Present and Future Computing Requirements for Linear Algebra Algorithms on High Performance Computers

Jim Demmel

UC Berkeley

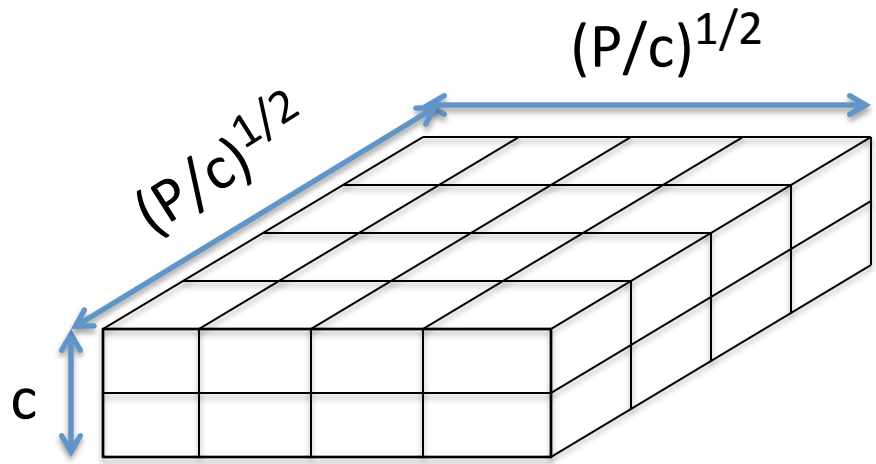
NERSC ASCR Requirements for 2017
January 15, 2014
LBNL

Outline

- Goal: improved widely used linear algebra (and other) libraries to
  - Minimize data movement (most expensive operation, measured in time or energy)
  - Develop reproducible versions
  - Automatically determine minimum precision needed to attain needed accuracy
- What HW/SW features do we need to do this?

2.5D Matrix Multiplication

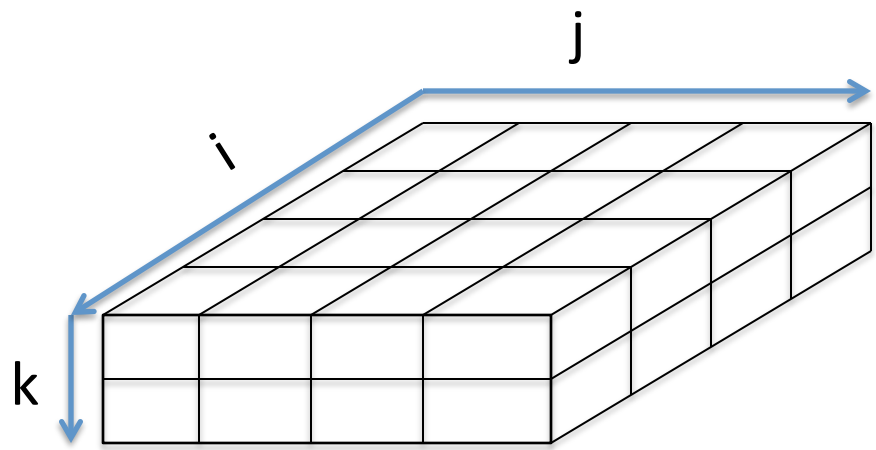
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



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

2.5D Matrix Multiplication

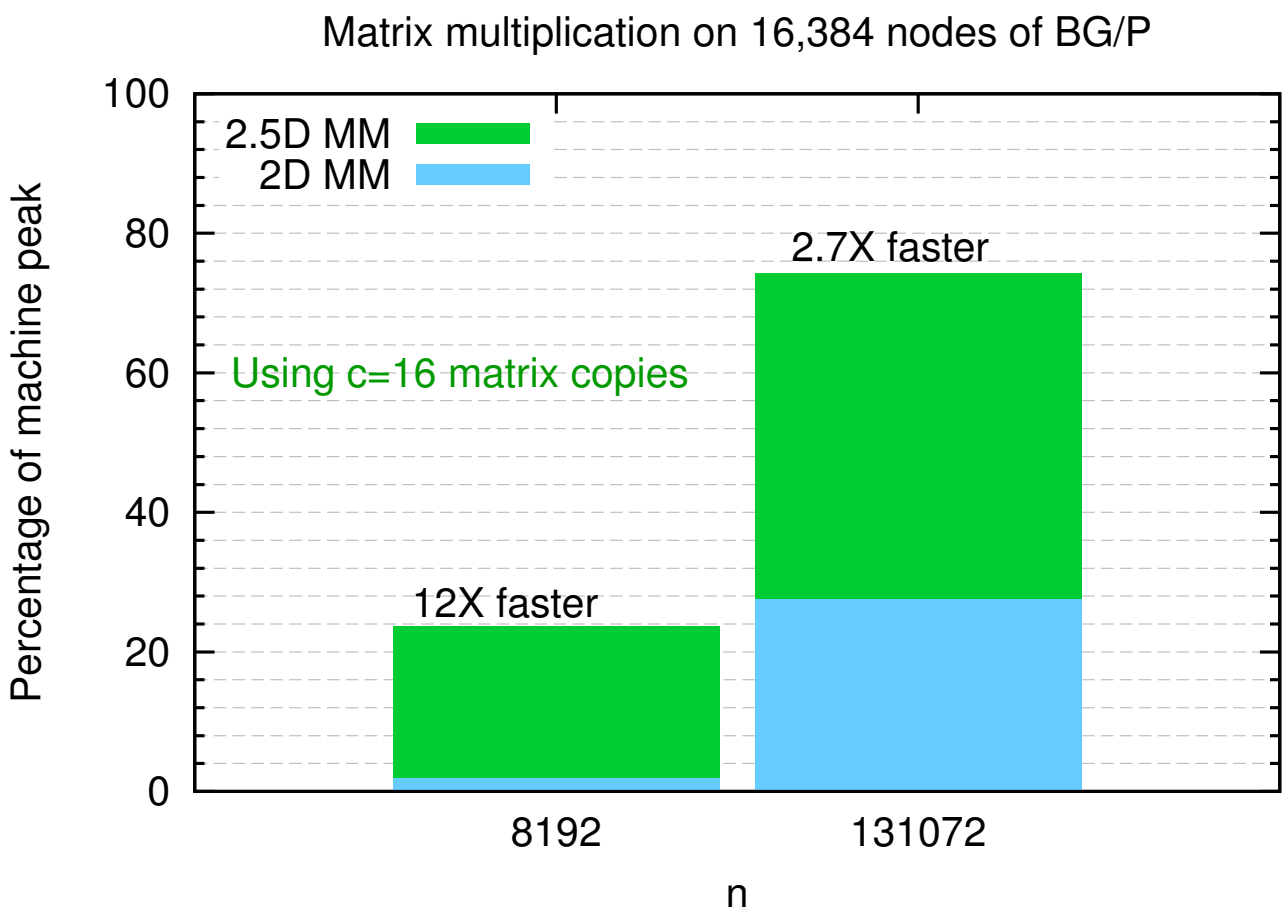
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid



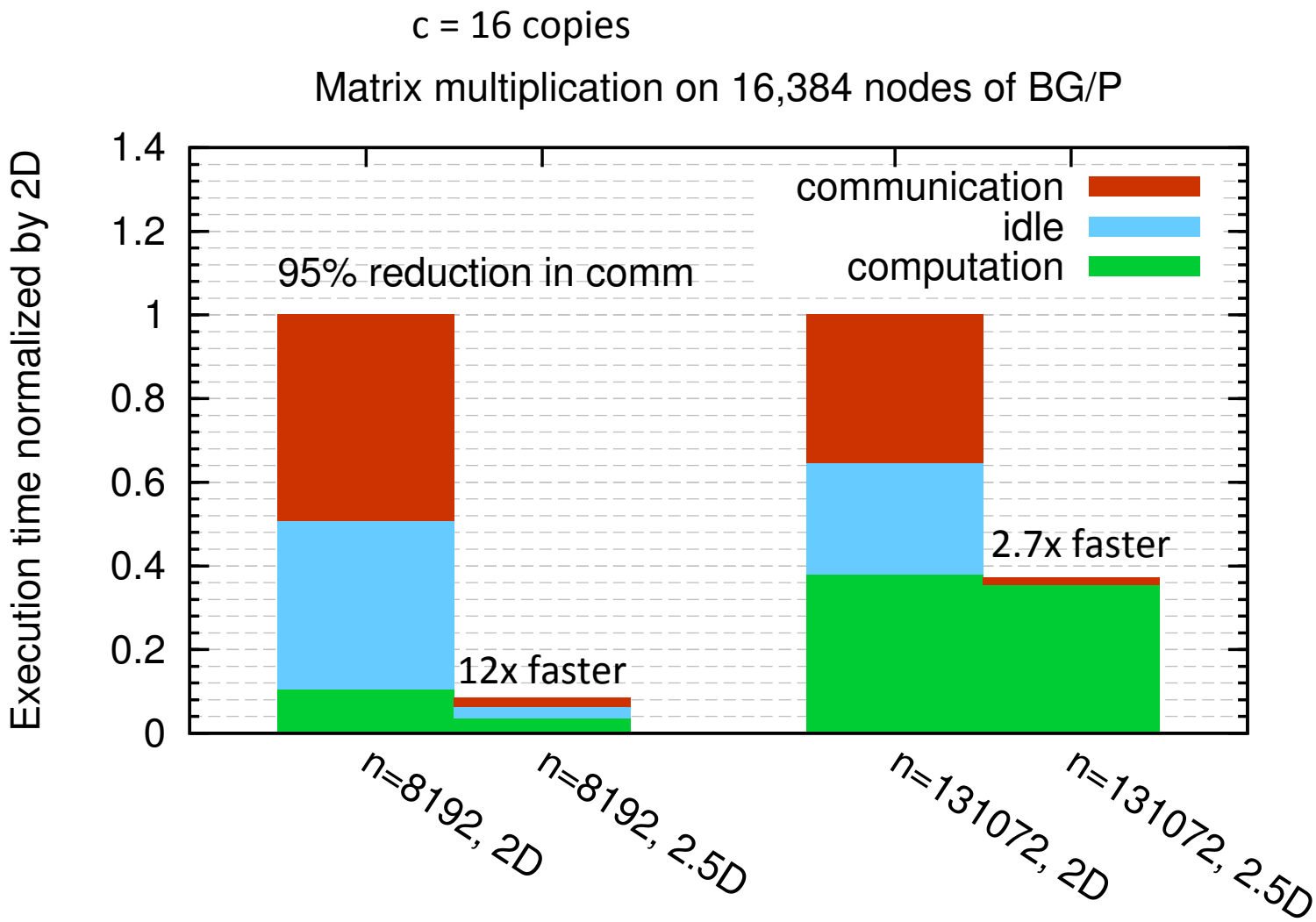
Initially P(i,j,0) owns A(i,j) and B(i,j) each of size $n(c/P)^{1/2} \times n(c/P)^{1/2}$

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

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



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



Distinguished Paper Award, EuroPar'11 (Solomonik, D.)

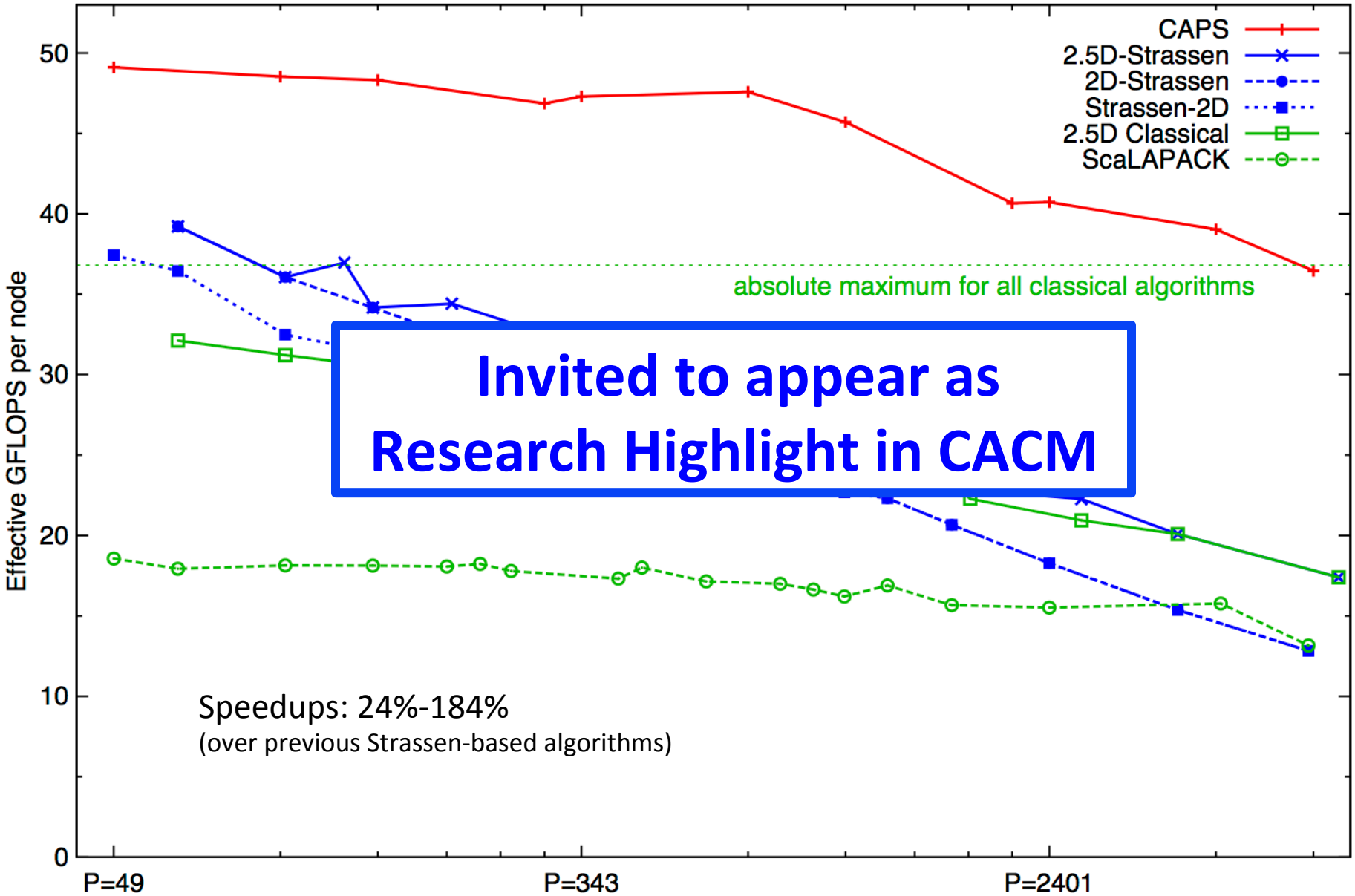
SC'11 paper by Solomonik, Bhatele, D.

