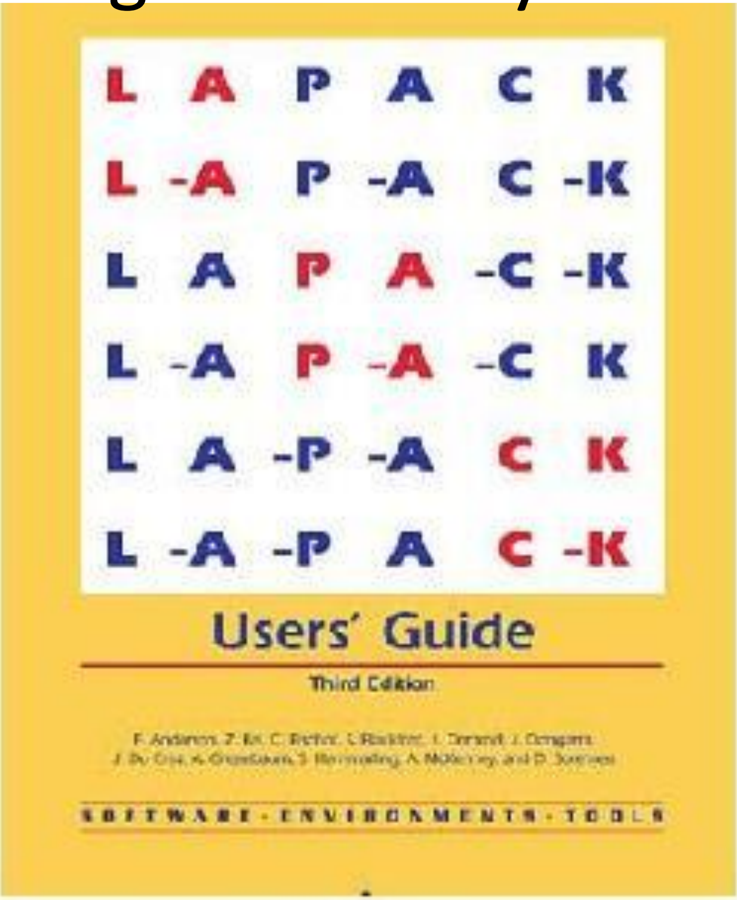
The Past, Present and Future of High Performance Linear Algebra Libraries

Jim Demmel
EECS & Math Departments
UC Berkeley

Supercomputing 10

The Past, the Present …

- LAPACK – sequential/shared memory parallel dense linear algebra library



L A P A C K

L -A P -A C -K

L A P A -C -K

L -A P -A -C K

L A -P -A C K

L -A -P A C -K

Users' Guide

Third Edition

E. Anderson, Z. Bai, C. Bischof, L. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen

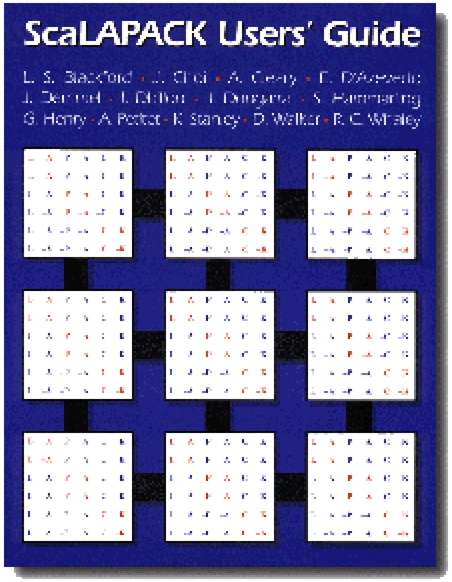
SOFTWARE · ENVIRONMENTS · TOOLS

The Past, the Present …

- LAPACK contributors
  - Jack Dongarra, Julien Langou, Julie Langou, Ed Anderson, Zhaojun Bai, David Bailey, Chris Bischof, Susan Blackford, Zvonomir Bujanovic, Karen Braman, Ralph Byers, Inderjit Dhillon, Zlatko Drmac, Peng Du, Jeremy Du Croz, Mark Fahey, Anne Greenbaum, Ming Gu, Fred Gustavson, Deaglan Halligan, Sven Hammarling, Greg Henry, Yozo Hida, Nick Higham, Bo Kagstrom, William Kahan, Daniel Kressner, Ren-Cang Li, Xiaoye Li, Craig Lucas, Osni Marques, Peter Mayes, Alan McKenney, Beresford Parlett, Antoine Petitet, Peter Poromaa, Enrique Quintana-Orti, Gregorio Quintana-Orti, Giuseppe Radicati, Huan Ren, Jason Riedy, Jeff Rutter, Danny Sorensen, Ken Stanley, Xiaobai Sun, Brian Sutton, Francoise Tisseur, Robert van de Geijn, Kresimir Veselic, Christof Voemel, Jerzy Wasniewski + many undergrads

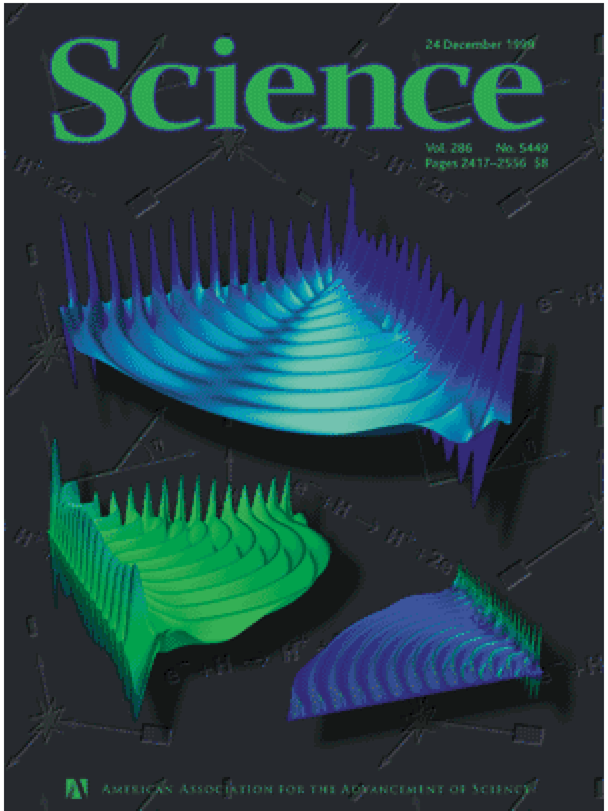
The Past, the Present …

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – distributed memory parallel dense linear algebra library
    - Jack Dongarra, Susan Blackford, Jaeyoung Choi, Andy Cleary, Ed D’Azevedo, Inderjit Dhillon, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David Walker, Clint Whaley, and many other contributors


ScaLAPACK Users' Guide
L. S. Blackford · J. Choi · A. Cleary · E. D'Azevedo
J. Demmel · I. Dhillon · J. Dongarra · S. Hammarling
G. Henry · A. Petitet · K. Stanley · D. Walker · R. C. Whaley

The Past, the Present …

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for Ax=b
  - Xiaoye Sherry Li



