CS 267
Dense Linear Algebra: Parallel Gaussian Elimination

James Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr14

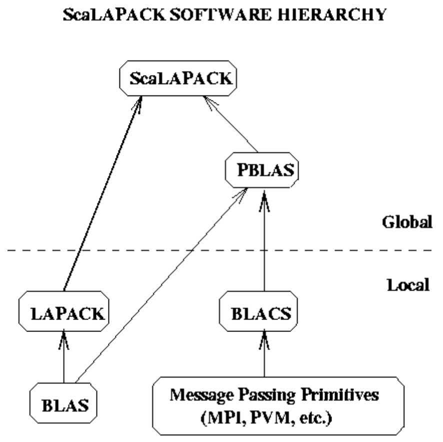
Outline

- Review Gaussian Elimination (GE) for solving Ax=b
- Optimizing GE for caches on sequential machines
  - using matrix-matrix multiplication (BLAS and LAPACK)
- Minimizing communication for sequential GE
  - Not LAPACK, but Recursive LU minimizes bandwidth (latency possible)
- Data layouts on parallel machines
- Parallel Gaussian Elimination (ScaLAPACK)
- Minimizing communication for parallel GE
  - Not ScaLAPACK (yet), but "Comm-Avoiding LU" (CALU)
  - Same idea for minimizing bandwidth and latency in sequential case
- Summarize rest of dense linear algebra
- Dynamically scheduled LU for Multicore
- LU for Heterogeneous computers (CPU + GPU)

Summary of Matrix Multiplication

- Goal: Multiply n x n matrices C = A·B using $O(n^3)$ arithmetic operations, minimizing data movement
- Sequential
  - Assume fast memory of size $M < 3n^2$, count slow mem. refs.
  - Thm: need $\Omega(n^3/M^{1/2})$ slow mem. refs. and $\Omega(n^3/M^{3/2})$ messages
  - Attainable using "blocked" or "recursive" matrix multiply
- Parallel
  - Assume P processors, $O(n^2/P)$ data per processor
  - Thm: need $\Omega(n^2/P^{1/2})$ words sent and $\Omega(P^{1/2})$ messages
  - Attainable by Cannon, nearly by SUMMA
    - SUMMA used in practice (PBLAS)
  - c copies of data $\Rightarrow c^{1/2}$ times fewer words, $c^{3/2}$ fewer messages
- Which other linear algebra problems can we do with as little data movement?
  - Today: Solve Ax=b in detail, summarize what's known, open

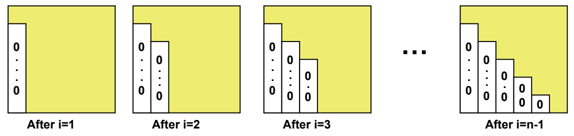
Sca/LAPACK Overview


ScaLAPACK SOFTWARE HIERARCHY
ScaLAPACK
PBLAS
Global
Local
LAPACK
BLACS
BLAS
Message Passing Primitives (MPI, PVM, etc.)

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)
```



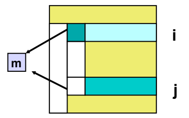
Refine GE Algorithm (1)

- 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)
```



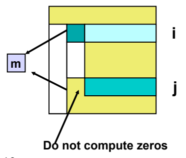
Refine GE Algorithm (2)

- Last version

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

- Don't compute what we already know: zeros below diagonal in column i

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



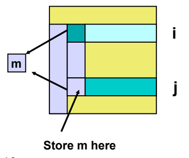
Refine GE Algorithm (3)

- Last version

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

- Store multipliers m below diagonal in zeroed entries for later use

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



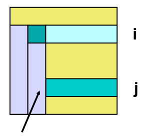
Refine GE Algorithm (4)

- **Last version**

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

- **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)
```



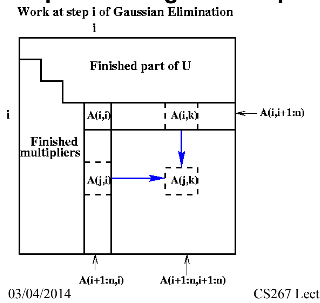
Store all m's here before updating rest of matrix

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)**

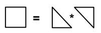


```
for i = 1 to n-1
    A(i+1:n,i) = A(i+1:n,i) * ( 1 / A(i,i) )
        … BLAS 1 (scale a vector)
    A(i+1:n,i+1:n) = A(i+1:n , i+1:n )
        - A(i+1:n , i) * A(i , i+1:n)
        … BLAS 2 (rank-1 update)
```

What GE really computes

```
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)
```

- Call the strictly lower triangular matrix of multipliers M, and let L = I+M
- Call the upper triangle of the final matrix U
- *Lemma (LU Factorization):* If the above algorithm terminates (does not divide by zero) then A = L*U

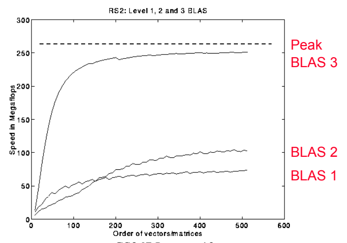


- Solving A*x=b using GE
  - Factorize A = L*U using GE (cost = 2/3 n^3 flops)
  - Solve L*y = b for y, using substitution (cost = n^2 flops)
  - Solve U*x = y for x, using substitution (cost = n^2 flops)
- Thus A*x = (L*U)*x = L*(U*x) = L*y = b as desired

Problems with basic GE algorithm

```
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)
```

- 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…)



Pivoting in Gaussian Elimination

- A = [0 1] fails completely because can't divide by A(1,1)=0
  [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

Gaussian Elimination with Partial Pivoting (GEPP)

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

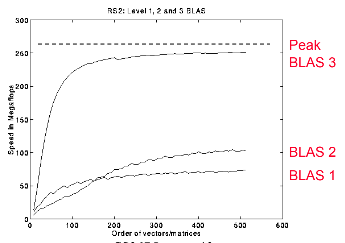
```
for i = 1 to n-1
    find and record k where |A(k,i)| = max{i ≤ j ≤ n} |A(j,i)|
        … i.e. largest entry in rest of column i
    if |A(k,i)| = 0
        exit with a warning that A is singular, or nearly so
    elseif k ≠ i
        swap rows i and k of A
    end if
    A(i+1:n,i) = A(i+1:n,i) / A(i,i)        … each |quotient| ≤ 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/sgetf2.f
- Standard approach – but communication costs?

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)
```

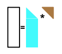


Converting BLAS2 to BLAS3 in GEPP

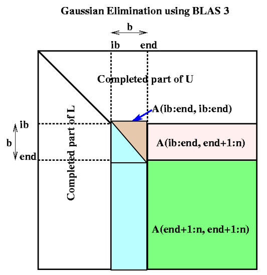
- Blocking
  - Used to optimize matrix-multiplication
  - Harder here because of data dependencies in GEPP
- BIG IDEA: Delayed Updates
  - Save updates to "trailing matrix" from several consecutive BLAS2 (rank-1) updates
  - Apply many updates simultaneously in one BLAS3 (matmul) operation
- Same idea works for much of dense linear algebra
  - Not eigenvalue problems or SVD – need more ideas
- First Approach: Need to choose a block size b
  - Algorithm will save and apply b updates
  - b should be small enough so that active submatrix consisting of b columns of A fits in cache
  - b should be large enough to make BLAS3 (matmul) fast

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

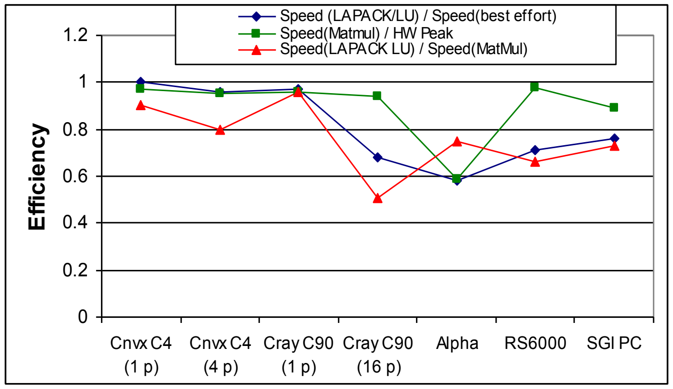
```
for  ib = 1 to n-1 step b     … Process matrix b columns at a time
    end = ib + b-1             … Point to end of block of b columns
    apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P’ * L’ * U’
    … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
    A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)        … update next b rows of U
    A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
            - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                             … apply delayed updates with single matrix-multiply
                             … with inner dimension b
```



(For a correctness proof, see on-line notes from CS267 / 1996.)



Efficiency of Blocked GEPP
(all parallelism “hidden” inside the BLAS)



Communication Lower Bound for GE

- Matrix Multiplication can be “reduced to” GE
- Not a good way to do matmul but it shows that GE needs at least as much communication as matmul
- Does blocked GEPP minimize communication?

$$\begin{bmatrix} I & 0 & -B \\ A & I & 0 \\ 0 & 0 & I \end{bmatrix} = \begin{bmatrix} I & & \\ A & I & \\ 0 & 0 & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 & -B \\ & I & A\cdot B \\ & & I \end{bmatrix}$$

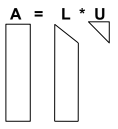
Does LAPACK’s GEPP Minimize Communication?

```
for  ib = 1 to n-1 step b     … Process matrix b columns at a time
    end = ib + b-1             … Point to end of block of b columns
    apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P’ * L’ * U’
    … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
    A(ib:end , end+1:n) = LL^-1 * A(ib:end , end+1:n)        … update next b rows of U
    A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
            - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                             … apply delayed updates with single matrix-multiply
                             … with inner dimension b
```

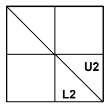
- Case 1: $n \geq M$ - huge matrix – attains lower bound
  - $b = M^{1/2}$ optimal, dominated by matmul
- Case 2: $n \leq M^{1/2}$ - small matrix – attains lower bound
  - Whole matrix fits in fast memory, any algorithm attains lower bound
- Case 3: $M^{1/2} < n < M$ - medium size matrix – not optimal
  - Can’t choose b to simultaneously optimize matmul and BLAS2 GEPP of n x b submatrix
  - Worst case: Exceed lower bound by factor $M^{1/6}$ when $n = M^{2/3}$
- Detailed counting on backup slides

Alternative cache-oblivious GE formulation (1/2)

- Toledo (1997)
  - Describe without pivoting for simplicity
  - "Do left half of matrix, then right half"



```
function [L,U] = RLU (A)   … assume A is m by n
    if (n=1)   L = A/A(1,1),  U = A(1,1)
    else
        [L1,U1] = RLU( A(1:m , 1:n/2)) … do left half of A
           … let L11 denote top n/2 rows of L1
        A( 1:n/2 , n/2+1 : n ) = $L11^{-1}$ * A( 1:n/2 , n/2+1 : n )
           … update top n/2 rows of right half of A
        A( n/2+1: m, n/2+1:n ) = A( n/2+1: m, n/2+1:n )
           - A( n/2+1: m, 1:n/2 ) * A( 1:n/2 , n/2+1 : n )
           … update rest of right half of A
        [L2,U2] = RLU( A(n/2+1:m , n/2+1:n) ) … do right half of A
        return [ L1,[0;L2] ] and [U1, [ A(.,.) ; U2 ] ]
```



Alternative cache-oblivious GE formulation (2/2)