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.
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E MT(cP) + \varepsilon_E T(cP)\ \}$
  $= E(P)$
- Perfect scaling extends to N-body, Strassen, …

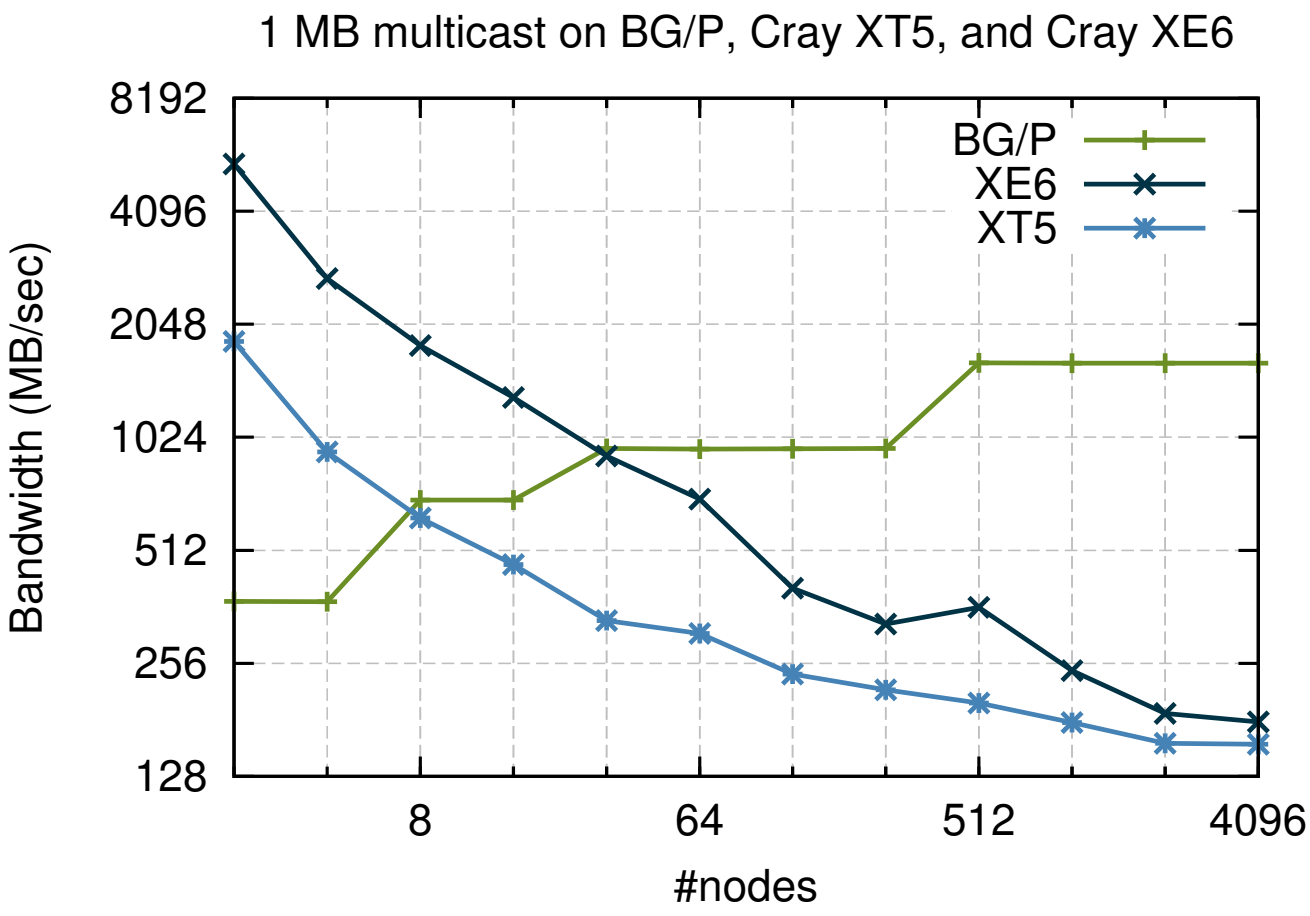
Performance Benchmarking, Strong Scaling Plot

Franklin (Cray XT4) n = 94080



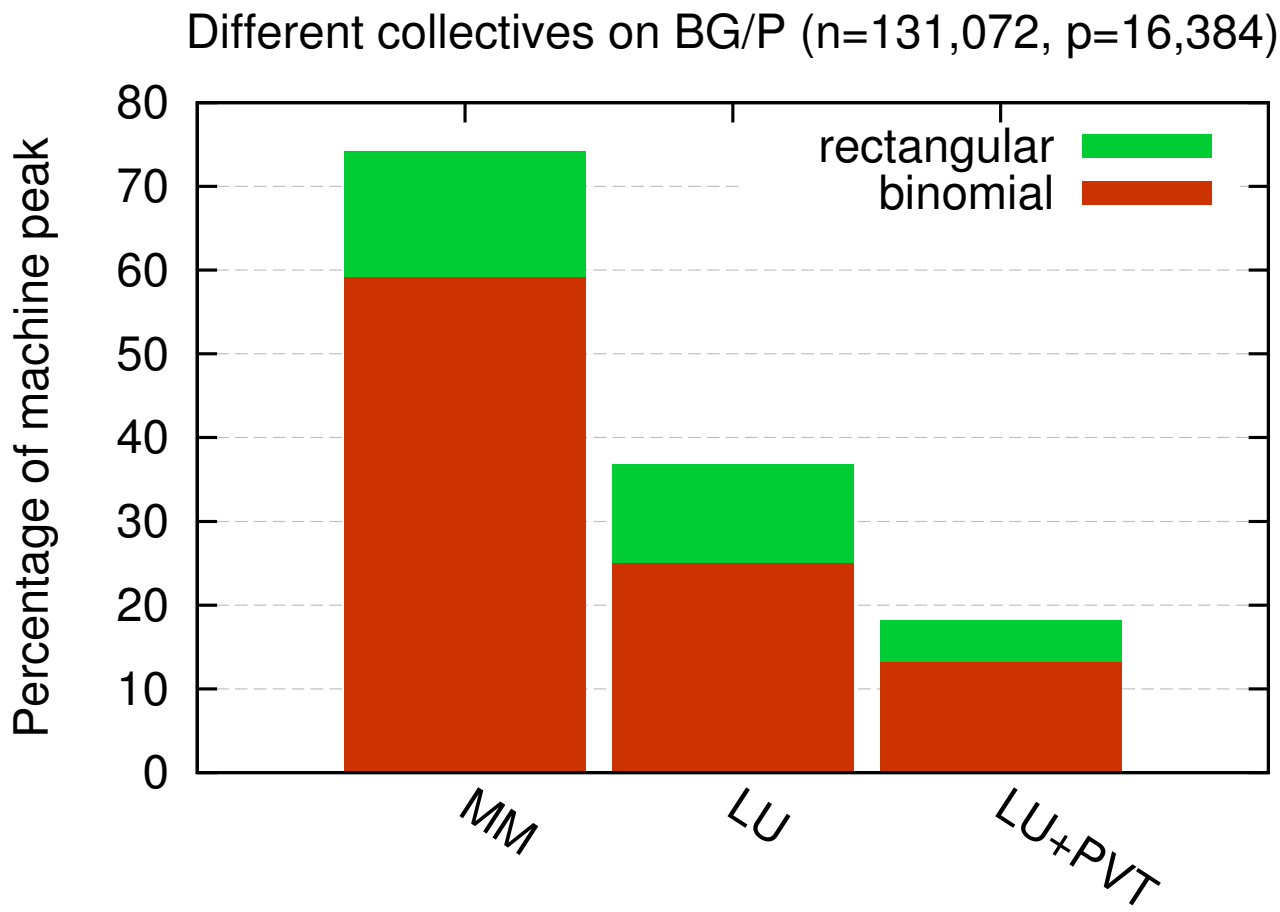
Invited to appear as Research Highlight in CACM

Cost of topology-unaware communication (1/3)



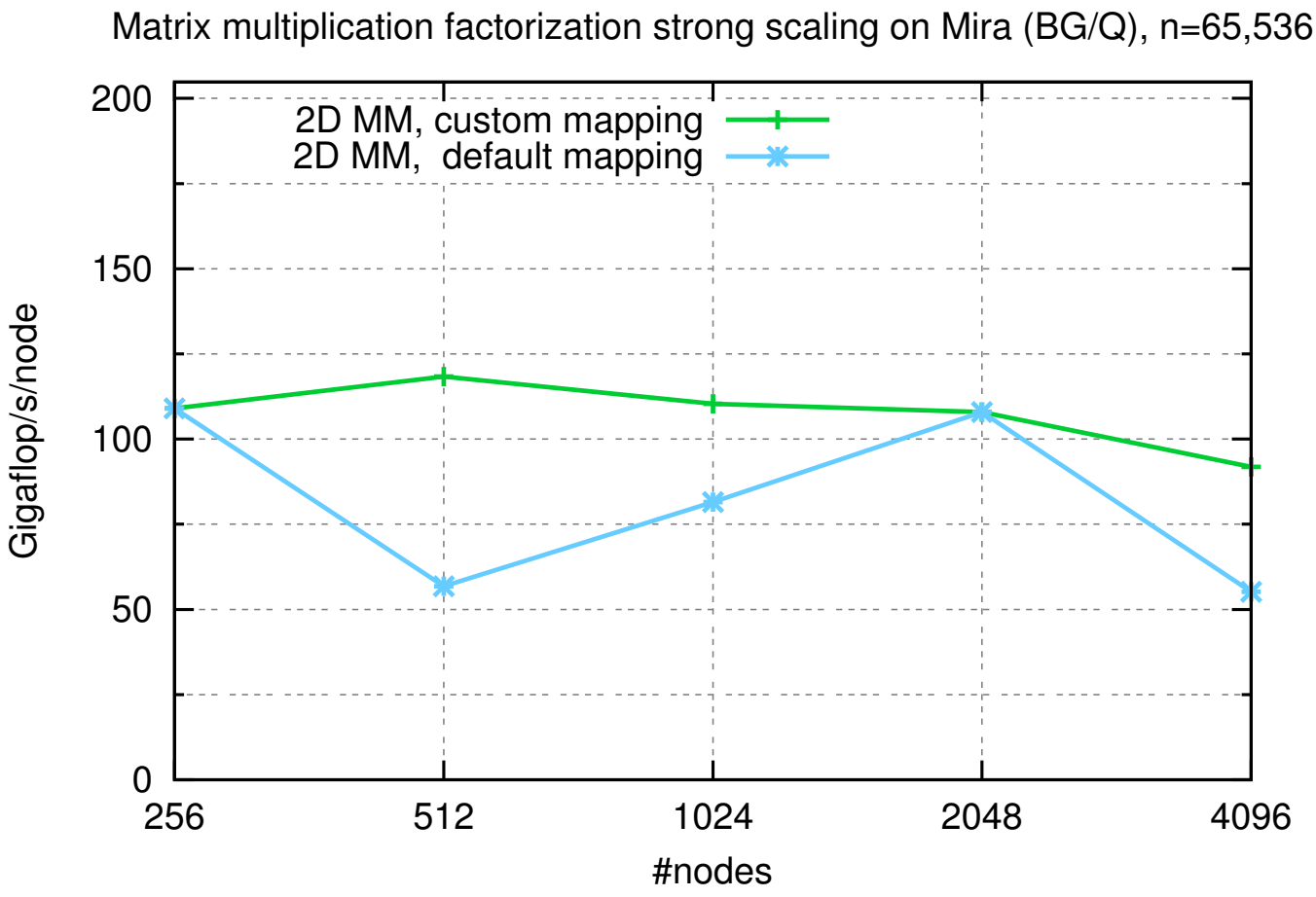
Source: Edgar Solomonik

Cost of topology-unaware communication (2/3)



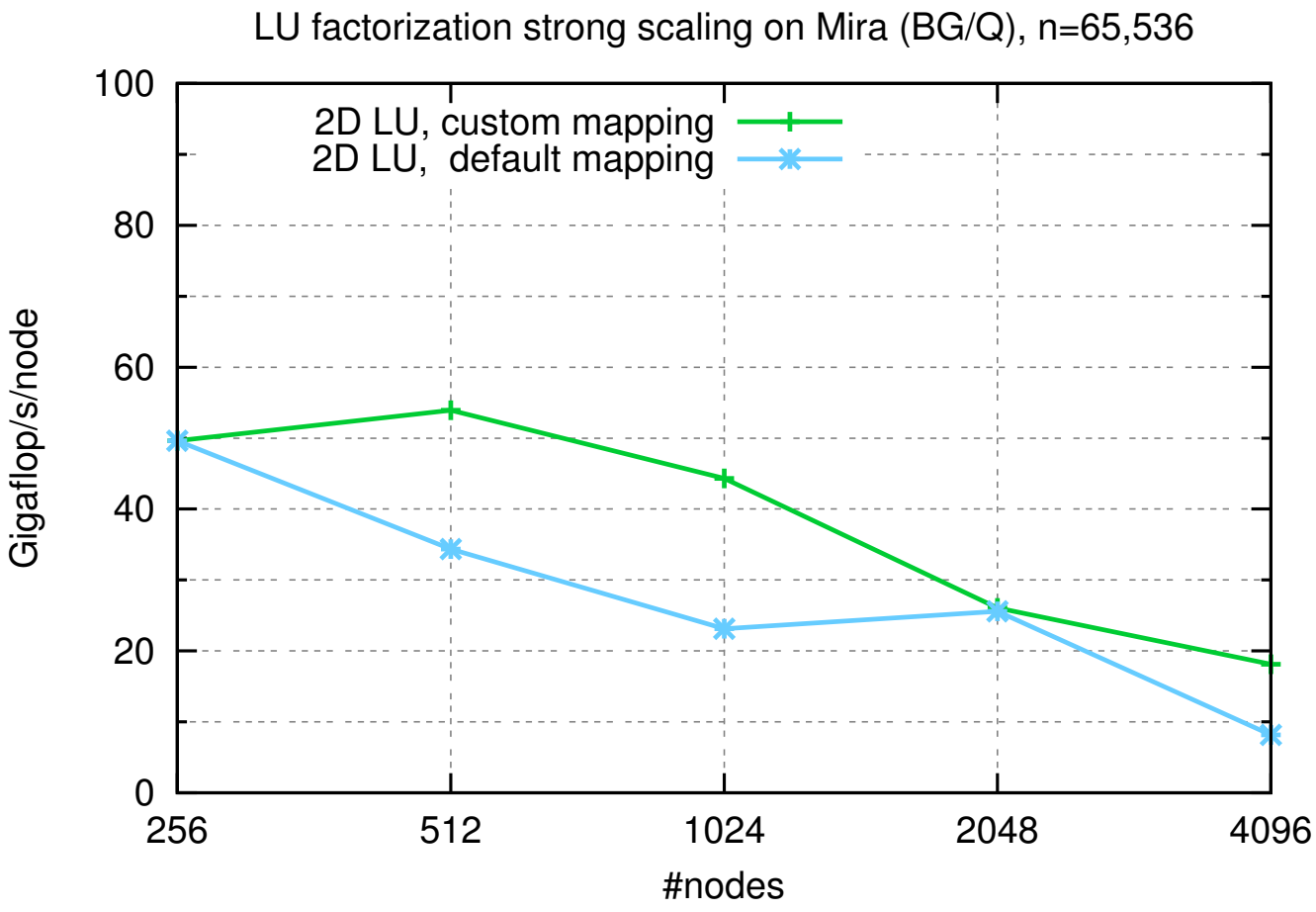
Source: Edgar Solomonik

Cost of topology-unaware communication (3/3)



Source: Edgar Solomonik

Cost of topology-unaware communication



Source: Edgar Solomonik

What features do we need for high performance?

