

Communication-Avoiding Algorithms

Jim Demmel

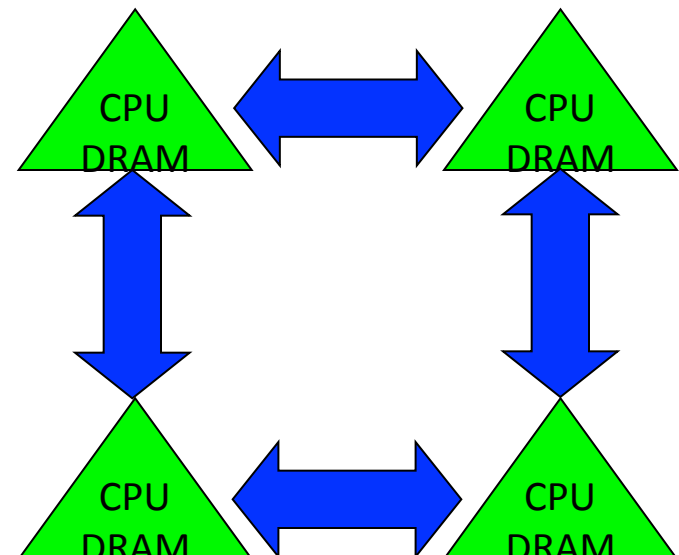
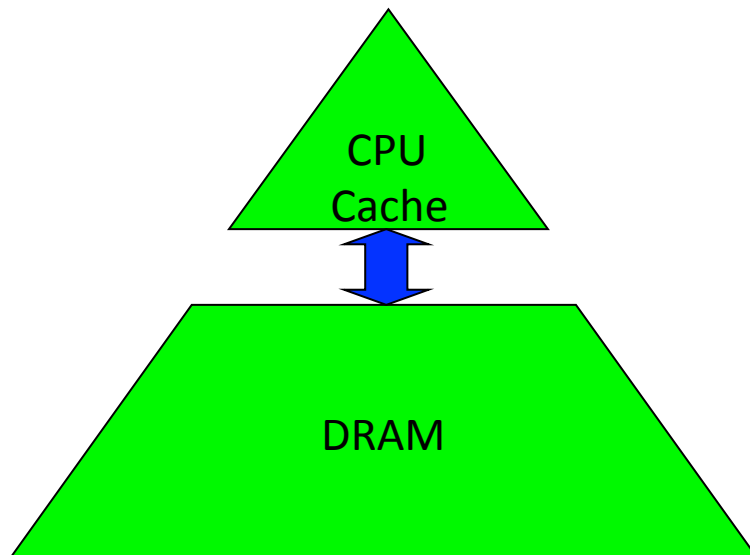
EECS & Math Departments

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).



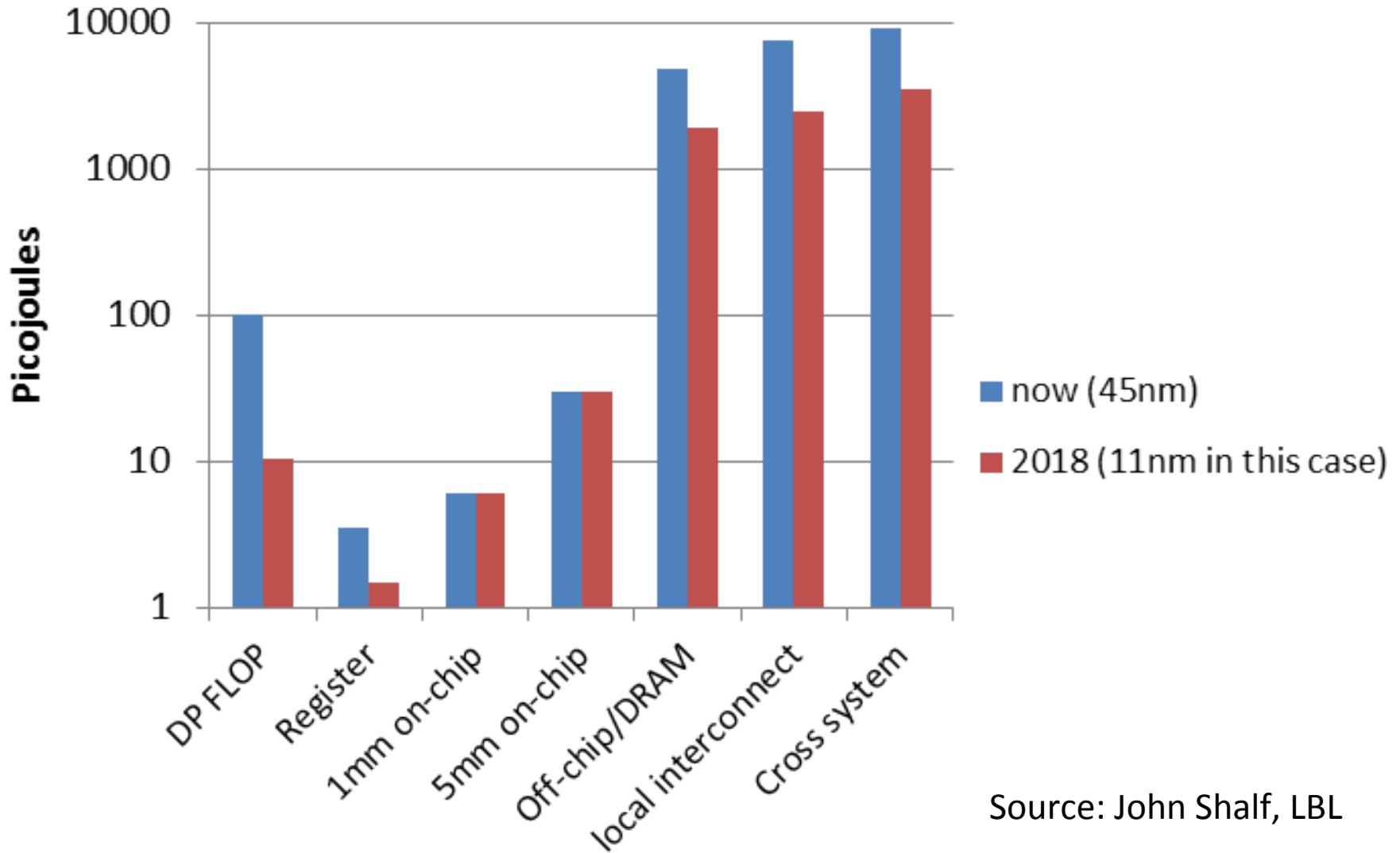
Why avoid communication? (2/3)

- Running time of an algorithm is sum of 3 terms:
 - # flops * time_per_flop
 - # words moved / bandwidth
 - # messages * latency } communication
- Time_per_flop \ll 1/ bandwidth \ll latency
 - Gaps growing exponentially with time [FOOSC]

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

- Avoid communication to save time

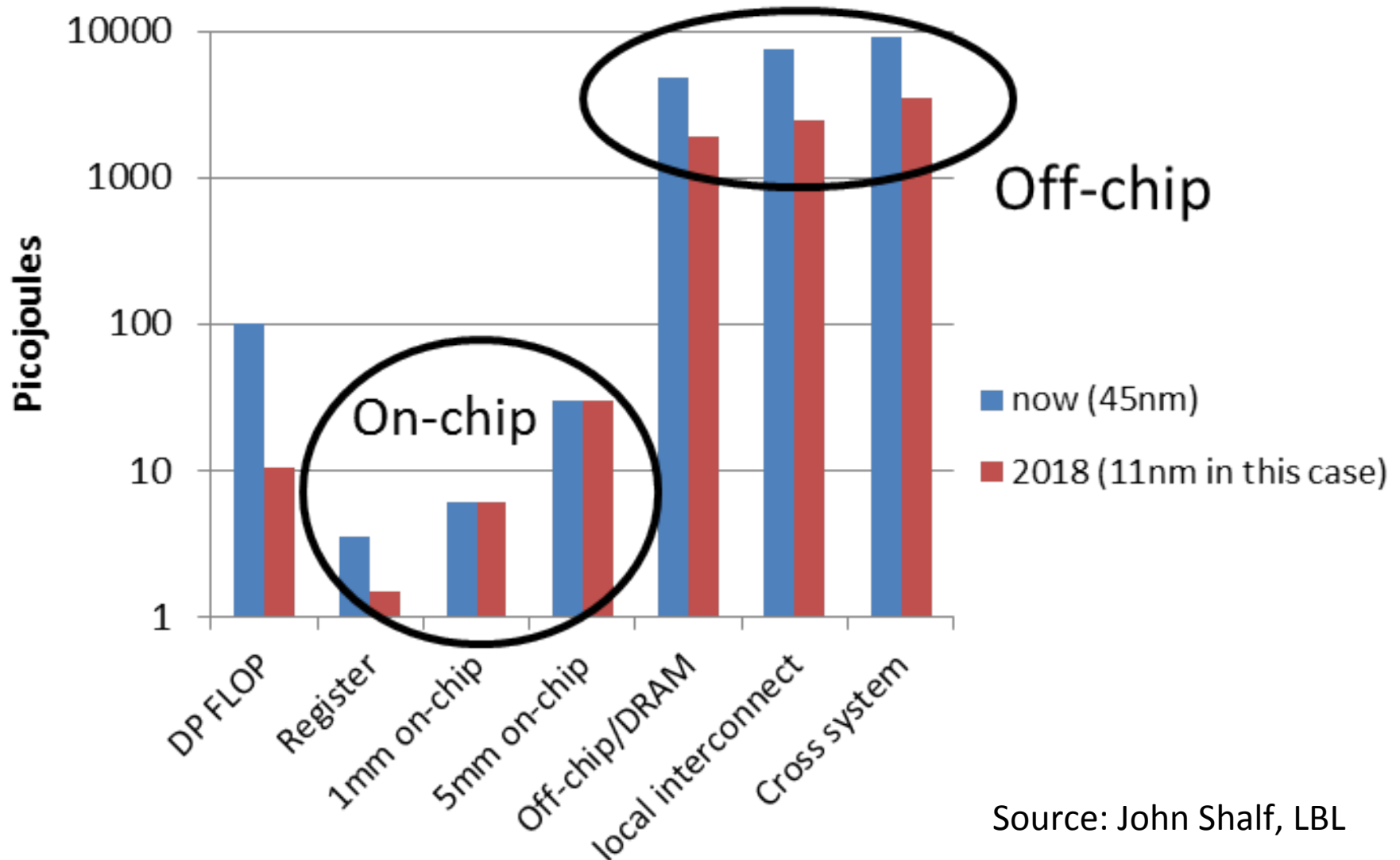
Why Minimize Communication? (3/3)



