

CS 267 Parallel Matrix Multiplication

Kathy Yelick

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

3/9/2004

CS267 Lecture 13

1

Parallel Numerical Algorithms

- Lecture schedule:
 - 3/8: Dense Matrix Products
 - BLAS 1: Vector operations
 - BLAS 2: Matrix-Vector operations
 - BLAS 3: Matrix-Matrix operations
 - Use of Performance models in algorithm design
 - 3/10: Dense Matrix Solvers
 - 3/12: Matrix Multiply context results (HW1)
 - 310 Soda at 1:30pm
 - 3/15: Sparse Matrix Products
 - 3/17: Sparse Direct Solvers

3/9/2004

CS267 Lecture 13

2

Parallel Vector Operations

Some common vector operations for vectors x, y, z :

- Vector add: $z = x + y$
 - Trivial to parallelize if vectors are aligned
- AXPY: $z = a * x + y$ (where a is scalar)
 - Broadcast a , followed by independent $*$ and $+$
- Dot product: $s = \sum_j x[j] * y[j]$
 - Independent $*$ followed by $+$ reduction

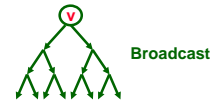
3/9/2004

CS267 Lecture 13

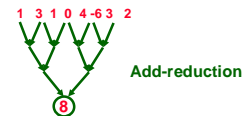
3

Broadcast and reduction

- Broadcast of 1 value to p processors in $\log p$ time



- Reduction of p values to 1 in $\log p$ time
- Takes advantage of associativity in $+$, $*$, \min , \max , etc.



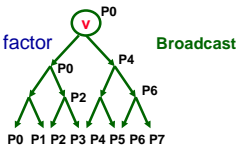
3/9/2004

CS267 Lecture 13

4

Broadcast Algorithms

- Sequential or "centralized" algorithm
 - P_0 sends value to $P-1$ other processors in sequence
 - $O(P)$ algorithm
 - Note: variations in UPC/Titanium model based on whether P_0 writes to all others, or others read from P_0
- Tree-based algorithm
 - May vary branching factor
 - $O(\log P)$ algorithm



- If broadcasting large data blocks, may break into pieces and pipeline

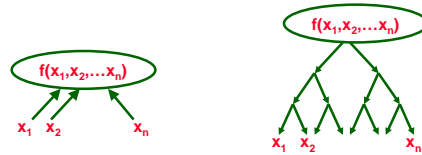
3/9/2004

CS267 Lecture 13

5

Lower Bound on Parallel Performance

- To compute a function of n inputs x_1, \dots, x_n
- Given only binary operations on our machine.
 - In 1 time step, output depends on at most 2 inputs
 - In 2 time steps, output depends on at most 4 inputs
 - Adding a time step increases possible inputs by at most $2x$
 - In $k = \log n$ time steps, output depends on at most n inputs
- A function of n inputs requires at least $\log n$ parallel steps.



3/9/2004

CS267 Lecture 13

6

E.g., Fibonacci via Matrix Multiply Prefix

$$F_{n+1} = F_n + F_{n-1}$$

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

Can compute all F_n by matmul_prefix on

$$\left[\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right]$$

then select the upper left entry

3/9/2004

CS267 Lecture 13 Slide source: Alan Edelman, MIT

Segmented Operations

Inputs = Ordered Pairs
(operand, boolean)
e.g. (x, T) or (x, F)

Change of
segment indicated
by switching T/F

+ 2	(y, T)	(y, F)
(x, T)	(x+y, T)	(y, F)
(x, F)	(y, T)	(x⊕y, F)

e. g.