```
function [L,U] = RLU (A)   … assume A is m by n
    if (n=1)   L = A/A(1,1),  U = A(1,1)
    else
        [L1,U1] = RLU( A(1:m , 1:n/2)) … do left half of A
           … let L11 denote top n/2 rows of L1
        A( 1:n/2 , n/2+1 : n ) = $L11^{-1}$ * A( 1:n/2 , n/2+1 : n )
           … update top n/2 rows of right half of A
        A( n/2+1: m, n/2+1:n ) = A( n/2+1: m, n/2+1:n )
           - A( n/2+1: m, 1:n/2 ) * A( 1:n/2 , n/2+1 : n )
           … update rest of right half of A
        [L2,U2] = RLU( A(n/2+1:m , n/2+1:n) ) … do right half of A
        return [ L1,[0;L2] ] and [U1, [ A(.,.) ; U2 ] ]
```

- $W(m,n) = W(m,n/2) + O(\max(m\cdot n, m\cdot n^2/M^{1/2})) + W(m-n/2,n/2)$

$\leq 2 \cdot W(m,n/2) + O(\max(m\cdot n, m\cdot n^2/M^{1/2}))$

$= O(m\cdot n^2/M^{1/2} + m\cdot n\cdot \log M)$

$= O(m\cdot n^2/M^{1/2})$ if $M^{1/2}\cdot \log M = O(n)$

Still doesn't minimize latency, but fixable CLASS PROJECT

Explicitly Parallelizing Gaussian Elimination

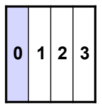
- Parallelization steps
  - Decomposition: identify enough parallel work, but not too much
  - Assignment: load balance work among threads
  - Orchestrate: communication and synchronization
  - Mapping: which processors execute which threads (locality)
- Decomposition
  - In BLAS 2 algorithm nearly each flop in inner loop can be done in parallel, so with n^2 processors, need 3n parallel steps, O(n log n) with pivoting

```
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)
```

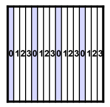
  - This is too fine-grained, prefer calls to local matmuls instead
  - Need to use parallel matrix multiplication
- Assignment and Mapping
  - Which processors are responsible for which submatrices?

Different Data Layouts for Parallel GE

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



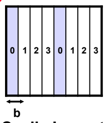
1) 1D Column Blocked Layout



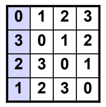
Load balanced, but can't easily use BLAS3

2) 1D Column Cyclic Layout

Can trade load balance and BLAS3 performance by choosing b, but factorization of block column is a bottleneck



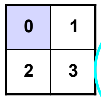
3) 1D Column Block Cyclic Layout



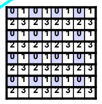
Complicated addressing, May not want full parallelism In each column, row

4) Block Skewed Layout

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



5) 2D Row and Column Blocked Layout

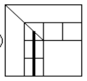


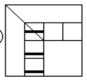
The winner!

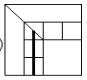
6) 2D Row and Column Block Cyclic Layout

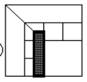
Distributed Gaussian Elimination with a 2D Block Cyclic Layout

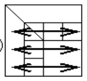
```
for ib = 1 to n-1 step b
    end = min( ib+b-1, n )
    for i = ib to end
        (1)  find pivot row k, column broadcast
        (2)  swap rows k and i in block column, broadcast row k
        (3)  A( i+1:n , i) = A( i+1:n , i) / A( i , i)
        (4)  A( i+1:n , i+1:end) -= A( i+1:n , i) * A( i , i+1:end)
    end for
(5)  broadcast all swap information right and left
(6)  apply all rows swaps to other columns
```

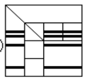
(1) 

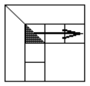
(2) 

(3) 

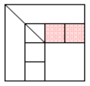
(4) 

(5) 

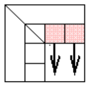
(6) 

(7) 

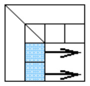
(7) Broadcast LL right

(8) 

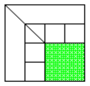
(8) A(ib:end , end+1:n) = LL \ A(ib:end , end+1:n)

(9) 

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

(10) 

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

(11) 

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

Matrix multiply of
green = green - blue * pink

Review of Parallel MatMul

- **Want Large Problem Size Per Processor**

PDGEMM = PBLAS matrix multiply

Observations:
- **For fixed N, as P increasesn 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**

Performance of PBLAS

Speed in Mflops of PDGEMM

Machine	Procs	Block	N		
		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 Paragon	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

Efficiency = MFlops(PDGEMM)/(Procs*MFlops(DGEMM))

Machine	Peak/	DGEMM	Procs	N		
	proc	Mflops		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 Paragon	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

PDGESV = ScaLAPACK Parallel 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 $(2*n^3)/(2/3*n^3) = 3$ times faster, but communication makes it a little slower.**

03/04/2014

Performance of ScaLAPACK LU

Efficiency = MFlops(PDGESV)/MFlops(PDGEMM)

Machine	Procs	Block	N		
		Size	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(PDGESV)/Time(PDGEMM)

Machine	Procs	Block	N		
		Size	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		.88	.54	.47
	64		1.18	.62	.49

Does ScaLAPACK Minimize Communication?

- Lower Bound: O(n^2 / $P^{1/2}$) words sent in O($P^{1/2}$) mess.
  - Attained by Cannon and SUMMA (nearly) for matmul
- ScaLAPACK:
  - O(n^2 log P / $P^{1/2}$) words sent – close enough
  - O(n log P) messages – too large
  - Why so many? One reduction costs O(log P) per column to find maximum pivot, times n = #columns
- Need to abandon partial pivoting to reduce #messages
  - Suppose we have n x n matrix on $P^{1/2}$ x $P^{1/2}$ processor grid
  - Goal: For each panel of b columns spread over $P^{1/2}$ procs, identify b "good" pivot rows in one reduction
    - Call this factorization TSLU = "Tall Skinny LU"
  - Several natural bad (numerically unstable) ways explored, but good way exists
    - SC08, "Communication Avoiding GE", D., Grigori, Xiang

Choosing Rows by "Tournament Pivoting"

$$W^{nxb} = \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{pmatrix} = \begin{pmatrix} P_1 \cdot L_1 \cdot U_1 \\ P_2 \cdot L_2 \cdot U_2 \\ P_3 \cdot L_3 \cdot U_3 \\ P_4 \cdot L_4 \cdot U_4 \end{pmatrix}$$

Choose b pivot rows of W_1, call them W_1'
Choose b pivot rows of W_2, call them W_2'
Choose b pivot rows of W_3, call them W_3'
Choose b pivot rows of W_4, call them W_4'

$$\begin{pmatrix} W_1' \\ W_2' \\ W_3' \\ W_4' \end{pmatrix} = \begin{pmatrix} P_{12} \cdot L_{12} \cdot U_{12} \\ P_{34} \cdot L_{34} \cdot U_{34} \end{pmatrix}$$

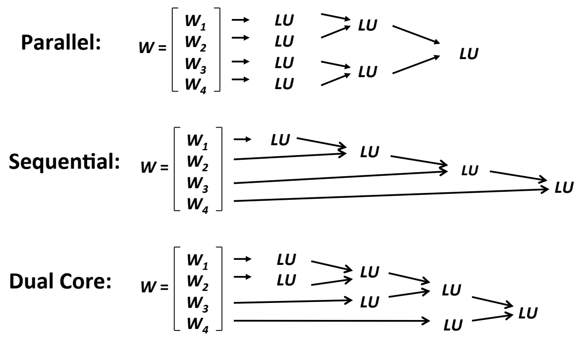
Choose b pivot rows, call them W_{12}'
Choose b pivot rows, call them W_{34}'

$$\begin{pmatrix} W_{12}' \\ W_{34}' \end{pmatrix} = P_{1234} \cdot L_{1234} \cdot U_{1234}$$

Choose b pivot rows

Go back to W and use these b pivot rows
(move them to top, do LU without pivoting)
Not the same pivots rows chosen as for GEPP
Need to show numerically stable (D., Grigori, Xiang, '11)

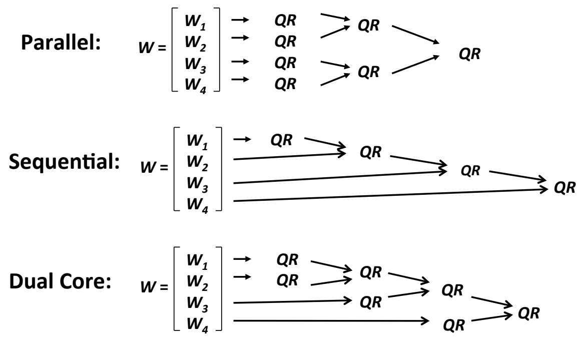
Minimizing Communication in TSLU



Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can Choose reduction tree dynamically

Same idea for QR of Tall-skinny matrix (TSQR)



First step of SVD of Tall-Skinny matrix

CALU – Communication-Avoiding LU

- Substitute TSLU for panel factorization in usual LU
- Thm: Tournament Pivoting (TP) as stable as Partial Pivoting (PP) in following sense: TP gets same results as PP applied to different input matrix whose entries are blocks taken from input A
  - So if you trusted PP before, you should trust TP now…
- Worst case pivot growth factor on n x n matrix
  - Proven: 2^{nH-1} where H = 1 + height of reduction tree
  - Attained: 2^{n-1}, same as PP
- There are examples where PP is exponentially less stable than TP, and vice-versa, so neither is always better than the other
- Extensive numerical testing confirms stability

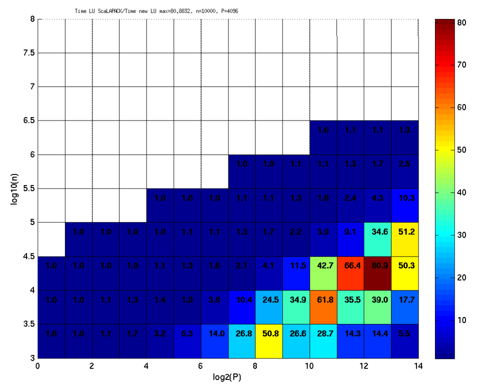
Performance vs ScaLAPACK LU

- TSLU
  - IBM Power 5
    - Up to 4.37x faster (16 procs, 1M x 150)
  - Cray XT4
    - Up to 5.52x faster (8 procs, 1M x 150)
- CALU
  - IBM Power 5
    - Up to 2.29x faster (64 procs, 1000 x 1000)
  - Cray XT4
    - Up to 1.81x faster (64 procs, 1000 x 1000)
- See INRIA Tech Report 6523 (2008), paper at SC08

TSQR Performance Results

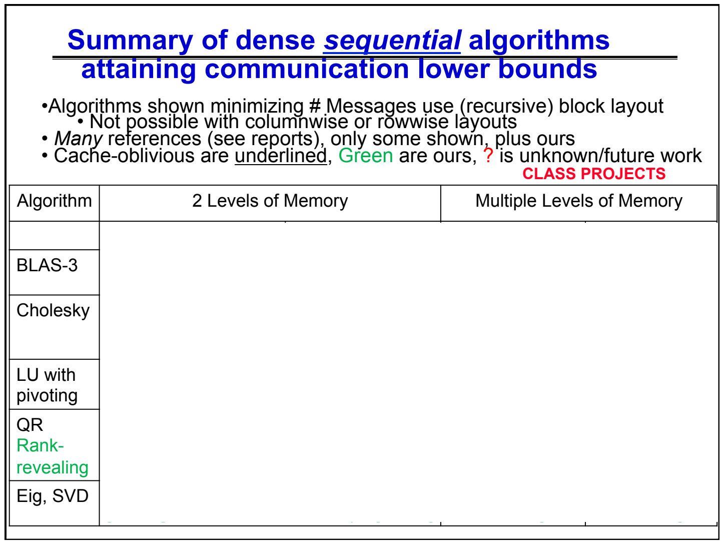
- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - Cloud – (Gleich and Benson) ~2 map-reduces
- Sequential
  - "Infinite speedup" for out-of-core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished
- SVD costs about the same
- Joint work with Grigori, Hoemmen, Langou, Anderson, Ballard, Keutzer, others

CALU speedup prediction for a Petascale machine - up to 81x faster



Petascale machine with 8192 procs, each at 500 GFlops/s, a bandwidth of 4 GB/s.

$$\gamma = 2\cdot 10^{-12} s, \alpha = 10^{-5} s, \beta = 2\cdot 10^{-9} s/word.$$

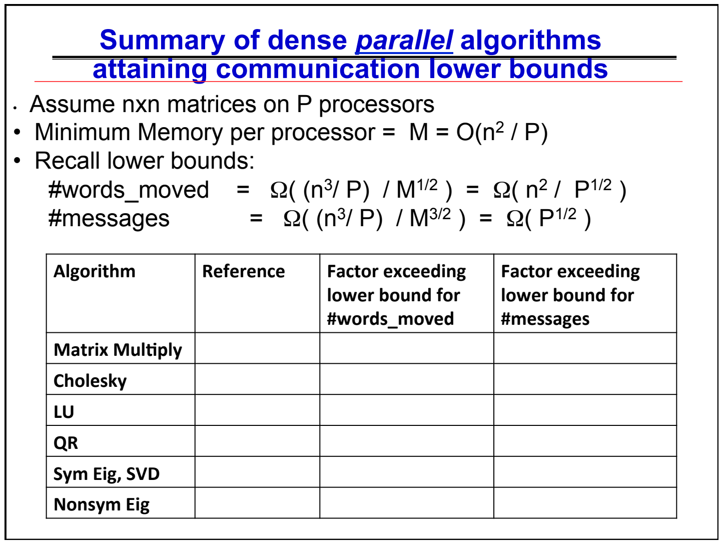


Summary of dense *sequential* algorithms attaining communication lower bounds

- Algorithms shown minimizing # Messages use (recursive) block layout
  - Not possible with columnwise or rowwise layouts
- *Many* references (see reports), only some shown, plus ours
- Cache-oblivious are underlined, Green are ours, ? is unknown/future work

CLASS PROJECTS

Algorithm	2 Levels of Memory	Multiple Levels of Memory
BLAS-3		
Cholesky		
LU with pivoting		
QR Rank-revealing		
Eig, SVD		



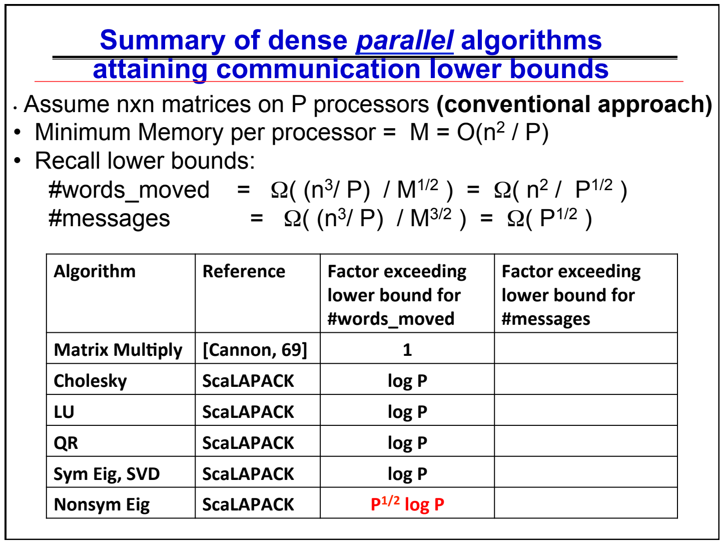
Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors
- Minimum Memory per processor = M = O(n^2 / P)
- Recall lower bounds:

  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix Multiply			
Cholesky			
LU			
QR			
Sym Eig, SVD			
Nonsym Eig			



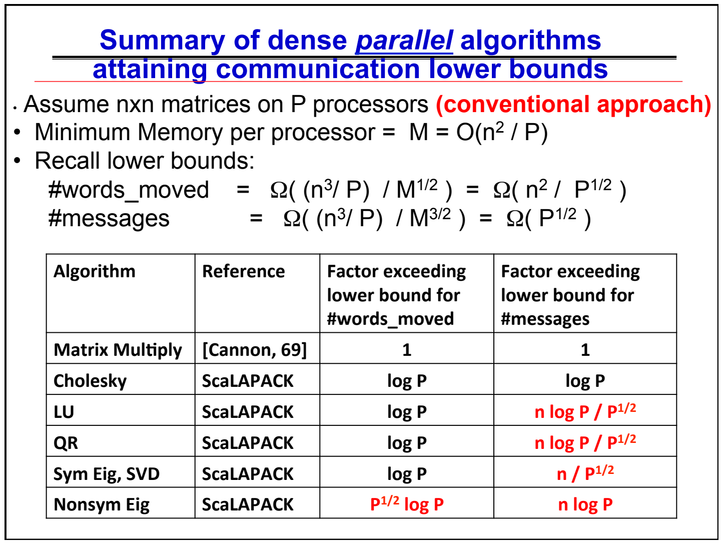
Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors **(conventional approach)**
- Minimum Memory per processor = M = O(n^2 / P)
- Recall lower bounds:

  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix Multiply	[Cannon, 69]	1	
Cholesky	ScaLAPACK	log P	
LU	ScaLAPACK	log P	
QR	ScaLAPACK	log P	
Sym Eig, SVD	ScaLAPACK	log P	
Nonsym Eig	ScaLAPACK	$P^{1/2}$ log P	



Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors **(conventional approach)**
- Minimum Memory per processor = M = O(n^2 / P)
- Recall lower bounds:

  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix Multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	ScaLAPACK	log P	n log P / $P^{1/2}$
QR	ScaLAPACK	log P	n log P / $P^{1/2}$
Sym Eig, SVD	ScaLAPACK	log P	n / $P^{1/2}$
Nonsym Eig	ScaLAPACK	$P^{1/2}$ log P	n log P

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors (**better**)
- Minimum Memory per processor = M = $O(n^2 / P)$
- Recall lower bounds:

  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$
  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$

CLASS PROJECTS

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix Multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	[GDX10]	log P	log P
QR	[DGHL08]	log P	$\log^3$ P
Sym Eig, SVD	[BDD11]	log P	$\log^3$ P
Nonsym Eig	[BDD11]	log P	$\log^3$ P

Can we do even better?

- Assume nxn matrices on P processors
- **Use c copies of data:** M = $O(cn^2 / P)$ per processor
- Increasing M reduces lower bounds:

  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / (c^{1/2} P^{1/2}))$
  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2} / c^{3/2})$

CLASS PROJECTS

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix Multiply	[DS11,SBD11]	polylog P	polylog P
Cholesky	[SD11, in prog.]	polylog P	c^2 polylog P
LU	[DS11,SBD11]	polylog P	c^2 polylog P
QR	Via Cholesky QR	polylog P	c^2 polylog P
Sym Eig, SVD	?		
Nonsym Eig	?		

Dense Linear Algebra on Recent Architectures

- **Multicore**
  - How do we schedule all parallel tasks to minimize idle time?
- **GPUs**
  - Heterogeneous computer: consists of functional units (CPU and GPU) that are good at different tasks
  - How do we divide the work between the GPU and CPU to take maximal advantage of both?
  - Challenging now, will get more so as platforms become more heterogeneous

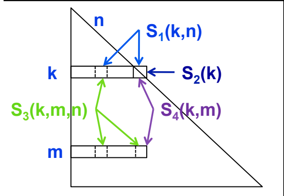
Multicore: Expressing Parallelism with a DAG

- **DAG = Directed Acyclic Graph**
  - S1 → S2 means statement S2 "depends on" statement S1
  - Can execute in parallel any Si without input dependencies
- **For simplicity, consider Cholesky A = LL^T, not LU**
  - N by N matrix, numbered from A(0,0) to A(N-1,N-1)
  - "Left looking" code: at step k, completely compute column k of L