- Ability to allocate "regular" subsets of a machine, with simple interconnection topologies, and use corresponding topology-aware communication algorithms
  - What about dragonfly? Experiments on-going
- Access to accurate performance counters for autotuning
- Ability to measure energy/power consumed in various system components, again for tuning

Reproducibility

- Get the same answer when you run the same program more than once on the same hardware, without slowing down (much)
  - Important for debugging, correctness, uncertainty quantification
- Hard because floating point arithmetic not associative (summation order matters)
- Run on same hardware but perhaps different
  - Order of execution (eg reduction trees)
  - Subset of resources (#procs)

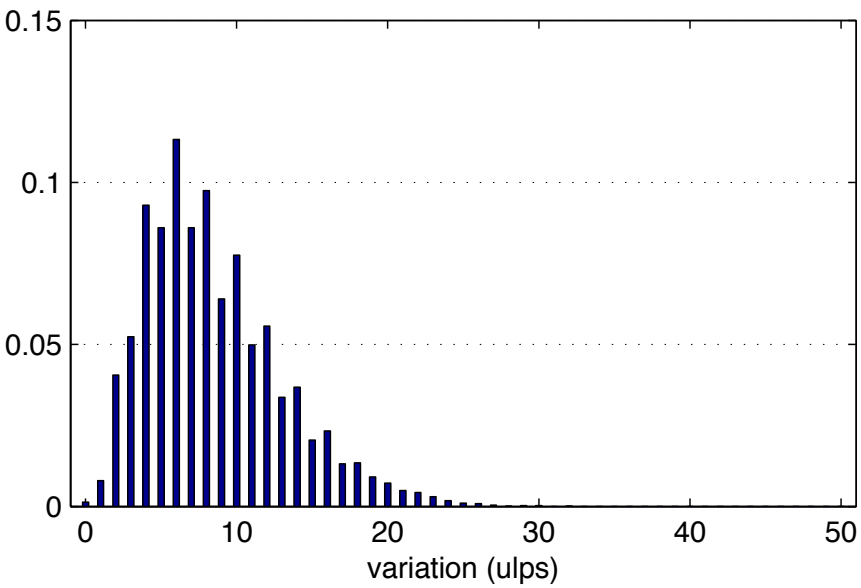
Reproducible Floating Point Computation

- 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

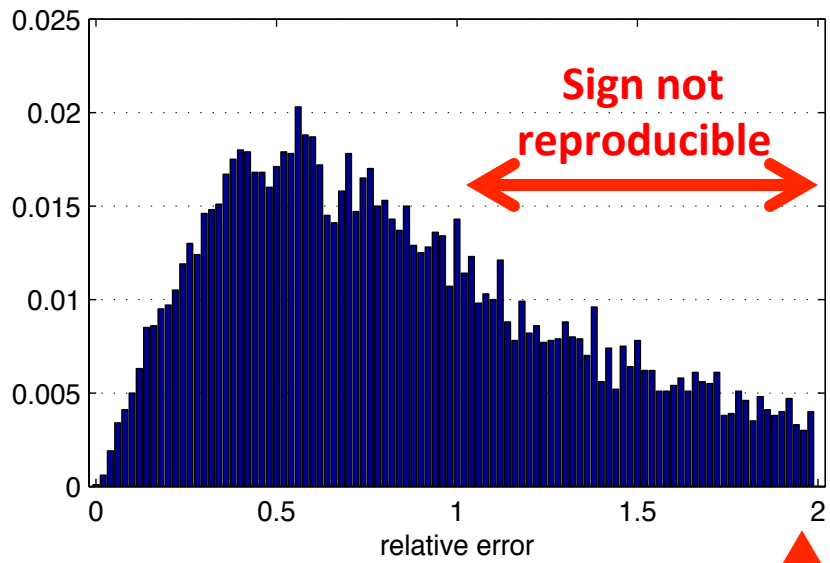
Vector size: 1e6. Data aligned to 16-byte boundaries. For each input vector:

- 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

Source: Diep Nguyen



Relative Error for Orthogonal vectors

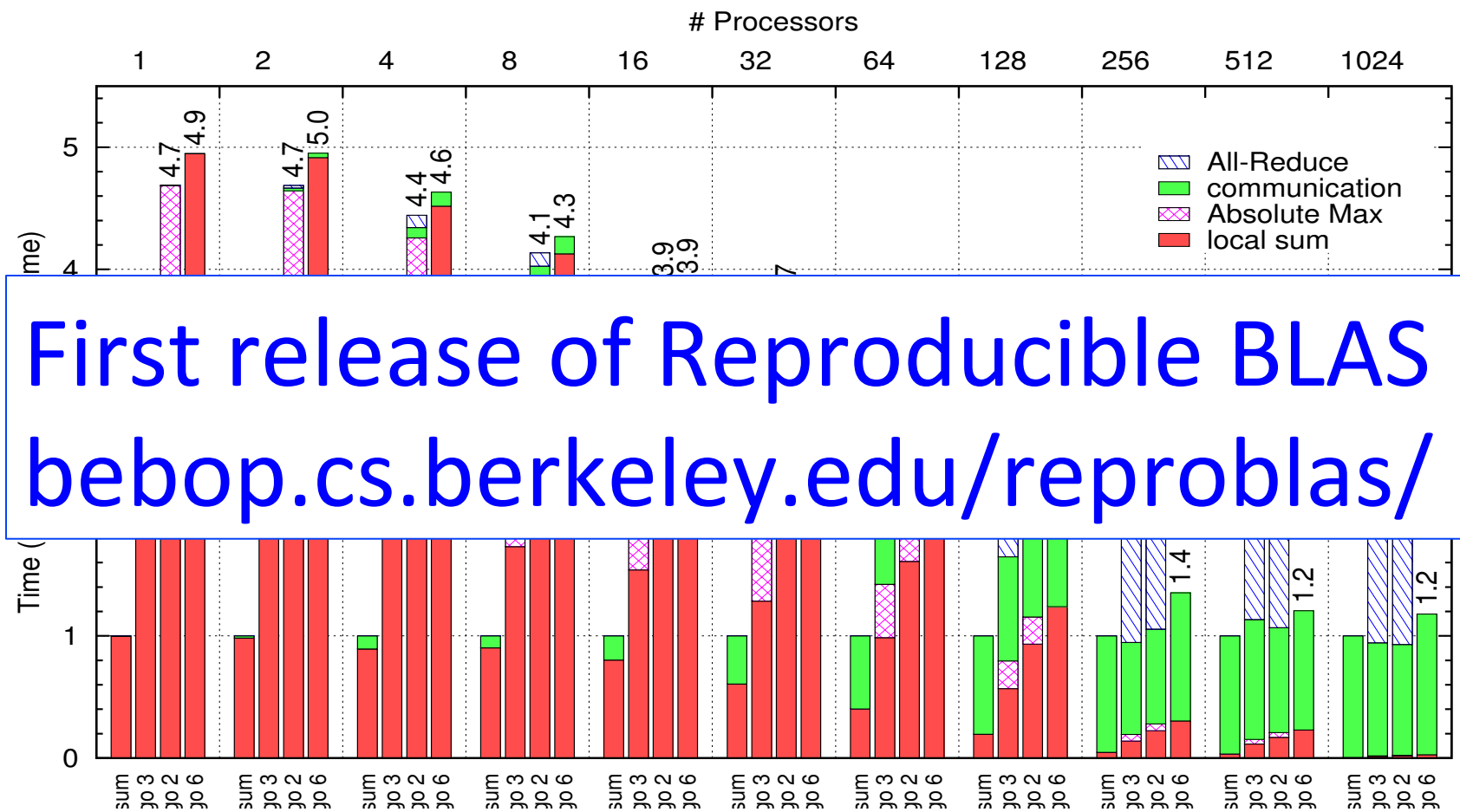
Same magnitude opposite signs

Goals/Approaches for Reproducibility

- Consider summation or dot product
- 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.)
  - Prerounding technique (Nguyen, D.)

Performance results on 1024 proc Cray XC30

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



First release of Reproducible BLAS

bebop.cs.berkeley.edu/reproblas/

Source: Diep Nguyen

What features do we need for Efficient Reproducibility?

- Ideally, provided by vendors (eg MKL, sort of)
- Otherwise:
  - Extended precision
  - Atomic operations
  - Specialized operations embedded in network

Automatic Precision Tuning

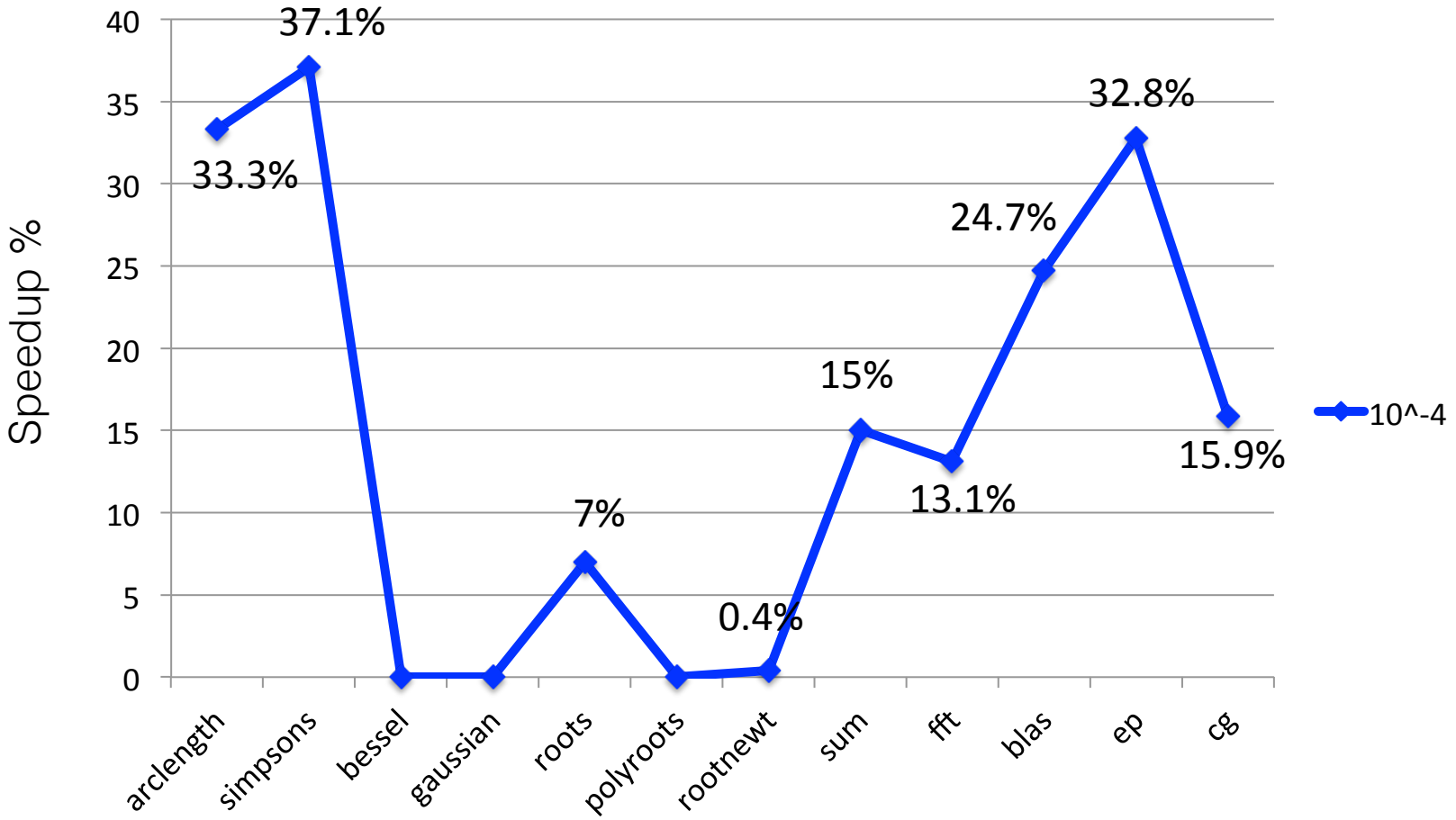
- Automatically determine least precision needed in all variables/operations in order to attain a user-supplied accuracy metric
  - Save time, memory, energy
- Precimonious = "parsimonious with precision"
  - Tool that automatically searches for largest subset of variables whose precision can be lowered

Experimental Results

	Original Type Configuration					Proposed Type Configuration Error threshold: 10^{-4}					
	Program	L	D	F	C	L	D	F	S	# Config	mm:ss
GSL	bessel	0	18	0	0	0	18	0	0	130	37:11
	gaussian	0	52	0	0	0	52	0	0	201	16:12
	roots	0	19	0	0	0	0	19	0	3	1:03
	polyroots	0	28	0	0	0	28	0	0	336	43:17
	rootnewt	0	12	0	0	0	4	8	0	61	16:56
	sum	0	31	0	0	0	9	22	0	325	28:14
	fft	0	22	0	0	0	0	22	0	3	1:16
	blas	0	17	0	0	0	0	17	0	3	1:06
NAS	EP	0	13	0	4	0	5	8	4	111	23:53
	CG	0	32	0	3	0	2	30	3	44	0:57
	arclength	9	0	0	3	0	2	7	3	33	0:40
	simpsons	9	0	0	2	0	0	9	2	4	0:07