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 \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

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
 - 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, Armando Fox, David Gleich, Laura Grigori, Ming Gu, Mike Heroux, Mark Hoemmen, Olga Holtz, Kurt Keutzer, Julien Langou, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang, Kathy Yelick
- Michael Anderson, Grey Ballard, Austin Benson, Abhinav Bhatele, Aydin Buluc, Erin Carson, Maryam Dehnavi, Michael Driscoll, Evangelos Georganas, Nicholas Knight, Penporn Koanantakool, Ben Lipshitz, Marghoob Mohiyuddin, Oded Schwartz, Edgar Solomonik
- Other members of ParLab, BEBOP, CACHE, EASI, FASTMath, MAGMA, PLASMA, TOPS projects
- Thanks to NSF, DOE, UC Discovery, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle
- bebop.cs.berkeley.edu

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
 - 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, $Ax = \lambda x$, SVD, etc
 - 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 “n³-like” linear algebra

- Let M = “fast” memory size (per processor)

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

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

Lower bound for all “n³-like” linear algebra

- Let M = “fast” memory size (per processor)

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

$$\#messages_sent \geq \#words_moved / largest_message_size$$

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

Lower bound for all “n³-like” linear algebra

- Let M = “fast” memory size (per processor)

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

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

- 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

Summary of dense parallel algorithms attaining communication lower bounds

- Assume $n \times n$ 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?

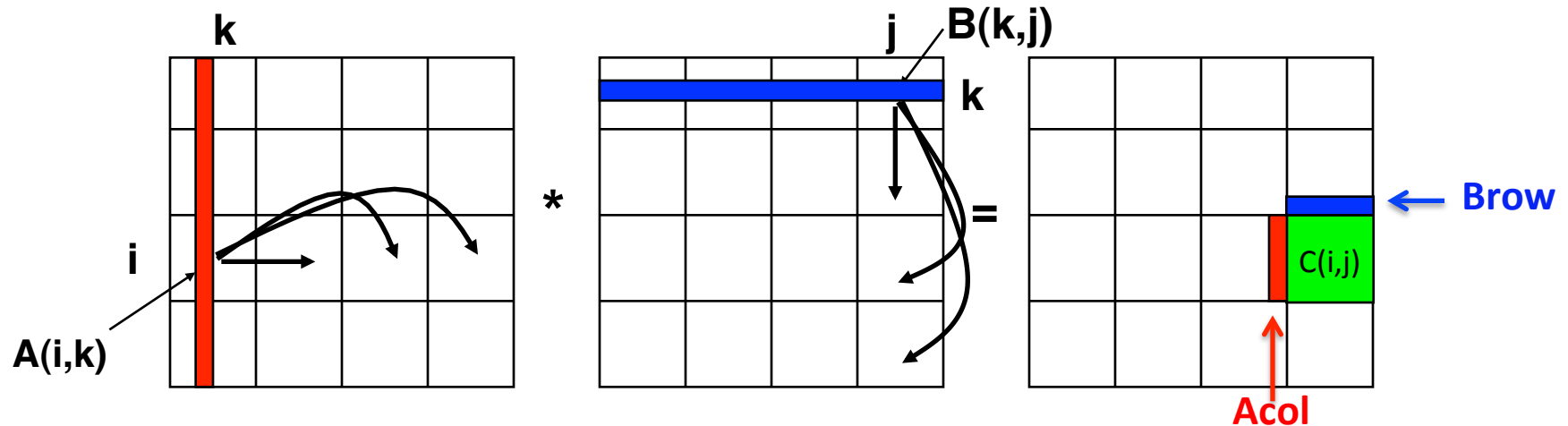
Can we do better?

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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
 - 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– $n \times n$ matmul on $P^{1/2} \times P^{1/2}$ grid (nearly) optimal using minimum memory $M=O(n^2/P)$

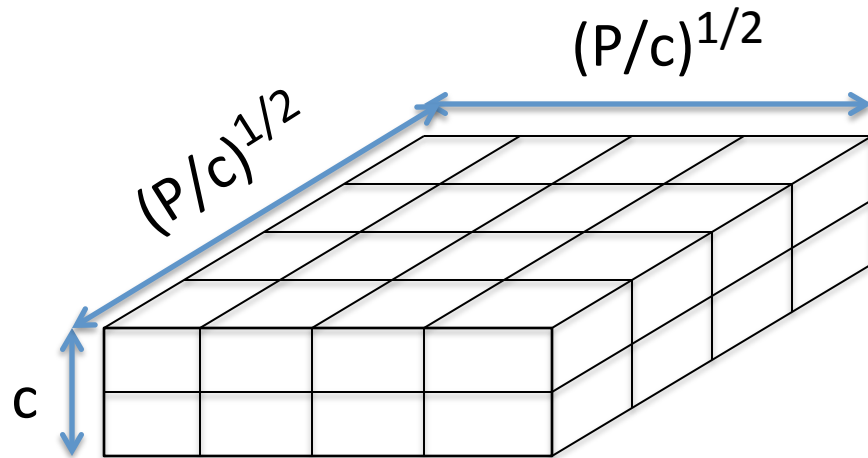


Using more than the minimum memory

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

2.5D Matrix Multiplication

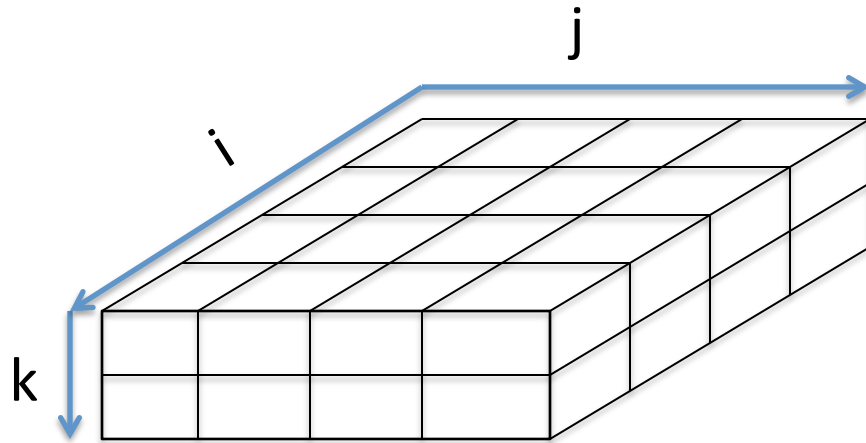
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid



Example: $P = 32$, $c = 2$

2.5D Matrix Multiplication

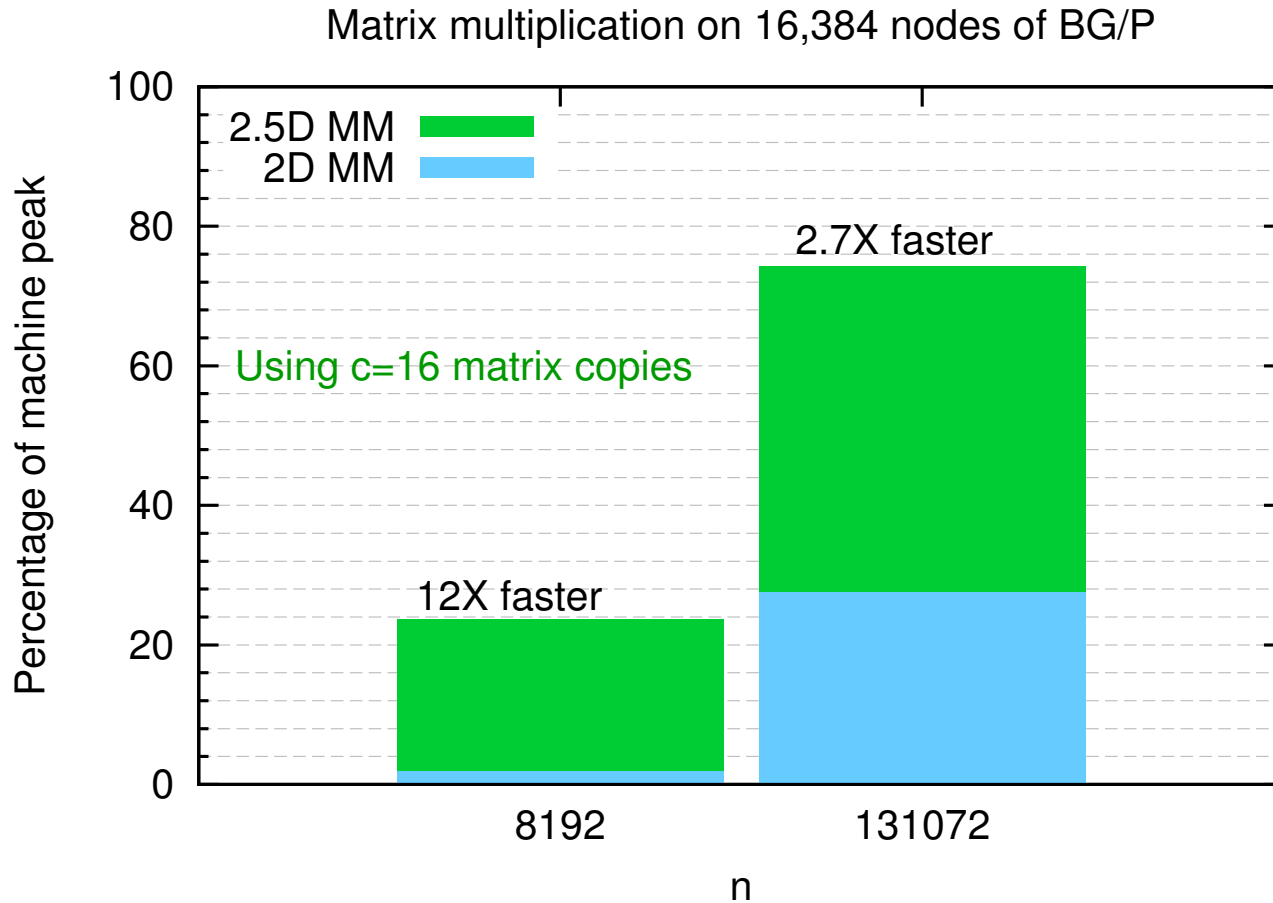
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid



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

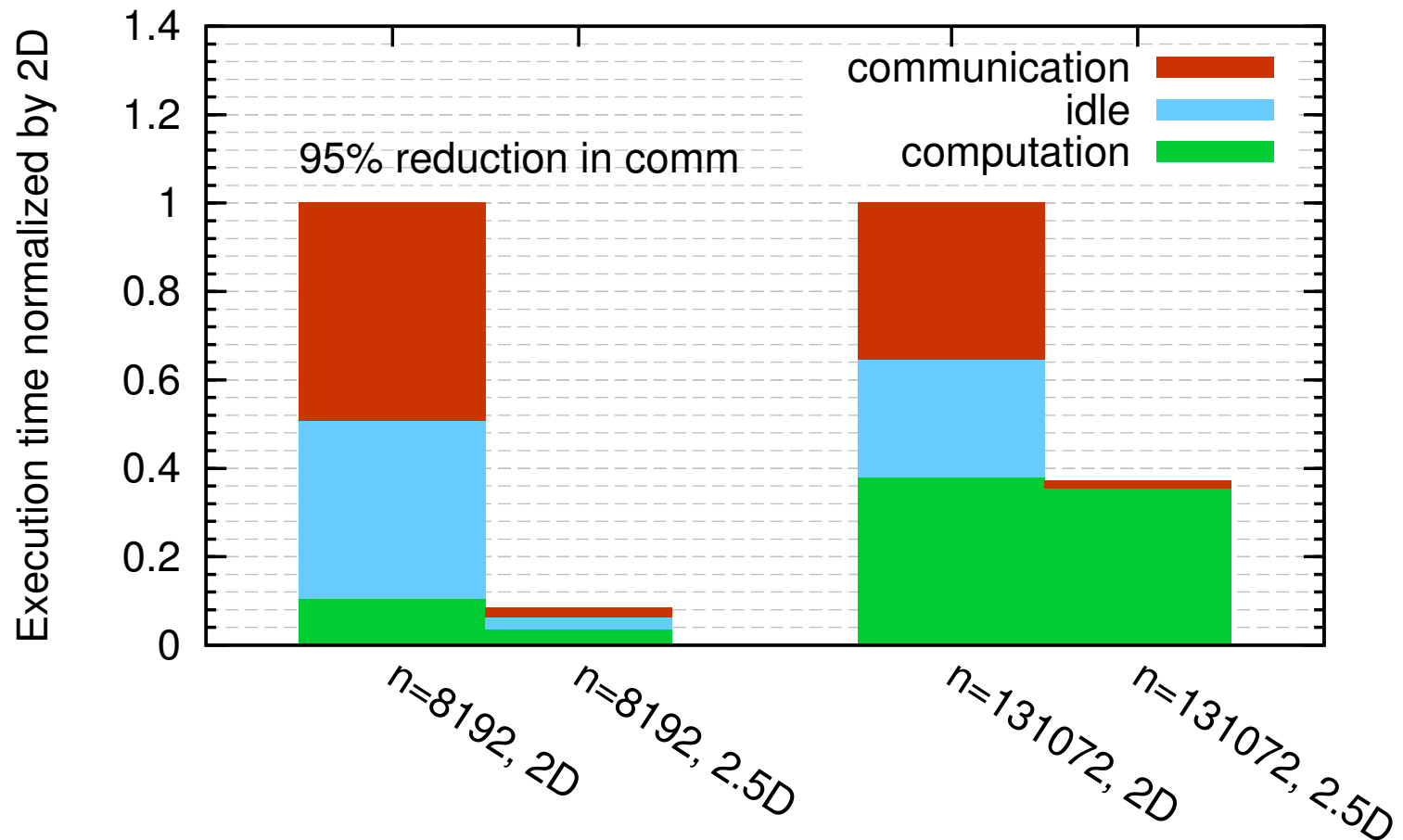
2.5D Matmul on BG/P, 16K nodes / 64K cores



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

c = 16 copies

Matrix multiplication on 16,384 nodes of BG/P



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 $c \rightarrow$ total memory increases by a factor of c
- Notation for timing model:
 - $\gamma_T, \beta_T, \alpha_T =$ 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:
 - $\gamma_E, \beta_E, \alpha_E =$ joules for same operations
 - $\delta_E =$ joules per word of memory used per sec
 - $\epsilon_E =$ joules per sec for leakage, etc.
- $E(cP) = cP \{ n^3/(cP) [\gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})] + \delta_E MT(cP) + \epsilon_E T(cP) \}$
 $= E(P)$
- 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 ($c>1$)
 - 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
 - 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 =
 $\Omega(M(n/M^{1/2})^{\lg 7}/P)$

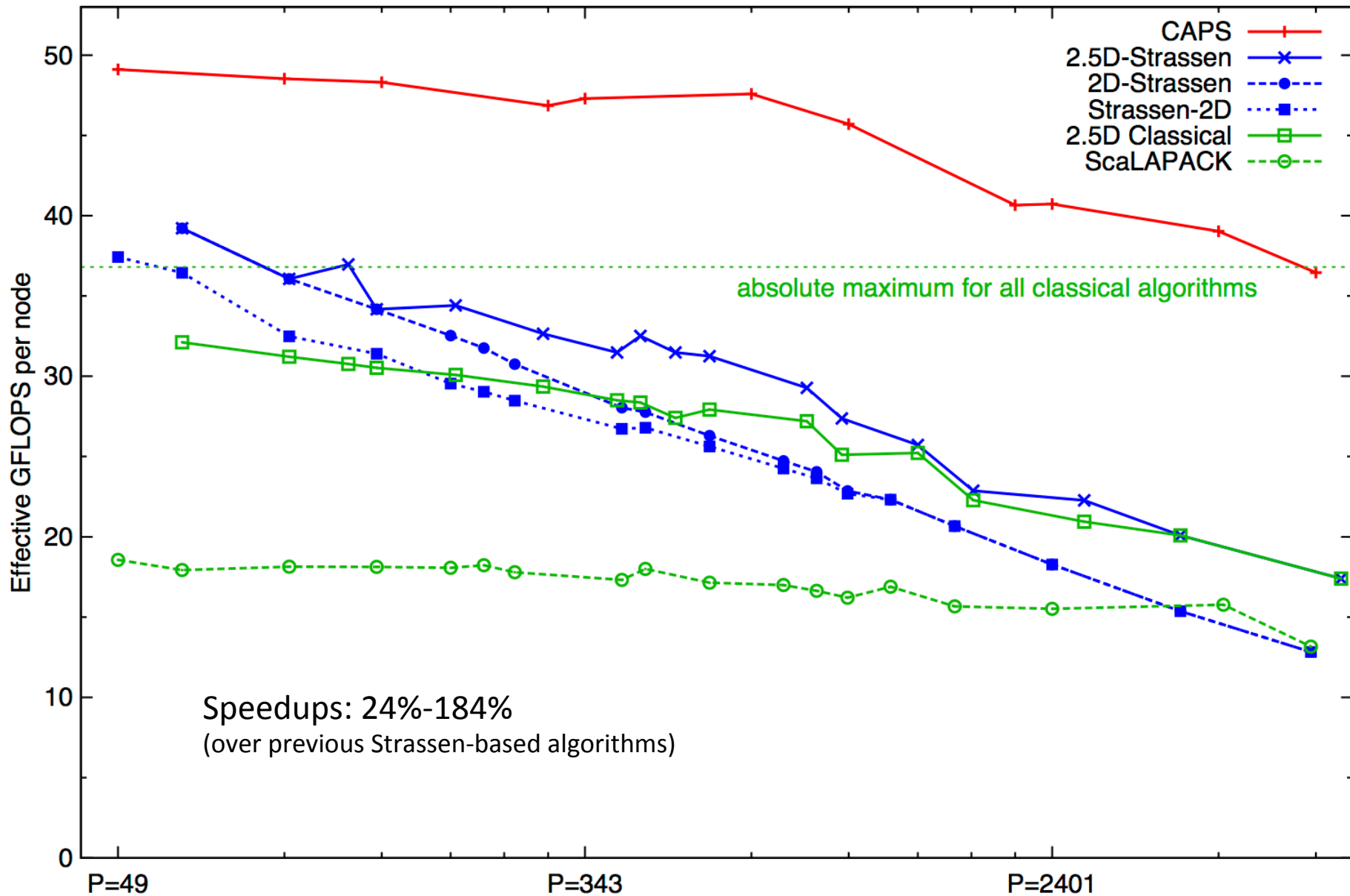
Strassen-like
 $O(n^\omega)$ matmul:

#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?