```
for k = 0 to N-1
   for n = 0 to k-1
      A(k,k) = A(k,k) – A(k,n)*A(k,n)
   A(k,k) = sqrt(A(k,k))
   for m = k+1 to N-1
      for n = 0 to k-1
         A(m,k) = A(m,k) – A(m,n)*A(k,n)
      A(m,k) = A(m,k) / A(k,k)
```

Expressing Parallelism with a DAG - Cholesky

```
              for k = 0 to N-1
                 for n = 0 to k-1
$S_1(k,n)$             A(k,k) = A(k,k) – A(k,n)*A(k,n)
$S_2(k)$          A(k,k) = sqrt(A(k,k))
                 for m = k+1 to N-1
                    for n = 0 to k-1
$S_3(k,m,n)$              A(m,k) = A(m,k) – A(m,n)*A(k,n)
$S_4(k,m)$          $A(m,k) = A(m,k) \cdot A(k,k)^{-1}$
```



DAG has $\approx N^3/6$ vertices:

$S_1(k,n) \rightarrow S_2(k)$ for n=0:k-1

$S_3(k,m,n) \rightarrow S_4(k,m)$ for n=0:k-1

$S_2(k) \rightarrow S_4(k,m)$ for m=k+1:N

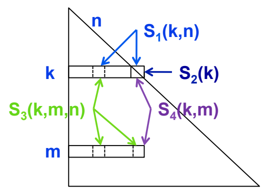
$S_4(k,m) \rightarrow S_3(k',m,k)$ for k'>k

$S_4(k,m) \rightarrow S_3(k,m',k)$ for m'>m

Expressing Parallelism with a DAG – Block Cholesky

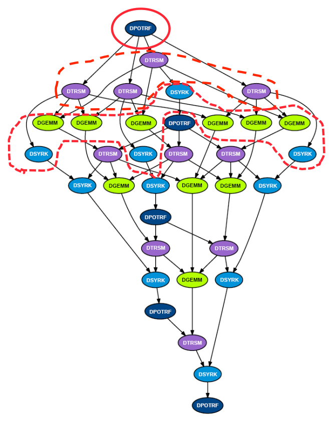
• Each A[i,j] is a b-by-b block