Source: Cindy Rubio Gonzalez

Speedup for Error Threshold 1e-4

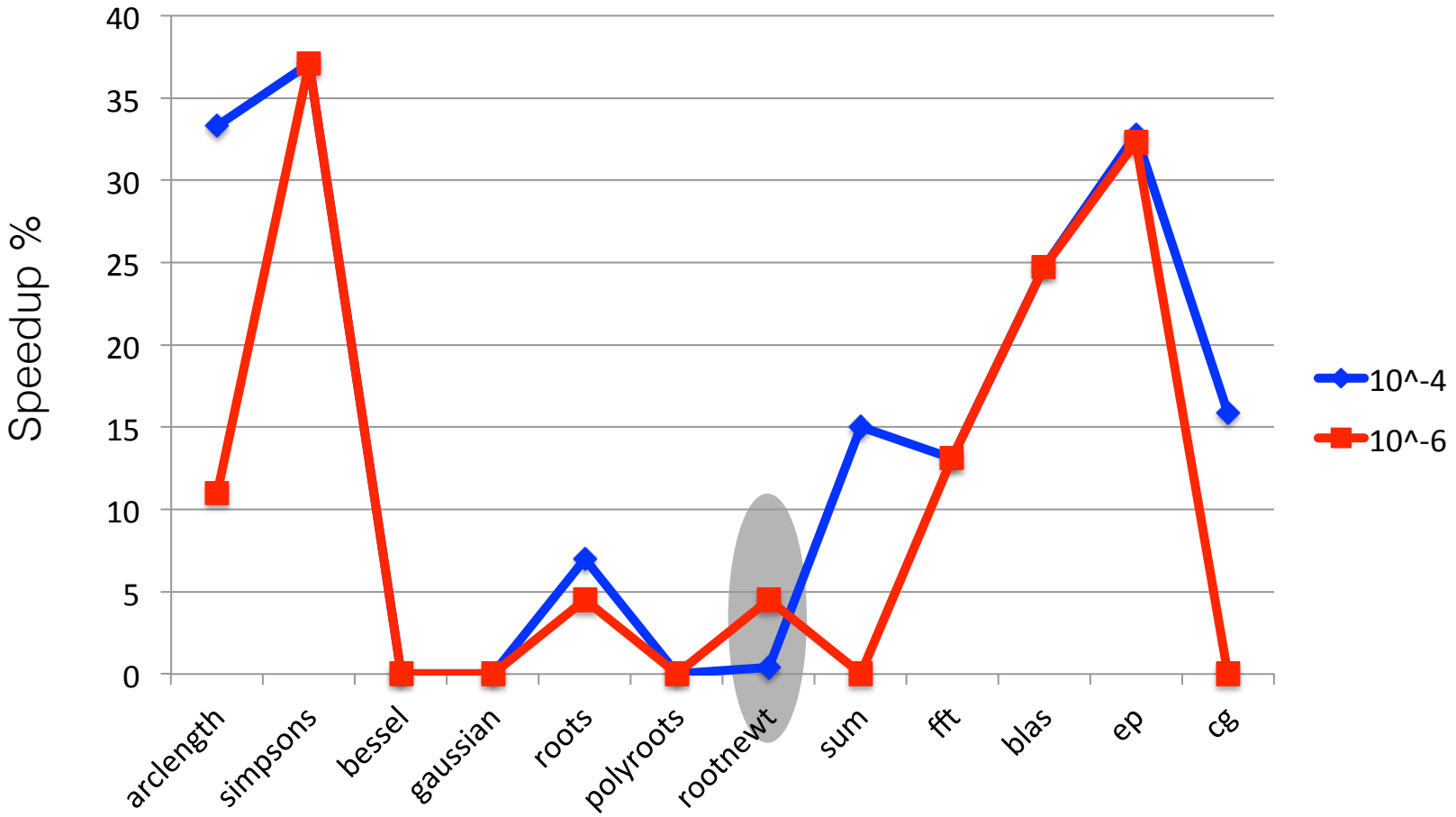


Source: Cindy Rubio Gonzalez

What features do we need for effective precision tuning?

- Single and double precision, at least
- More levels of precision could be used
- Accurate performance and energy counters to allow effective tuning

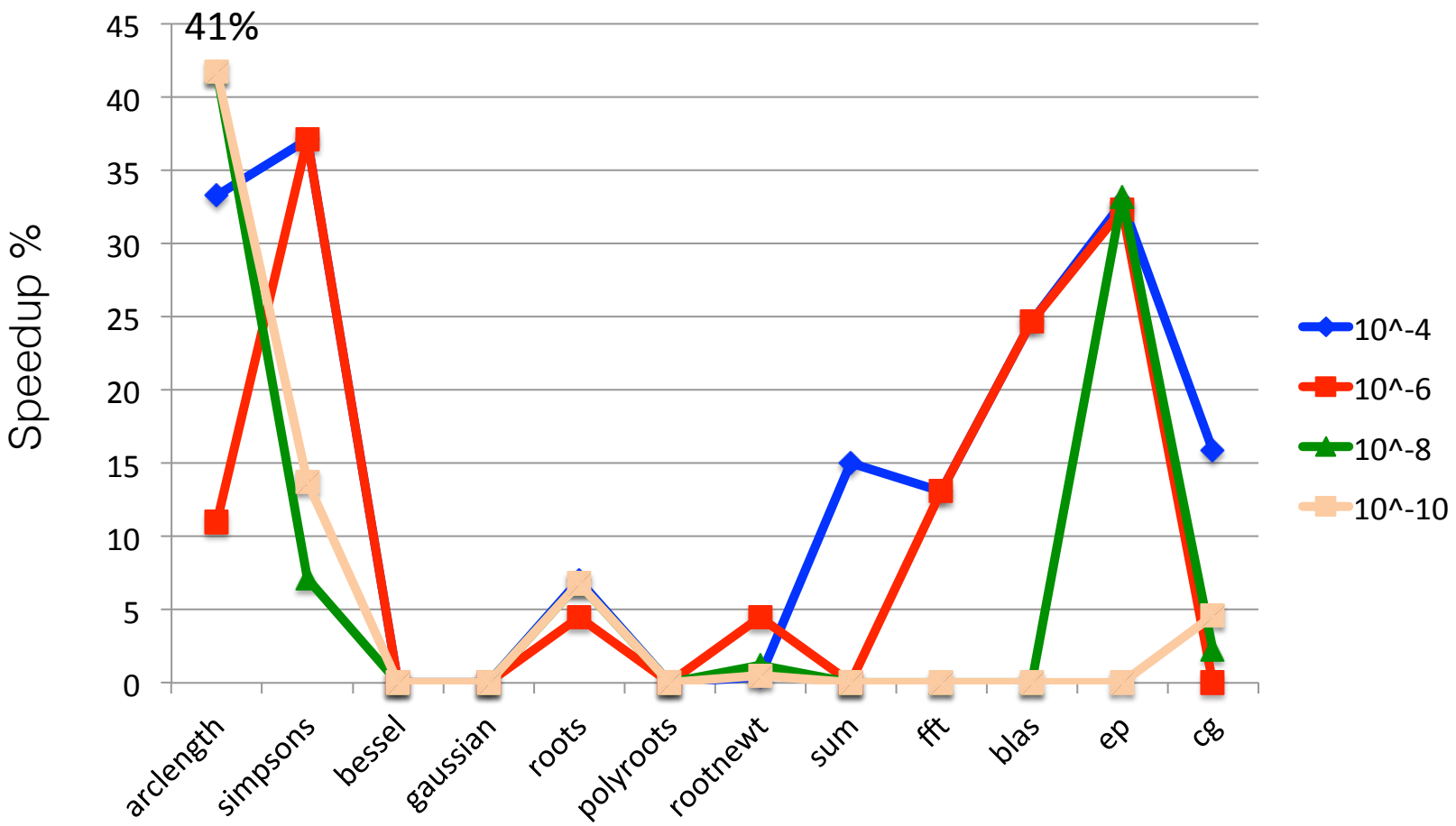
Speedup for Various Error Thresholds



Error threshold 10^{-6} → slightly larger speedup than error threshold 10^{-4}
(4.5% vs. 0.4% for rootnewt program)

Source: Cindy Rubio Gonzalez

Speedup for Various Error Thresholds



Source: Cindy Rubio Gonzalez

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, Diep Nguyen, Oded Schwartz, Edgar Solomonik, Omer Spillinger
- Koushik Sen, William Kahan, David Bailey, David Hough, Cindy Rubio-Gonzalez, Ahn Cuong
- 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 ParLab, ASPIRE, BEBOP, CACHE, EASI, FASTMath, MAGMA, PLASMA
- Thanks to DOE, NSF, UC Discovery, INRIA, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle
- **bebop.cs.berkeley.edu**

Extra Slides

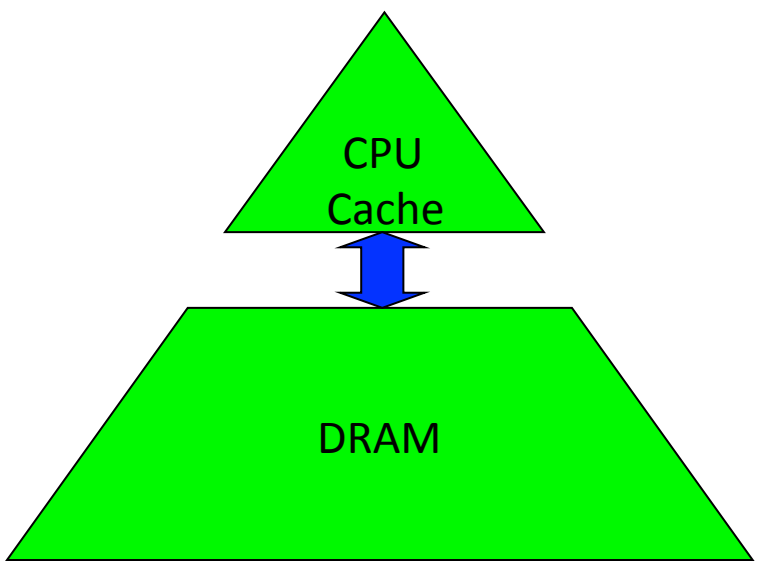
Communication-Avoiding Algorithms for Linear Algebra and Beyond

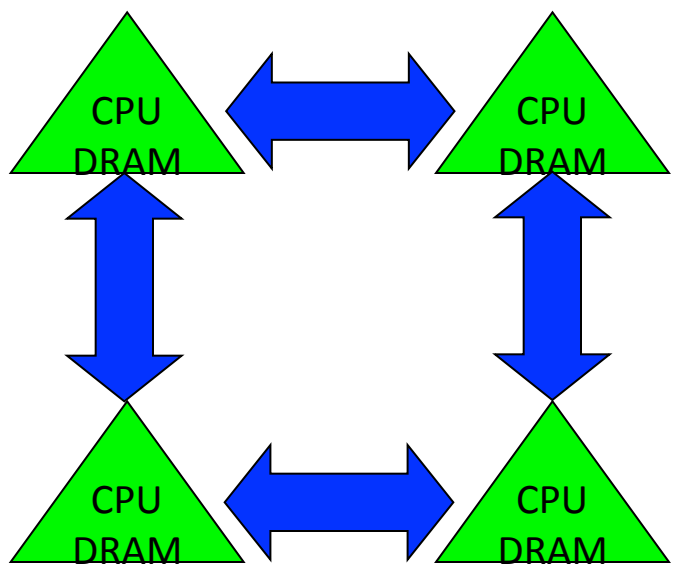
Jim Demmel
EECS & Math Departments
UC Berkeley

