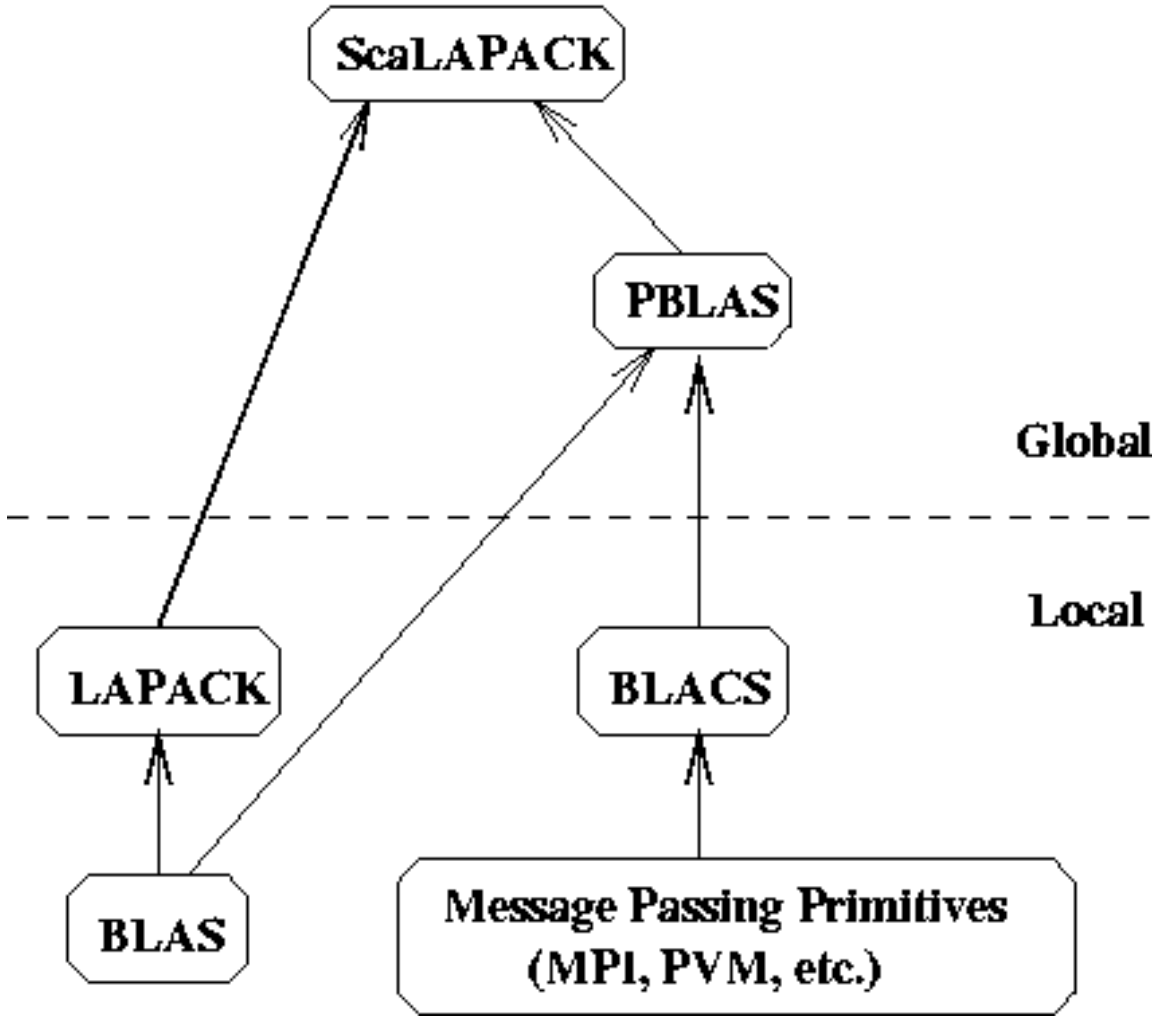
ScaLAPACK SOFTWARE HIERARCHY



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

Performance of ScaLAPACK QR (Least squares)

