Introduction to Communication-Avoiding Algorithms

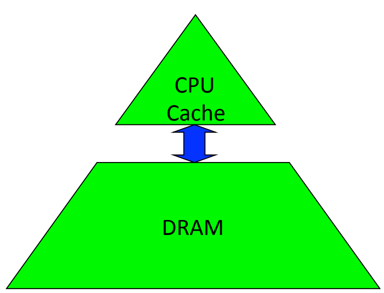
www.cs.berkeley.edu/~demmel

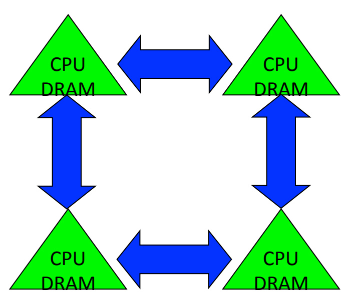
Jim Demmel, and many, many others …

Why avoid communication? (1/2)

Algorithms have two costs (measured in time or energy):

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).





Why avoid communication? (2/3)

- Running time of an algorithm is sum of 3 terms:
  – # flops * time_per_flop
  – # words moved / bandwidth
  – # messages * latency

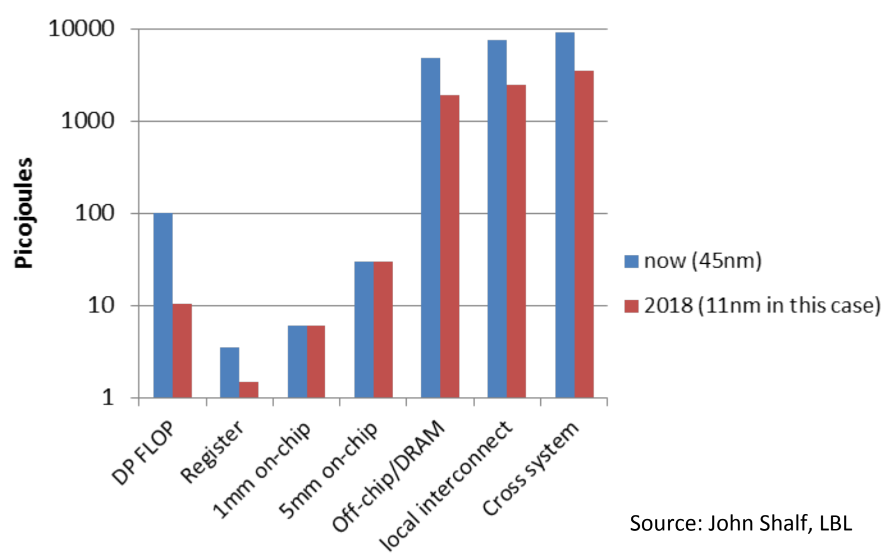
  communication
- Time_per_flop << 1/ bandwidth << latency
  - Gaps growing exponentially with time [FOSC]

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- Avoid communication to save time

3

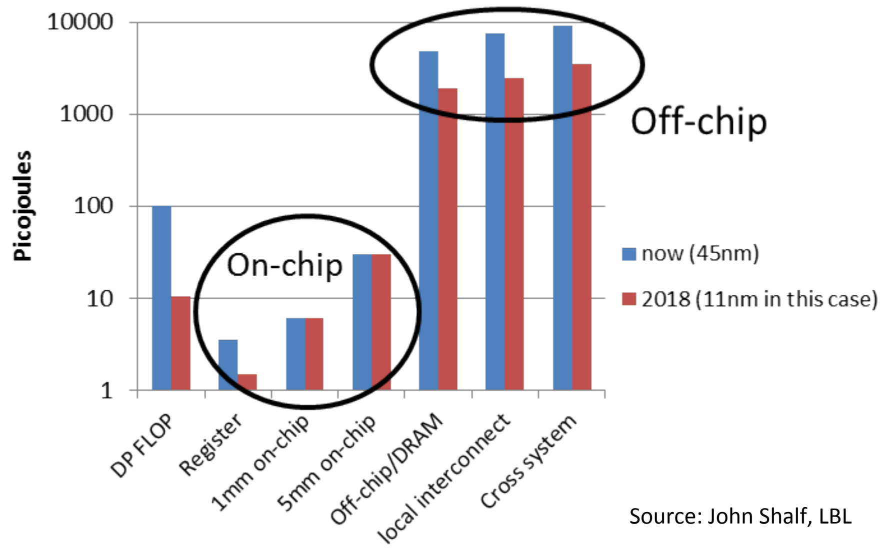
Why Minimize Communication? (2/2)



Source: John Shalf, LBL

Why Minimize Communication? (2/2)

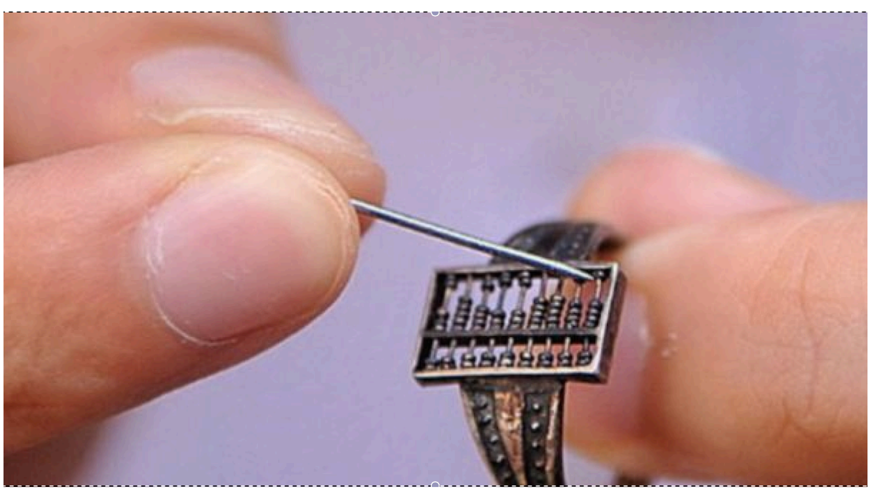
Minimize communication to save energy



Source: John Shalf, LBL

Alternative Cost Model for Algorithms?

Total distance moved by beads on an abacus



Goals

- Redesign algorithms to *avoid* communication
  - Between all memory hierarchy levels
    - L1 ↔ L2 ↔ DRAM ↔ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible
- Lots of open problems / potential class projects

7

Sample Speedups

- Up to **12x** faster for 2.5D matmul on 64K core IBM BG/P
- Up to **3x** faster for tensor contractions on 2K core Cray XE/6
- Up to **6.2x** faster for All-Pairs-Shortest-Path on 24K core Cray CE6
- Up to **2.1x** faster for 2.5D LU on 64K core IBM BG/P
- Up to **11.8x** faster for direct N-body on 32K core IBM BG/P
- Up to **13x** faster for Tall Skinny QR on Tesla C2050 Fermi NVIDIA GPU
- Up to **6.7x** faster for symeig(band A) on 10 core Intel Westmere
- Up to **2x** faster for 2.5D Strassen on 38K core Cray XT4
- Up to **4.2x** faster for MiniGMG benchmark bottom solver, using CA-BiCGStab (**2.5x** for overall solve)
  – **2.5x** / **1.5x** for combustion simulation code

8

President Obama cites Communication-Avoiding Algorithms in the FY 2012 Department of Energy Budget Request to Congress:

"New Algorithm Improves Performance and Accuracy on Extreme-Scale Computing Systems. **On modern computer architectures, communication between processors takes longer than the performance of a floating point arithmetic operation by a given processor.** ASCR researchers have developed a new method, derived from commonly used linear algebra methods, to **minimize communications between processors and the memory hierarchy, by reformulating the communication patterns specified within the algorithm**. This method has been implemented in the TRILINOS framework, a highly-regarded suite of software, which provides functionality for researchers around the world to solve large scale, complex multi-physics problems."

FY 2010 Congressional Budget, Volume 4, FY2010 Accomplishments, Advanced Scientific Computing Research (ASCR), pages 65-67.

CA-GMRES (Hoemmen, Mohiyuddin, Yelick, JD)
"Tall-Skinny" QR (Grigori, Hoemmen, Langou, JD)

Summary of CA Algorithms

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible
  - Autotuning to find optimal implementation
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra

Outline

- "Direct" Linear Algebra
  - Lower bounds on communication
  - New algorithms that attain these lower bounds
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra
- Related work

Outline

- "Direct" Linear Algebra
  - Lower bounds on communication
  - New algorithms that attain these lower bounds
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra
- Related work

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul

13

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent $\geq$ #words_moved / largest_message_size

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

14

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent (per processor) = Ω(#flops (per processor) / $M^{3/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)

SIAM SIAG/Linear Algebra Prize, 2012
Ballard, D., Holtz, Schwartz

15

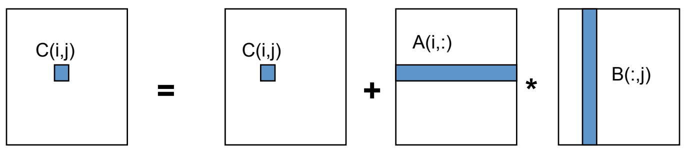
Can we attain these lower bounds?

- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Often not
- If not, are there other algorithms that do?
  - Yes, for much of dense linear algebra
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations
- Only a few sparse algorithms so far
- Lots of work in progress / possible projects
- Case study: Matrix Multiply

16

Naïve Matrix Multiply

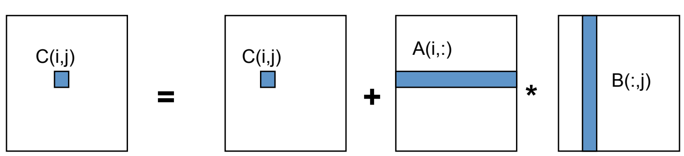
```
{implements C = C + A*B}
for i = 1 to n

  for j = 1 to n


     for k = 1 to n
        C(i,j) = C(i,j) + A(i,k) * B(k,j)
```



17

Naïve Matrix Multiply

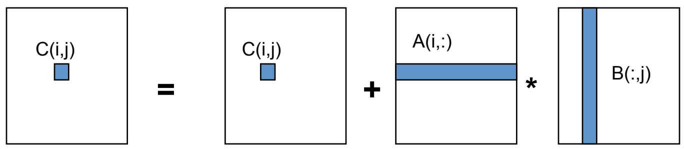
```
{implements C = C + A*B}
for i = 1 to n
 {read row i of A into fast memory}
  for j = 1 to n
     {read C(i,j) into fast memory}
     {read column j of B into fast memory}
     for k = 1 to n
        C(i,j) = C(i,j) + A(i,k) * B(k,j)
     {write C(i,j) back to slow memory}
```



18

Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
 {read row i of A into fast memory}          … n² reads altogether
  for j = 1 to n
    {read C(i,j) into fast memory}           … n² reads altogether
    {read column j of B into fast memory}    … n³ reads altogether
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    {write C(i,j) back to slow memory}       … n² writes altogether
```



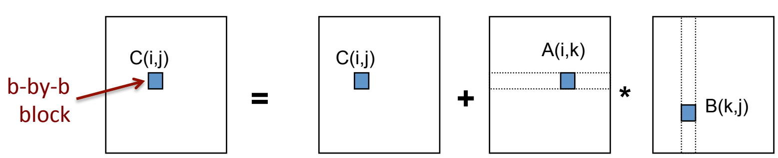
$n^3 + 3n^2$ reads/writes altogether – dominates $2n^3$ arithmetic

19

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where
b is called the block size; assume 3 b-by-b blocks fit in fast memory