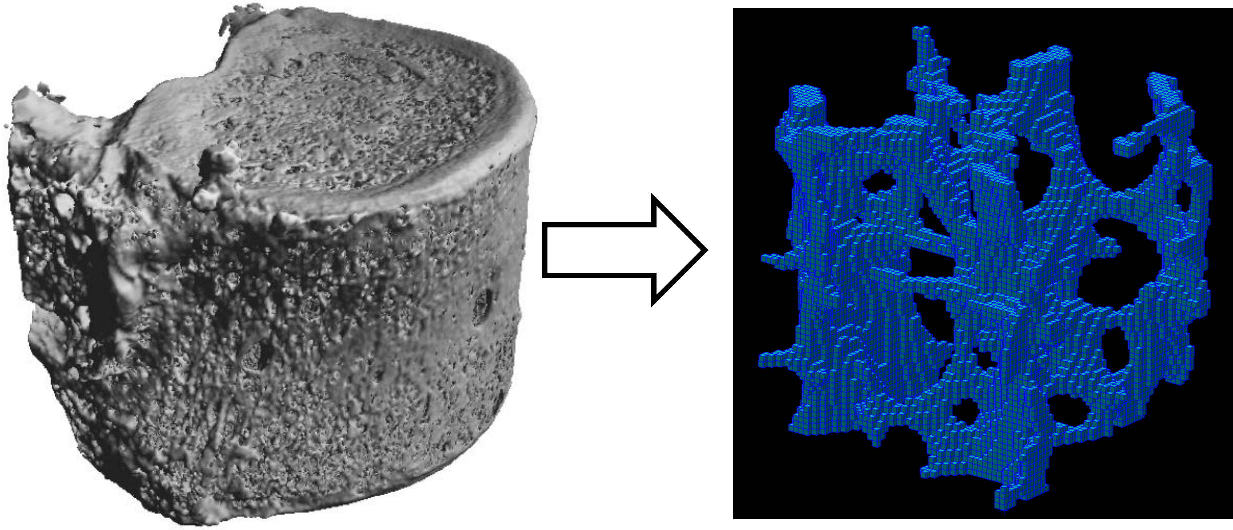
The Past, the Present …

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for Ax=b
- Autotuning
  - PhiPAC for matrix multiplication
    - Jeff Bilmes, Krste Asanovic, Rich Vuduc, Sriram Iyer, CheeWhye Chin, Dominic Lam

  - OSKI for sparse-matrix-vector multiplication (SpMV)
    - Rich Vuduc, Kathy Yelick, Rajesh Nishtala, Ben Lee, Shoaib Kamil, Jen Hsu

The Past, the Present …

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for Ax=b
- Autotuning – matmul and SpMV
- Prometheus – parallel unstructured FE solver
  - Mark Adams
  - Gordon Bell Prize, 2004



The Past, the Present …

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for Ax=b
- Autotuning - matmul and SpMV
- Prometheus – parallel unstructured FE solver
- CS267 – on-line course on parallel computing
  - Kathy Yelick, David Culler, Horst Simon
  - Google “parallel computing course” (I’m feeling lucky)

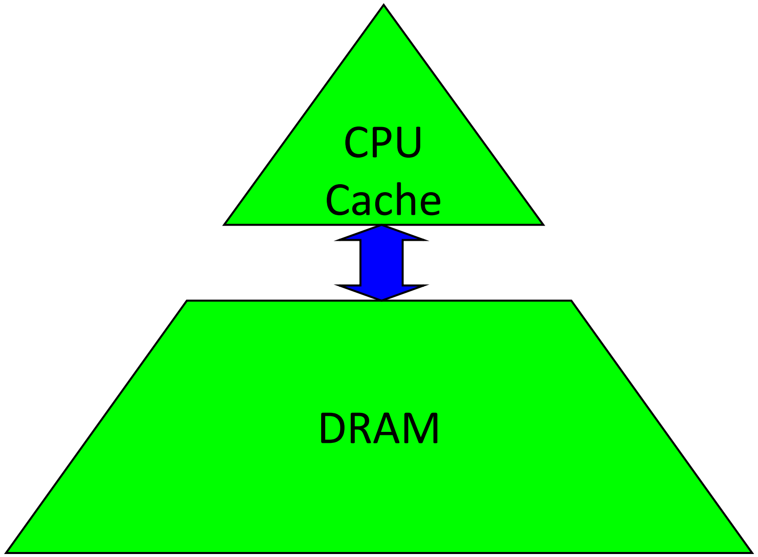
… The Future

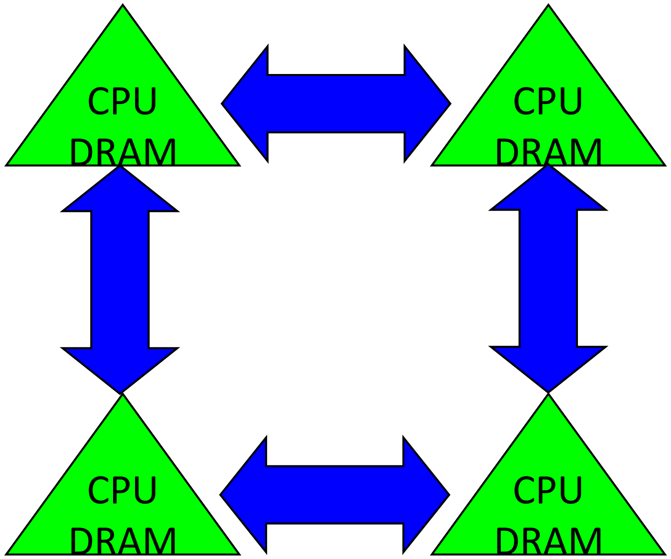
- Why we need to “avoid communication,” i.e. avoid moving data
- “Direct” Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - New algorithms that attain these lower bounds
    - *Not* in libraries like Sca/LAPACK (yet!)
    - Large speed-ups possible
- “Iterative” Linear Algebra
  - Ditto for Krylov Subspace Methods

Why avoid communication? (1/2)

Algorithms have two costs:

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





Why avoid communication? (2/2)

- 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, 2004)

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

- Goal : reorganize linear algebra to *avoid* communication
  - Between all memory hierarchy levels
    - L1 ⟷ L2 ⟷ DRAM ⟷ network, etc
  - Not just *hiding* communication (speedup $\leq$ 2x)
  - Arbitrary speedups possible

Direct linear algebra: Prior Work on Matmul

- Assume n^3 algorithm for C=A*B (i.e. not Strassen-like)
- Sequential case, with fast memory of size M
  - Lower bound on #words moved to/from slow memory $= \Omega(n^3 / M^{1/2})$ [Hong, Kung, 81]
  - Attained using “blocked” or cache-oblivious algorithms

- Parallel case on P processors:
  - Assume load balanced, one copy each of A, B, C
  - Lower bound on #words communicated $= \Omega(n^2 / P^{1/2})$ [Irony, Tiskin, Toledo, 04]
  - Attained by Cannon’s Algorithm

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size per processor
- Parallel case: assume either load or memory balanced

#words_moved by at least one processor =
Ω(#flops_per_processor / $M^{1/2}$)

#messages_sent by at least one processor =
Ω(#flops_per_processor / $M^{3/2}$)

- Holds for
  - 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)

Can we attain these lower bounds?

- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Mostly not
- If not, are there other algorithms that do?
  - Yes, for dense linear algebra
- Only a few sparse algorithms so far (eg Cholesky)

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

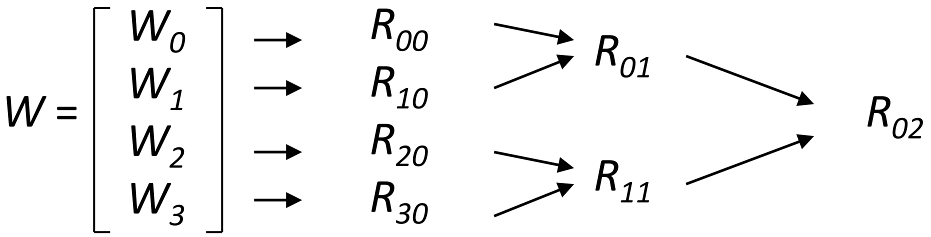
$$W = \left(\begin{array}{c} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{array}\right) = \left(\begin{array}{c} Q_{00}\,R_{00} \\ \hline Q_{10}\,R_{10} \\ \hline Q_{20}\,R_{20} \\ \hline Q_{30}\,R_{30} \end{array}\right) = \left(\begin{array}{cccc} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline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

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

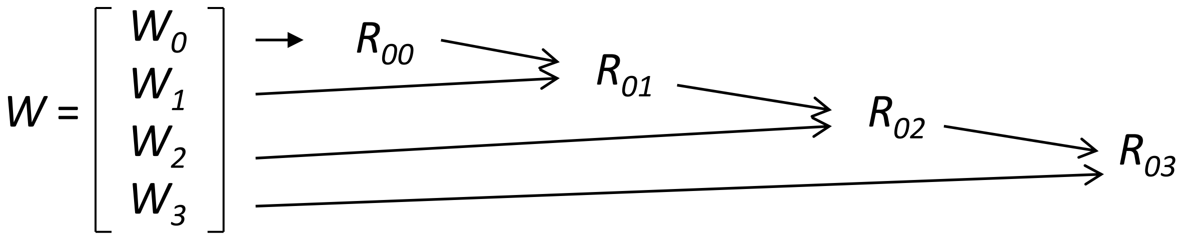
$$\left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right) = \left(Q_{02}\,R_{02} \right)$$

Minimizing Communication in TSQR

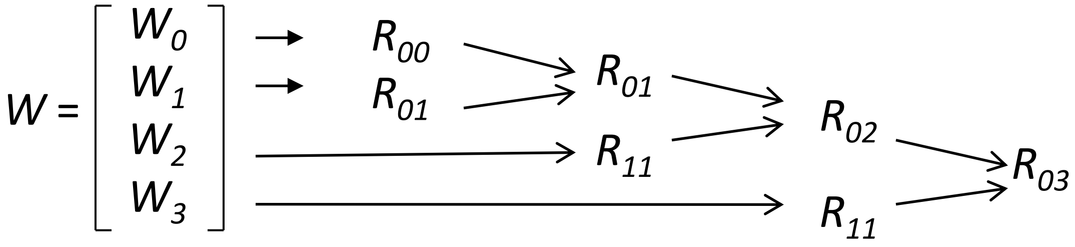
Parallel:



Sequential:



Dual Core:



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

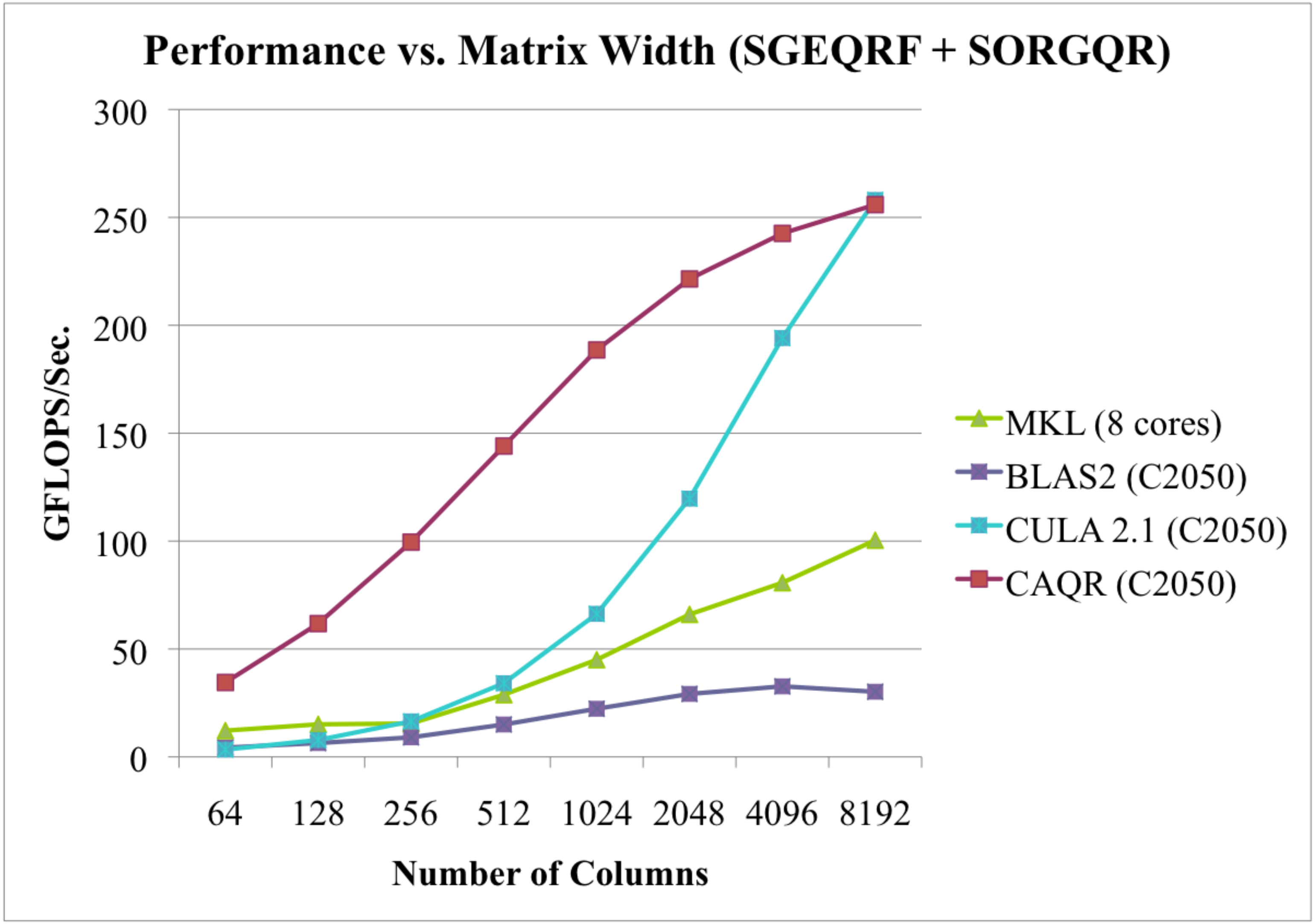
Can choose reduction tree dynamically

TSQR Performance Results

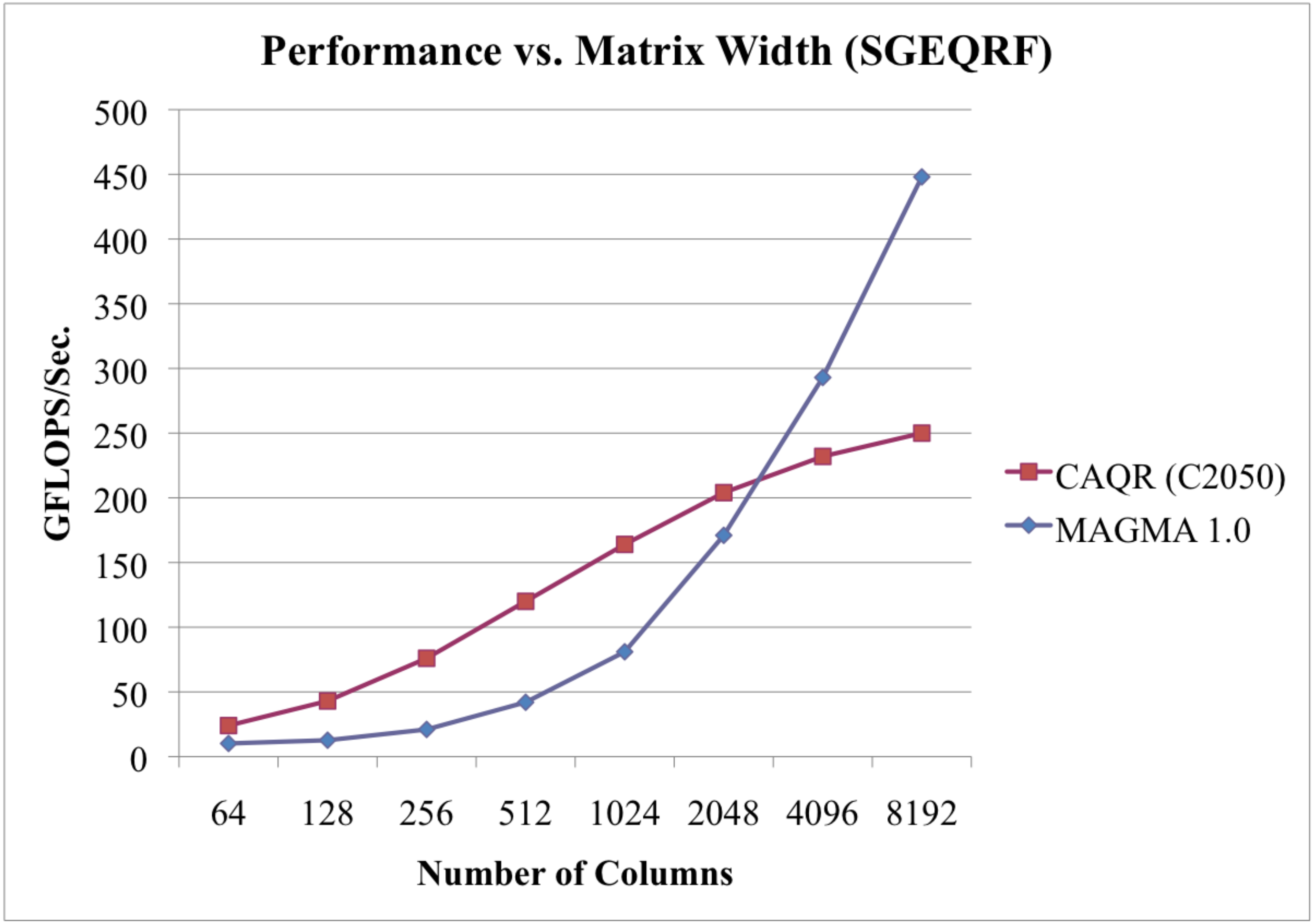
- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Grid – **4x** on 4 cities (Dongarra et al)
  - Cloud – early result – up and running using Mesos
- 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
- What about QR for a general matrix?

Data from Grey Ballard, Mark Hoemmen, Laura Grigori, Julien Langou, Jack Dongarra, Michael Anderson

CAQR on a GPU (Fermi C2050) (1/2)
# rows = 8192

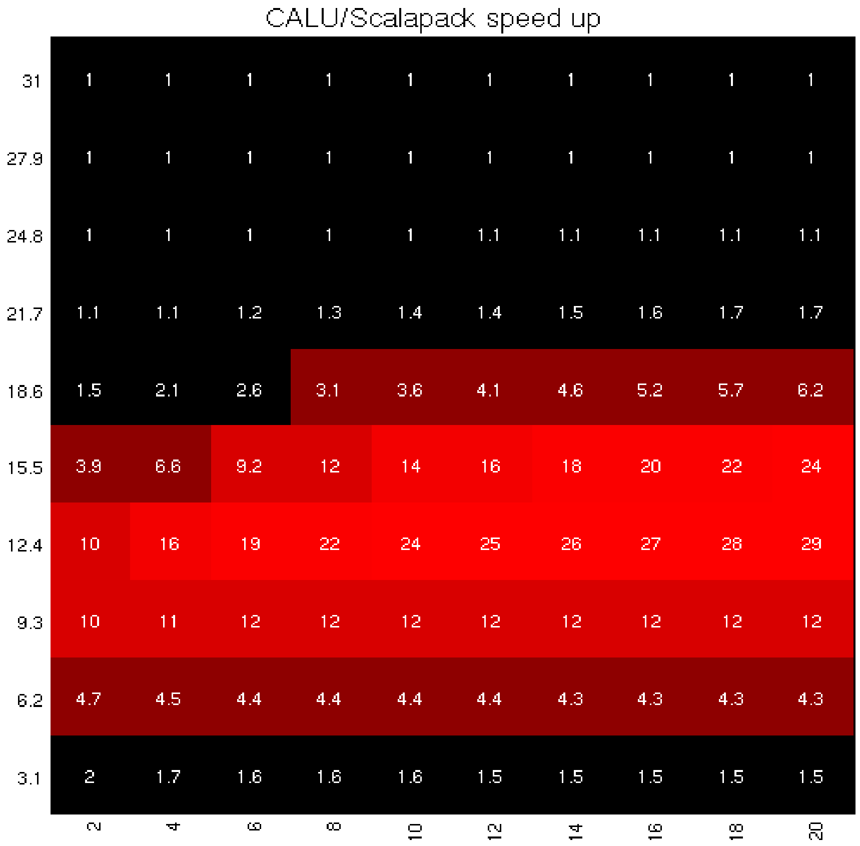


CAQR on a GPU (Fermi C2050) (2/2)
#rows = 8192



Biggest speedup over MAGMA QR is 13x for 1M x 192

Preliminary Exascale predicted speedups for CA-LU vs ScaLAPACK-LU



CALU/Scalapack speed up

	2	4	6	8	10	12	14	16	18	20
31	1	1	1	1	1	1	1	1	1	1
27.9	1	1	1	1	1	1	1	1	1	1
24.8	1	1	1	1	1	1.1	1.1	1.1	1.1	1.1
21.7	1.1	1.1	1.2	1.3	1.4	1.4	1.5	1.6	1.7	1.7
18.6	1.5	2.1	2.6	3.1	3.6	4.1	4.6	5.2	5.7	6.2
15.5	3.9	6.6	9.2	12	14	16	18	20	22	24
12.4	10	16	19	22	24	25	26	27	28	29
9.3	10	11	12	12	12	12	12	12	12	12
6.2	4.7	4.5	4.4	4.4	4.4	4.4	4.3	4.3	4.3	4.3
3.1	2	1.7	1.6	1.6	1.6	1.5	1.5	1.5	1.5	1.5

$\log_2 (n^2/p) =$

$\log_2$ (memory_per_proc)

$\log_2$ (p)

Summary of Direct Linear Algebra

- New Lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - New ways to “pivot”
  - Eigenvalues and SVD
  - Exploiting extra memory for better strong scaling
  - Heterogeneous architectures
  - Strassen-like algorithms
  - Autotuning

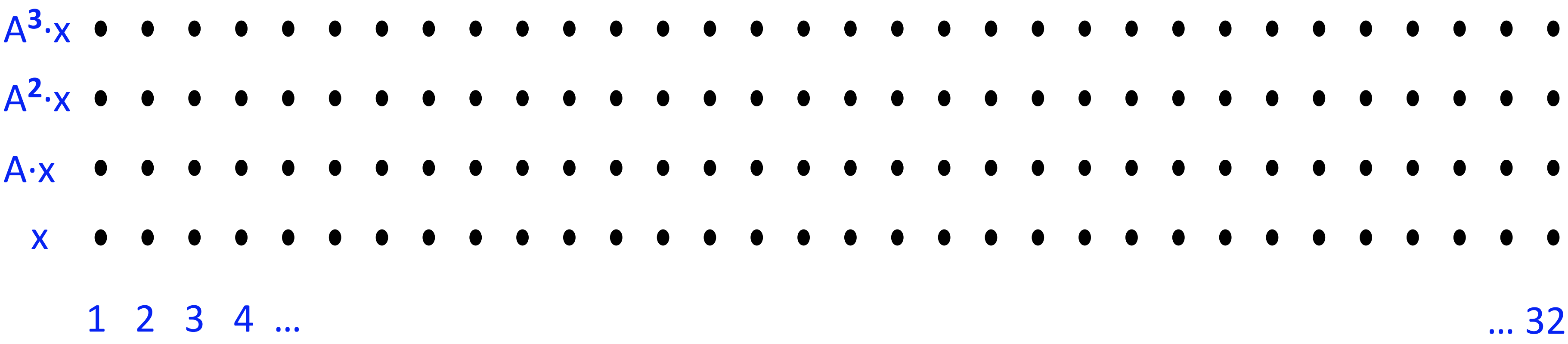
Avoiding Communication in Iterative Linear Algebra