1	2	3	4	5	6	7	8	
T	T	F	F	F	T	F	T	
Result	1	3	3	7	12	6	7	8

3/9/2004

CS267 Lecture 13

14

The "Myth" of log n

- The $\log_2 n$ parallel steps is **not** the main reason for the usefulness of parallel prefix.
- Say $n = 1000p$ (1000 summands per processor)
 - Cost = (2000 adds) + ($\log_2 P$ message passings)

↑
fast & embarassingly parallel
(2000 local adds are serial for each processor of course)

3/9/2004

CS267 Lecture 13

15

End of Digression

- Summary of data parallel operations
 - Vector add, etc. is embarrassingly parallel
 - Broadcast used for axpy operations
 - Reduction used for dot product
 - Parallel prefix (scan) is a variation on reduction with partial results
 - Useful in parallelizing surprising algorithms
 - If something seems serial, try this
- Now back to our regular programming
 - We have covered the idea with most BLAS1 (vector) operations
 - Now onto vector/matrix (BLAS2) and matrix-matrix (BLAS3)

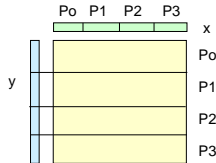
3/9/2004

CS267 Lecture 13

16

Parallel Matrix-Vector Product

- Compute $y = y + A \cdot x$, where A is a dense matrix



- Layout:
 - 1D by rows
- Algorithm:
 - Foreach processor i
 - Broadcast $x(i)$
 - Compute $y(i) = A(i) \cdot x$
- $A(i)$ refers to the n by n/p block row that processor i owns, $x(i)$ and $y(i)$ similarly refer to segments of x, y owned by i
- Algorithm uses the formula

$$y(i) = y(i) + A(i) \cdot x = y(i) + \sum_j A(i)_j \cdot x(j)$$

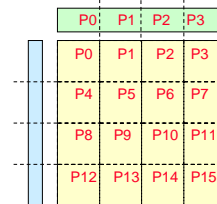
3/9/2004

CS267 Lecture 13

17

Matrix-Vector Product

- A column layout of the matrix eliminates the broadcast
 - But adds a reduction to update the destination
- A blocked layout uses a broadcast and reduction, both on a subset of processors
 - \sqrt{p} for square processor grid



3/9/2004

CS267 Lecture 13

18

Parallel Matrix Multiply

- Computing $C=A*B$
- Using basic algorithm: $2*n^3$ Flops
- Variables are:
 - Data layout
 - Topology of machine
 - Scheduling communication
- Use of performance models for algorithm design
 - **Message Time** = "latency" + #words * time-per-word
 $= \alpha + n*\beta$

3/9/2004

CS267 Lecture 13

19

Latency Bandwidth Model

- Network of fixed number P of processors
 - fully connected
 - each with local memory
- Latency (α)
 - accounts for varying performance with number of messages
 - gap (g) in logP model may be more accurate cost if messages are pipelined
- Inverse bandwidth (β)
 - accounts for performance varying with volume of data
- Efficiency (in any model):
 - serial time / (p * parallel time)
 - perfect (linear) speedup \rightarrow efficiency = 1

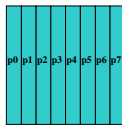
3/9/2004

CS267 Lecture 13

20

Matrix Multiply with 1D Column Layout

- Assume matrices are $n \times n$ and n is divisible by p



May be a reasonable assumption for analysis, not for code

- $A(i)$ refers to the n by n/p block column that processor i owns (similarly for $B(i)$ and $C(i)$)
- $B(i,j)$ is the n/p by n/p subblock of $B(i)$
 - in rows $j*n/p$ through $(j+1)*n/p$
- Algorithm uses the formula
 $C(i) = C(i) + A*B(i) = C(i) + \sum_j A(j)*B(j,i)$

3/9/2004

CS267 Lecture 13

21

Matrix Multiply: 1D Layout on Bus or Ring

- Algorithm uses the formula
 $C(i) = C(i) + A*B(i) = C(i) + \sum_j A(j)*B(j,i)$
- First consider a bus-connected machine without broadcast: only one pair of processors can communicate at a time (ethernet)
- Second consider a machine with processors on a ring: all processors may communicate with nearest neighbors simultaneously

3/9/2004

CS267 Lecture 13

22

MatMul: 1D layout on Bus without Broadcast

Naïve algorithm:

```

C(myproc) = C(myproc) + A(myproc)*B(myproc,myproc)
for i = 0 to p-1
  for j = 0 to p-1 except i
    if (myproc == i) send A(i) to processor j
    if (myproc == j)
      receive A(i) from processor i
      C(myproc) = C(myproc) + A(i)*B(i,myproc)
  barrier
    
```