```
for i = 1 to n/b
  for j = 1 to n/b
    {read block C(i,j) into fast memory}
    for k = 1 to n/b
        {read block A(i,k) into fast memory}
        {read block B(k,j) into fast memory}
         C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
     {write block C(i,j) back to slow memory}
```

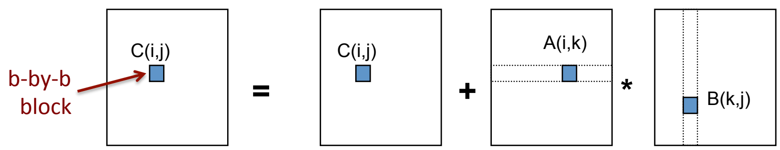


20

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where
b is called the block size; assume 3 b-by-b blocks fit in fast memory

```
for i = 1 to n/b
  for j = 1 to n/b
    {read block C(i,j) into fast memory}        ... b² × (n/b)² = n² reads
    for k = 1 to n/b
      {read block A(i,k) into fast memory}   ... b² × (n/b)³ = n³/b reads
      {read block B(k,j) into fast memory}   ... b² × (n/b)³ = n³/b reads
      C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory}  ... b² × (n/b)² = n² writes
```



$2n^3/b + 2n^2$ reads/writes << $2n^3$ arithmetic - Faster!

21

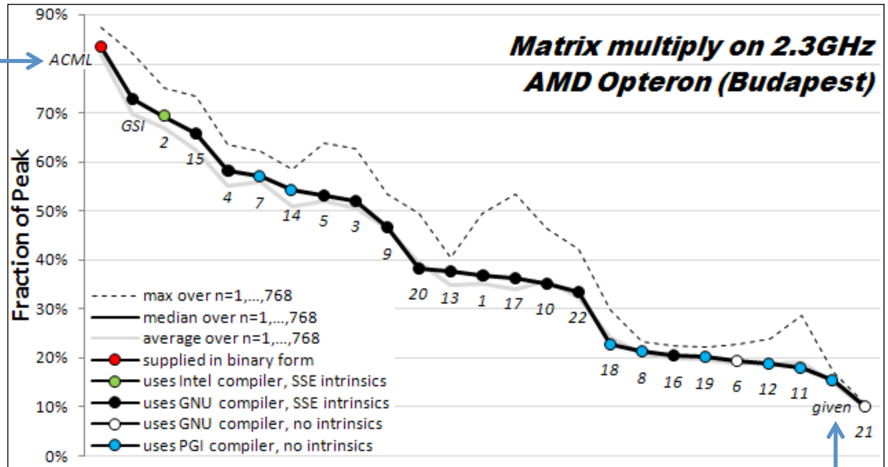
Does blocked matmul attain lower bound?

- Recall: if 3 b-by-b blocks fit in fast memory of size M, then #reads/writes = $2n^3/b + 2n^2$
- Make b as large as possible: $3b^2 \leq M$, so #reads/writes $\geq 2n^3/(M/3)^{1/2} + 2n^2$
- Attains lower bound = Ω (#flops / $M^{1/2}$)

- But what if we don't know M?
- Or if there are multiple levels of fast memory?
- How do we write the algorithm?

22

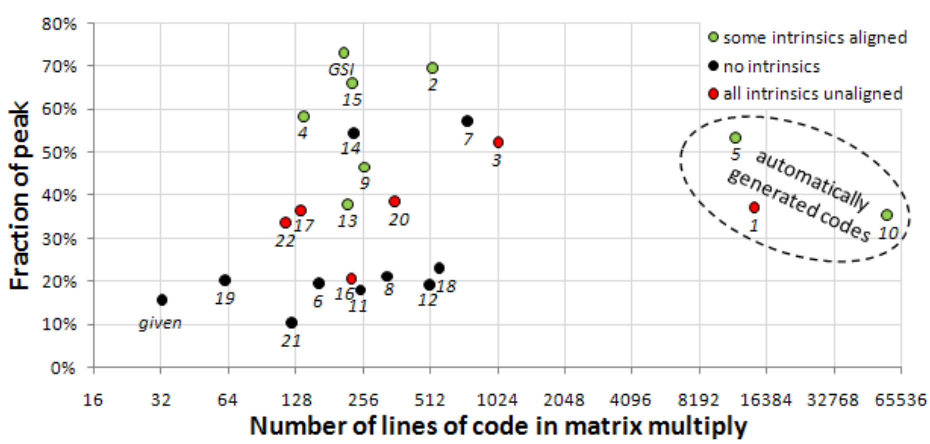
How hard is hand-tuning matmul, anyway?



- Results of 22 student teams trying to tune matrix-multiply, in CS267 Spr09
- Students given "blocked" code to start with (7x faster than naïve)
  - Still hard to get close to vendor tuned performance (ACML) (another 6x)
- For more discussion, see www.cs.berkeley.edu/~volkov/cs267.sp09/hw1/results/

23

How hard is hand-tuning matmul, anyway?



24

Recursive Matrix Multiplication (RMM) (1/2)

- For simplicity: square matrices with $n = 2^m$
- $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

  $= \begin{pmatrix} A_{11}\cdot B_{11} + A_{12}\cdot B_{21} & A_{11}\cdot B_{12} + A_{12}\cdot B_{22} \\ A_{21}\cdot B_{11} + A_{22}\cdot B_{21} & A_{21}\cdot B_{12} + A_{22}\cdot B_{22} \end{pmatrix}$
- True when each A_{ij} etc 1x1 or n/2 x n/2

```
func C = RMM (A, B, n)
   if n = 1, C = A * B, else
     {  C11 = RMM (A11 , B11 , n/2) + RMM (A12 , B21 , n/2)
        C12 = RMM (A11 , B12 , n/2) + RMM (A12 , B22 , n/2)
        C21 = RMM (A21 , B11 , n/2) + RMM (A22 , B21 , n/2)
        C22 = RMM (A21 , B12 , n/2) + RMM (A22 , B22 , n/2)  }
   return
```

25

Recursive Matrix Multiplication (RMM) (2/2)

