Communication-Avoiding Algorithms for Linear Algebra and Beyond

Jim Demmel

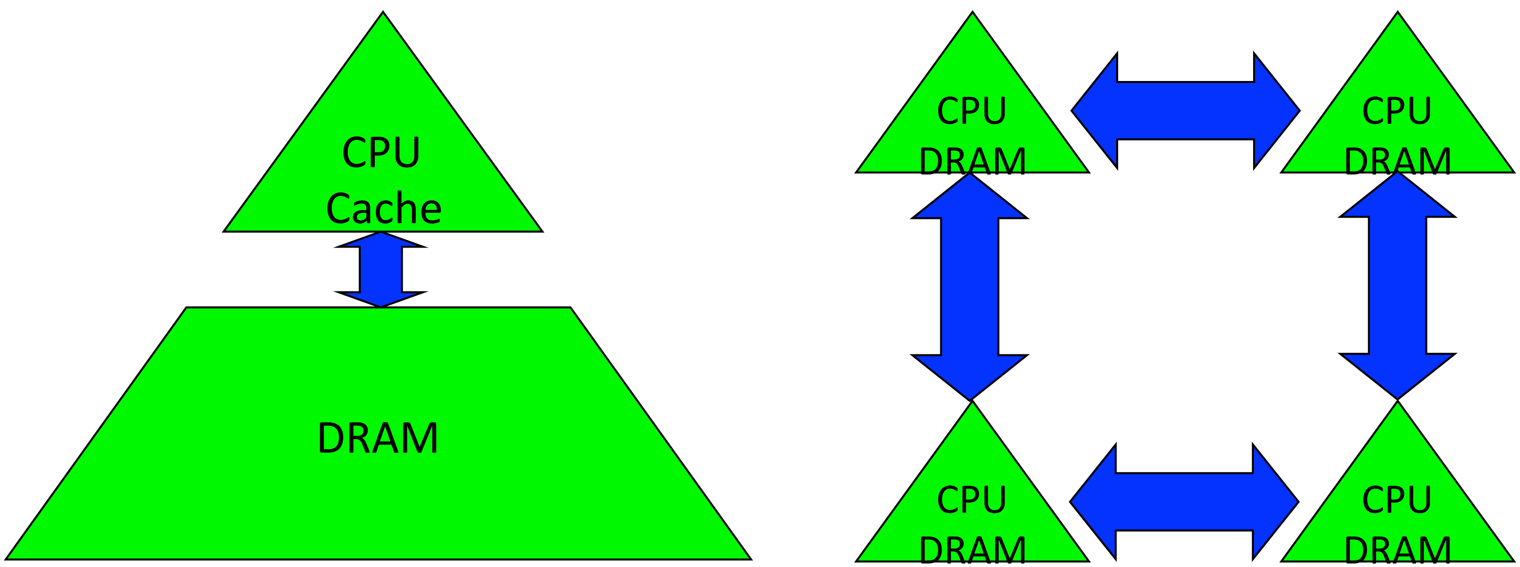
Math & EECS Departments

UC Berkeley

Why avoid communication? (1/3)

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

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

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%

Why avoid communication? (2/3)

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

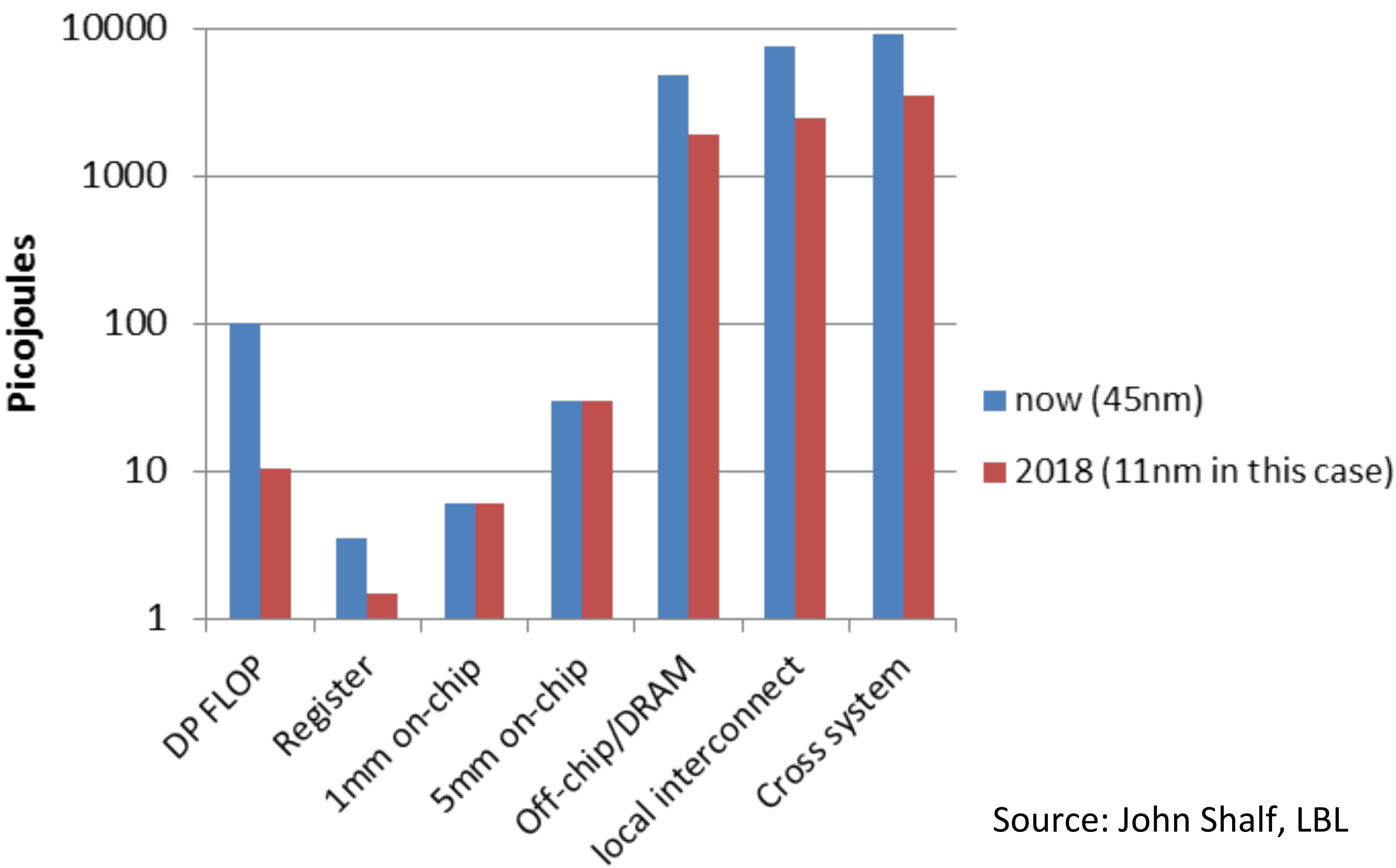
  communication

- 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%

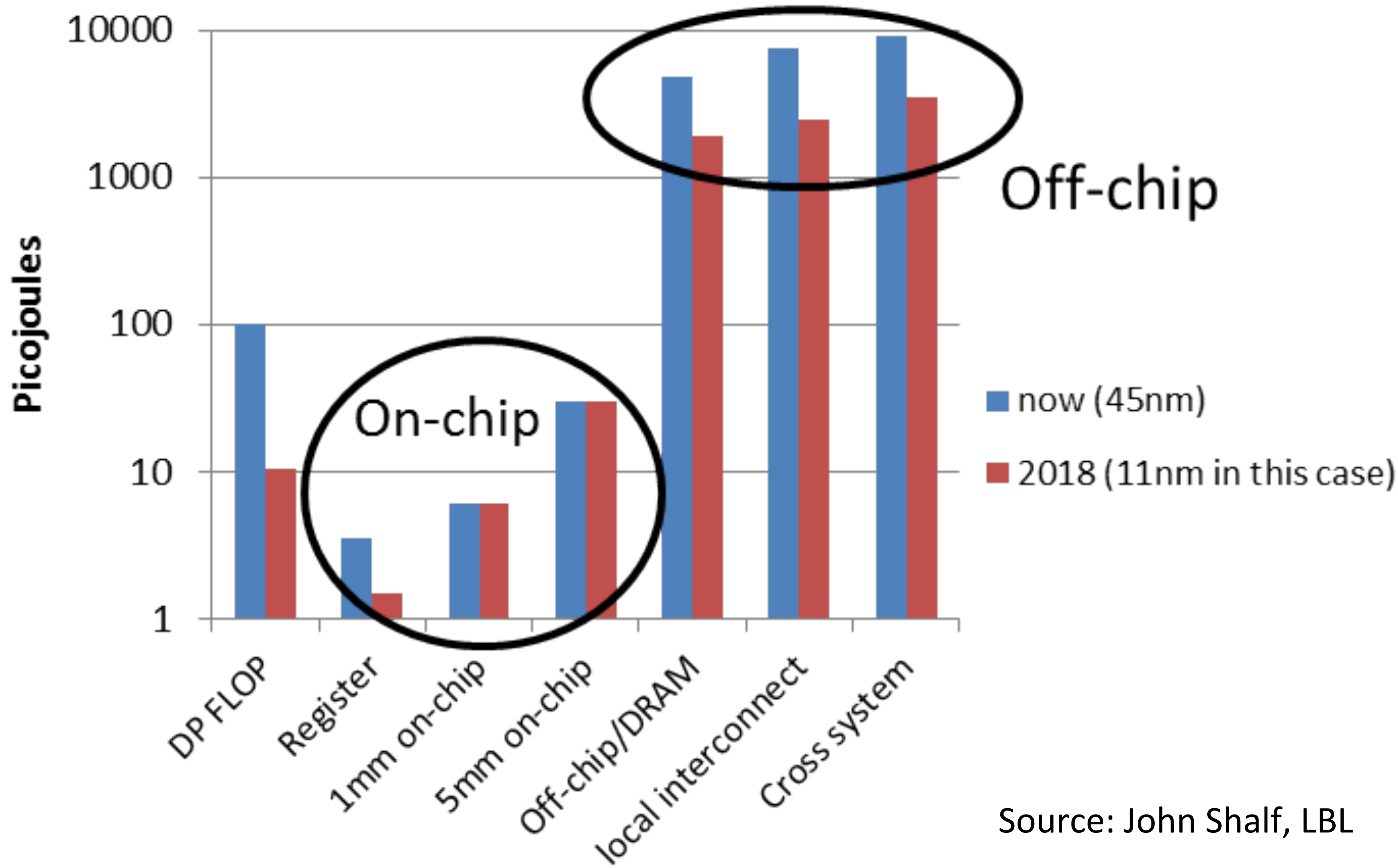
- Avoid communication to save time

Why Minimize Communication? (3/3)



Source: John Shalf, LBL

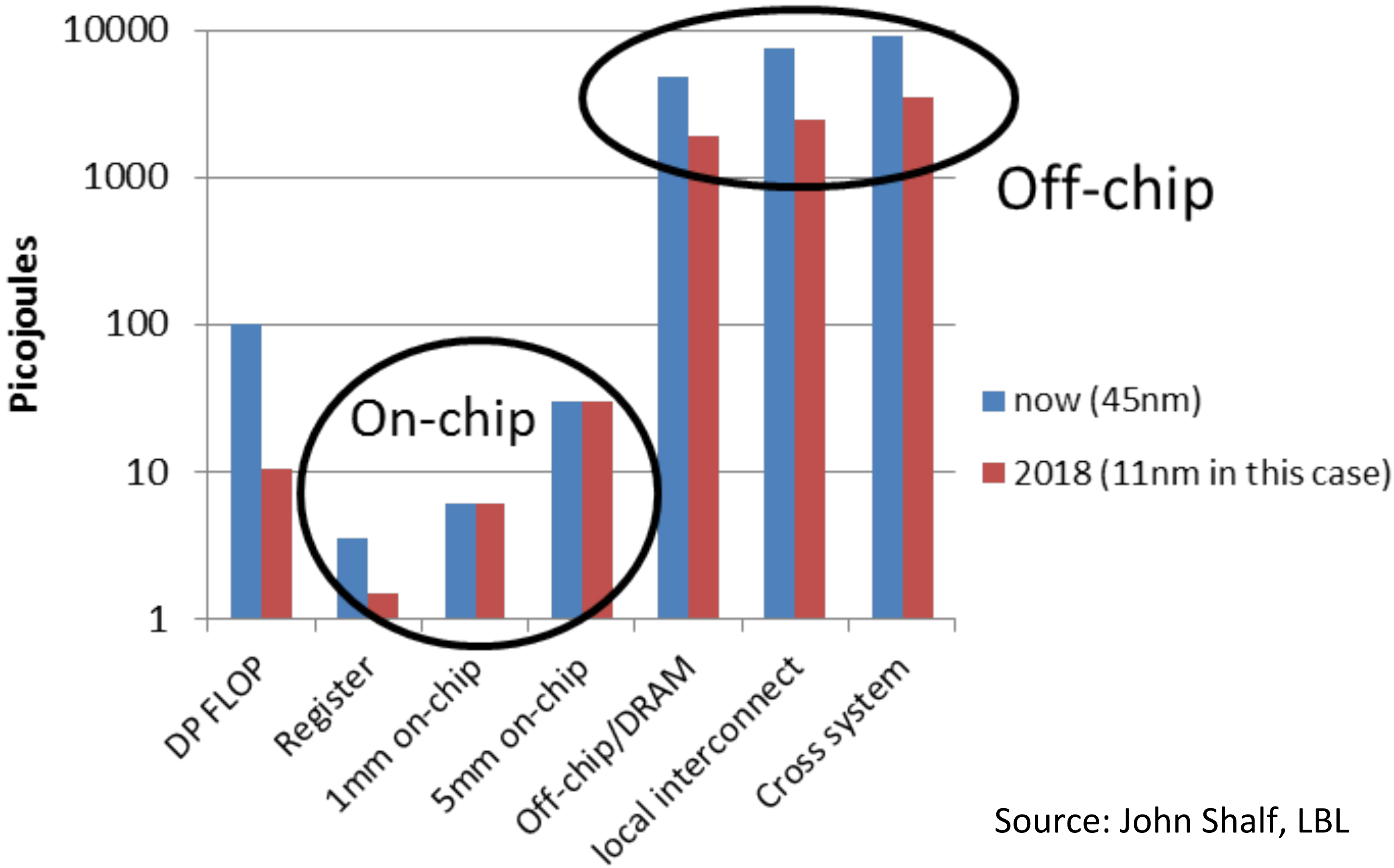
Why Minimize Communication? (3/3)



Source: John Shalf, LBL

Why Minimize Communication? (3/3)

Minimize communication to save energy

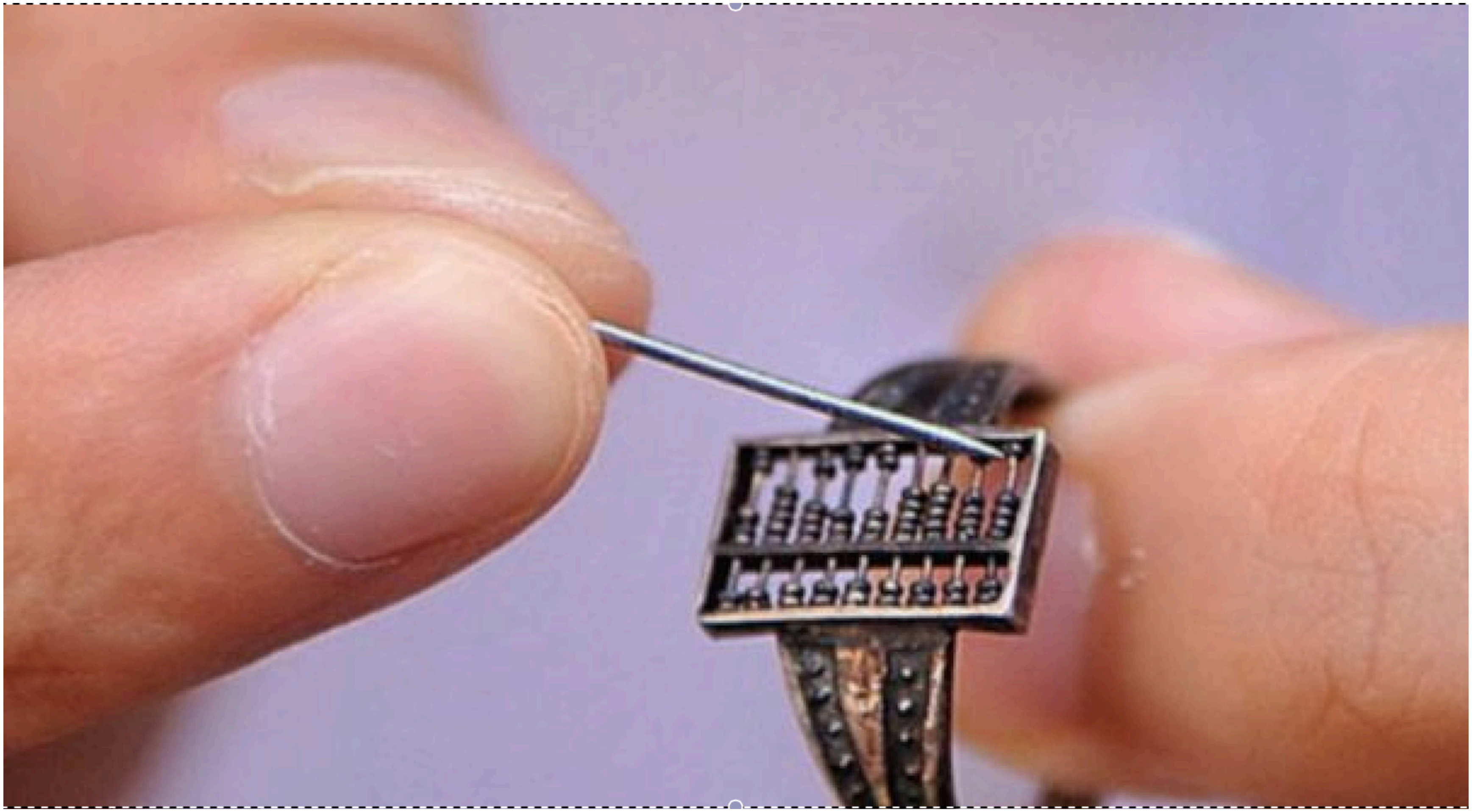


Source: John Shalf, LBL

Alternative Cost Model for Algorithms?

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 $\leftrightarrow$ L2 $\leftrightarrow$ DRAM $\leftrightarrow$ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible

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

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.

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)

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to “any algorithm with arrays”
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

Summary of CA Linear Algebra

- “Direct” Linear Algebra

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible
  - Autotuning to find optimal implementation

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - 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 "Iterative" Linear Algebra

Lower bound for all "n^3-like" linear algebra

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

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

Lower bound for all "n^3-like" 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

Lower bound for all “n^3-like” 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

Lower bound for all "n^3-like" 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, 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)

Lower bound for all "n^3-like" linear algebra

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

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

#messages_sent ≥ #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)

Lower bound for all “n^3-like” 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)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

Lower bound for all "n^3-like" 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

Can we attain these lower bounds?

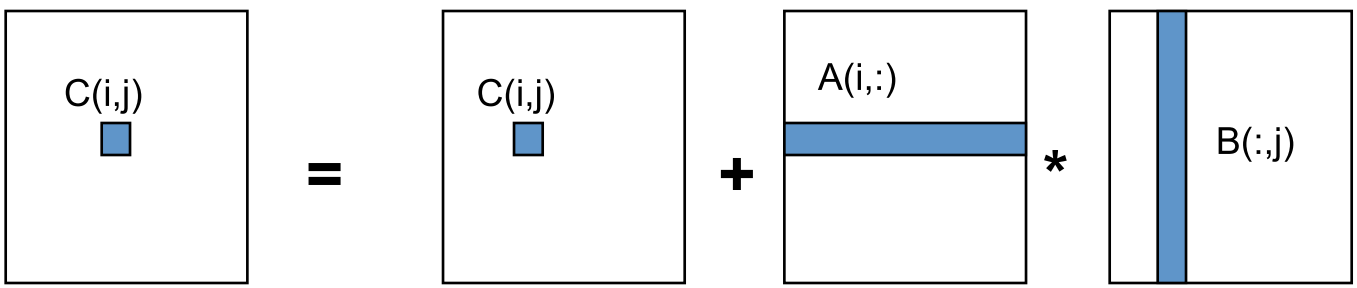
- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Often not
- If not, are there other algorithms that do?
  - Yes, for much of dense linear algebra, APSP
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations (need those too!)
- Only a few sparse algorithms so far
  - Ex: Matmul of "random" sparse matrices
  - Ex: Sparse Cholesky of matrices with "large" separators
- Lots of work in progress

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

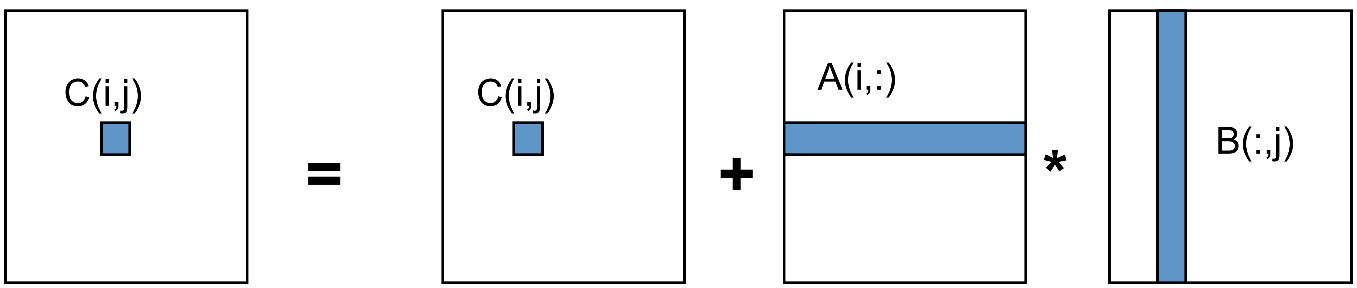
Naïve Matrix Multiply

```
{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)
```



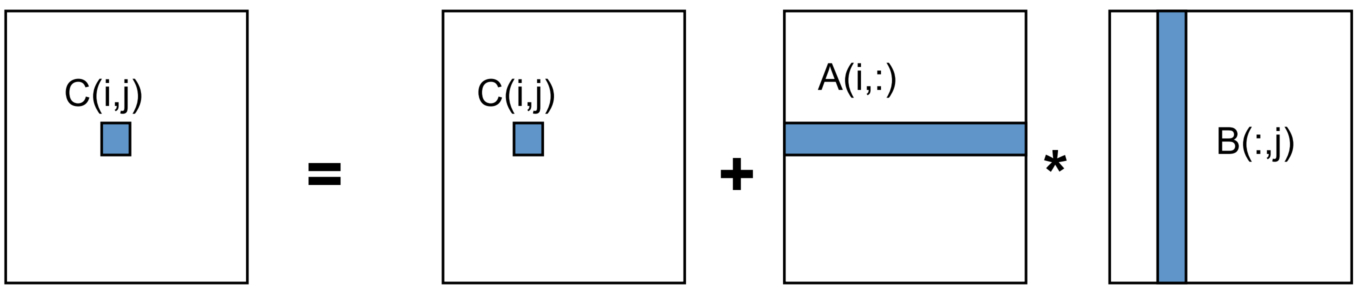
Naïve Matrix Multiply

```
{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}
```



Naïve Matrix Multiply

{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory} … n^2 reads altogether
    for j = 1 to n
      {read C(i,j) into fast memory} … n^2 reads altogether
      {read column j of B into fast memory} … n^3 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} … n^2 writes altogether