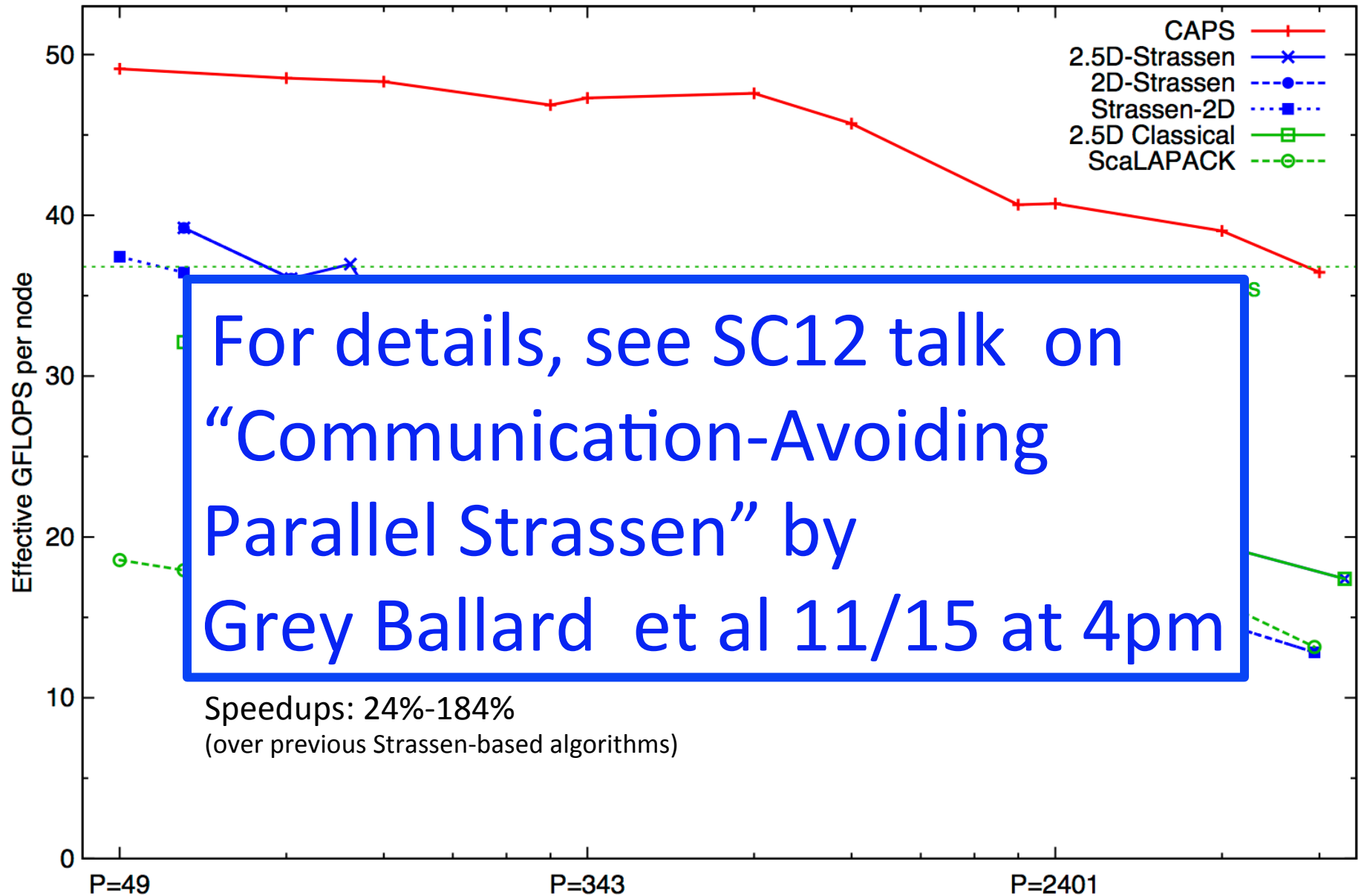
Strong Scaling of Strassen and other Matmuls

Franklin (Cray XT4) n = 94080



Strong Scaling of Strassen and other Matmuls

Franklin (Cray XT4) n = 94080



Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
 - 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 $i_1 = 1:b:n$, for $j_1 = 1:b:n$, for $k_1 = 1:b:n$
for $i_2 = 0:b-1$, for $j_2 = 0:b-1$, for $k_2 = 0:b-1$
 $i=i_1+i_2$, $j = j_1+j_2$, $k = k_1+k_2$
 $C(i,j)+=A(i,k)*B(k,j)$ } $b \times b$ matmul
- Thm: Picking $b = M^{1/2}$ attains lower bound:
 $\#words_moved = \Omega(n^3/M^{1/2})$
- Where does $1/2$ come from?

New Thm applied to Matmul

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

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

- Solve LP for $x = [x_i, x_j, x_k]^T$: $\max \mathbf{1}^T x$ s.t. $\Delta x \leq \mathbf{1}$
 - Result: $x = [1/2, 1/2, 1/2]^T$, $\mathbf{1}^T x = 3/2 = s$
- Thm: #words_moved = $\Omega(n^3/M^{s-1}) = \Omega(n^3/M^{1/2})$
 Attained by block sizes $M^{x_i}, M^{x_j}, M^{x_k} = M^{1/2}, M^{1/2}, M^{1/2}$

New Thm applied to Direct N-Body

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

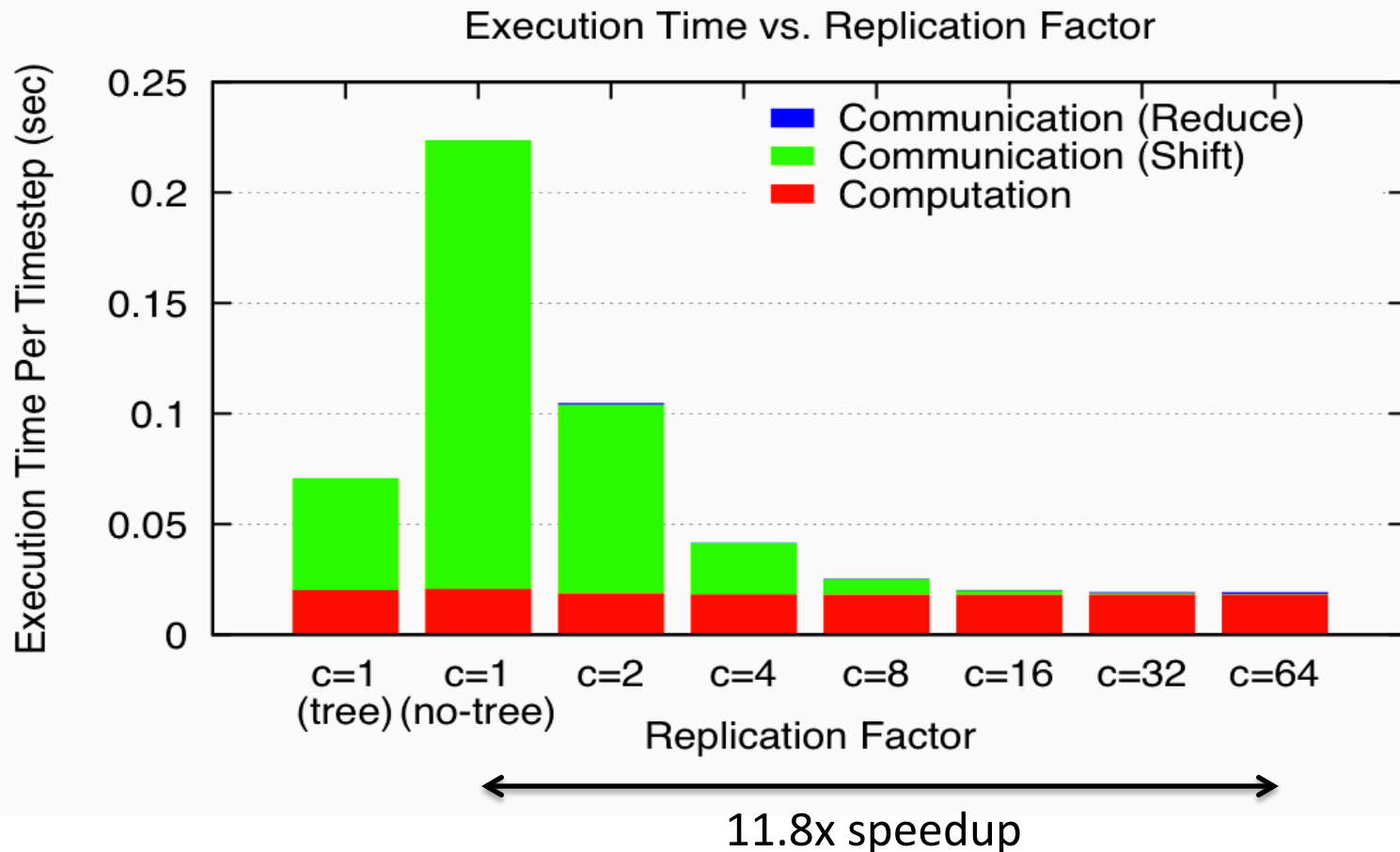
$$\Delta = \begin{matrix} & \begin{matrix} i & j \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{matrix} F \\ P(i) \\ P(j) \end{matrix} \end{matrix}$$

- Solve LP for $x = [x_i, x_j]^T$: $\max \mathbf{1}^T x$ s.t. $\Delta x \leq \mathbf{1}$
 - Result: $x = [1, 1]$, $\mathbf{1}^T x = 2 = s$
- Thm: $\#words_moved = \Omega(n^2/M^{s-1}) = \Omega(n^2/M^1)$
 Attained by block sizes $M^{x_i}, M^{x_j} = M^1, M^1$

N-Body Speedups on IBM-BG/P (Intrepid)

8K cores, 32K particles

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



New Thm applied to Random Code

- for $i_1=1:n$, for $i_2=1:n$, ... , for $i_6=1:n$
 - $A_1(i_1,i_3,i_6) += \text{func}_1(A_2(i_1,i_2,i_4),A_3(i_2,i_3,i_5),A_4(i_3,i_4,i_6))$
 - $A_5(i_2,i_6) += \text{func}_2(A_6(i_1,i_4,i_5),A_3(i_3,i_4,i_6))$

- Record array indices in matrix Δ

$$\Delta = \begin{matrix} & \begin{matrix} i_1 & i_2 & i_3 & i_4 & i_5 & i_6 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} & \begin{matrix} A_1 \\ A_2 \\ A_3 \\ A_3,A_4 \\ A_5 \\ A_6 \end{matrix} \end{matrix}$$

- Solve LP for $x = [x_1, \dots, x_7]^T$: $\max \mathbf{1}^T x$ s.t. $\Delta x \leq \mathbf{1}$
 - Result: $x = [2/7, 3/7, 1/7, 2/7, 3/7, 4/7]$, $\mathbf{1}^T x = 15/7 = s$
- Thm: $\# \text{words_moved} = \Omega(n^6 / M^{s-1}) = \Omega(n^6 / M^{8/7})$
 Attained by block sizes $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

Summary and Ongoing Work

- Communication lower bounds extend to any program accessing arrays with subscripts that are “linear functions” of loop indices
 - Ex: $A(i_1+2*i_2, i_3-i_2)$, $B(\text{pointer}(i_4+i_5))$, ...
 - Sparsity, indirect addressing, parallelism all included
- When can we write down LP for lower bound explicitly?
 - General case hard: Hilbert’s 10th problem over rationals
 - Works for many special cases
 - Ex: subscripts are subsets of loop indices
 - Can always approximate lower bound
- When can we attain lower bound?
 - Works for many special cases (if loop dependencies permit)
 - Have yet to find a case where we cannot attain lower bound
 - Can we prove this?
- Can extend “perfect scaling” results for time and energy
- Incorporate into compilers

