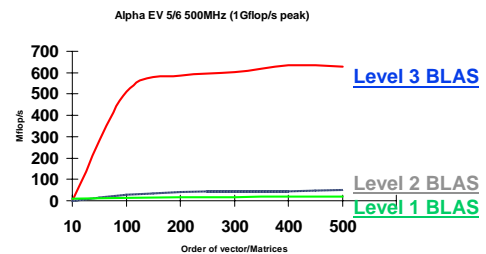
8

BLAS for Performance



9

BLAS for Performance



BLAS 3 (n-by-n matrix matrix multiply) vs
BLAS 2 (n-by-n matrix vector multiply) vs
BLAS 1 (saxpy of n vectors)

10

Fast linear algebra kernels: BLAS

- Simple linear algebra kernels such as matrix-matrix multiply
- More complicated algorithms can be built from these basic kernels.
- The interfaces of these kernels have been standardized as the Basic Linear Algebra Subroutines (BLAS).
- Early agreement on standard interface (~1980)
- Led to portable libraries for vector and shared memory parallel machines.
- On distributed memory, there is a less-standard interface called the PBLAS

11

Level 1 BLAS

- Operate on vectors or pairs of vectors
 - perform $O(n)$ operations;
 - return either a vector or a scalar.
- saxpy
 - $y(i) = a * x(i) + y(i)$, for $i=1$ to n .
 - s stands for single precision, daxpy is for double precision, caxpy for complex, and zaxpy for double complex,
- sscal $y = a * x$, for scalar a and vectors x, y
- sdot computes $s = \sum_{i=1}^n x(i)*y(i)$

12

Level 2 BLAS

- Operate on a matrix and a vector;
 - return a matrix or a vector;
 - $O(n^2)$ operations
- sgemv*: matrix-vector multiply
 - $y = y + A*x$
 - where A is m -by- n , x is n -by-1 and y is m -by-1.
- sger*: rank-one update
 - $A = A + y*x^T$, i.e., $A(i,j) = A(i,j) + y(i)*x(j)$
 - where A is m -by- n , y is m -by-1, x is n -by-1,
 - *strsv*: triangular solve
 - solves $y = T*x$ for x , where T is triangular

13

Level 3 BLAS

- Operate on pairs or triples of matrices
 - returning a matrix;
 - complexity is $O(n^3)$.
- sgemm*: Matrix-matrix multiplication
 - $C = C + A*B$,
 - where C is m -by- n , A is m -by- k , and B is k -by- n
- strsm*: multiple triangular solve
 - solves $Y = T*X$ for X ,
 - where T is a triangular matrix, and X is a rectangular matrix.

14

Review of the BLAS

- Building blocks for all linear algebra
- Parallel versions call serial versions on each processor
 - So they must be fast!
- Recall $q = \# \text{ flops} / \# \text{ mem refs}$
 - The larger is q , the faster the algorithm can go in the presence of memory hierarchy
 - "axpy": $y = \alpha*x + y$, where α scalar, x and y vectors

BLAS level	Ex.	# mem refs	# flops	q
1	"Axy", Dot prod	$3n$	$2n^1$	$2/3$
2	Matrix- vector mult	n^2	$2n^2$	2
3	Matrix- matrix mult	$4n^2$	$2n^3$	$n/2$

15

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
 - www.nersc.gov/news/reports/newNERSResults050703.pdf
 - Joint effort between DOE, DARPA, and NSF



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

ScaLAPACK

16

Motivation (1)

3 Basic Linear Algebra Problems

- Linear Equations: Solve $Ax=b$ for x
- Least Squares: Find x that minimizes $\|r\|_2 \equiv \sqrt{\sum r_i^2}$ where $r = Ax - b$
 - Statistics: Fitting data with simple functions
- Eigenvalues: Find λ and x where $Ax = \lambda x$
 - Vibration analysis, e.g., earthquakes, circuits
- Singular Value Decomposition: $A^T Ax = \sigma^2 x$
 - Data fitting, Information retrieval

Lots of variations depending on structure of A

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

17

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
 - LINPACK Benchmark (www.top500.org)
 - "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

18

Winner of TOPS 500 (LINPACK Benchmark)

Year	Machine	Tflops	Factor faster	Peak Tflops	Num Procs	N
2004	Blue Gene / L, IBM	70.7	2.0	91.8	32768	.93M
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