Scales well,
nearly full machine speed

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

Time(PDGELS)/Time(PDGEMM)					
Machine	Procs	Block Size	N		
			2000	4000	10000
Cray T3E	4	32	1.2	1.1	
	16		1.5	1.2	1.1
	64		2.6	1.4	1.2
IBM SP2	4	50	1.3		
	16		2.3	1.3	
	64		3.6	1.8	1.2
Intel XP/S GP Paragon	4	32	1.1		
	16		1.6	1.1	
	64		3.0	1.4	1.1
Berkeley NOW	4	32	1.3	.9	
	32		1.4	1.0	.9
	64		1.8	1.1	.9

Performance of Symmetric Eigensolvers

Old version,
pre 1998 Gordon Bell Prize

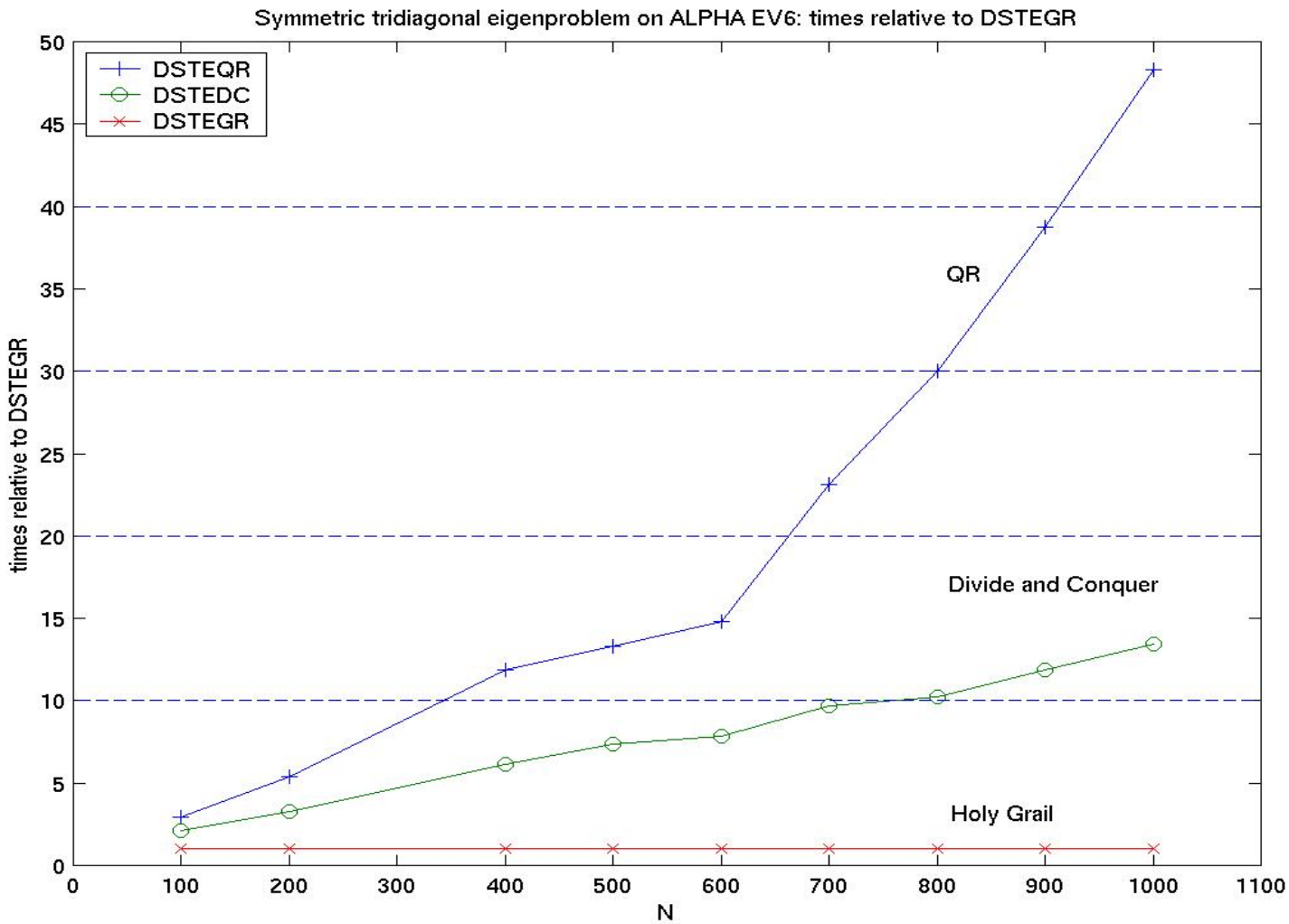
Still have ideas to accelerate
Project Available!