```
func C = RMM (A, B, n)
   if n=1, C = A * B, else
     {  C11 = RMM (A11 , B11 , n/2) + RMM (A12 , B21 , n/2)
        C12 = RMM (A11 , B12 , n/2) + RMM (A12 , B22 , n/2)
        C21 = RMM (A21 , B11 , n/2) + RMM (A22 , B21 , n/2)
        C22 = RMM (A21 , B12 , n/2) + RMM (A22 , B22 , n/2)  }
   return
```

A(n) = # arithmetic operations in RMM(. , . , n)
$= 8 \cdot A(n/2) + 4(n/2)^2$ if $n > 1$, else 1
$= 2n^3$ … same operations as usual, in different order

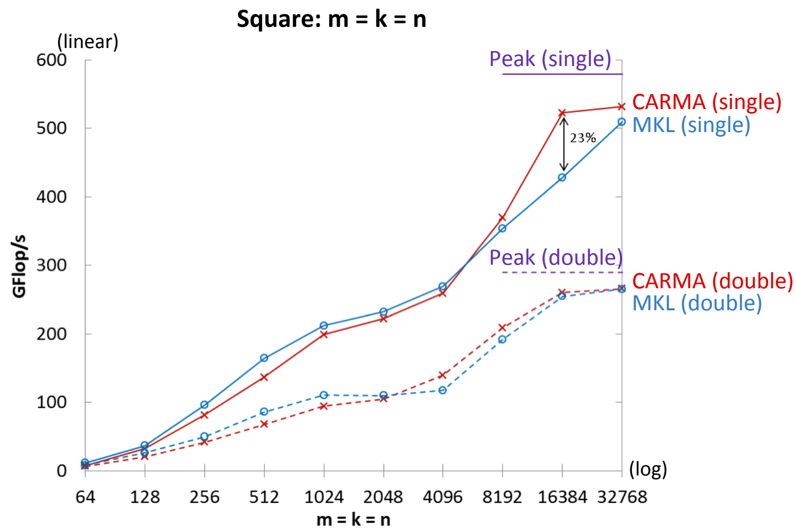
W(n) = # words moved between fast, slow memory by RMM(. , . , n)
$= 8 \cdot W(n/2) + 12(n/2)^2$ if $3n^2 > M$, else $3n^2$
$= O(n^3 / M^{1/2} + n^2)$ … same as blocked matmul

“Cache oblivious”, works for memory hierarchies, but not panacea

26

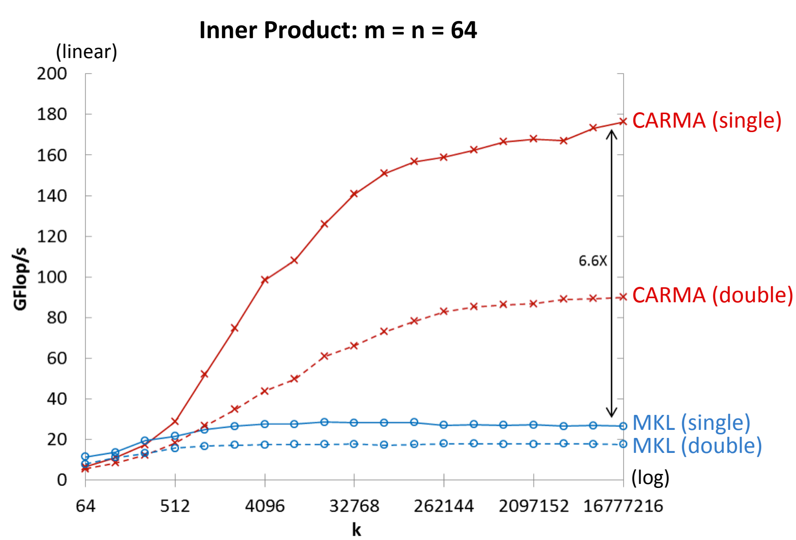
CARMA Performance: Shared Memory

Intel Emerald: 4 Intel Xeon X7560 x 8 cores, 4 x NUMA

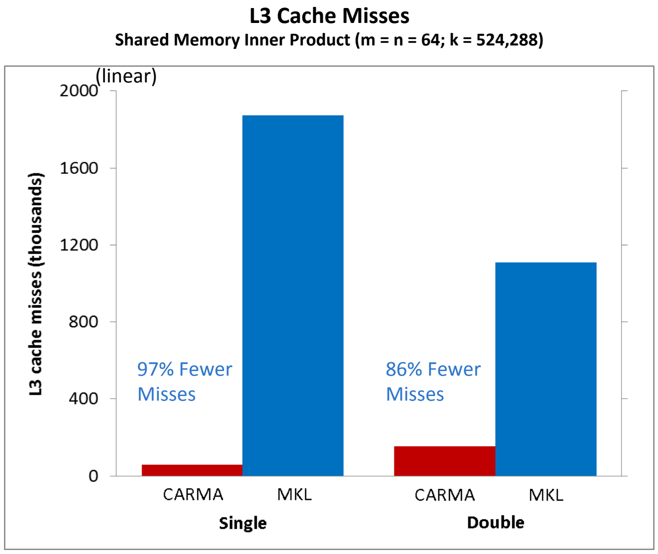


CARMA Performance: Shared Memory

Intel Emerald: 4 Intel Xeon X7560 x 8 cores, 4 x NUMA



Why is CARMA Faster?



Parallel MatMul with 2D Processor Layout

- P processors in $P^{1/2} \times P^{1/2}$ grid
  - Processors communicate along rows, columns
- Each processor owns $n/P^{1/2} \times n/P^{1/2}$ submatrices of A,B,C
- Example: P=16, processors numbered from P_{00} to P_{33}
  - Processor P_{ij} owns submatrices A_{ij}, B_{ij} and C_{ij}

C = A * B

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

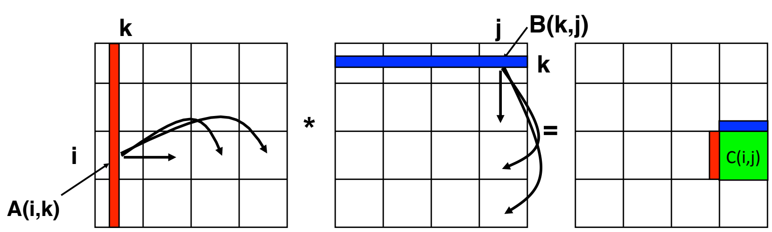
P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

SUMMA Algorithm

- SUMMA = Scalable Universal Matrix Multiply
  - Attains lower bounds:
    - Assume fast memory size M = O(n^2/P) per processor – 1 copy of data
    - #words_moved = Ω(#flops / $M^{1/2}$) = Ω((n^3/P) / (n^2/P)$^{1/2}$) = Ω(n^2 / $P^{1/2}$)
    - #messages = Ω(#flops / $M^{3/2}$) = Ω((n^3/P) / (n^2/P)$^{3/2}$) = Ω($P^{1/2}$)
  - Can accommodate any processor grid, matrix dimensions & layout
  - Used in practice in PBLAS = Parallel BLAS
    - www.netlib.org/lapack/lawns/lawn{96,100}.ps

31

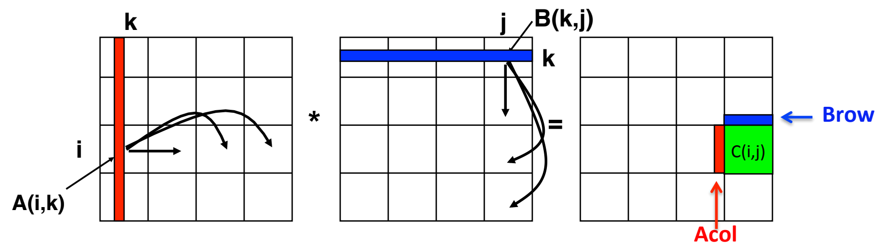
SUMMA – n x n matmul on $P^{1/2}$ x $P^{1/2}$ grid



- C(i, j) is n/$P^{1/2}$ x n/$P^{1/2}$ submatrix of C on processor P_{ij}
- A(i,k) is n/$P^{1/2}$ x b submatrix of A
- B(k,j) is b x n/$P^{1/2}$ submatrix of B
- C(i,j) = C(i,j) + Σ_k A(i,k)*B(k,j)
  - summation over submatrices
- Need not be square processor grid

32

SUMMA– n x n matmul on $P^{1/2}$ x $P^{1/2}$ grid



For k=0 to n/b-1
  for all i = 1 to $P^{1/2}$
    owner of A(i,k) broadcasts it to whole processor row (using binary tree)
  for all j = 1 to $P^{1/2}$
    owner of B(k,j) broadcasts it to whole processor column (using bin. tree)
  Receive A(i,k) into Acol
  Receive B(k,j) into Brow
  C_myproc = C_myproc + Acol * Brow

- Attains bandwidth lower bound
- Attains latency lower bound if b near maximum $n/P^{1/2}$

33

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 $= \Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$
- Does ScaLAPACK attain these bounds?
  - For #words_moved: mostly, except nonsym. Eigenproblem
  - For #messages: asymptotically worse, except Cholesky
- New algorithms attain all bounds, up to polylog(P) factors
  - Cholesky, LU, QR, Sym. and Nonsym eigenproblems, SVD
    - Needed to replace partial pivoting in LU
    - Need randomization for Nonsym eigenproblem (so far)

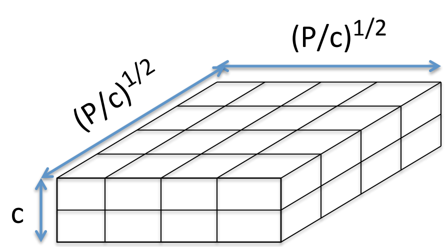
Can we do Better?

Can we do better?

- Aren't we already optimal?
- Why assume $M = O(n^2/P)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?
  - Special case: "3D Matmul": uses $M = O(n^2/P^{2/3})$
    - Dekel, Nassimi, Sahni [81], Bernsten [89], Agarwal, Chandra, Snir [90], Johnson [93], Agarwal, Balle, Gustavson, Joshi, Palkar [95]
    - Processors arranged in $P^{1/3} \times P^{1/3} \times P^{1/3}$ grid
  - $M = O(n^2/P^{2/3})$ is $P^{1/3}$ times the minimum
    - Not always that much memory available…

2.5D Matrix Multiplication

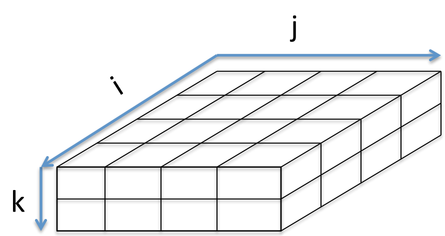
- Assume can fit cn^2/P data per processor, $c>1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

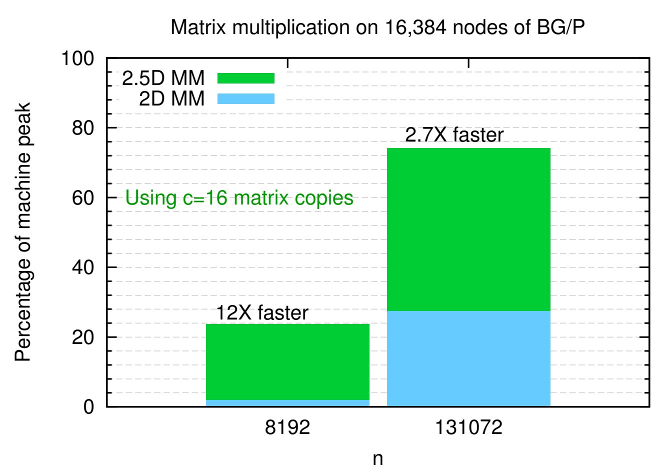
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



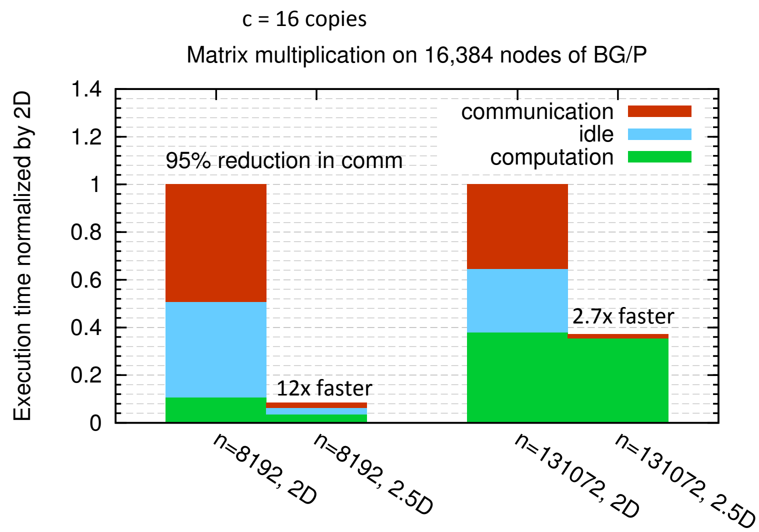
Initially P(i,j,0) owns A(i,j) and B(i,j) each of size $n(c/P)^{1/2}$ x $n(c/P)^{1/2}$

(1) P(i,j,0) broadcasts A(i,j) and B(i,j) to P(i,j,k)
(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of Σ_m A(i,m)*B(m,j)
(3) Sum-reduce partial sums Σ_m A(i,m)*B(m,j) along k-axis so P(i,j,0) owns C(i,j)

2.5D Matmul on BG/P, 16K nodes / 64K cores



2.5D Matmul on BG/P, 16K nodes / 64K cores



Distinguished Paper Award, EuroPar'11
SC'11 paper by Solomonik, Bhatele, D.

Perfect Strong Scaling – in Time and Energy (1/2)

- Every time you add a processor, you should use its memory M too
- Start with minimal number of procs: $PM = 3n^2$
- Increase P by a factor of c ➔ total memory increases by a factor of c
- Notation for timing model:
  - γ_T, β_T, α_T = secs per flop, per word_moved, per message of size m
- $T(cP) = n^3/(cP) [\gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})]$
  $= T(P)/c$
- Notation for energy model:
  - γ_E, β_E, α_E = joules for same operations
  - δ_E = joules per word of memory used per sec
  - ε_E = joules per sec for leakage, etc.
- $E(cP) = cP \{ n^3/(cP) [\gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})] + \delta_E M T(cP) + \varepsilon_E T(cP) \}$
  $= E(P)$
- Limit: $c \le P^{1/3}$ (3D algorithm), if starting with 1 copy of inputs

Perfect Strong Scaling – in Time and Energy (2/2)

- Perfect scaling extends to N-body, Strassen, …
- Can prove lower bounds on network (eg 3D torus for matmul)
- We can use these models to answer many questions, including:

- What is the minimum energy required for a computation?
- Given a maximum allowed runtime **T** , what is the minimum energy **E** needed to achieve it?
- Given a maximum energy budget **E** , what is the minimum runtime **T** that we can attain?
- The ratio **P** = **E/T** gives us the average power required to run the algorithm. Can we minimize the average power consumed?