Source: Jack Dongarra (UTK)

19

Current Records for Solving Small Dense Systems

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

Machine	Megaflops		
	n=100	n=1000	Peak
NEC SX 8 (8 proc, 2 GHz) (1 proc, 2 GHz)		75140 14960	128000 16000
...			
Palm Pilot III	.00169		

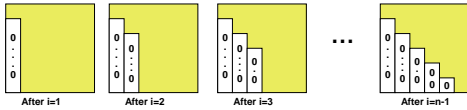
20

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)
    
```



21

Refine GE Algorithm (1)

- Initial Version

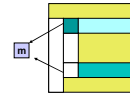
```

... 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)
    
```



22

Refine GE Algorithm (2)

- Last version

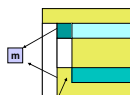
```

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

23

Refine GE Algorithm (3)

- Last version

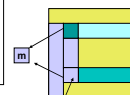
```

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

24

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

25

Refine GE Algorithm (5)

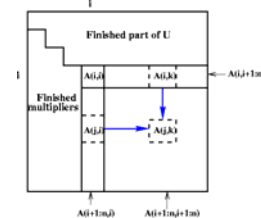
Last version

```

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 i 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)
  
```

26

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

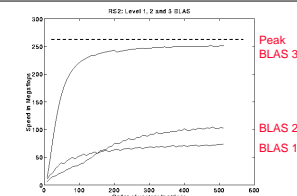
27

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)
  
```



28

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

29

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/sqetf2.f>

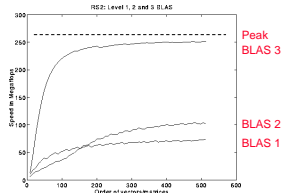
30

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)
    
```



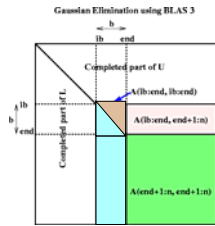
Converting BLAS2 to BLAS3 in GEPP

- **Blocking**
 - Used to optimize matrix-multiplication
 - Harder here because of data dependencies in GEPP
- **BIG IDEA: Delayed Updates**
 - Save updates to "trailing matrix" from several consecutive BLAS2 updates
 - Apply many updates simultaneously in one BLAS3 operation
- Same idea works for much of dense linear algebra
 - Open questions remain
- **First Approach: 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)

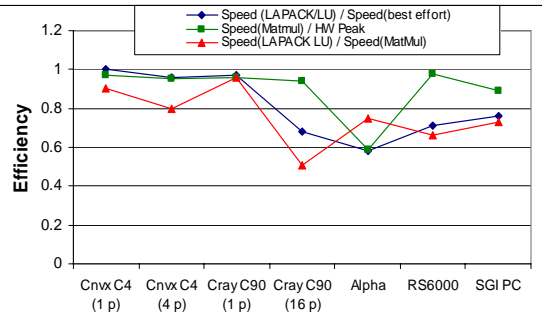
```

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^T * L^T * U^T
... 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-line notes from CS267 / 1996.)

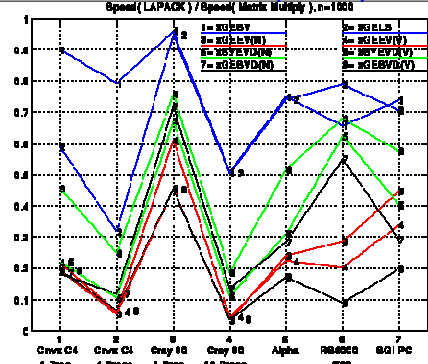
Efficiency of Blocked GEPP



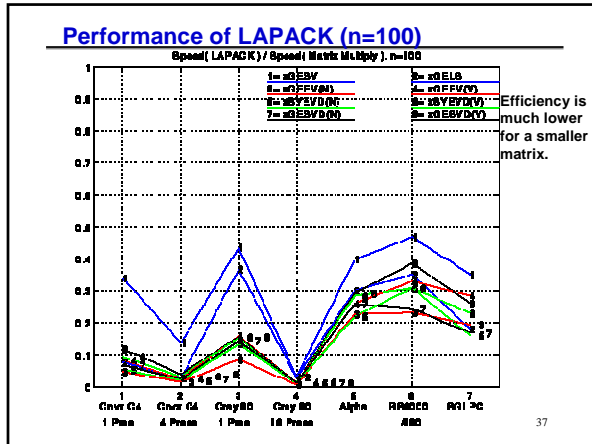
Overview of LAPACK and ScaLAPACK

- 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, Matlab ...
- Many algorithmic innovations remain
 - Projects available

Performance of LAPACK (n=1000)



Performance of Eigenvalues, SVD, etc.