Time(PDSYEVX)/Time(PDGEMM) (bisection + inverse iteration)				
Machine	Procs	Block Size	N 2000	N 4000
Cray T3E	4	32	10	
	16		13	10
	64		29	14
IBB SP2	16	50	24	
	64		40	29
Intel XP/S GP Paragon	16	32	22	
	64		34	20
Berkeley NOW	16	32	20	
	32		24	52

Old Algorithm,
plan to abandon

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

The “Holy Grail” of Eigensolvers for Symmetric matrices



To be propagated throughout LAPACK and ScaLAPACK

Performance of SVD (Singular Value Decomposition)

Have good ideas to speedup
Project available!

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

Performance of Nonsymmetric Eigensolver (QR iteration)

Hardest of all to parallelize

Time(PDLAHQR)/Time(PDGEMM)				
Machine	Procs	Block Size	N	
			1000	1500
Intel XP/S MP Paragon	16	50	123	97

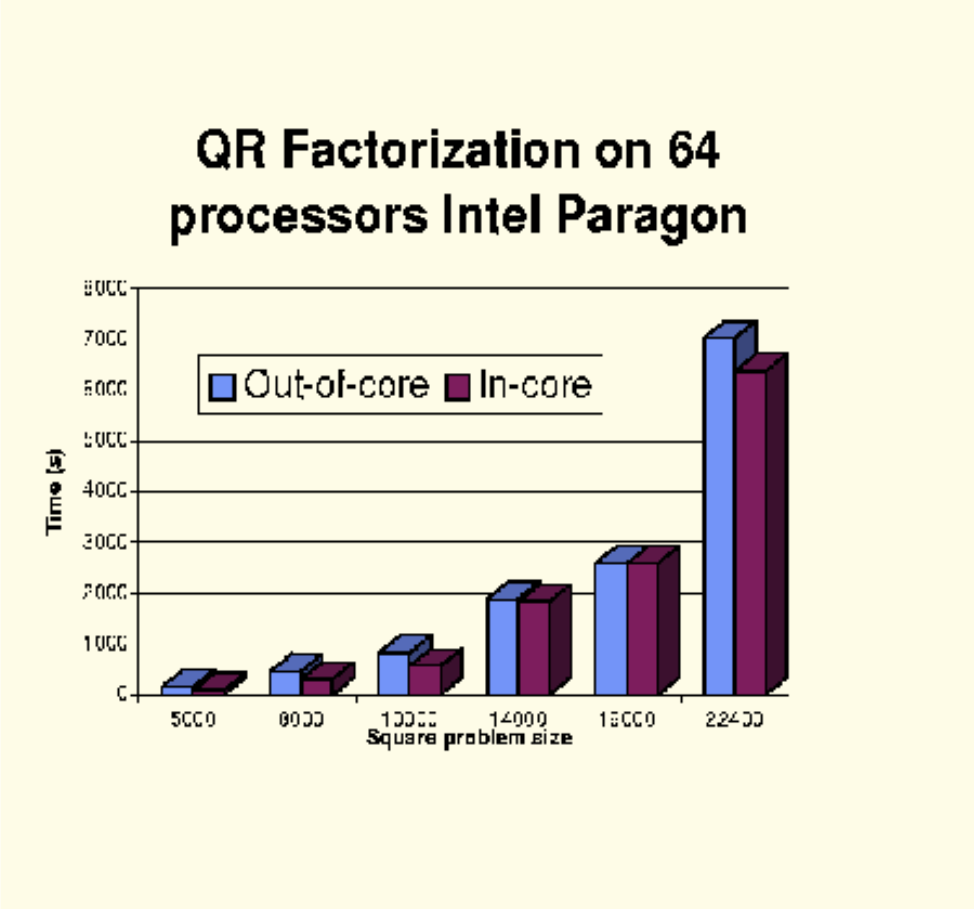
Out-of-Core Performance Results for Least Squares

- 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



A small software project ...