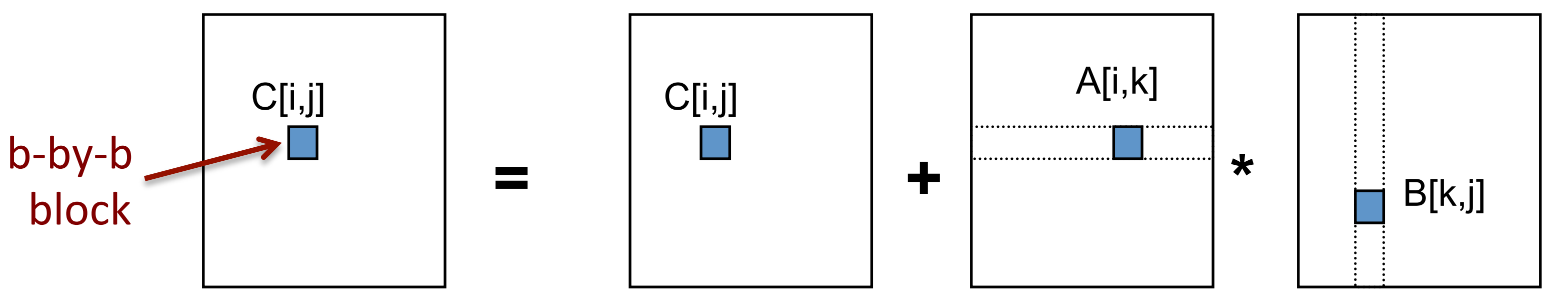


$n^3 + 3n^2$ reads/writes altogether – dominates $2n^3$ arithmetic

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

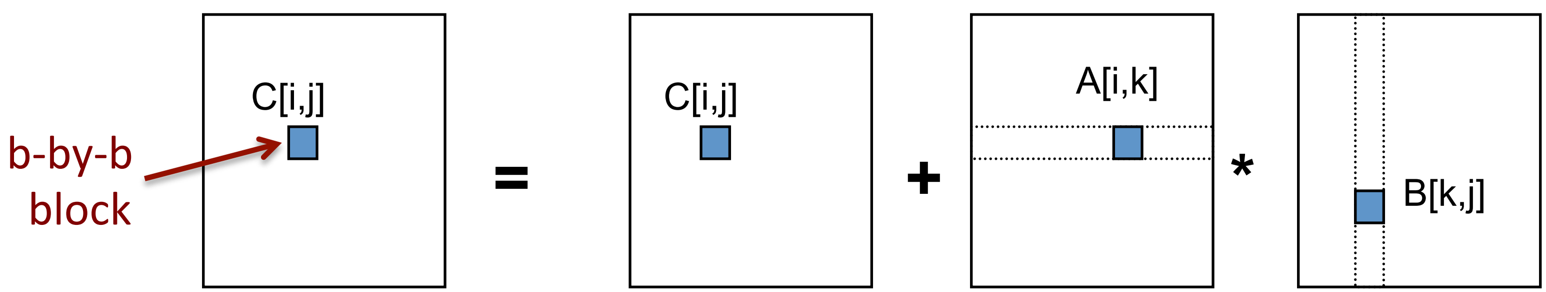
```
for i = 1 to n/b
  for j = 1 to n/b
    {read block C[i,j] into fast memory}
    for k = 1 to n/b
        {read block A[i,k] into fast memory}
        {read block B[k,j] into fast memory}
         C[i,j] = C[i,j] + A[i,k] * B[k,j] {do a matrix multiply on b-by-b blocks}
    {write block C[i,j] back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



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} ... $b^2 \times (n/b)^2 = n^2$ reads
    for k = 1 to n/b
      {read block A[i,k] into fast memory} ... $b^2 \times (n/b)^3 = n^3/b$ reads
      {read block B[k,j] into fast memory} ... $b^2 \times (n/b)^3 = n^3/b$ reads
      C[i,j] = C[i,j] + A[i,k] * B[k,j] {do a matrix multiply on b-by-b blocks}
    {write block C[i,j] back to slow memory} ... $b^2 \times (n/b)^2 = n^2$ writes



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

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: $3b^2 \le M$, so #reads/writes $\ge 3^{1/2}n^3/M^{1/2} + 2n^2$
- Attains lower bound $= \Omega$ (#flops / $M^{1/2}$)

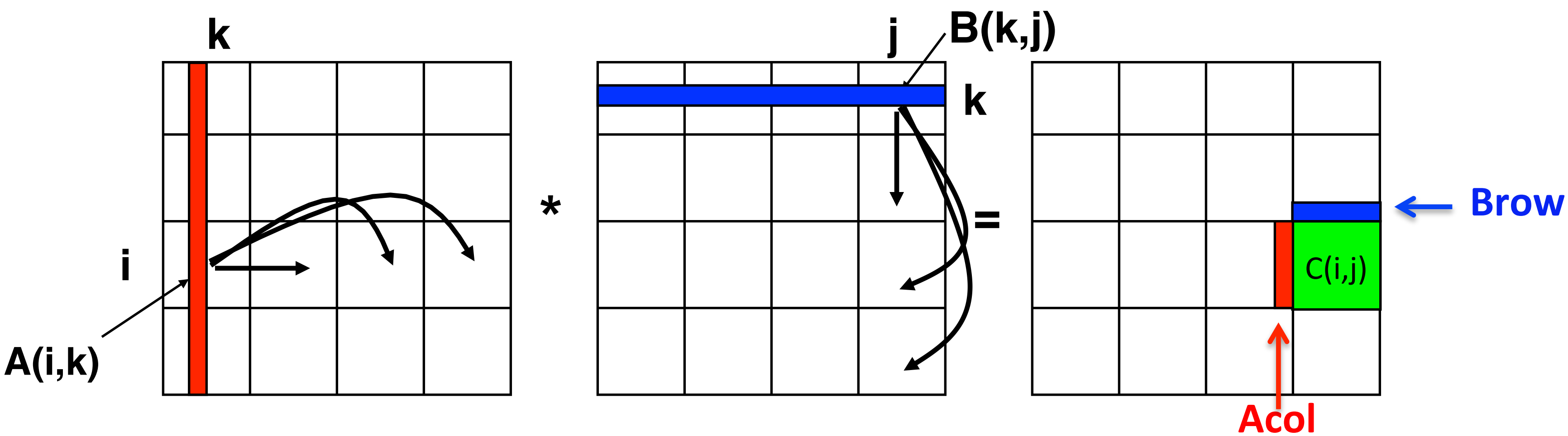
Does blocked matmul attain lower bound?

- 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 $\geq 3^{1/2}n^3/M^{1/2} + 2n^2$
- Attains lower bound $= \Omega$ (#flops / $M^{1/2}$)

- But what if we don't know M?
- Or if there are multiple levels of fast memory?
- Can use "Cache Oblivious" algorithm (divide and conquer)

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

(nearly) optimal using minimum memory $M=O(n^2/P)$



For k=0 to n/b-1 … b = block size = #cols in A(i,k) = #rows in B(k,j)

for all i = 1 to $P^{1/2}$

owner of A(i,k) broadcasts it to processor row i

for all j = 1 to $P^{1/2}$

owner of B(k,j) broadcasts it to processor column k

Receive A(i,k) into Acol

Receive B(k,j) into Brow

C_myproc = C_myproc + Acol * Brow

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

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)

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?

Can we do better?

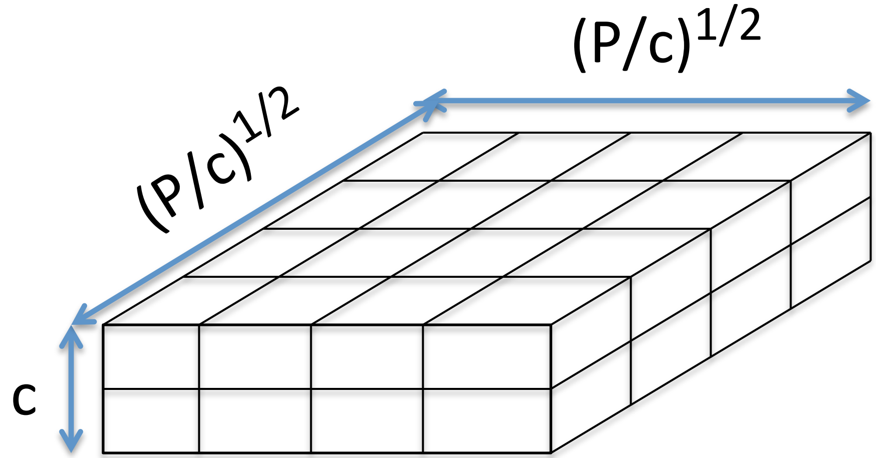
- Aren't we already optimal?
- Why assume $M = O(n^2/p)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?
- Special case: "3D Matmul"
  - Uses $M = O(n^2/p^{2/3})$
  - Dekel, Nassimi, Sahni [81], Bernsten [89], Agarwal, Chandra, Snir [90], Johnson [93], Agarwal, Balle, Gustavson,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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to “any algorithm with arrays”
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

2.5D Matrix Multiplication

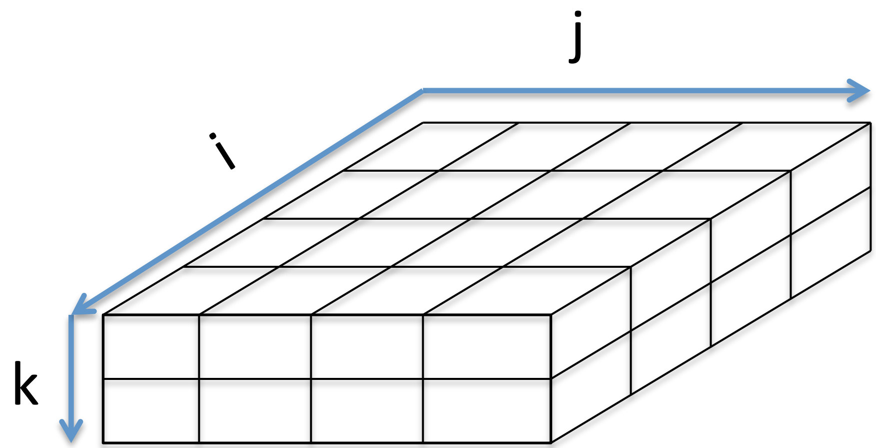
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

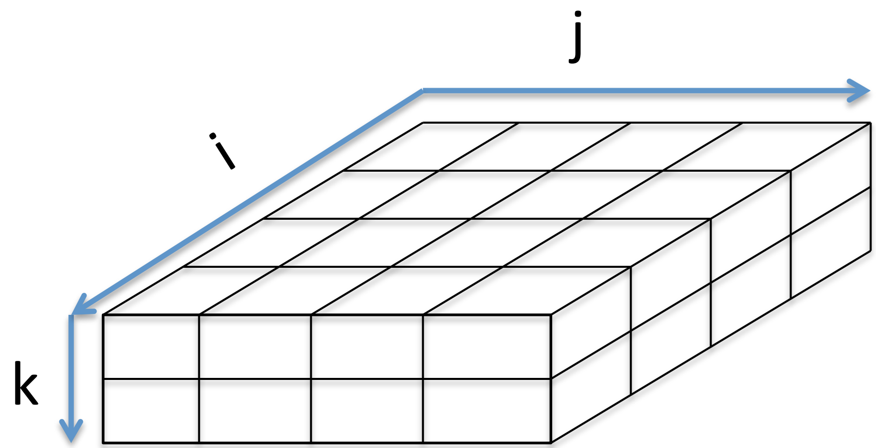
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



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

2.5D Matrix Multiplication

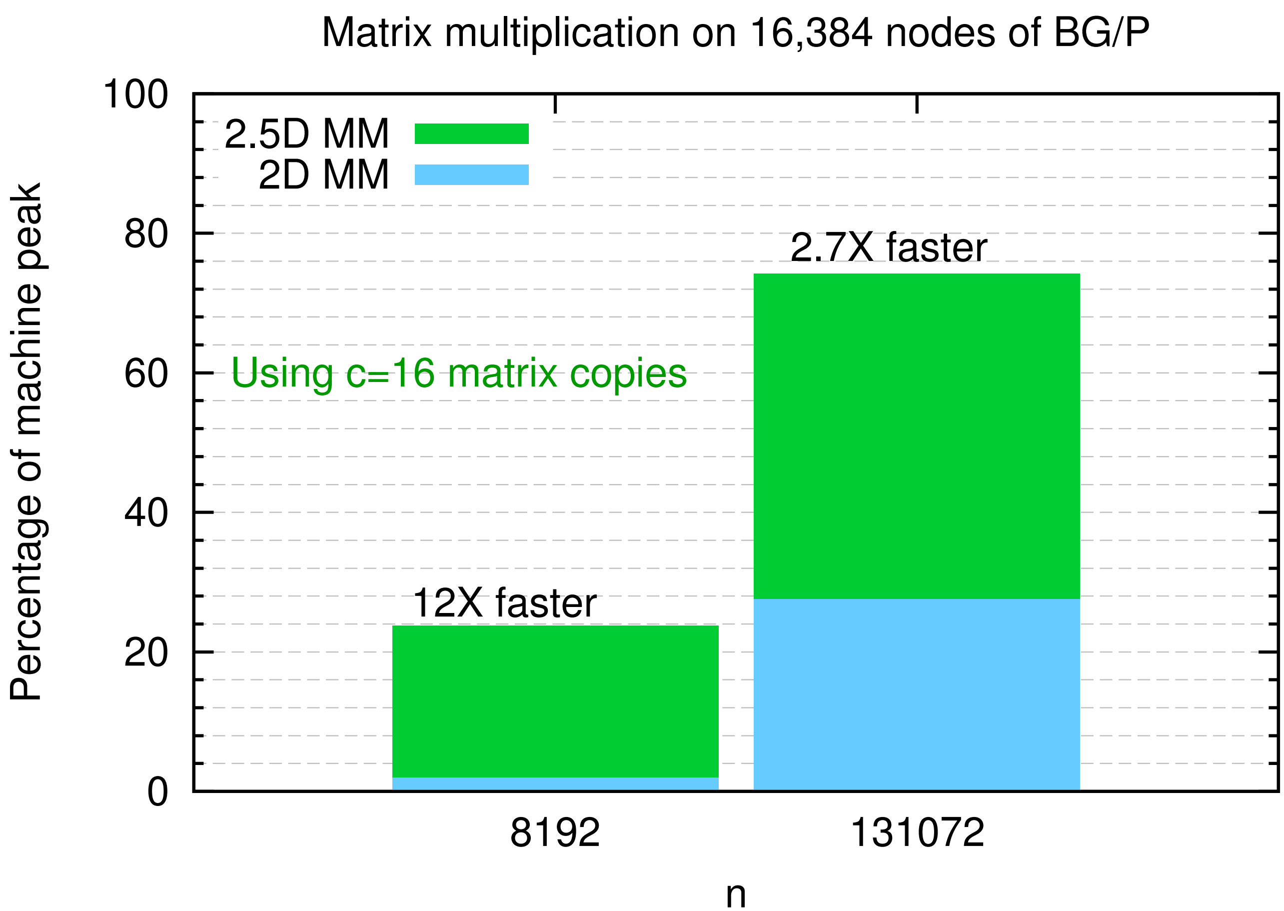
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



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

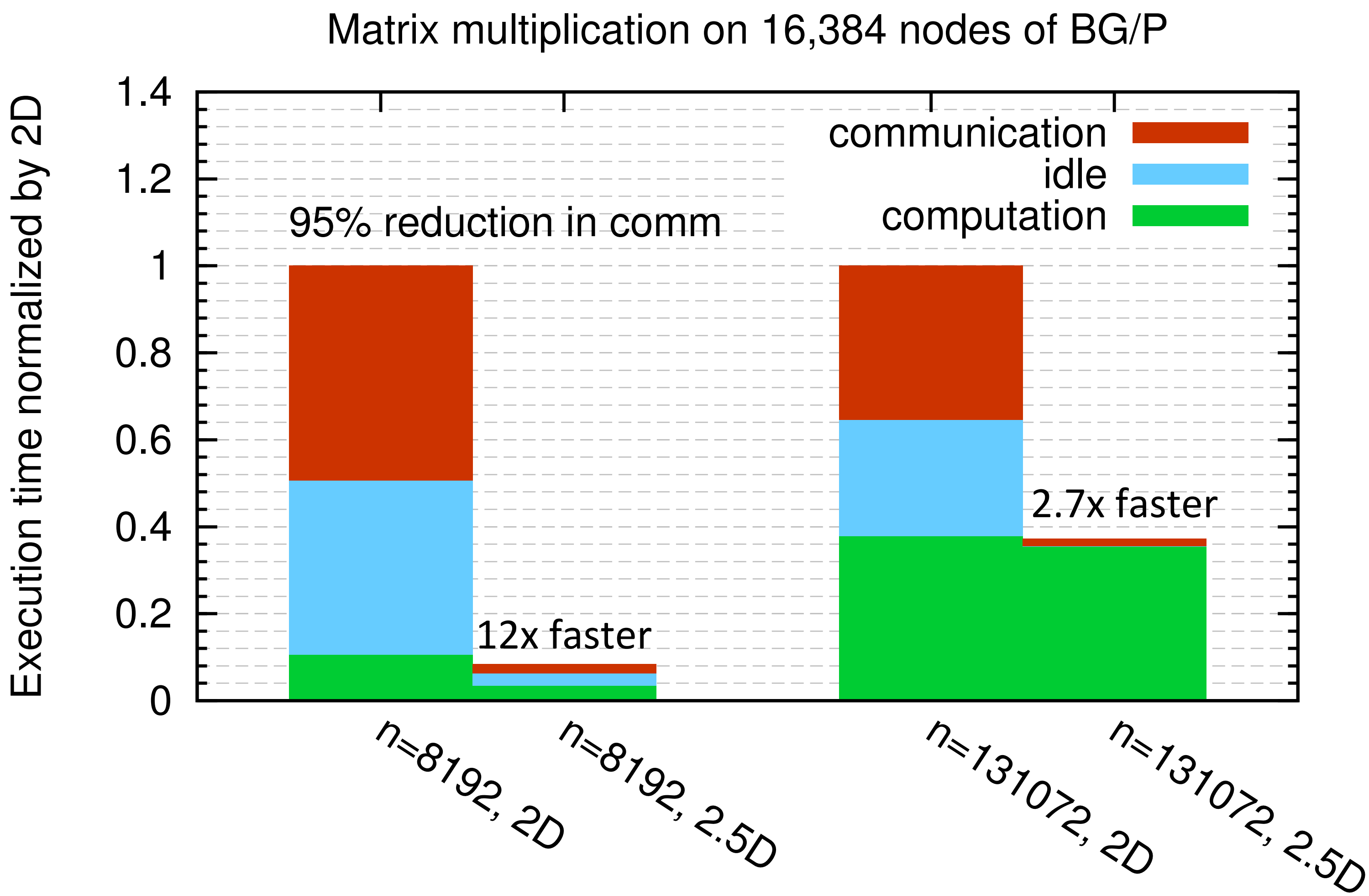
(1) P(i,j,0) broadcasts A(i,j) and B(i,j) to P(i,j,k)
(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of Σ_m A(i,m)*B(m,j)
(3) Sum-reduce partial sums Σ_m A(i,m)*B(m,j) along k-axis so P(i,j,0) owns C(i,j)

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



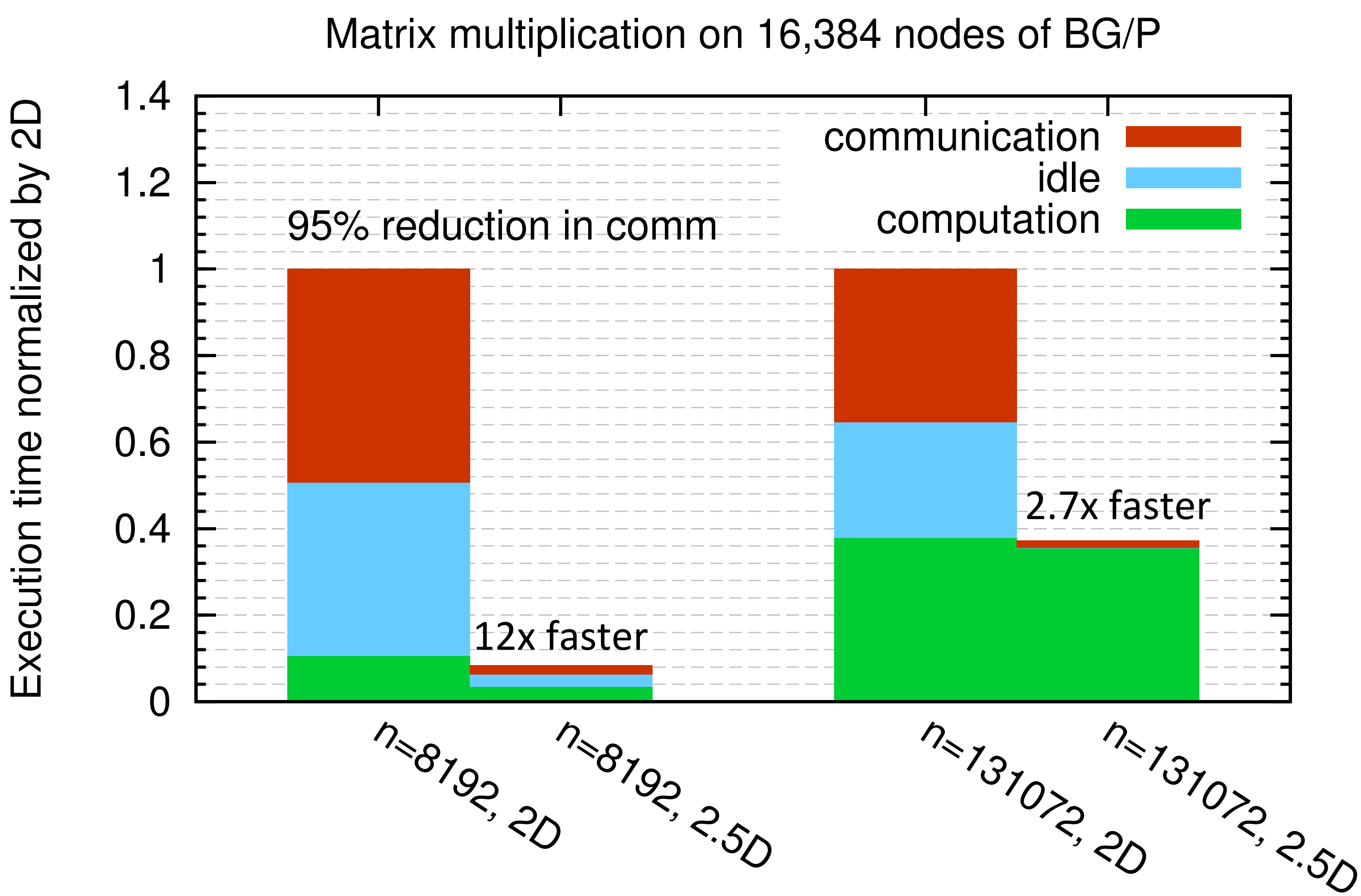
2.5D Matmul on BG/P, 16K nodes / 64K cores

c = 16 copies



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

c = 16 copies



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

Perfect Strong Scaling – in Time and Energy

Perfect Strong Scaling – in Time and Energy

- 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

Perfect Strong Scaling – in Time and Energy

- 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

Perfect Strong Scaling – in Time and Energy

- 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
- $T(cP) = n^3/(cP)\ [\ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})\ ]$

Perfect Strong Scaling – in Time and Energy

- 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
- $T(cP) = n^3/(cP)\ [\ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})\ ]$

  $= T(P)/c$

Perfect Strong Scaling – in Time and Energy

- 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.

Perfect Strong Scaling – in Time and Energy

- 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.
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E M T(cP) + \varepsilon_E T(cP)\ \}$

Perfect Strong Scaling – in Time and Energy

- 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.
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E M T(cP) + \varepsilon_E T(cP)\ \}$

  $= E(P)$

Perfect Strong Scaling – in Time and Energy

- 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.
- $E(cP) = cP \, \{ \, n^3/(cP) \, [\, \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2}) \,] + \delta_E M T(cP) + \varepsilon_E T(cP) \, \}$
  $= E(P)$
- Extends to N-body, Strassen, …
- Can prove lower bounds on needed network (eg 3D torus for matmul)

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to “any algorithm with arrays”
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

TSQR: QR of a Tall, Skinny matrix

$$W = \left(\begin{array}{c} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{array}\right)$$

TSQR: QR of a Tall, Skinny matrix

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

TSQR: QR of a Tall, Skinny matrix

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

TSQR: QR of a Tall, Skinny matrix

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

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

TSQR: QR of a Tall, Skinny matrix

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

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

TSQR: QR of a Tall, Skinny matrix

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

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

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

TSQR: QR of a Tall, Skinny matrix

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

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

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

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} }

TSQR: An Architecture-Dependent Algorithm

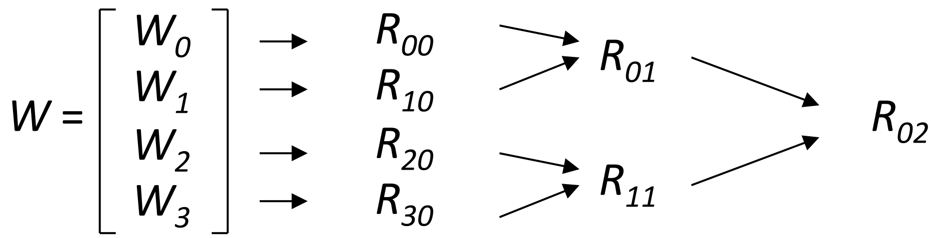
Parallel:

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

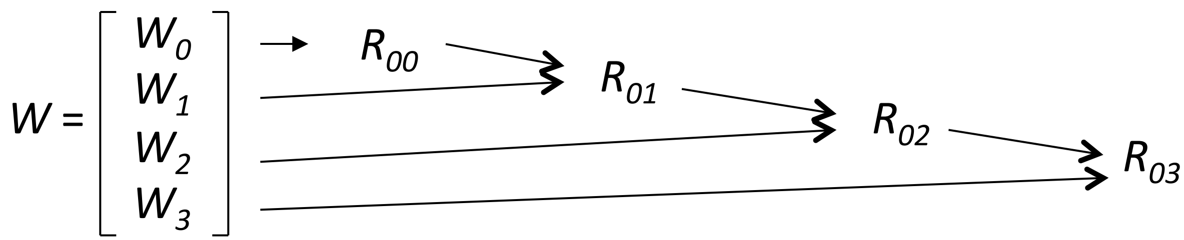
$$\begin{tikzcd}
W_0 \arrow[r] & R_{00} \arrow[dr] & & \\
W_1 \arrow[r] & R_{10} \arrow[r] & R_{01} \arrow[dr] & \\
W_2 \arrow[r] & R_{20} \arrow[r] & R_{11} \arrow[r] & R_{02} \\
W_3 \arrow[r] & R_{30} \arrow[ur] & &
\end{tikzcd}$$

TSQR: An Architecture-Dependent Algorithm

Parallel:

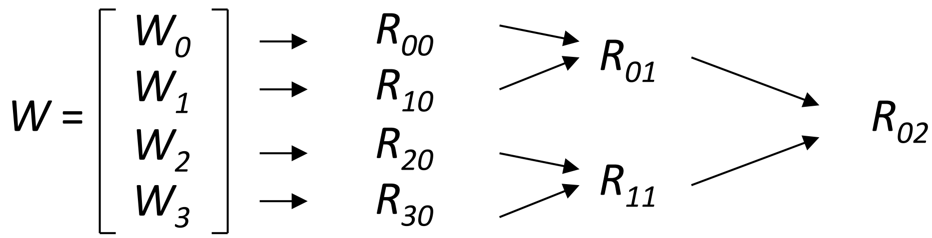


Sequential:

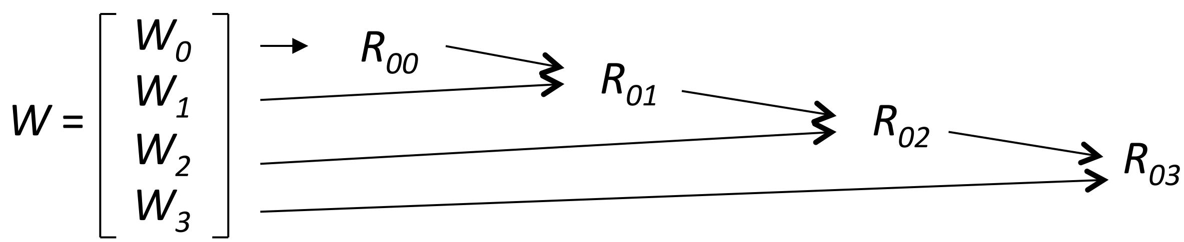


TSQR: An Architecture-Dependent Algorithm

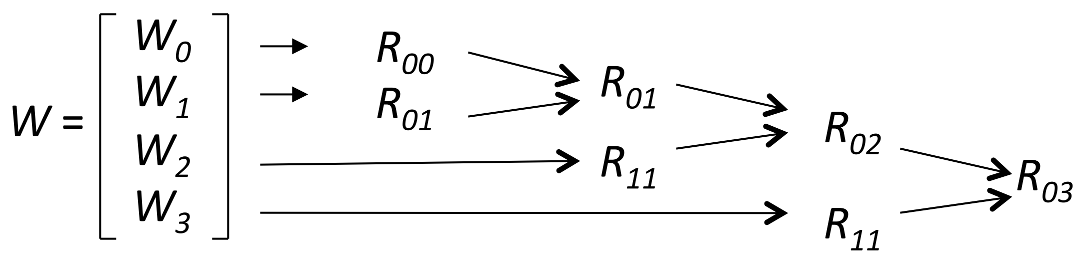
Parallel:



Sequential:

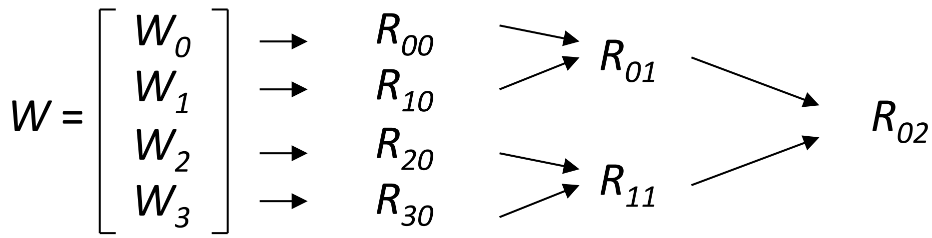


Dual Core:

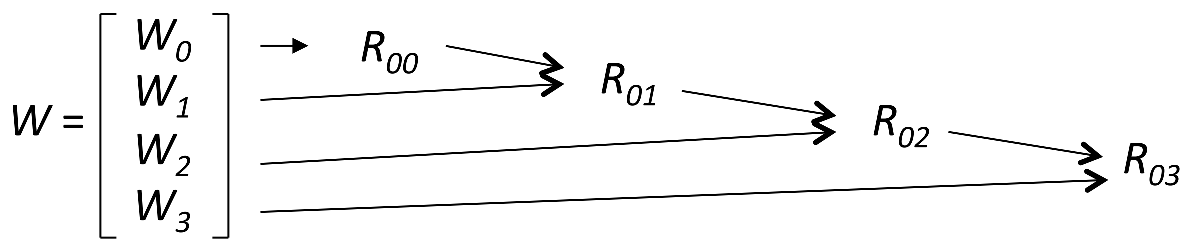


TSQR: An Architecture-Dependent Algorithm

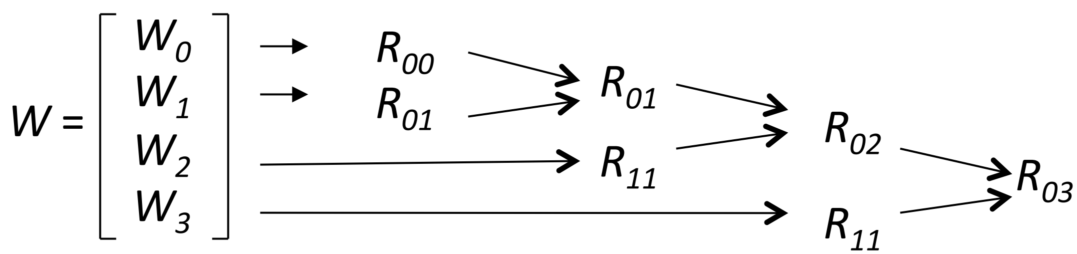
Parallel:



Sequential:



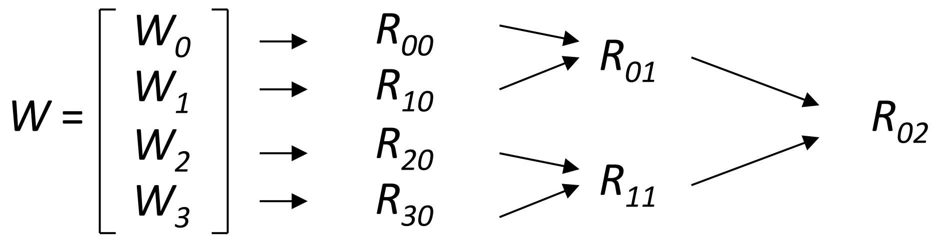
Dual Core:



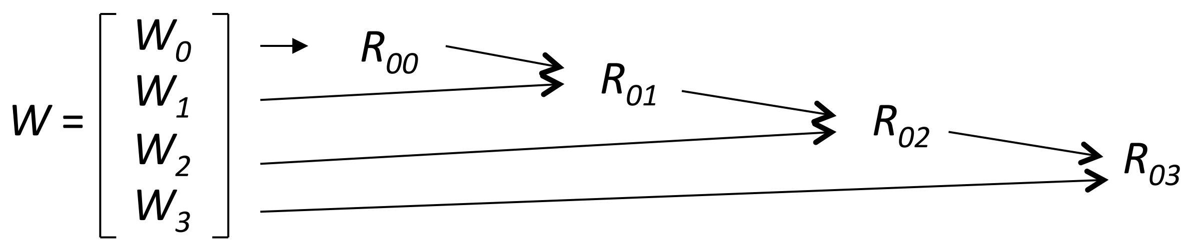
Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

TSQR: An Architecture-Dependent Algorithm

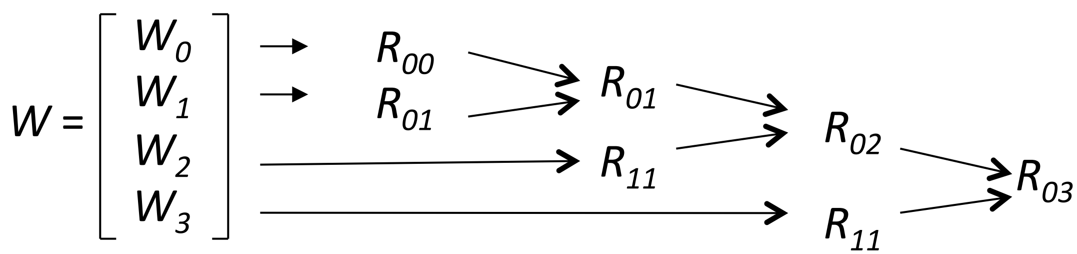
Parallel:



Sequential:



Dual Core:



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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

Communication Lower Bounds for Strassen-like matmul algorithms

Communication Lower Bounds for Strassen-like matmul algorithms

Classical
$O(n^3)$ matmul:

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

Communication Lower Bounds for Strassen-like matmul algorithms

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

Communication Lower Bounds for Strassen-like matmul algorithms

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

Communication Lower Bounds for Strassen-like matmul algorithms

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

Communication Lower Bounds for Strassen-like matmul algorithms

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

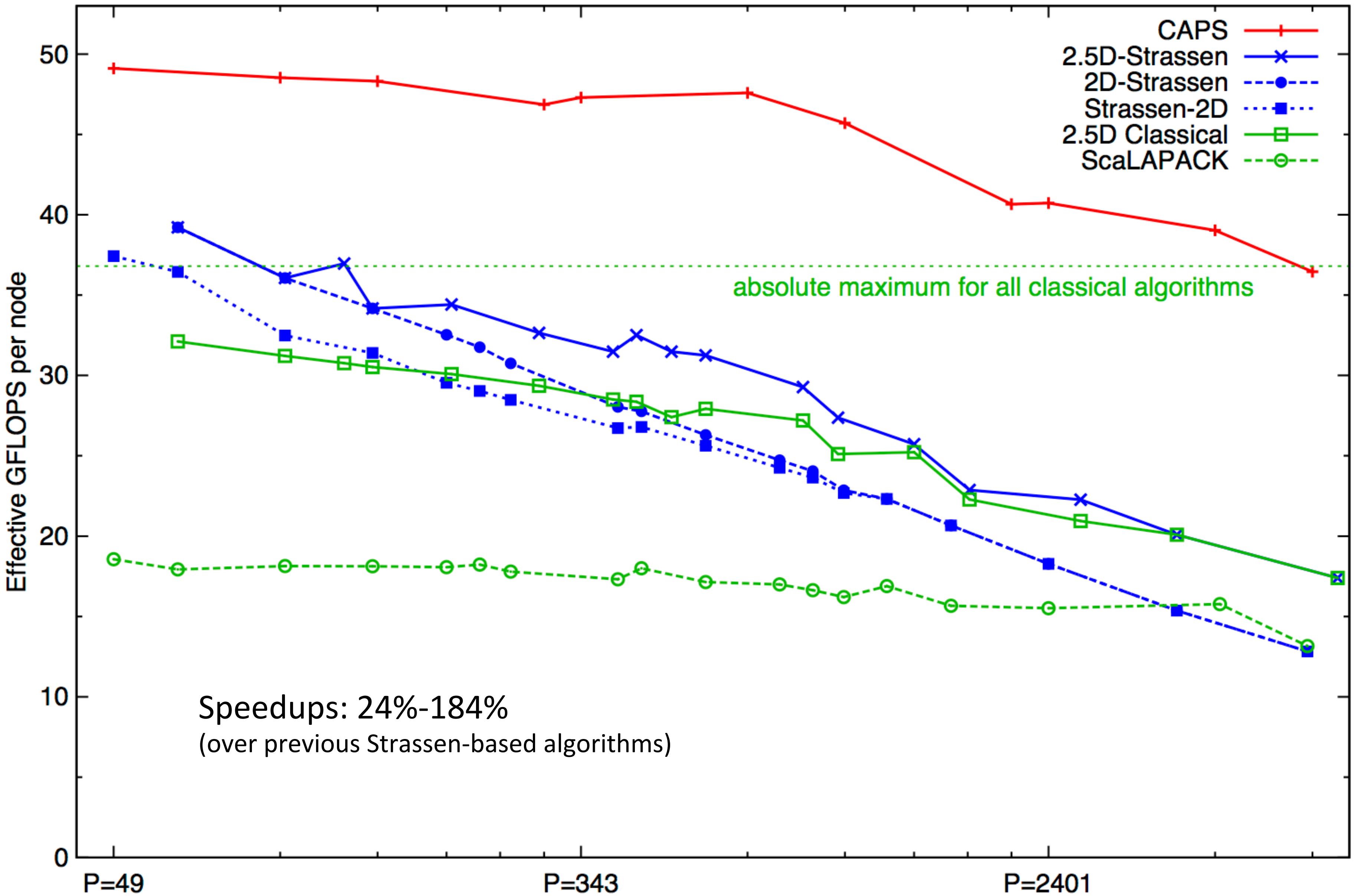
Communication Lower Bounds for Strassen-like matmul algorithms

Classical $O(n^3)$ matmul:	Strassen's $O(n^{\lg 7})$ matmul:	Strassen-like $O(n^{\omega})$ matmul:
#words_moved = $\Omega(M(n/M^{1/2})^3/P)$	#words_moved = $\Omega(M(n/M^{1/2})^{\lg 7}/P)$	#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, also in JACM
- Is the lower bound attainable?

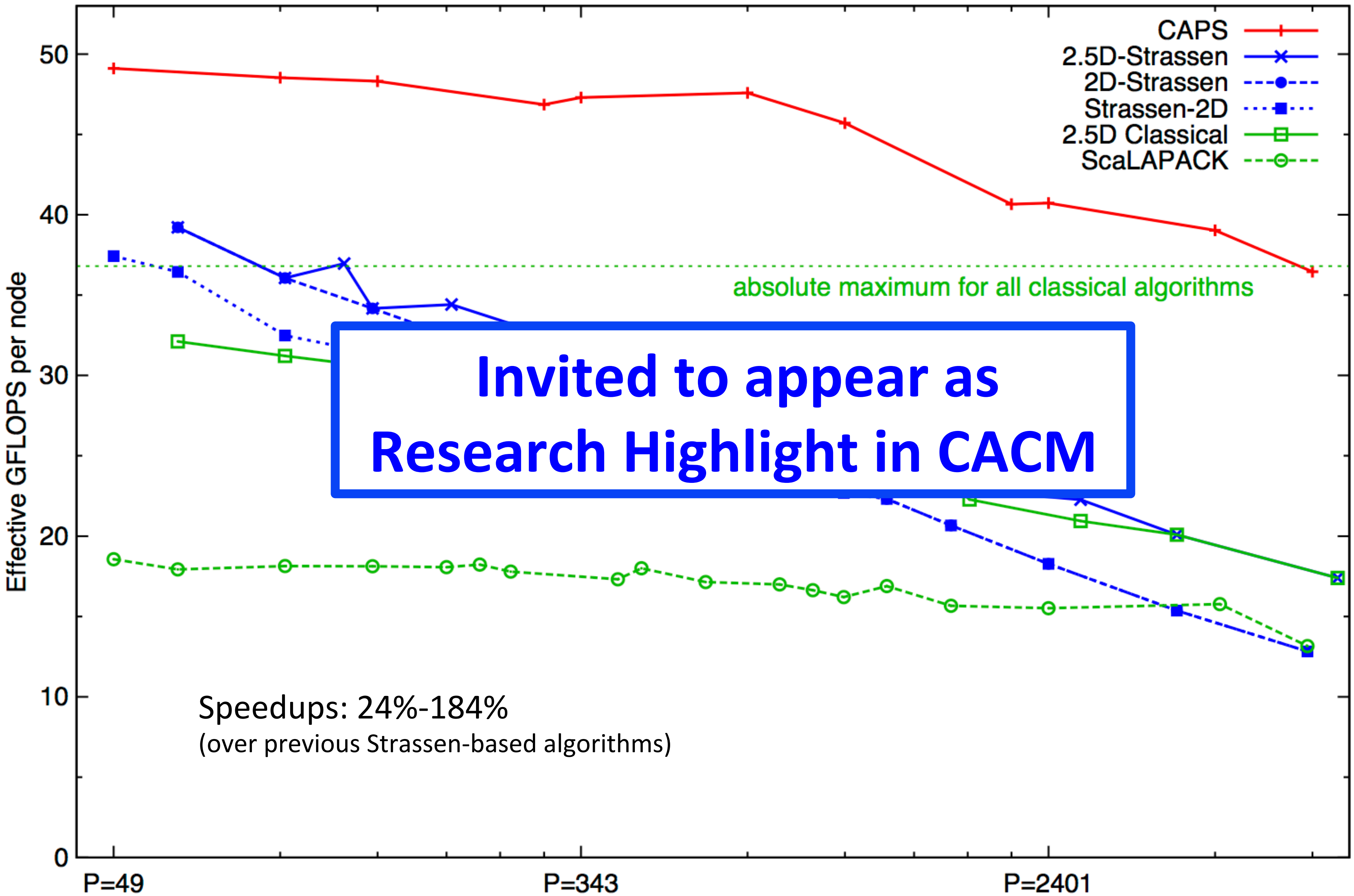
Performance Benchmarking, Strong Scaling Plot

Franklin (Cray XT4) n = 94080



Performance Benchmarking, Strong Scaling Plot

Franklin (Cray XT4) n = 94080



Invited to appear as Research Highlight in CACM

Speedups: 24%-184%

(over previous Strassen-based algorithms)

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

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)
```

