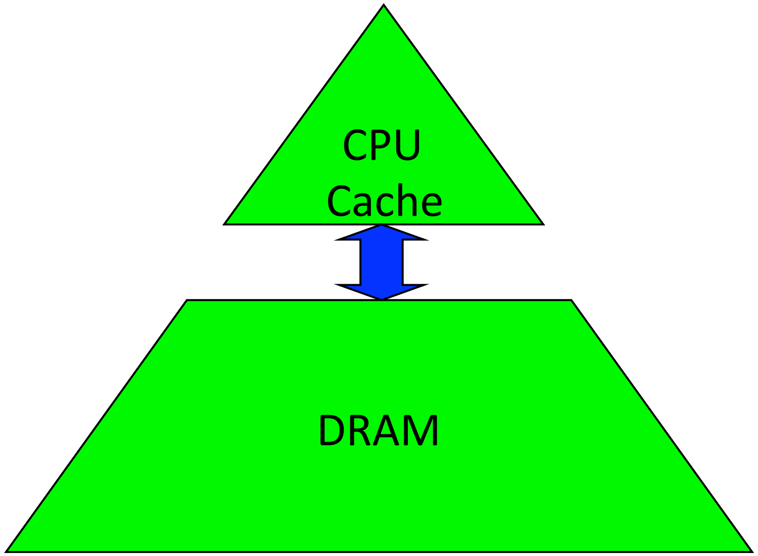
Communication-Avoiding Algorithms

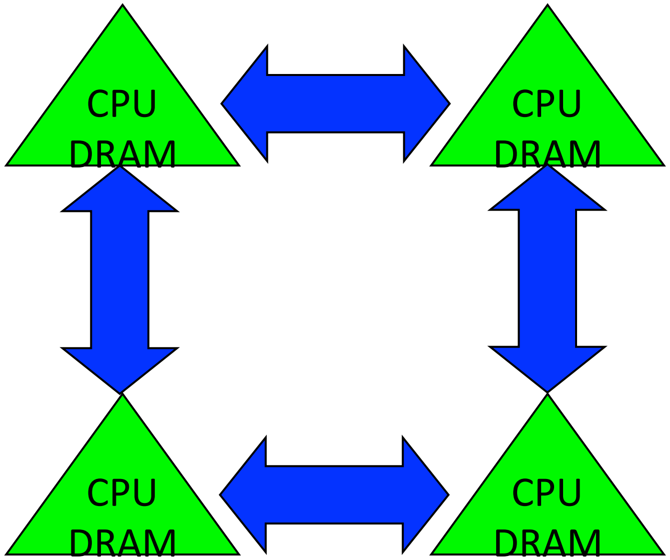
Jim Demmel
EECS & Math Departments
UC Berkeley

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

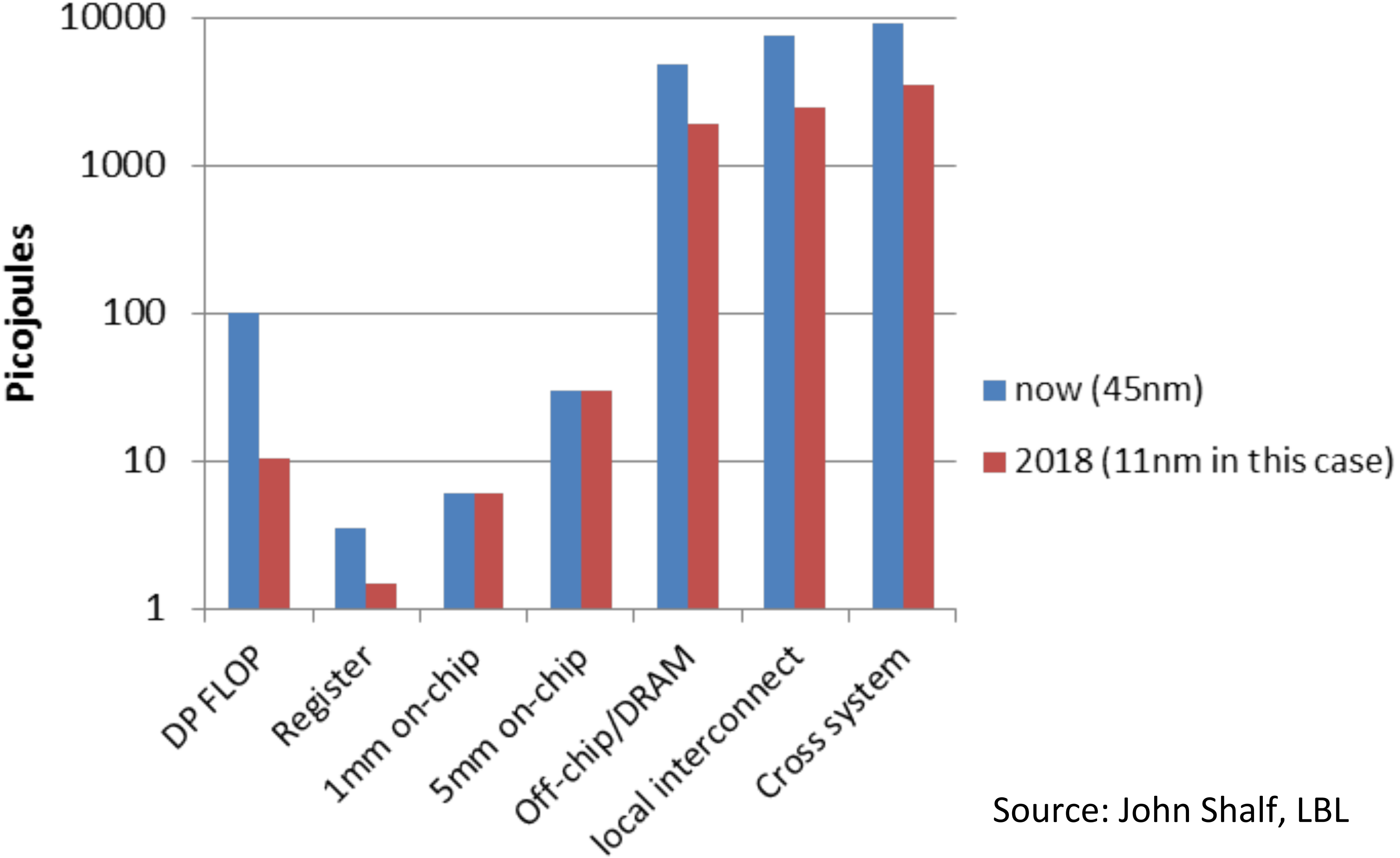
communication

- 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

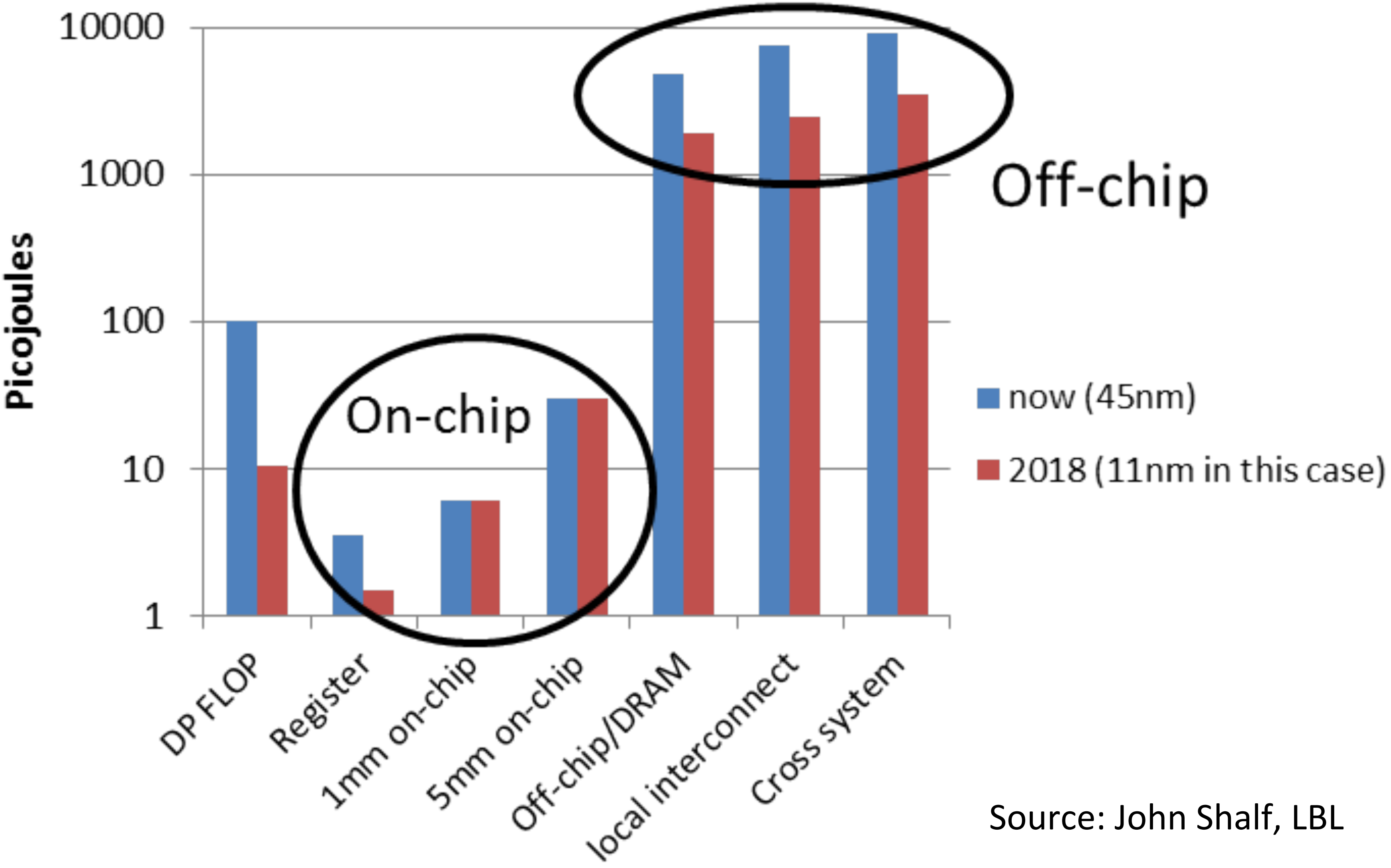
Why Minimize Communication? (3/3)



Source: John Shalf, LBL

Why Minimize Communication? (3/3)

Minimize communication to save energy



Source: John Shalf, LBL

Goals

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

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
  - TSQR: Tall-Skinny QR
  - CA $O(n^3)$ 2.5D Matmul
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm
- CA-Krylov methods

Collaborators and Supporters

- Michael Christ, Jack Dongarra, Ioana Dumitriu, David Gleich, Laura Grigori, Ming Gu, Olga Holtz, Julien Langou, Tom Scanlon, Kathy Yelick
- Grey Ballard, Austin Benson, Abhinav Bhatele, Aydin Buluc, Erin Carson, Maryam Dehnavi, Michael Driscoll, Evangelos Georganas, Nicholas Knight, Penporn Koanantakool, Ben Lipshitz, Oded Schwartz, Edgar Solomonik, Hua Xiang
- Other members of ParLab, BEBOP, CACHE, EASI, FASTMath, MAGMA, PLASMA, TOPS projects
  - bebop.cs.berkeley.edu
- Thanks to NSF, DOE, UC Discovery, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle

Outline

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

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) / $M^{1/2}$)

- 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) / $M^{1/2}$)

#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)

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
  - 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
- Lots of work in progress