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
 - 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**

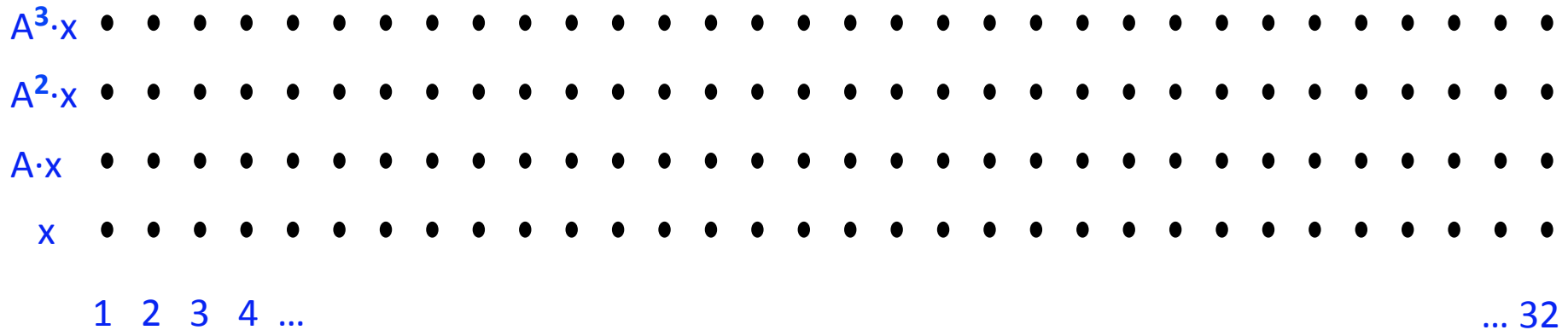
Avoiding Communication in Iterative Linear Algebra

- k -steps of iterative solver for sparse $Ax=b$ or $Ax=\lambda x$
 - Does k SpMV's 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 : $[Ax, A^2x, \dots, A^kx]$

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

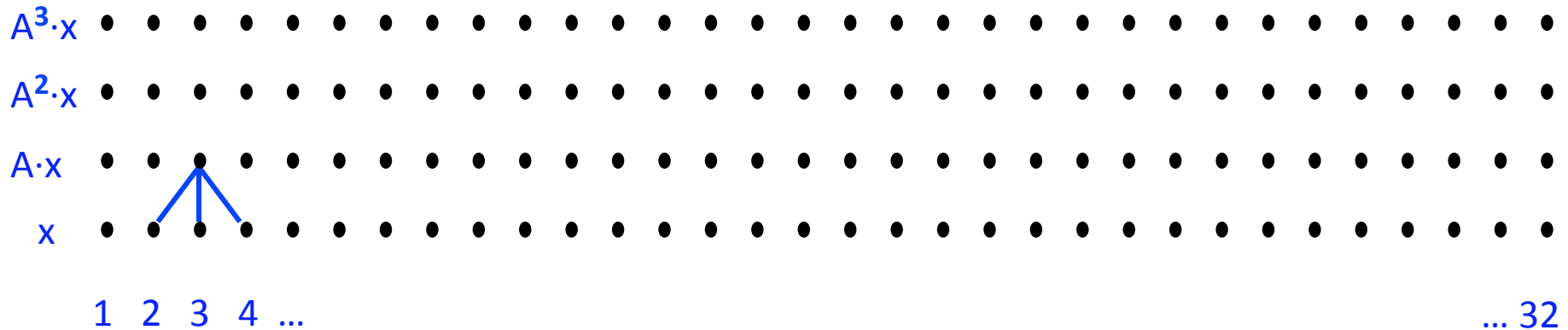


- Example: A tridiagonal, $n=32$, $k=3$
- Works for any “well-partitioned” A

Communication Avoiding Kernels:

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

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

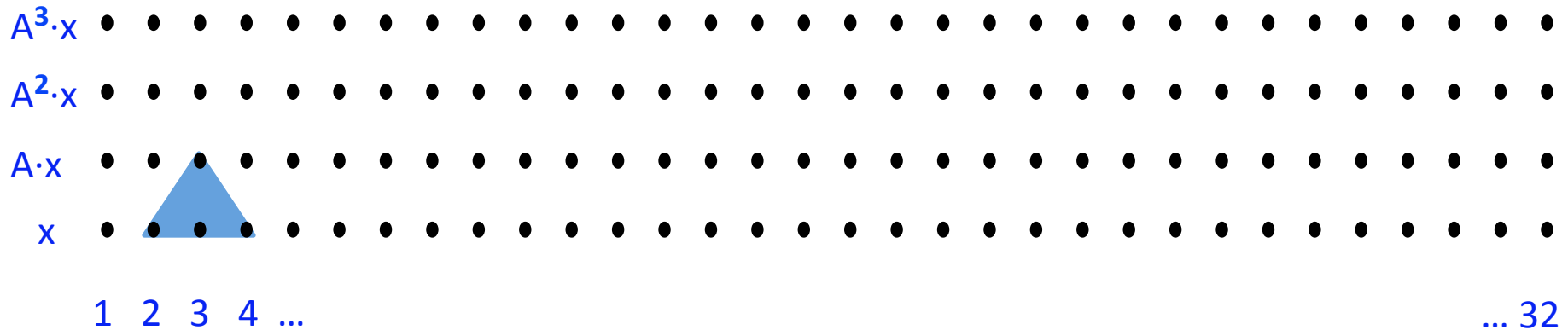


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

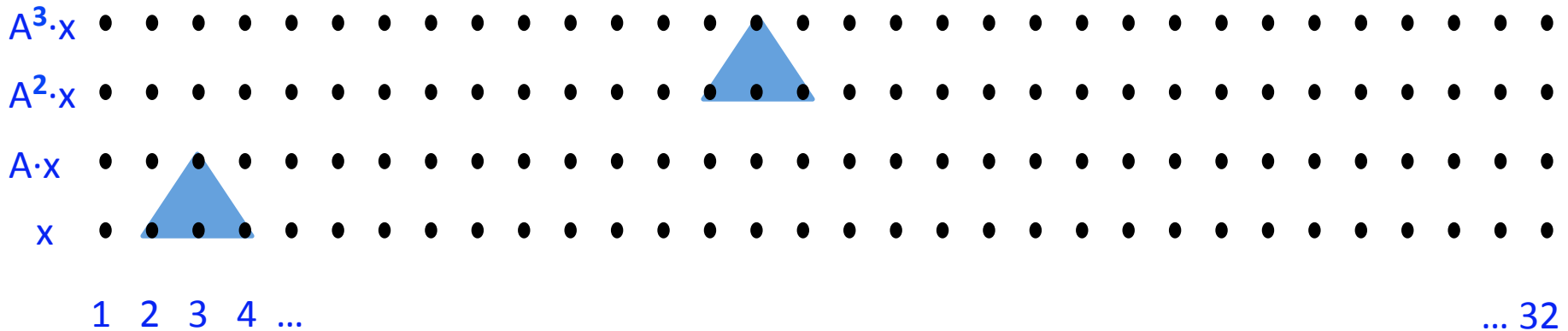


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

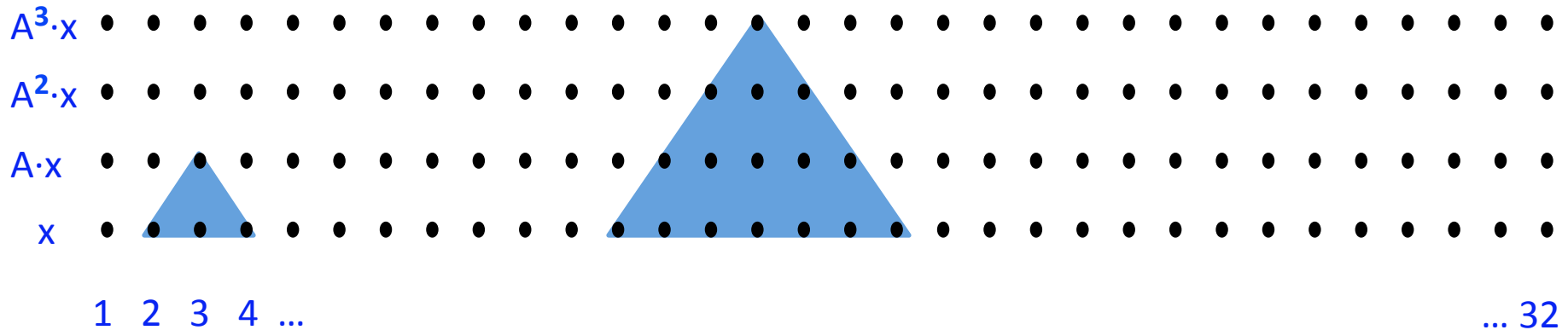


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

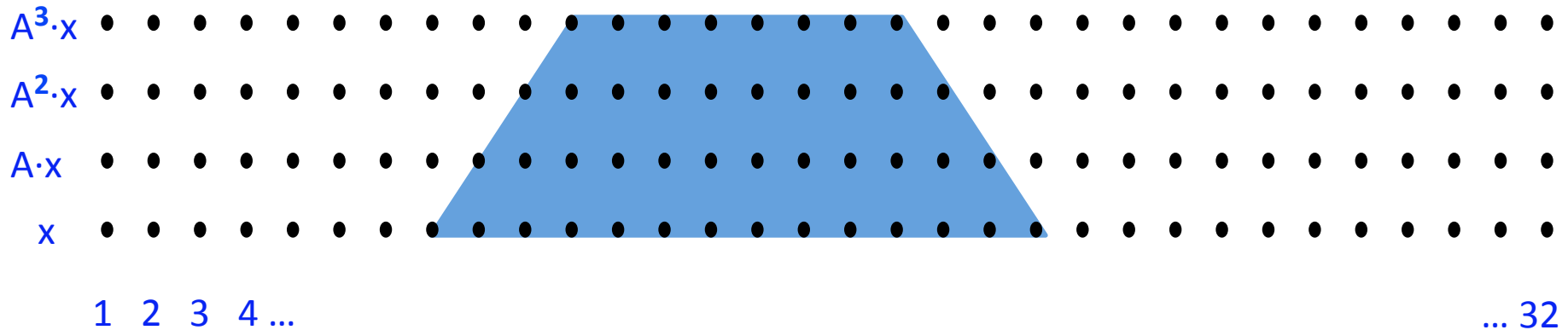


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

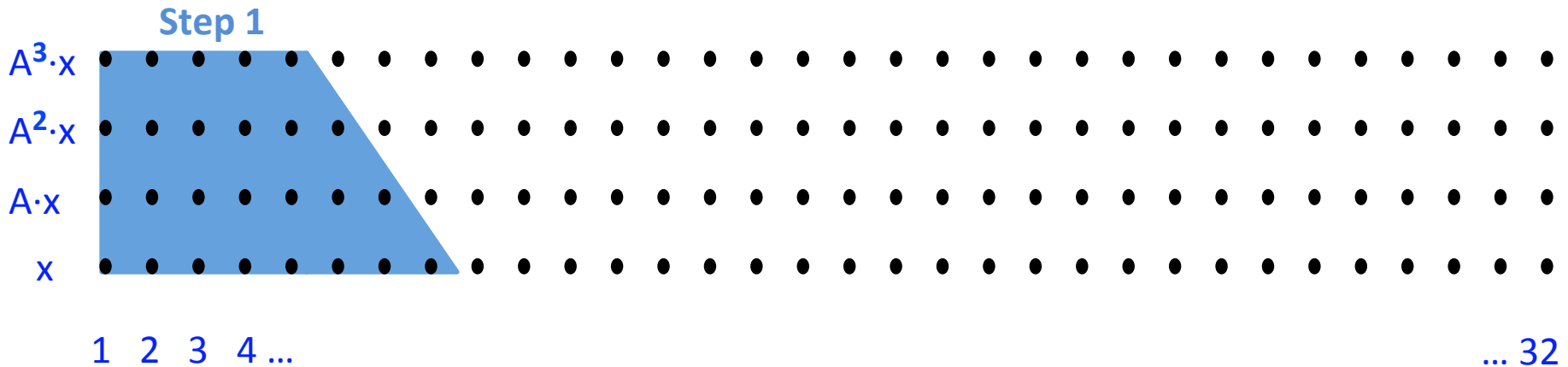


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

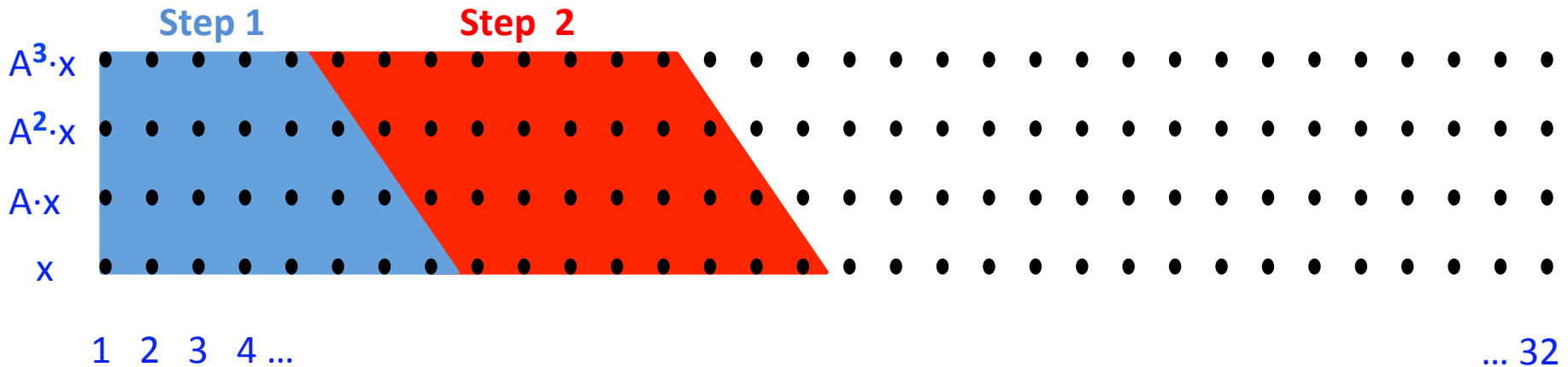


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

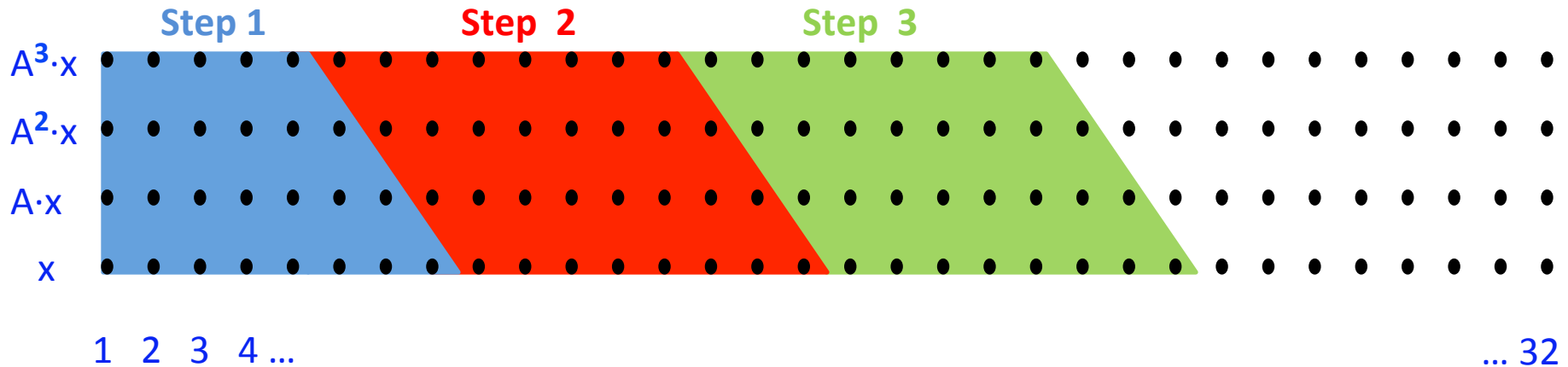


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

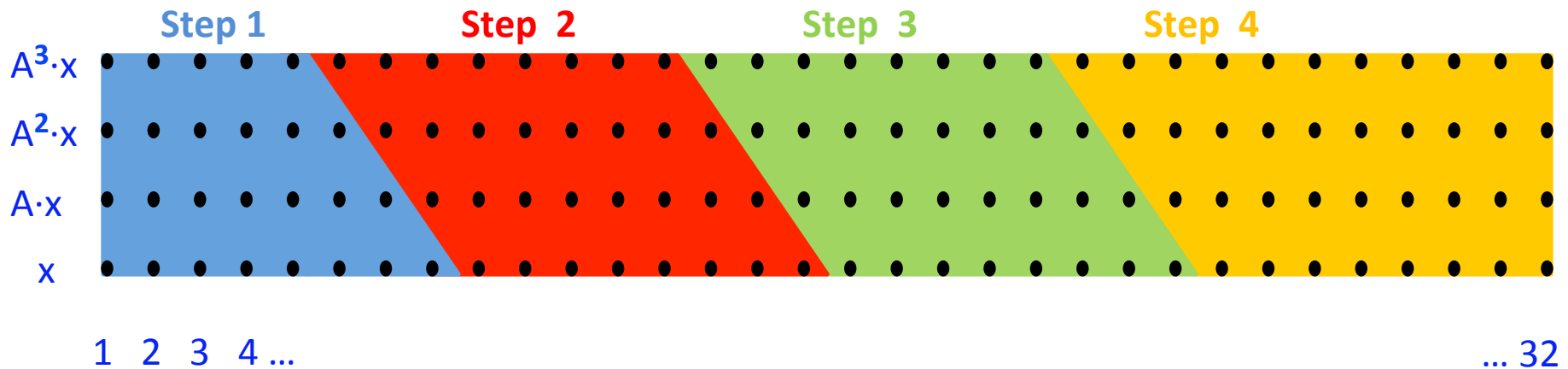


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

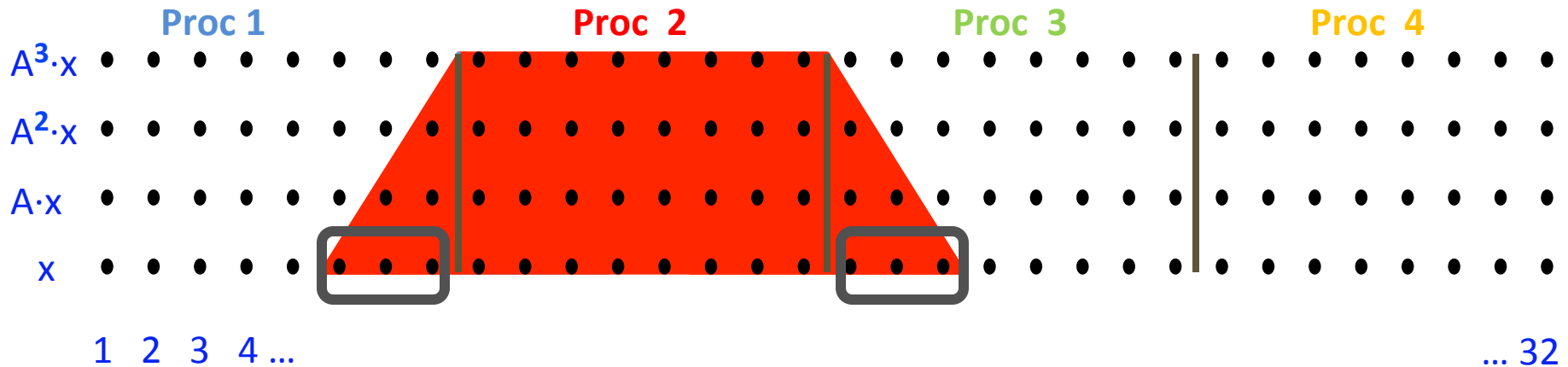


- Example: A tridiagonal, $n=32$, $k=3$

Communication Avoiding Kernels:

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

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

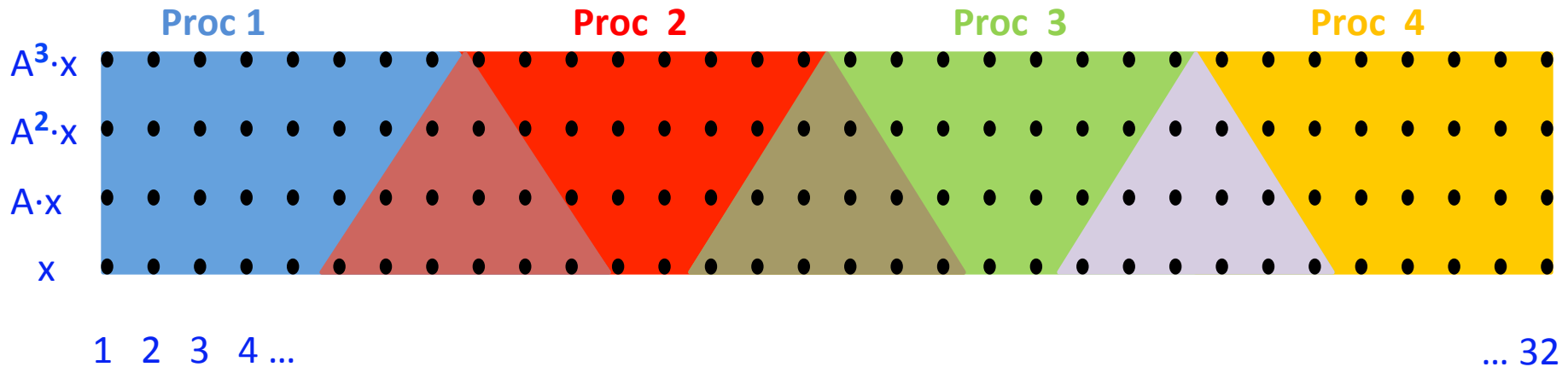


- Example: A tridiagonal, $n=32$, $k=3$
- Each processor communicates once with neighbors

Communication Avoiding Kernels:

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

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



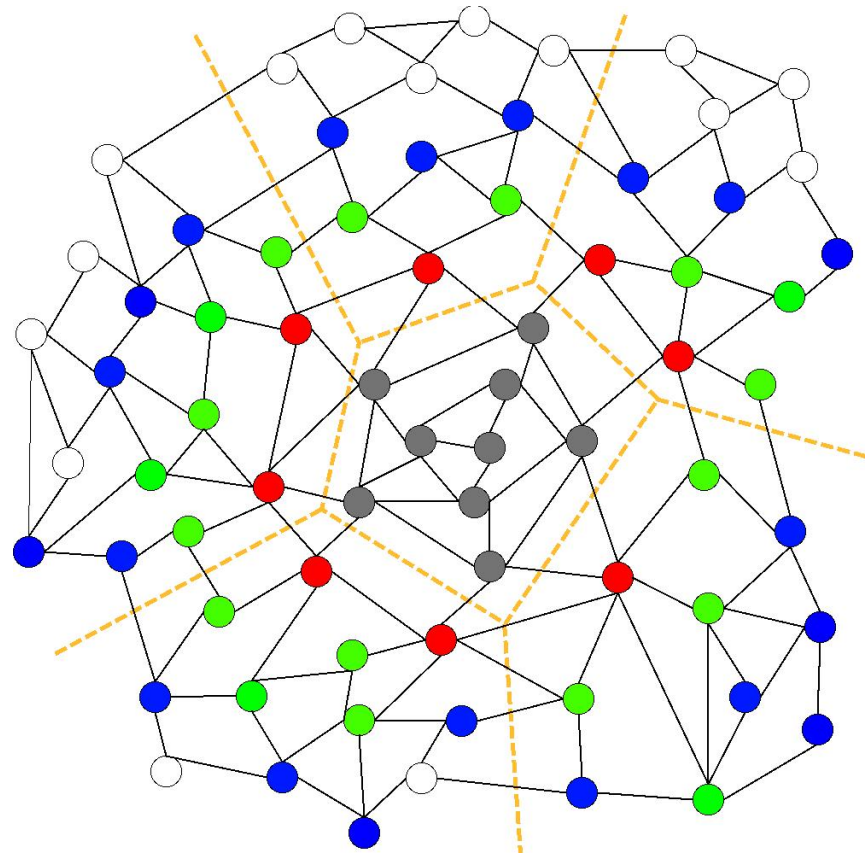
- Example: A tridiagonal, $n=32$, $k=3$
- Each processor works on (overlapping) trapezoid

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

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 $\text{span}\{b, Ab, \dots, A^k b\}$ minimizing $\|Ax - b\|_2$

Standard GMRES

for $i=1$ to k

$w = A \cdot v(i-1)$... *SpMV*

MGS($w, v(0), \dots, v(i-1)$)

update $v(i), H$

endfor

solve LSQ problem with H

Communication-avoiding GMRES

$W = [v, Av, A^2v, \dots, A^k v]$

$[Q, R] = \text{TSQR}(W)$

... *“Tall Skinny QR”*

build H from R

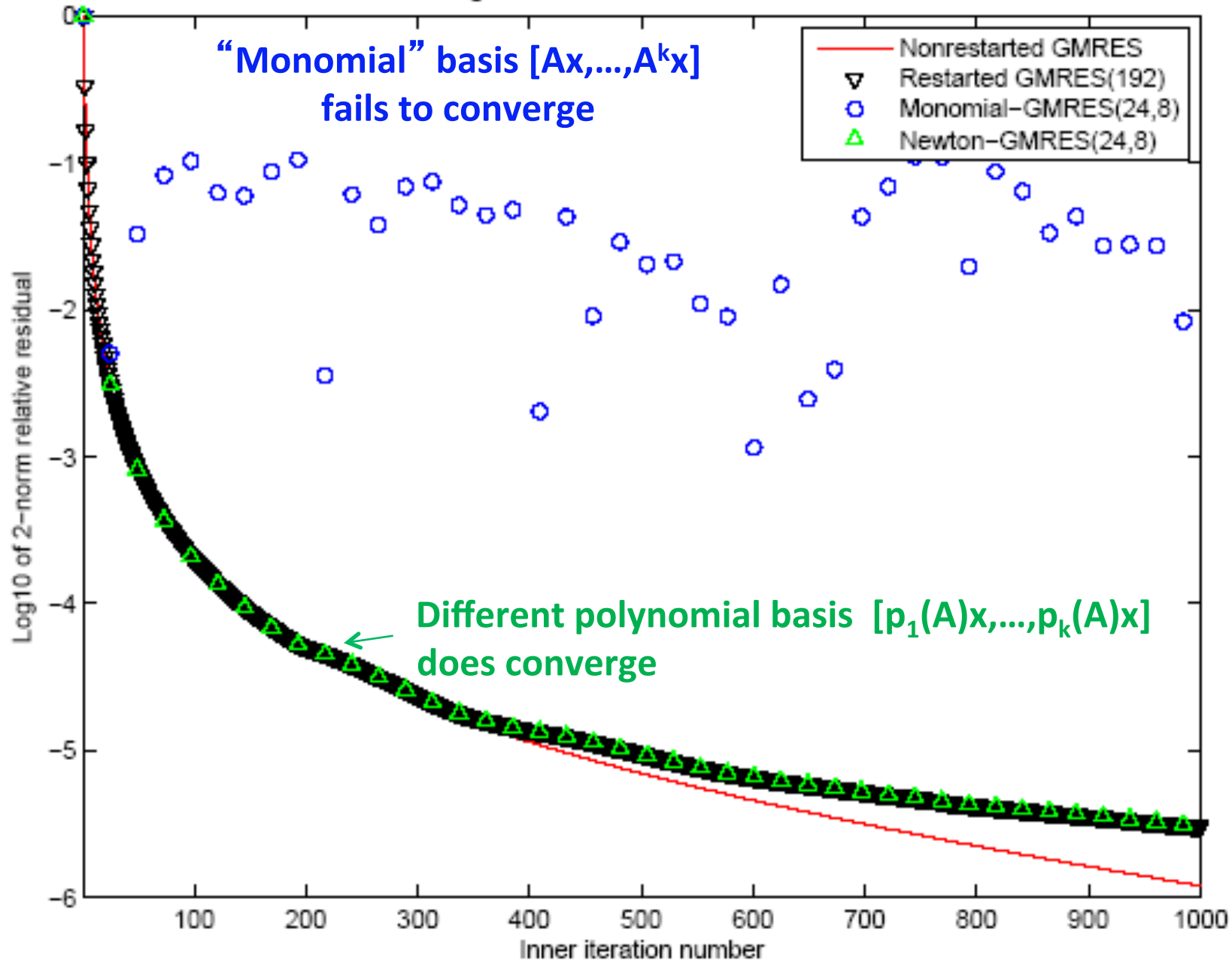
solve LSQ problem with H

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 diag-cond-1.000000e-11: rel. 2-nrm resid.