Cost of inner loop:

computation: $2*n*(n/p)^2 = 2*n^3/p^2$
 communication: $\alpha + \beta*n^2/p$

3/9/2004

CS267 Lecture 13

23

Naïve MatMul (continued)

Cost of inner loop:

computation: $2*n*(n/p)^2 = 2*n^3/p^2$
 communication: $\alpha + \beta*n^2/p$... approximately

Only 1 pair of processors (i and j) are active on any iteration, and of those, only i is doing computation
 \Rightarrow the algorithm is almost entirely serial

Running time:

$= (p*(p-1) + 1)*\text{computation} + p*(p-1)*\text{communication}$
 $\sim 2*n^3 + p^2*\alpha + p*n^2*\beta$

this is worse than the serial time and grows with p

3/9/2004

CS267 Lecture 13

24

Matmul for 1D layout on a Processor Ring

- Pairs of processors can communicate simultaneously

```
Copy A(myproc) into Tmp
C(myproc) = C(myproc) + Tmp*B(myproc, myproc)
for j = 1 to p-1
    Send Tmp to processor myproc+1 mod p
    Receive Tmp from processor myproc-1 mod p
    C(myproc) = C(myproc) + Tmp*B(myproc-j mod p, myproc)
```

- Same idea as for gravity in simple sharks and fish algorithm
 - May want double buffering in practice for overlap
 - Ignoring deadlock details in code
- Time of inner loop = $2^*(\alpha + \beta*n^2/p) + 2^n*(n/p)^2$

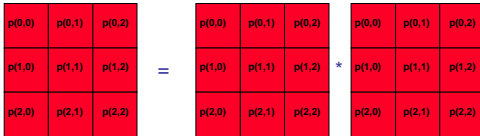
Matmul for 1D layout on a Processor Ring

- Time of inner loop = $2^*(\alpha + \beta*n^2/p) + 2^n*(n/p)^2$
- Total Time = $2^n*n^*(n/p)^2 + (p-1) * \text{Time of inner loop}$
 - $\sim 2^n*n^3/p + 2^n*p*\alpha + 2^n*\beta*n^2$
- Optimal for 1D layout on Ring or Bus, even with with Broadcast:
 - Perfect speedup for arithmetic
 - A(myproc) must move to each other processor, costs at least $(p-1)*\text{cost of sending } n^*(n/p) \text{ words}$

- Parallel Efficiency = $2^n*n^3 / (p * \text{Total Time})$
 - $= 1/(1 + \alpha * p^2/(2^n*n^3) + \beta * p/(2^n*n))$
 - $= 1/(1 + O(p/n))$
- Grows to 1 as n/p increases (or α and β shrink)

MatMul with 2D Layout

- Consider processors in 2D grid (physical or logical)
- Processors can communicate with 4 nearest neighbors
 - Broadcast along rows and columns