Outline

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

TSQR: QR of a Tall, Skinny matrix

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

$$\left(\begin{array}{c} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{array}\right) = \left(\begin{array}{c} Q_{01}\ R_{01} \\ \hline Q_{11}\ R_{11} \end{array}\right)$$

$$\left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right) = \left(\begin{array}{c} Q_{02}\ R_{02} \end{array}\right)$$

TSQR: QR of a Tall, Skinny matrix

$$W = \left(\begin{array}{c} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{array}\right) = \left(\begin{array}{c} Q_{00}\,R_{00} \\ \hline Q_{10}\,R_{10} \\ \hline Q_{20}\,R_{20} \\ \hline Q_{30}\,R_{30} \end{array}\right) = \left(\begin{array}{cccc} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{array}\right)$$

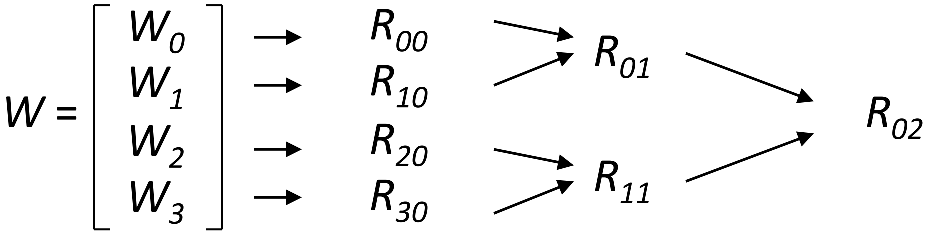
$$\left(\begin{array}{c} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{array}\right) = \left(\begin{array}{c} Q_{01}\,R_{01} \\ \hline Q_{11}\,R_{11} \end{array}\right) = \left(\begin{array}{cc} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot \left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right)$$

$$\left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right) = \left(\begin{array}{c} Q_{02}\,R_{02} \end{array}\right)$$

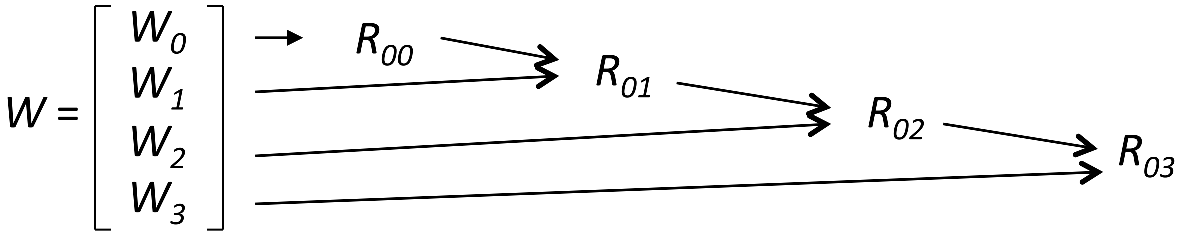
Output = $\{ Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} \}$

TSQR: An Architecture-Dependent Algorithm

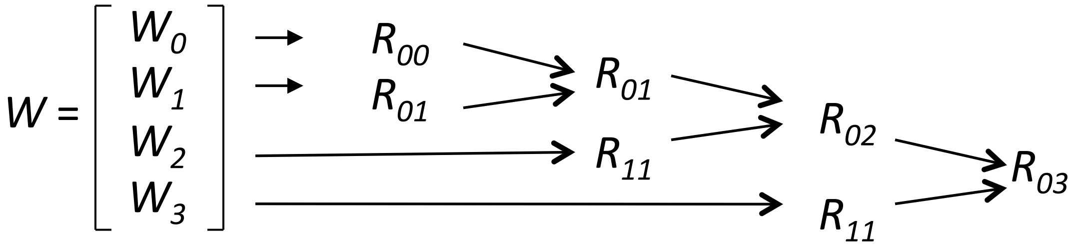
Parallel:



Sequential:



Dual Core:



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

Can choose reduction tree dynamically

TSQR Performance Results

- Parallel
  - 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 (Dongarra, Langou et al)
  - Cloud – **1.6x slower than 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

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

Can we do Better?

Can we do better?

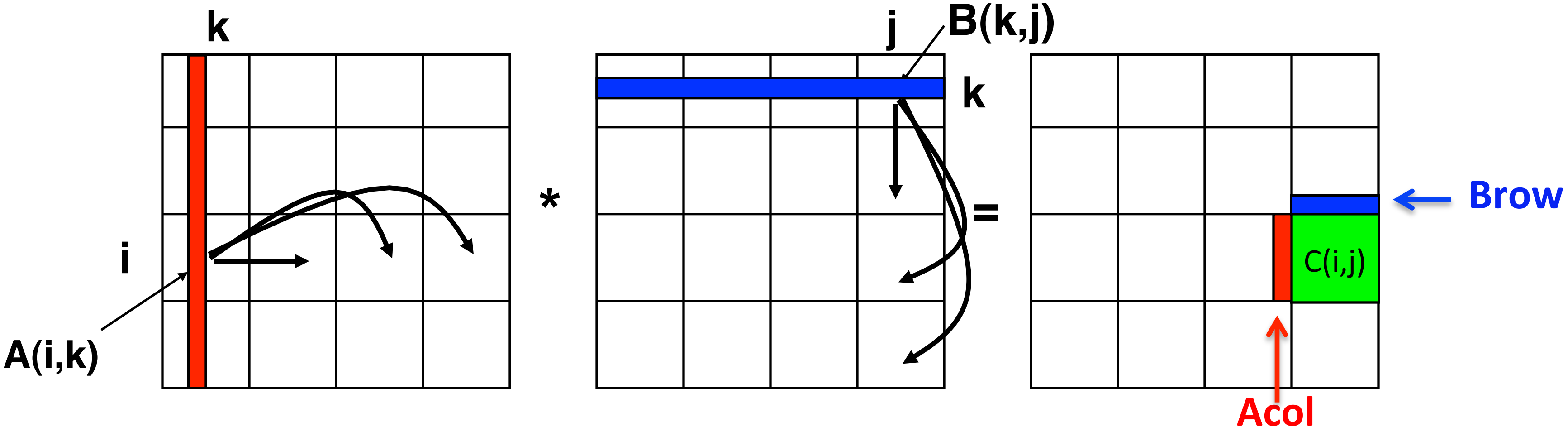
- Aren't we already optimal?
- Why assume $M = O(n^2/p)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?

Outline

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

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

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

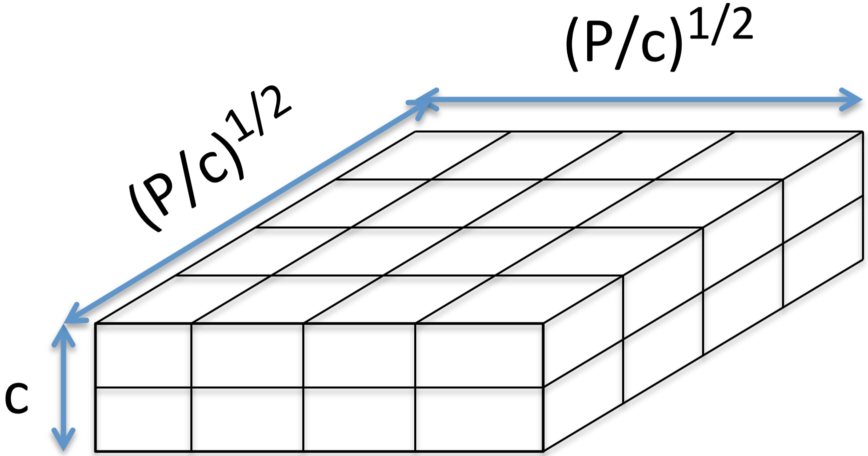


Using more than the minimum memory

- What if matrix small enough to fit c>1 copies, so $M = cn^2/P$?
  - #words_moved = $\Omega($ #flops / $M^{1/2}$ $)$ = $\Omega(n^2 / (c^{1/2} P^{1/2}))$
  - #messages = $\Omega($ #flops / $M^{3/2}$ $)$ = $\Omega(P^{1/2} / c^{3/2})$
- Can we attain new lower bound?

2.5D Matrix Multiplication

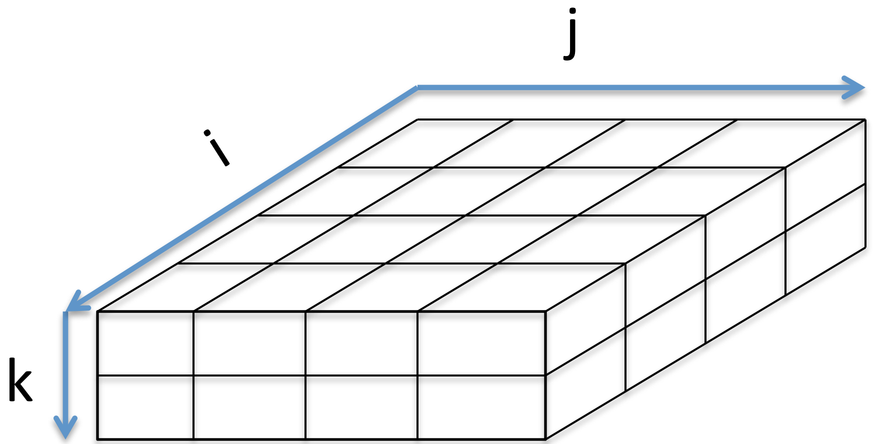
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



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



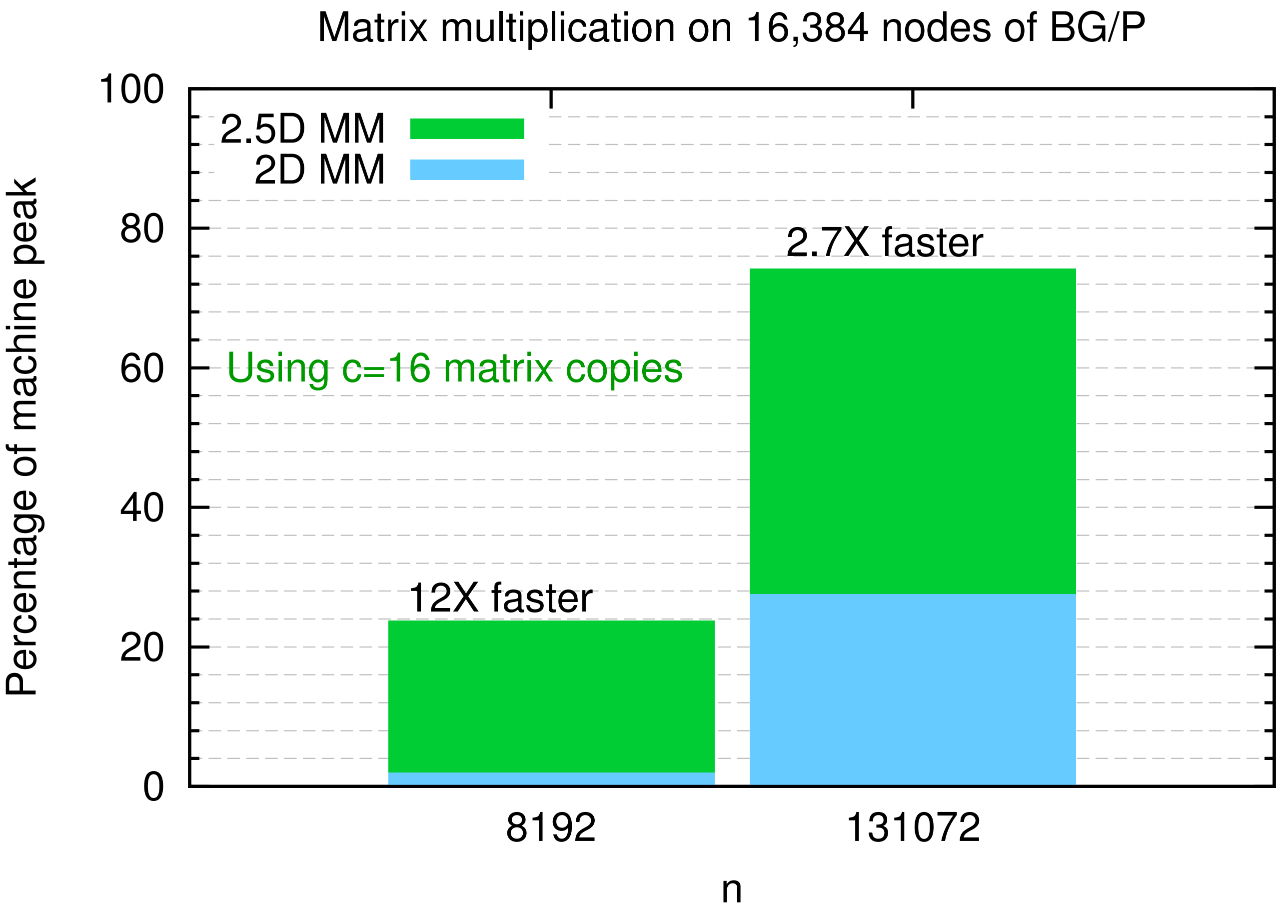
Initially $P(i,j,0)$ owns $A(i,j)$ and $B(i,j)$ each of size $n(c/P)^{1/2}$ x $n(c/P)^{1/2}$