```
                          for k = 0 to N/b-1
                             for n = 0 to k-1
SYRK:     $S_1(k,n)$              $A[k,k] = A[k,k] – A[k,n]*A[k,n]^T$
POTRF:    $S_2(k)$           A[k,k] = unblocked_Cholesky(A[k,k])
                             for m = k+1 to N/b-1
                                for n = 0 to k-1
GEMM:     $S_3(k,m,n)$                $A[m,k] = A[m,k] – A[m,n]*A[k,n]^T$
TRSM:     $S_4(k,m)$              $A[m,k] = A[m,k] \cdot A[k,k]^{-1}$
```



Same DAG, but only $\approx(N/b)^3/6$ vertices

Sample Cholesky DAG with #blocks in any row or column = N/b = 5



• Note implied order of summation from left to right

• Not necessary for correctness, but it does reflect what the sequential code does

• Can process DAG in any order respecting dependences

Slide courtesy of Jakub Kurzak, UTK

Scheduling options

• *Static* (pre-assign tasks to processors) vs *Dynamic* (idle processors grab ready jobs from work-queue)
  - If dynamic, does scheduler take user hints/priorities?

• Respect locality (eg processor must have some task data in its cache) vs not

• Build and store entire DAG to schedule it (which may be very large, $(N/b)^3$), vs Build just the next few "levels" at a time (smaller, but less information for scheduler)

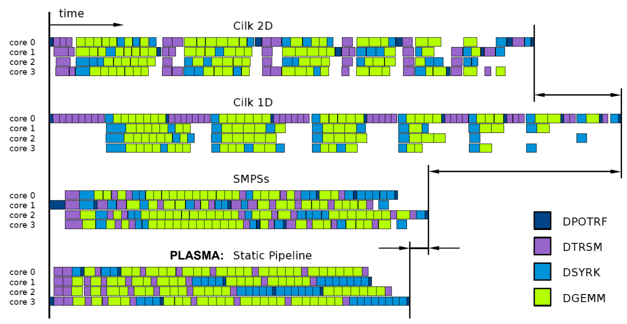
• Programmer builds DAG & schedule vs Depend on compiler or run-time system
  - Ease of programming, vs not exploiting user knowledge
  - If compiler, how conservative is detection of parallelism?
  - Generally useful, not just linear algebra

Schedulers tested

- Cilk
  - programmer-defined parallelism
  - spawn – creates independent tasks
  - sync – synchronizes a sub-branch of the tree
- SMPSs
  - dependency-defined parallelism
  - pragma-based annotation of tasks (directionality of the parameters)
- PLASMA (Static Pipeline)
  - programmer-defined (hard-coded)
  - apriori processing order
  - stalling on dependencies

Slide courtesy of Jakub Kurzak, UTK

Measured Results for Tiled Cholesky



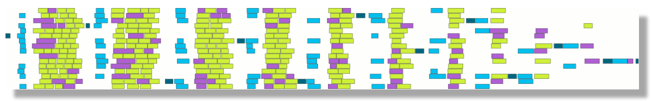
- Measured on Intel Tigerton 2.4 GHz
- Cilk 1D: one task is whole panel, but with “look ahead”
- Cilk 2D: tasks are blocks, scheduler steals work, little locality
- PLASMA works best

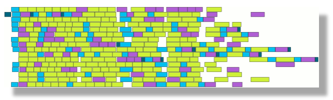
Slide courtesy of Jakub Kurzak, UTK

More Measured Results for Tiled Cholesky

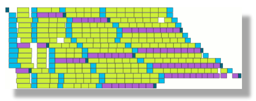
- Measured on Intel Tigerton 2.4 GHz

Cilk





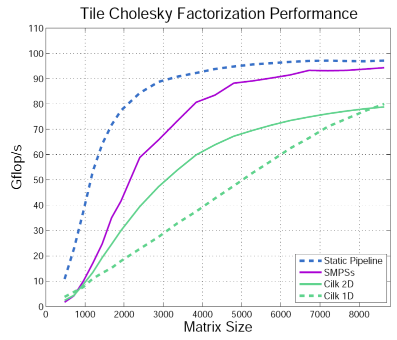
SMPSs



PLASMA (Static Pipeline)

Slide courtesy of Jakub Kurzak, UTK

Still More Measured Results for Tiled Cholesky



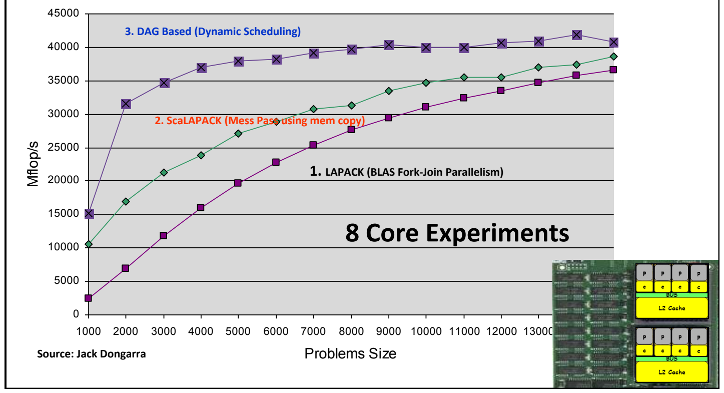
- PLASMA (static pipeline) – best
- SMPSs – somewhat worse
- Cilk 2D – inferior
- Cilk 1D – still worse

quad-socket, quad-core (16 cores total) Intel Tigerton 2.4 GHz

Slide courtesy of Jakub Kurzak, UTK

Intel's Clovertown Quad Core

3 Implementations of LU factorization
Quad core w/2 sockets per board, w/ 8 Threads



Scheduling on Multicore – Next Steps

- PLASMA 2.6.0 released 12/2013
  - Includes BLAS, Cholesky, QR, LU, LDL^T, eig, svd
  - icl.cs.utk.edu/plasma/
- Future of PLASMA
  - Continue adding functions
  - Add dynamic scheduling
    - QUARK dynamic scheduler released 12/2011
    - DAGs for eigenproblems are too complicated to do by hand
  - Still assume homogeneity of available cores
    - What about GPUs, or mixtures of CPUs and GPUs?
  - MAGMA
    - icl.cs.utk.edu/magma

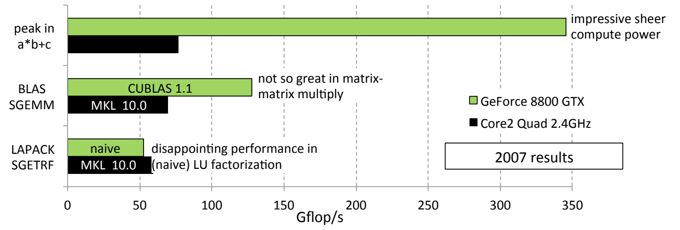
CLASS PROJECTS

Dense Linear Algebra on GPUs

- Source: Vasily Volkov's SC08 paper
  - Best Student Paper Award
- New challenges
  - More complicated memory hierarchy
  - Not like "L1 inside L2 inside …",
    - Need to choose which memory to use carefully
    - Need to move data manually
  - GPU does some operations much faster than CPU, but not all
  - CPU and GPU fastest using different data layouts

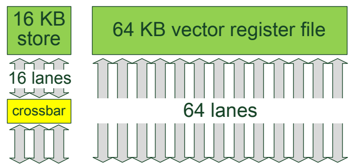
Motivation

- NVIDIA released CUBLAS 1.0 in 2007, which is BLAS for GPUs
- This enables a straightforward port of LAPACK to GPU
  - Consider single precision only



- Goal: understand bottlenecks in the dense linear algebra kernels
  - Requires detailed understanding of the GPU architecture
  - Result 1: New coding recommendations for high performance on GPUs
  - Result 2: New , fast variants of LU, QR, Cholesky, other routines

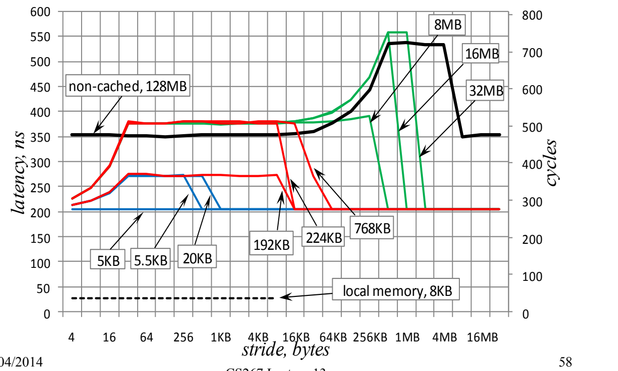
GPU Memory Hierarchy



- Register file is the fastest and the largest on-chip memory
  - Constrained to vector operations only
- Shared memory permits indexed and shared access
  - However, 2-4x smaller and 4x lower bandwidth than registers
    - Only 1 operand in shared memory is allowed versus 4 register operands
  - Some instructions run slower if using shared memory

Memory Latency on GeForce 8800 GTX

Repeat *k = A[k]* where *A[k] = (k + stride) mod array_size*



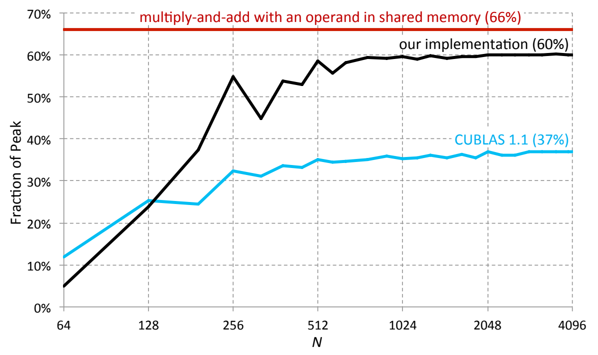
(Some new) NVIDIA coding recommendations

- Minimize communication with CPU memory
- Keep as much data in registers as possible
  - Largest, fastest on-GPU memory
  - Vector-only operations
- Use as little shared memory as possible
  - Smaller, slower than registers; use for communication, sharing only
  - Speed limit: 66% of peak with one shared mem argument
- Use vector length VL=64, not max VL = 512
  - Strip mine longer vectors into shorter ones
- Final matmul code similar to Cray X1 or IBM 3090 vector codes

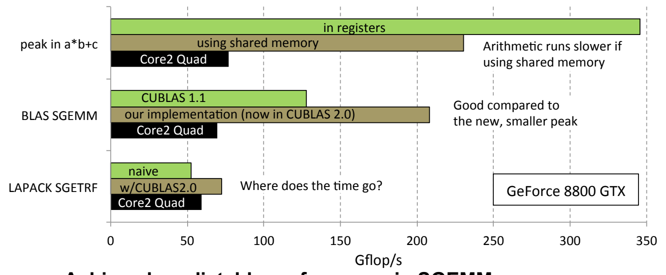
```
__global__ void sgemmNN( const float *A, int lda, const float *B, int ldb, float* C, int ldc, int k, float alpha, float beta )
{
    A += blockIdx.x * 64 + threadIdx.x + threadIdx.y*16;
    B += threadIdx.x + ( blockIdx.y * 16 + threadIdx.y ) * ldb;                  Compute pointers to the data
    C += blockIdx.x * 64 + threadIdx.x + (threadIdx.y + blockIdx.y * ldc ) * 16;
    __shared__ float bs[16][17];
    float c[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};           Declare the on-chip storage
    const float *Blast = B + k;
    do
    {
#pragma unroll
        for( int i = 0; i < 16; i += 4 )
            bs[threadIdx.x][threadIdx.y+i]  = B[i*ldb];          Read next B's block
        B += 16;
        __syncthreads();
#pragma unroll
        for( int i = 0; i < 16; i++, A += lda )
            {
            c[0] += A[0]*bs[i][0];    c[1] += A[0]*bs[i][1];    c[2] += A[0]*bs[i][2];    c[3] += A[0]*bs[i][3];
            c[4] += A[0]*bs[i][4];    c[5] += A[0]*bs[i][5];    c[6] += A[0]*bs[i][6];    c[7] += A[0]*bs[i][7];        The bottleneck:
            c[8] += A[0]*bs[i][8];    c[9] += A[0]*bs[i][9];    c[10] += A[0]*bs[i][10];c[11] += A[0]*bs[i][11];   Read A's columns
            c[12] += A[0]*bs[i][12];c[13] += A[0]*bs[i][13];c[14] += A[0]*bs[i][14];c[15] += A[0]*bs[i][15];      Do Rank-1 updates
    }
        __syncthreads();
    } while( B < Blast );
    for( int i = 0; i < 16; i++, C += ldc )
        C[0] = alpha*c[i] + beta*C[0];                    Store C's block to memory
}
```

New code vs. CUBLAS 1.1

Performance in multiplying two *NxN* matrices on GeForce 8800 GTX:



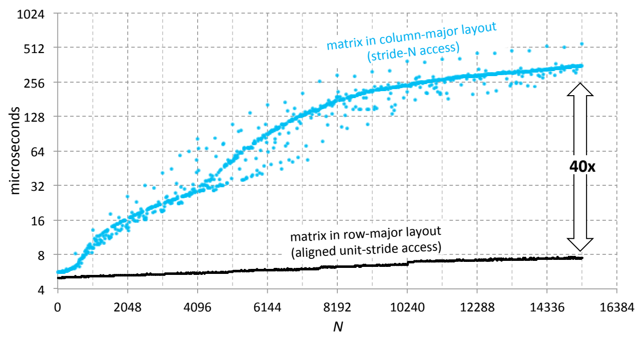
The Progress So Far



- **Achieved predictable performance in SGEMM**
  - **Which does $O(N^3)$ work in LU factorization**
- **But LU factorization (naïve SGETRF) still underperforms**
  - **Must be due to the rest $O(N^2)$ work done in BLAS1 and BLAS2**
  - **Why $O(N^2)$ work takes so much time?**

Row-Pivoting in LU Factorization

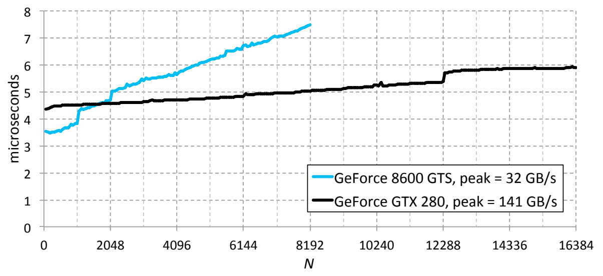
Exchange two rows of an *NxN* matrix (SSWAP in CUBLAS 2.0):



Row pivoting in column-major layout on GPU is very slow
This alone consumes half of the runtime in naïve SGETRF

BLAS1 Performance

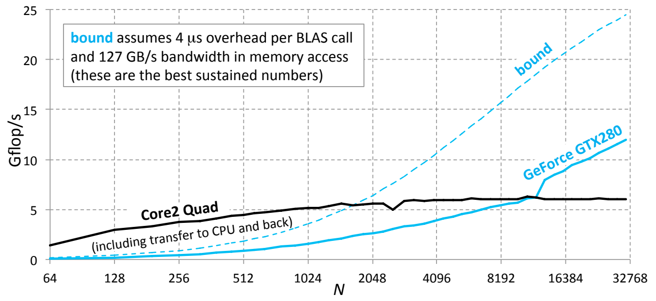
Scale a column of an *NxN* matrix that fits in the GPU memory (assumes aligned, unit-stride access)



- **Peak bandwidth of these GPUs differs by a factor of 4.4**
- **But runtimes are similar**
- **Small tasks on GPU are overhead bound**

Panel Factorization

Factorizing *N*x64 matrix in GPU memory using LAPACK's SGETF2:

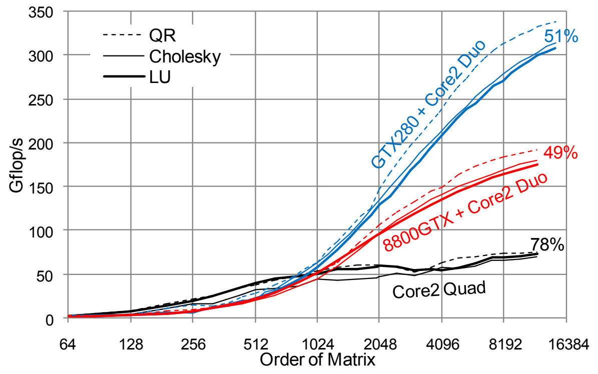


- **Invoking small BLAS operations on GPU from CPU is slow**
- **Can we call a sequence of BLAS operations from GPU?**
  - **Requires barrier synchronization after each parallel BLAS operation**
  - **Barrier is possible but requires sequential consistency for correctness**

Design of fast matrix factorizations on GPU

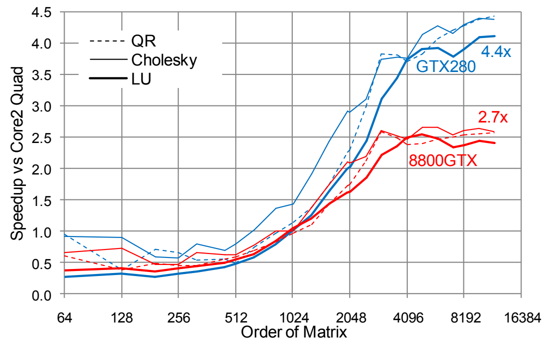
- **Use GPU for matmul only, not BLAS2 or BLAS1**
- **Factor panels on CPU**
- **Use "look-ahead" to overlap CPU and GPU work**
  - **GPU updates matrix while CPU factoring next panel**
- **Use row-major layout on GPU, column-major on CPU**
  - **Convert on the fly**
- **Substitute triangular solves LX= B with multiply by L^{-1}**
  - **For stability CPU needs to check || L^{-1} ||**
- **Use variable-sized panels for load balance**
- **For two GPUs with one CPU, use column-cyclic layout on GPUs**

Raw Performance of Factorizations on GPU



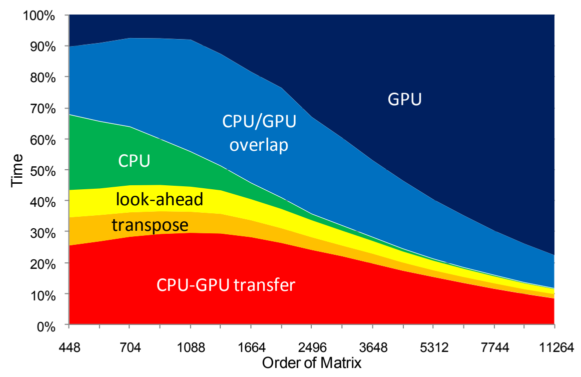
Speedup of Factorizations on GPU over CPU

GPU only useful on large enough matrices



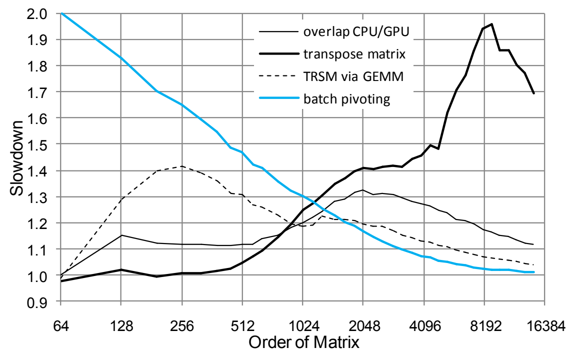
Where does the time go?

- Time breakdown for LU on 8800 GTX

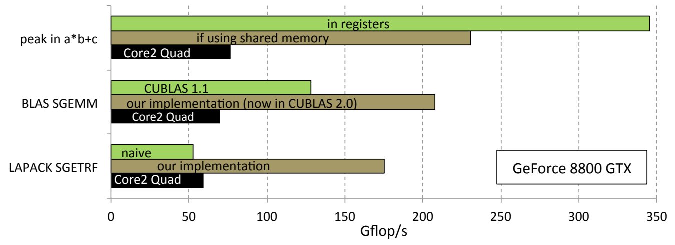


Importance of various optimizations on GPU

- Slowdown when omitting one of the optimizations on GTX 280



Results for matmul, LU on NVIDIA



- What we' ve achieved:
  - Identified realistic peak speed of GPU architecture
  - Achieved a large fraction of this peak in matrix multiply
  - Achieved a large fraction of the matrix multiply rate in dense factorizations

Class Projects

- Pick one (of many) functions/algorithms
- Pick a target parallel platform
- Pick a "parallel programming framework"
  - LAPACK – all parallelism in BLAS
  - ScaLAPACK – distributed memory using MPI
  - PLASMA – DAG scheduling on multicore
    - Parallel Linear Algebra for Scalable Multi-core Architectures
    - http://icl.cs.utk.edu/plasma/
  - MAGMA – DAG scheduling for heterogeneous platforms
    - Matrix Algebra on GPU and Multicore Architectures
    - http://icl.cs.utk.edu/magma/
  - Cloud
  - FLAME - http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage
- Design, implement, measure, model and/or compare performance
  - Can be missing entirely on target platform
  - May exist, but with a different programming framework

Extra Slides

What *could* go into a linear algebra library?

For all linear algebra problems

For all matrix/problem/data structures

For all data types

For all architectures and networks

For all programming interfaces

Produce best algorithm(s) w.r.t.
performance and accuracy
(including condition estimates, etc)

Need to prioritize, automate!

Other issues: dynamic resource allocation, fault tolerance, power
Many possible class projects

Missing Routines in Sca/LAPACK

		LAPACK	ScaLAPACK
Linear Equations	LU	xGESV	PxGESV
	LU + iterative refine	xGESVX	missing
	Cholesky	xPOSV	PxPOSV
	LDL^T	xSYSV	missing
Least Squares (LS)	QR	xGELS	PxGELS
	QR+pivot	xGELSY	missing
	SVD/QR	xGELSS	missing
	SVD/D&C	xGELSD	missing (intent?)
	SVD/MRRR	missing	missing
	QR + iterative refine.	missing	missing
Generalized LS	LS + equality constr.	xGGLSE	missing
	Generalized LM	xGGGLM	missing
	Above + Iterative ref.	missing	missing

More missing routines

		LAPACK	ScaLAPACK
Symmetric EVD	QR / Bisection+Invit	xSYEV / X	PxSYEV / X
	D&C	xSYEVD	PxSYEVD
	MRRR	xSYEVR	missing
Nonsymmetric EVD	Schur form	xGEES / X	missing (driver)
	Vectors too	xGEEV /X	missing
SVD	QR	xGESVD	PxGESVD
	D&C	xGESDD	missing (intent?)
	MRRR	missing	missing
	Jacobi	xGESVJ	missing
Generalized Symmetric EVD	QR / Bisection+Invit	xSYGV / X	PxSYGV / X
	D&C	xSYGVD	missing (intent?)
	MRRR	missing	missing
Generalized Nonsymmetric EVD	Schur form	xGGES / X	missing
	Vectors too	xGGEV / X	missing
Generalized SVD	Kogbetliantz	xGGSVD	missing (intent)
	MRRR	missing	missing

Possible class projects

- GPU related
  - Study available libraries, what's missing
  - Try new, unimplemented routines, compare performance
- Filling in gaps in ScaLAPACK/PLASMA/MAGMA libraries
  - User demand for various missing routines
  - LDL^T, QRP, updating/downdating, Eigenvalues, SVD, band routines, packed storage, error bounds
- "Communication avoiding" algorithms
  - Implement, compare performance to Sca/LAPACK
  - Algorithms that minimize communication over multiple levels of memory hierarchy or of parallelism
  - Algorithms that minimize time (including communication) on heterogeneous platforms
- Compare parallel programming frameworks
  - Compare performance/complexity of implementations in different frameworks
- More details available

Exploring the tuning space for Dense LA

- Algorithm tuning space includes
  - Underlying BLAS (PHiPAC, ATLAS)
  - Different layouts (blocked, recursive, …) and algorithms
  - Numerous block sizes, not just in underlying BLAS
  - Many possible layers of parallelism, many mappings to HW
  - Different traversals of underlying DAGs
    - Synchronous and asynchronous algorithms
  - "Redundant" algorithms for GPUs
  - New and old eigenvalue algorithms
  - Mixed precision (for speed or accuracy)
  - New "communication avoiding" algorithms for variations on standard factorizations
- Is there a concise set of abstractions to describe, generate tuning space?
  - Block matrices, factorizations (partial, tree, …), DAGs, …
  - PLASMA, FLAME, CSS, Spiral, Sequoia, Telescoping languages, Bernoulli, Rose, …
- Question: What fraction of dense linear algebra can be generated/tuned?
  - Lots more than when we started
    - Sequential BLAS -> Parallel BLAS -> LU -> other factorizations -> …
  - Most of dense linear algebra?
    - Not eigenvalue algorithms (on compact forms)
    - What fraction of LAPACK can be done?
    - "for all linear algebra problems…"
  - For all interesting architectures…?

ScaLAPACK Performance Models (1)

ScaLAPACK Operation Counts

$$T(N,P) = \frac{C_f\,N^3}{P}\,t_f + \frac{C_v\,N^2}{\sqrt{P}}\,t_v + \frac{C_m\,N}{NB}\,t_m, \qquad T_{seq}(N,P) = C_f\,N^3\,t_f.$$

$$E(N,P) = \left(1 + \frac{1}{NB}\,\frac{C_m\,t_m}{C_f\,t_f}\,\frac{P}{N^2} + \frac{C_v\,t_v}{C_f\,t_f}\,\frac{\sqrt{P}}{N}\right)^{-1}.$$

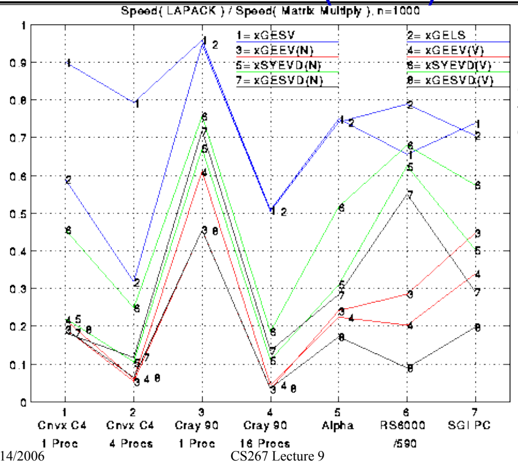
t_f = 1
t_m = α
t_v = β
NB = brow=bcol
√P = prow = pcol

Driver	Options	C_f	C_v	C_m
PxGESV	1 right hand side	2/3	$3 + 1/4\,\log_2 P$	$NB\,(6 + \log_2 P)$
PxPOSV	1 right hand side	1/3	$2 + 1/2\,\log_2 P$	$4 + \log_2 P$
PxGELS	1 right hand side	4/3	$3 + \log_2 P$	$2\,(NB\,\log_2 P + 1)$
PxSYEVX	eigenvalues only	4/3	$5/2\log_2 P$	$17/2\,NB + 2$
PxSYEVX	eigenvalues and eigenvectors	10/3	$5\log_2 P$	$17/2\,NB + 2$
PxSYEV	eigenvalues only	4/3	$5/2\log_2 P$	$17/2\,NB + 2$
PxSYEV	eigenvalues and eigenvectors	22/3	$5\log_2 P$	$17/2\,NB + 2$
PxGESVD	singular values only	26/3	$10\log_2 P$	$17NB$
PxGESVD	singular values and left and right singular vectors	38/3	$14\log_2 P$	$17NB$
PxLAHQR	eigenvalues only	5	$9/2(\sqrt{P}) * \log_2 P$ $+8\,N/NB$	$9\,(2 + \log_2 P)\,N$
PxLAHQR	full Schur form	18	$9/2(\sqrt{P}) * \log_2 P$ $+8\,N/NB$	$9\,(2 + \log_2 P)\,N$

Overview of LAPACK and ScaLAPACK

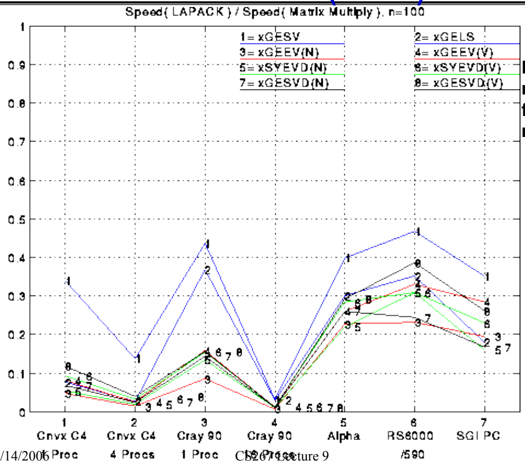
- Standard library for dense/banded linear algebra
  - Linear systems: A*x=b
  - Least squares problems: min$_x$ || A*x-b ||$_2$
  - Eigenvalue problems: Ax = λx, Ax = λBx
  - Singular value decomposition (SVD): A = UΣV^T
- Algorithms reorganized to use BLAS3 as much as possible
- Basis of math libraries on many computers, Matlab …
- Many algorithmic innovations remain
  - Projects available

Performance of LAPACK (n=1000)



Performance of Eigen-values, SVD, etc.

Performance of LAPACK (n=100)

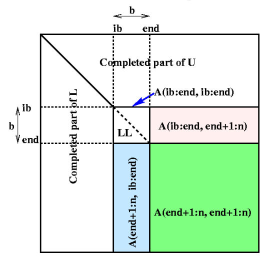


Efficiency is much lower for a smaller matrix.

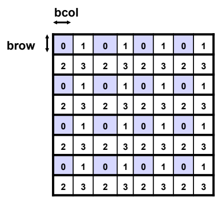
Review: BLAS 3 (Blocked) GEPP