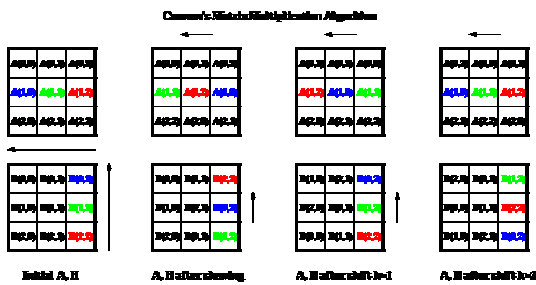


- Assume p is square s x s grid

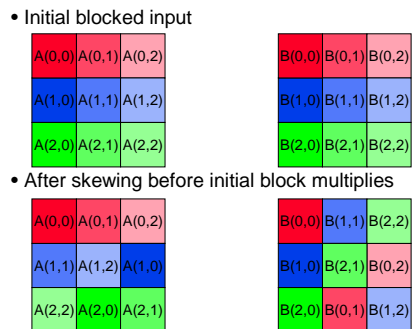
Cannon's Algorithm

```
... C(i,j) = C(i,j) + \sum_k A(i,k)*B(k,j)
... assume s = sqrt(p) is an integer
for i=0 to s-1 ... "skew" A
    left-circular-shift row i of A by i
    ... so that A(i,j) overwritten by A(i,(j+i)mod s)
for i=0 to s-1 ... "skew" B
    up-circular-shift B column i of B by i
    ... so that B(i,j) overwritten by B((i+j)mod s, j)
for k=0 to s-1 ... sequential
    for i=0 to s-1 and j=0 to s-1 ... all processors in parallel
        C(i,j) = C(i,j) + A(i,j)*B(i,j)
    left-circular-shift each row of A by 1
    up-circular-shift each row of B by 1
```

Cannon's Matrix Multiplication



Initial Step to Skew Matrices in Cannon



- Initial blocked input
- After skewing before initial block multiplies

Skewing Steps in Cannon

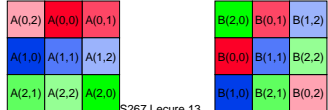
• First step



• Second



• Third



3/9/2004

CS267 Lecture 13

31

Cost of Cannon's Algorithm

```

forall i=0 to s-1
    ... recall s = sqrt(p)
    left-circular-shift row i of A by i ... cost = s*(alpha + beta*n^2/p)
forall i=0 to s-1
    up-circular-shift B column i of B by i ... cost = s*(alpha + beta*n^2/p)
for k=0 to s-1
    forall i=0 to s-1 and j=0 to s-1
        C(i,j) = C(i,j) + A(i,i)*B(i,j) ... cost = 2*(n/s)^3 = 2*n^3/p^3/2
        left-circular-shift each row of A by 1 ... cost = alpha + beta*n^2/p
        up-circular-shift each row of B by 1 ... cost = alpha + beta*n^2/p
    
```

- Total Time = $2*n^3/p + 4*s*\alpha + 4*\beta*n^2/s$
- Parallel Efficiency = $2*n^3 / (p * \text{Total Time})$
 $= 1 / (1 + \alpha * 2*(s/n)^3 + \beta * 2*(s/n))$
 $= 1 / (1 + O(\sqrt{p}/n))$
- Grows to 1 as $n/s = n/\sqrt{p} = \sqrt{\text{data per processor}}$ grows
- Better than 1D layout, which had Efficiency = $1 / (1 + O(p/n))$

3/9/2004

CS267 Lecture 13

32

Drawbacks to Cannon

- Hard to generalize for
 - p not a perfect square
 - A and B not square
 - Dimensions of A, B not perfectly divisible by $s=\sqrt{p}$
 - A and B not "aligned" in the way they are stored on processors
 - block-cyclic layouts
- Memory hog (extra copies of local matrices)

3/9/2004

CS267 Lecture 13

33

SUMMA Algorithm

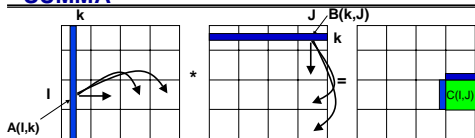
- SUMMA = Scalable Universal Matrix Multiply
- Slightly less efficient, but simpler and easier to generalize
- Presentation from van de Geijn and Watts
 - www.netlib.org/lapack/lawns/lawn96.ps
 - Similar ideas appeared many times
- Used in practice in PBLAS = Parallel BLAS
 - www.netlib.org/lapack/lawns/lawn100.ps

3/9/2004

CS267 Lecture 13

34

SUMMA



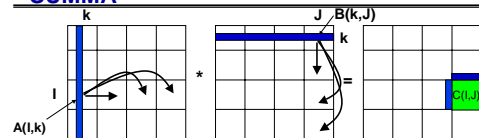
- I, J represent all rows, columns owned by a processor
- k is a single row or column
 - or a block of b rows or columns
- $C(I, J) = C(I, J) + \sum_k A(I, k) * B(k, J)$
- Assume a p_r by p_c processor grid ($p_r = p_c = 4$ above)
 - Need not be square