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

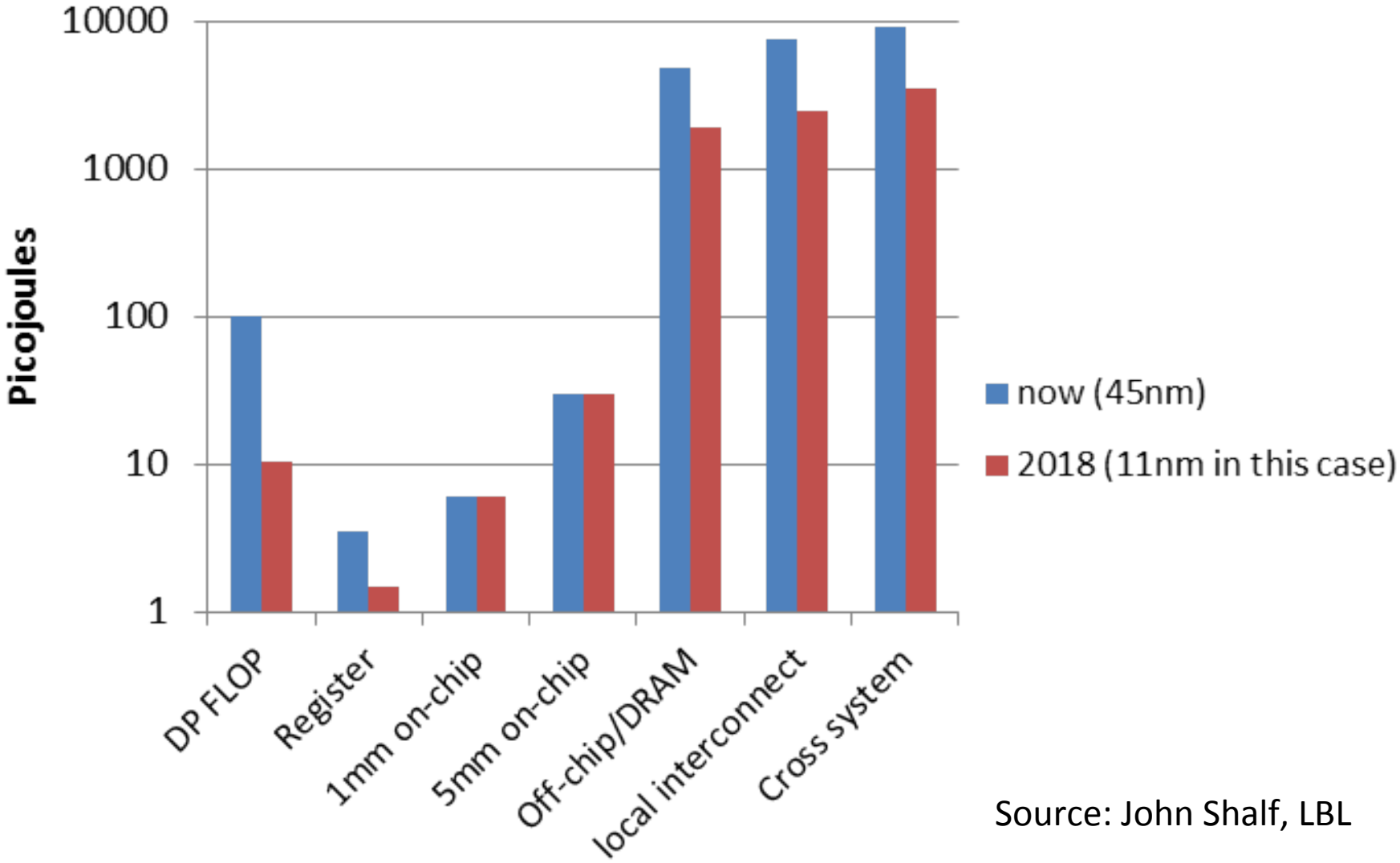
  } communication

- 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

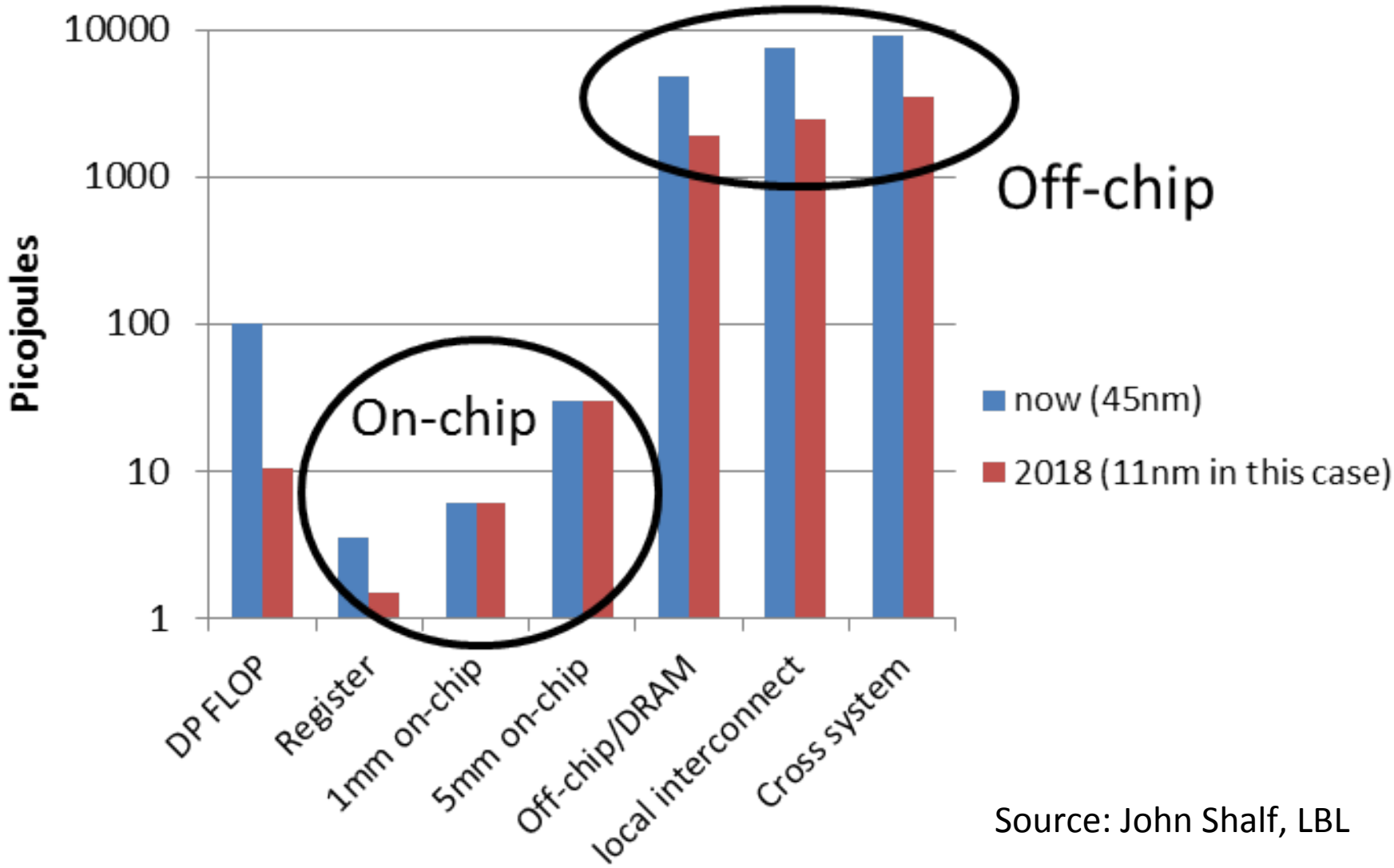
Why Minimize Communication? (3/3)



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 $\leftrightarrow$ L2 $\leftrightarrow$ DRAM $\leftrightarrow$ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible
- New lower bounds, optimal algorithms, for
  - Direct linear algebra (matmul, LU, QR, SVD, etc.)
  - Iterative linear algebra (conjugate grad., etc.)
  - Other algorithms that access arrays (n-body, etc.)

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
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

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!)
  - Being added to Sca/LAPACK, PLASMA, MAGMA, …
- Only a few sparse algorithms so far
- Lots of work in progress
- Ditto for iterative linear algebra

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

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

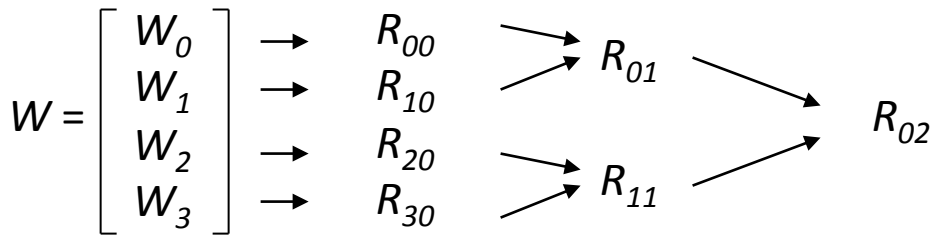
$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\,R_{01} \\ \hline Q_{11}\,R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix}$$

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

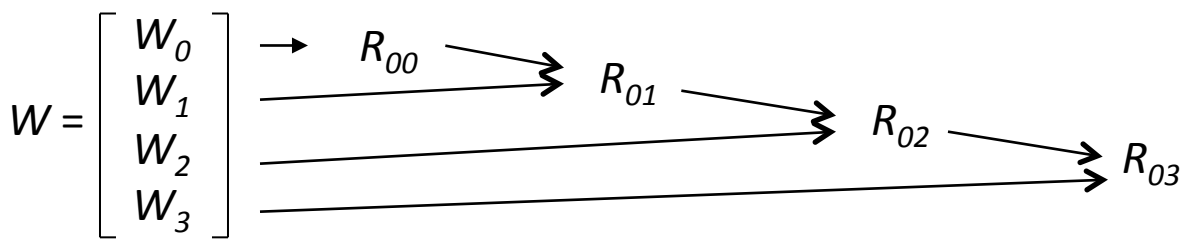
Output = { Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} }

TSQR: An Architecture-Dependent Algorithm

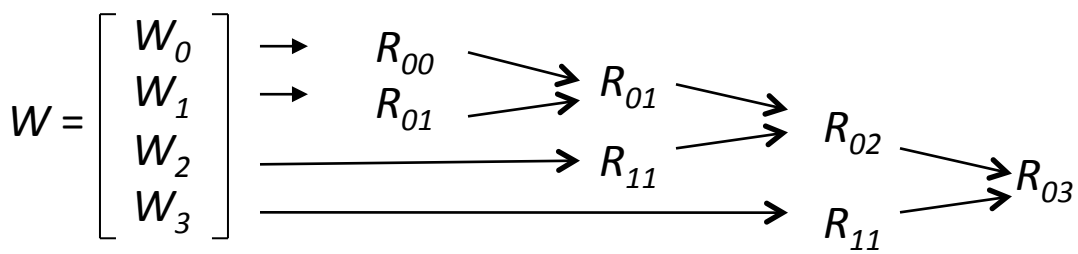
Parallel:



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 vs 1 city (Dongarra, Langou et al)
  - Cloud – (Gleich and Benson) **~2 map-reduces**
- Sequential
  - "**Infinite speedup**" for out-of-core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished
- SVD costs about the same
- Joint work with Grigori, Hoemmen, Langou, Anderson, Ballard, Keutzer, others

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
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

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

Ongoing Work

- Lots more work on
  - Algorithms:
    - BLAS, LU, LDL^T, QR, other pivoting schemes, eigenproblems, …
      - QR with column pivoting
    - Sparse matrices, 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)
  - 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:	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?

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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- 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:
  ... write A as n/b-by-n/b matrix of b-by-b blocks A[i,j]
  ... ditto for B, C
  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-by-b matrix multiply

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

New Thm applied to Matmul

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

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

- Solve LP for $x = [xi,xj,xk]^T$: max $\mathbf{1}^T x$ s.t. $\Delta x \leq \mathbf{1}$
  - 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

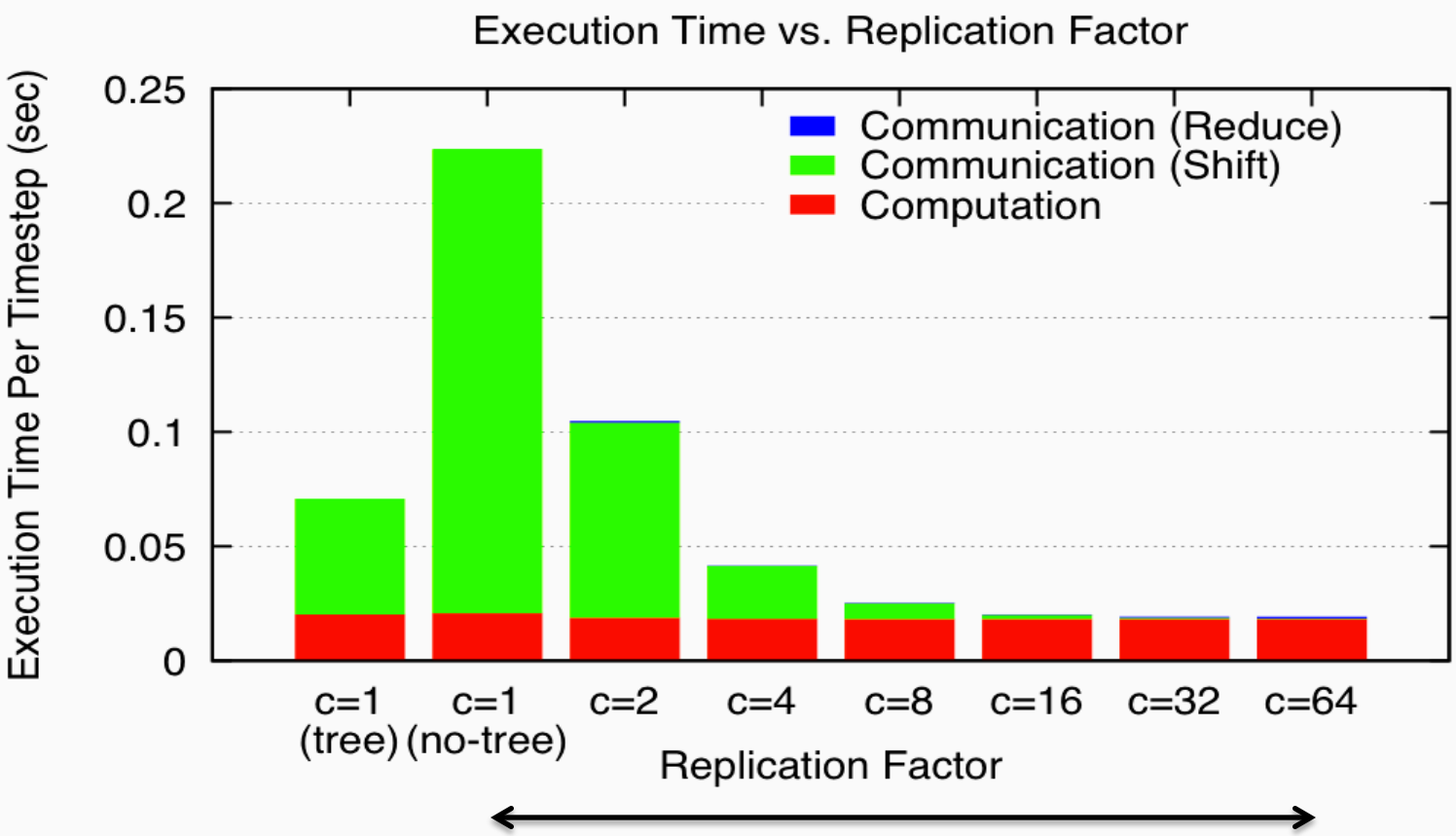
$$\Delta = \begin{array}{cc} \begin{array}{cc} i & j \end{array} & \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{array}{c} F \\ P(i) \\ P(j) \end{array} \end{array}$$

- Solve LP for $x = [xi,xj]^T$: max $\mathbf{1}^T x$ s.t. $\Delta x \leq \mathbf{1}$
  - 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 Join

- For i=1:n, for j=1:n

  if predicate(R(i),S(j)), output(i,j) = func(R(i),S(j))
- Same as n-body, except for predicate
- $Z = \{(i,j), 1 \le i,j \le n\} = Z_T \cup Z_F = \{\text{pred. True}\} \cup \{\text{pred. False}\}$
- Apply Thm separately to Z_T and Z_F , take max of bounds
- Thm: #words_moved = $\Omega(\max(n^2/M^1, |Z_T|)$
- Attained by block sizes $M^{xi}, M^{xj} = M^1, M^1$
- Suggests replicating data could make join faster

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

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

- Solve LP for $x = [x1,\ldots,x7]^T$: 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 $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

Summary of Results Beyond LA (1/3)

- Extend communication lower bound proof from linear algebra to any program with
  - Inner loop iterations indexed by $(i_1,\ldots,i_d)$
  - Arrays in inner loop subscripted by linear functions of indices
  - Ex: $A(i_1,i_2-i_1,3i_1-4i_2+7i_4,\ldots)$, $B(ptr(i_5+6i_6))$, …
  - Can be dense or sparse, sequential or parallel, …
- Based on recent result of Bennett/Carbery/Christ/Tao
  - Generalizes **H**older, **B**rascamp-**L**ieb, Loomis-Whitney
  - Get linear program with one inequality per subgroup $H < Z^d$
  - Solution of linear program is s_{HBL}
  - Thm: #words_moved = Ω(#loop_iterations/M^{sHBL-1})