Participants

Krste Asanovic (UC Berkeley)
Richard Barrett (U. Tenn)
Jeff Bilmes (UC Berkeley)
Susan Blackford (ORNL)
Tony Chan (UCLA)
Jaeyoung Choi (LBNL)
Ed D'Azevedo (ORNL)
Inderjit Dhillon (UC Berkeley)
Jack Dongarra (U Tenn, ORNL)
Jeremy Du Croz (NAG)
Stan Eisenstat (Yale)
John Gilbert (Xerox PARC)
Sven Hammarling (NAG)
Greg Henry (Intel)
Steve Huss-Lederman (SRC)
W. Kahan (UC Berkeley)
Rencang Li (UC Berkeley)
Joseph Liu (York)
Antoine Petitet (U Tenn)
Roldan Pozo (U Tenn)
Huan Ren (UC Berkeley)
Charles Romine (ORNL)
Ivan Slapničar (U Split)
Ken Stanley (UC Berkeley)
Bernard Tourancheau (U Tenn)
Robert van de Geijn (U Texas)
Paul Van Dooren (U Illinois)
David Walker (ORNL)
Kathy Yelick (UC Berkeley)
Zhaojun Bai (U Kentucky)
Michael Berry (U Tenn)
Chris Bischof (ANL)
Soumen Chakrabarti (UC Berkeley)
Chee-Whye Chin (UC Berkeley)
Andy Cleary (LLNL)
Jim Demmel (UC Berkeley)
June Donato (ORNL)
Zlatko Drmač (U Hagen)
Victor Eijkhout (UCLA)
Vince Fernando (NAG)
Ming Gu (UC Berkeley, LBL)
Mike Heath (U Illinois)
Dominic Lam (UC Berkeley)
Bo Kågström (U Umeå)
Youngbae Kim (U Tenn)
Xiaoye Li (UC Berkeley)
Beresford Parlett (UC Berkeley)
Peter Poromaa (U Umeå)
Padma Raghavan (U Illinois)
Howard Robinson (UC Berkeley)
Jeff Rutter (UC Berkeley)
Dan Sorensen (Rice U)
Xiaobai Sun (ANL)
Anna Tsao (SRC)
Henk van der Vorst (Utrecht U)
Krešimir Veselić (U Hagen)
Clint Whaley (U Tenn)

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

Supported by ARPA, NSF, DOE

Extra Slides

Parallelizing Gaussian Elimination

- **Recall parallelization steps from Lecture 3**
  - **Decomposition:** identify enough parallel work, but not too much
  - **Assignment:** load balance work among threads
  - **Orchestrate:** communication and synchronization
  - **Mapping:** which processors execute which threads
- **Decomposition**
  - In BLAS 2 algorithm nearly each flop in inner loop can be done in parallel, so with n^2 processors, need 3n parallel steps

```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)          … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )  … BLAS 2 (rank-1 update)
          - A(i+1:n , i) * A(i , i+1:n)
```

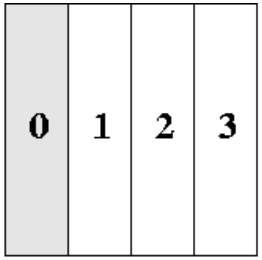
  - This is too fine-grained, prefer calls to local matmuls instead

Assignment of parallel work in GE

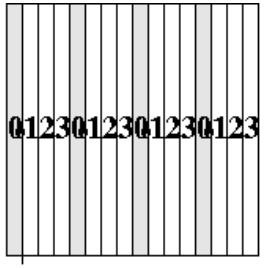
- Think of assigning submatrices to threads, where each thread responsible for updating submatrix it owns
  - "owner computes" rule natural because of locality
- What should submatrices look like to achieve load balance?

Different Data Layouts for Parallel GE (on 4 procs)

Bad load ba
P0 idle afte
n/4 steps



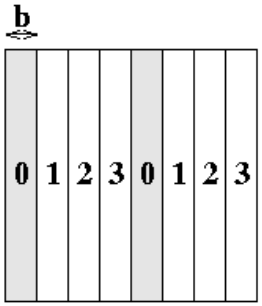
1) Column Blocked Layout



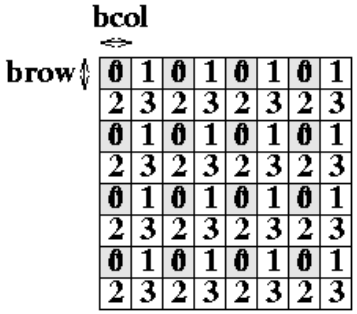
2) Column Cyclic Layout

t can't easily
S3

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

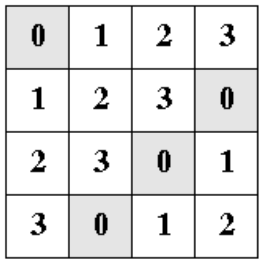


3) Column Block Cyclic Layout



4) Row and Column Block Cyclic Layout

nner!