- 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, 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)
    - **New: O(log p) messages - optimal**
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation

Communication Avoiding Kernels:

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

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

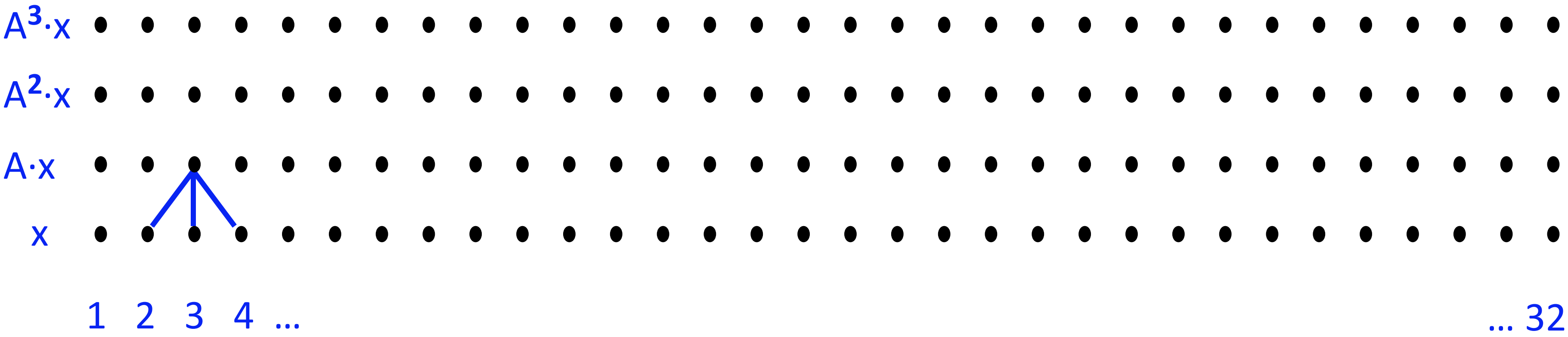


- Example: A tridiagonal, n=32, k=3
- Works for any “well-partitioned” A

Communication Avoiding Kernels:

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

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

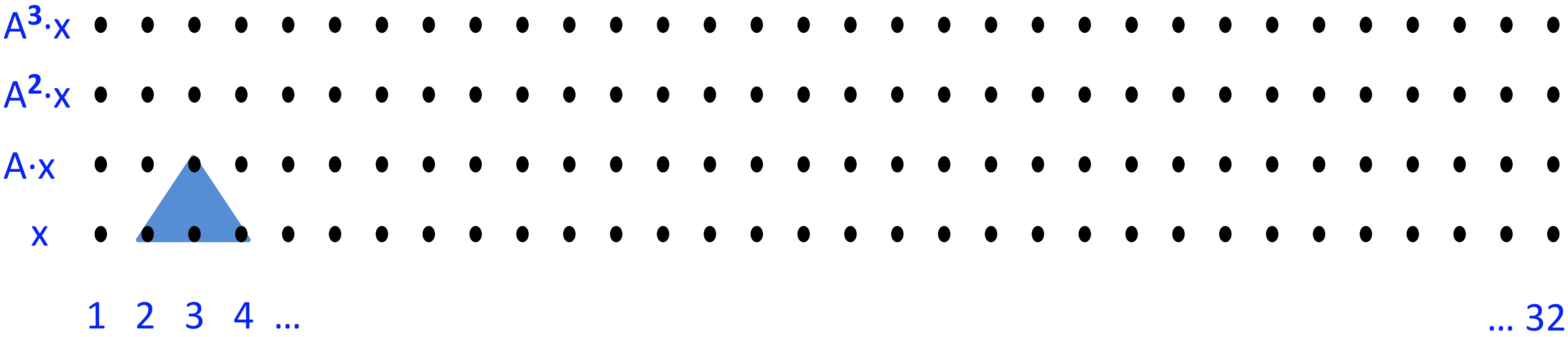


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

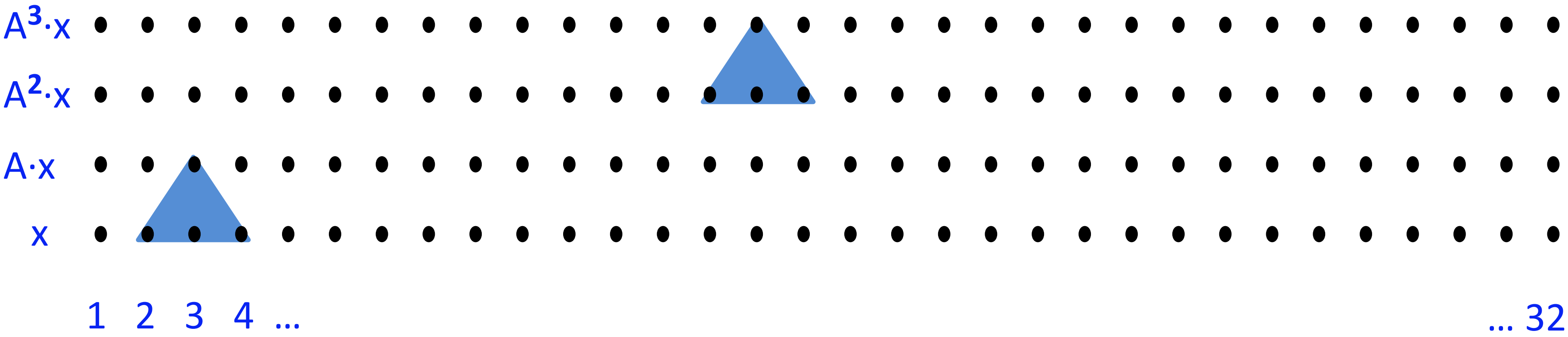


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

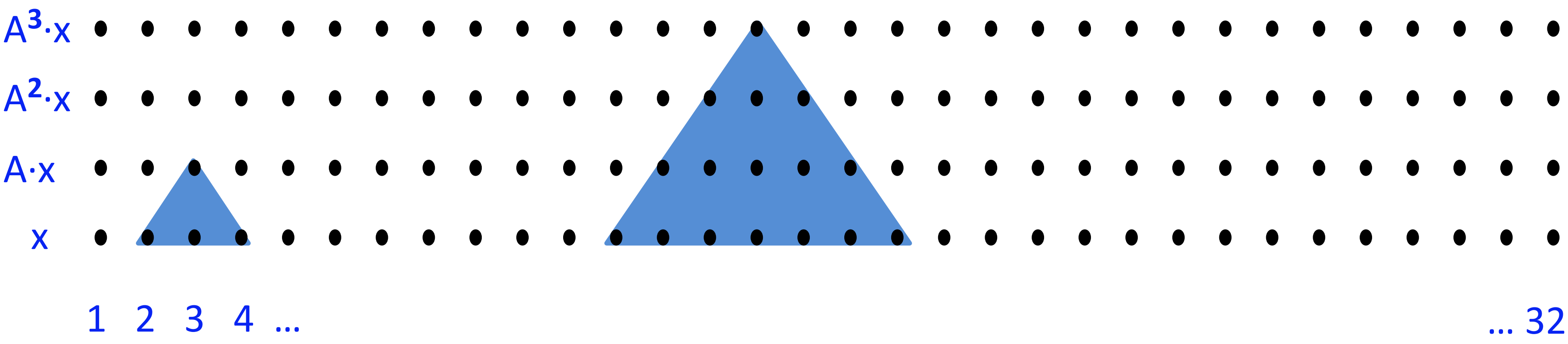


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

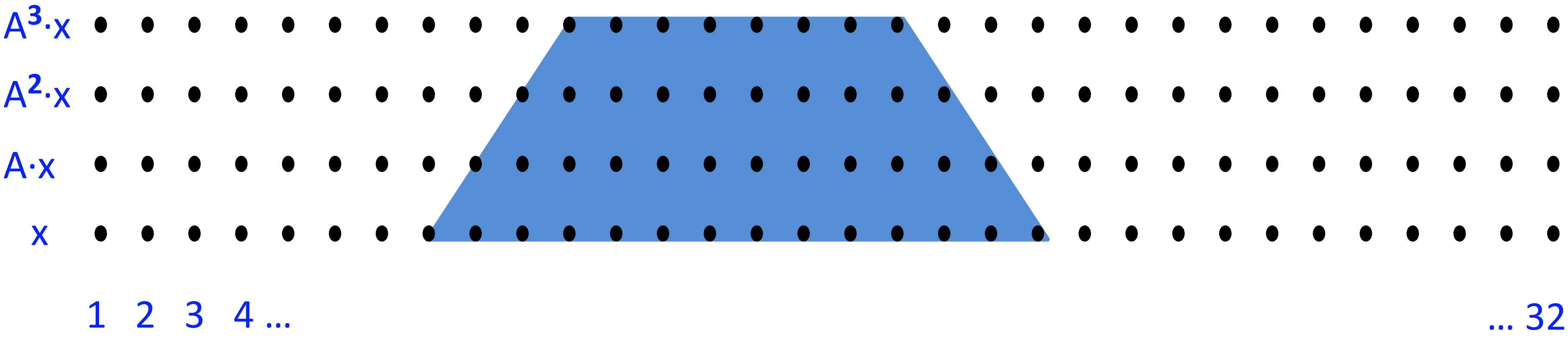


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

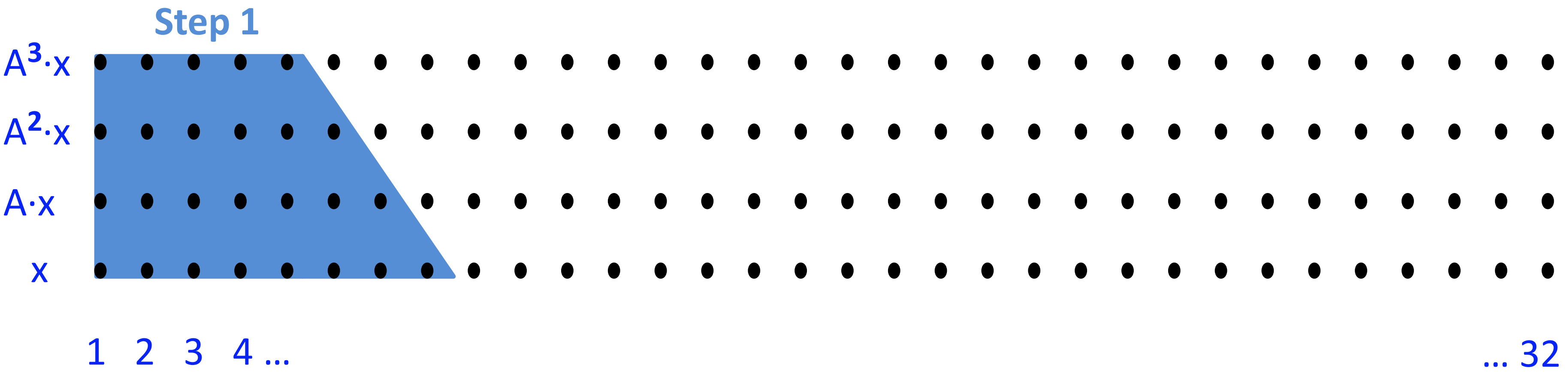


- Example: A tridiagonal, n=32, k=3

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]
- Sequential Algorithm