- Given an algorithm, problem size, number of processors and target energy efficiency (GFLOPS/W), can we determine a set of architectural parameters to describe a conforming computer architecture?
- See Andrew Gearhart's PhD thesis

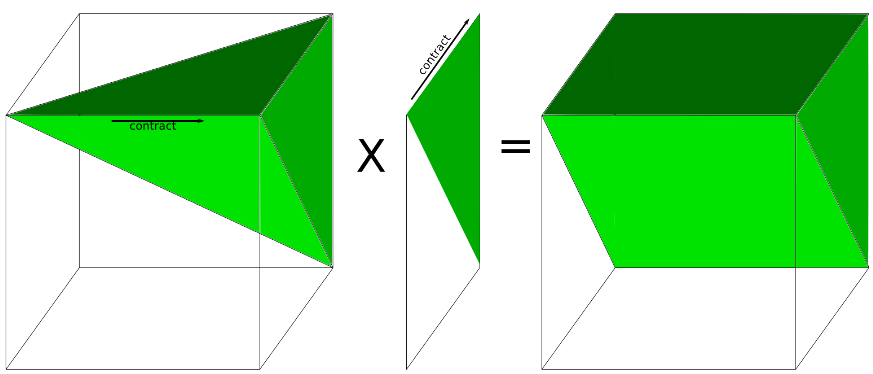
Handling Heterogeneity

- Suppose each of P processors could differ
  - γ_i = sec/flop, β_i = sec/word, α_i = sec/message, M_i = memory
- What is optimal assignment of work F_i to minimize time?
  - $T_i = F_i \gamma_i + F_i \beta_i / M_i^{1/2} + F_i \alpha_i / M_i^{3/2} = F_i [\gamma_i + \beta_i / M_i^{1/2} + \alpha_i / M_i^{3/2}] = F_i \xi_i$
  - Choose F_i so $\Sigma_i F_i = n^3$ and minimizing $T = \max_i T_i$
  - Answer: $F_i = n^3 (1/\xi_i)/\Sigma_j (1/\xi_j)$ and $T = n^3/\Sigma_j (1/\xi_j)$
- Optimal Algorithm for nxn matmul
  - Recursively divide into 8 half-sized subproblems
  - Assign subproblems to processor i to add up to F_i flops
- Works for Strassen, other algorithms…

Application to Tensor Contractions

- Ex: $C(i,j,k) = \Sigma_{mn} A(i,j,m,n)*B(m,n,k)$
  - Communication lower bounds apply
- Complex symmetries possible
  - Ex: $B(m,n,k) = B(k,m,n) = \ldots$
  - d-fold symmetry can save up to d!-fold flops/memory

$C(i,j,k) = \Sigma_m A(i,j,m)*B(m,k)$



A
3-fold symm

B
2-fold symm

C
2-fold symm

Application to Tensor Contractions

- Ex: $C(i,j,k) = \Sigma_{mn} A(i,j,m,n)*B(m,n,k)$
  - Communication lower bounds apply
- Complex symmetries possible
  - Ex: $B(m,n,k) = B(k,m,n) = \ldots$
  - d-fold symmetry can save up to d!-fold flops/memory
- Heavily used in electronic structure calculations
  - Ex: NWChem, for coupled cluster (CC) approach to Schroedinger eqn.
- CTF: Cyclops Tensor Framework
  - Exploits 2.5D algorithms, symmetries
  - Up to **3x faster** running CC than NWChem on 3072 cores of Cray XE6
  - Solomonik, Hammond, Matthews

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix}$$

$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\ R_{01} \\ \hline Q_{11}\ R_{11} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02}\ R_{02} \end{pmatrix}$$

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix} = \begin{pmatrix} Q_{00} R_{00} \\ \hline Q_{10} R_{10} \\ \hline Q_{20} R_{20} \\ \hline Q_{30} R_{30} \end{pmatrix} = \begin{pmatrix} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{pmatrix} \cdot \begin{pmatrix} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{pmatrix}$$

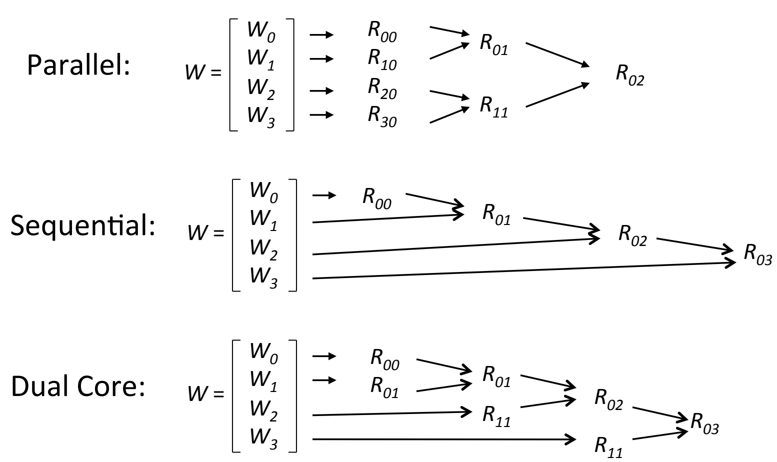
$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01} R_{01} \\ \hline Q_{11} R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02} R_{02} \end{pmatrix}$$

Output = $\{ Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} \}$

47

TSQR: An Architecture-Dependent Algorithm



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

Can choose reduction tree dynamically

TSQR Performance Results

- Parallel Speedups
  - Up to **8x** on 8 core Intel Clovertown
  - Up to **6.7x** on 16 processor Pentium cluster
  - Up to **4x** on 32 processor IBM Blue Gene
  - Up to **13x** on NVidia GPU
  - Up to **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - **Only 1.6x slower** on Cloud than just accessing data twice (Gleich and Benson)
- Sequential Speedup
  - "**Infinite**" for out-of-core on PowerPC laptop
- SVD costs about the same
- Joint work with Grigori, Hoemmen, Langou, Anderson, Ballard, Keutzer, others

49

Using similar idea for TSLU as TSQR: Use reduction tree, to do "Tournament Pivoting"

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

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'

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

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

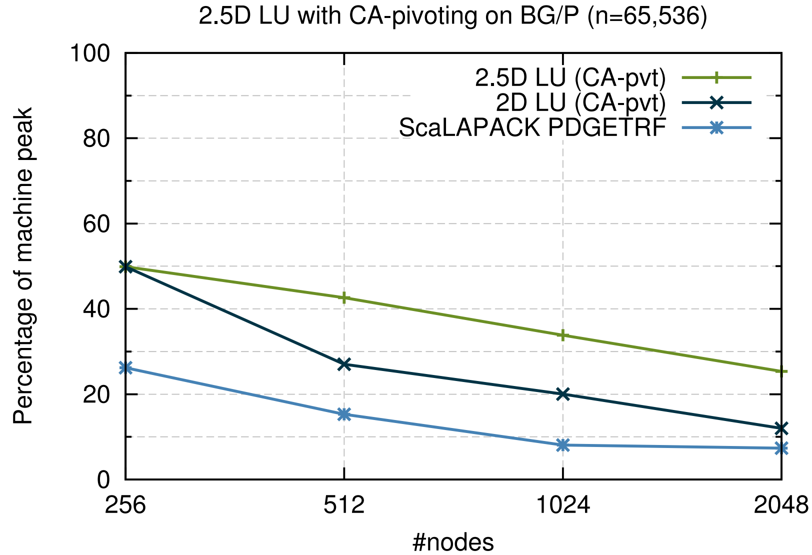
$$\begin{pmatrix} W_{12}' \\ W_{34}' \end{pmatrix} = P_{1234} \cdot L_{1234} \cdot U_{1234}$$

Choose b pivot rows

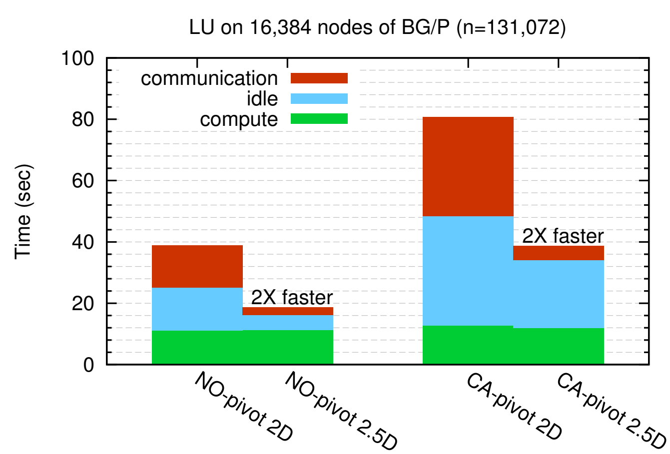
- Go back to W and use these b pivot rows
  - Move them to top, do LU without pivoting
  - Extra work, but lower order term
- Thm: As numerically stable as Partial Pivoting on a larger matrix

50

LU Speedups from Tournament Pivoting and 2.5D



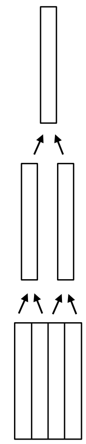
2.5D vs 2D LU With and Without Pivoting



Thm: Perfect Strong Scaling impossible, because Latency*Bandwidth = $\Omega(n^2)$

Other CA algorithms

- Need for pivoting arises beyond LU, in QR
  - Choose permutation P so that leading columns of A*P = Q*R span column space of A – Rank Revealing QR (RRQR)
  - Usual approach like Partial Pivoting
    - Put longest column first, update rest of matrix, repeat
    - Hard to do using BLAS3 at all, let alone hit lower bound
  - Use Tournament Pivoting
    - Each round of tournament selects best b columns from two groups of b columns, either using usual approach or something better (Gu/Eisenstat)
    - Thm: This approach ``reveals the rank'' of A in the sense that the leading rxr submatrix of R has singular values "near" the largest r singular values of A; ditto for trailing submatrix
  - Idea extends to other pivoting schemes
    - Cholesky with diagonal pivoting
    - LU with complete pivoting
    - LDL^T with complete pivoting



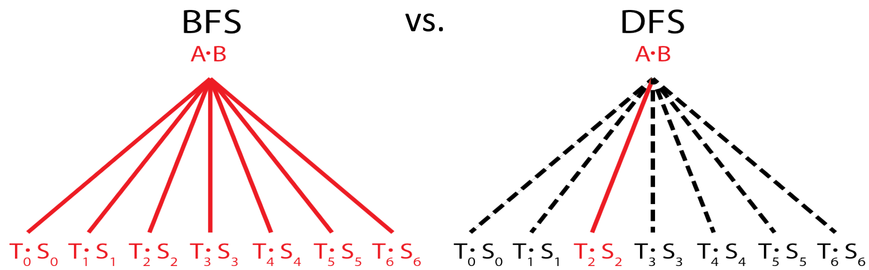
53

Communication Lower Bounds for Strassen-like matmul algorithms

Classical $O(n^3)$ matmul: #words_moved = $\Omega(M(n/M^{1/2})^3/P)$	Strassen's $O(n^{\lg 7})$ matmul: #words_moved = $\Omega(M(n/M^{1/2})^{\lg 7}/P)$	Strassen-like $O(n^{\omega})$ matmul: #words_moved = $\Omega(M(n/M^{1/2})^{\omega}/P)$

- Proof: graph expansion (different from classical matmul)
  - Strassen-like: DAG must be "regular" and connected
- Extends up to $M = n^2 / p^{2/\omega}$
- Best Paper Prize (SPAA'11), Ballard, D., Holtz, Schwartz appeared in JACM
- Is the lower bound attainable?

Communication Avoiding Parallel Strassen *(CAPS)*



Runs all 7 multiplies in parallel
Each on P/7 processors
Needs 7/4 as much memory

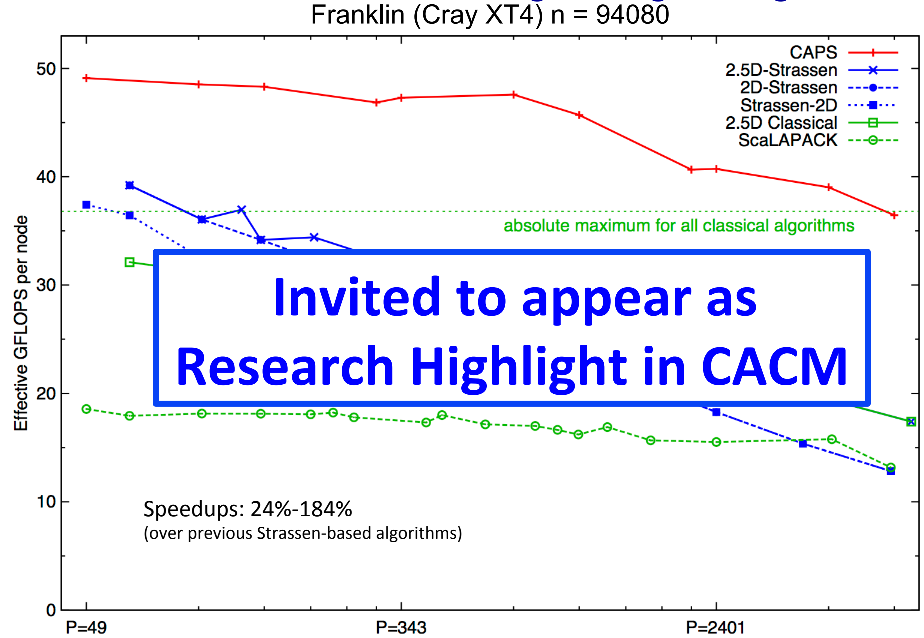
Runs all 7 multiplies sequentially
Each on all P processors
Needs 1/4 as much memory