3/9/2004

CS267 Lecture 13

35

SUMMA



```

For k=0 to n-1 ... or n/b-1 where b is the block size
    ... = # cols in A(I,k) and # rows in B(k,J)
    for all I = 1 to p_r ... in parallel
        owner of A(I,k) broadcasts it to whole processor row
    for all J = 1 to p_c ... in parallel
        owner of B(k,J) broadcasts it to whole processor column
    Receive A(I,k) into Acol
    Receive B(k,J) into Brow
    C( myproc , myproc ) = C( myproc , myproc ) + Acol * Brow
    
```

3/9/2004

CS267 Lecture 13

36

SUMMA performance

- To simplify analysis only, assume $s = \sqrt{p}$

```

For k=0 to n/b-1
  for all I = 1 to s ... s = sqrt(p)
    owner of A(I,k) broadcasts it to whole processor row
    ... time = log s * (alpha + beta * b * n/s), using a tree
  for all J = 1 to s
    owner of B(k,J) broadcasts it to whole processor column
    ... time = log s * (alpha + beta * b * n/s), using a tree
  Receive A(I,k) into Acol
  Receive B(k,J) into Brow
  C(myproc, myproc) = C(myproc, myproc) + Acol * Brow
  ... time = 2*(n/s)^2 * b
    
```

Total time = $2 * n^3 / p + \alpha * \log p * n / b + \beta * \log p * n^2 / s$

3/9/2004

CS267 Lecture 13

37

SUMMA performance

- Total time = $2 * n^3 / p + \alpha * \log p * n / b + \beta * \log p * n^2 / s$
- Parallel Efficiency = $1 / (1 + \alpha * \log p * p / (2 * \beta * n^2) + \beta * \log p * s / (2 * n))$
- ~Same β term as Cannon, except for $\log p$ factor
 $\log p$ grows slowly so this is ok
- Latency (α) term can be larger, depending on b
 When $b=1$, get $\alpha * \log p * n$
 As b grows to n/s , term shrinks to $\alpha * \log p * s$ ($\log p$ times Cannon)
- Temporary storage grows like $2 * b * n / s$
- Can change b to tradeoff latency cost with memory

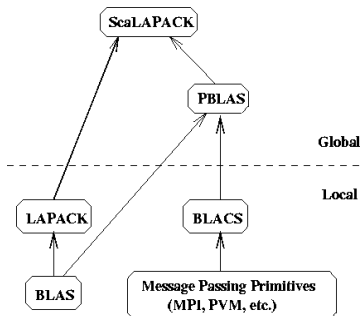
3/9/2004

CS267 Lecture 13

38

ScaLAPACK Parallel Library

ScaLAPACK SOFTWARE HIERARCHY



3/9/2004

CS267 Lecture 13

39

Performance of PBLAS

Machine	Speed in Mflops of PDGEMM		N		
	Procs	Block Size	2000	4000	10000
Cray T3E	4=2x2	32	1055	1070	0
	16=4x4		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 Passport	4	32	330	0	0
	16		1233	1281	0
	64		4496	4864	5257
Berkeley NOW	4	32	463	470	0
	32=4x8		2490	2822	3450
	64		4130	5457	6647

PDGEMM = PBLAS routine for 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 (same as above):
 Efficiency always at least 48%
 For fixed N, as P increases, efficiency drops
 For fixed P, as N increases, efficiency increases

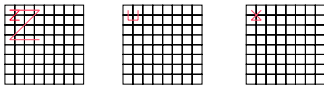
Machine	Peak/proc	DGEMM Mflops	Procs	N		
				2000	4000	10000
Cray T3E	600	360	4	.73	.74	
			16	.63	.70	.75
			64	.58	.62	.73
IBM SP2	266	200	4	.94		
			16	.79	.89	
			64	.48	.68	.84
Intel XP/S MP Passport	100	90	4	.92		
			16	.86	.89	
			64	.78	.84	.91
Berkeley NOW	334	129	4	.90	.91	
			32	.60	.68	.84
			64	.50	.66	.81