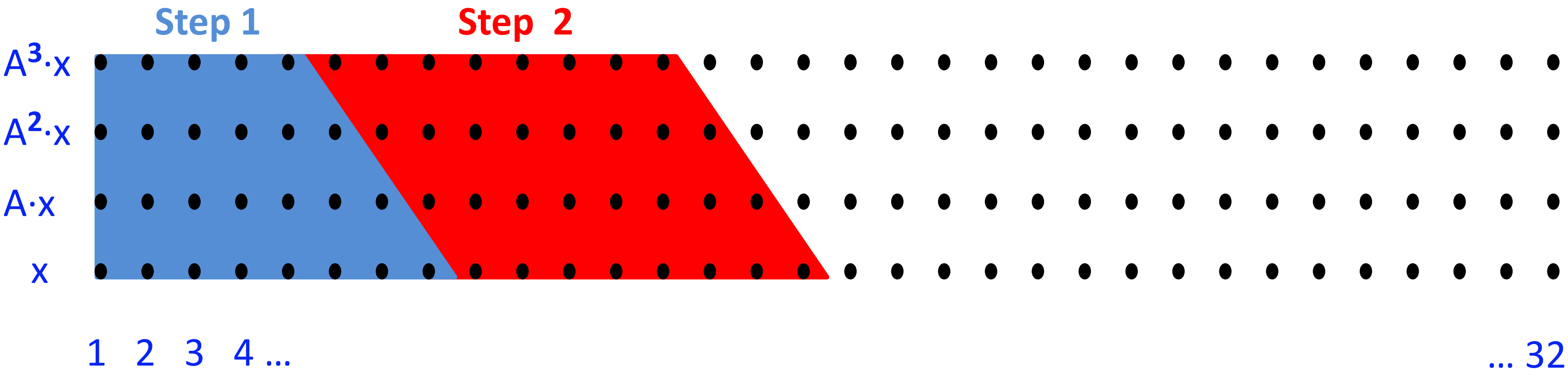


- Example: A tridiagonal, n=32, k=3

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

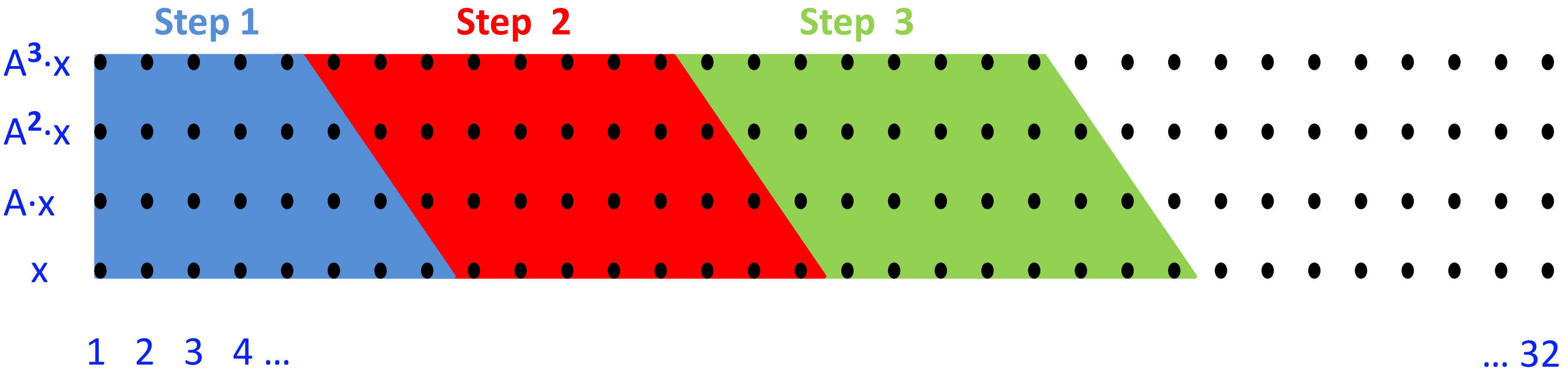


- Example: A tridiagonal, n=32, k=3

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

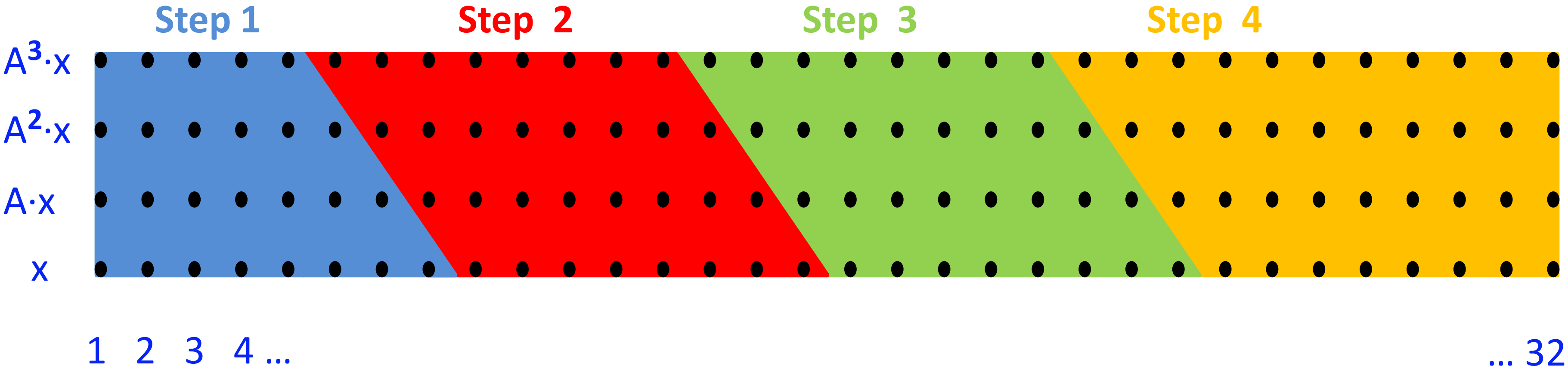


- Example: A tridiagonal, n=32, k=3

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

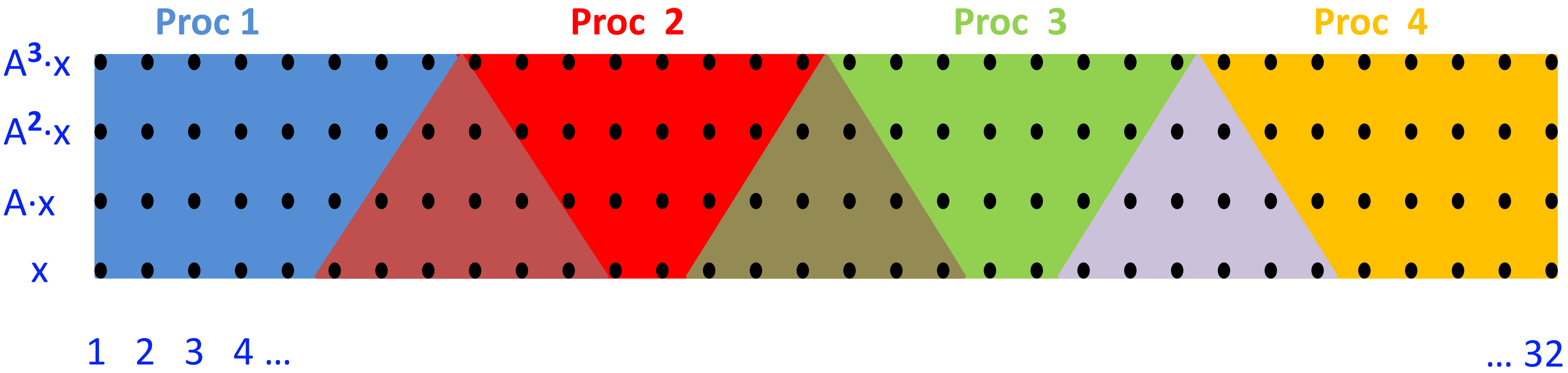


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

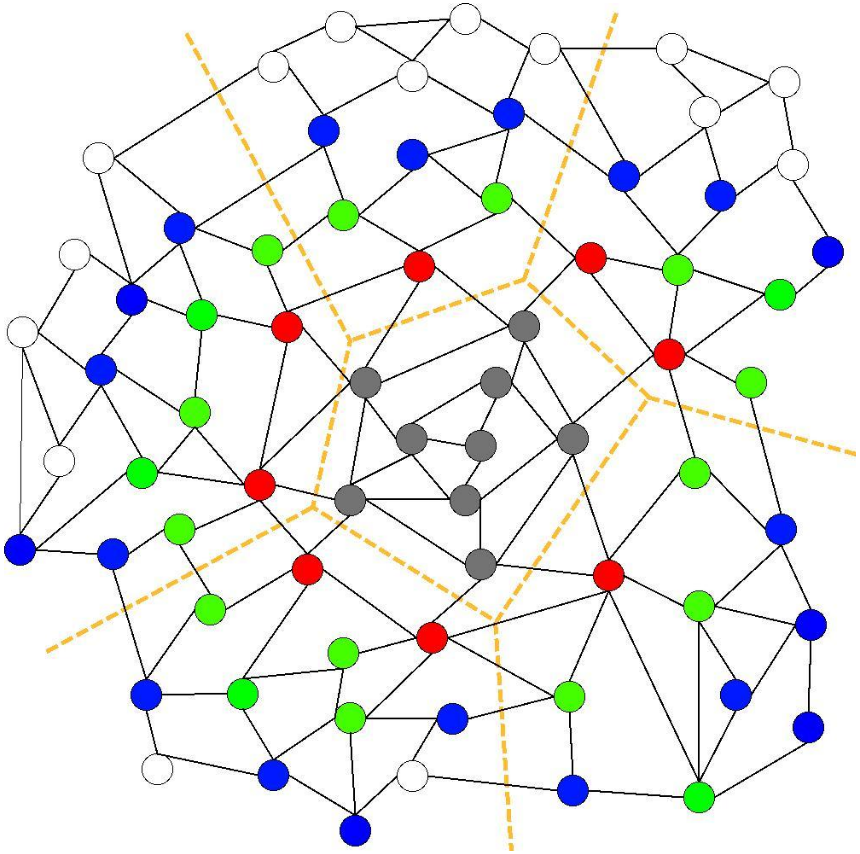


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

Remotely Dependent Entries for [x,Ax,A^2x,A^3x], A irregular, multiple processors