5) Block Skewed Layout

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 objec

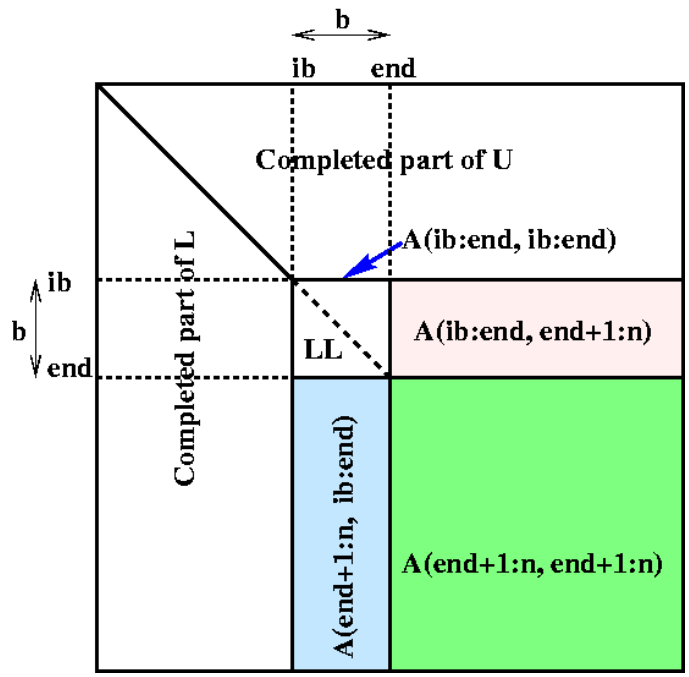
Analysis of MOM for Parallel Implementation

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

BLAS 3 (Blocked) GEPP, using Delayed Updates

```
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
BLAS 3 A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
            - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                                  … apply delayed updates with single matrix-multiply
                                  … with inner dimension b
```

Gaussian Elimination using BLAS 3



BLAS2 version of Gaussian Elimination with Partial Pivoting (GEPP)

```
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]
                 … 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
```

How to proceed:

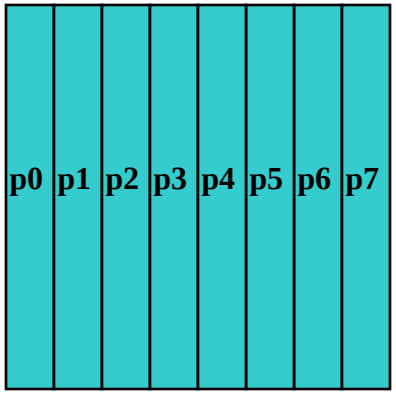
- Consider basic parallel matrix multiplication algorithms on simple layouts
  - Performance modeling to choose best one
    - Time (message) = latency + #words * time-per-word
    - $= \alpha + n*\beta$
- Briefly discuss block-cyclic layout
- PBLAS = Parallel BLAS

Parallel Matrix Multiply

- Computing C=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

1D Layout

- Assume matrices are n x n and n is divisible by p



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

$$C(i) = C(i) + A*B(i) = C(i) + \sum_{j} A(j)*B(j,i)$$

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

Naïve MatMul for 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: $\alpha + \beta*n^2/p$