```
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^-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
```

Gaussian Elimination using BLAS 3



Row and Column Block Cyclic Layout



0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

- 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 ≠ pcol and brow ≠ bcol possible, even desireable

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=bcol.
- shaded regions indicate processors busy with computation or communication.
- unnecessary to have a barrier between each step of the algorithm, e.g.. steps 9, 10, and 11 can be pipelined

ScaLAPACK Performance Models (2)

Compare Predictions and Measurements

IBM SP2[a]	P	Values of N									
		2000		5000		7500		10000		15000	
		Est	Obt	Est	Obt	Est	Obt	Est	Obt	Est	Obt
PDGESV	4	357	421	632	603						
(LU)	16	497	722	1581	1543	2116	1903	2424	2149		
	64	502	924	2432	3017	4235	4295	5793	5596	7992	7057
PDPOSV	4	530	462	669	615						
(Cholesky)	16	1315	1081	2083	1811	2366	2118	2535	2312		
	64	2577	1807	5327	4431	6709	5727	7661	6826	8887	8084

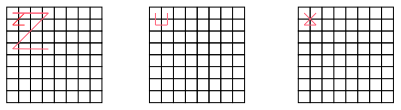
[a]One process spawned per node and one computational IBM POWER2 590 processor per node.

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

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
  - "Recursive Block Algorithms and Hybrid Data Structures," Elmroth, Gustavson, Jonsson, Kagstrom, SIAM Review, 2004
  - http://www.cs.umu.se/research/parallel/recursion/

Gaussian Elimination via a Recursive Algorithm

F. Gustavson and S. Toledo

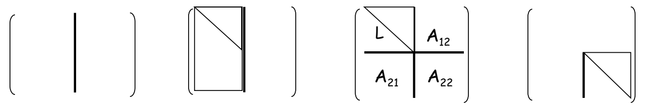
LU Algorithm:

```
1: Split matrix into two rectangles (m x 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
```

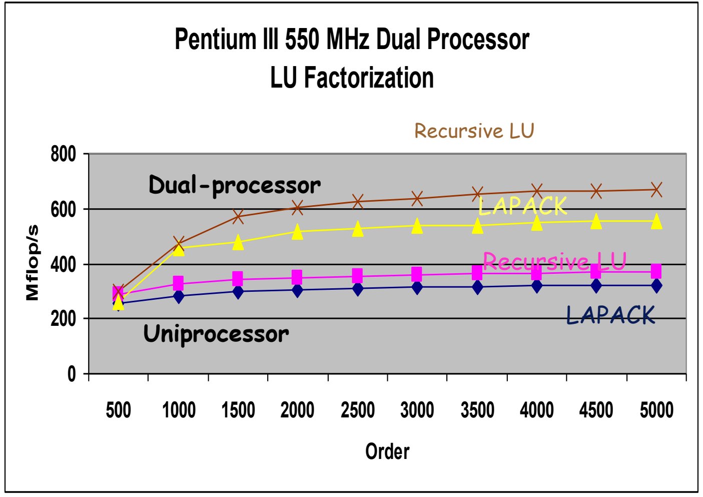


Most of the work in the matrix multiply
Matrices of size n/2, n/4, n/8, …

Source: Jack Dongarra

Recursive Factorizations

- Just as accurate as conventional method
- Same number of operations
- Automatic variable-size blocking
  - Level 1 and 3 BLAS only !
- Simplicity of expression
- Potential for efficiency while being “cache oblivious”
  - But shouldn’t recur down to single columns!
- 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).



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)

Out of “Core” Algorithms

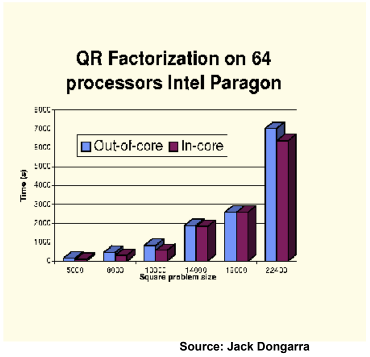
Out-of-Core Performance Results for Least Squares

- 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 memory

Much harder to hide latency of disk

QR much easier than LU because no pivoting needed for QR



Source: Jack Dongarra

Some contributors (incomplete list)

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

Upcoming related talks

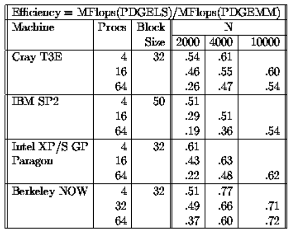
- SIAM Conference on Parallel Processing in Scientific Computing
  - San Francisco, Feb 22-24
  - http://www.siam.org/meetings/pp06/index.htm
  - Applications, Algorithms, Software, Hardware
  - 3 Minisymposia on Dense Linear Algebra on Friday 2/24
    - MS41, MS47(*), MS56
- Scientific Computing Seminar,
  - “An O(n log n) tridiagonal eigensolver”, Jonathan Moussa
  - Wednesday, Feb 15, 11-12, 380 Soda
- Special Seminar
  - Towards Combinatorial Preconditioners for Finite-Elements Problems”, Prof. Sivan Toledo, Technion
  - Tuesday, Feb 21, 1-2pm, 373 Soda

Extra Slides

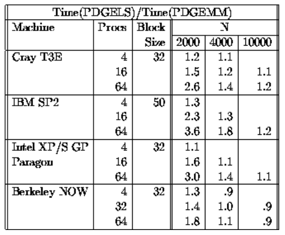
QR (Least Squares)

Scales well,
nearly full machine speed

Performance of ScaLAPACK QR (Least squares)



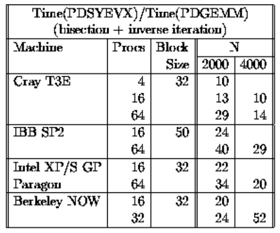
Efficiency = MFlops(PDGELS)/MFlops(PDGEMM)					
Machine	Procs	Block	N		
		Size	2000	4000	10000
Cray T3E	4	32	.54	.61	
	16		.46	.55	.60
	64		.26	.47	.54
IBM SP2	4	50	.51		
	16		.29	.51	
	64		.19	.36	.54
Intel XP/S GP Paragon	4	32	.61		
	16		.43	.63	
	64		.22	.48	.62
Berkeley NOW	4	32	.51	.77	
	32		.49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



Time(PDGELS)/Time(PDGEMM)					
Machine	Procs	Block	N		
		Size	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

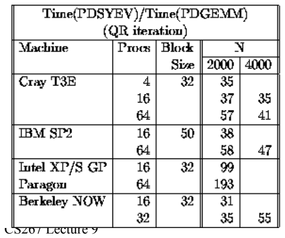
Performance of Symmetric Eigensolvers

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



Time(PDSYEVX)/Time(PDGEMM) (bisection + inverse iteration)				
Machine	Procs	Block	N	
		Size	2000	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

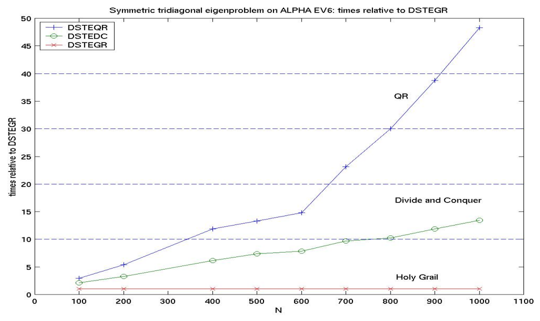
Initial algorithm:
Numerically stable
Easily parallelized
Slow: will abandon



Time(PDSYEV)/Time(PDGEMM) (QR iteration)				
Machine	Procs	Block	N	
		Size	2000	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

Scalable Symmetric Eigensolver and SVD

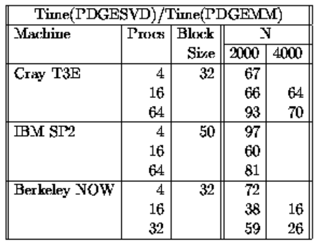
The "Holy Grail" (Parlett, Dhillon, Marques)
Perfect Output complexity (O(n * #vectors)), Embarrassingly parallel, Accurate



To be propagated throughout LAPACK and ScaLAPACK

Performance of SVD
(Singular Value Decomposition)

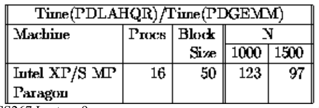
Have good ideas to speedup
Project available!



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

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 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

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?

Computational Electromagnetics (MOM)

The main steps in the solution process are

- Fill: computing the matrix elements of A
- Factor: factoring the dense matrix A
- Solve: solving for one or more excitations b
- Field Calc: computing the fields scattered from the object

Analysis of MOM for Parallel Implementation

Task	Work	Parallelism	Parallel Speed
Fill	O(n**2)	embarrassing	low
→ Factor	O(n**3)	moderately diff.	very high
Solve	O(n**2)	moderately diff.	high
Field Calc.	O(n)	embarrassing	high

BLAS2 version of GE 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
```