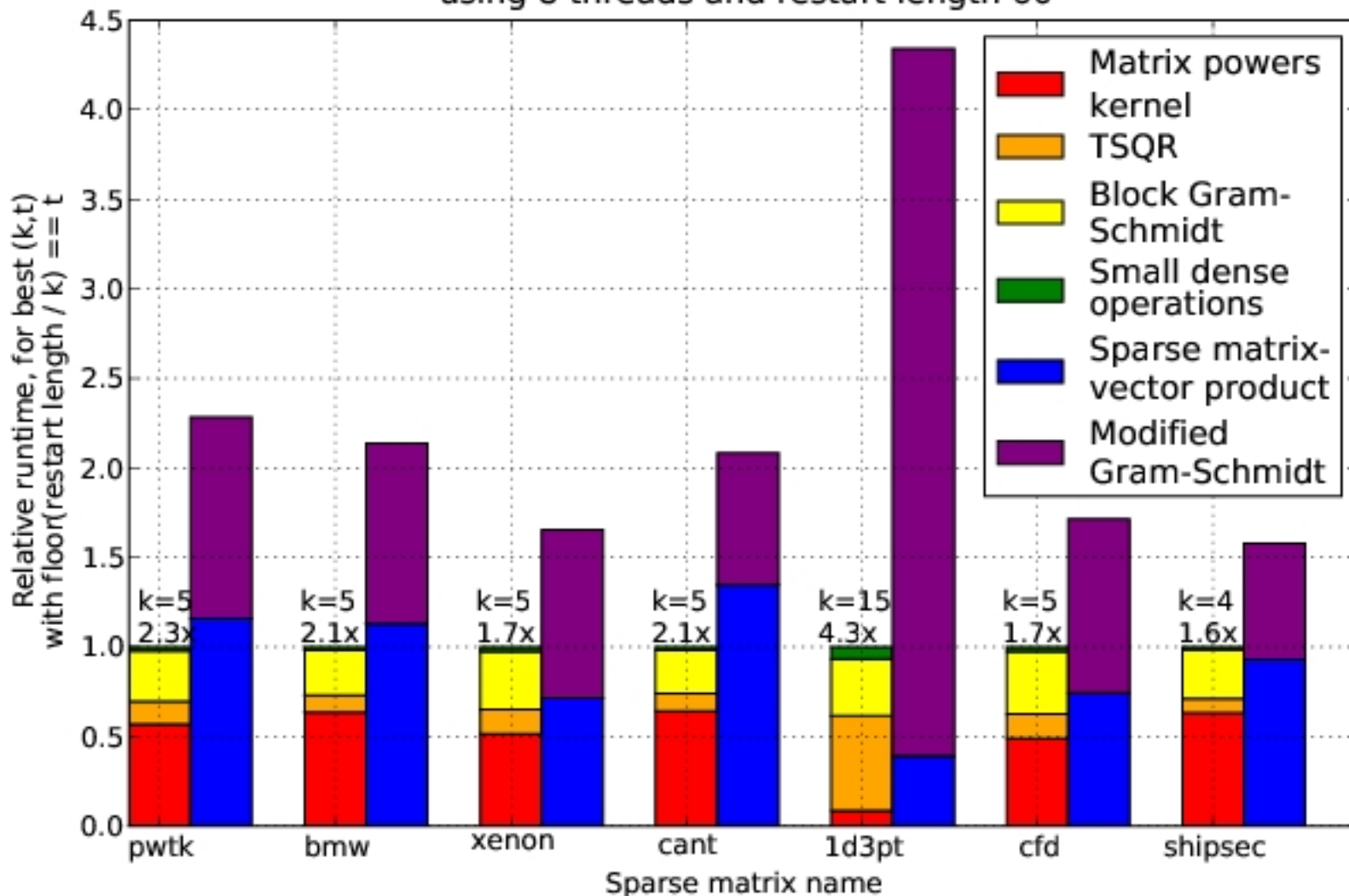


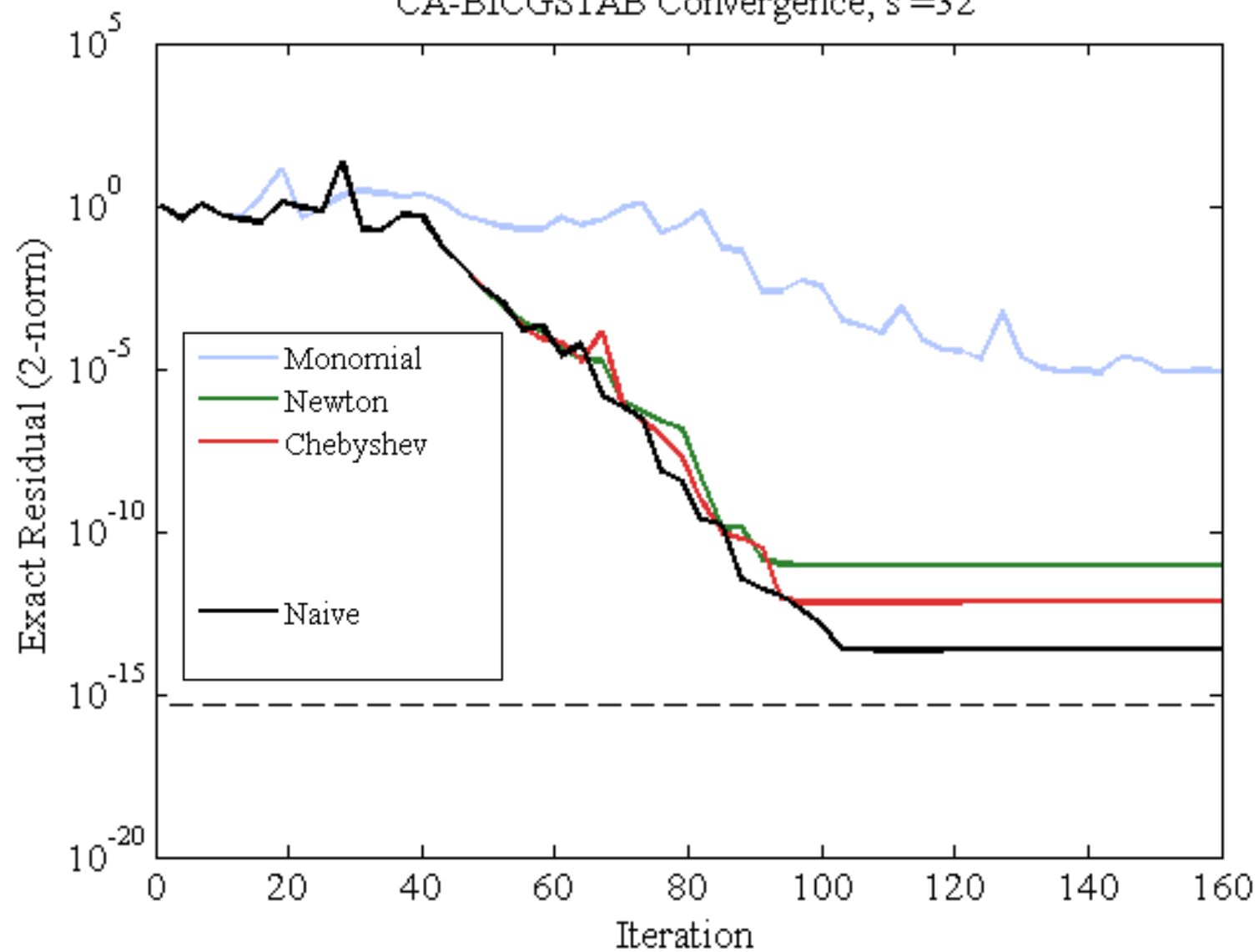
Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

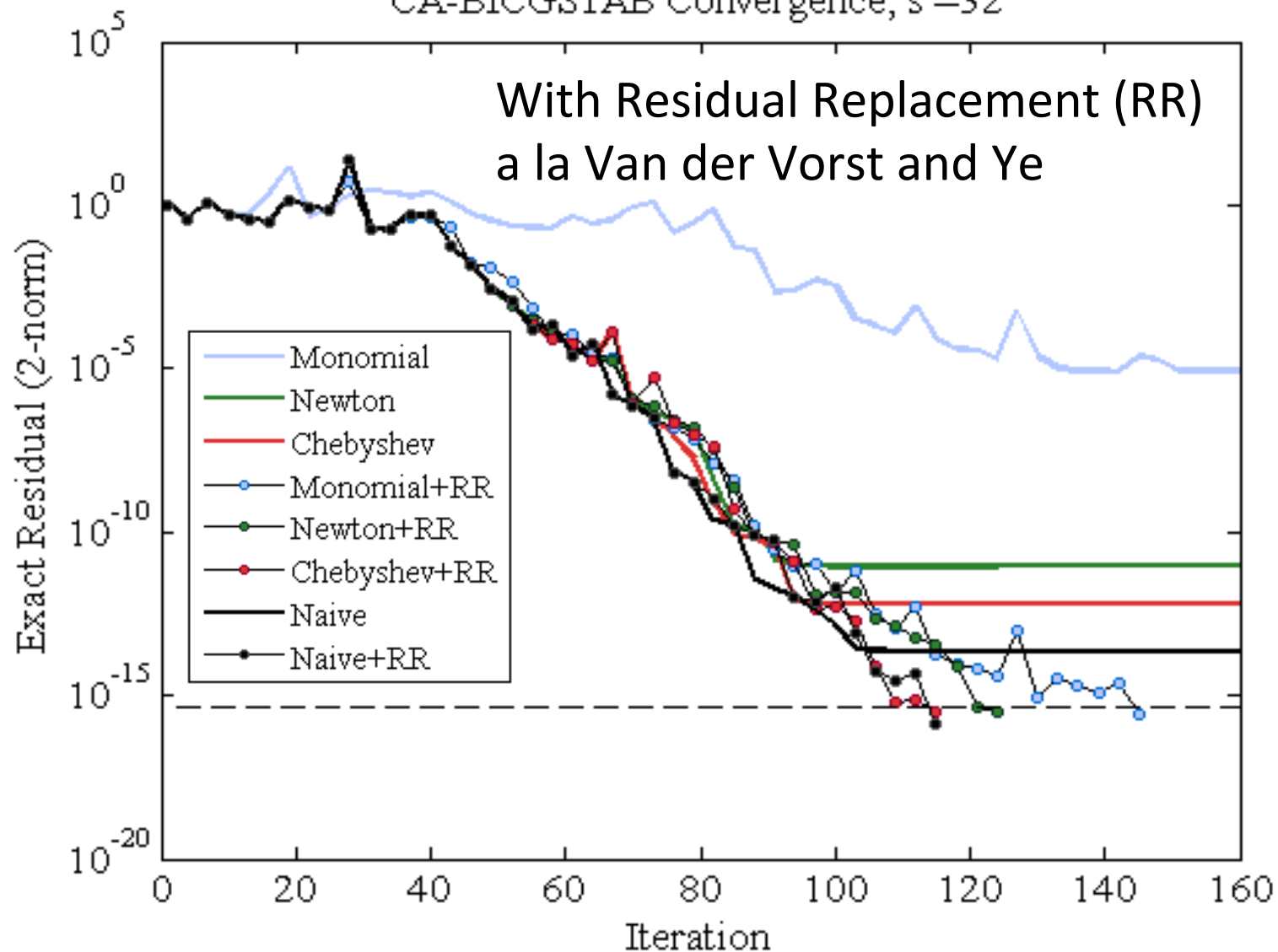
[MHDY09]

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60



CA-BICGSTAB Convergence, $s = 32$ 

CA-BICGSTAB Convergence, $s=32$



	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
 - pOSKI for SpMV – available at bebop.cs.berkeley.edu
 - 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
 - All slides, video available from 2012
 - Prerecorded version planned in Spring 2013
 - www.xsede.org
 - Free supercomputer accounts to do homework!
 - Free autograding of homework!

Summary

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

Don't Communic...