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

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?

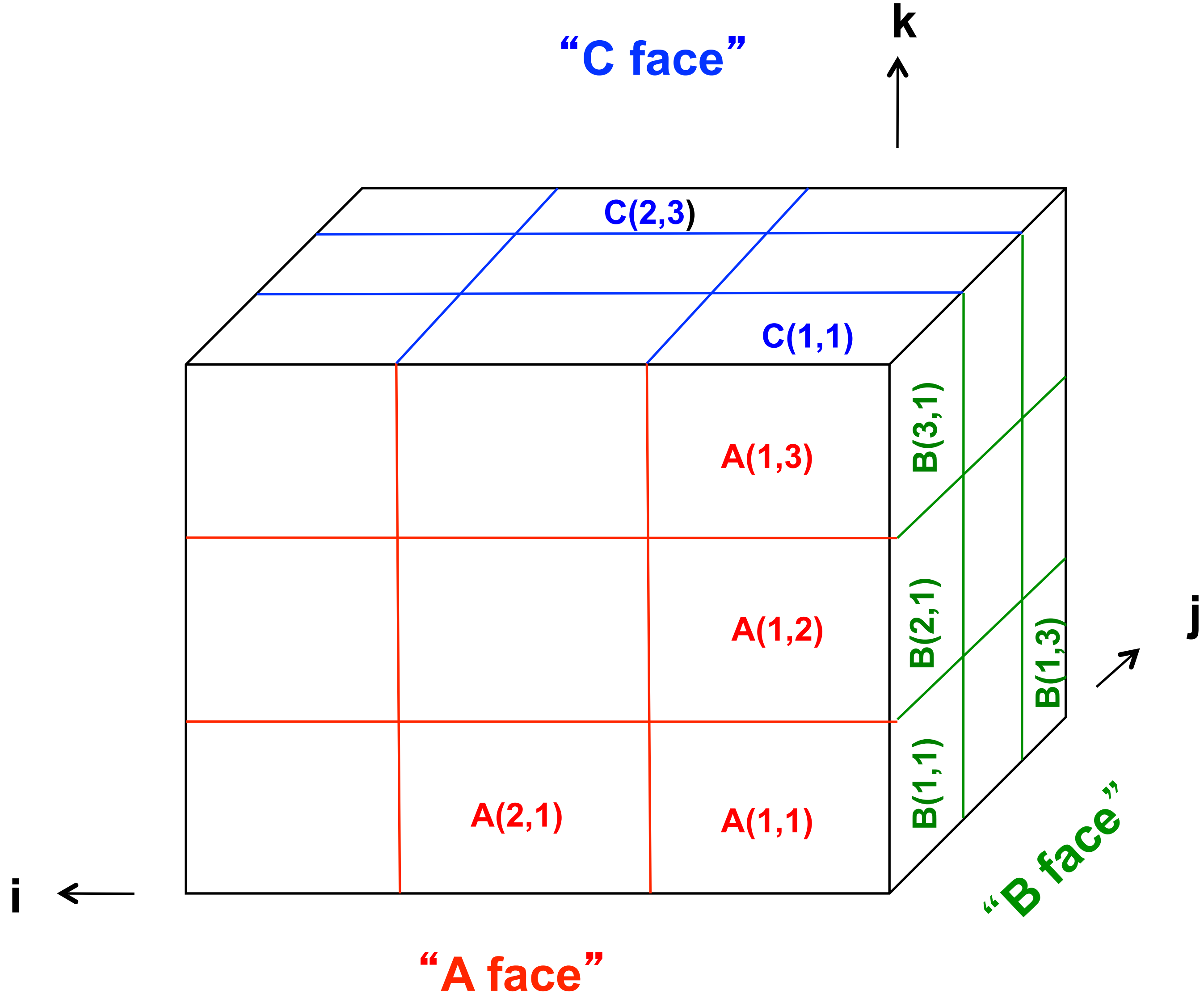
Where do lower and matching upper bounds on communication come from? (1/3)

- Originally for C = A*B by Irony/Tiskin/Toledo (2004)
- Proof idea
  - Suppose we can upper bound #operations doable with data in fast memory of size M, by #operations ≤ G
  - So to do F = #total_operations, need to fill fast memory at least F/G times, and so #words_moved ≥ MF/G
- Hard part: finding G