(1) $P(i,j,0)$ broadcasts $A(i,j)$ and $B(i,j)$ to $P(i,j,k)$

(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of $\Sigma_m A(i,m)*B(m,j)$

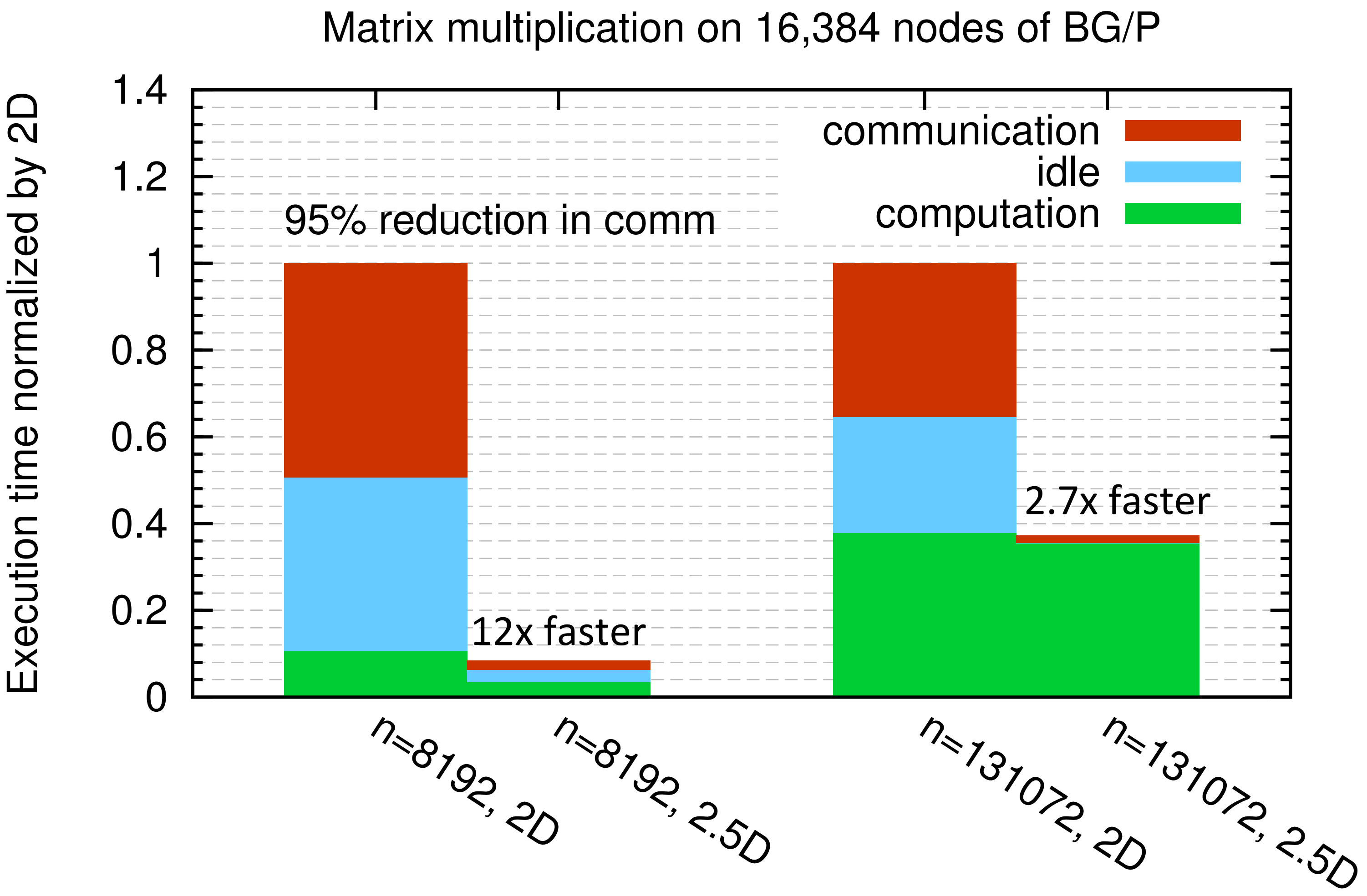
(3) Sum-reduce partial sums $\Sigma_m A(i,m)*B(m,j)$ along k-axis so $P(i,j,0)$ owns $C(i,j)$

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



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

c = 16 copies



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

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 scaling extends to N-body, Strassen, …

Ongoing Work

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

Outline

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

Communication Lower Bounds for Strassen-like matmul algorithms

Classical
$O(n^{3})$ matmul:

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

Strassen's
$O(n^{\lg 7})$ matmul:

#words_moved =
$\Omega\ (M(n/M^{1/2})^{\lg 7}/P)$

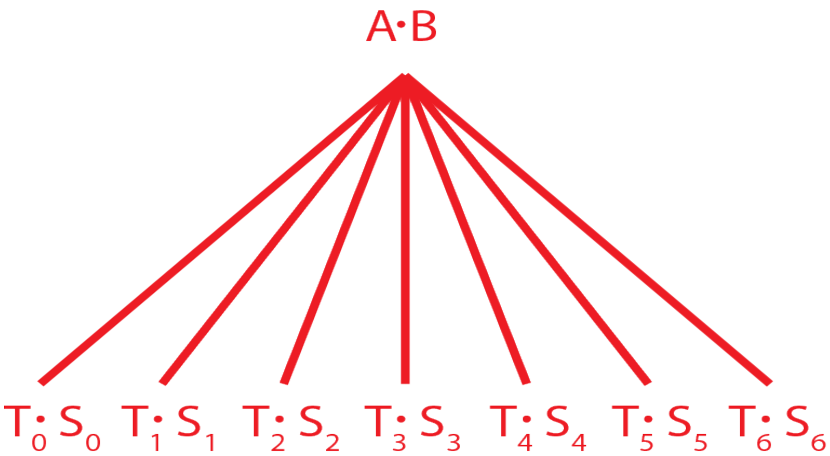
Strassen-like
$O(n^{\omega})$ matmul:

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

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

Communication Avoiding Parallel Strassen *(CAPS)*

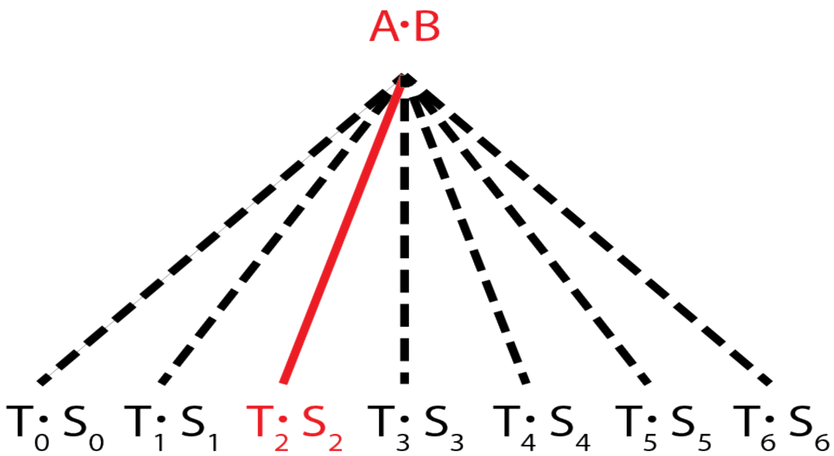
BFS vs. DFS



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

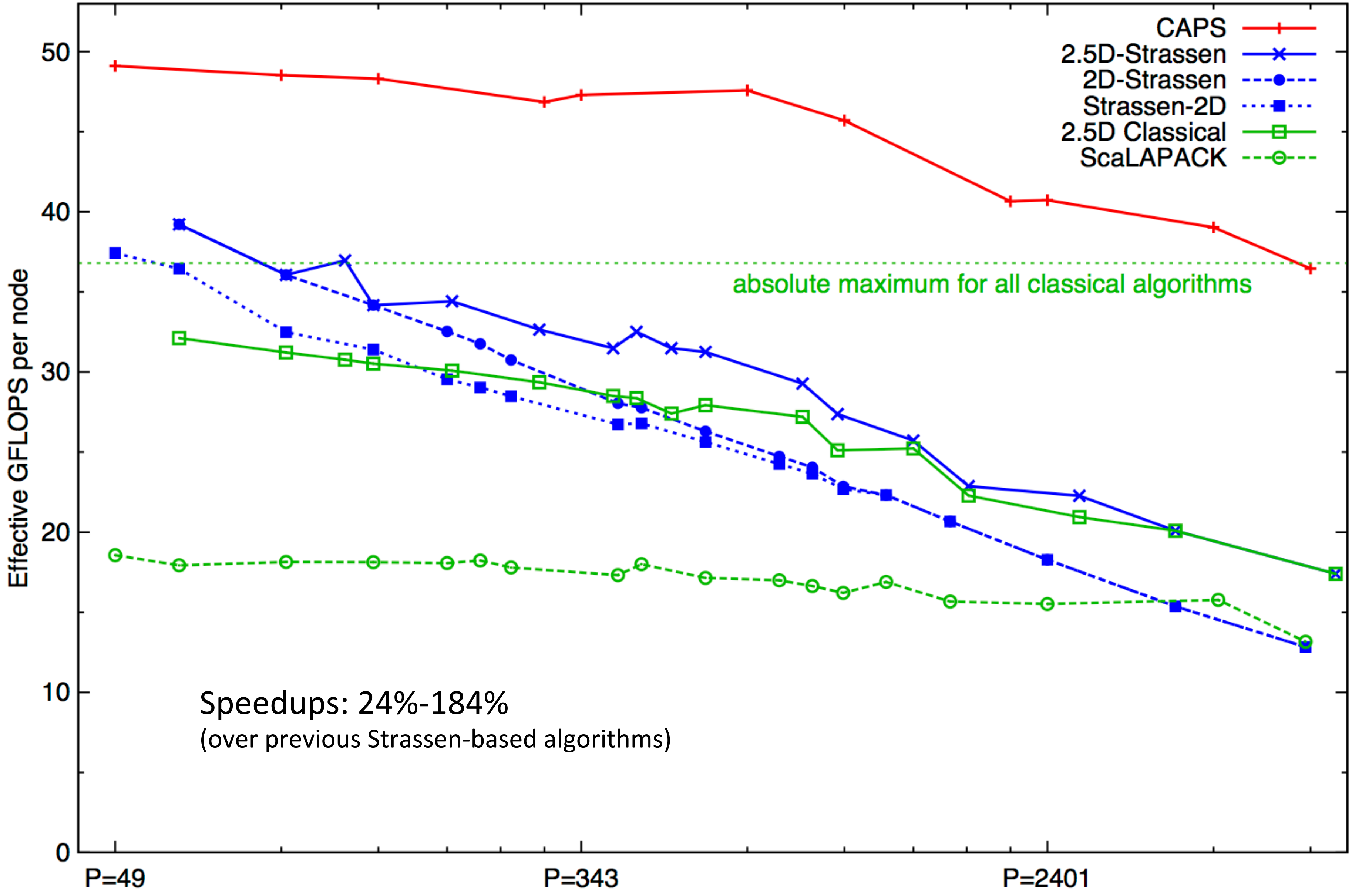
Needs 1/4 as much memory

```
CAPS
    If EnoughMemory and $P \geq 7$
        then BFS step
        else DFS step
    end if
```