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)

$A(i+1:n,i+1:n) = A(i+1:n, i+1:n) - A(i+1:n, i) * A(i, i+1:n)$... BLAS 2 (rank-1 update)

- Assignment
 - Which processors are responsible for which submatrices?

38

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

39

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

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

Machine	Proces	Block Size	Speed in Mflops of PDGEMM		
			N=2000	N=4000	N=10000
Cray T3E	4=2x2	32	1055	1070	0
	16=4x4	32	3630	4005	4292
	64=8x8	32	13456	14267	16755
IBM SP2	4	50	755	0	0
	16	50	2514	2850	0
	64	50	6205	6709	10774
Intel XT/S MP*	4	32	330	0	0
	16	32	1233	1281	0
	64	32	4496	4864	5257
Berkeley NOW	4	32	463	470	0
	16	32	2490	2822	3450
	64	32	4130	5457	6647

Machine	Peak/proc	DGEMM Mflops	Efficiency		
			N=2000	N=4000	N=10000
Cray T3E	600	360	4	.73	.74
	16	360	.63	.70	.75
	64	360	.58	.62	.73
IBM SP2	266	200	4	.94	
	16	200	.79	.89	
	64	200	.68	.84	
Intel XT/S MP	100	90	4	.92	
	16	90	.86	.89	
	64	90	.78	.84	.91
Berkeley NOW	334	129	4	.90	.91
	16	129	.60	.68	.84
	64	129	.50	.66	.81

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 $A(ib:n, ib:end) = P * L * U$

... 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

$A(end+1:n, end+1:n) = A(end+1:n, end+1:n)$... update next b rows of U

$A(end+1:n, ib:end) = A(ib:end, end+1:n)$... apply delayed updates with single matrix-multiply

... with inner dimension b

41

Row and Column Block Cyclic Layout

- processors and matrix blocks are distributed in a 2d array
- proW-by-pcol array of processors
- broW-by-bcol matrix blocks

- pcol-fold parallelism in any column, and calls to the BLAS2 and BLAS3 on matrices of size brow-by-bcol

- serial bottleneck is eased

- proW \neq pcol and brow \neq bcol possible, even desirable

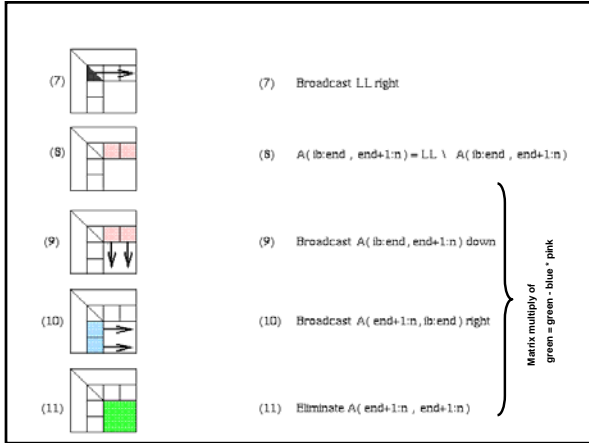
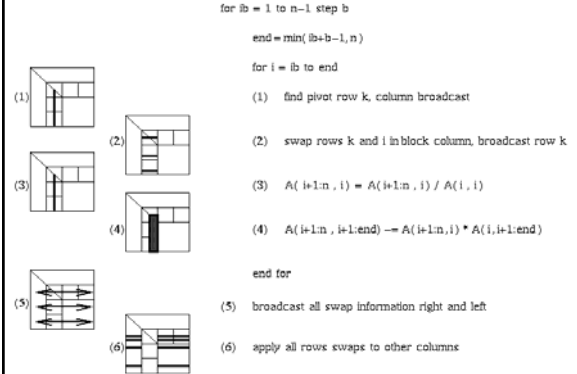
42