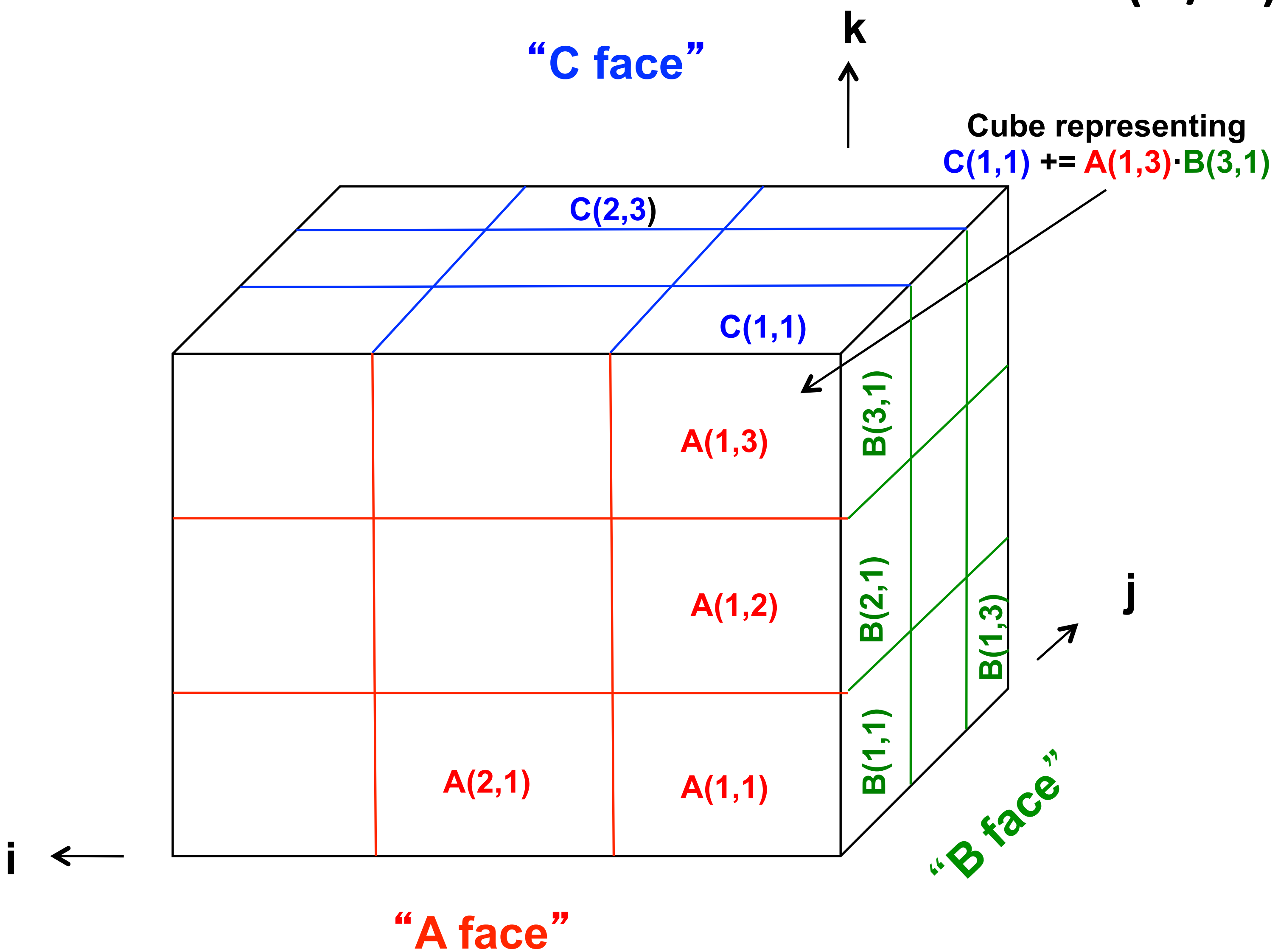
Where do lower and matching upper bounds on communication come from? (1/3)

- Originally for C = A*B by Irony/Tiskin/Toledo (2004)
- Proof idea
  - Suppose we can upper bound #operations doable with data in fast memory of size M, by #operations ≤ G
  - So to do F = #total_operations, need to fill fast memory at least F/G times, and so #words_moved ≥ MF/G
- Hard part: finding G
- Harder part: Attaining lower bound
  - Need to "block" all operations to perform ~G operations on every chunk of M words of data

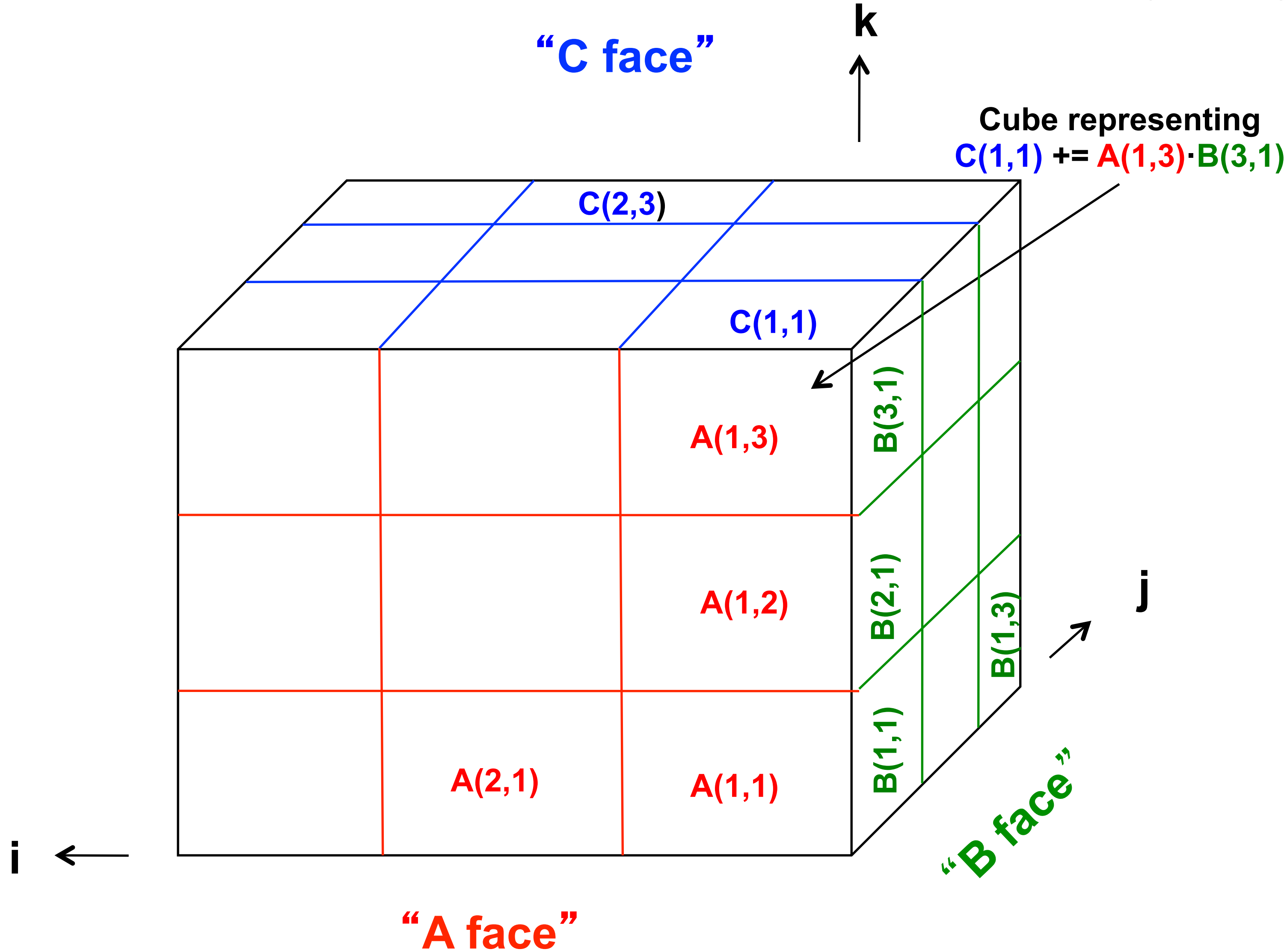
Proof of communication lower bound (2/3)



Proof of communication lower bound (2/3)

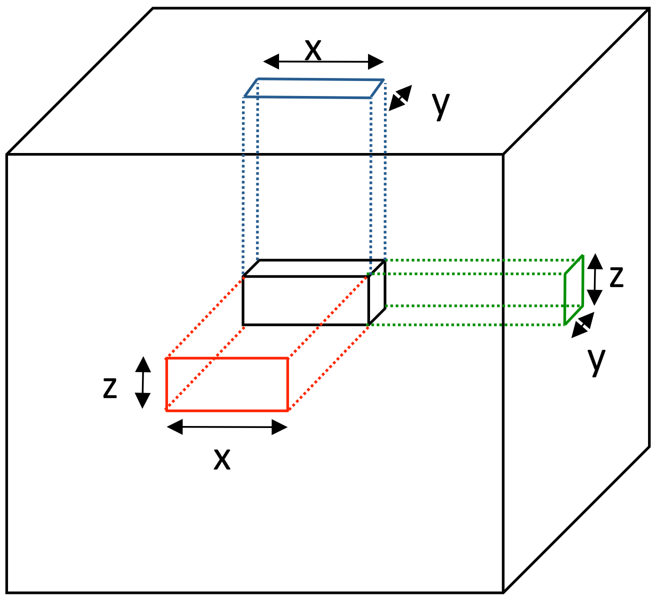


Proof of communication lower bound (2/3)



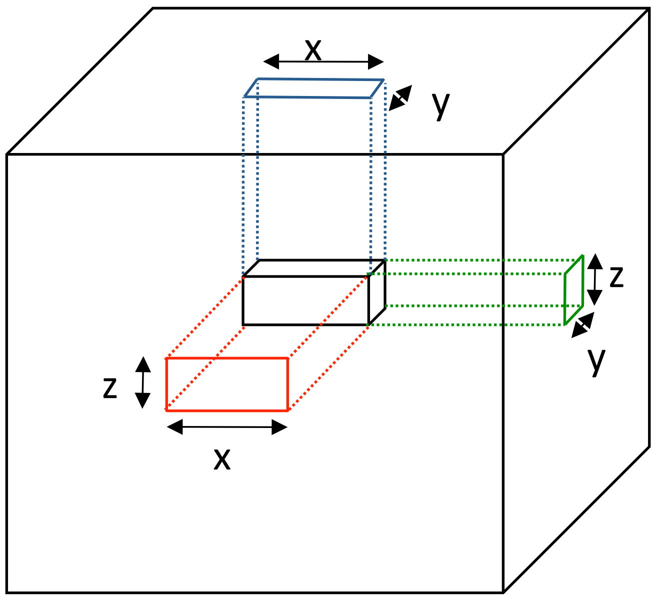
- If we have at most M "A squares", M "B squares", and M "C squares", how many cubes G can we have?

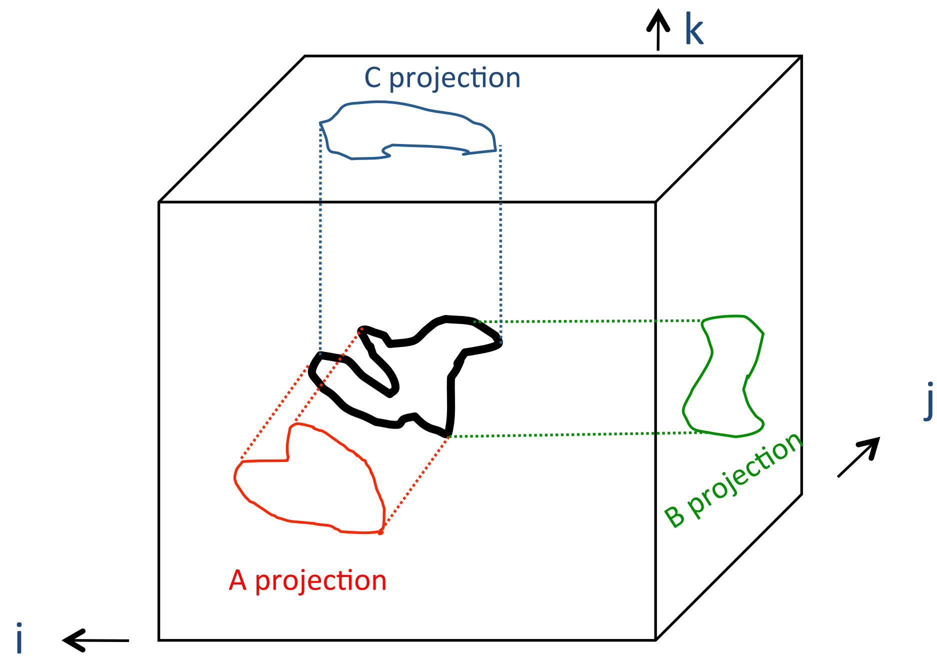
Proof of communication lower bound (3/3)



G = # cubes in black box with
side lengths x, y and z
= Volume of black box
= x·y·z
= (xz · zy · yx)$^{1/2}$
= (#A□s · #B□s · #C□s)$^{1/2}$
≤ M $^{\mathbf{3/2}}$

Proof of communication lower bound (3/3)





G = # cubes in black box with
side lengths x, y and z
= Volume of black box
= x·y·z
= $(xz \cdot zy \cdot yx)^{1/2}$
= (#A□s · #B□s · #C□s $)^{1/2}$
$\leq M^{3/2}$

(i,k) is in A projection if (i,j,k) in 3D set
(j,k) is in B projection if (i,j,k) in 3D set
(i,j) is in C projection if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)
G = # cubes in 3D set = Volume of 3D set
≤ (area(A projection) · area(B projection) ·
area(C projection)) $^{1/2}$
$\leq M^{3/2}$

New theorem, applied to Matmul
(Christ, D., Knight, Scanlon, Yelick)

- 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

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

New theorem, applied to Matmul

(Christ, D., Knight, Scanlon, Yelick)

- 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

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{cccc} i & j & k & \\ \left(\begin{array}{c}1\\0\\1\end{array}\right.\!\!\!\! & \begin{array}{c}0\\1\\1\end{array} & \!\!\!\!\left.\begin{array}{c}1\\1\\0\end{array}\right) & \begin{array}{c}A\\B\\C\end{array} \end{array}$$

- Solve LP for $x = [xi,xj,xk]^T$: max $\mathbf{1}^T x$ s.t. $\Delta\, x \le \mathbf{1}$
  – Result: $x = [1/2,\ 1/2,\ 1/2]^T$, $\mathbf{1}^T x = 3/2 = s_{\mathbf{HBL}}$

New theorem, applied to Matmul

(Christ, D., Knight, Scanlon, Yelick)

- 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

$$\Delta = \begin{array}{c} \\ \\ \\ \end{array}\begin{array}{ccc|c} i & j & k & \\ \end{array}$$

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

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

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

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

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

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{cc} i & j \\ \end{array}$$

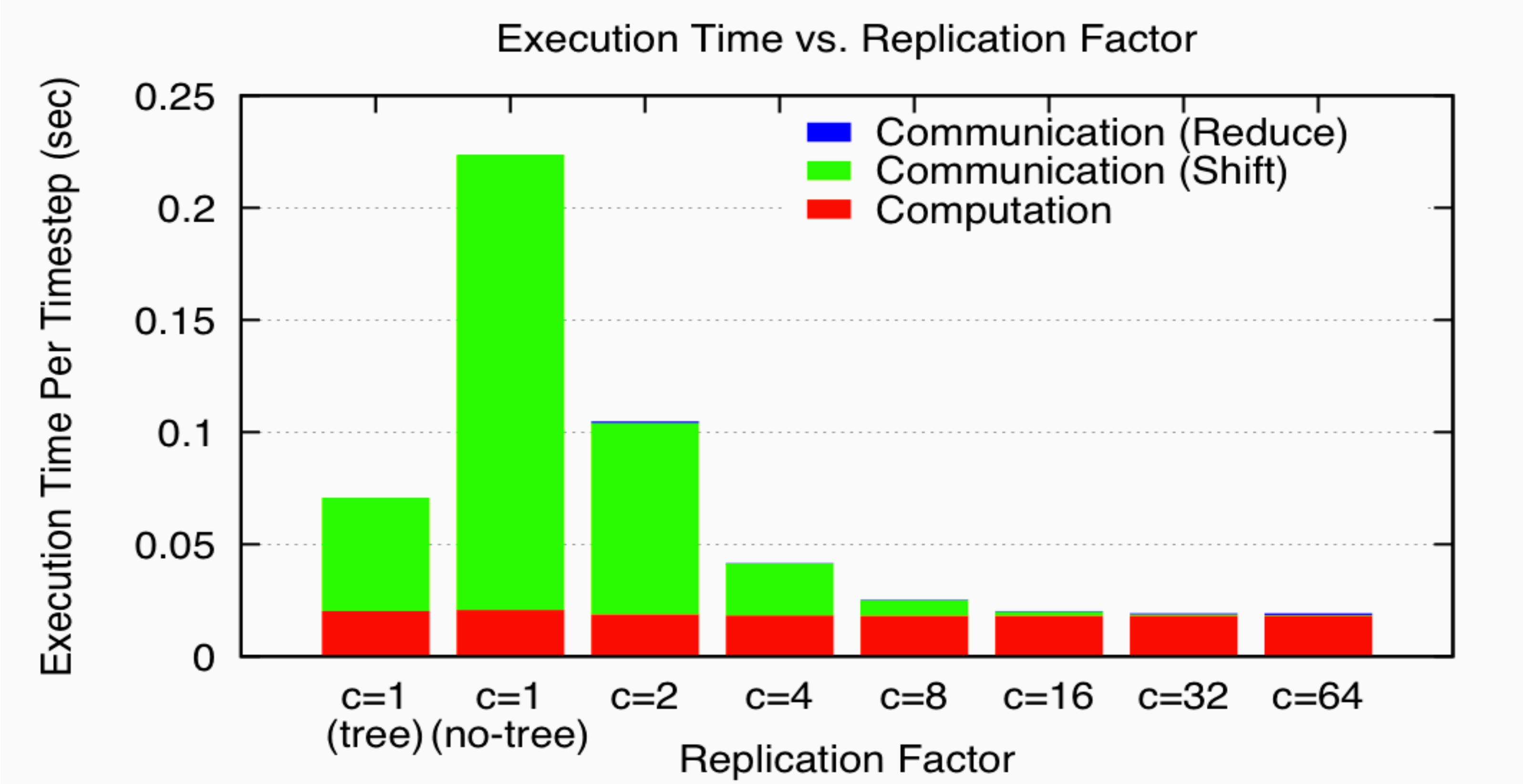
$$\begin{array}{ccc} & i & j & \\ \Delta = & \left(\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{array}\right) & \begin{array}{c} F \\ P(i) \\ P(j) \end{array} \end{array}$$

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

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

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

K. Yelick, E. Georganas, M. Driscoll, P. Koanantakool, E. Solomonik



11.8x speedup

Some Applications

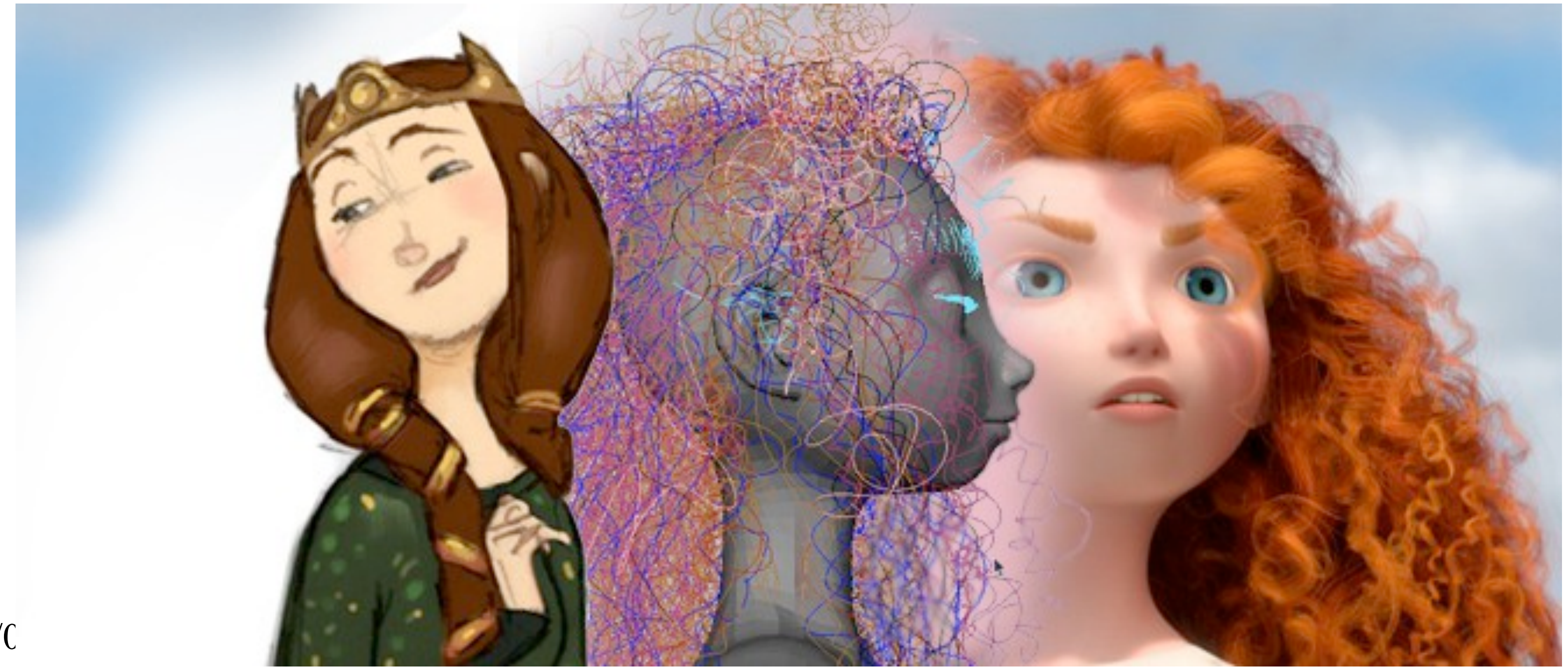
- Gravity, Turbulence, Molecular Dynamics, Plasma Simulation, …

Some Applications

- Gravity, Turbulence, Molecular Dynamics, Plasma Simulation, …
- Electron-Beam Lithography Device Simulation

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



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

$$\Delta = \begin{array}{c} \begin{array}{cccccc} i1 & i2 & i3 & i4 & i5 & i6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ A1 \\ A2 \\ A3 \\ A3,A4 \\ A5 \\ A6 \end{array}$$

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

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccc} i1 & i2 & i3 & i4 & i5 & i6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ A1 \\ A2 \\ A3 \\ A3,A4 \\ A5 \\ A6 \end{array}$$

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

  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

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)

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)

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), …

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 group homomorphisms, eg

  ϕ_C (i1,i2,…,ik) = (i1+2*i3-i7)

  ϕ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

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 group homomorphisms, 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, optimal algorithms for *any* program that looks like this

General Communication Bound

- Given subset of loop iterations, how much data do we need?

General Communication Bound

- Given subset of loop iterations, how much data do we need?
  - Given subset S of Z^k, group homomorphisms ϕ_1, ϕ_2, …, ϕ_m bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, … , $|\phi_m(S)|$

General Communication Bound

- Given subset of loop iterations, how much data do we need?
  - Given subset S of Z^k, group homomorphisms $\phi_1, \phi_2, \ldots, \phi_m$ bound $|S|$ in terms of $|\phi_1(S)|, |\phi_2(S)|, \ldots, |\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1,\ldots,s_m$:

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

General Communication Bound

- Given subset of loop iterations, how much data do we need?
  - Given subset S of Z^k, group homomorphisms $\phi_1, \phi_2, \ldots, \phi_m$ bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, … , $|\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1,\ldots,s_m$:

  for all subgroups $H < Z^k$, $\quad \mathrm{rank}(H) \le \Sigma_j\, s_j * \mathrm{rank}(\phi_j(H))$
- Thm (extension of Christ/Tao/Carbery/Bennett): Given $s_1,\ldots,s_m$

$$|S| \le \Pi_j\, |\phi_j(S)|^{s_j}$$

General Communication Bound

- Given subset of loop iterations, how much data do we need?
  - Given subset S of Z^k, group homomorphisms $\phi_1, \phi_2, \ldots, \phi_m$ bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, … , $|\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1,\ldots,s_m$:

  for all subgroups $H < Z^k$, $\quad \text{rank}(H) \le \Sigma_j\, s_j * \text{rank}(\phi_j(H))$
- Thm (extension of Christ/Tao/Carbery/Bennett): Given $s_1,\ldots,s_m$

  $$|S| \le \Pi_j\, |\phi_j(S)|^{s_j}$$
- Thm: Given a program with array refs given by ϕ_j, choose s_j to minimize $s_{HBL} = \Sigma_j\, s_j$ subject to HBL-LP. Then

  $$\#\text{words\_moved} = \Omega\,(\#\text{iterations}/M^{s_{HBL}-1})$$

Is this bound attainable (1/2)?