Summary of Results Beyond LA (2/3)

- Can we write down the lower bound?
  - One inequality per subgroup $H < Z^d$, but still finitely many
  - Thm (bad news): Writing down all the inequalities equivalent to solving Hilbert's 10th Problem over Q
  - Thm (good news): Another LP has same solution, is decidable
  - Thm (better news): Easy to write down the LP explicitly in many cases of interest (eg when subscripts are just subsets of indices)
  - Also easy to get upper/lower bounds on s_{HBL}

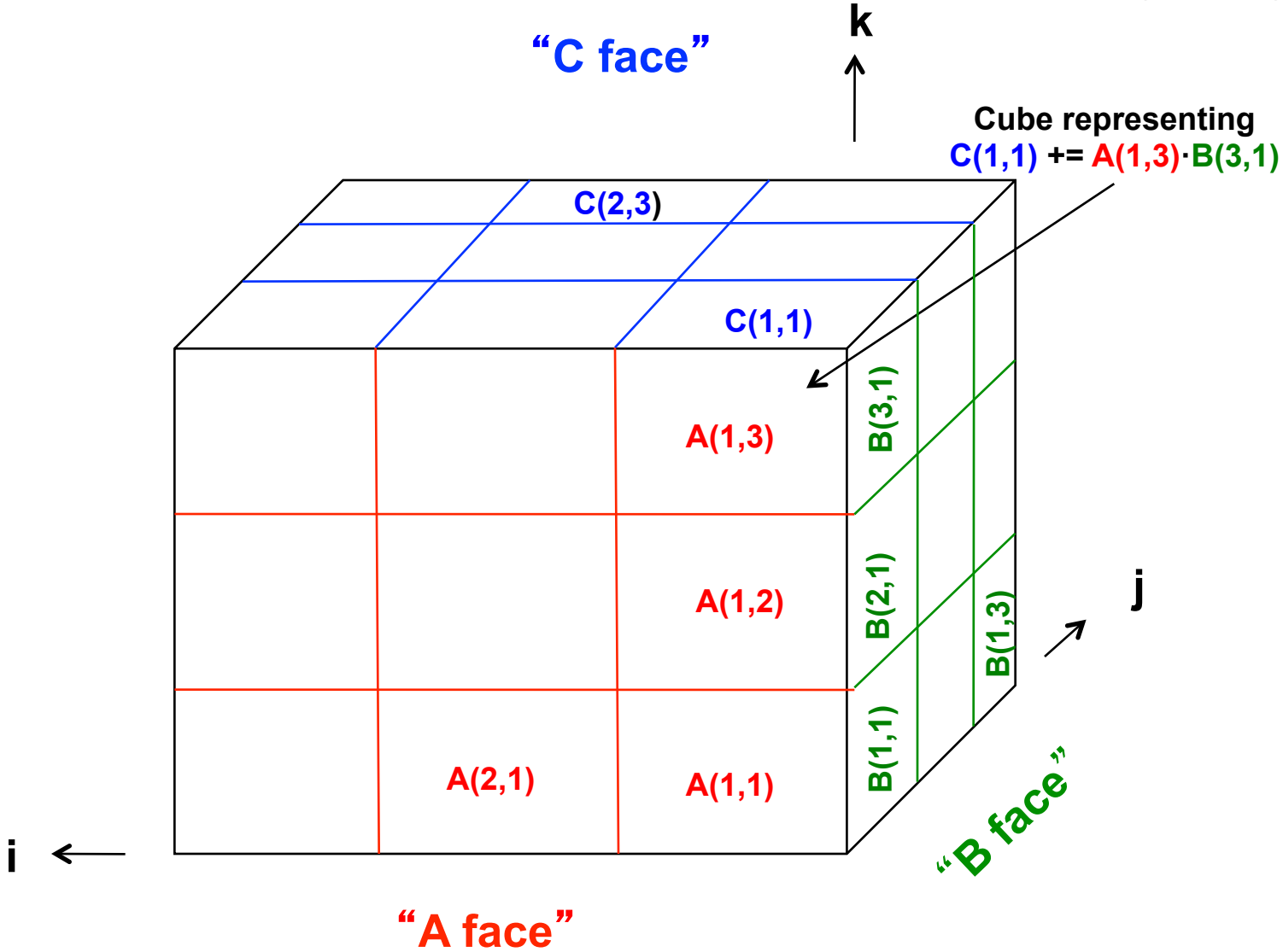
Summary of Results Beyond LA (3/3)

- Can we attain the lower bound?
  - Depends on loop dependencies
  - Best case: none, or reductions (like matmul)
  - Thm: When subscripts are just subsets of indices, the solution x of the dual LP tells us the optimal tile sizes M^{x1}, M^{x2},…,M^{xd}
  - Ex: linear algebra, n-body, join, "random code", …
  - Conjecture: always attainable (modulo dependencies)

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

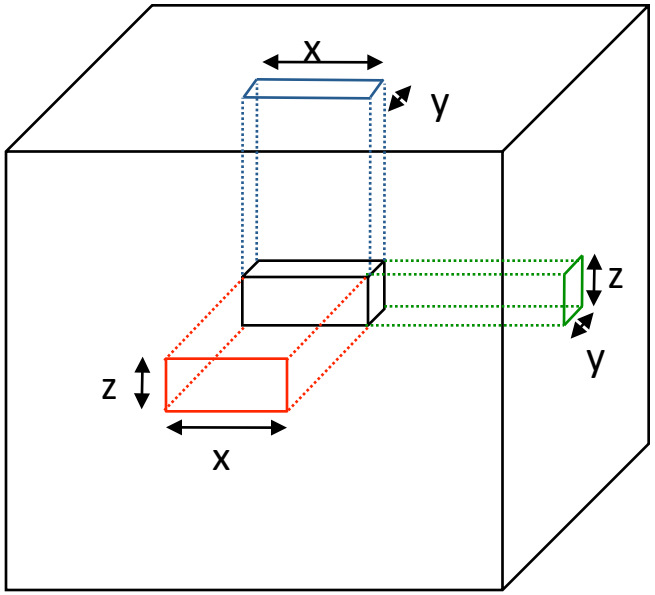
- Originally for C = A*B by Irony/Tiskin/Toledo (2004)
- Proof idea
  - Suppose we can bound #useful_operations ≤ G doable with data in fast memory of size M
  - So to do F = #total_operations, need to fill fast memory F/G times, and so #words_moved ≥ MF/G
- Hard part: finding G
- Attaining lower bound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

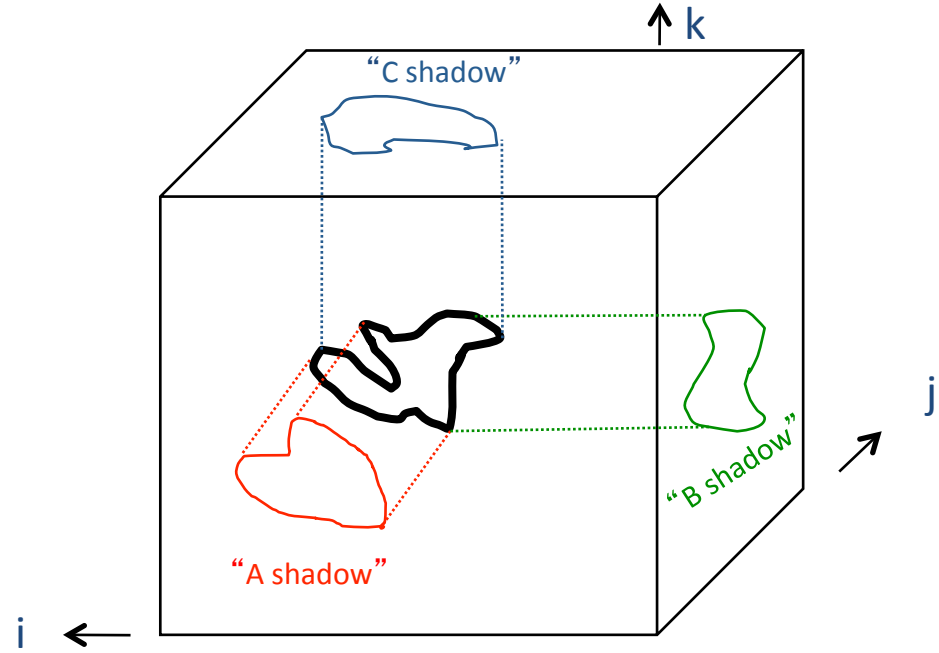
Proof of communication lower bound (2/3)



- 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
= $x \cdot y \cdot z$
= $(xz \cdot zy \cdot yx)^{1/2}$
= $(\#A\square s \cdot \#B\square s \cdot \#C\square s)^{1/2}$
$\leq M^{3/2}$

(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
$\leq$ (area(A shadow) $\cdot$ area(B shadow) $\cdot$ area(C shadow)) $^{1/2}$
$\leq M^{3/2}$

So #words_moved $\geq MF/G \geq F/M^{1/2}$

Generalizing Communication Lower Bounds

- "Thm": Lower bound extends to any algorithm that
  - Iterates over set indexed by i1, i2, i3, … ,ik (eg nested loops)
  - Accesses arrays like A(i1,i2+3*i3-2*i4,i3+4*i5,…), B(pntr(i2),…)
  - Lower bound becomes Ω (#loop_iterations/$M^{S_{HBL}}$)
    - Exponent S_{HBL} is solution of linear program
  - Can write down optimal algorithms in many cases of interest
    - Linear Algebra, N-body, data-base-join, …
    - Can extend "perfect strong scaling in time/energy"
- Conjecture: An algorithm attaining lower bound always exists (modulo dependencies)
  - Eventually: integrate 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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- 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, Num. Stability

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
  - Prerecorded version broadcast in Spring 2013
    - www.xsede.org
    - Free supercomputer accounts to do homework
    - Free autograding of homework

Collaborators and Supporters

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

Summary

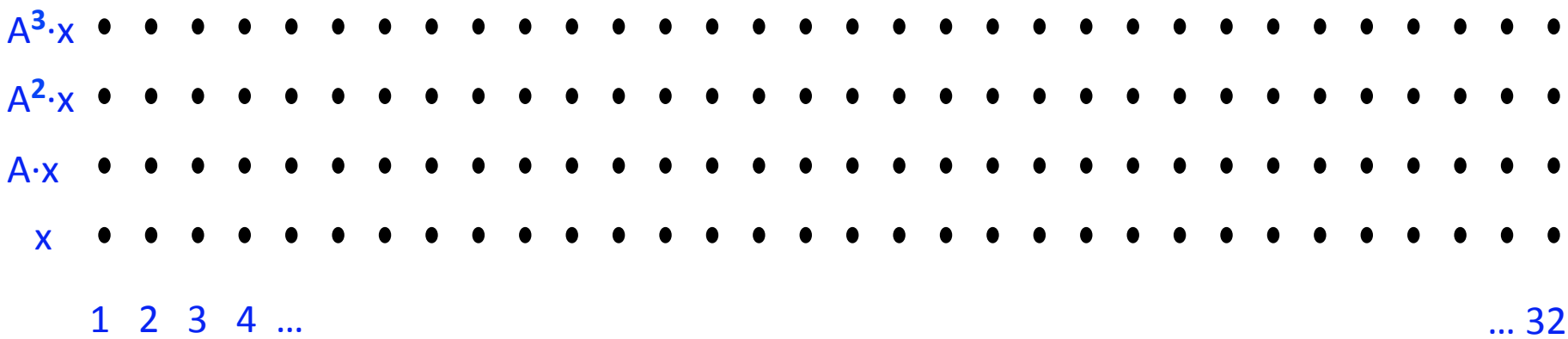
Time to redesign all linear algebra, n-body, …
algorithms and software
(and compilers)

Don’t Communic…

Communication Avoiding Kernels:

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

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

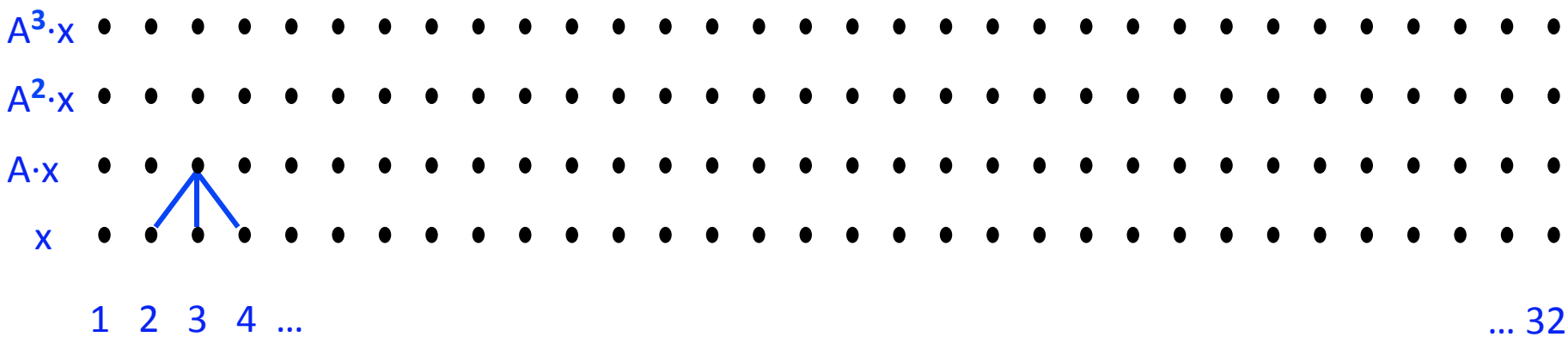


- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:

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

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

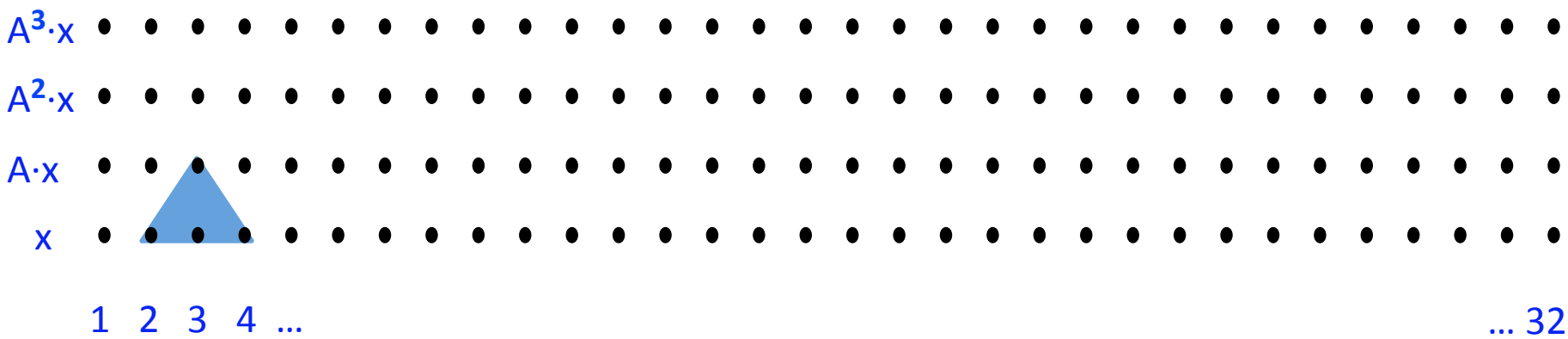


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

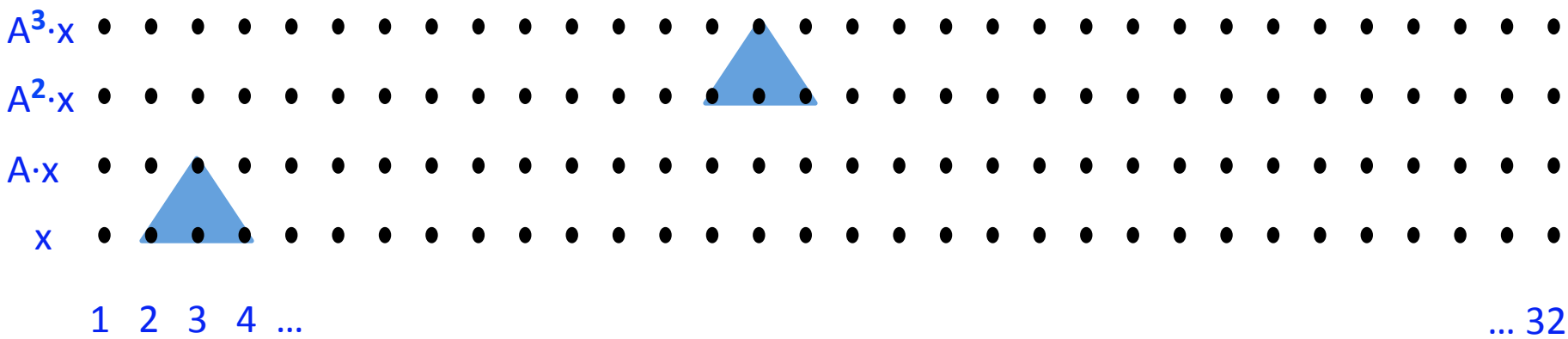


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

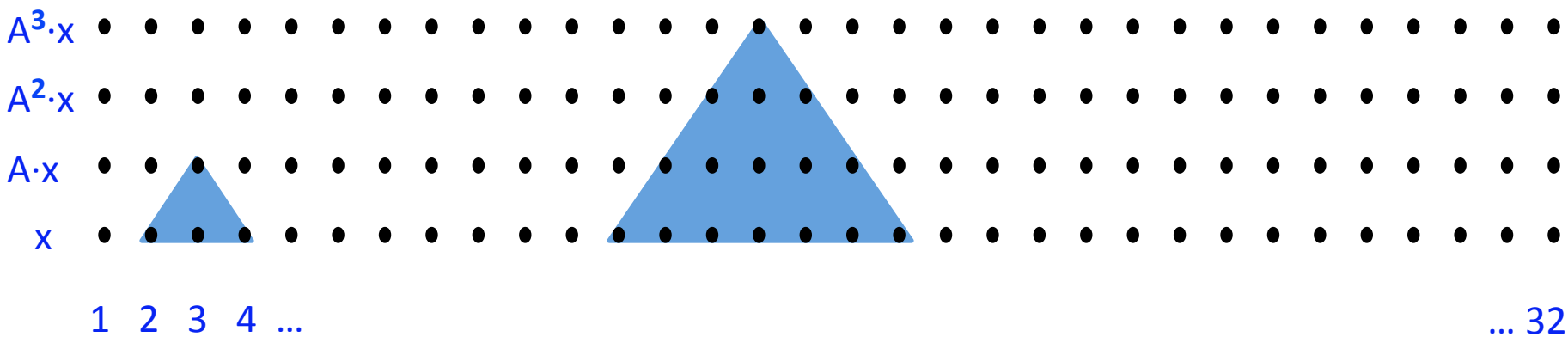


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

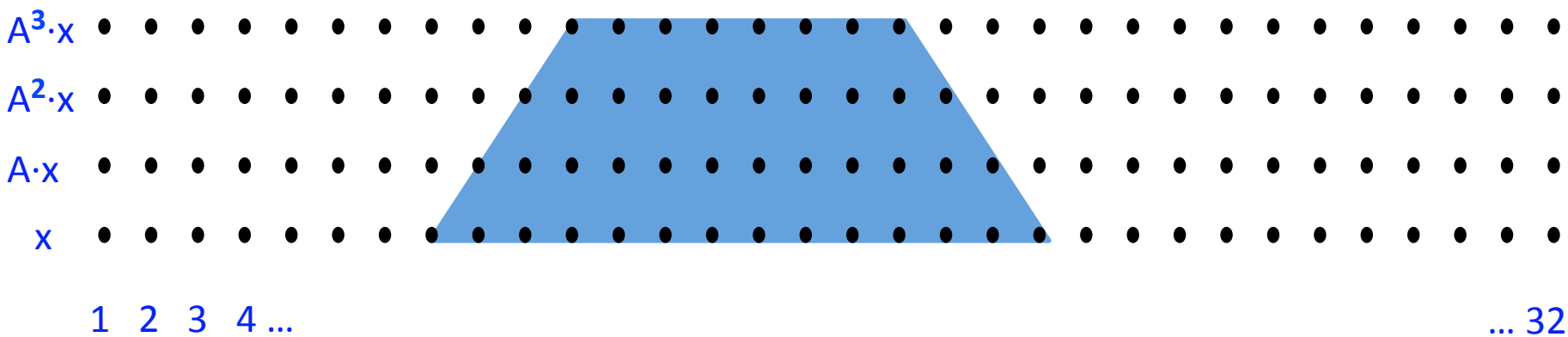


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

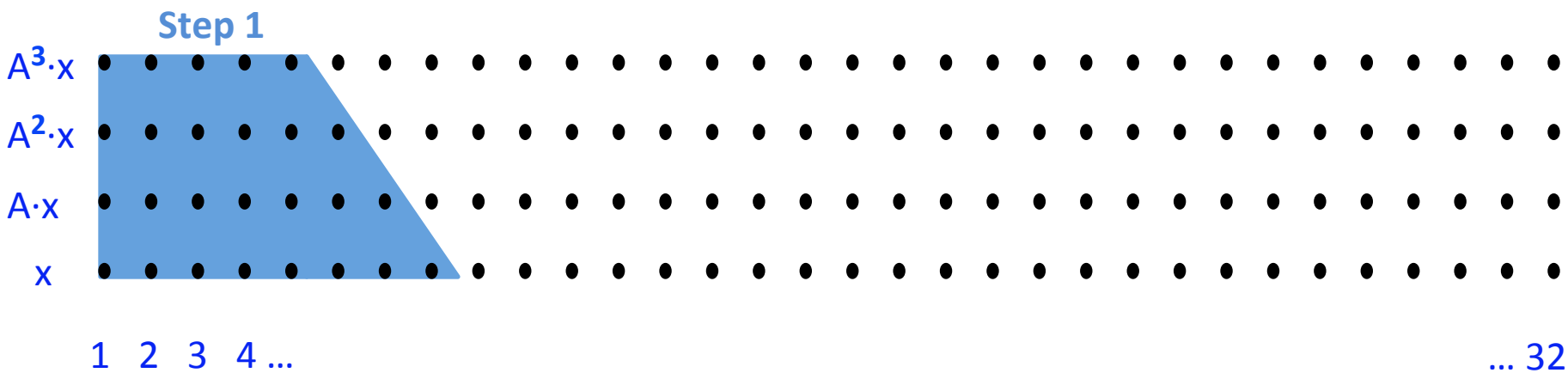


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

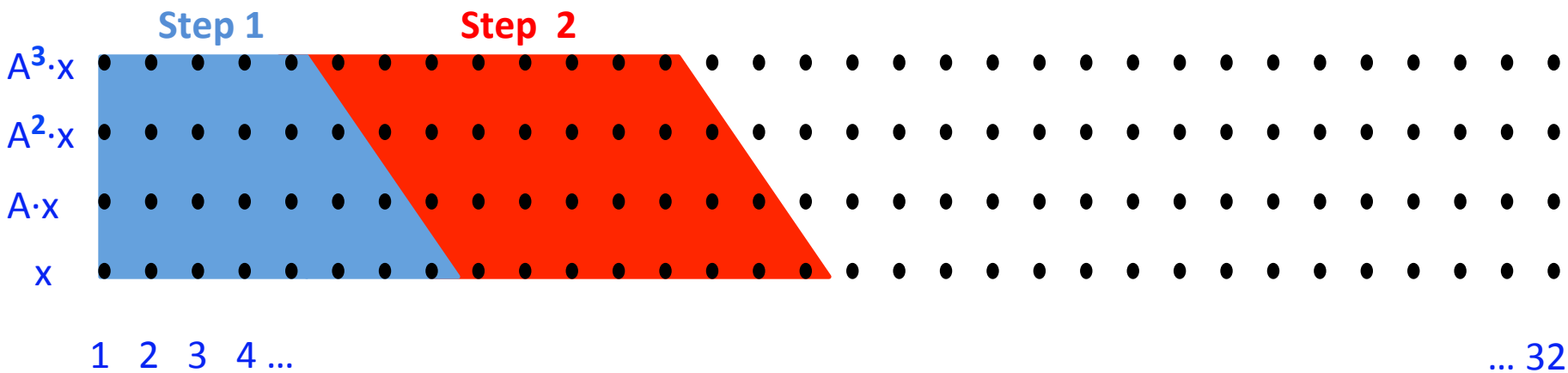


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

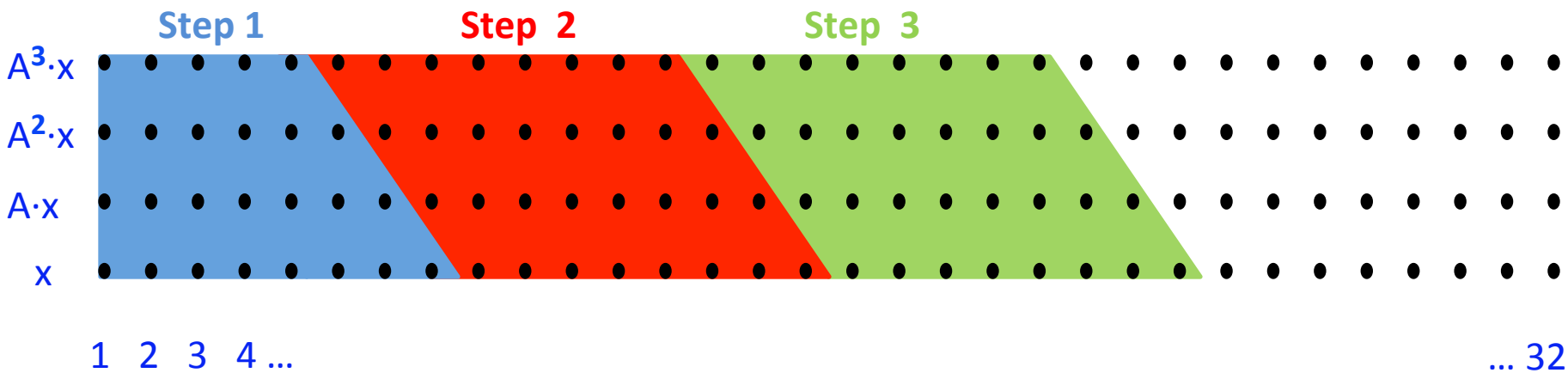


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

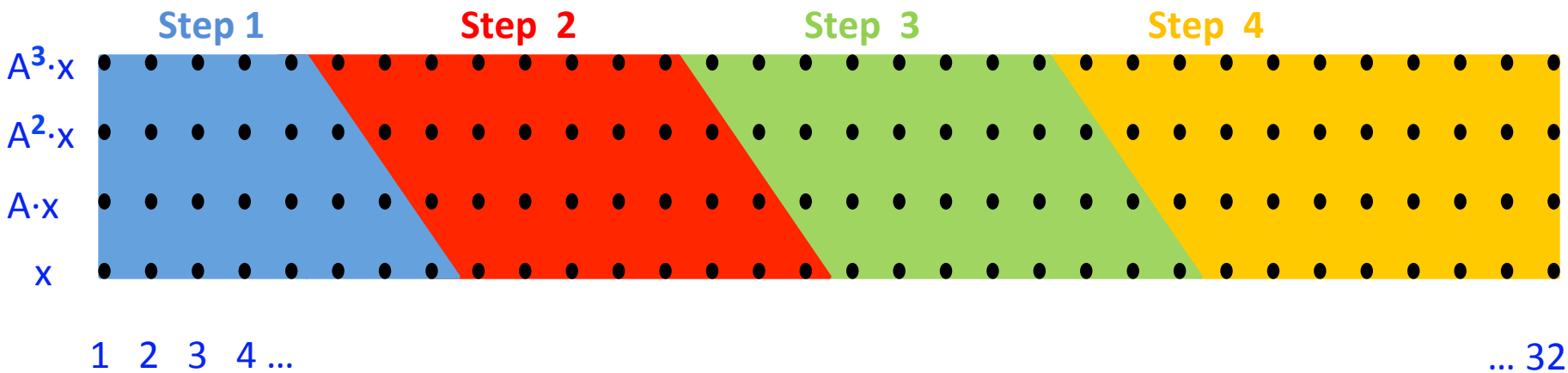


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

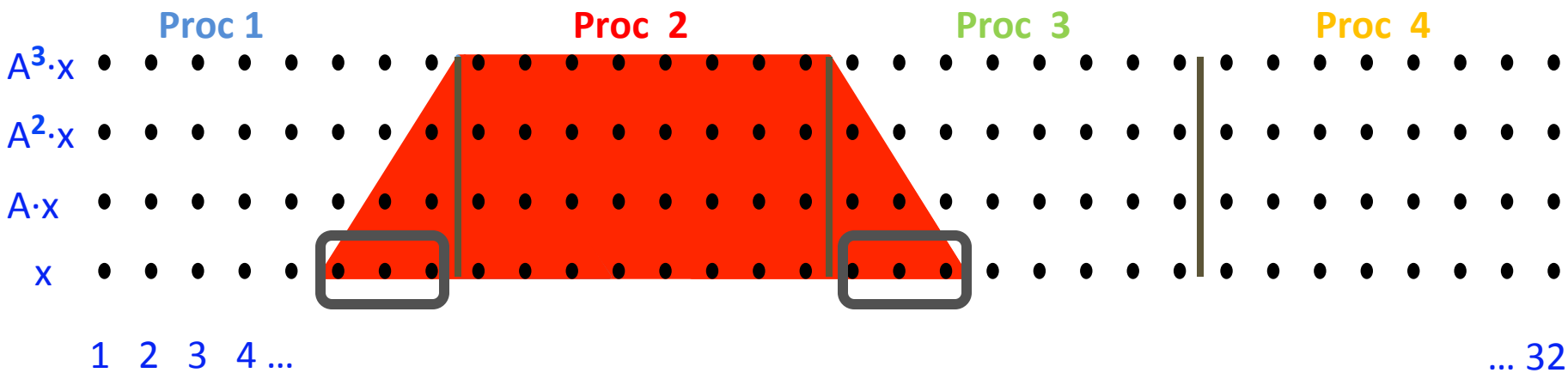


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

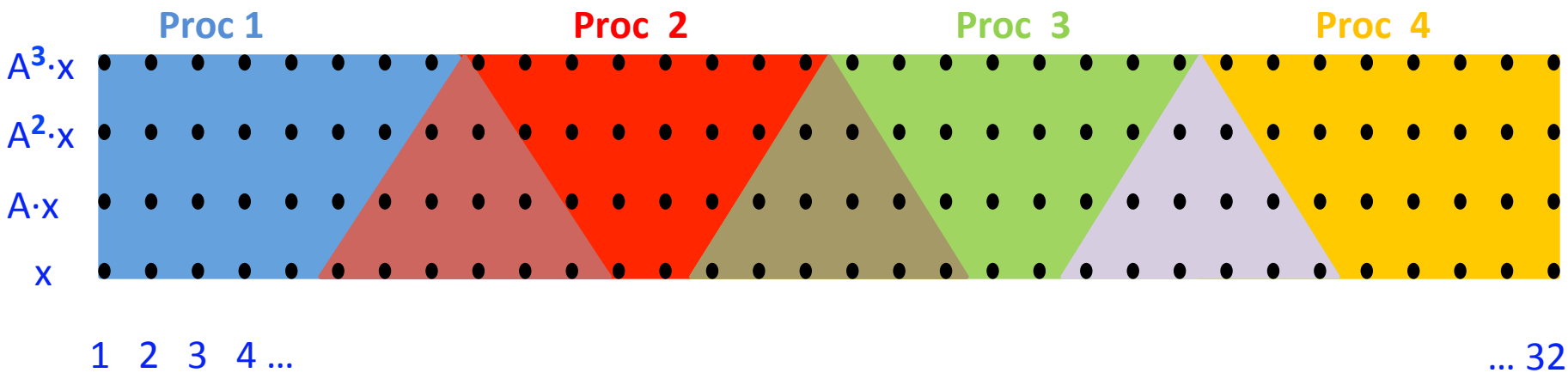


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

Communication Avoiding Kernels:

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

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



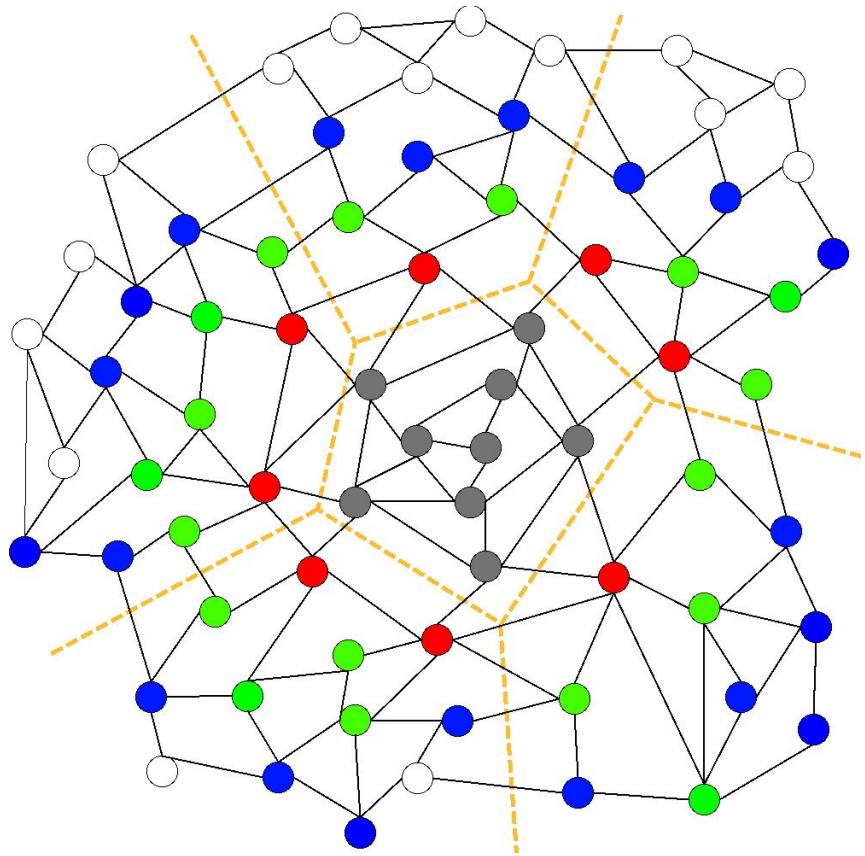
- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

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,…,A^kb} minimizing || Ax-b $||_2$

Standard GMRES

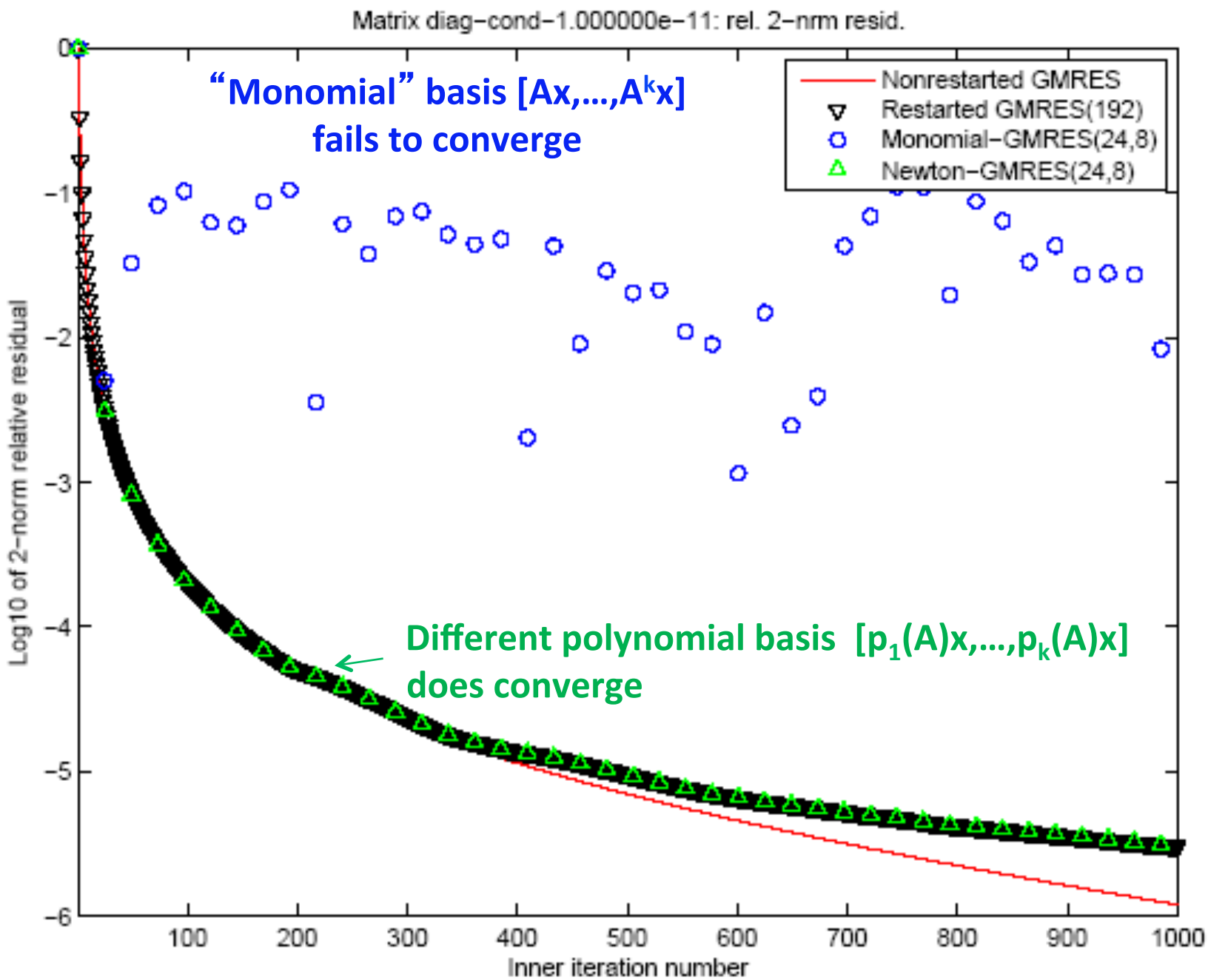
```
for i=1 to k
   w = A · v(i-1)   … SpMV
   MGS(w, v(0),…,v(i-1))
   update v(i), H
endfor
solve LSQ problem with H
```

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

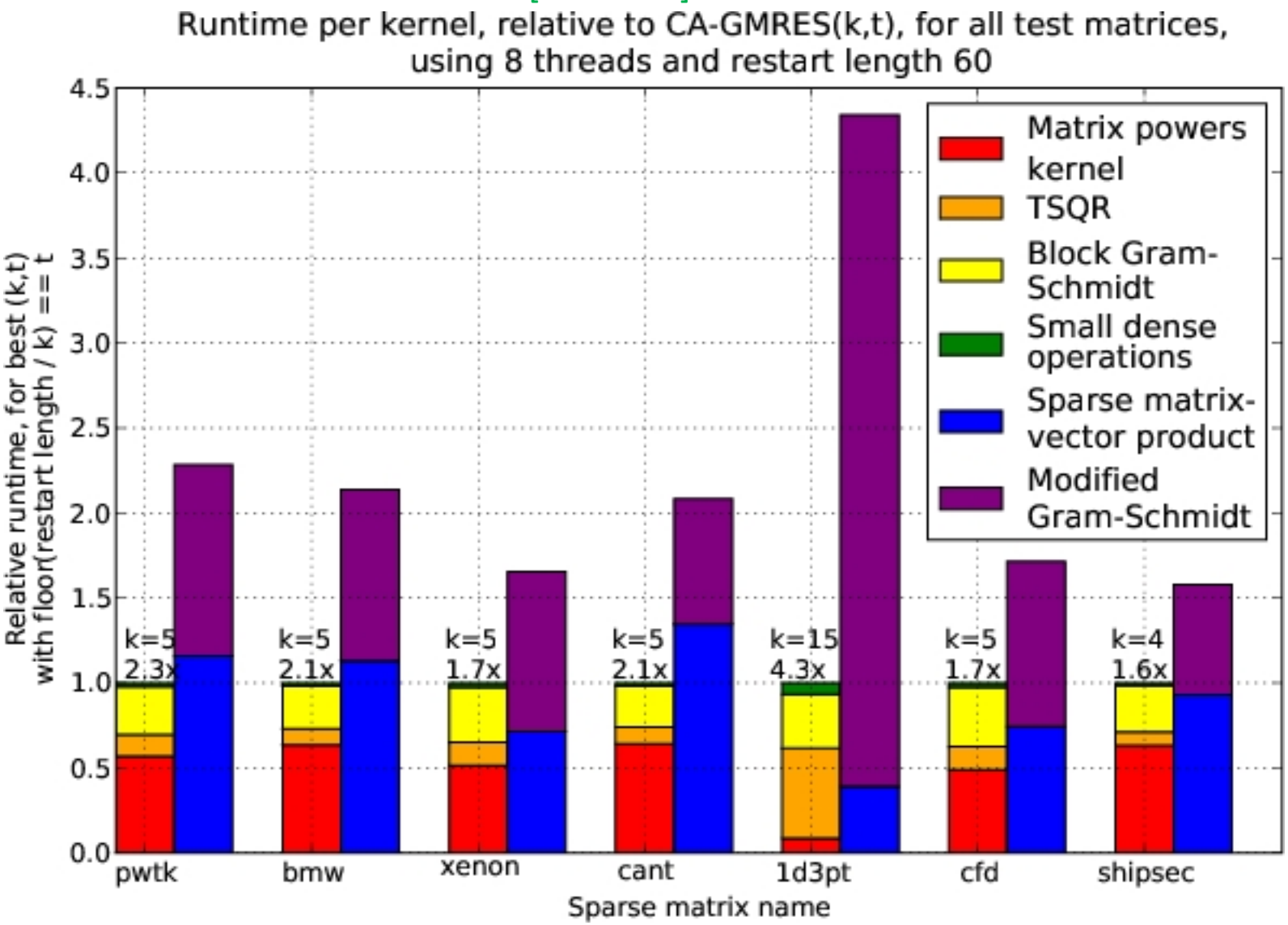
- Oops – W from power method, precision lost!


Matrix diag−cond−1.000000e−11: rel. 2−nrm resid.
"Monomial" basis [Ax,…,A^k x]
fails to converge
Nonrestarted GMRES
Restarted GMRES(192)
Monomial−GMRES(24,8)
Newton−GMRES(24,8)
Different polynomial basis [p_1(A)x,…,p_k(A)x]
does converge
Log10 of 2−norm relative residual
Inner iteration number
0
−1
−2
−3
−4
−5
−6
100
200
300
400
500
600
700
800
900
1000

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

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. $\quad s_j := r_j - \alpha_j Ap_j$
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. $\quad r_{j+1} := s_j - \omega_j As_j$
9. $\quad \beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $\quad p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo

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"