Distributed GE with a 2D Block Cyclic Layout

- block size b in the algorithm and the block sizes b_{row} and b_{col} in the layout satisfy $b=b_{col}$.
- 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

43

Distributed Gaussian Elimination with a 2D Block Cyclic Layout



LAPACK and ScaLAPACK Status

- “One-sided Problems” are scalable
 - In Gaussian elimination, A factored into product of 2 matrices $A = LU$ by premultiplying A by sequence of simpler matrices
 - Asymptotically 100% BLAS3
 - LU (“Linpack Benchmark”)
 - Cholesky, QR
- “Two-sided Problems” are harder
 - A factored into product of 3 matrices by pre and post multiplication
 - Half BLAS2, not all BLAS3
 - Eigenproblems, SVD
 - Nonsymmetric eigenproblem hardest
- Narrow band problems hardest (to do BLAS3 or parallelize)
 - Solving and eigenproblems
- www.netlib.org/lapack/sca lapack

46

ScaLAPACK Performance Models (1)

ScaLAPACK Operation Counts

$$T(N, P) = \frac{C_f N^3}{P} + \frac{C_m N^2}{\sqrt{P}} + \frac{C_n N}{NB} + \tau_{com} \quad T_{seq}(N, P) = C_f N^3 + \tau_{com}$$

$$E(N, P) = \left(1 + \frac{1}{NB} \frac{C_m}{C_f} \frac{P}{N^2} + \frac{C_n}{C_f} \frac{\sqrt{P}}{N} \right)^{-1}$$

$$t_f = 1$$

$$t_m = \alpha$$

$$t_n = \beta$$

$$NB = \text{brow} = \text{bcol}$$

$$\sqrt{P} = \text{prow} = \text{pcol}$$

Driver	Options	C_f	C_m	C_n
PzCGESV	1 right hand side	$2/3$	$3 + 1/4 \log_2 P$	$NB(3 + \log_2 P)$
PzCGBV	1 right hand side	$1/3$	$3 + 1/2 \log_2 P$	$4 + \log_2 P$
PzGELS	1 right hand side	$4/3$	$3 + \log_2 P$	$2(NB \log_2 P - 1)$
PzSYEVX	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2 NB - 2$
PzSYEVX	eigenvalues and eigenvectors	$10/3$	$5 \log_2 P$	$17/2 NB - 2$
PzSYEV	eigenvalues only	$4/3$	$5/2 \log_2 P$	$17/2 NB - 2$
PzSYEV	eigenvalues and eigenvectors	$12/3$	$5 \log_2 P$	$17/2 NB - 2$
PzGESVD	singular values only	$26/3$	$10 \log_2 P$	$17NB$
PzGESVD	singular values and left and right singular vectors	$38/3$	$14 \log_2 P$	$17NB$
PzAHQR	eigenvalues only	5	$5/2(\sqrt{P}) + \log_2 P$	$5(2 + \log_2 P)N$
PzAHQR	full Schur form	18	$5/2(\sqrt{P}) + \log_2 P$	$5(2 + \log_2 P)N$

47

ScaLAPACK Performance Models (2)

Compare Predictions and Measurements

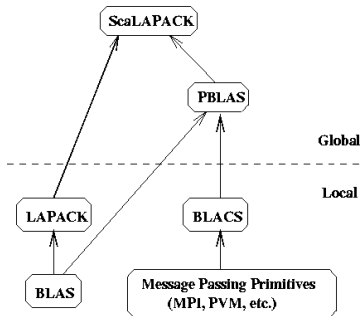
TRW SP2 ^a	P	Values of N									
		5000		5000		7600		10000		15000	
		Est	Obs	Est	Obs	Est	Obs	Est	Obs	Est	Obs
PDGESV (LU)	4	367	421	632	603						
	16	487	722	1381	1543	2119	1909	2454	2145		
	64	832	924	2432	3017	4235	4295	5763	5666	7392	7067
PLPGBV (Cholesky)	4	530	462	968	615						
	16	1312	1081	2088	1811	2367	2118	2830	2812		
	64	2577	1807	5227	4431	6709	6727	7821	6828	8087	8064

^aOne process spanned per node and one computational IBM POWERPC 680 processor per node.