Is this bound attainable (1/2)?

- But first: Can we write it down?

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q
  - conjectured to be undecidable

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q
  - conjectured to be undecidable
- Thm: (good news) Another LP with same solution is decidable (but expensive):

```
Let L = (V1,V2,…) be countable list of all subspaces of Q^n
i=0
repeat
    i = i + 1
until  polytope determined by inequalities from (V1,…,Vi)  is right one
```

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q
  - conjectured to be undecidable
- Thm: (good news) Another LP with same solution is decidable (but expensive):

  Let L = (V1,V2,…) be countable list of all subspaces of Q^n
  i=0
  repeat
      i = i + 1
  until polytope determined by inequalities from (V1,…,Vi) is right one

- Thm: (better news) Enough to use L = lattice of kernels of $\phi_1, \phi_2, \ldots, \phi_m$
  - Similar to result of Valdimarsson in continuum case

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10th problem over Q
  - conjectured to be undecidable
- Thm: (good news) Another LP with same solution is decidable (but expensive):

```
Let L = (V1,V2,…) be countable list of all subspaces of Q^n
i=0
repeat
    i = i + 1
until  polytope determined by inequalities from (V1,…,Vi)  is right one
```

- Thm: (better news) Enough to use L = lattice of kernels of $\phi_1, \phi_2, \ldots, \phi_m$
  - Similar to result of Valdimarsson in continuum case
  - Corollary (Dedekind) If m=3, only need to consider 28 subspaces

Is this bound attainable (2/2)?

- Given bound, need to reorder loop iterations, or assign them to different processors, to maximize |S| = #_loop_iterations using data that fits in memory: $|\phi_1(S)| + |\phi_2(S)| + \ldots + |\phi_m(S)| \le M$
- Assume best case: can execute iterations in any order
  - Ex: matmul, because just summing
- Thm: When all $\phi_{\mathbf{k}}$ = {subset of indices}, *dual* of HBL-LP gives optimal tile sizes:

  HBL-LP: minimize $1^T * s$ s.t. $s^T * \Delta \ge 1^T$, $\Delta(k,j) = \mathrm{rank}(\phi_{\mathbf{k}}(<i_{\mathbf{j}}>))$

  Dual-HBL-LP: maximize $1^T * x$ s.t. $\Delta * x \le 1$

  Then for sequential algorithm, tile $i_{\mathbf{j}}$ by M^{x_j}
- Ex: Matmul: $s = [\ 1/2\ ,\ 1/2\ ,\ 1/2\ ]^T = x$
- Ex: N-body, “random code”
- Extends to case where HBL-LP determined by independent groups
  - Tiling code means tesselating Z^k with polytopes

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms
- Have yet to find a case where we cannot attain lower bound – can we prove this?

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms
- Have yet to find a case where we cannot attain lower bound – can we prove this?
- Hardest, practical case: Loop-carried dependencies
  - Ex: C(i,j) = func (C(i,j), A(i,k), B(k,j), C(i-1,j))
  - Only some reorderings/tesselations correct
  - How close can we get to "optimal"?

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms
- Have yet to find a case where we cannot attain lower bound – can we prove this?
- Hardest, practical case: Loop-carried dependencies
  - Ex: C(i,j) = func (C(i,j), A(i,k), B(k,j), C(i-1,j))
  - Only some reorderings/tesselations correct
  - How close can we get to "optimal"?
- Extend "perfect scaling" results for time and energy by using extra memory
  - "n.5D algorithms"

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms
- Have yet to find a case where we cannot attain lower bound – can we prove this?
- Hardest, practical case: Loop-carried dependencies
  - Ex: C(i,j) = func (C(i,j), A(i,k), B(k,j), C(i-1,j))
  - Only some reorderings/tesselations correct
  - How close can we get to "optimal"?
- Extend "perfect scaling" results for time and energy by using extra memory
  - "n.5D algorithms"
- Incorporate into compilers

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

Avoiding Communication in Iterative Linear Algebra

- k-steps of iterative solver for sparse $Ax=b$ or $Ax=\lambda x$
  - Does k SpMVs with A and starting vector
  - Many such "Krylov Subspace Methods"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …

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"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- Goal: minimize communication
  - Assume matrix "well-partitioned"

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"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- 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**

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"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- 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**

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"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- 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
  - Challenges: Poor partitioning, Preconditioning, Num. Stability

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Lower bound proof for linear algebra
  - Extending lower bounds to "any algorithm with arrays"
  - Progress toward optimal algorithms
- CA-Krylov methods
- Conclusions

For more details

- Bebop.cs.berkeley.edu
  - 155 page survey in Acta Numerica
- CS267 – Berkeley's Parallel Computing Course
  - Live broadcast in Spring 2015
    - www.cs.berkeley.edu/~demmel
    - All slides, video available
  - Prerecorded version broadcast in Spring 2014/5
    - 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, 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 FASTMath, ParLab, ASPIRE, BEBOP, CACHE, EASI, 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)

Summary

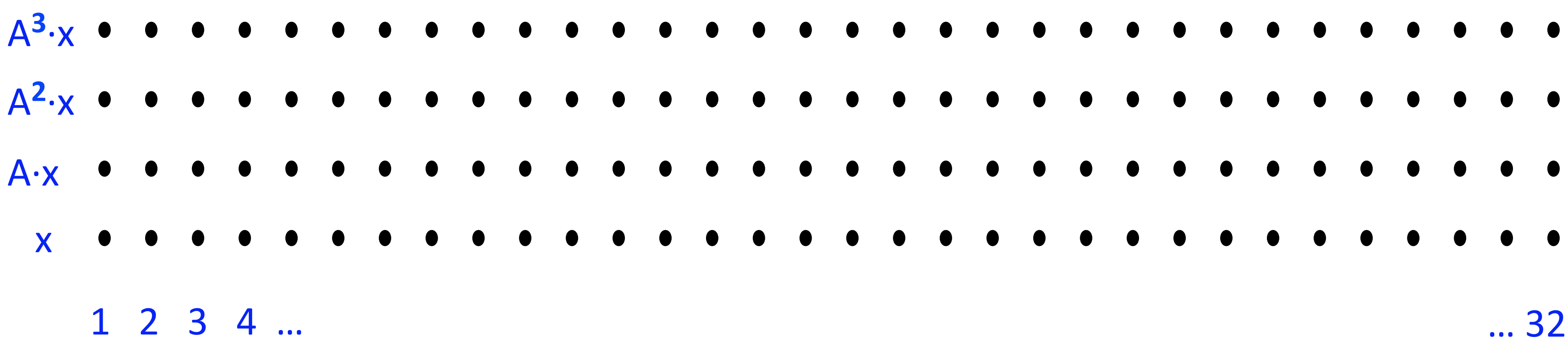
Time to redesign all linear algebra, n-body, …
algorithms and software
(and compilers)

Don't Communic…

EXTRA SLIDES

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

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

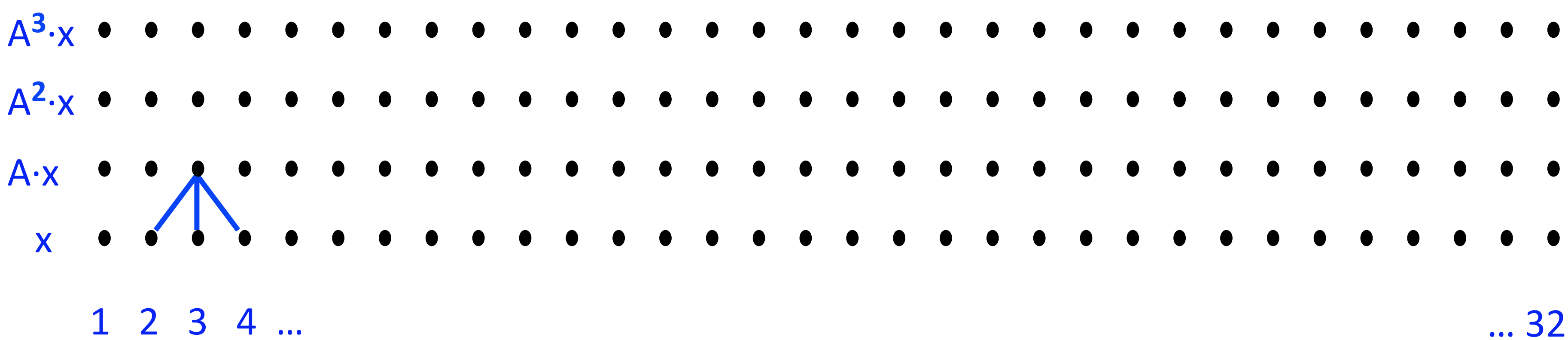


- Example: A tridiagonal, n=32, k=3
- Works for any “well-partitioned” A

Communication Avoiding Kernels:

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

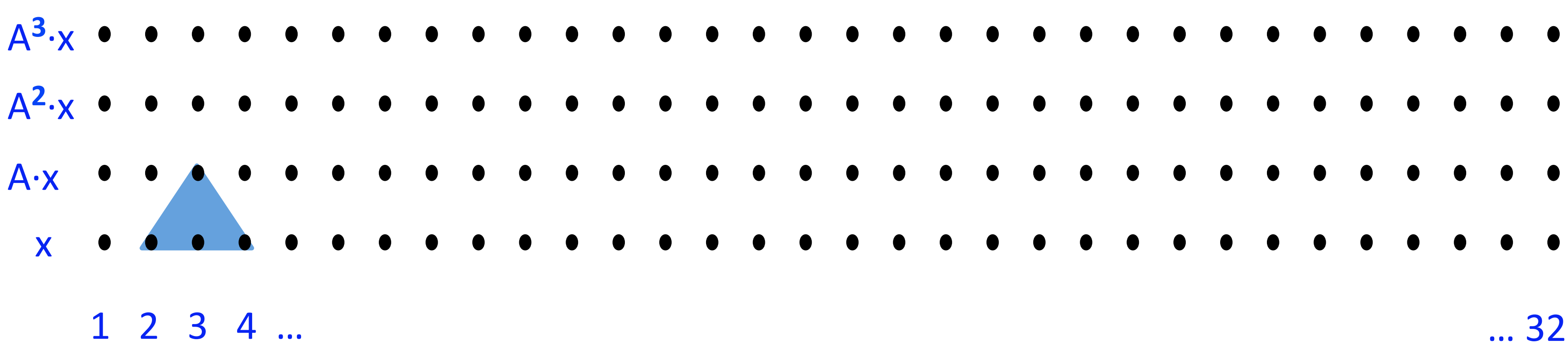
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



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

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

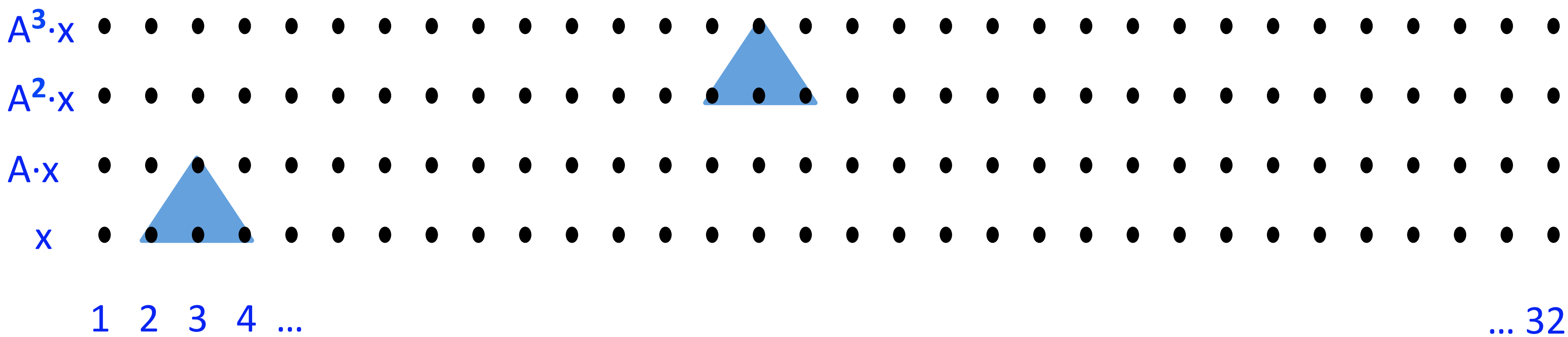
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



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

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

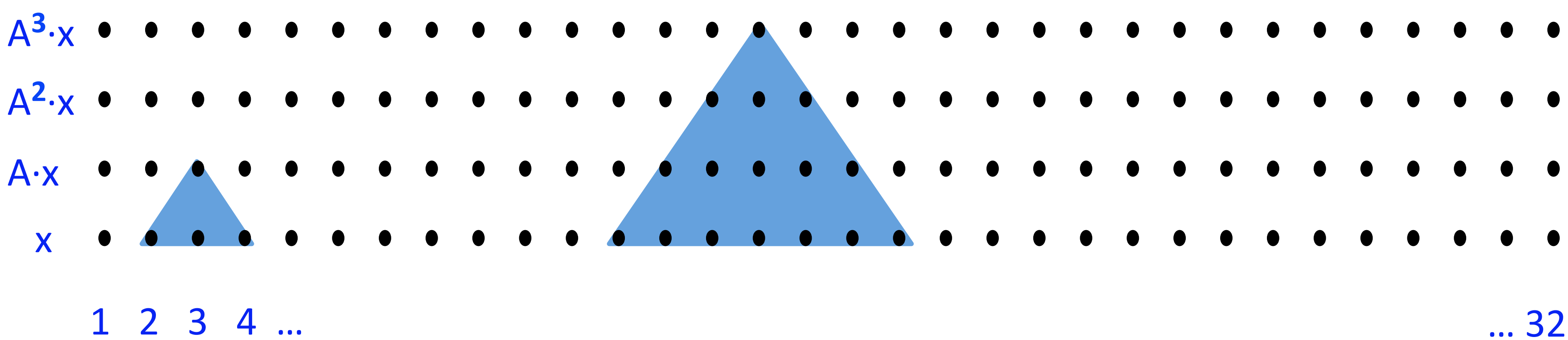
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



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

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

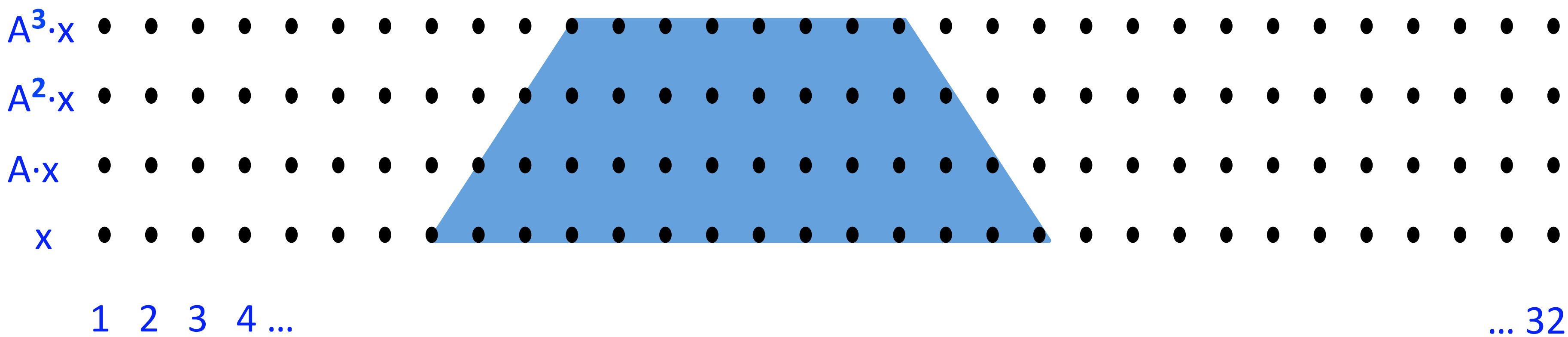
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



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

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

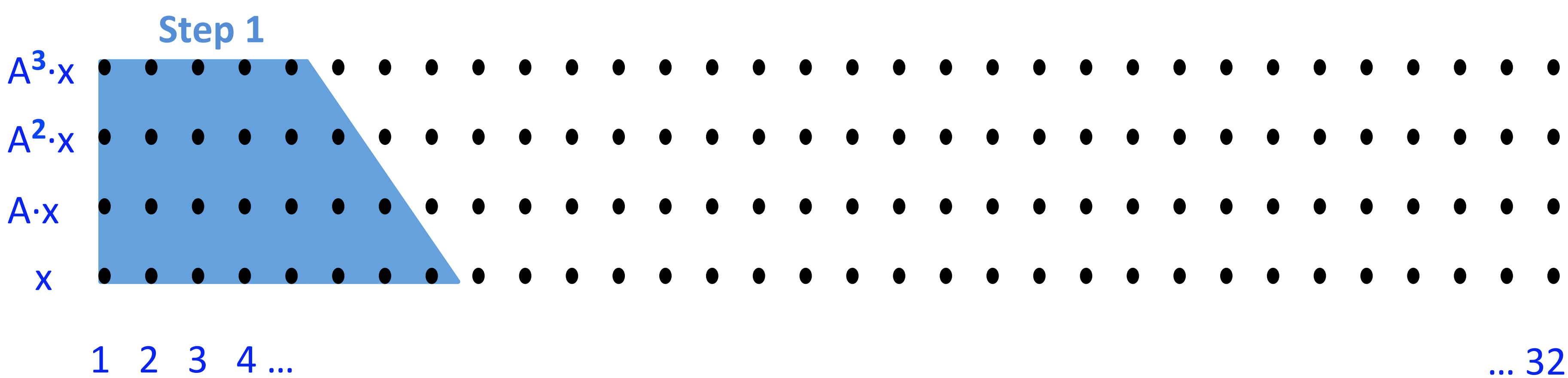
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

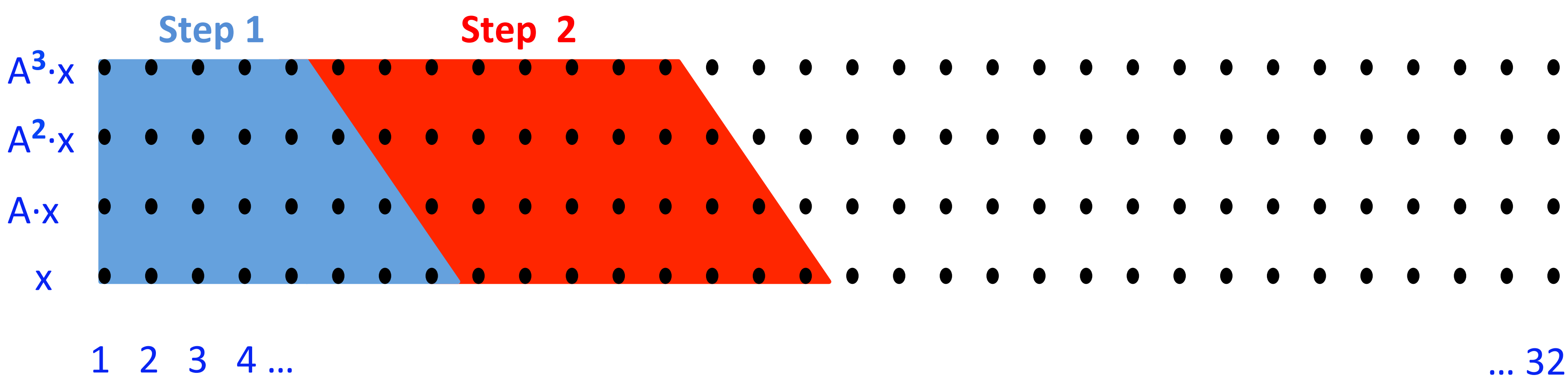
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



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

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

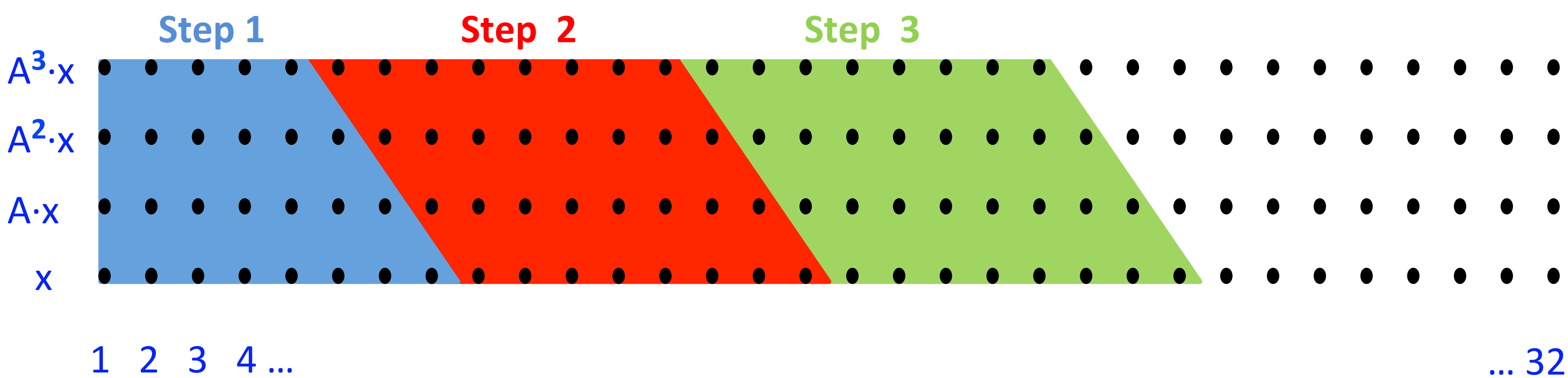
- Replace k iterations of $y = A\cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



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

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

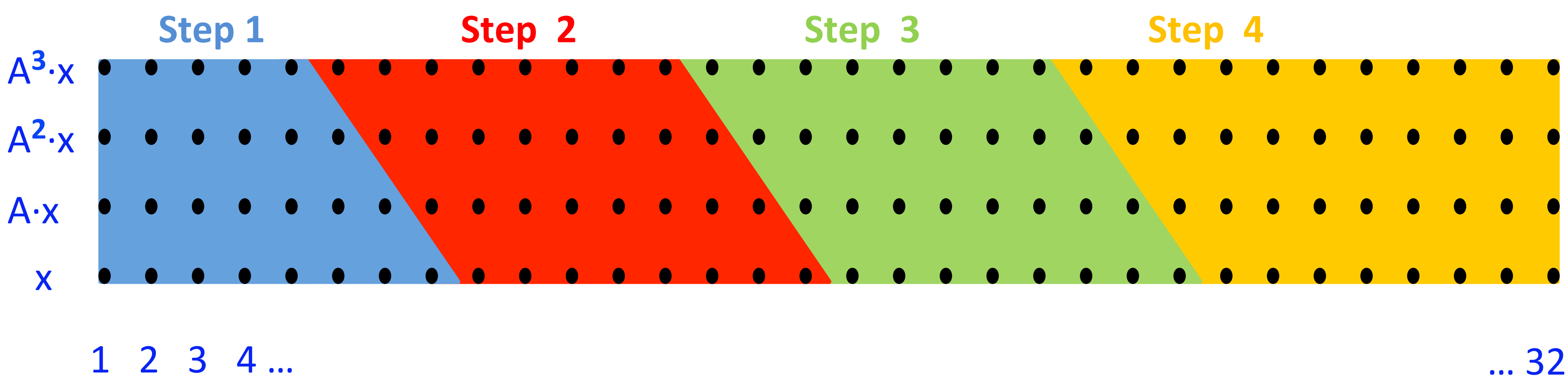
- Replace k iterations of $y = A\cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