```
CAPS
  If EnoughMemory and P ≥ 7
    then BFS step
    else DFS step
  end if
```

In practice, how to best interleave BFS and DFS is a "tuning parameter"

Performance Benchmarking, Strong Scaling Plot

Franklin (Cray XT4) n = 94080



Invited to appear as Research Highlight in CACM

What about fast algorithms for the rest of linear algebra?

- "Fast Matrix Multiplication is Stable"
  - JD, I. Dumitriu, O. Holtz, R. Kleinberg (2007)
- "Fast Linear Algebra is Stable"
  - JD, I. Dumitriu, O. Holtz (2007)
- "Sequential Communication Bounds for Fast Linear Algebra"
  - G. Ballard, JD, O. Holtz, O. Schwartz (EECS-2012-36)
- Parallel communication bounds?
- Implementations?

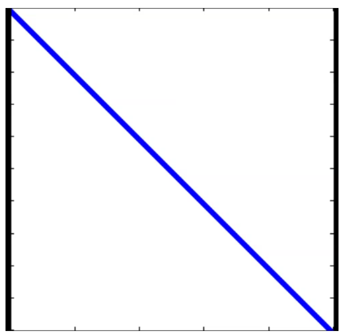
Symmetric Band Reduction

- Grey Ballard and Nick Knight
- $A \Rightarrow QAQ^T = T$, where
  - $A=A^T$ is banded
  - T tridiagonal
  - Similar idea for SVD of a band matrix
- Use alone, or as second phase when A is dense:
  - Dense $\Rightarrow$ Banded $\Rightarrow$ Tridiagonal
- Implemented in LAPACK's sytrd
- Algorithm does not satisfy communication lower bound theorem for applying orthogonal transformations
  - It can communicate even less!

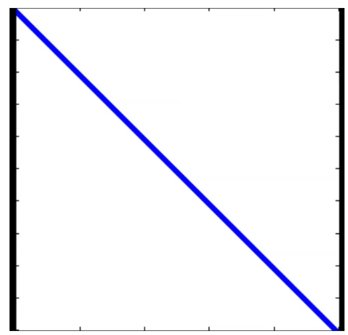
Conventional vs CA - SBR

Touch all data 4 times

Touch all data once



Conventional



Communication-Avoiding

Many tuning parameters
Right choices reduce #words_moved by factor M/bw, not just $M^{1/2}$

Speedups of Sym. Band Reduction vs LAPACK's DSBTRD

- Up to **17x** on Intel Gainestown, vs MKL 10.0
  - n=12000, b=500, 8 threads
- Up to **12x** on Intel Westmere, vs MKL 10.3
  - n=12000, b=200, 10 threads
- Up to **25x** on AMD Budapest, vs ACML 4.4
  - n=9000, b=500, 4 threads
- Up to **30x** on AMD Magny-Cours, vs ACML 4.4
  - n=12000, b=500, 6 threads

- Neither MKL nor ACML benefits from multithreading in DSBTRD
  - Best sequential speedup vs MKL: **1.9x**
  - Best sequential speedup vs ACML: **8.5x**

What about sparse matrices? (1/3)

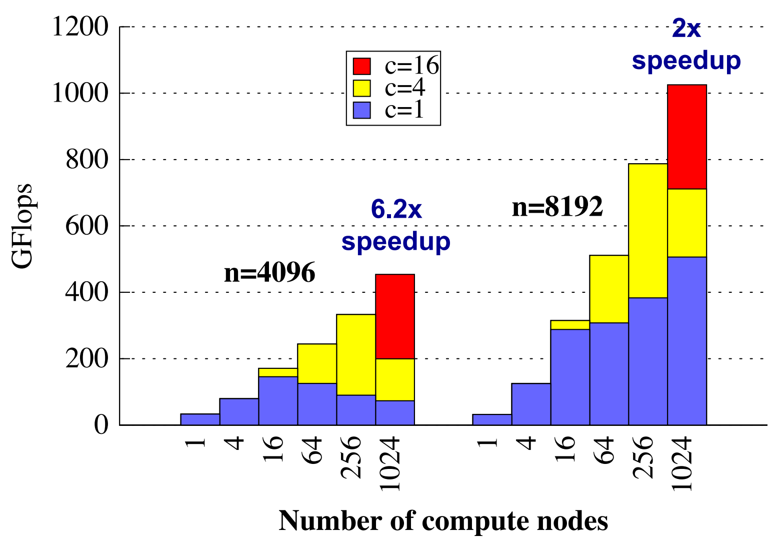
- If matrix quickly becomes dense, use dense algorithm
- Ex: All Pairs Shortest Path using Floyd-Warshall
- Similar to matmul: Let D = A, then
  - for k = 1:n, for i = 1:n, for j=1:n
    - D(i,j) = min(D(i,j), D(i,k) + D(k,j))
- But can't reorder outer loop for 2.5D, need another idea
- Abbreviate D(i,j) = min(D(i,j), $\min_k$(A(i,k)+B(k,j)) by D = A⊙B
  - Dependencies ok, 2.5D works, just different semiring
- Kleene's Algorithm:

D = **DC-APSP**(A,n)
  D = A, Partition D = [[D11,D12];[D21,D22]] into n/2 x n/2 blocks
  D11 = **DC-APSP**(D11,n/2),
  D12 = D11 ⊙ D12, D21 = D21 ⊙ D11, D22 = D21 ⊙ D12,
  D22 = **DC-APSP**(D22,n/2),
  D21 = D22 ⊙ D21, D12 = D12 ⊙ D22, D11 = D12 ⊙ D21,

61

Performance of 2.5D APSP using Kleene

Strong Scaling on Hopper (Cray XE6 with 1024 nodes = 24576 cores)



62

What about sparse matrices? (2/3)

- If parts of matrix becomes dense, optimize those
- Ex: Cholesky on matrix A with good separators
- Thm (Lipton,Rose,Tarjan,'79) If all balanced separators of G(A) have at least w vertices, then G(chol(A)) has clique of size w
  - Need to do dense Cholesky on w x w submatrix
- Thm: #Words_moved = $\Omega(w^3/M^{1/2})$ etc
- Thm (George,'73) Nested dissection gives optimal ordering for 2D grid, 3D grid, similar matrices
  - w = n for 2D n x n grid, w = n^2 for 3D n x n x n grid
- Sequential multifrontal Cholesky attains bounds
- PSPACES (Gupta, Karypis, Kumar) is a parallel sparse multifrontal Cholesky package
  - Attains 2D and 2.5D lower bounds (using optimal dense Cholesky on separators)

63

What about sparse matrices? (3/3)

- If matrix stays very sparse, lower bound unattainable, new one?
- Ex: A*B, both diagonal: no communication in parallel case
- Ex: A*B, both are Erdos-Renyi: Prob(A(i,j)≠0) = d/n, d << $n^{1/2}$,iid
- Assumption: Algorithm is *sparsity-independent*: assignment of data and work to processors is sparsity-pattern-independent (but zero entries need not be communicated or operated on)
- Thm: A parallel algorithm that is sparsity-independent and load balanced for Erdos-Renyi matmul satisfies (in expectation) #Words_moved = $\Omega(\min(dn/P^{1/2} , d^2n/P))$
  - Proof exploits fact that reuse of entries of C = A*B unlikely
- Contrast general lower bound: #Words_moved = $\Omega(d^2n/(P\mathbf{M}^{1/2}))$
- Attained by divide-and-conquer algorithm that splits matrices along dimensions most likely to minimize cost
- Recent result (P. Koanantakool et al, IPDPS'16): Dense*Sparse

64

Summary of Direct Linear Algebra (1/2)

- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of ongoing work / possible projects on
  - Algorithms:
    - LDL^T, QR with pivoting, other pivoting schemes, low rank factorizations, eigenproblems…
    - All-pairs-shortest-path, …
    - Both 2D (c=1) and 2.5D (c>1)
    - But only bandwidth may decrease with c>1, not latency
    - Sparse matrices
  - Platforms:
    - Multicore, cluster, GPU, cloud, heterogeneous, low-energy, …
  - Software:
    - Integration into Sca/LAPACK, PLASMA, MAGMA,…
- Integration of CTF into quantum chemistry/DFT applications
  - Aquarius, with ANL, UT Austin on IBM BG/Q, Cray XC30
  - Qbox, with LLNL, IBM, on IBM BG/Q
  - Q-Chem, work in progress
- Integration into big data analysis system based on Spark at AMPLab

Summary of Direct Linear Algebra (2/2)

- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of ongoing work / possible projects on
  - Algorithms:
    - LDL^T, QR with pivoting, other pivoting schemes, low rank factorizations, eigenproblems …
    - Compare fast QR (
    - All-pairs-shortest-path, …
    - Both 2D (c=1) and 2.5D (c>1)
    - But only bandwidth may decrease with c>1, not latency
    - Sparse matrices
  - Platforms:
    - Multicore, cluster, GPU, cloud, heterogeneous, low-energy, …
  - Software:
    - Integration into Sca/LAPACK, PLASMA, MAGMA,…

Outline

- "Direct" Linear Algebra
  - Lower bounds on communication
  - New algorithms that attain these lower bounds
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra
- Related Work

Recall optimal sequential Matmul

- Naïve code
  for i=1:n, for j=1:n, for k=1:n,
  C(i,j)+=A(i,k)*B(k,j)

- "Blocked" code
  for i = 1:n/b, for j = 1:n/b, for k = 1:n/b
  C[i,j]+=A[i,k]*B[k,j] … b x b matmul

- Thm: Picking b = $M^{1/2}$ attains lower bound:
  #words_moved = $\Omega(n^3/M^{1/2})$
- Where does **1/2** come from?

New Thm applied to Matmul

- for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
- Record array indices in matrix Δ

$$\Delta = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{ccc} i & j & k \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ A \\ B \\ C \end{array}$$

- Solve LP for x = [xi,xj,xk]^T: max $\mathbf{1}^T$x s.t. Δ x ≤ **1**
  - Result: x = [1/2, 1/2, 1/2]^T, $\mathbf{1}^T$x = 3/2 = e
- Thm: #words_moved = $\Omega(n^3/M^{e-1}) = \Omega(n^3/M^{1/2})$

  Attained by block sizes $M^{xi}, M^{xj}, M^{xk} = M^{1/2}, M^{1/2}, M^{1/2}$

New Thm applied to Direct N-Body

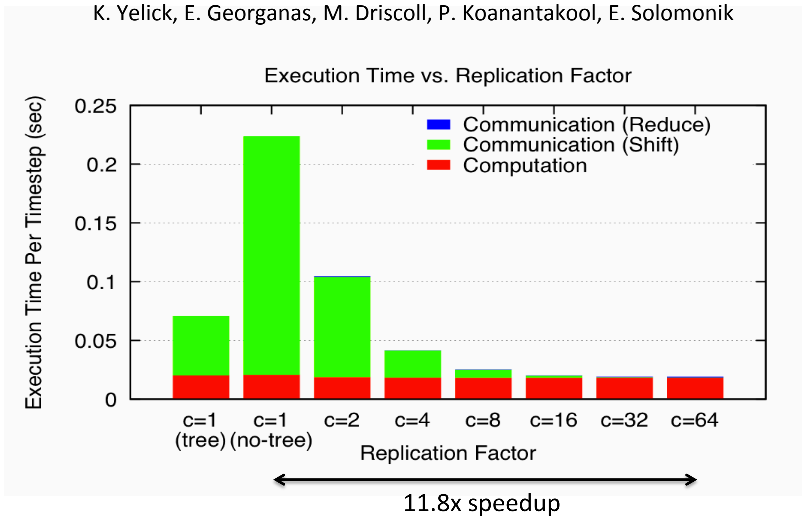
- for i=1:n, for j=1:n, F(i) += force(P(i) , P(j))
- Record array indices in matrix Δ

$$\Delta = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cc} i & j \end{array} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \begin{array}{c} \\ F \\ P(i) \\ P(j) \end{array}$$

- Solve LP for x = [xi,xj]^T: max $\mathbf{1}^T$x s.t. Δ x ≤ **1**
  - Result: x = [1,1], $\mathbf{1}^T$x = 2 = e
- Thm: #words_moved = $\Omega(n^2/M^{e-1}) = \Omega(n^2/M^{1})$

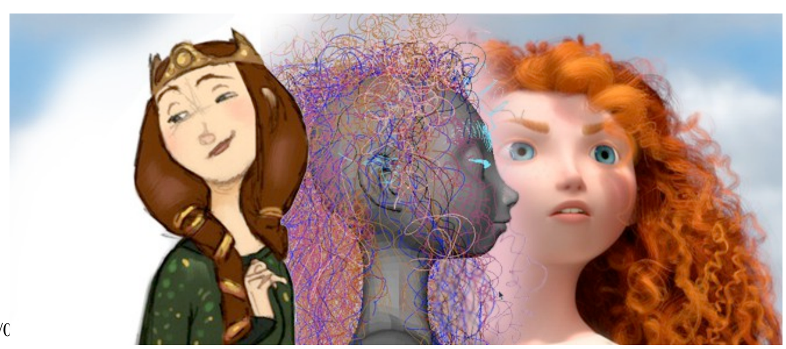
  Attained by block sizes $M^{xi}, M^{xj} = M^1, M^1$

N-Body Speedups on IBM-BG/P (Intrepid) 8K cores, 32K particles



Some Applications

- Gravity, Turbulence, Molecular Dynamics, Plasma Simulation, …
- Electron-Beam Lithography Device Simulation
- Hair ...
  - www.fxguide.com/featured/brave-new-hair/
  - graphics.pixar.com/library/CurlyHairA/paper.pdf



04/0 72

New Thm applied to Random Code

- for i1=1:n, for i2=1:n, … , for i6=1:n
  A1(i1,i3,i6) += func1(A2(i1,i2,i4),A3(i2,i3,i5),A4(i3,i4,i6))
  A5(i2,i6) += func2(A6(i1,i4,i5),A3(i3,i4,i6))
- Record array indices in matrix Δ

$$
\Delta = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array}
\begin{array}{cccccc}
i1 & i2 & i3 & i4 & i5 & i6 \\
\end{array}
$$

$$
\Delta = \left(\begin{array}{cccccc}
1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0
\end{array}\right)
\begin{array}{l}
A1 \\ A2 \\ A3 \\ A3,A4 \\ A5 \\ A6
\end{array}
$$

- Solve LP for $x = [x1,\ldots,x6]^T$: max $\mathbf{1}^T x$ s.t. $\Delta x \le \mathbf{1}$
  – Result: $x = [2/7,3/7,1/7,2/7,3/7,4/7]$, $\mathbf{1}^T x = 15/7 = e$
- Thm: #words_moved = $\Omega(n^6/M^{e-1}) = \Omega(n^6/M^{8/7})$
  Attained by block sizes $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

Approach to generalizing lower bounds

- Matmul
  for i=1:n, for j=1:n, for k=1:n,
  C(i,j)+=A(i,k)*B(k,j)

=> for (i,j,k) in S = subset of Z^3
Access locations indexed by (i,j), (i,k), (k,j)

- General case
  for i1=1:n, for i2 = i1:m, … for ik = i3:i4
  C(i1+2*i3-i7) = func(A(i2+3*i4,i1,i2,i1+i2,…),B(pnt(3*i4)),…)
  D(something else) = func(something else), …

=> for (i1,i2,…,ik) in S = subset of Z^k
Access locations indexed by "projections", eg
ϕ_C (i1,i2,…,ik) = (i1+2*i3-i7)
ϕ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

- Goal: Communication lower bounds and optimal algorithms for *any* program that looks like this

General Communication Bound

- Thm: Given a program with array refs given by projections ϕ_j, then there is an $e \geq 1$ such that

  #words_moved = Ω (#iterations/M^{e-1})

  where e is the the value of a linear program:

  minimize $e = \Sigma_j e_j$ subject to

  $\text{rank}(H) \leq \Sigma_j e_j * \text{rank}(\phi_j(H))$ for all subgroups $H < Z^k$

- Proof depends on recent result in pure mathematics by Christ/Tao/Carbery/Bennett

Is this bound attainable (1/2)?

- But first: Can we write it down?
  - One inequality per subgroup $H < Z^d$, but still finitely many!
  - Thm (bad news): Writing down all inequalities in LP reduces to Hilbert's 10th problem over Q
    - Could be undecidable: open question
  - Thm (good news): Another LP has same solution, is decidable (but expensive so far)
  - Thm: (better news) Easy to write LP down explicitly in many cases of interest:
    - When at most 3 arrays
    - When at most 4 loop indices
    - When subscripts are subsets of indices

Is this bound attainable (2/2)?

- Depends on loop dependencies
- Best case: none, or reductions (matmul)
- Thm: When all subscripts are subsets of indices, the solution x of the dual LP gives optimal tile sizes: M^{x1}, M^{x2}, …
- Ex: Linear algebra, n-body, "random code," join, …
- Conjecture: always attainable (modulo dependencies): work in progress / class project
- Long term goal: incorporate in compilers

Ongoing Work

- Automate generation of lower bounds
- Extend "perfect scaling" results for time and energy by using extra memory
- Have yet to find a case where we cannot attain lower bound (dependencies permitting)
  – can we prove this?
- Incorporate into compilers

Outline

- "Direct" Linear Algebra
  - Lower bounds on communication
  - New algorithms that attain these lower bounds
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra
- Related work

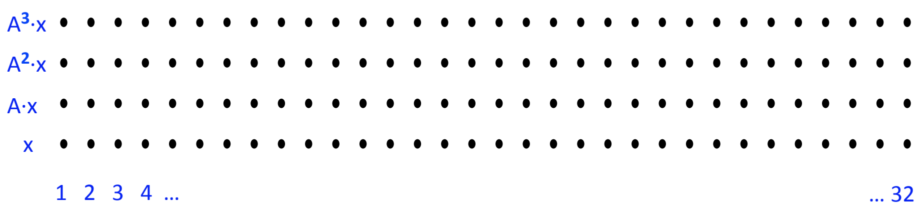
Avoiding Communication in Iterative Linear Algebra

- k-steps of iterative solver for sparse Ax=b or Ax=λx
  - Does k SpMVs with A and starting vector