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,

an of those, only i is doing computation

=> the algorithm is almost entirely serial

Running time**: (p*(p-1) + 1)*computation + p*(p-1)*communication**

$\sim= 2*n^3 + p^2*\alpha + p*n^2*\beta$

this is worse than the serial time and grows with p

Better Matmul for 1D layout on a Processor Ring

- Proc i can communicate with Proc(i-1) and Proc(i+1) simultaneously for all i

```
Copy A(myproc) into Tmp
C(myproc) = C(myproc) + T*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
- Time of inner loop = $2*(\alpha + \beta*n^2/p) + 2*n*(n/p)^2$
- Total Time = $2*n*(n/p)^2 + (p-1) *$ Time of inner loop

  $\sim 2*n^3/p + 2*p*\alpha + 2*\beta*n^2$

- 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)*cost of sending n*(n/p) words
- Parallel Efficiency = $2*n^3 / (p * \text{Total Time}) = 1/(1 + \alpha * p^2/(2*n^3) + \beta * p/(2*n))$

  $= 1/(1 + O(p/n))$

  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

p(0,0)	p(0,1)	p(0,2)
p(1,0)	p(1,1)	p(1,2)
p(2,0)	p(2,1)	p(2,2)

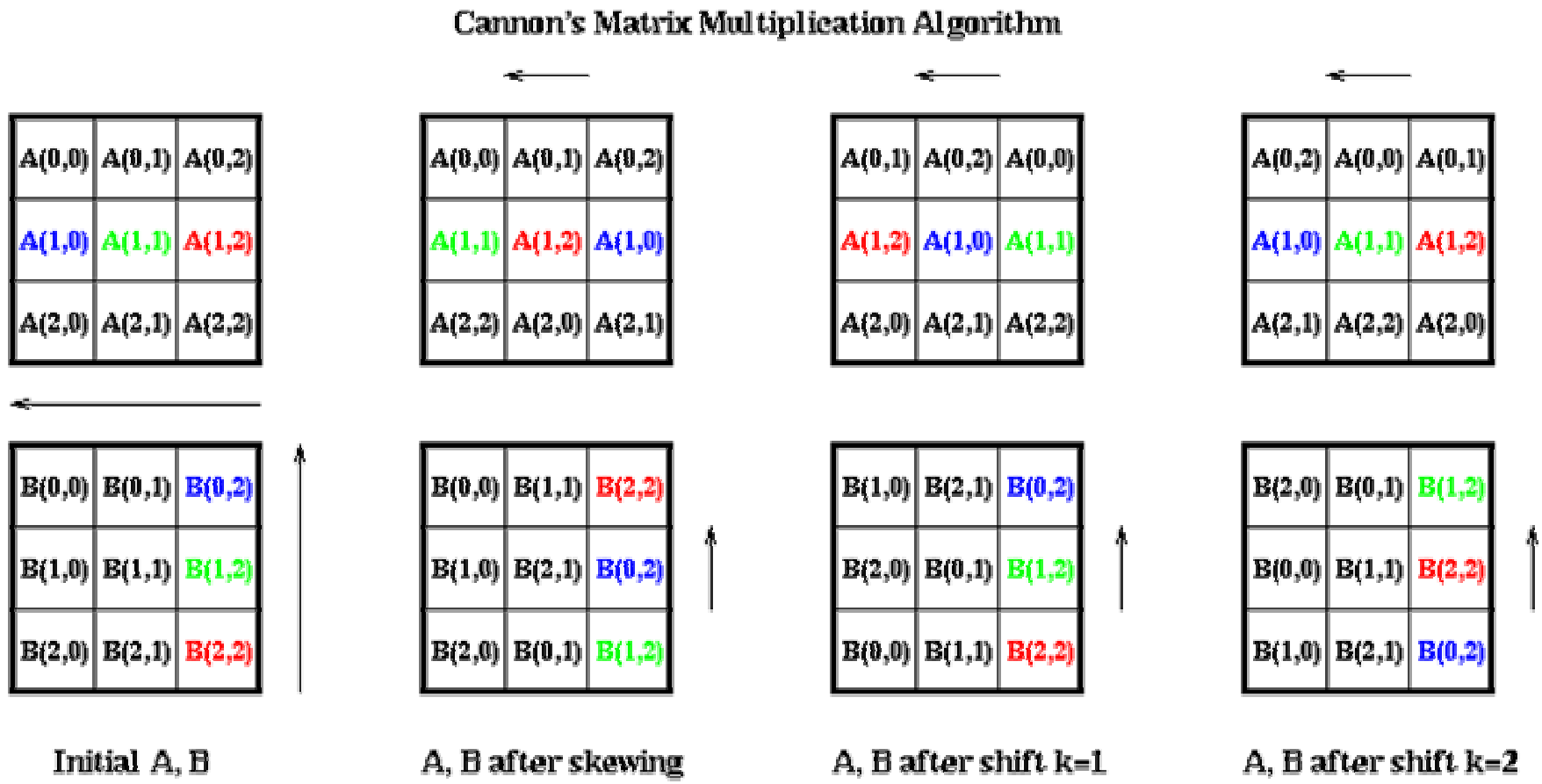
Cannon's Algorithm

… $C(i,j) = C(i,j) + \sum_k A(i,k)*B(k,j)$

… assume s = sqrt(p) is an integer