- 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

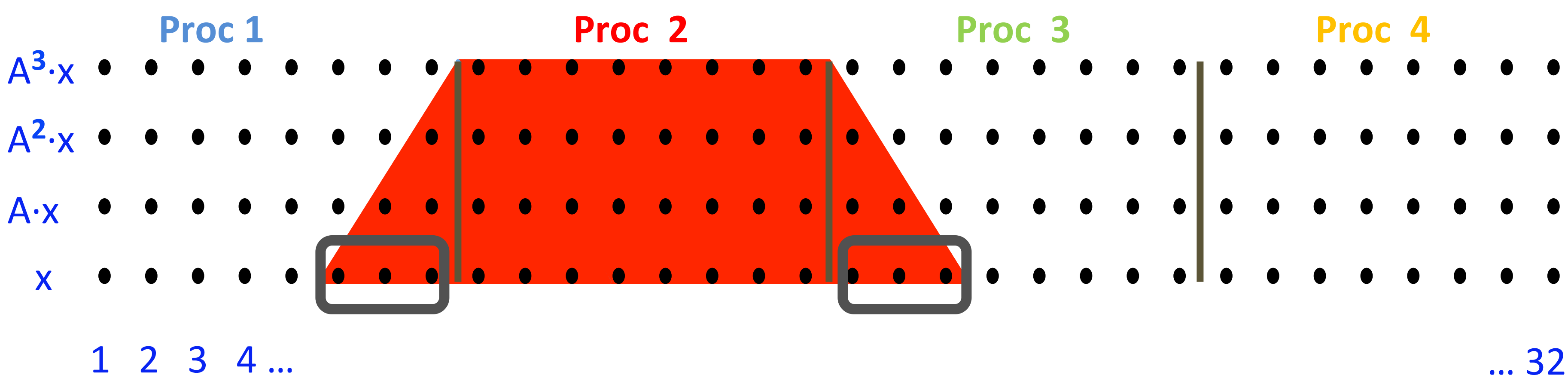


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

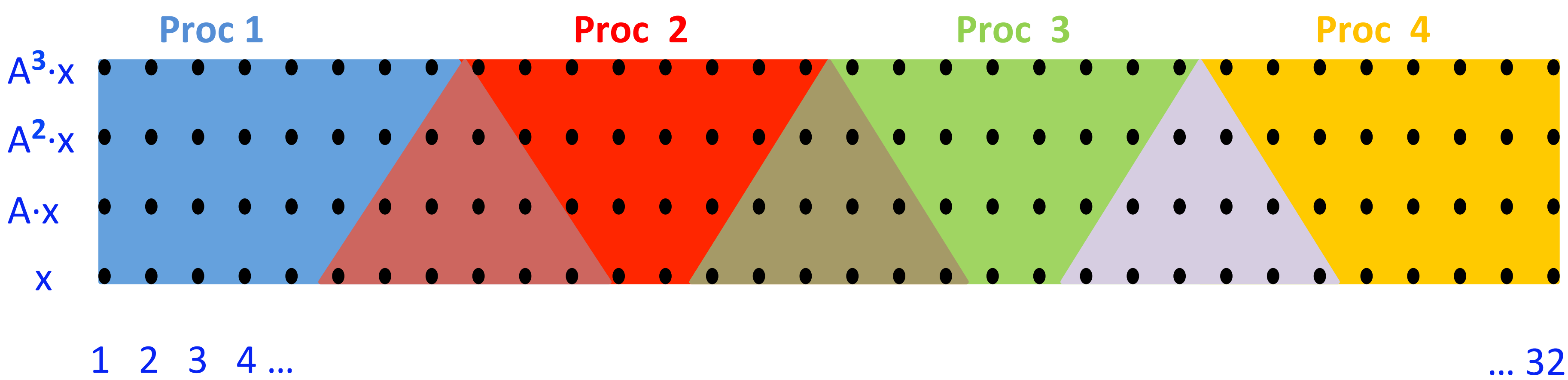


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

Communication Avoiding Kernels:

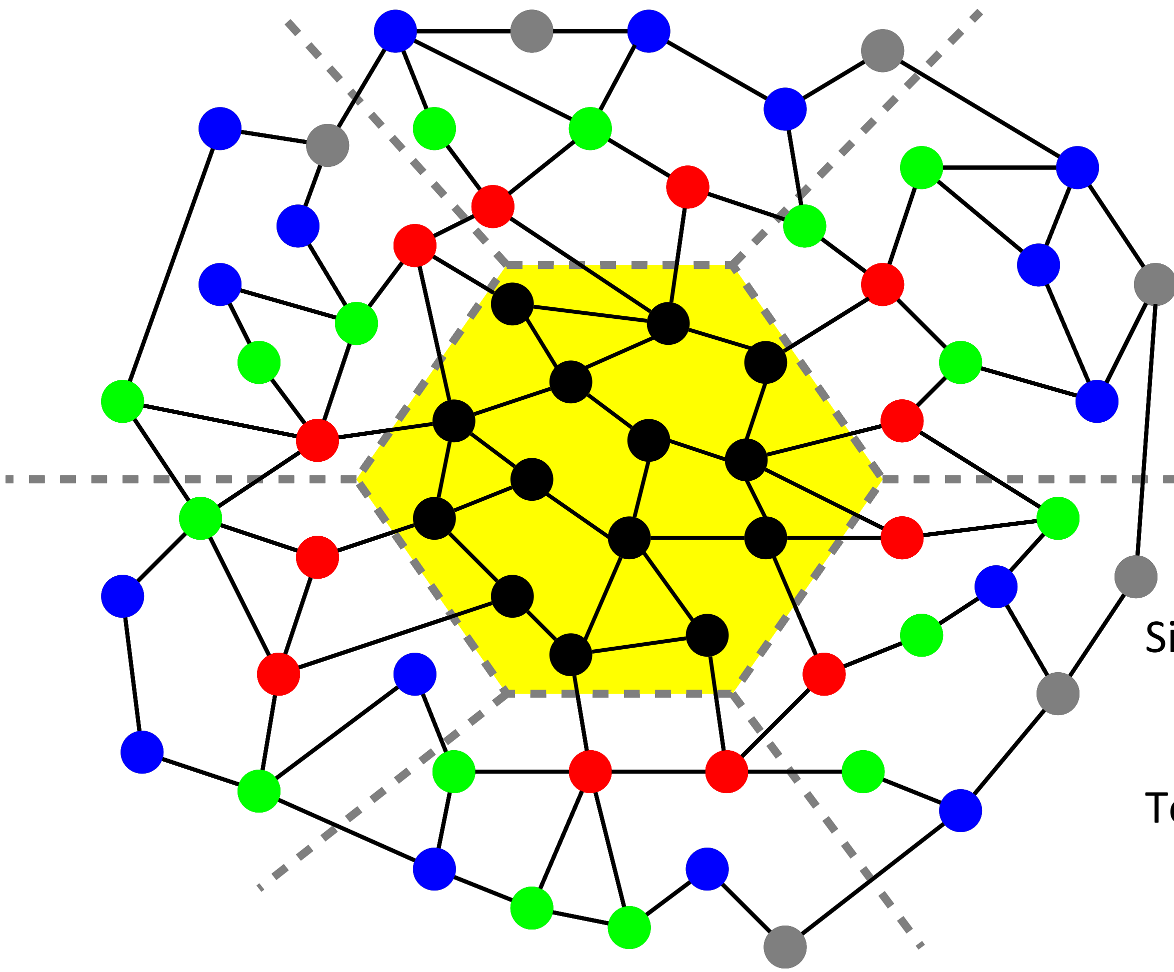
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

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



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

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$ on a general matrix (nearest k neighbors on a graph)



Simple block-row partitioning → (hyper)graph partitioning

Top-to-bottom processing → Traveling Salesman Problem

Same idea for general sparse matrices: k-wide neighboring region

Minimizing Communication of GMRES to solve Ax=b

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

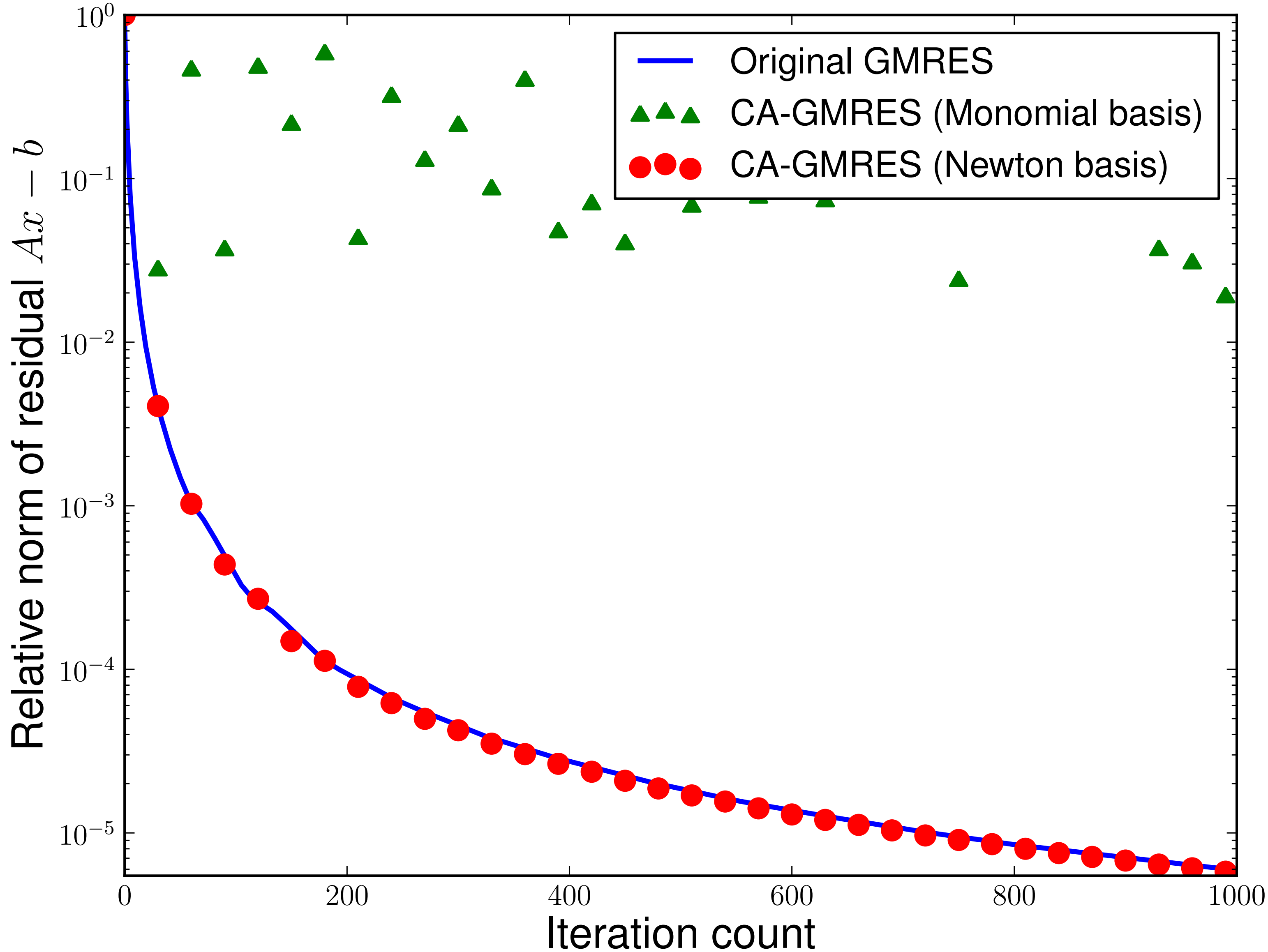
Standard GMRES
  for i=1 to k
    w = A · v(i-1)   *… SpMV*
    MGS(w, v(0),…,v(i-1))
    update v(i), H
  endfor
  solve LSQ problem with H

Communication-avoiding GMRES
  W = [v, Av, A^2v, … , 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!

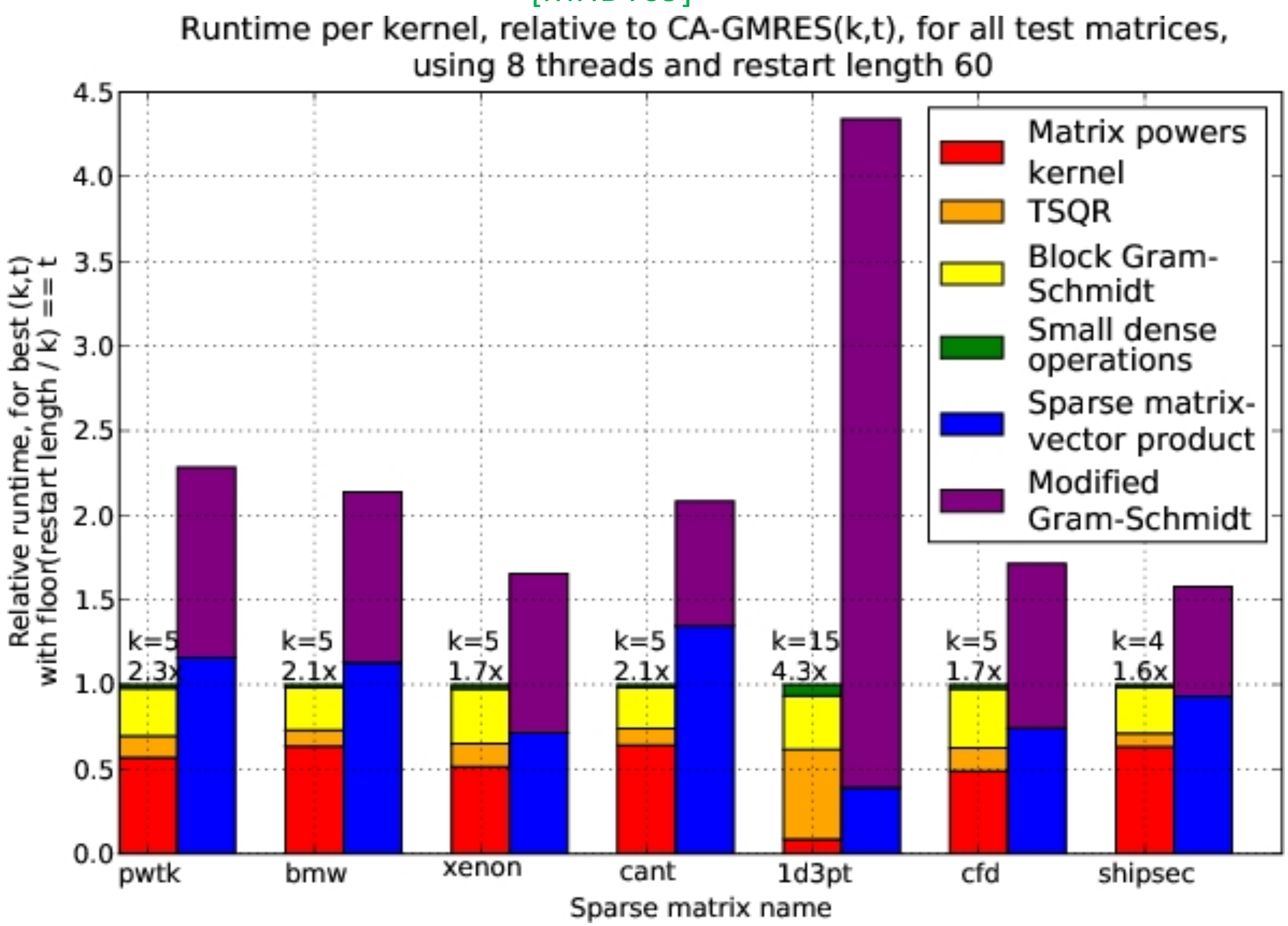
Matrix Powers Kernel + TSQR in GMRES



Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



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 $k = 0, 1, \ldots,$ until convergence, Do

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

//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

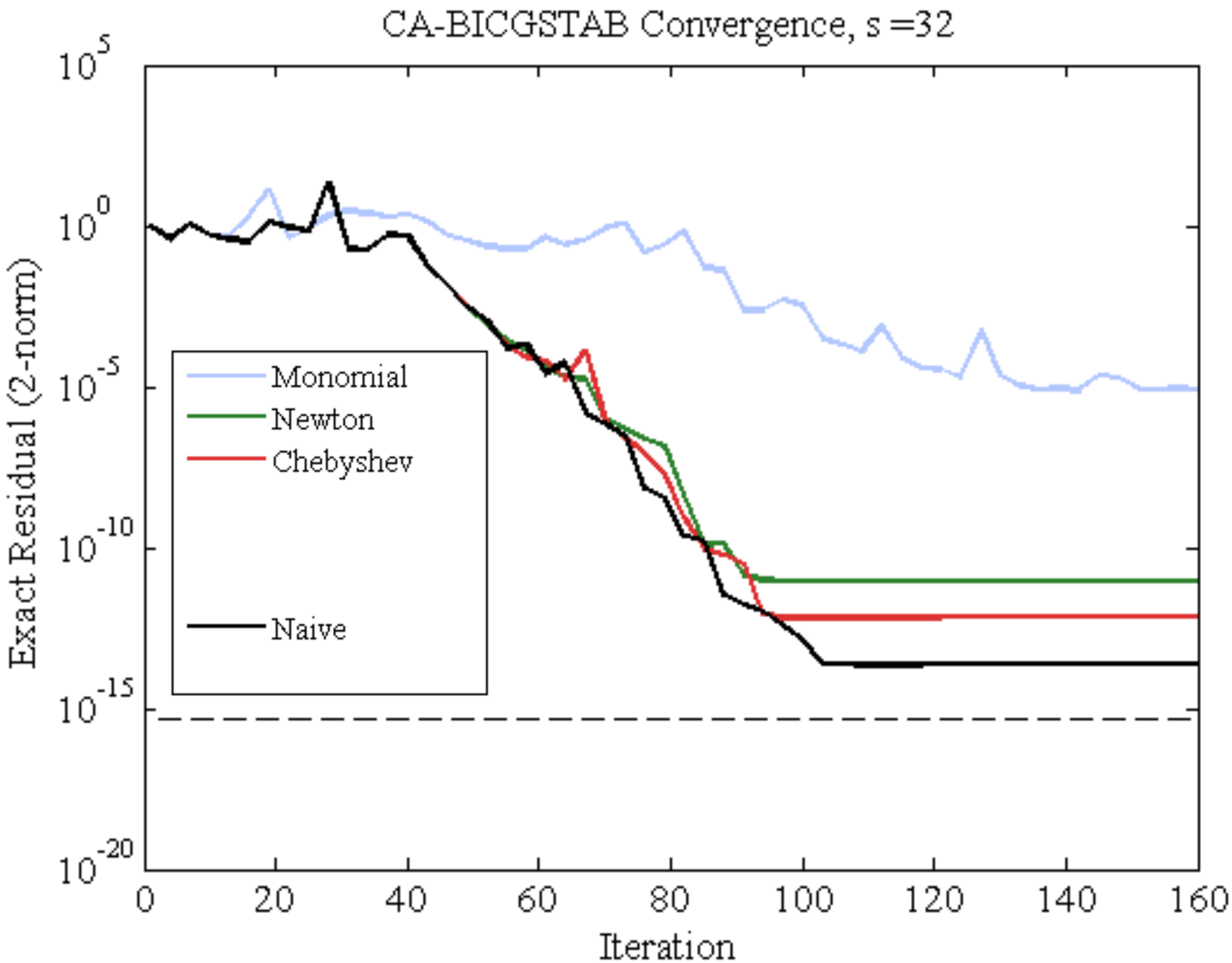
$$b_{sk+j+1}^{\ell} = d_{sk+j+1}^{\ell} + \beta_{sk+j} b_{sk+j}^{\ell} - \beta_{sk+j}\omega_{sk+j} b_{sk+j}^{\ell+1}$$

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

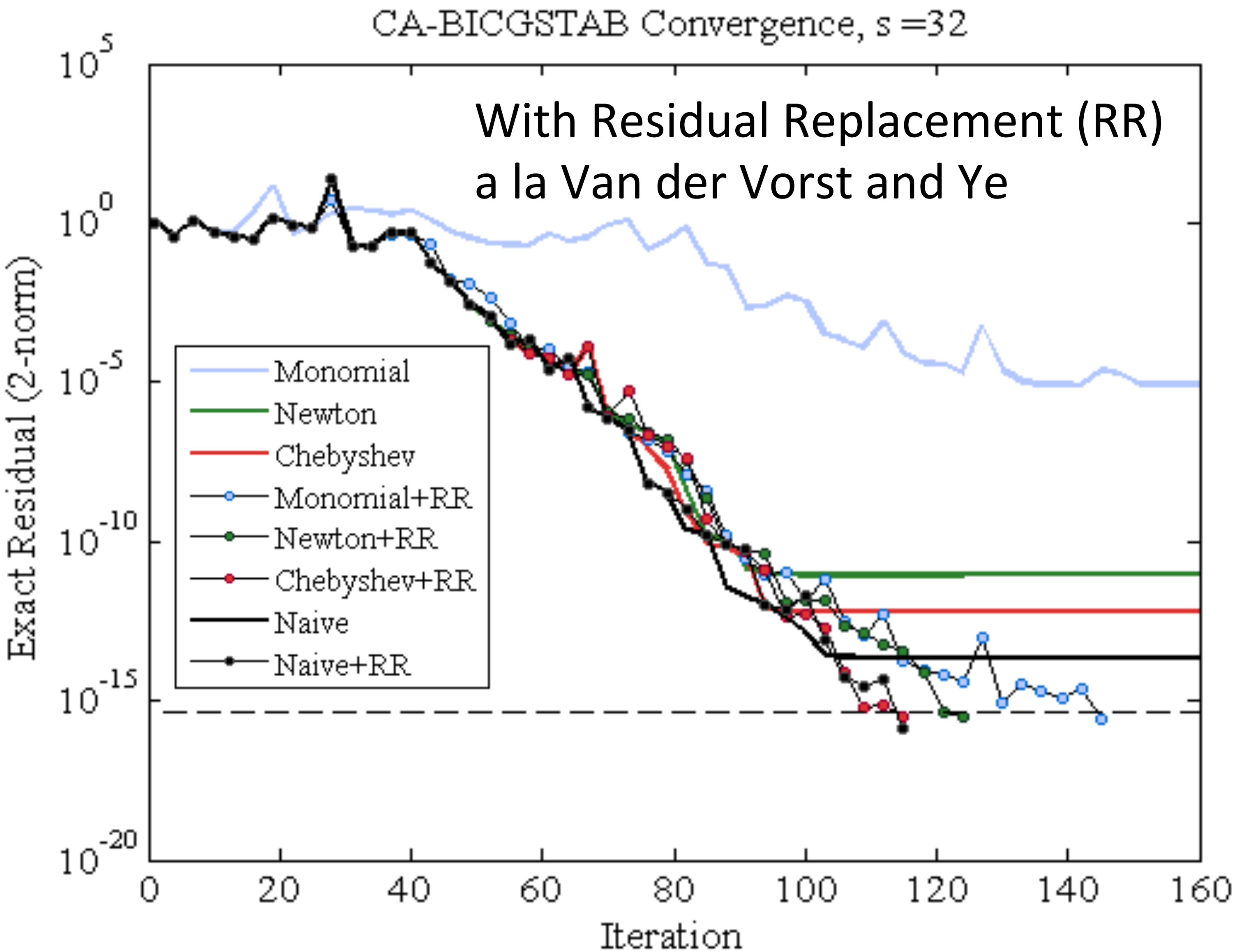
EndDo

EndDo

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



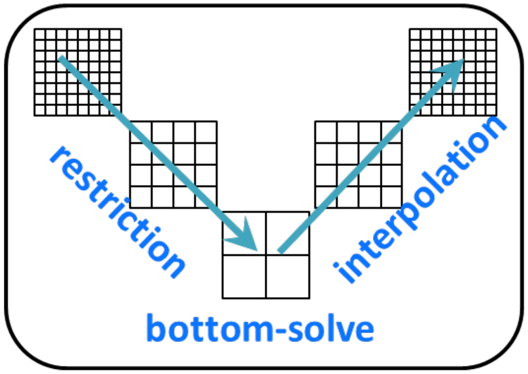
CA-BICGSTAB Convergence, s =32
Exact Residual (2-norm)
Iteration
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
0
20
40
60
80
100
120
140
160
Monomial
Newton
Chebyshev
Naive



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

Speedups for GMG w/CA-KSM Bottom Solve

- Compared **BICGSTAB** vs. **CA-BICGSTAB** with s = 4 (monomial basis)
- Hopper at NERSC (Cray XE6), weak scaling: Up to 4096 MPI processes (1 per chip, 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
- Speedups for two BoxLib applications:
  - 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

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
    - Hierarchically Semiseparable Matrices
  - Autotuning and synthesis
    - Different kinds of "sparse matrices"

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm
- CA-Krylov methods
- Related Topics
  - Write-Avoiding Algorithms
  - Reproducilibity