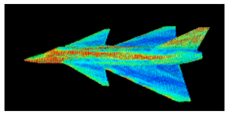
Computational Electromagnetics – Solve Ax=b

- •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

Computational Electromagnetics

- Discretize surface into triangular facets using standard modeling tools

- Amplitude of currents on surface are unknowns



- Integral equation is discretized into a set of linear equations

image: NW Univ. Comp. Electromagnetics Laboratory http://nueml.ece.nwu.edu/

Computational Electromagnetics (MOM)

After discretization the integral equation has the form

$$A\ x\ =\ b$$

where

A is the (dense) impedance matrix,

x is the unknown vector of amplitudes, and

b is the excitation vector.

(see Cwik, Patterson, and Scott, Electromagnetic Scattering on the Intel Touchstone Delta, IEEE Supercomputing '92, pp 538 - 542)

Results for Parallel Implementation on Intel Delta

Task	Time (hours)
Fill (compute n^2 matrix entries)	9.20
(embarrassingly parallel but slow)	
Factor (Gaussian Elimination, $O(n^3)$)	8.25
(good parallelism with right algorithm)	
Solve ($O(n^2)$)	2 .17
(reasonable parallelism with right algorithm)	
Field Calc. ($O(n)$)	0.12
(embarrassingly parallel and fast)	

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

Computational Chemistry – Ax = λ x

- Seek energy levels of a molecule, crystal, etc.
  - Solve Schroedinger's Equation for energy levels = eigenvalues
  - Discretize to get $Ax = \lambda Bx$, solve for eigenvalues λ and eigenvectors x
  - A and B large 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

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

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

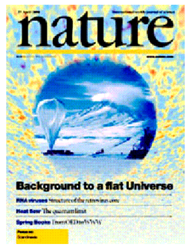
Winner of TOPS 500 (LINPACK Benchmark)

Year	Machine	Tflops	Factor faster	Peak Tflops	Num Procs	N
2004	Blue Gene / L, IBM	70.7	2.0	91.8	32768	.93M
2002 2003	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)

Success Stories for Sca/LAPACK

- Widely used
  - Adopted by Mathworks, Cray, Fujitsu, HP, IBM, IMSL, NAG, NEC, SGI, …
  - >84M(56M in 2006) web hits @ Netlib (incl. CLAPACK, LAPACK95)
- New Science discovered through the solution of dense matrix systems
  - 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/newNERSCresults050703.pdf



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

ScaLAPACK

Motivation (1)

3 Basic Linear Algebra Problems

1. Linear Equations: Solve Ax=b for x
2. Least Squares: Find x that minimizes $||r||_2 \equiv \sqrt{\Sigma\, r_i^2}$ where r=Ax-b
   - Statistics: Fitting data with simple functions

3a. Eigenvalues: Find λ and x where Ax = λ x
   - Vibration analysis, e.g., earthquakes, circuits

3b. Singular Value Decomposition: $A^TAx=\sigma^2x$
   - Data fitting, Information retrieval

Lots of variations depending on structure of A
   - A symmetric, positive definite, banded, …

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

Current Records for Solving Dense Systems (2007)

www.netlib.org, click on Performance Database Server

Machine	Gigaflops n=100	n=1000	Any n	Peak
IBM BlueGene/L (213K procs)			478K (478 Teraflops) (n=2.5M)	596K
NEC SX 8				
(8 proc, 2 GHz)		75.1		128
(1 proc, 2 GHz)	2.2	15.0		16
…				
Palm Pilot III	.00000169 (1.69 Kiloflops)			

Making TSLU Numerically Stable

- Stability Goal: Make ||A – PLU|| very small: O(machine_precision · ||A||)
- Details matter
  - Going up the tree, we could do LU either on original rows of W, or computed rows of U
  - Only first choice stable
- Thm: New scheme as stable as Partial Pivoting (PP) in following sense: get same results as PP applied to different input matrix whose entries are blocks taken from input A
- CALU – Communication Avoiding LU for general A
  - Use TSLU for panel factorizations
  - Apply to rest of matrix
  - Cost: redundant panel factorizations (extra $O(n^2)$ flops – ok)
- Benefit:
  - Stable in practice, but *not* same pivot choice as GEPP
  - One reduction operation per panel: reduces latency to minimum

Making TSLU Stable

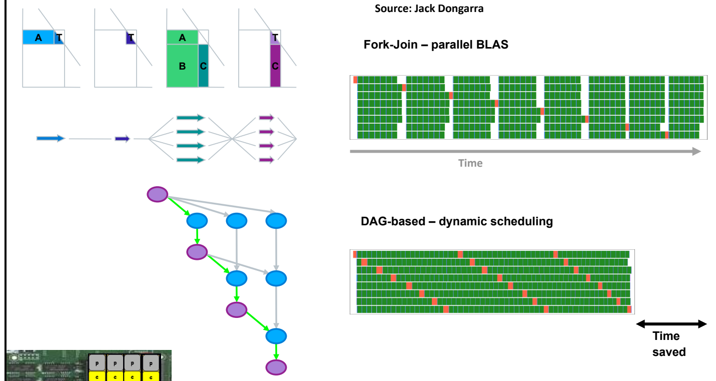
- Break n x b panel into $P^{1/2}$ submatrices of size n/ $P^{1/2}$ x b each
  - Think of each submatrix assigned to leaf of binary tree
- At each leaf, run GE with partial pivoting (GEPP) to identify b "good" pivot rows
- At each internal tree node, TSLU selects b pivot rows from 2b candidates from its 2 child nodes
  - Does this by running GEPP on 2b *original* rows selected by child nodes
- When TSLU done, permute b selected rows to top of original matrix, redo b steps of LU without pivoting
- Thm: Same results as GEPP on different input matrix whose entries are the same magnitudes as original
- CALU – Communication Avoiding LU for general A
  - Use TSLU for panel factorizations
  - Apply to rest of matrix
  - Cost: redundant panel factorizations
- Benefit:
  - Stable in practice, but *not* same pivot choice as GEPP
  - One reduction operation per panel

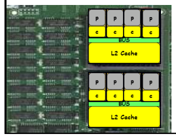
LAPACK and ScaLAPACK Scalability

- "One-sided Problems" are scalable
  - Linear systems Ax=b, and least squares $\min_x ||Ax-b||_2$
  - 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
  - Can minimize communication, some open problems:
    - Multiple levels of memory hierarchy
    - "Heterogeneous" platforms with multiple speeds (eg CPU+GPU)
- "Two-sided Problems" are harder
  - Eigenvalue problems, SVD
  - A factored into product of 3 matrices by pre and post multiplication
  - ~Half BLAS2, not all BLAS3
- Narrow band problems hardest (to do BLAS3 or parallelize)
  - Solving and eigenvalue problems

Fork-Join vs. Dynamic Execution on Multicore

Source: Jack Dongarra





Experiments on
Intel's Quad Core Clovertown
with 2 Sockets w/ 8 Treads

121

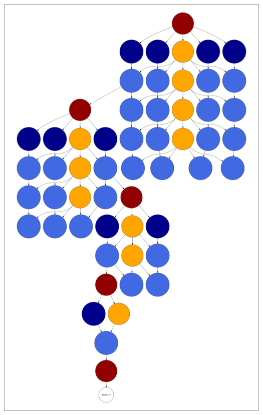
Achieving Asynchronicity on Multicore

Source: Jack Dongarra

The matrix factorization can be represented as a DAG:
- nodes: tasks that operate on "tiles"
- edges: dependencies among tasks

Tasks can be scheduled asynchronously and in any order as long as dependencies are not violated.

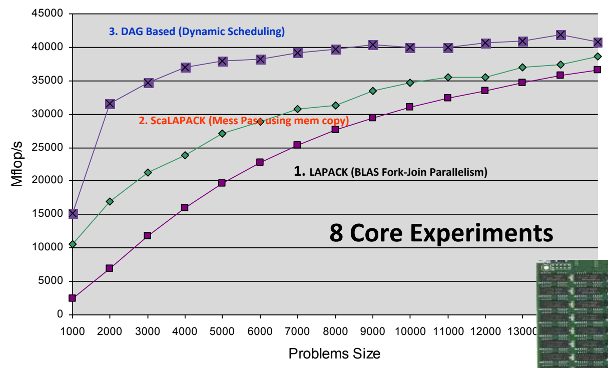
Systems: PLASMA for multicore
MAGMA for CPU/GPU

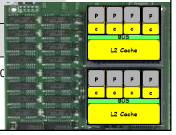


Intel's Clovertown Quad Core

Source: Jack Dongarra

3 Implementations of LU factorization
Quad core w/2 sockets per board, w/ 8 Treads

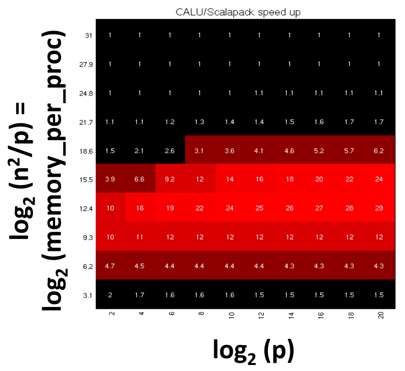




Exascale Machine Parameters

- 2^30 ≈ 1,000,000 nodes
- 1024 cores/node (a billion cores!)
- 100 GB/sec interconnect bandwidth
- 400 GB/sec DRAM bandwidth
- 1 microsec interconnect latency
- 50 nanosec memory latency
- 32 Petabytes of memory
- 1/2 GB total L1 on a node

Exascale predicted speedups for CA-LU vs ScaLAPACK-LU



Which algs for LU (and QR) reach lower bounds?

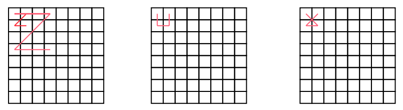
- LU for solving Ax=b, QR for least squares
- LAPACK attains neither, depending on relative size of M, n
- Recursive sequential algs minimize bandwidth, not latency
  - Toledo for LU, Elmroth/Gustavson for QR
- ScaLAPACK attains bandwidth lower bound
  - But sends too many messages
- New LU and QR algorithms do attain both lower bounds, both sequential and parallel
  - LU: need to abandon partial pivoting (but still stable)
  - QR: similar idea of reduction tree as for LU
  - Neither new alg works for multiple memory hierarchy levels
    - Open question!
  - See EECS TR 2008-89 for QR, SC08 paper for LU

Do any Cholesky algs reach lower bounds?