3/9/2004

40

Recursive Layouts

- For both cache hierarchies and parallelism, recursive layouts may be useful
- Z-Morton, U-Morton, and X-Morton Layout



- Also Hilbert layout and others
- What about the user's view?
 - Fortunately, many problems can be solved on a permutation
 - Never need to actually change the user's layout

3/9/2004

CS267 Lecture 13

41

Summary of Parallel Matrix Multiplication

- 1D Layout
 - Bus without broadcast - slower than serial
 - Nearest neighbor communication on a ring (or bus with broadcast): Efficiency = $1 / (1 + O(p/n))$
- 2D Layout
 - Cannon
 - Efficiency = $1 / (1 + O(\sqrt{p}/n))$
 - Hard to generalize for general p , n , block cyclic, alignment
 - SUMMA
 - Efficiency = $1 / (1 + O(\log p * p / (b * n^2) + \log p * \sqrt{p}/n))$
 - Very General
 - b small => less memory, lower efficiency
 - b large => more memory, high efficiency
 - Recursive layouts

3/9/2004

CS267 Lecture 13

42

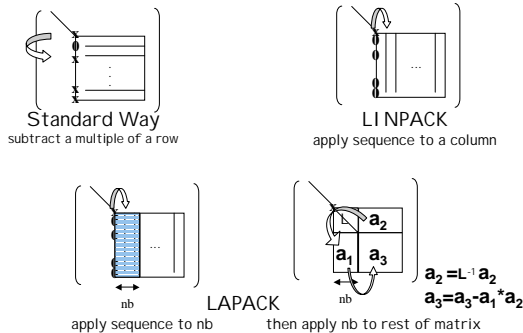
Extra Slides

3/9/2004

CS267 Lecture 13

43

Gaussian Elimination



3/9/2004

CS267 Lecture 13

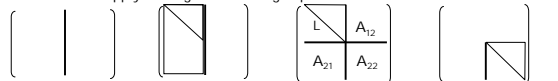
Slide source: Dongarra 44

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

3/9/2004

CS267 Lecture 13

Slide source: Dongarra 45

Recursive Factorizations

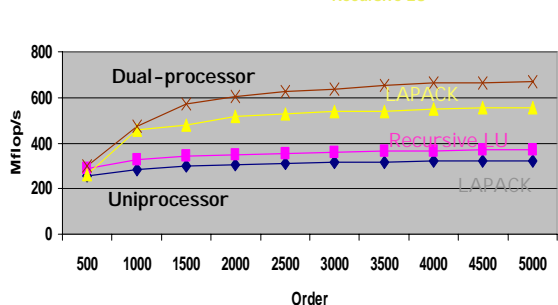
- Just as accurate as conventional method
- Same number of operations
- Automatic variable blocking
 - Level 1 and 3 BLAS only!
- Extreme clarity and simplicity of expression
- Highly efficient
- The recursive formulation is just a rearrangement of the point-wise LINPACK algorithm
- The standard error analysis applies (assuming the matrix operations are computed the "conventional" way).

3/9/2004

CS267 Lecture 13

Slide source: Dongarra 46

Pentium III 550 MHz Dual Processor LU Factorization

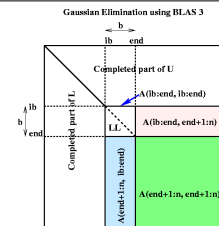


Slide source: Dongarra

Review: BLAS 3 (Blocked) GEPP

```

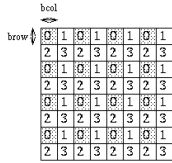
for ib = 1 to n-1 step b ... Process matrix b columns at a time
end = ib + b-1 ... Point to end of block of b columns
apply BLAS2 version of GEPP to get A(ib:n, ib:end) = P * L * U
... let LL denote the strict lower triangular part of A(ib:end, ib:end) + I
BLAS 3 { A(ib:end, end+1:n) = LL * A(ib:end, 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
    
```



3/9/2004

48

Review: 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- $bcol$

4) Row and Column Block Cyclic Layout

serial bottleneck is eased

need not be symmetric in rows and columns

3/9/2004

CS267 Lecture 13

49

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 busy processors or communication performed.