48

ScaLAPACK Overview

ScaLAPACK SOFTWARE HIERARCHY



49

PDGESV = ScaLAPACK Parallel LU

Performance of ScaLAPACK LU

Since it can run no faster than its inner loop (PDGEMM), we measure:
Efficiency = Speed(PDGESV)/Speed(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 $(2/3)n^3 / (2/3)n^3 = 3$ times faster, but communication makes it a little slower.

Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.67	.82	.84
	16		.44	.65	.75
	64		.18	.47	
IBM SP2	4	50	.56	.59	.66
	16		.29	.32	
	64		.15	.32	
Intel XP/S MP ²	4	32	.64	.64	.75
	16		.37	.66	
	64		.16	.42	
Berkeley NOW	4	32	.76	.69	.71
	16		.38	.62	
	64		.28	.54	.69

Machine	Procs	Block Size	Time(PDGESV)/Time(PDGEMM)		
			2000	4000	10000
Cray T3E	4	32	.50	.40	.40
	16		.75	.51	.40
	64		1.86	.72	.45
IBM SP2	4	50	.60	1.16	.64
	16		2.24	1.03	.51
	64				
Intel XP/S GP ²	4	32	.52	.50	.44
	16		.89	.50	.47
	64		2.08	.79	.44
Berkeley NOW	4	32	.44	.47	.47
	16		.88	.54	.47
	64		1.18	.62	.49

QR (Least Squares)

Performance of ScaLAPACK QR (Least squares)

Scales well,
nearly full machine speed

Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.54	.61	.60
	16		.46	.55	.54
	64		.26	.47	
IBM SP2	4	50	.51	.51	.54
	16		.29	.51	
	64		.19	.36	
Intel XP/S GP ²	4	32	.61	.63	.62
	16		.43	.63	
	64		.22	.48	
Berkeley NOW	4	32	.51	.77	.71
	16		.49	.66	
	64		.37	.60	.72

Machine	Procs	Block Size	Time(PDQRSL)/Time(PDGEMM)		
			2000	4000	10000
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	1.2
	16		2.3	1.3	1.2
	64		3.6	1.8	1.2
Intel XP/S GP ²	4	32	1.1	1.1	1.1
	16		1.6	1.4	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

Performance of Symmetric Eigensolvers

Current algorithm:
Faster than initial algorithm
Occasional numerical instability
New, faster and more stable algorithm planned

Machine	Procs	Block Size	N	
			2000	4000
Cray T3E	4	32	10	10
	16		13	14
	64		29	14
IBM SP2	4	50	24	29
	16		40	29
	64		40	29
Intel XP/S GP ²	4	32	22	20
	16		34	20
	64		34	20
Berkeley NOW	4	32	20	24
	16		24	24
	32		24	52

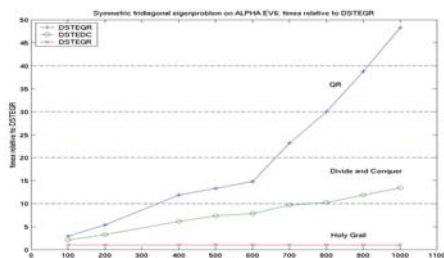
Initial algorithm:
Numerically stable
Easily parallelized
Slow: will abandon

Machine	Procs	Block Size	N	
			2000	4000
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 ²	4	32	99	193
	16		31	31
	64		31	31
Berkeley NOW	4	32	35	55
	16		35	55
	32		35	55

52

Scalable Symmetric Eigensolver and SVD

The "Holy Grail" (Parlett, Dhillon, Marques)
Perfect Output complexity $O(n * \#vectors)$, Embarrassingly parallel, Accurate



53

Performance of SVD (Singular Value Decomposition)

Have good ideas to speedup
Project available!

Machine	Procs	Block Size	N	
			2000	4000
Cray T3E	4	32	67	64
	16		66	64
	64		93	70
IBM SP2	4	50	97	60
	16		60	81
	64		81	
Berkeley NOW	4	32	72	16
	16		38	16
	32		59	26

Performance of Nonsymmetric Eigensolver (QR iteration)

Hardest of all to parallelize

Machine	Procs	Block Size	N	
			1000	1500
Intel XP/S MP ²	16	50	123	97
	Paragon			

54

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
 - $A_{i,i+1} = (A_i + A_{i+1}^{-1})/2 \rightarrow$ 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