  - Many such "Krylov Subspace Methods"
- Goal: minimize communication
  - Assume matrix "well-partitioned"
  - Serial implementation
    - Conventional: O(k) moves of data from slow to fast memory
    - **New**: **O(1) moves of data – optimal**
  - Parallel implementation on p processors
    - Conventional: O(k log p) messages (k SpMV calls, dot prods)
    - **New: O(log p) messages - optimal**
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation

80

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

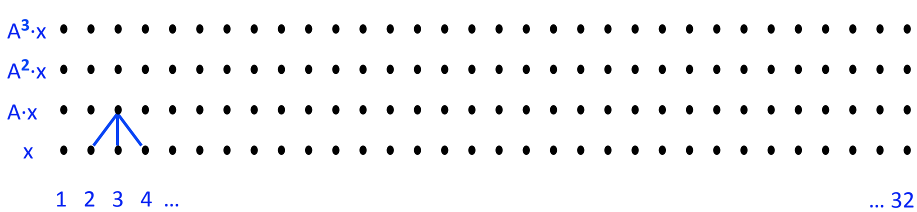
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

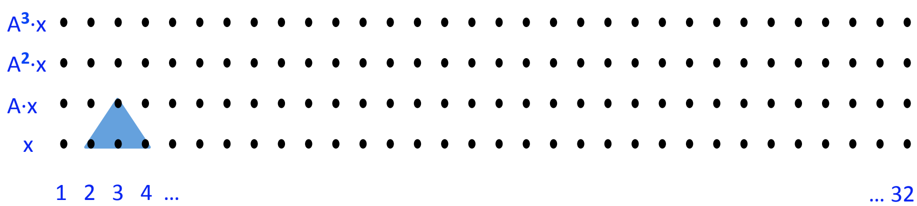
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

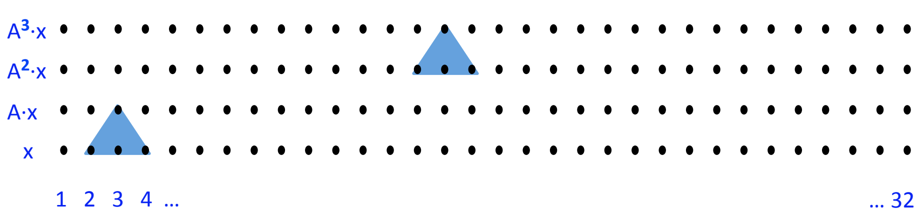
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

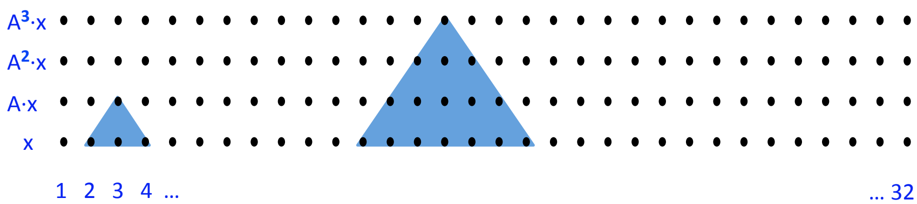
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

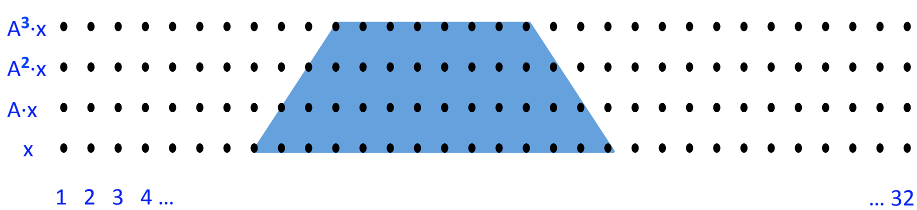
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

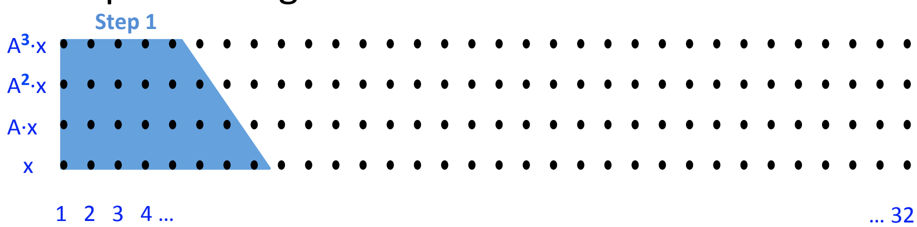
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

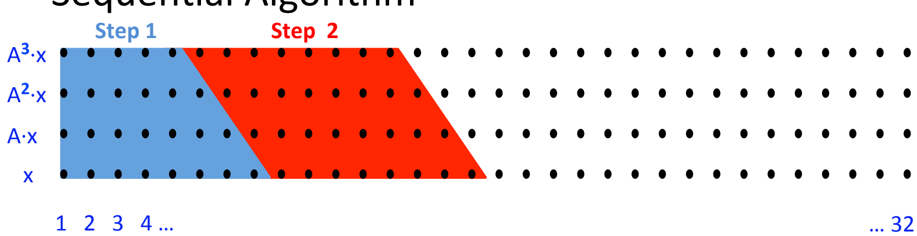
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

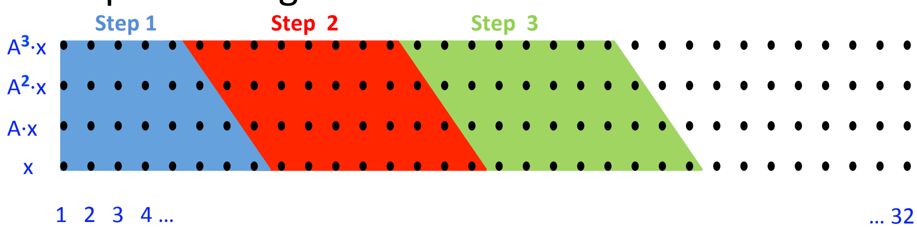
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

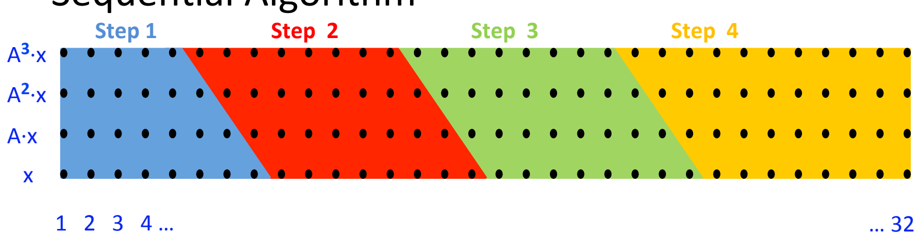
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm

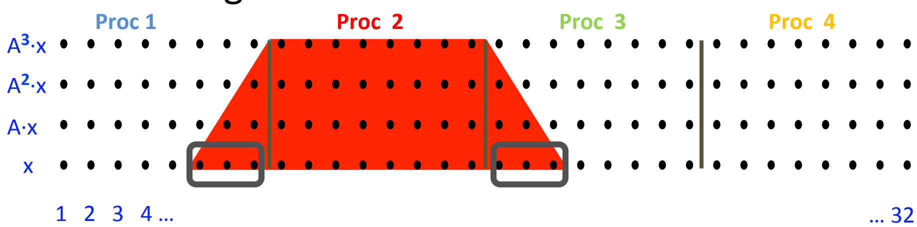


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Parallel Algorithm

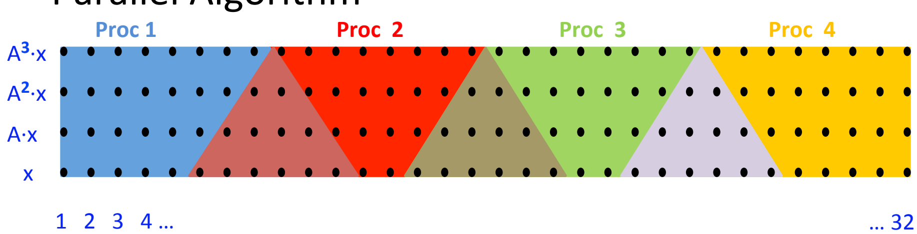


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

Communication Avoiding Kernels:

The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Parallel Algorithm



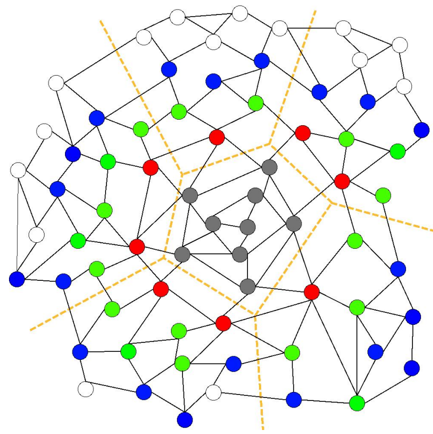
- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

Same idea works for general sparse matrices

Simple block-row partitioning ➔ (hyper)graph partitioning

Left-to-right processing ➔ Traveling Salesman Problem



Minimizing Communication of GMRES to solve Ax=b

- GMRES: find x in span$\{b,Ab,\ldots,A^kb\}$ minimizing $\|\, Ax-b \,\|_2$

Standard GMRES

for i=1 to k
  w = A · v(i-1)   *… SpMV*
  MGS(w, v(0),…,v(i-1))
  update v(i), H
endfor
solve LSQ problem with H

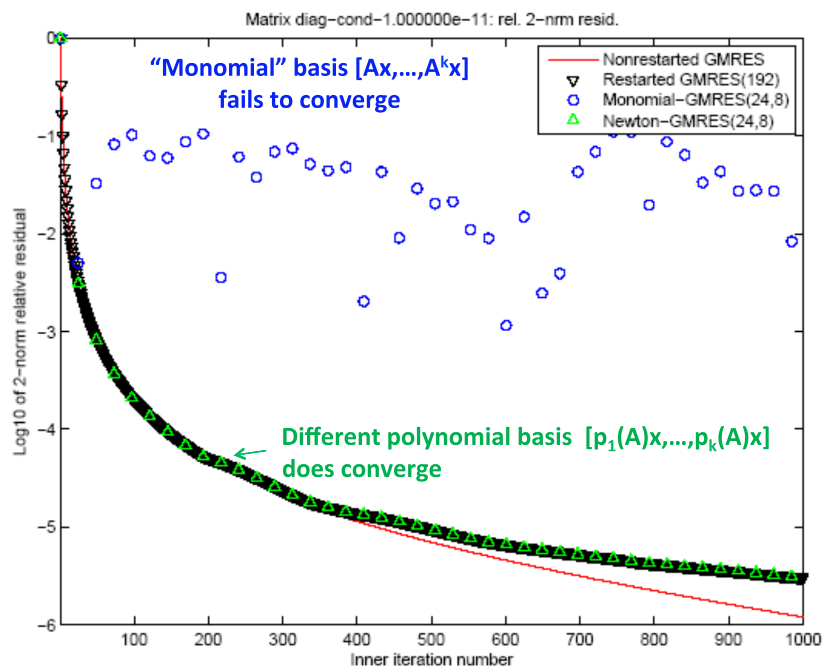
Communication-avoiding GMRES

$W = [\, v, Av, A^2v, \ldots, A^kv \,]$
[Q,R] = TSQR(W)
  … *"Tall Skinny QR"*
build H from R
solve LSQ problem with H

Sequential case: #words moved decreases by a factor of k
Parallel case: #messages decreases by a factor of k

•Oops – W from power method, precision lost!

94

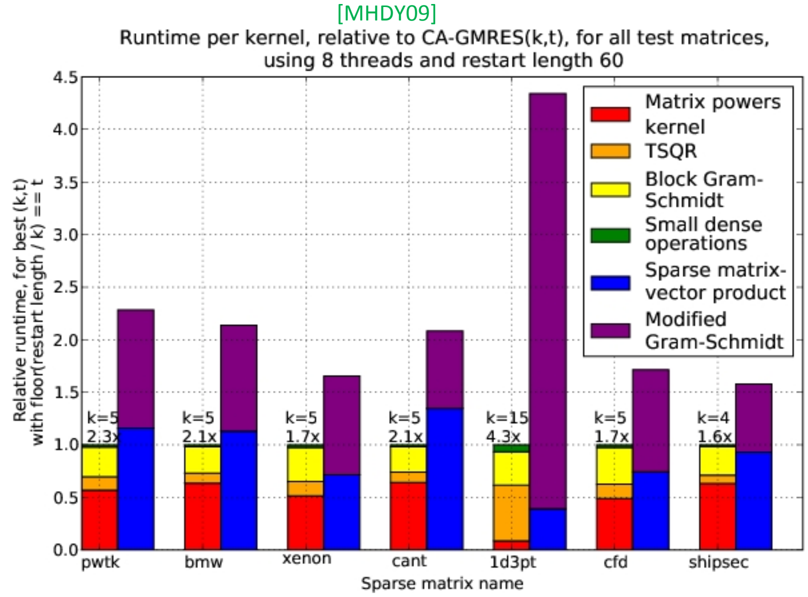


95

Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



96

CA-BiCGStab

Compute $r_0 = b - Ax_0$. Choose $r_0^\star$ arbitrary.
Set $p_0 = r_0$, $q_{-1} = 0_{N\times 1}$.
For $k = 0, 1, \dots,$ until convergence, Do

$P = [p_{sk}, Ap_{sk}, \dots, A^s p_{sk}]$
$Q = [q_{sk-1}, Aq_{sk-1}, \dots, A^s q_{sk-1}]$
$R = [r_{sk}, Ar_{sk}, \dots, A^s r_{sk}]$

//Compute the $1 \times (3s+3)$ Gram vector.
$g = (r_0^\star)^T [P, Q, R]$
//Compute the $(3s+3) \times (3s+3)$ Gram matrix

$$G = \begin{bmatrix} P^T \\ Q^T \\ R^T \end{bmatrix} \begin{bmatrix} P & Q & R \end{bmatrix}$$

For $\ell = 0$ to s,

$$b_{sk}^\ell = \left[B_1(:, \ell)^T, 0_{s+1}^T, 0_{s+1}^T\right]^T$$
$$c_{sk-1}^\ell = \left[0_{s+1}^T, B_2(:, \ell)^T, 0_{s+1}^T\right]^T$$
$$d_{sk}^\ell = \left[0_{s+1}^T, 0_{s+1}^T, B_3(:, \ell)^T\right]^T$$

For $j = 0$ to $\lfloor \frac{s}{2} \rfloor - 1$, Do

$$\alpha_{sk+j} = \frac{<g, d_{sk+j}^0>}{<g, b_{sk+j}^1>}$$
$$q_{sk+j} = r_{sk+j} - \alpha_{sk+j}[P, Q, R]b_{sk+j}^1$$

For $\ell = 0$ to $s - 2j + 1$, Do

$$c_{sk+j}^\ell = d_{sk+j}^\ell - \alpha_{sk+j} b_{sk+j-1}^{\ell+1}$$

//such that $[P, Q, R]\, c_{sk+j}^\ell = A^\ell q_{sk+j}$

$$\omega_{sk+j} = \frac{<c_{sk+j+1}^1, Gc_{sk+j+1}^0>}{<c_{sk+j+1}^1, Gc_{sk+j+1}^1>}$$
$$x_{sk+j+1} = x_{sk+j} + \alpha_{sk+j} p_{sk+j} + \omega_{sk+j} q_{sk+j}$$
$$r_{sk+j+1} = q_{sk+j} - \omega_{sk+j}[P, Q, R]c_{sk+j+1}^1$$

For $\ell = 0$ to $s - 2j$, Do

$$d_{sk+j+1}^\ell = c_{sk+j+1}^\ell - \omega_{sk+j} c_{sk+j+1}^{\ell+1}$$

//such that $[P, Q, R]\, d_{sk+j+1}^\ell = A^\ell r_{sk+j+1}$

$$\beta_{sk+j} = \frac{<g, d_{sk+j+1}^0>}{<g, d_{sk+j}^0>} \times \frac{\alpha}{\omega}$$
$$p_{sk+j+1} = r_{sk+j+1} + \beta_{sk+j} p_{sk+j} - \beta_{sk+j}\omega_{sk+j}[P, Q, R]b_{sk+j}^1$$

For $\ell = 0$ to $s - 2j$, Do

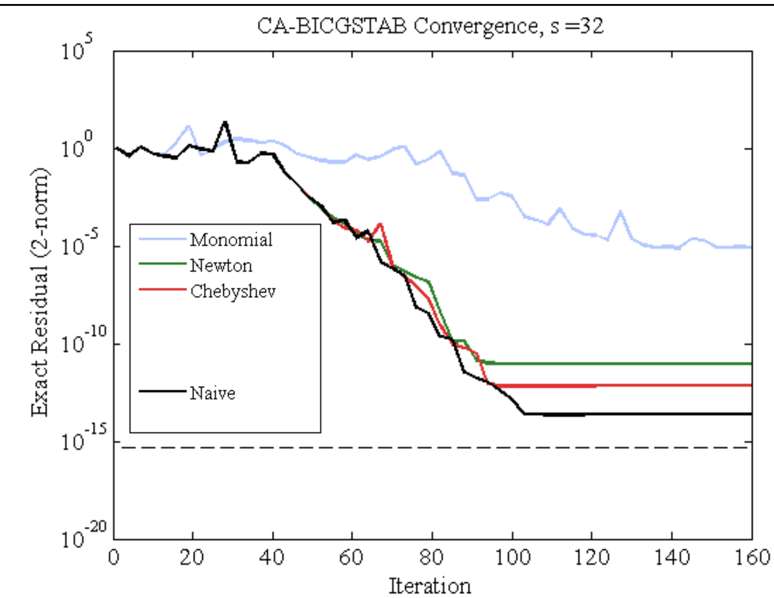
$$b_{sk+j+1}^\ell = d_{sk+j+1}^\ell + \beta_{sk+j} b_{sk+j}^\ell - \beta_{sk+j}\omega_{sk+j} b_{sk+j}^{\ell+1}$$

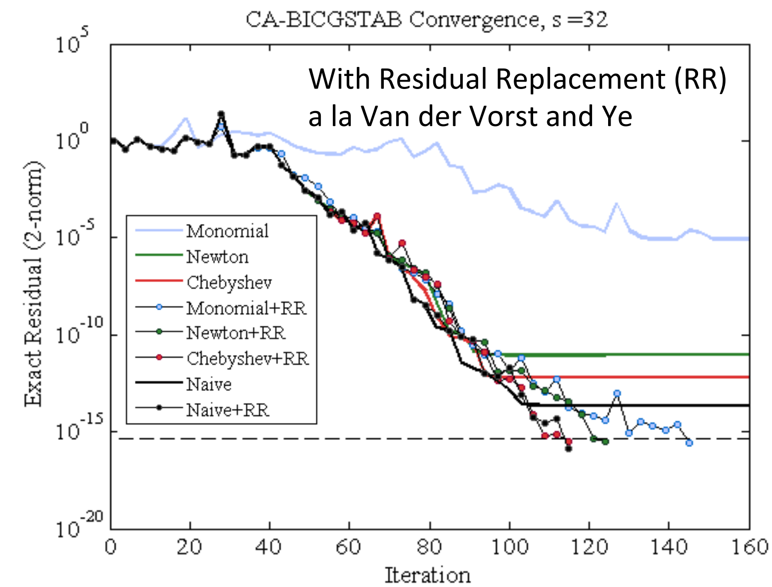
//such that $[P, Q, R]\, b_{sk+j+1}^\ell = A^\ell p_{sk+j+1}$.

EndDo
EndDo

1. Compute $r_0 := b - Ax_0$; r_0^* arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \dots,$ until convergence Do:
4. $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $s_j := r_j - \alpha_j Ap_j$
6. $\omega_j := (As_j, s_j)/(As_j, As_j)$
7. $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8. $r_{j+1} := s_j - \omega_j As_j$
9. $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo

97

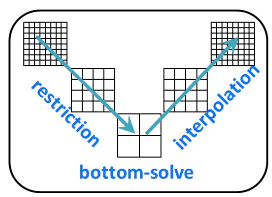




	Naive	Monomial	Newton	Chebyshev
Replacement Its.	74 **(1)**	[7, 15, 24, 31, …, 92, 97, 103] **(17)**	[67, 98] **(2)**	68 **(1)**

Sample Application Speedups

- Geometric Multigrid (GMG) w CA Bottom Solver
  - Compared **BICGSTAB** vs. **CA-BICGSTAB** with s = 4
  - Hopper at NERSC (Cray XE6), weak scaling: Up to 4096 MPI processes (24,576 cores total)



  - Speedups for miniGMG benchmark (HPGMG benchmark predecessor)
    - **4.2x** in bottom solve, **2.5x** overall GMG solve
  - Implemented as a solver option in BoxLib and CHOMBO AMR frameworks
    - 3D LMC (a low-mach number combustion code)