Best way to interleave BFS and DFS is an tuning parameter

Performance Benchmarking, Strong Scaling Plot

Franklin (Cray XT4) n = 94080



Speedups: 24%-184%
(over previous Strassen-based algorithms)

Outline

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

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

New Thm applied to Matmul

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

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

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

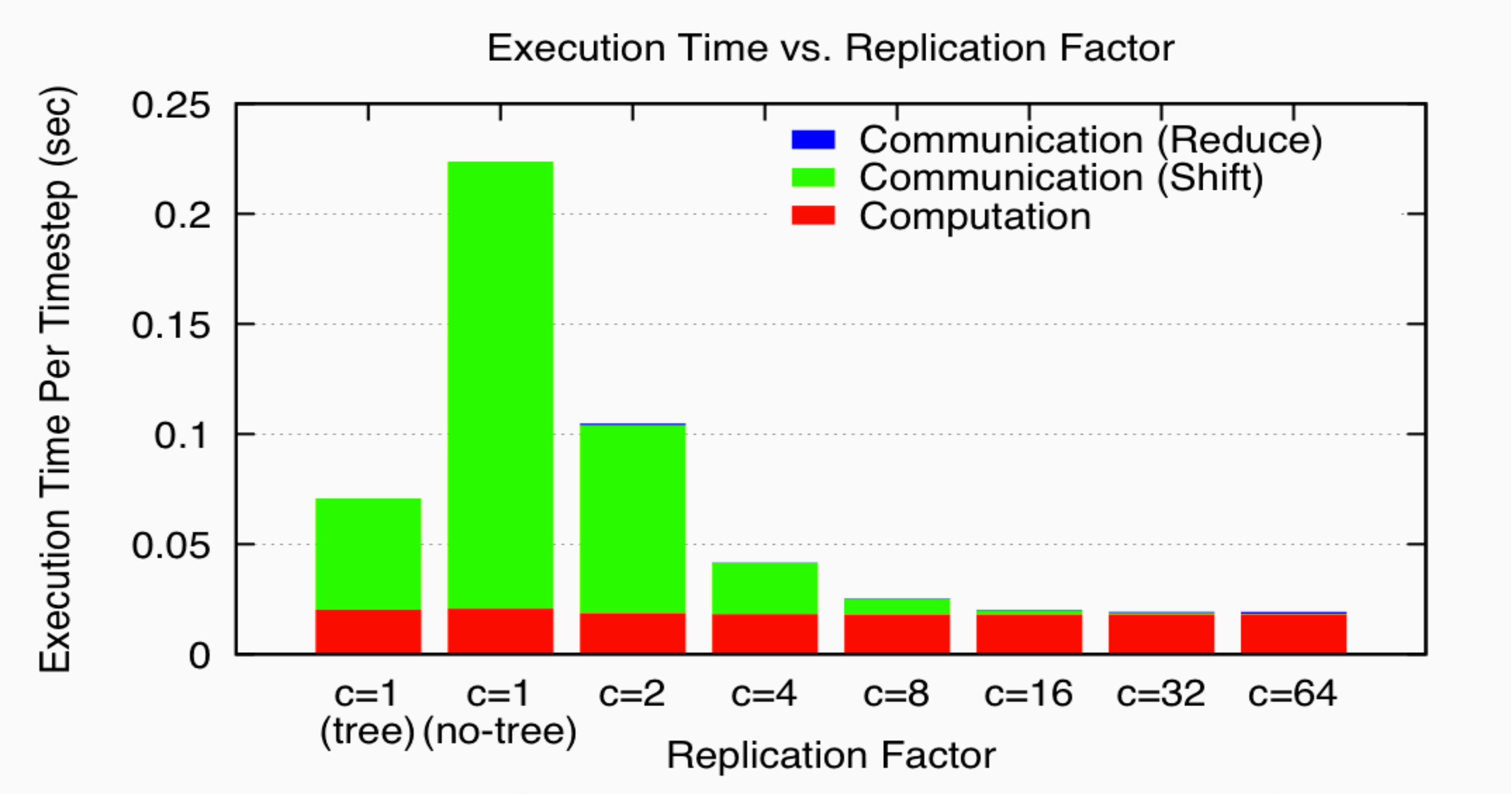
$$\Delta = \begin{array}{c c c} i & j & \\ \left(\begin{matrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{matrix}\right. & \left.\right) & \begin{matrix} F \\ P(i) \\ P(j) \end{matrix} \end{array}$$

- 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

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

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

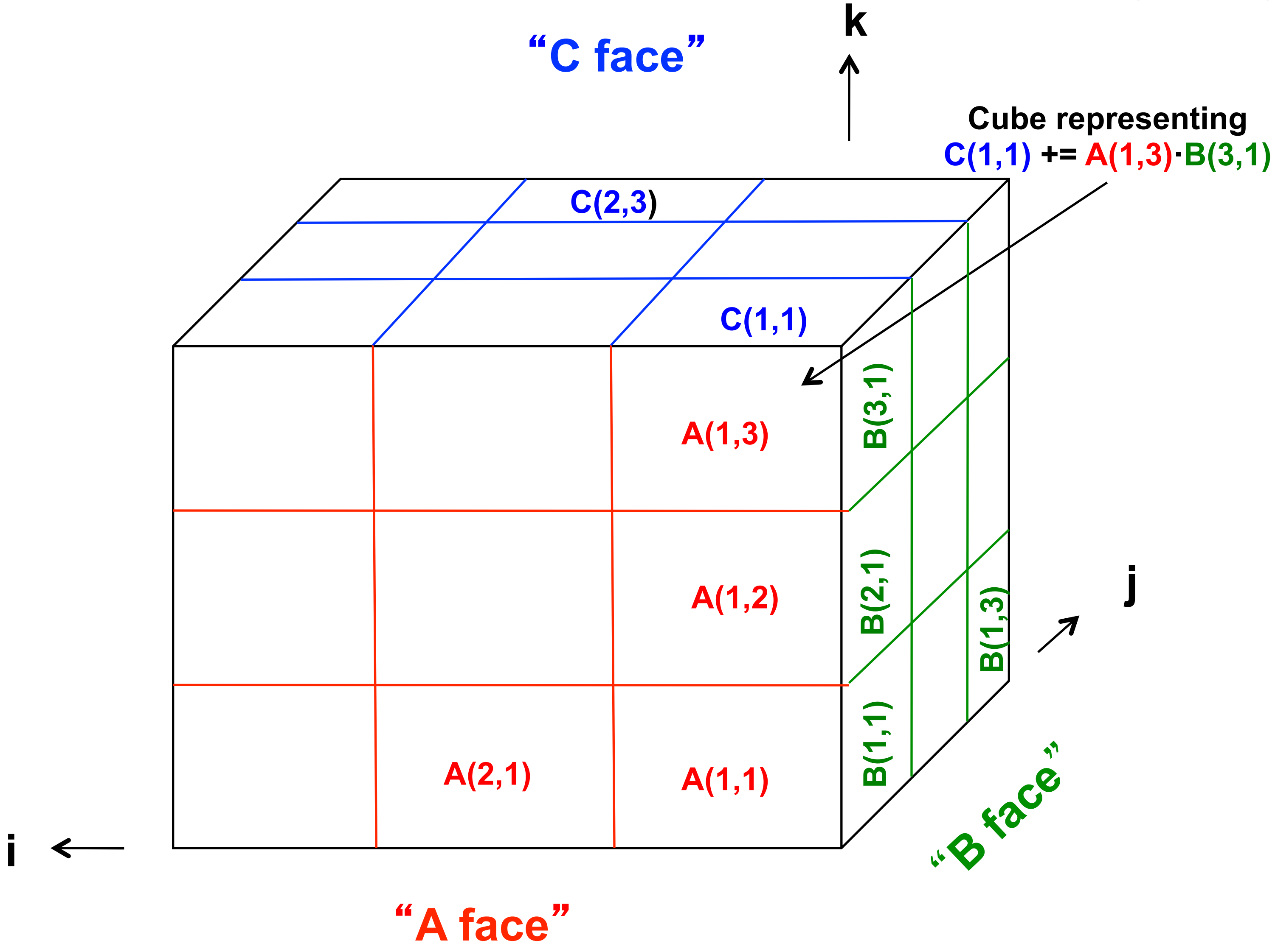
- Solve LP for $x = [x1,\ldots,x7]^T$: max $\mathbf{1}^T x$ s.t. $\Delta\, x \le \mathbf{1}$
  - Result: $x = [2/7,3/7,1/7,2/7,3/7,4/7]$, $\mathbf{1}^T x = 15/7 = s_{HBL}$
- Thm: #words_moved = $\Omega(n^6/M^{s_{HBL}-1}) = \Omega(n^6/M^{8/7})$

  Attained by block sizes $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

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

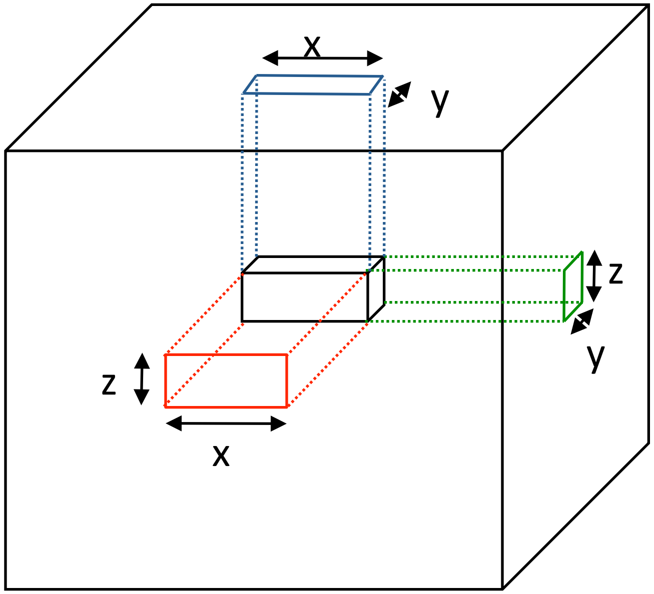
- Originally for C = A*B by Irony/Tiskin/Toledo (2004)
- Proof idea
  - Suppose we can bound #useful_operations ≤ G doable with data in fast memory of size M
  - So to do F = #total_operations, need to fill fast memory F/G times, and so #words_moved ≥ MF/G
- Hard part: finding G
- Attaining lower bound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