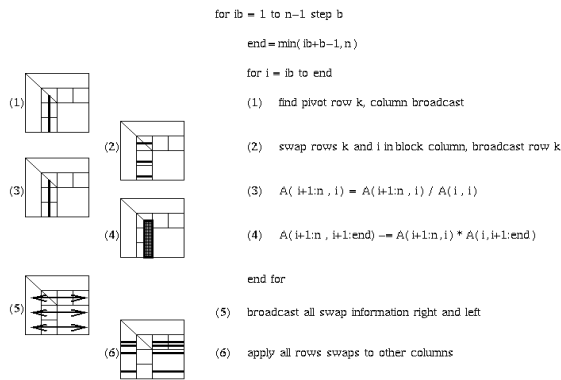
unnecessary to have a barrier between each step of the algorithm, e.g.. step 9, 10, and 11 can be pipelined

3/9/2004

CS267 Lecture 13

50

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 l in block column, broadcast row k

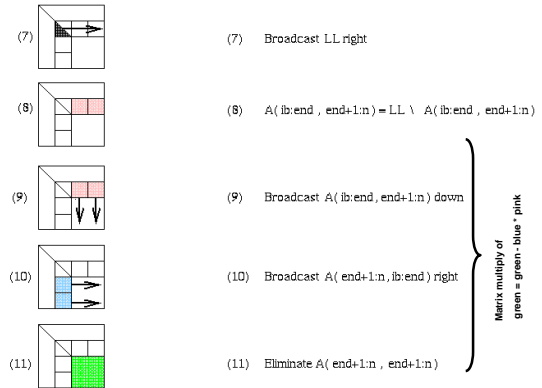
(3) $A(i+1:n, i) = A(i+1:n, i) / A(i, i)$

(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) $A(ib:end, end+1:n) = LL \ A(ib:end, end+1:n)$

(9) Broadcast $A(ib:end, end+1:n)$ down

(10) Broadcast $A(end+1:n, ib:end)$ right

(11) Eliminate $A(end+1:n, end+1:n)$

Matrix multiply of
green = green * blue * pink

Performance of ScaLAPACK LU

Efficiency = $M\text{Flops}(\text{PDGESSV}) / M\text{Flops}(\text{PDGEMM})$

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	50	.56		
	16		.29	.52	
	64		.15	.32	.66
Intel XP/S MP Paragon	4	32	.64		
	16		.37	.66	
	64		.16	.42	.75
Berkeley NOW	4	32	.76		
	32		.38	.62	.71
	64		.28	.54	.69

Time(PDGESSV) / Time(PDGEMM)

Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	.50	.40	
	16		.75	.51	.40
	64		1.86	.72	.45
IBM SP2	4	50	.60		
	16		1.16	.64	
	64		2.24	1.03	.51
Intel XP/S GP Paragon	4	32	.52		
	16		.89	.50	
	64		2.08	.79	.44
Berkeley NOW	4	32	.44		
	32		.48	.54	.47
	64		1.18	.62	.49

3/9/2004

PDGESSV = ScaLAPACK parallel LU routine

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

53

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

3/9/2004

CS267 Lecture 13

54

Performance of ScalAPACK QR (Least squares)

Scales well, nearly full machine speed

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

Machine	Procs	Block Size	Time(PDGELS)/Time(PDGEMM)		
			2000	4000	10000
Cray T3E	4	32	1.2	1.1	1.1
	16	64	1.5	1.2	1.1
	64	64	2.6	1.4	1.2
IBM SP2	4	50	1.3	1.3	1.3
	16	64	2.3	1.8	1.9
	64	64	3.6	1.8	1.9
Intel XP/S GP Paragon	4	32	1.1	1.1	1.1
	16	64	1.6	1.1	1.1
	64	64	3.0	1.4	1.1
Berkeley NOW	4	32	1.3	1.9	1.9
	16	64	1.4	1.0	1.0
	64	64	1.8	1.1	1.1

3/9/2004 CS2

Performance of Symmetric Eigensolvers

Old version, pre 1998 Gordon Bell Prize

Still have ideas to accelerate Project Available!