      - **2.5x** in bottom solve, **1.5x** overall GMG solve
    - 3D Nyx (an N-body and gas dynamics code)
      - **2x** in bottom solve, **1.15x** overall GMG solve
- Solve Horn-Schunck Optical Flow Equations
  - Compared **CG** vs. **CA-CG** with s = 3, **43%** faster on NVIDIA GT 640 GPU

Tuning space for Krylov Methods

- Classifications of sparse operators for avoiding communication
  - Explicit indices or nonzero entries cause most communication, along with vectors
  - Ex: With stencils (all implicit) all communication for vectors

Indices

Nonzero entries

	Explicit (O(nnz))	Implicit (o(nnz))
Explicit (O(nnz))	CSR and variations	Vision, climate, AMR, …
Implicit (o(nnz))	Graph Laplacian	Stencils

- Operations
  - $[x, Ax, A^2x, \ldots, A^kx]$ or $[x, p_1(A)x, p_2(A)x, \ldots, p_k(A)x]$
  - Number of columns in x
  - $[x, Ax, A^2x, \ldots, A^kx]$ and $[y, A^Ty, (A^T)^2y, \ldots, (A^T)^ky]$, or $[y, A^TAy, (A^TA)^2y, \ldots, (A^TA)^ky]$,
  - return all vectors or just last one
- Cotuning and/or interleaving
  - $W = [x, Ax, A^2x, \ldots, A^kx]$ and {TSQR(W) or W^TW or … }
  - Ditto, but throw away W
- Preconditioned versions

Summary of Iterative Linear Algebra

- New Lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - Many different algorithms reorganized
    - More underway, more to be done
  - Need to recognize stable variants more easily
  - Preconditioning
    - "Underlapping" instead of "overlapping" Domain Decomposition
    - Hierarchically Semiseparable Matrices
  - Autotuning and synthesis
    - pOSKI for SpMV – available at bebop.cs.berkeley.edu
    - Different kinds of "sparse matrices"

Outline

- "Direct" Linear Algebra
  - Lower bounds on communication
  - New algorithms that attain these lower bounds
- Ditto for programs accessing arrays (eg n-body)
- Ditto for "Iterative" Linear Algebra
- Related work
  - Write-Avoiding Algorithms
  - Reproducibility

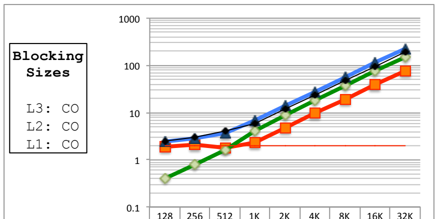
Write-Avoiding Algorithms

- What if writes are more expensive than reads?
  - Nonvolatile Memory (Flash, PCM, …)
  - Saving intermediates to disk in cloud (eg Spark)
  - Extra coherency traffic in shared memory
- Can we design "write-avoiding (WA)" algorithms?
  - Goal: find and attain better lower bound for writes
  - Thm: For classical matmul, possible to do asymptotically fewer writes than reads to given layer of memory hierarchy
  - Thm: Cache-oblivious algorithms cannot be write-avoiding
  - Thm: Strassen and FFT cannot be write-avoiding

Measured L3-DRAM traffic on Intel Nehalem Xeon-7560

Optimal #DRAM reads = $O(n^3/M^{1/2})$

Optimal #DRAM writes = n^2

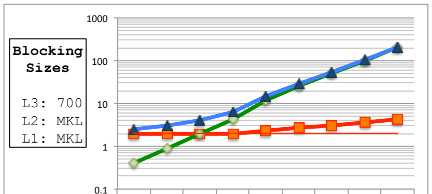


	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	1.9	2.1	1.8	2.3	4.8	9.8	19.5	39.6	78.5
L3_VICTIMS.E	0.4	0.8	1.6	4.2	8.8	17.9	36.6	75.4	147.5
LLC_S_FILLS.E	2.4	2.8	3.8	6.9	14	28.1	56.5	115.5	226.6
Misses on Ideal Cache	2.512	3.024	4.048	6	12	24	48	96	192
Write L.B.	2	2	2	2	2	2	2	2	2

Cache-Oblivious Matmul

#DRAM reads close to optimal

#DRAM writes much larger



	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	2	2	1.9	2	2.3	2.7	3	3.6	4.4
L3_VICTIMS.E	0.4	0.9	1.9	4.2	11.8	25.3	50.7	101.7	203.2
LLC_S_FILLS.E	2.5	3	4.1	6.5	14.5	28.3	54	105.7	208.2
Write L.B.	2	2	2	2	2	2	2	2	2

Write-Avoiding Matmul

Total L3 misses close to optimal

Total DRAM writes much larger

Reproducible Floating Point Computation

- Do you get the same answer if you run the same program twice with the same input?
  - Not even on your multicore laptop!
- Floating point addition is nonassociative, summation order not reproducible
- First release of the ReproBLAS
  - Reproducible BLAS 1, independent of data order, number of processors, data layout, reduction tree, …
  - Sequential and distributed memory (MPI)
- bebop.cs.berkeley.edu/reproblas

More possible class projects

- Could be one or more of
  - Extend lower bounds, new algorithms, compare existing algorithms, use algorithms to improve existing applications, performance analysis/measurement, compiler infrastructure, present existing results…
- Extend to machine learning algorithms, other of the ``13 motifs''

For more details

- Bebop.cs.berkeley.edu
  - 155 page survey in Acta Numerica
- CS267 – Berkeley's Parallel Computing Course
  - Live broadcast in Spring 2016
    - www.cs.berkeley.edu/~demmel
    - All slides, video available
  - Prerecorded version broadcast since Spring 2013
    - www.xsede.org
    - Free supercomputer accounts to do homework
    - Free autograding of homework

Collaborators and Supporters

- **James Demmel**, **Kathy Yelick**, Michael Anderson, Grey Ballard, Erin Carson, Aditya Devarakonda, Michael Driscoll, David Eliahu, Andrew Gearhart, Evangelos Georganas, Nicholas Knight, Penporn Koanantakool, Ben Lipshitz, Oded Schwartz, Edgar Solomonik, Omer Spillinger
- Austin Benson, Maryam Dehnavi, Mark Hoemmen, Shoaib Kamil, Marghoob Mohiyuddin
- Abhinav Bhatele, Aydin Buluc, Michael Christ, Ioana Dumitriu, Armando Fox, David Gleich, Ming Gu, Jeff Hammond, Mike Heroux, Olga Holtz, Kurt Keutzer, Julien Langou, Devin Matthews, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang
- Jack Dongarra, Jakub Kurzak, Dulceneia Becker, Ichitaro Yamazaki, …
- Sivan Toledo, Alex Druinsky, Inon Peled
- Laura Grigori**,** Sebastien Cayrols, Simplice Donfack, Mathias Jacquelin, Amal Khabou, Sophie Moufawad, Mikolaj Szydlarski
- Members of ParLab, ASPIRE, BEBOP, CACHE, EASI, FASTMath, MAGMA, PLASMA
- Thanks to DOE, NSF, UC Discovery, INRIA, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle
- bebop.cs.berkeley.edu

Summary

Time to redesign all linear algebra, n-body,…
algorithms and software
(and compilers…)

Don't Communic…

110