Proof of communication lower bound (2/3)



- 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

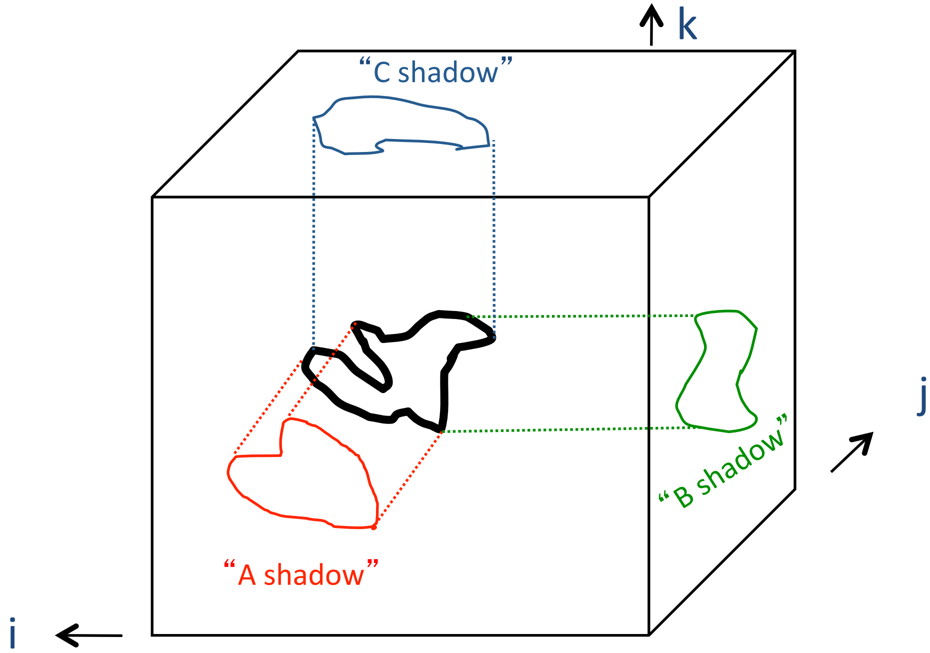
= Volume of black box

= x·y·z

= $(xz \cdot zy \cdot yx)^{1/2}$

= (#A□s · #B□s · #C□s $)^{1/2}$

$\le M^{3/2}$



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

Thm (Loomis & Whitney, 1949)

G = # cubes in 3D set = Volume of 3D set

$\le$ (area(A shadow) · area(B shadow) · area(C shadow)$)^{1/2}$

$\le M^{3/2}$

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 / points in S given bounds on #points in its images ϕ_C (S), ϕ_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 $H < Z^k$, $\quad \text{rank}(H) \le \Sigma_j\, s_j * \text{rank}(\phi_j(H))$
- 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 (1/2)?

- But first: Can we write it down?
  - Thm: (bad news) Reduces to Hilbert's 10th problem over Q (conjectured to be undecidable)
  - Thm: (good news) Can write it down explicitly in many cases of interest (eg all ϕ_j = {subset of indices})
  - Thm: (good news) Easy to approximate
    - If you miss a constraint, the lower bound may be too large (i.e. s_{HBL} too small) but still worth trying to attain
    - Tarski-decidable to get superset of constraints (may get . s_{HBL} too large)

Is this bound attainable (2/2)?

- Depends on loop dependencies
- Best case: none, or reductions (matmul)
- Thm: When all ϕ_j = {subset of indices}, dual of HBL-LP gives optimal tile sizes:

  HBL-LP: minimize 1^T*s s.t. $s^T*\Delta \geq 1^T$

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

  Then for sequential algorithm, tile i_j by M^{x_j}
- Ex: Matmul: $s = [\ 1/2\ ,\ 1/2\ ,\ 1/2\ ]^T = x$
- Extends to unimodular transforms of indices

Ongoing Work

- Identify more decidable cases
  - Works for any 3 nested loops, or 3 different subscripts
- Automate generation of approximate LPs
- Extend "perfect scaling" results for time and energy by using extra memory
- Have yet to find a case where we cannot attain lower bound – can we prove this?
- Incorporate into compilers

Outline

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

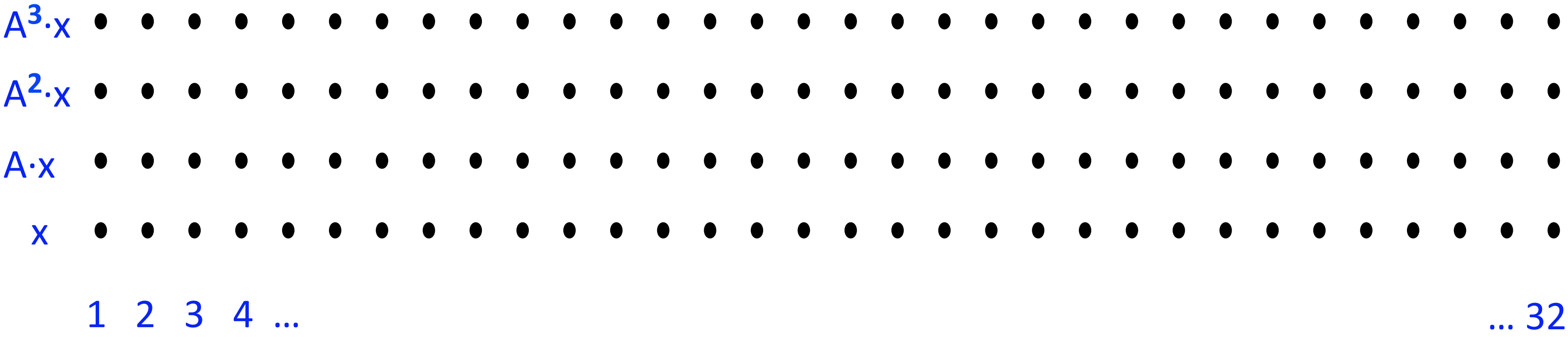
Avoiding Communication in Iterative Linear Algebra

- 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, Stability

Communication Avoiding Kernels:

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

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

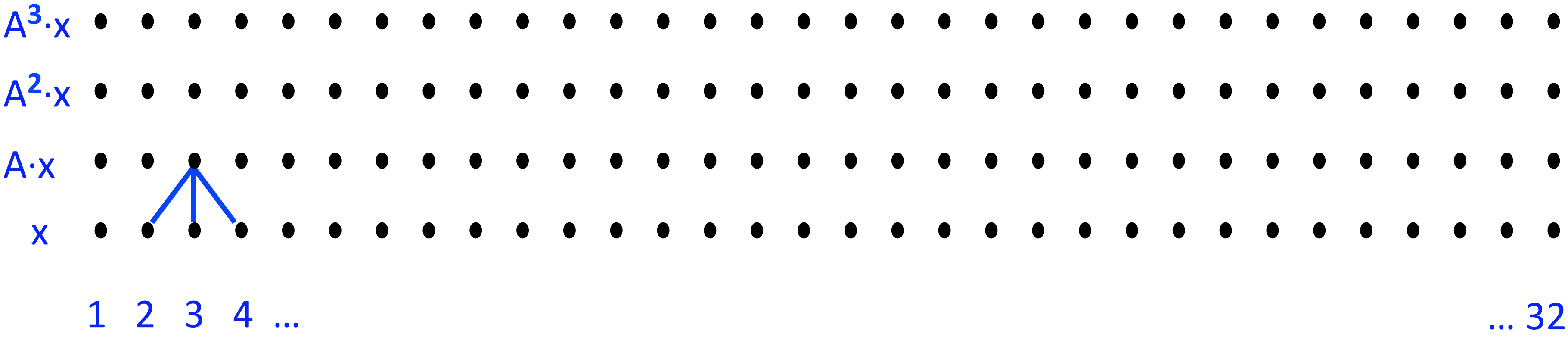


- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:

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

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

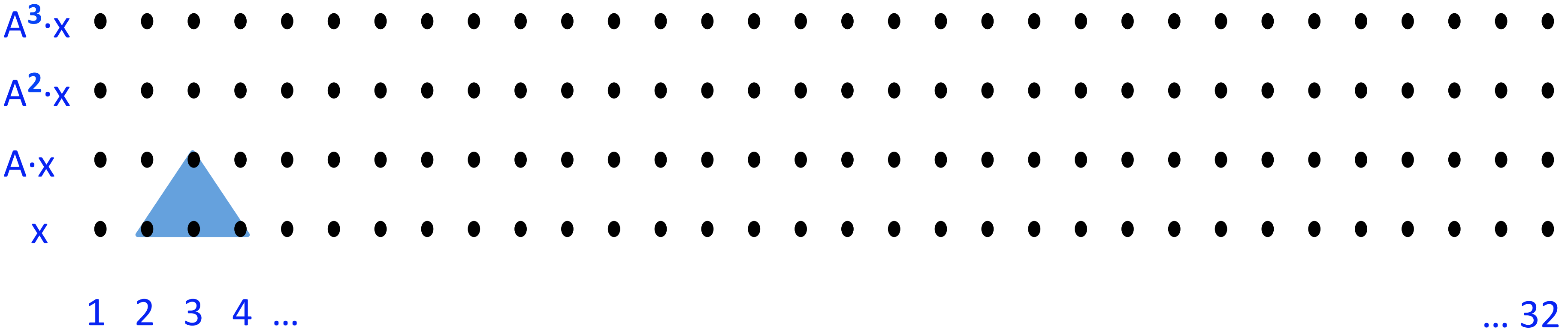


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

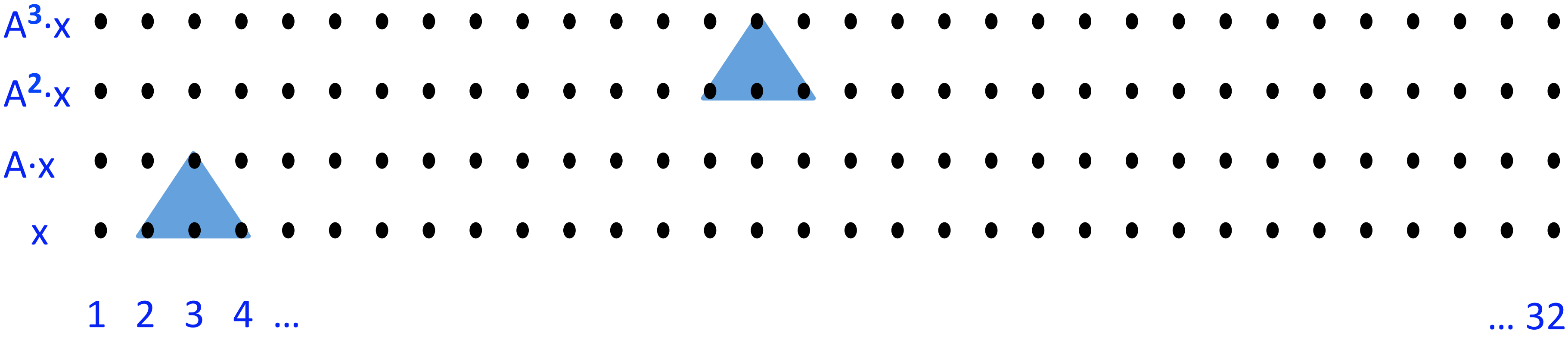


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

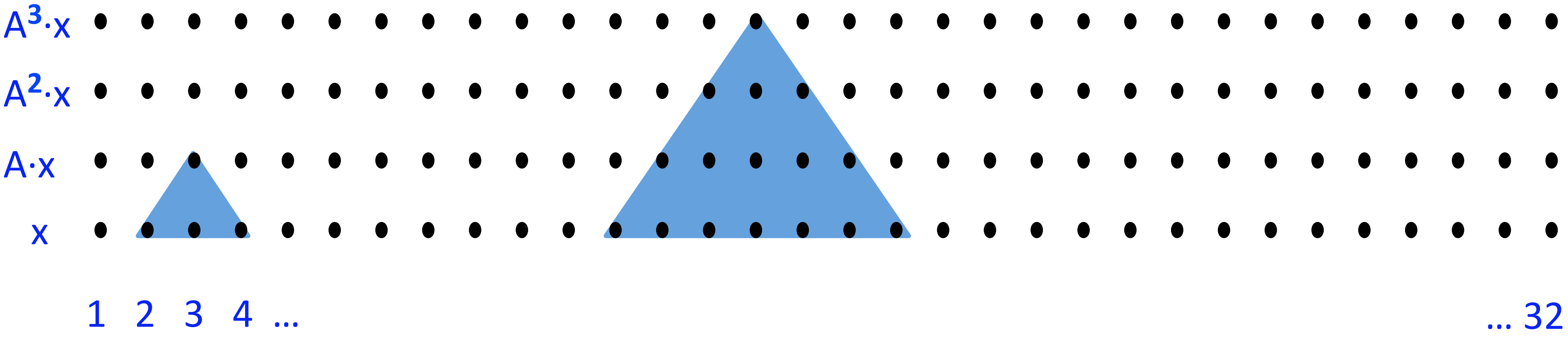


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

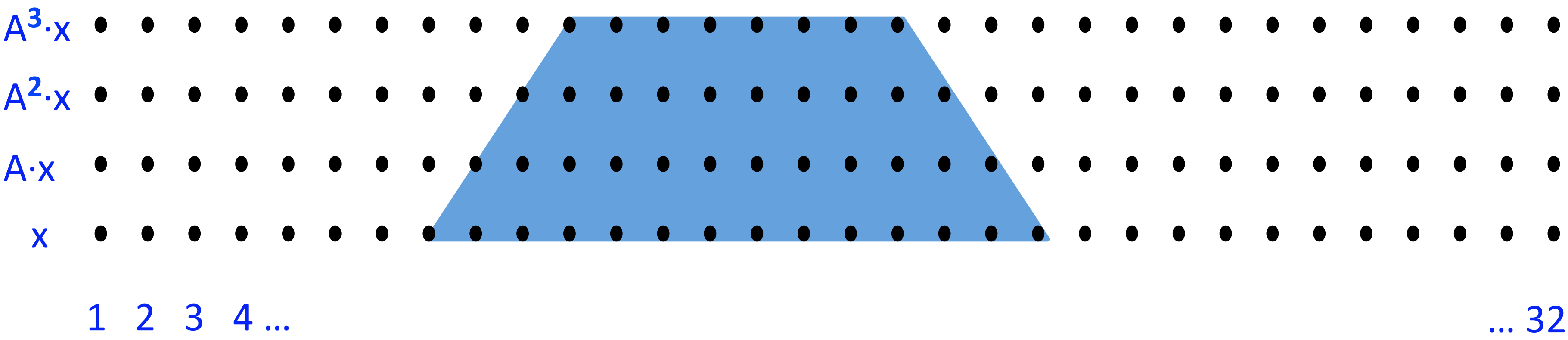


- Example: A tridiagonal, n=32, k=3

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]