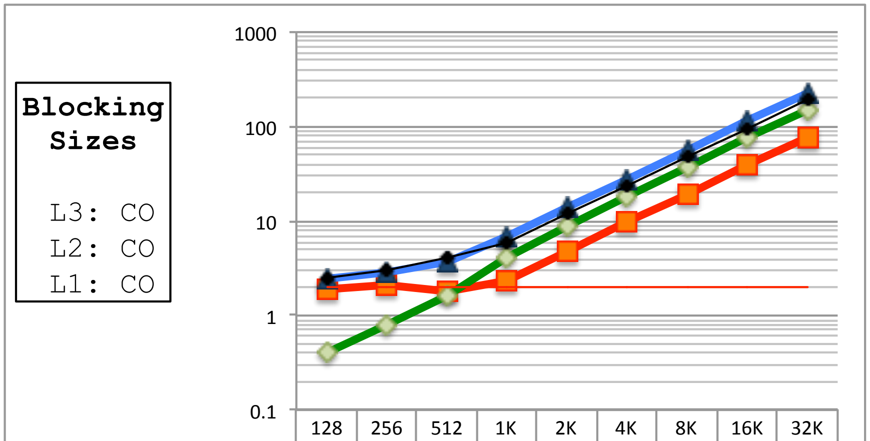
Write-Avoiding Algorithms

- 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

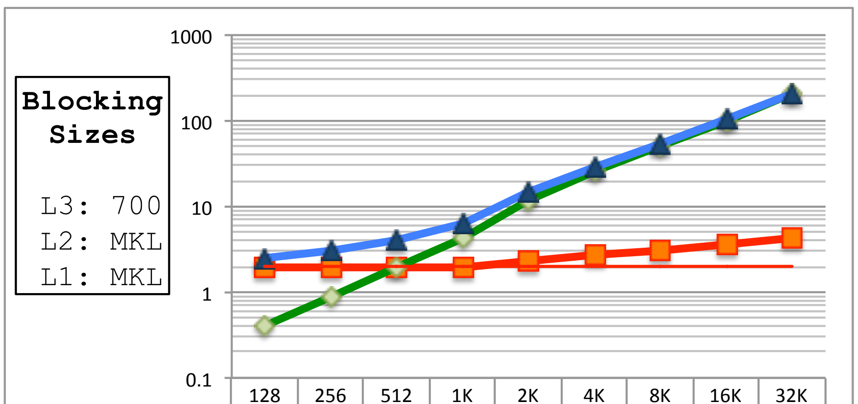


	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	1.9	2.1	1.8	2.3	4.8	9.8	19.5	39.6	78.5
L3_VICTIMS.E	0.4	0.8	1.6	4.2	8.8	17.9	36.6	75.4	147.5
LLC_S_FILLS.E	2.4	2.8	3.8	6.9	14	28.1	56.5	115.5	226.6
Misses on Ideal Cache	2.512	3.024	4.048	6	12	24	48	96	192
Write L.B.	2	2	2	2	2	2	2	2	2

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

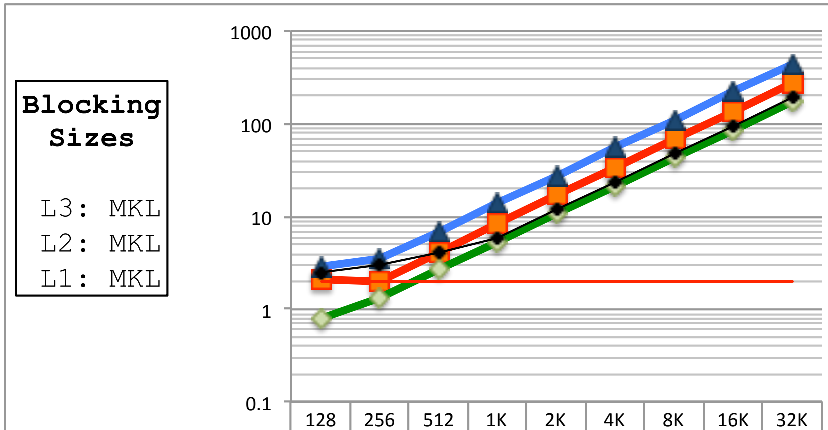
Total L3 misses close to optimal

Total DRAM writes much larger

Measured L3-DRAM traffic on Intel Nehalem Xeon-7560

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

Optimal #DRAM writes = n^2

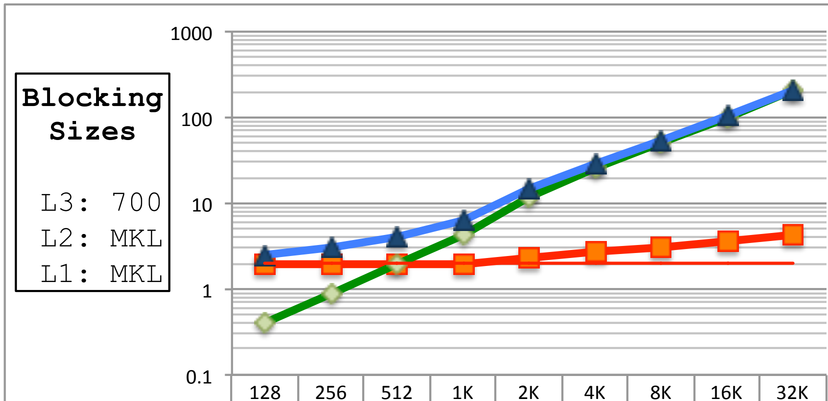


	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	2.1	2	4.1	8.4	17	34.2	68.5	137.2	274.4
L3_VICTIMS.E	0.8	1.3	2.7	5.3	10.7	21.6	43.5	86.5	172.9
LLC_S_FILLS.E	2.9	3.6	7	14	27.9	56	112.3	224.1	447.8
Misses on Ideal Cache	2.512	3.024	4.048	6	12	24	48	96	192
Write L.B.	2	2	2	2	2	2	2	2	2

Intel MKL Matmul

#DRAM reads >2x 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

Reproducibility

- Want bit-wise identical results from different runs of the same program on the same machine, possibly with different available resources
  - Needed for testing, debugging, correctness.
  - Requested by users (e.g. FEM, climate modeling, …)
- Hard just for summation, since floating point arithmetic is not associative because of roundoff
  - $(1e\text{-}16 + 1) - 1 \neq 1e\text{-}16 + (1 - 1)$

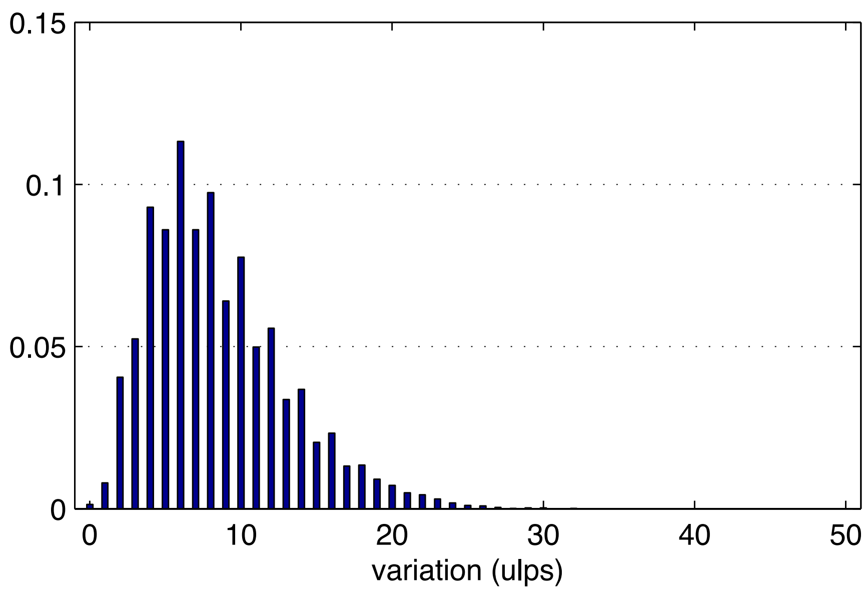
Reproducible Floating Point Computation

- Get bit-wise identical answer when you type a.out again
- NA-Digest submission on 8 Sep 2010
  - From Kai Diethelm, at GNS-MBH
  - Sought reproducible parallel sparse linear equation solver, demanded by customers (construction engineers), otherwise they don't believe results
  - Willing to sacrifice 40% - 50% of performance for it
- Email to ~110 Berkeley CSE faculty, asking about it
  - Most: "What?! How will I debug without reproducibility?"
  - Few: "I know better, and do careful error analysis"
  - S. Govindjee: needs it for fracture simulations
  - S. Russell: needs it for nuclear blast detection

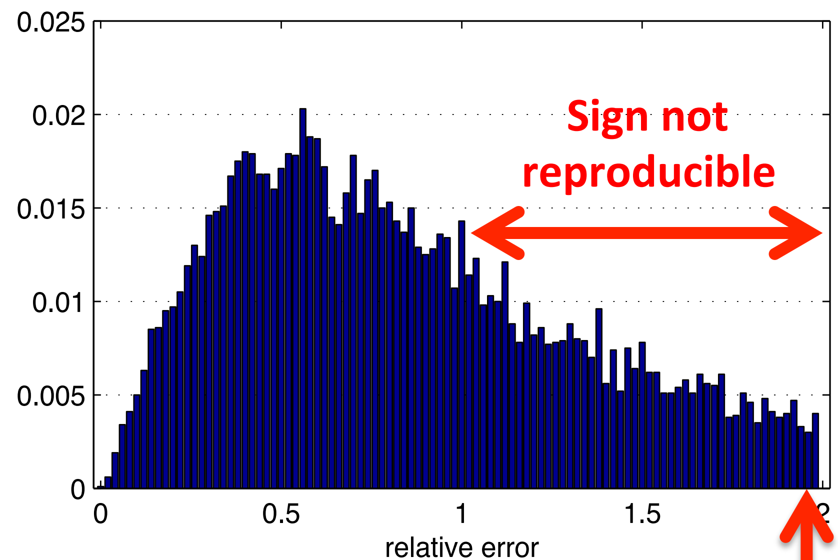
Intel MKL non-reproducibility

Vector size: 1e6. Data aligned to 16-byte boundaries. For each input vector:

- Dot products are computed using 1, 2, 3 or 4 threads
- Absolute error = maximum – minimum
- Relative error = Absolute error / maximum absolute value



Absolute Error for Random Vectors



Relative Error for Orthogonal vectors

Same magnitude opposite signs

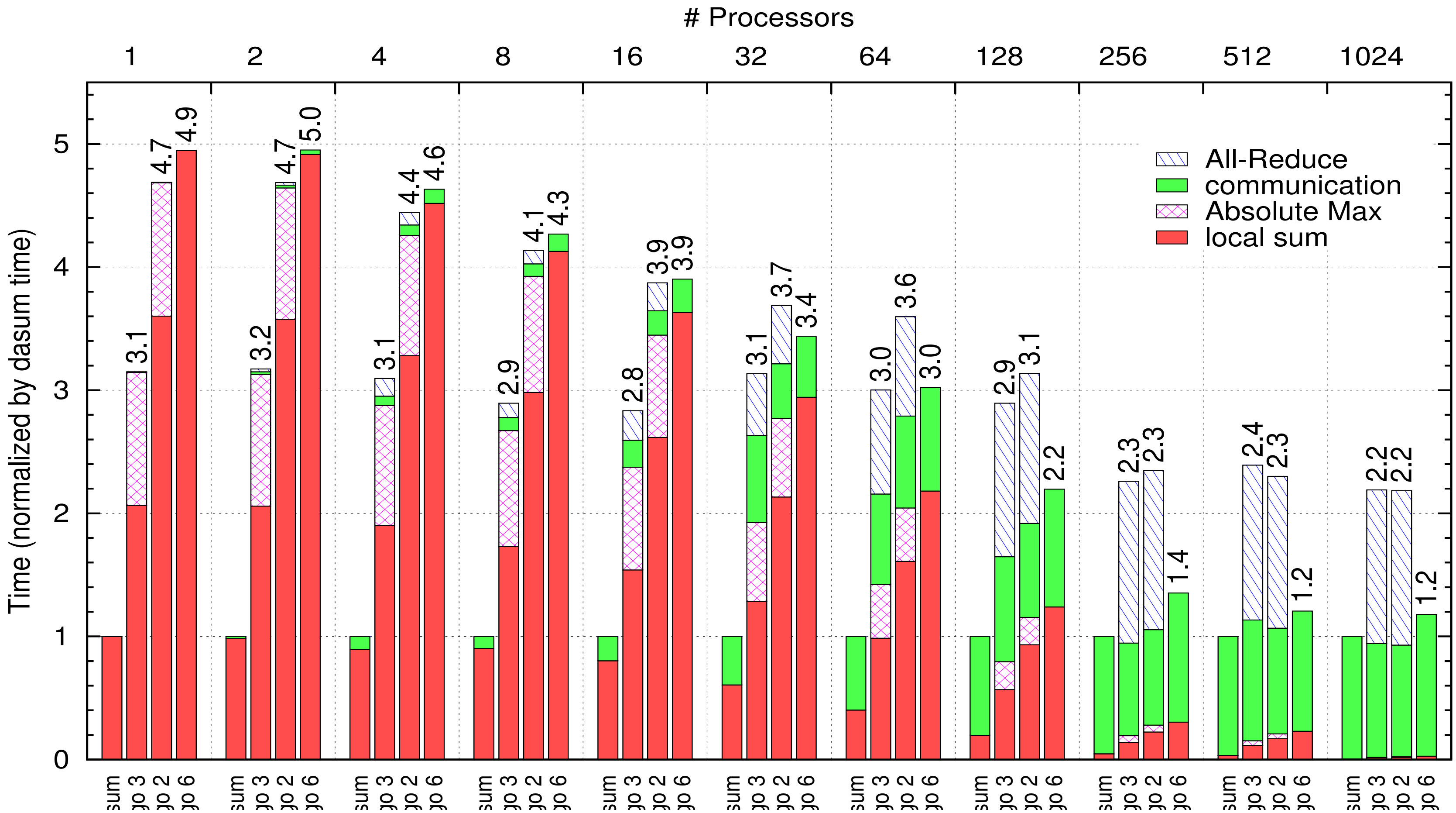
- Intel MKL 11.2 with CBWR: reproducible only for **fixed** number of threads and using the same instruction set (SSE2/AVX)

Goals/Approaches for Reproducible Sum

- Goals
  1. Same answer, independent of layout, #processors, order of summands
  2. Good performance (scales well)
  3. Portable (assume IEEE 754 only)
  4. User can choose accuracy
- Approaches
  - Guarantee fixed reduction tree (not 2. or 3.)
  - Use (very) high precision to get exact answer (not 2.)
  - Our approach (Nguyen, D.)
    - Oversimplified:
      1. Compute A = maximum absolute value (reproducible)
      2. Round all summands to one ulp in A
      3. Compute sum; rounding causes them to act like fixed point numbers
    - Possible with one reduction operation

Performance results on 1024 proc Cray XC30

1.2x to 3.2x slowdown vs fastest code, for n=1M



Reproducible Software and Hardware

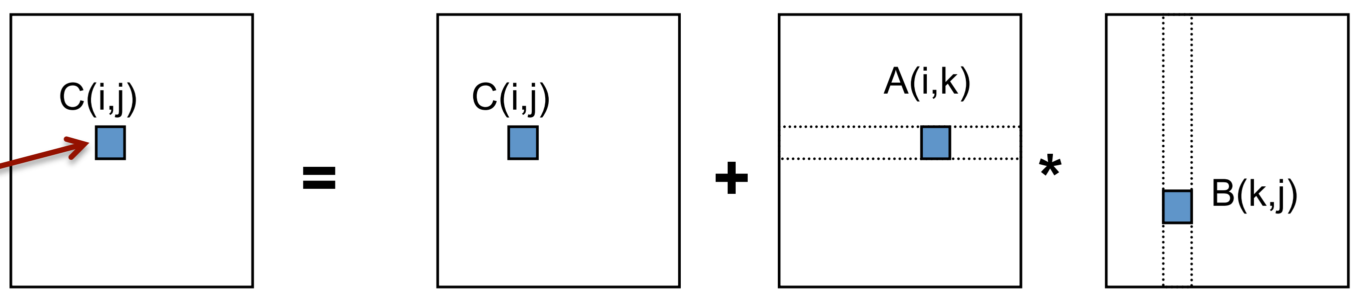
- Software for Reproducible BLAS 1 available at bebop.cs.berkeley.edu/reproblas
  - BLAS2, 3 under development
- Used Chisel (hardware design language) to design one new instruction that would make reproducible summation as fast (and more accurate) than standard summation
- Industrial interest

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}
```

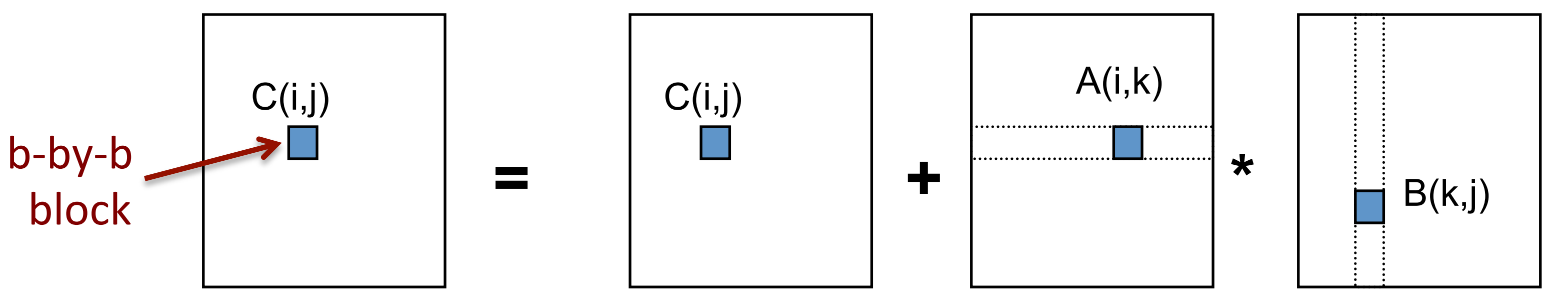
b-by-b block



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} … $b^2 \times (n/b)^2 = n^2$ reads
    for k = 1 to n/b
      {read block A(i,k) into fast memory} … $b^2 \times (n/b)^3 = n^3/b$ reads
      {read block B(k,j) into fast memory} … $b^2 \times (n/b)^3 = n^3/b$ 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} … $b^2 \times (n/b)^2 = n^2$ writes



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

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
{ C_{11} = RMM (A_{11} , B_{11} , n/2) + RMM (A_{12} , B_{21} , n/2)
C_{12} = RMM (A_{11} , B_{12} , n/2) + RMM (A_{12} , B_{22} , n/2)
C_{21} = RMM (A_{21} , B_{11} , n/2) + RMM (A_{22} , B_{21} , n/2)
C_{22} = RMM (A_{21} , B_{12} , n/2) + RMM (A_{22} , B_{22} , n/2) }
return

Recursive Matrix Multiplication (RMM) (2/2)

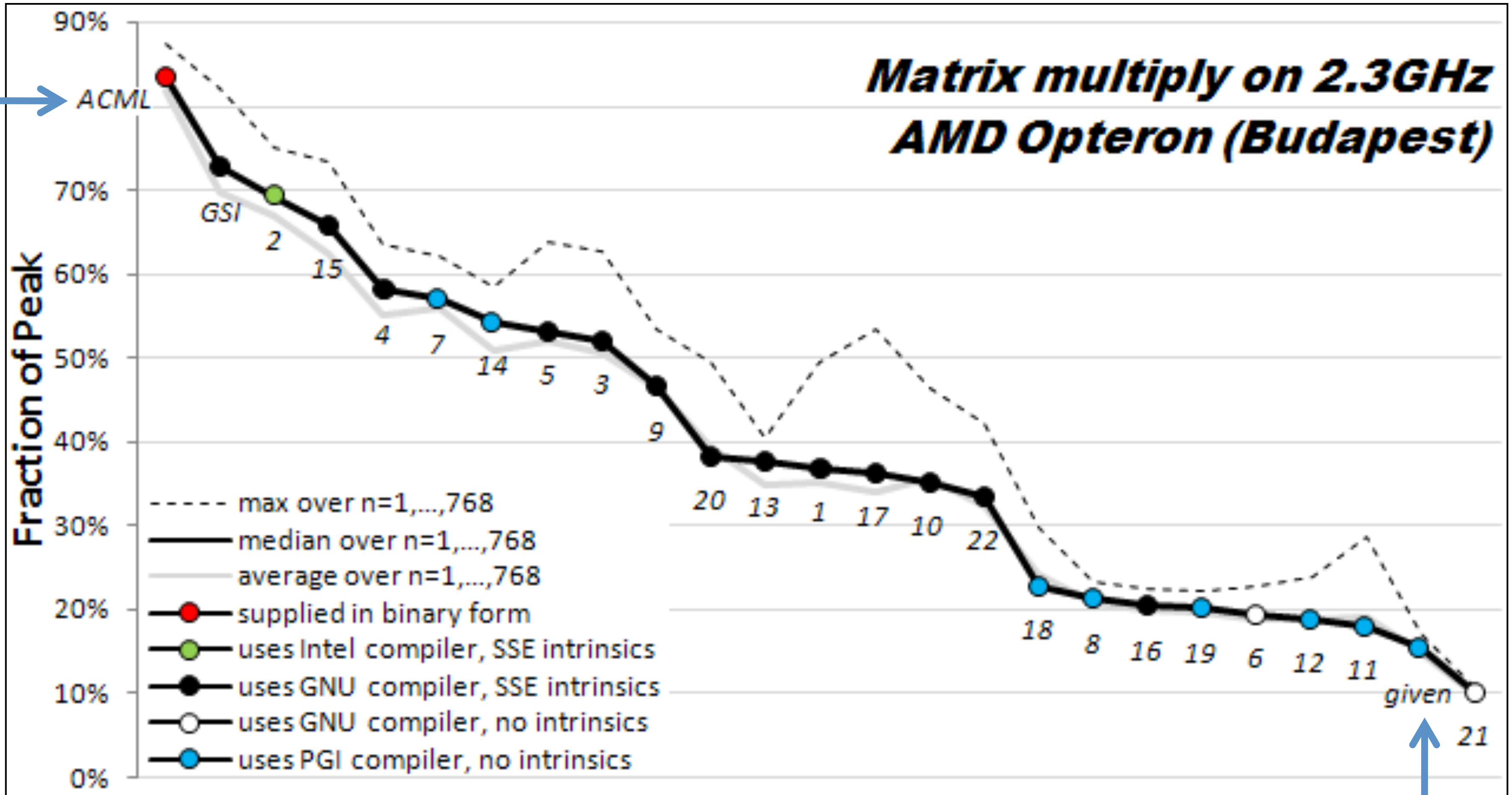
```
func C = RMM (A, B, n)
   if n=1, C = A * B, else
     {  C11 = RMM (A11 , B11 , n/2) + RMM (A12 , B21 , n/2)
        C12 = RMM (A11 , B12 , n/2) + RMM (A12 , B22 , n/2)
        C21 = RMM (A21 , B11 , n/2) + RMM (A22 , B21 , n/2)
        C22 = RMM (A21 , B12 , n/2) + RMM (A22 , B22 , n/2)  }
   return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

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

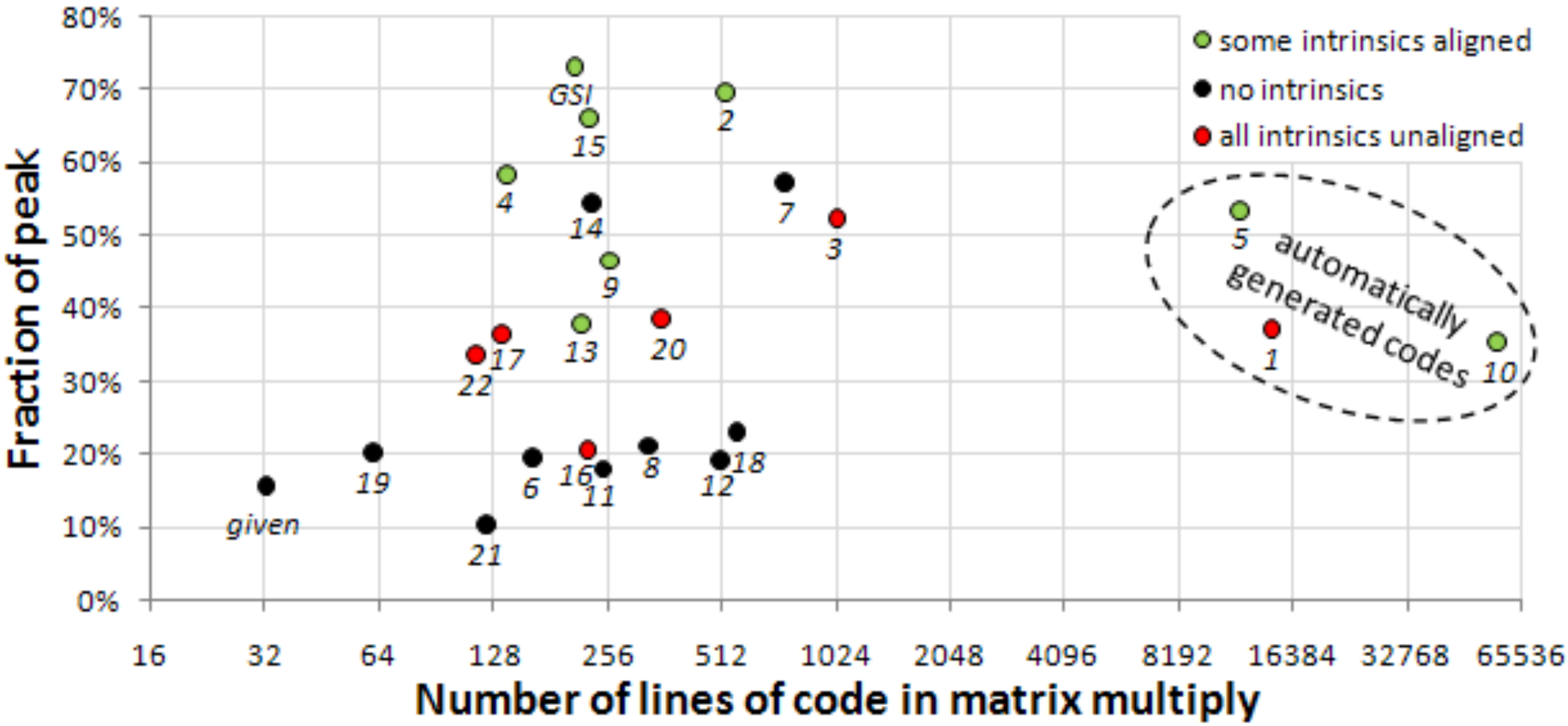
"Cache oblivious", works for memory hierarchies, but not panacea

How hard is hand-tuning matmul, anyway?