55

Out of "Core" Algorithms

Out-of-Core Performance Results for Least Squares

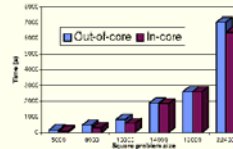
- Prototype code for Out-of-Core extension
- Linear solvers based on "Left-looking" variants of LU, QR, and Cholesky factorization
- Portable I/O interface for reading/writing ScaLAPACK matrices

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

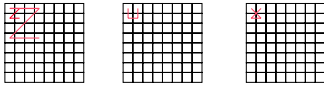
QR Factorization on 64 processors Intel Paragon



Source: Jack Dongarra

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/>

57

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 $n/2, n/4, n/8, \dots$

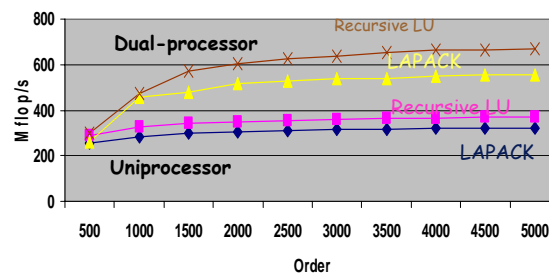
Source: Jack Dongarra

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).

59

Pentium III 550 MHz Dual Processor LU Factorization



Source: Jack Dongarra

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)

61

Next release of LAPACK and ScaLAPACK

- Class projects available
- www.cs.berkeley.edu/~demmel/Sca-LAPACK-Proposal.pdf
- New or improved LAPACK algorithms
 - Faster and/or more accurate routines for linear systems, least squares, eigenvalues, SVD
- Parallelizing algorithms for ScaLAPACK
 - Many LAPACK routines not parallelized yet
- Automatic performance tuning
 - Many tuning parameters in code

62

Some contributors (incomplete list)

Participants

Krzysztof Anasiewicz (UC Berkeley)	Zhaojun Bai (U Kentucky)
Richard Barrett (U. Tenn)	Michael Berry (U Tenn)
Jeff Bilmes (UC Berkeley)	Chris Bischof (ANL)
Steve Blackford (ORNL)	Sourav Chakrabarti (UC Berkeley)
Tony Chan (UCLA)	Chao-Whya Chin (UC Berkeley)
Jiapeng Chai (LBNL)	Andy Cleary (LENL)
Eli D'Azevedo (ORNL)	Jim Demmel (UC Berkeley)
Indrajit Dhillon (UC Berkeley)	Junzo Donato (ORNL)
Jack Dongarra (U. Tenn, ORNL)	Zlatko Drmač (U Hagen)
Jeremy Du Crocq (NAG)	Victor Eijkhout (UCLA)
Stan Eisenstat (Yale)	Vince Ferraro (NAG)
John Gilbert (Xerox PARC)	Ming Gu (UC Berkeley, LBL)
Sven Hammarling (NAG)	Mike Heath (U Illinois)
Greg Henry (Intel)	Dominik Kuhn (UC Berkeley)
Steve Huss-Lederman (SRC)	Bo Kågström (U Umeå)
W. Kahan (UC Berkeley)	Yongqiang Kim (U Tenn)
Hansang Li (UC Berkeley)	Xinyao Li (UC Berkeley)
Joseph Liu (York)	Bennett F. Poullet (UC Berkeley)
Antoine Puellet (U. Tenn)	Peter Puzosana (U Umeå)
Hokan Fonn (U Tenn)	Padma Raghavan (U Illinois)
Huan Ren (UC Berkeley)	Howard Robinson (UC Berkeley)
Charles Runtz (ORNL)	Jeff Ritter (UC Berkeley)
Ivan Sgambelza (U Spili)	Dan Sorensen (Rice U)
Ken Stanley (UC Berkeley)	Xinshui Sun (ANL)
Bernard Teunischout (U Tenn)	Anna Tsoi (SRC)
Robort van de Geijn (U Texas)	Henk van der Vorst (Utrecht U)
Paul Van Dooren (U Hagen)	Kristine Vassili (U Hagen)
David Walker (ORNL)	Clint Whaley (U Tenn)
Kathy Yalch (UC Berkeley)	

With the cooperation of:
Cray, IBM, Compaq, EEC, Fujitsu, NRC, NAG, EMSL

Supported by ARPA, NSF, DOE

63