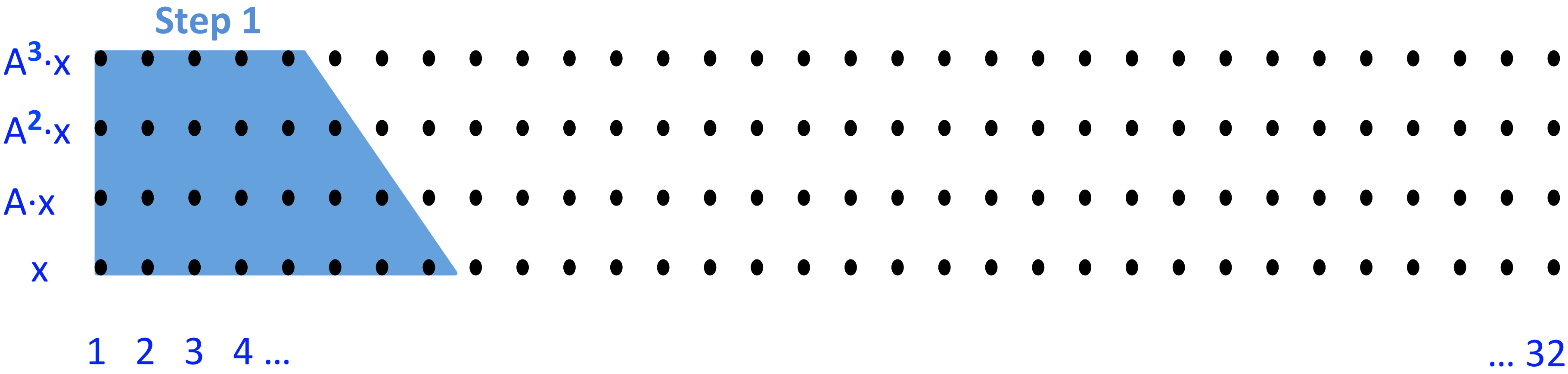


- Example: A tridiagonal, n=32, k=3

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

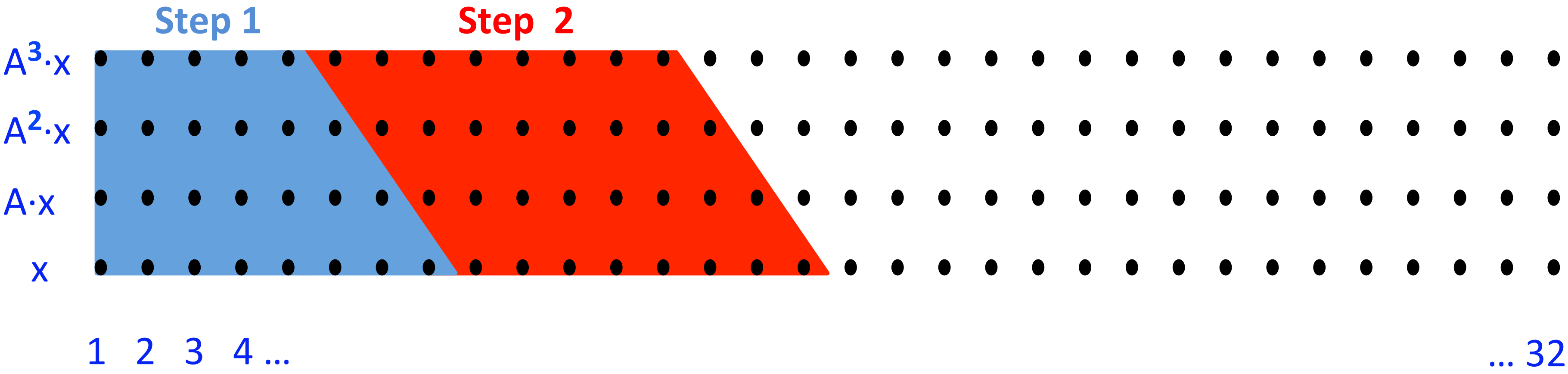


- Example: A tridiagonal, n=32, k=3

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

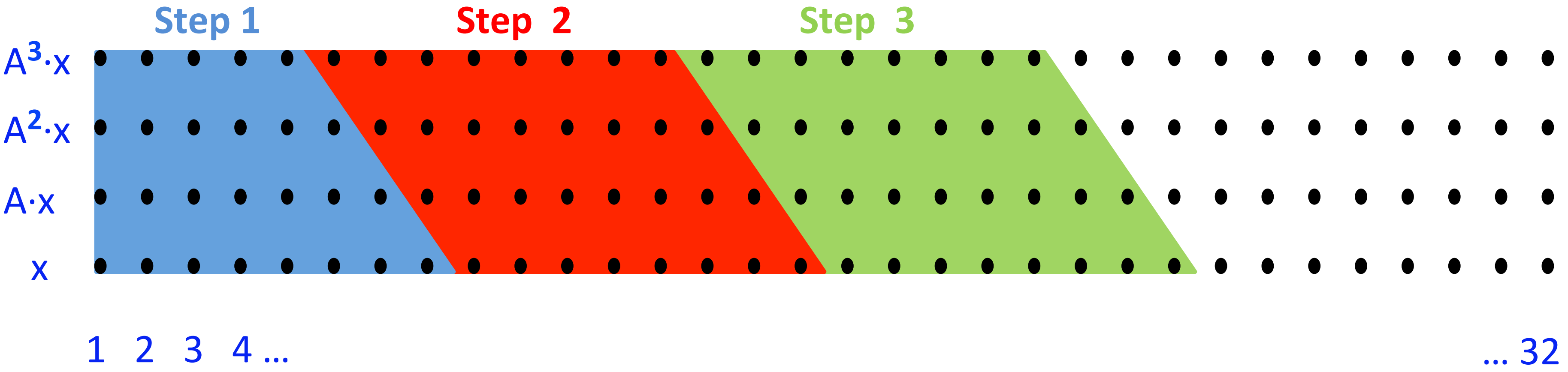


- Example: A tridiagonal, n=32, k=3

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

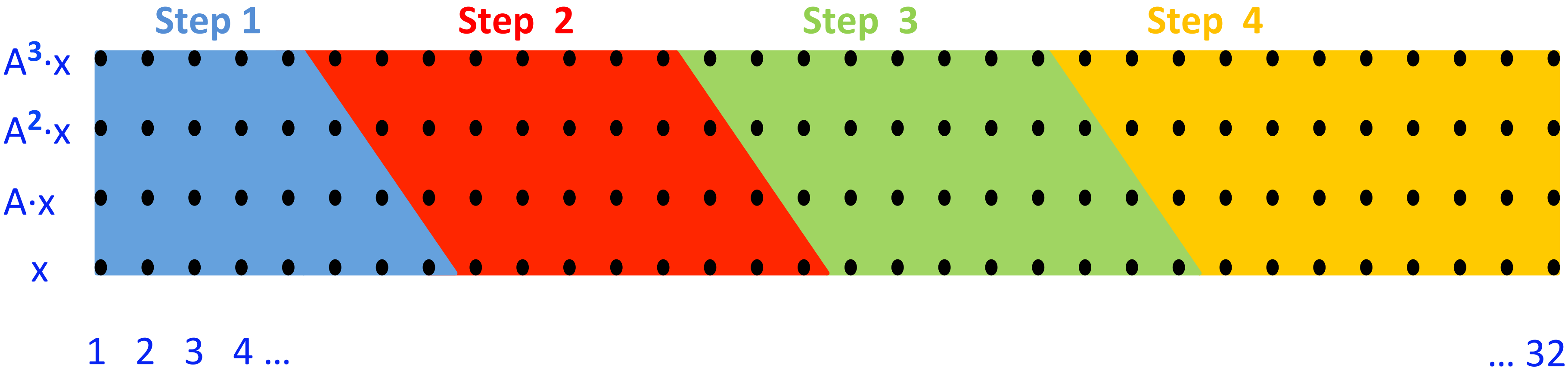


- Example: A tridiagonal, n=32, k=3

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

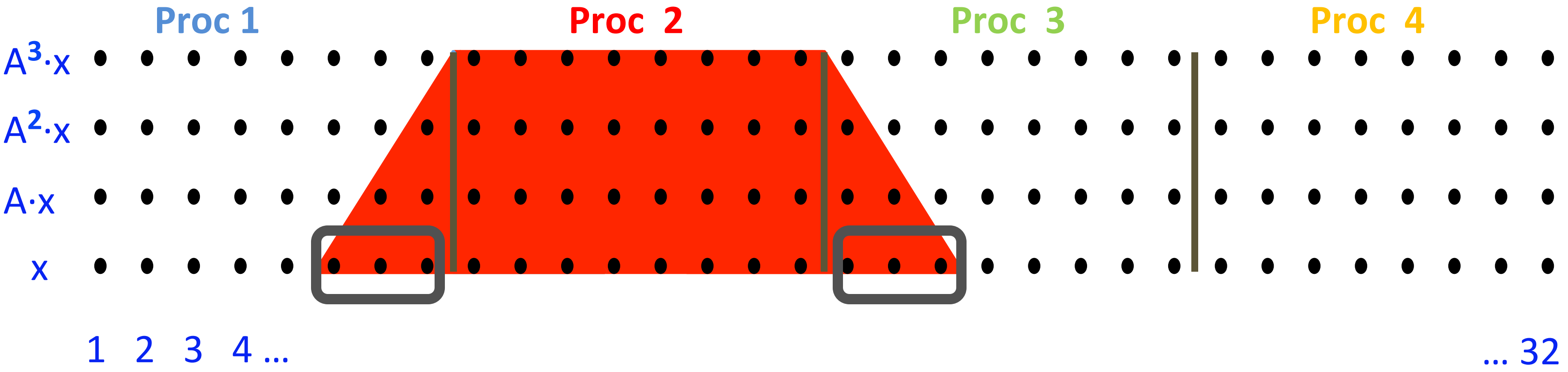


- Example: A tridiagonal, n=32, k=3

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

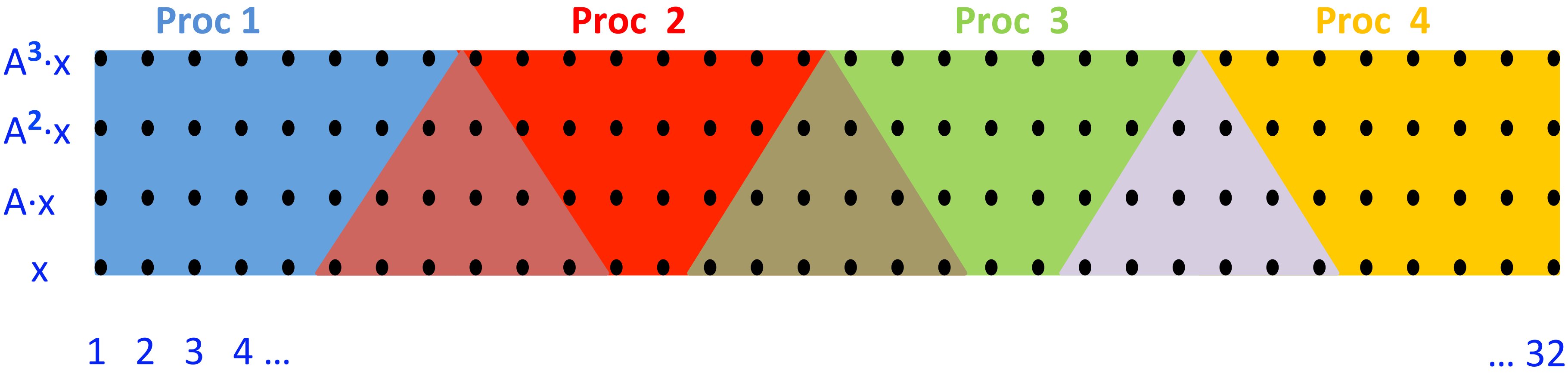


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

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

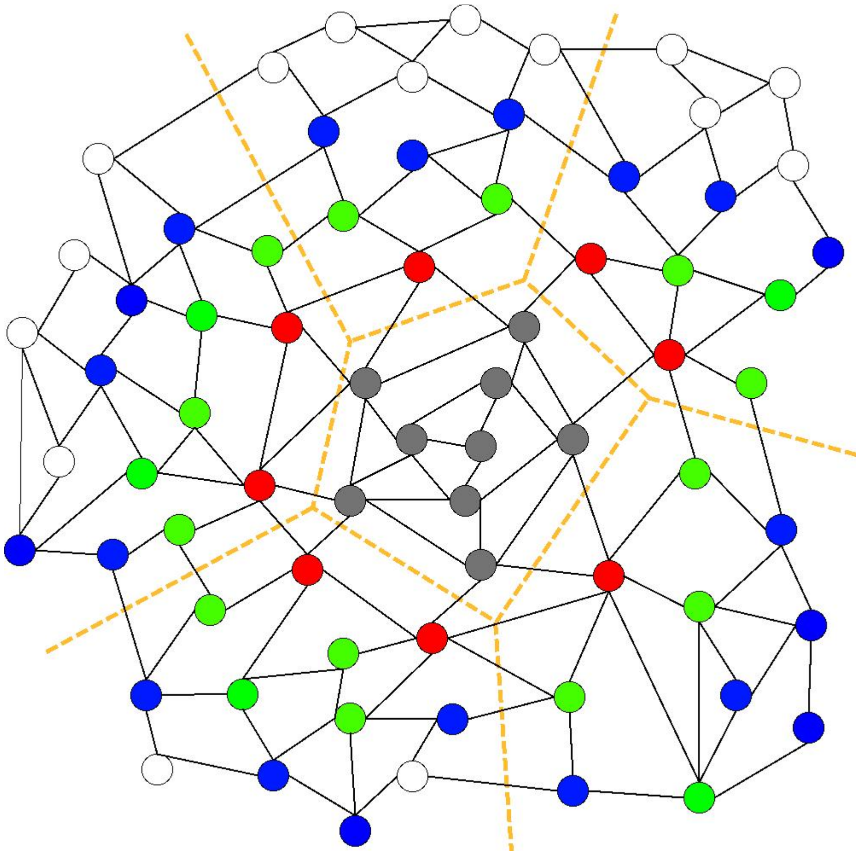
Communication Avoiding Kernels:

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

Same idea works for general sparse matrices

Simple block-row partitioning ➔
(hyper)graph partitioning

Top-to-bottom processing ➔
Traveling Salesman Problem



Minimizing Communication of GMRES to solve Ax=b

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

Standard GMRES

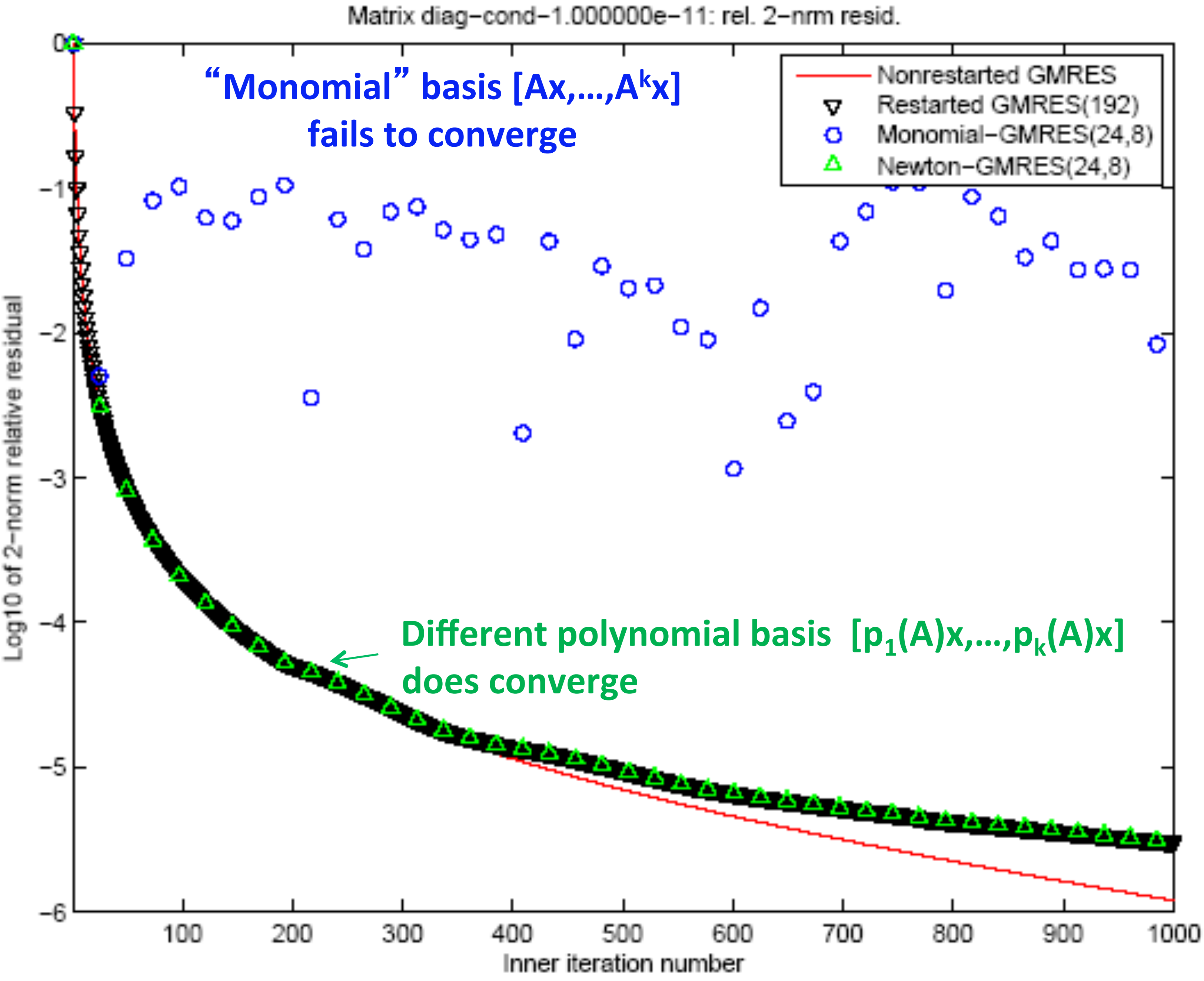
```
for i=1 to k
   w = A · v(i-1)   … SpMV
   MGS(w, v(0),…,v(i-1))
   update v(i), H
endfor
solve LSQ problem with H
```