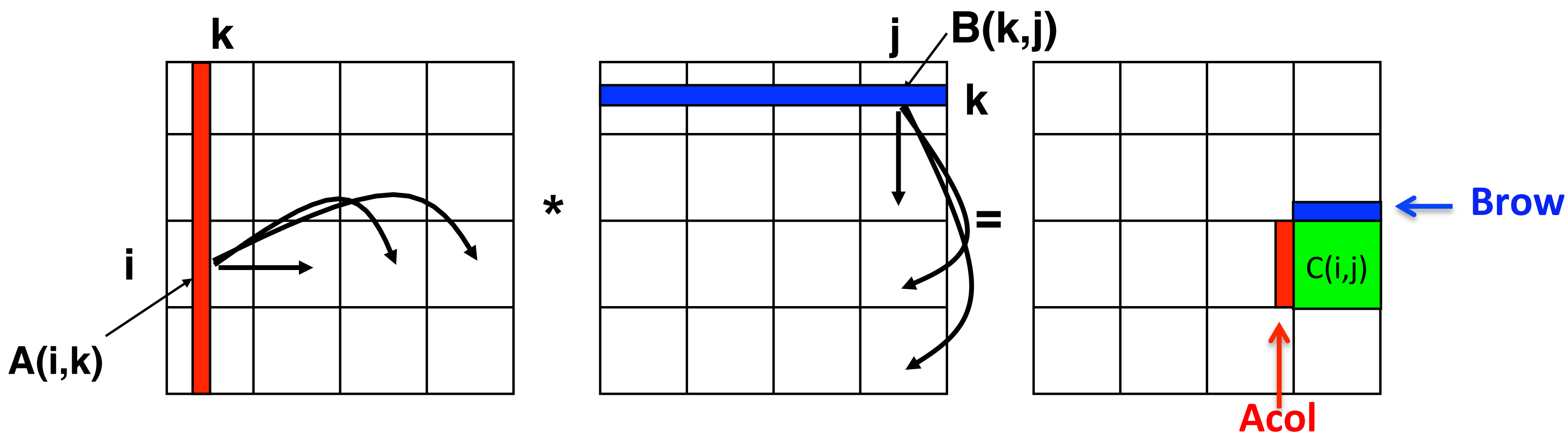
- **Results of 22 student teams trying to tune matrix-multiply, in CS267 Spr09**
- **Students given "blocked" code to start with (7x faster than naïve)**
  - **Still hard to get close to vendor tuned performance (ACML) (another 6x)**
- **For more discussion, see www.cs.berkeley.edu/~volkov/cs267.sp09/hw1/results/**

How hard is hand-tuning matmul, anyway?



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

(nearly) optimal using minimum memory $M=O(n^2/P)$



For k=0 to n/b-1 … b = block size = #cols in A(i,k) = #rows in B(k,j)

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

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
  - Processor (i,j,k) performs $C(i,j) = C(i,j) + A(i,k)*B(k,j)$, where each submatrix is $n/p^{1/3} \times n/p^{1/3}$
- Not always that much memory available…

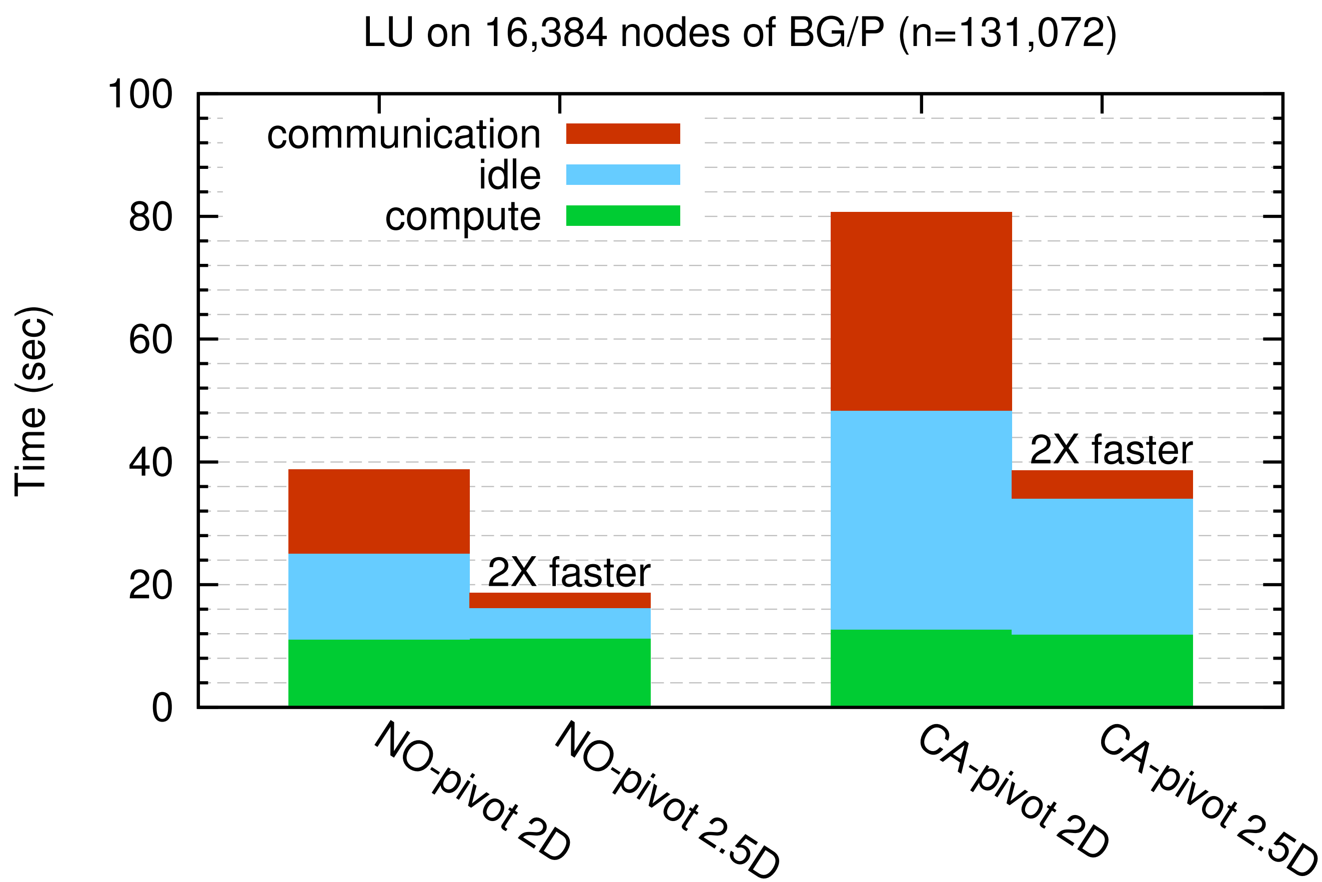
Perfect Strong Scaling – in Time and Energy (2/2)

- $T(cP) = n^3/(cP)\ [\ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})\ ] = T(P)/c$
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E MT(cP) + \varepsilon_E T(cP)\ \} = E(P)$

- Perfect scaling extends to N-body, Strassen, …
- 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?

2.5D vs 2D LU
With and Without Pivoting



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

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 – **1.6x slower than just accessing data twice** (Gleich and Benson)
- 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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - TSQR: Tall-Skinny QR
  - CA $O(n^3)$ 2.5D LU
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm
- CA-Krylov methods

Back to LU: 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}'

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

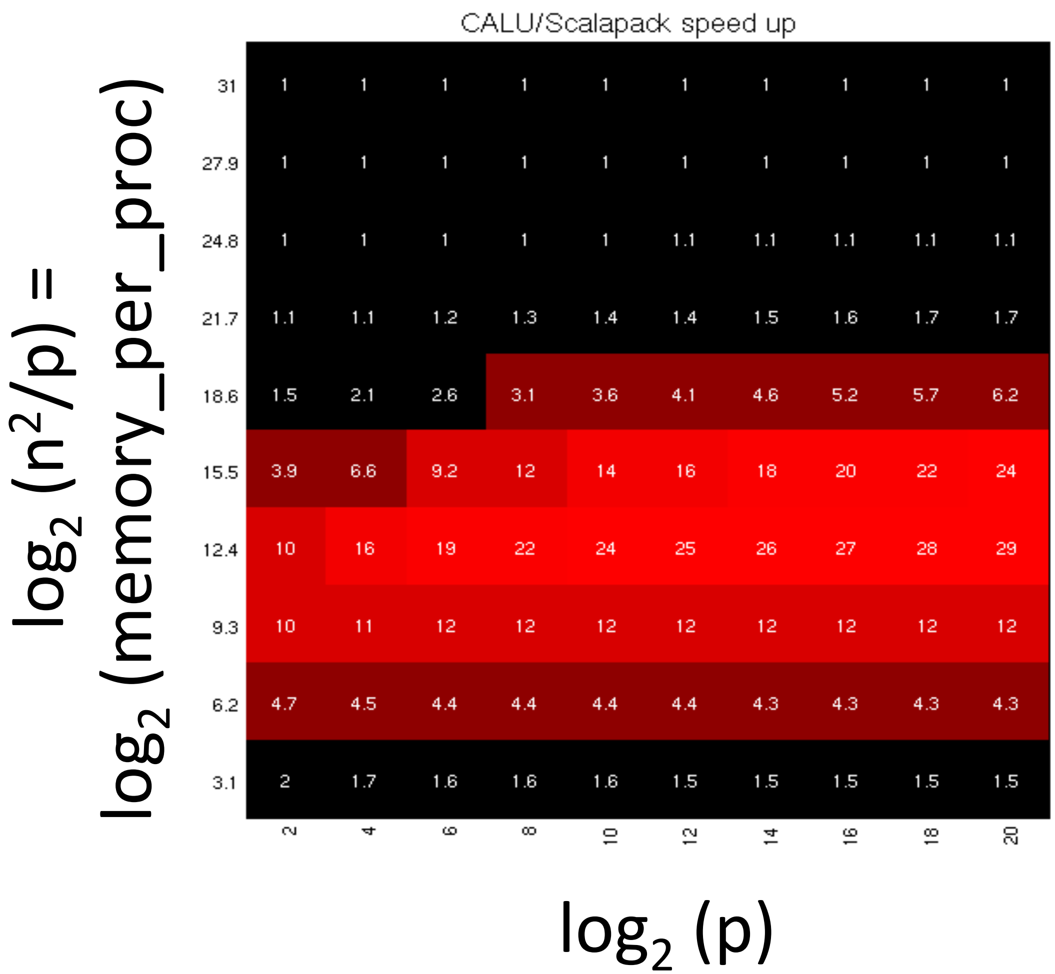
Choose b pivot rows

- 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

Exascale Machine Parameters Source: DOE Exascale Workshop

- 2^20 ≈ 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 Gaussian Elimination: 2D CA-LU vs ScaLAPACK-LU



Ongoing Work

- Lots more work on
  - Algorithms:
    - BLAS, LDL^T, QR with pivoting, other pivoting schemes, eigenproblems, …
    - Sparse matrices, structured matrices
    - All-pairs-shortest-path, …
    - Both 2D (c=1) and 2.5D (c>1)
    - But only bandwidth may decrease with c>1, not latency (eg LU)
  - Platforms:
    - Multicore, cluster, GPU, cloud, heterogeneous, low-energy, …
  - Software:
    - Integration into Sca/LAPACK, PLASMA, MAGMA,…
- Integration into applications
  - CTF (with ANL): symmetric tensor contractions

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)

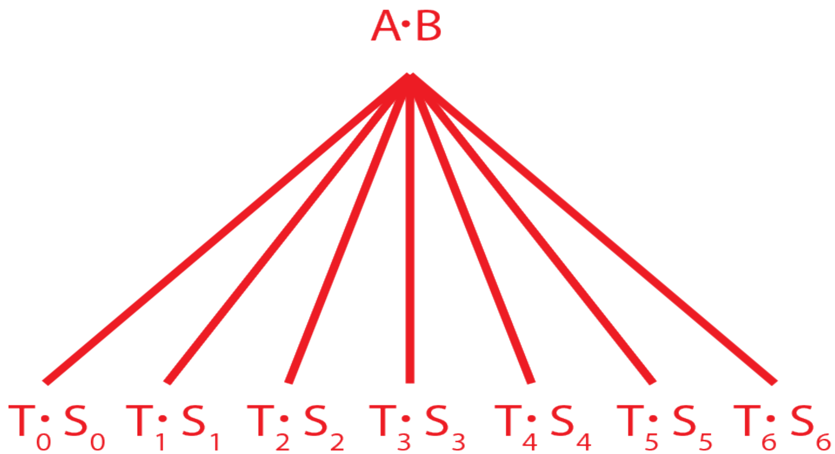
- "Blocked" code

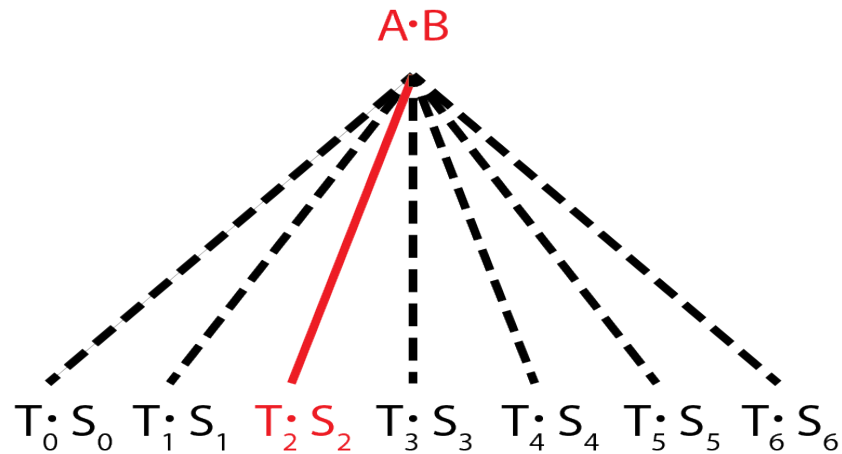
```
for i1 = 1:b:n,  for j1 = 1:b:n,   for k1 = 1:b:n
  for i2 = 0:b-1,  for j2 = 0:b-1,   for k2 = 0:b-1  ┐
     i=i1+i2,  j = j1+j2,  k = k1+k2                 ├ b x b matmul
     C(i,j)+=A(i,k)*B(k,j)                           ┘
```

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

Communication Avoiding Parallel Strassen *(CAPS)*

BFS vs. DFS





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

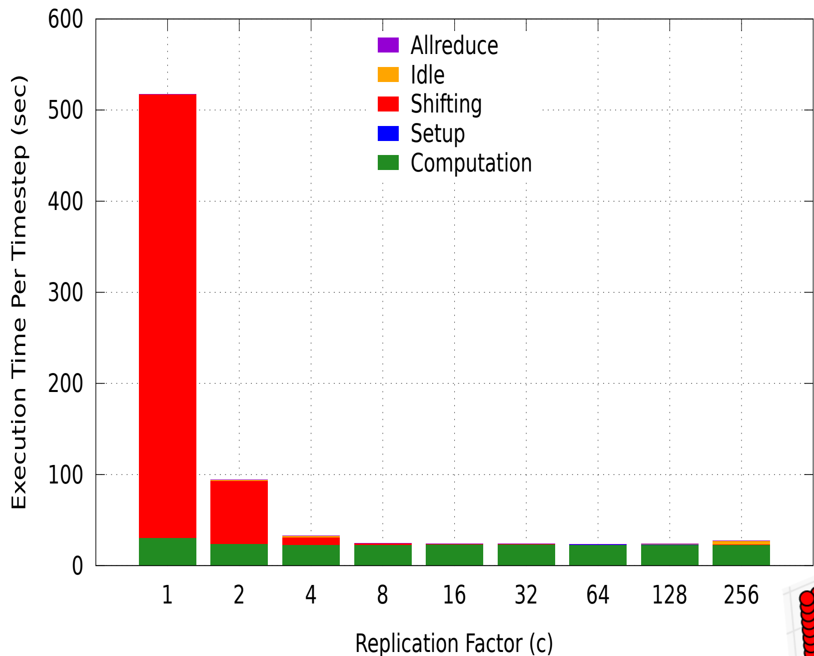
Runs all 7 multiplies sequentially
Each on all P processors
Needs 1/4 as much memory

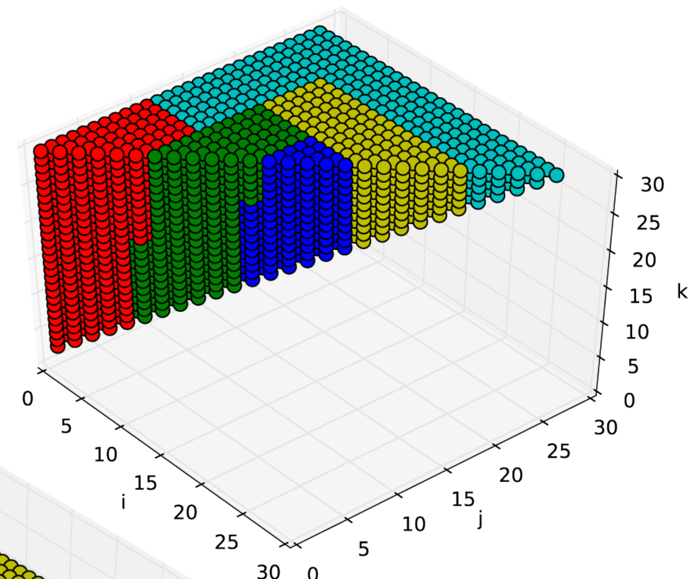
```
CAPS
  If EnoughMemory and P ≥ 7
    then BFS step
    else DFS step
  end if
```

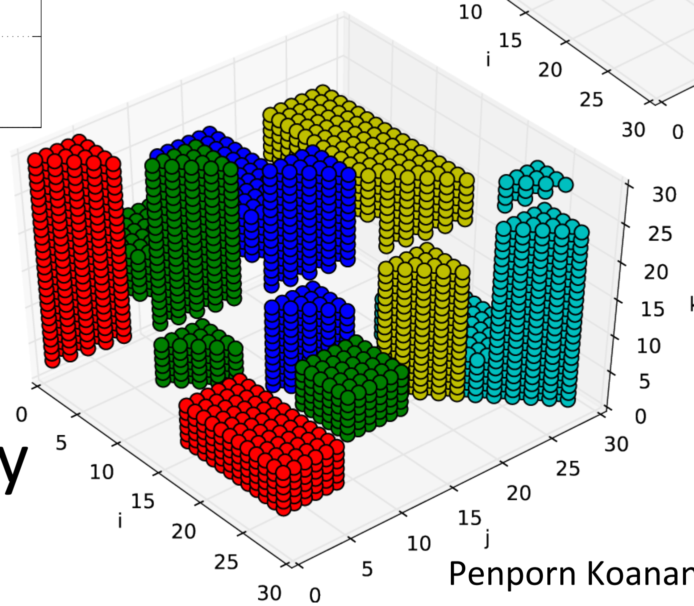
Best way to interleave BFS and DFS is a tuning parameter

Variable Loop Bounds are More Complicated

- Redundancy in n-body calculation f(i,j), f(j,i)
- k-way n-body problems ("k-body") has even more







- Can achieve both communication and computation (symmetry exploiting) optimal

Penporn Koanantakool and K. Yelick

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 group homomorphisms, eg

  ϕ_C (i1,i2,…,ik) = (i1+2*i3-i7)

  ϕ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

- Can we bound #loop_iterations (= |S|)

  given bounds on #points in its images, i.e. bounds on $|\phi_C(S)|$, $|\phi_A(S)|$, … ?

General Communication Bound

- Given S subset of Z^k, group homomorphisms ϕ_1, ϕ_2, …, bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, … , $|\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1,\ldots,s_m$:

  for all subgroups $H < Z^k$, $\quad \mathrm{rank}(H) \le \Sigma_j\ s_j * \mathrm{rank}(\phi_j(H))$
- Thm (Christ/Tao/Carbery/Bennett): Given $s_1,\ldots,s_m$

$$|S| \le \Pi_j\ |\phi_j(S)|^{s_j}$$

- Thm: Given a program with array refs given by ϕ_j, choose s_j to minimize $s_{HBL} = \Sigma_j\ s_j$ subject to HBL-LP. Then

$$\#\text{words\_moved} = \Omega\ (\#\text{iterations}/M^{s_{HBL}-1})$$

Is this bound attainable?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q (conjectured to be undecidable)
- Thm: (good news) Another LP with same solution is decidable
- Depends on loop dependencies
  - Best case: none, or reductions (like matmul)
- Thm: In many cases, solution x of Dual HBL-LP gives optimal tiling
  - Ex: For Matmul, x = [1/2, 1/2, 1/2] so optimal tiling is $M^{1/2} \times M^{1/2} \times M^{1/2}$

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

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

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

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

Is this bound attainable (1/2)?

- But first: Can we write it down?
- Thm: (bad news) HBL-LP reduces to Hilbert's 10^{th} problem over Q
  - conjectured to be undecidable
- Thm: (good news) Another LP with same solution is decidable (but expensive):

  Let L = (V1,V2,…) be countable list of all subspaces of Q^n

  i=1

  repeat

      i = i + 1

  until polytope determined by inequalities from (V1,…,Vi) is right one

- Thm: (better news) Enough to search L = lattice of kernels of $\phi_1, \phi_2, \ldots, \phi_m$
  - Similar to result of Valdimarsson in continuum case
  - Corollary (Dedekind) If m=3, only need to consider 28 subspaces
- Thm: (even better news) Easy to write down LP explicitly in many simple cases of interest (eg all ϕ_j = {subset of indices})

Ongoing Work

- Accelerate decision procedure for lower bounds
  - Ex: At most 3 arrays, or 4 loop nests
- Have yet to find a case where we cannot attain lower bound – can we prove this?
- Extend "perfect scaling" results for time and energy by using extra memory
  - "n.5D algorithms"
- Incorporate into compilers