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

Machine	Procs	Block Size	N	
			2000	4000
Cray T3E	4	32	35	35
	16	64	37	35
	64	64	57	41
IBM SP2	16	50	38	47
	64	64	58	47
	64	64	58	47
Intel XP/S GP Paragon	16	32	99	193
	64	64	193	193
	64	64	31	35
Berkeley NOW	16	32	31	35
	32	32	35	55
	32	32	35	55

3/9/2004 CS267 Lecture 13 56

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	64	66	64
	64	64	93	70
IBM SP2	4	50	97	60
	16	64	60	60
	64	64	81	60
Berkeley NOW	4	32	72	16
	16	64	38	16
	32	64	59	26

Performance of Nonsymmetric Eigensolver (QR iteration)

Hardest of all to parallelize Have alternative, and would like to compare Project available!

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

3/9/2004 CS267 Lecture 13 57

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 SciLA-PACK 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

Moral: use QR to solve Ax=b

Projects available (perhaps very hard...)

3/9/2004

A small software project ...

Participants

Krzysztof Aravind (UC Berkeley)	Zhaohui Bai (U Kentucky)
Richard Barrett (U. Tenn)	Michael Barry (U Tenn)
Jeff Bilmes (UC Berkeley)	Chris Bischof (ANL)
Sumit Chakrabarti (ORNL)	Sourav Chakrabarti (UC Berkeley)
Tony Chan (UCCLA)	Chao-Wei Chin (UC Berkeley)
Jaeyoung Choi (LBNL)	Andy Cleary (LBNL)
Ed D'Azevedo (ORNL)	Jim Demmel (UC Berkeley)
Isacett Dierker (UC Berkeley)	Jose Domingo (ORNL)
Jack Dongarra (U. Tenn, ORNL)	Zhalgo Dymal (U Hagen)
Jieqiang Du (NAG)	Victor Eijkhout (UCCLA)
Stan Eisenstat (Yale)	Vince Ferraro (NAG)
John Gilbert (Xerox PARC)	Ming Gu (UC Berkeley, IBM)
Dave Hammarling (NAG)	Mike Heath (U Illinois)
Greg Henry (Intel)	Dominic Lam (UC Berkeley)
Steve Humlics (ORNL)	Bo Kågström (U Umeå)
W. Kahan (UC Berkeley)	Yongqian Kim (U Tenn)
Rancong Li (UC Berkeley)	Xiaoye Li (UC Berkeley)
Joseph Liu (Yale)	Benjamin Pader (UC Berkeley)
Anilize Peltier (U Tenn)	Peter Paresani (U Umeå)
Rudolf Poo (U Tenn)	Paloma Raghavan (U Illinois)
Huan Ren (UC Berkeley)	Howard Robinson (UC Berkeley)
Charles Rovine (ORNL)	Jeff Rutter (UC Berkeley)
Dan Sargison (U Spitz)	Dan Sussner (Rice U)
Ken Stanley (UC Berkeley)	Xiaohai Sun (ANL)
Bernard Tourassis (U Tenn)	Anna Tuma (SRG)
Robert van de Geijl (U Twente)	Henk van der Vorst (Utrecht U)
Paul Van Dooren (U Illinois)	Kresimir Veselic (U Hagen)
David Walker (ORNL)	Chit Whaley (U Tenn)
Kathy Yalick (UC Berkeley)	

With the cooperation of Cray, IBM, Convex, DEC, Fujitsu, NERC, NAG, IMSL

3/9/2004 Supported by ARPA, NSF, DOE 59

Work-Depth Model of Parallelism

- The work depth model:
 - The simplest model is used
 - For algorithm design, independent of a machine
- The work, W , is the total number of operations
- The depth, D , is the longest chain of dependencies
- The parallelism, P , is defined as W/D
- Specific examples include:
 - circuit model, each input defines a graph with ops at nodes
 - vector model, each step is an operation on a vector of elements
 - language model, where set of operations defined by language

3/9/2004 CS267 Lecture 13 60