Communication-avoiding GMRES

```
W = [ v, Av, A^2v, … , 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!



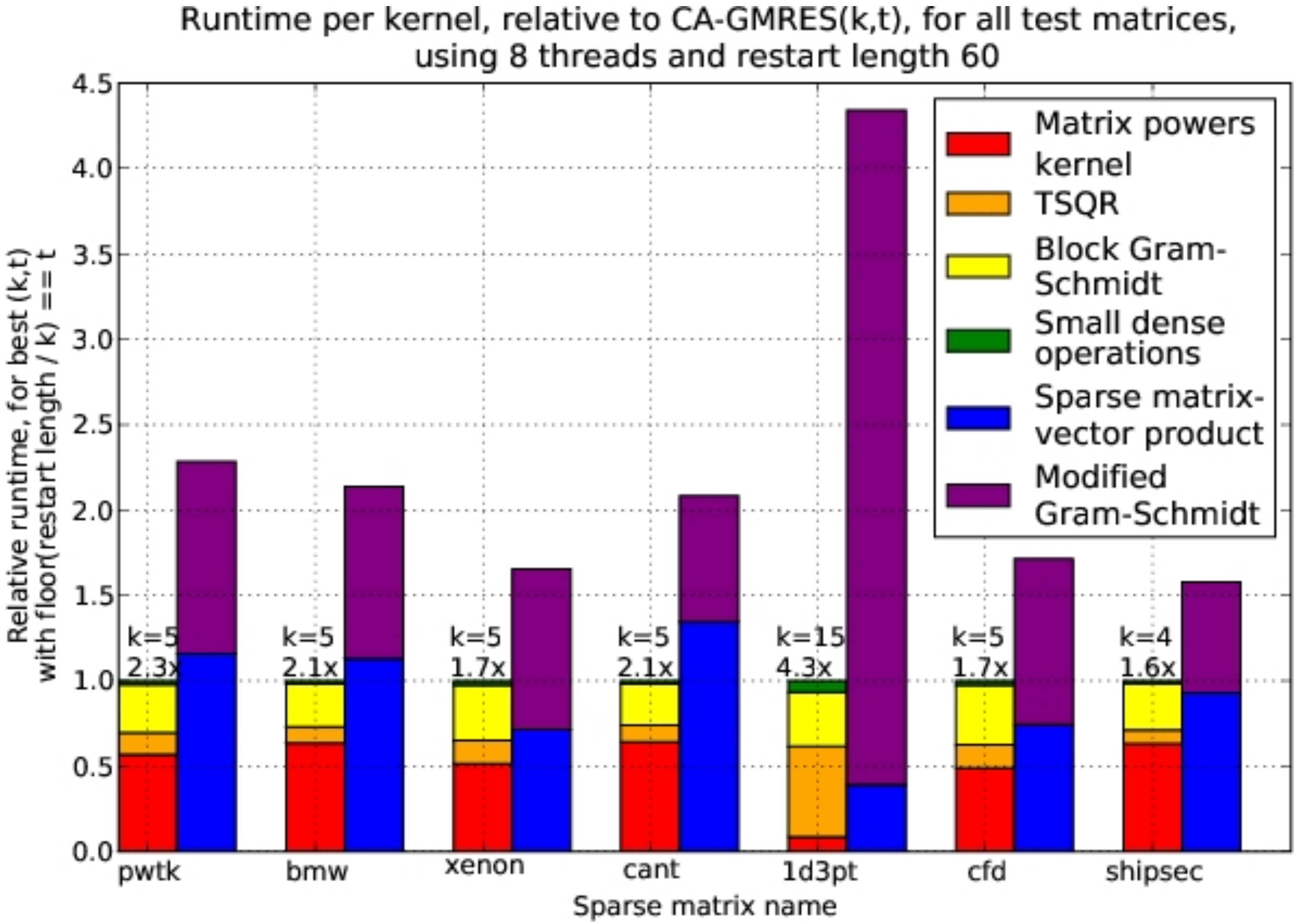
“Monomial” basis $[Ax,\ldots,A^kx]$ fails to converge

Different polynomial basis $[p_1(A)x,\ldots,p_k(A)x]$ does converge

Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

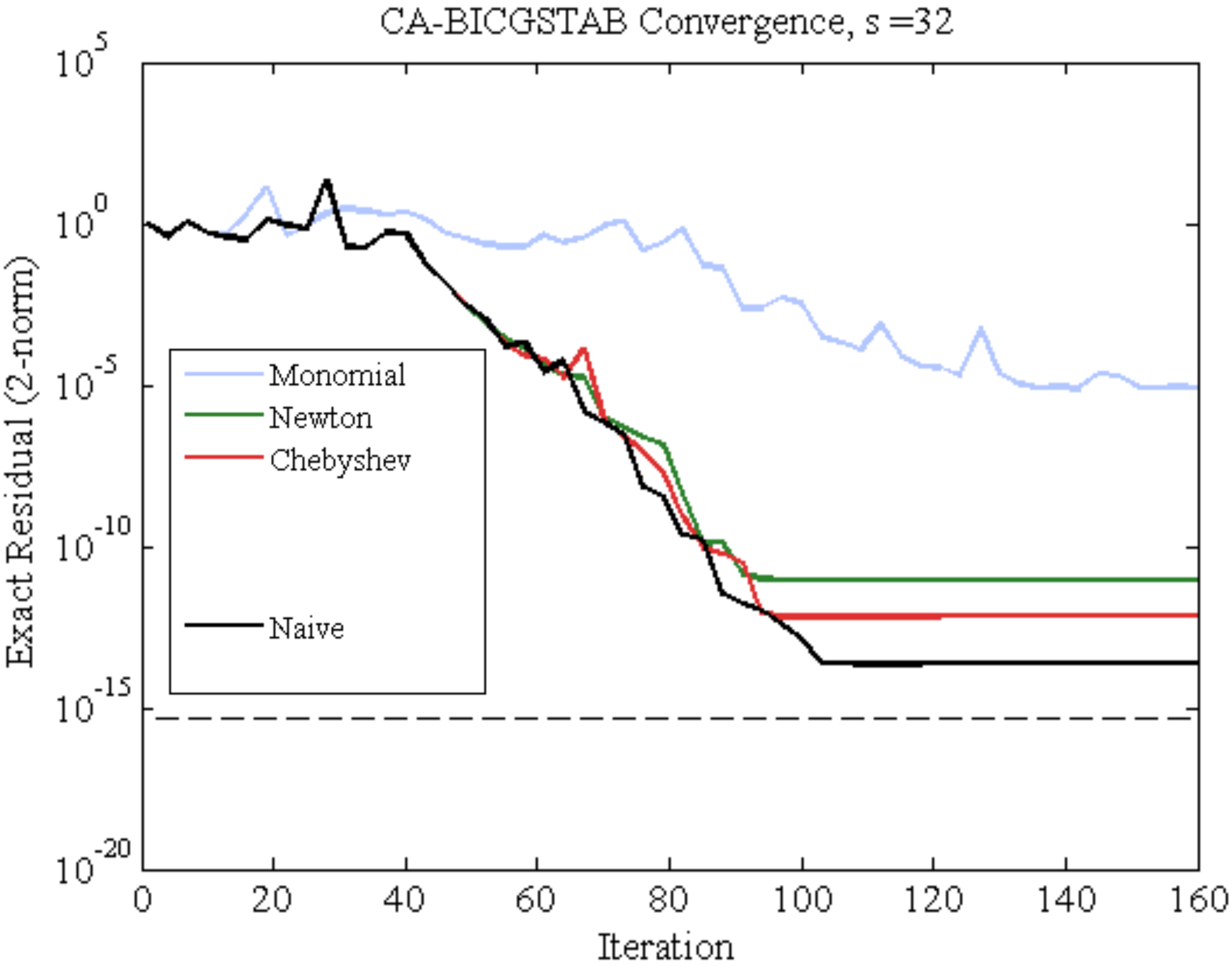
[MHDY09]

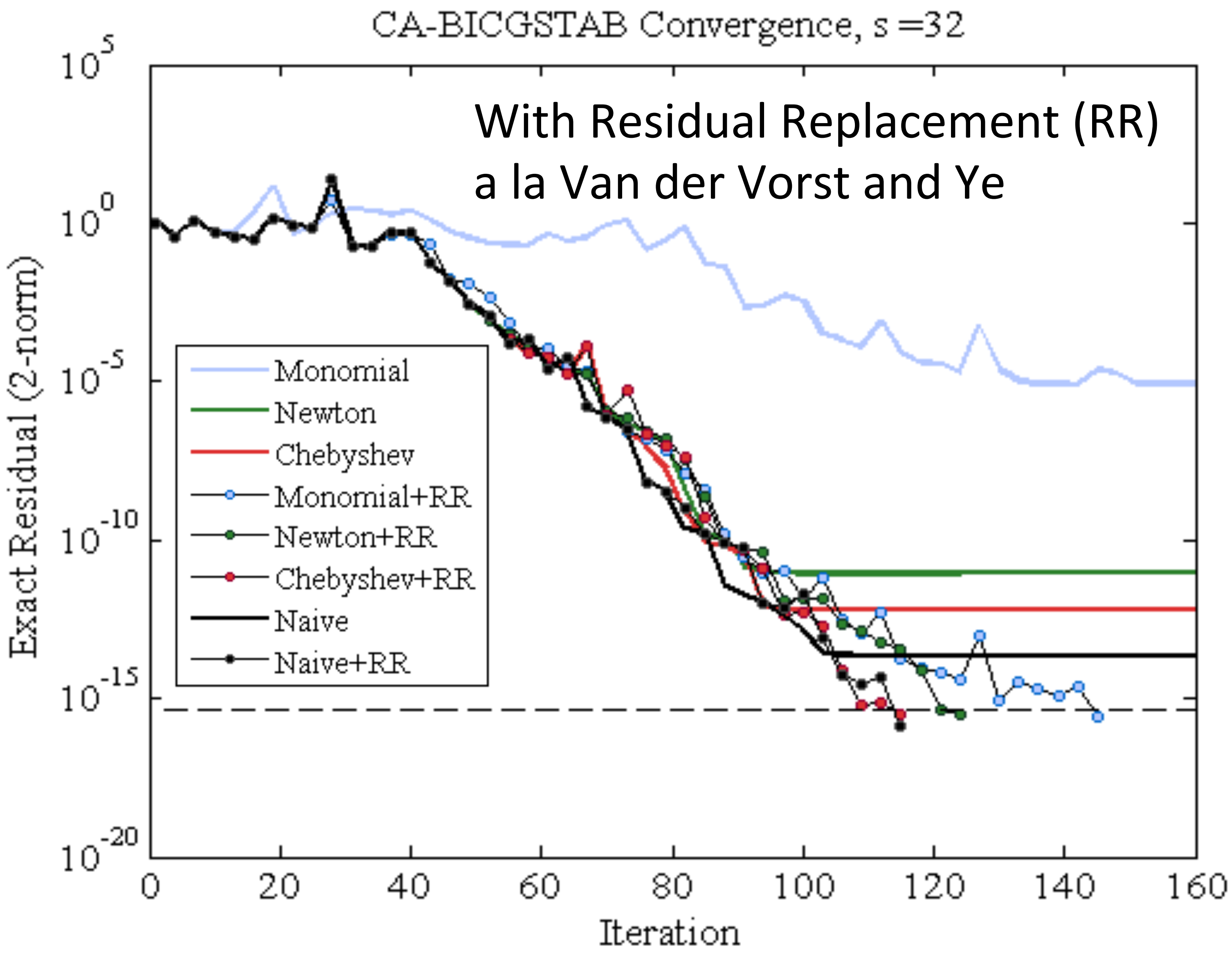


CA-BiCGStab

```
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 $\left\lfloor \frac{s}{2} \right\rfloor - 1$, 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
```

```
Compute $r_0 := b - Ax_0$; $r_0^*$ arbitrary;
$p_0 := r_0$.
For $j = 0, 1, \ldots,$ until convergence Do:
    $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
    $s_j := r_j - \alpha_j Ap_j$
    $\omega_j := (As_j, s_j)/(As_j, As_j)$
    $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
    $r_{j+1} := s_j - \omega_j As_j$
    $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
    $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
EndDo
```


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



With Residual Replacement (RR)
a la Van der Vorst and Ye

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

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"

For more details

- Bebop.cs.berkeley.edu
- CS267 – Berkeley's Parallel Computing Course
  - Live broadcast in Spring 2013
    - www.cs.berkeley.edu/~demmel
  - Prerecorded version planned in Spring 2013
    - www.xsede.org
    - Free supercomputer accounts to do homework!

Summary

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

Don’t Communic…