- Cholesky factors A = LL^T , for Ax=b when A=A^T and positive definite
  - Easier: Like LU, but half the arithmetic and no pivoting
- LAPACK (with right block size) or recursive Cholesky minimize bandwidth
  - Recursive: Ahmed/Pingali, Gustavson/Jonsson, Andersen/Gustavson/Wasniewski, Simecek/Tvrdik, a la Toledo
- LAPACK can minimize latency with blocked data structure
- Ahmed/Pingali minimize bandwidth and latency across multiple levels of memory hierarchy
  - Simultaneously minimize communication between all pairs L1/L2/L3/DRAM/disk/…
  - "Space-filling curve layout", "Cache-oblivious"
- ScaLAPACK minimizes bandwidth and latency (mod log P)
  - Need right choice of block size
- Details in EECS TR 2009-29

Space-Filling Curve Layouts

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



- Other variations possible
- What about the users?
  - Copy data into new format before solving?

Summary of dense *sequential* algorithms attaining communication lower bounds

- Algorithms shown minimizing # Messages use (recursive) block layout
  - Not possible with columnwise or rowwise layouts
- *Many* references (see reports), only some shown, plus ours
- Cache-oblivious are underlined, Green are ours, ? is unknown/future work

Algorithm	2 Levels of Memory		Multiple Levels of Memory	
	#Words Moved	and # Messages	#Words Moved	and #Messages
BLAS-3				
Cholesky				
LU with pivoting				
QR				
Eig, SVD				

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:
  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply			
Cholesky			
LU			
QR			
Sym Eig, SVD			
Nonsym Eig			

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:
  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	[illegible]	1
Cholesky	ScaLAP[illegible]	[illegible]	log P
LU	[illegible]	log P log P	log P $(n / P^{1/2}) \cdot \log P$
QR	[illegible] [illegible]APACK	log P log P	$\log^3 P$ $(n / P^{1/2}) \cdot \log P$
Sym Eig, SV[illegible]	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $n / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $n \cdot \log P$

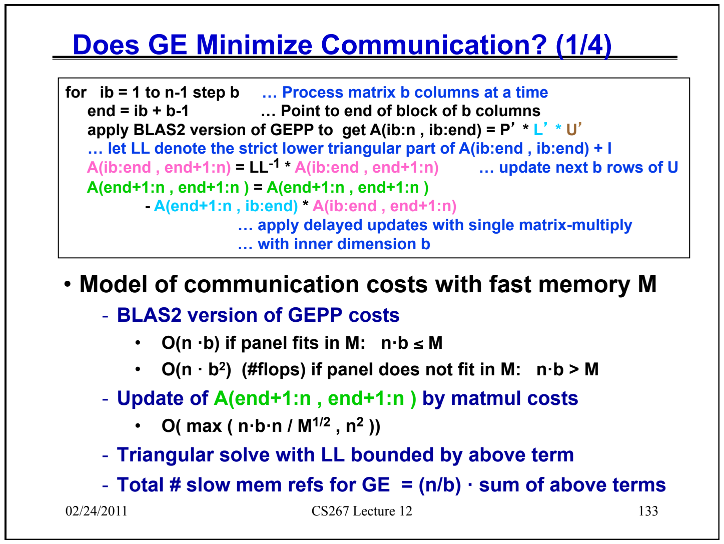
Can we do better?

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$?
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:
  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	[illegible]	1
Cholesky	ScaLAP[illegible]	[illegible]	log P
LU	[illegible]	log P log P	log P $(n / P^{1/2}) \cdot \log P$
QR	[illegible] [illegible]APACK	log P log P	$\log^3 P$ $(n / P^{1/2}) \cdot \log P$
Sym Eig, SV[illegible]	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $n / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $n \cdot \log P$

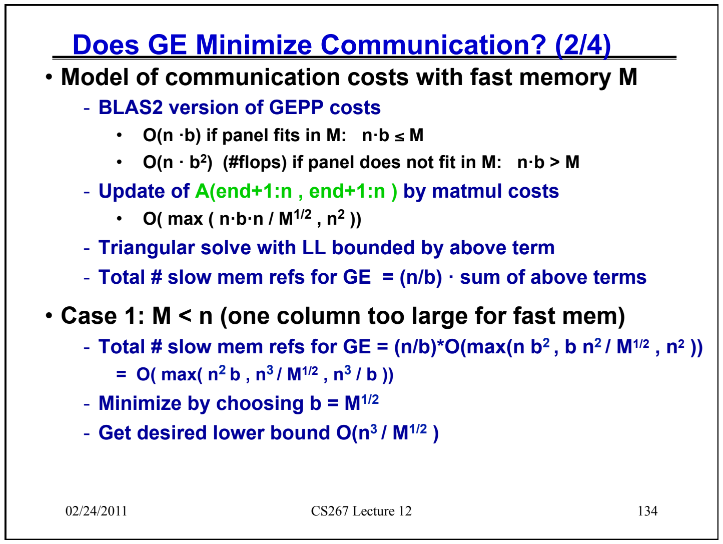
Can we do better?



Does GE Minimize Communication? (1/4)

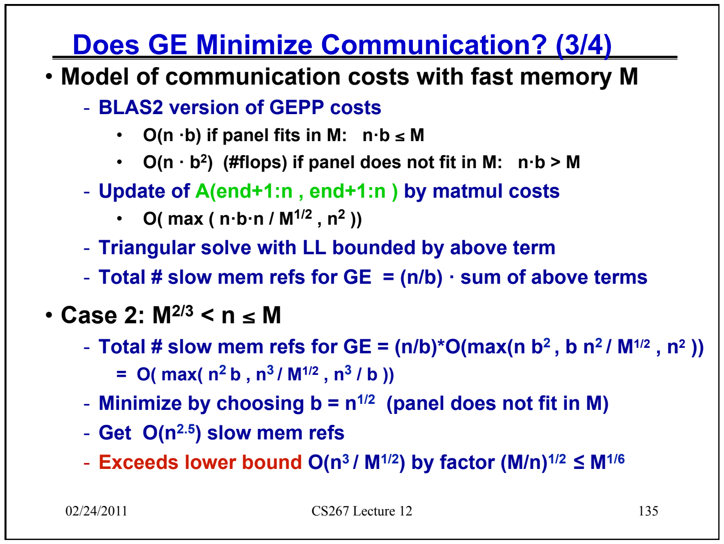
```
for  ib = 1 to n-1 step b     … Process matrix b columns at a time
   end = ib + b-1             … Point to end of block of b columns
   apply BLAS2 version of GEPP to  get A(ib:n , ib:end) = P’ * L’ * U’
   … let LL denote the strict lower triangular part of A(ib:end , ib:end) + I
   A(ib:end , end+1:n) = LL^{-1} * A(ib:end , end+1:n)        … update next b rows of U
   A(end+1:n , end+1:n ) = A(end+1:n , end+1:n )
           - A(end+1:n , ib:end) * A(ib:end , end+1:n)
                       … apply delayed updates with single matrix-multiply
                       … with inner dimension b
```

- Model of communication costs with fast memory M
  - BLAS2 version of GEPP costs
    - $O(n \cdot b)$ if panel fits in M: $n \cdot b \le M$
    - $O(n \cdot b^2)$ (#flops) if panel does not fit in M: $n \cdot b > M$
  - Update of A(end+1:n , end+1:n) by matmul costs
    - $O(\max (n \cdot b \cdot n / M^{1/2} , n^2))$
  - Triangular solve with LL bounded by above term
  - Total # slow mem refs for GE $= (n/b) \cdot$ sum of above terms



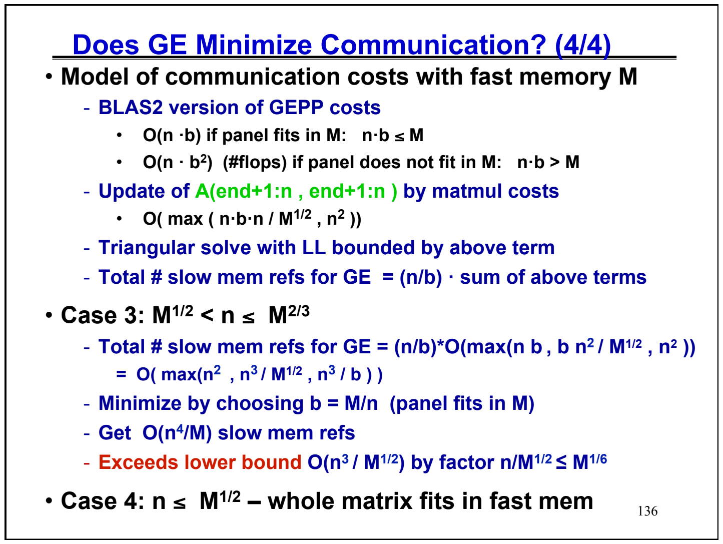
Does GE Minimize Communication? (2/4)

- Model of communication costs with fast memory M
  - BLAS2 version of GEPP costs
    - $O(n \cdot b)$ if panel fits in M: $n \cdot b \le M$
    - $O(n \cdot b^2)$ (#flops) if panel does not fit in M: $n \cdot b > M$
  - Update of A(end+1:n , end+1:n) by matmul costs
    - $O(\max (n \cdot b \cdot n / M^{1/2} , n^2))$
  - Triangular solve with LL bounded by above term
  - Total # slow mem refs for GE $= (n/b) \cdot$ sum of above terms
- Case 1: $M < n$ (one column too large for fast mem)
  - Total # slow mem refs for GE $= (n/b)*O(\max(n\, b^2 , b\, n^2 / M^{1/2} , n^2))$
    $= O(\max(n^2 b , n^3 / M^{1/2} , n^3 / b))$
  - Minimize by choosing $b = M^{1/2}$
  - Get desired lower bound $O(n^3 / M^{1/2})$



Does GE Minimize Communication? (3/4)

- Model of communication costs with fast memory M
  - BLAS2 version of GEPP costs
    - $O(n \cdot b)$ if panel fits in M: $n \cdot b \le M$
    - $O(n \cdot b^2)$ (#flops) if panel does not fit in M: $n \cdot b > M$
  - Update of A(end+1:n , end+1:n) by matmul costs
    - $O(\max (n \cdot b \cdot n / M^{1/2} , n^2))$
  - Triangular solve with LL bounded by above term
  - Total # slow mem refs for GE $= (n/b) \cdot$ sum of above terms
- Case 2: $M^{2/3} < n \le M$
  - Total # slow mem refs for GE $= (n/b)*O(\max(n\, b^2 , b\, n^2 / M^{1/2} , n^2))$
    $= O(\max(n^2 b , n^3 / M^{1/2} , n^3 / b))$
  - Minimize by choosing $b = n^{1/2}$ (panel does not fit in M)
  - Get $O(n^{2.5})$ slow mem refs
  - Exceeds lower bound $O(n^3 / M^{1/2})$ by factor $(M/n)^{1/2} \le M^{1/6}$



Does GE Minimize Communication? (4/4)

- Model of communication costs with fast memory M
  - BLAS2 version of GEPP costs
    - $O(n \cdot b)$ if panel fits in M: $n \cdot b \le M$
    - $O(n \cdot b^2)$ (#flops) if panel does not fit in M: $n \cdot b > M$
  - Update of A(end+1:n , end+1:n) by matmul costs
    - $O(\max (n \cdot b \cdot n / M^{1/2} , n^2))$
  - Triangular solve with LL bounded by above term
  - Total # slow mem refs for GE $= (n/b) \cdot$ sum of above terms
- Case 3: $M^{1/2} < n \le M^{2/3}$
  - Total # slow mem refs for GE $= (n/b)*O(\max(n\, b , b\, n^2 / M^{1/2} , n^2))$
    $= O(\max(n^2 , n^3 / M^{1/2} , n^3 / b))$
  - Minimize by choosing $b = M/n$ (panel fits in M)
  - Get $O(n^4/M)$ slow mem refs
  - Exceeds lower bound $O(n^3 / M^{1/2})$ by factor $n/M^{1/2} \le M^{1/6}$
- Case 4: $n \le M^{1/2}$ – whole matrix fits in fast mem