```
forall  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)
forall  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
    forall 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
```

Communication in Cannon



C(1,2) = A(1,0) * B(0,2) + A(1,1) * B(1,2) + A(1,2) * B(2,2)

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*(α + β*n^2/p)
forall  i=0 to s-1
    up-circular-shift B column i of B by i ... cost = s*(α + β*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,j)*B(i,j)   ... cost = 2*(n/s)^3 = 2*n^3/p^3/2
        left-circular-shift each row of A by 1   ... cost = α + β*n^2/p
        up-circular-shift each row of B by 1     ... cost = α + β*n^2/p
```

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

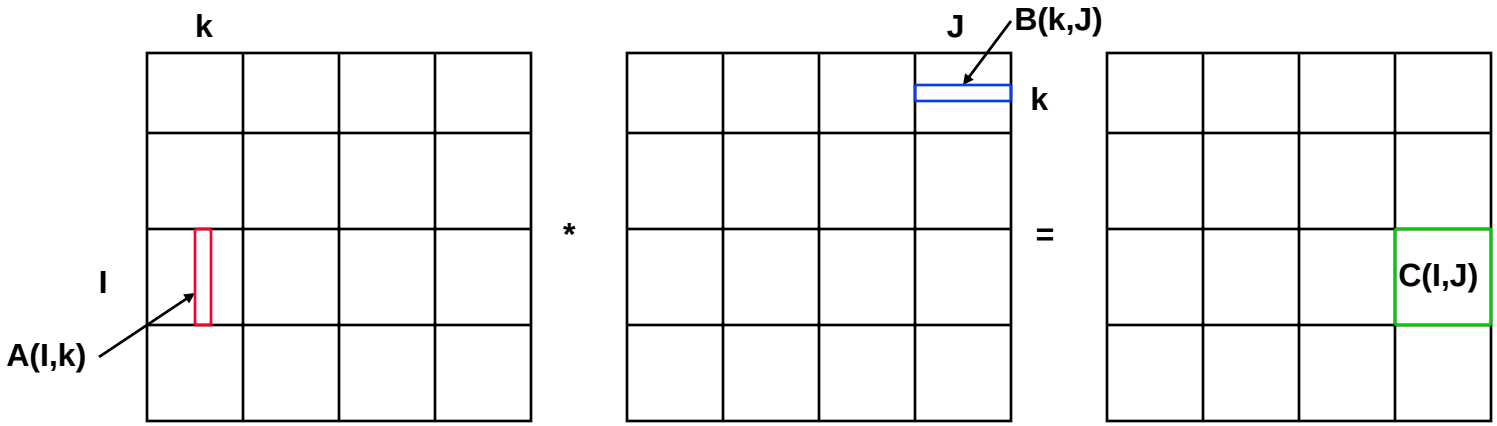
Drawbacks to Cannon

- 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)

SUMMA = Scalable Universal Matrix Multiply Algorithm

- 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

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) + Σ_k A(I,k)*B(k,J)
- Assume a p_r by p_c processor grid ($p_r = p_c = 4$ above)

```
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
```

SUMMA performance

```
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 *( α + β * 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 *( α + β * 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 = $2*n^3/p + \alpha * \log p * n/b + \beta * \log p * n^2/s$
- Parallel Efficiency = $1/(1 + \alpha * \log p * p / (2*b*n^2) + \beta * \log p * s/(2*n))$
- ~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

Summary of Parallel Matrix Multiply Algorithms

- **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 = $1/(1+O(p^{1/2}/n))$**
    - **Hard to generalize for general p, n, block cyclic, alignment**
  - **SUMMA**
    - **Efficiency = $1/(1 + O(\log p * p / (b*n^2) + \log p * p^{1/2}/n))$**
    - **Very General**
    - **b small => less memory, lower efficiency**
    - **b large => more memory, high efficiency**
  - **Gustavson et al**
    - **Efficiency = $1/(1 + O(p^{1/3}/n))$??**