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
```

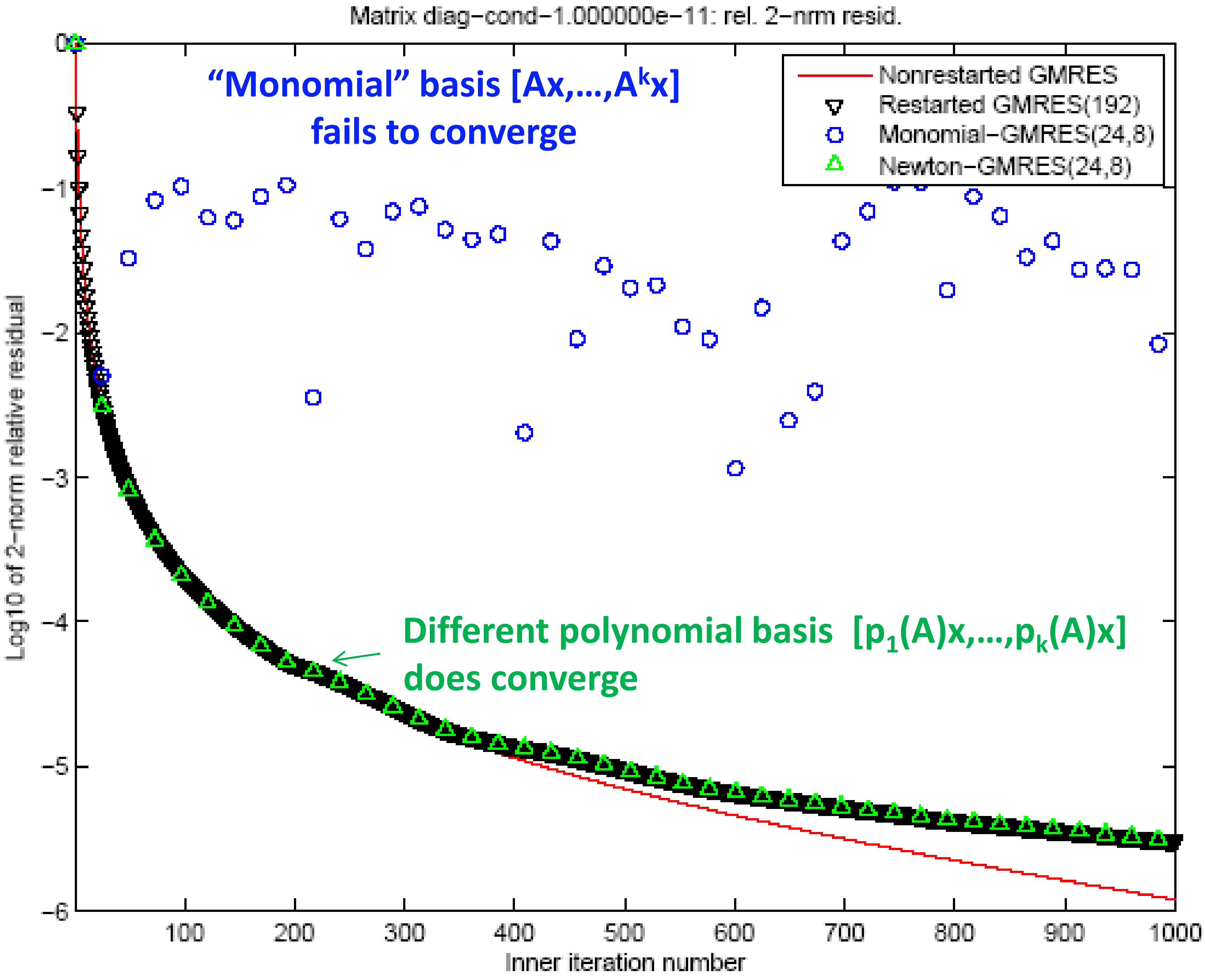
Communication-avoiding GMRES

```
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!



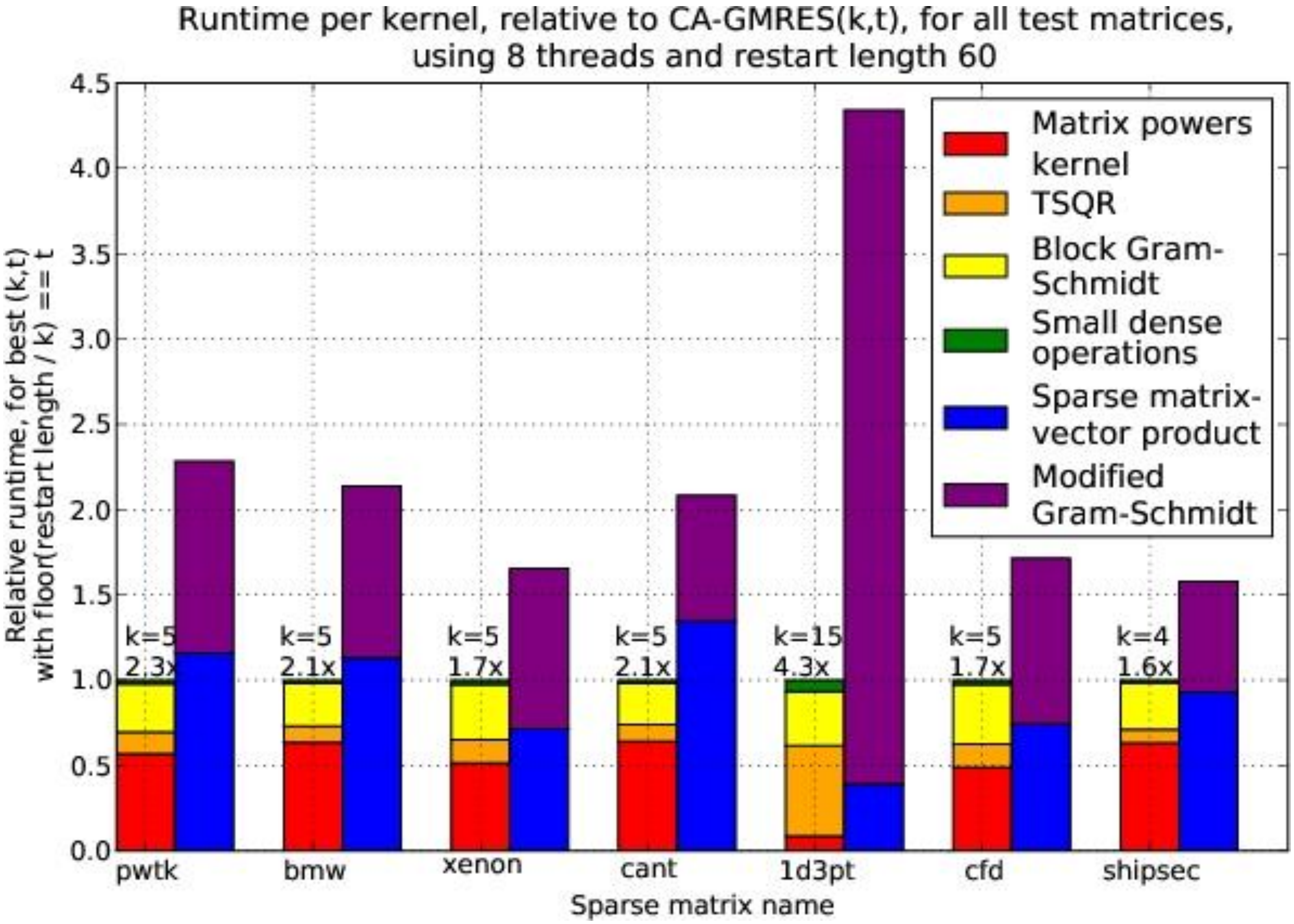
“Monomial” basis [Ax,…,A^{k}x] fails to converge

Different polynomial basis [p$_1$(A)x,…,p$_k$(A)x] does converge

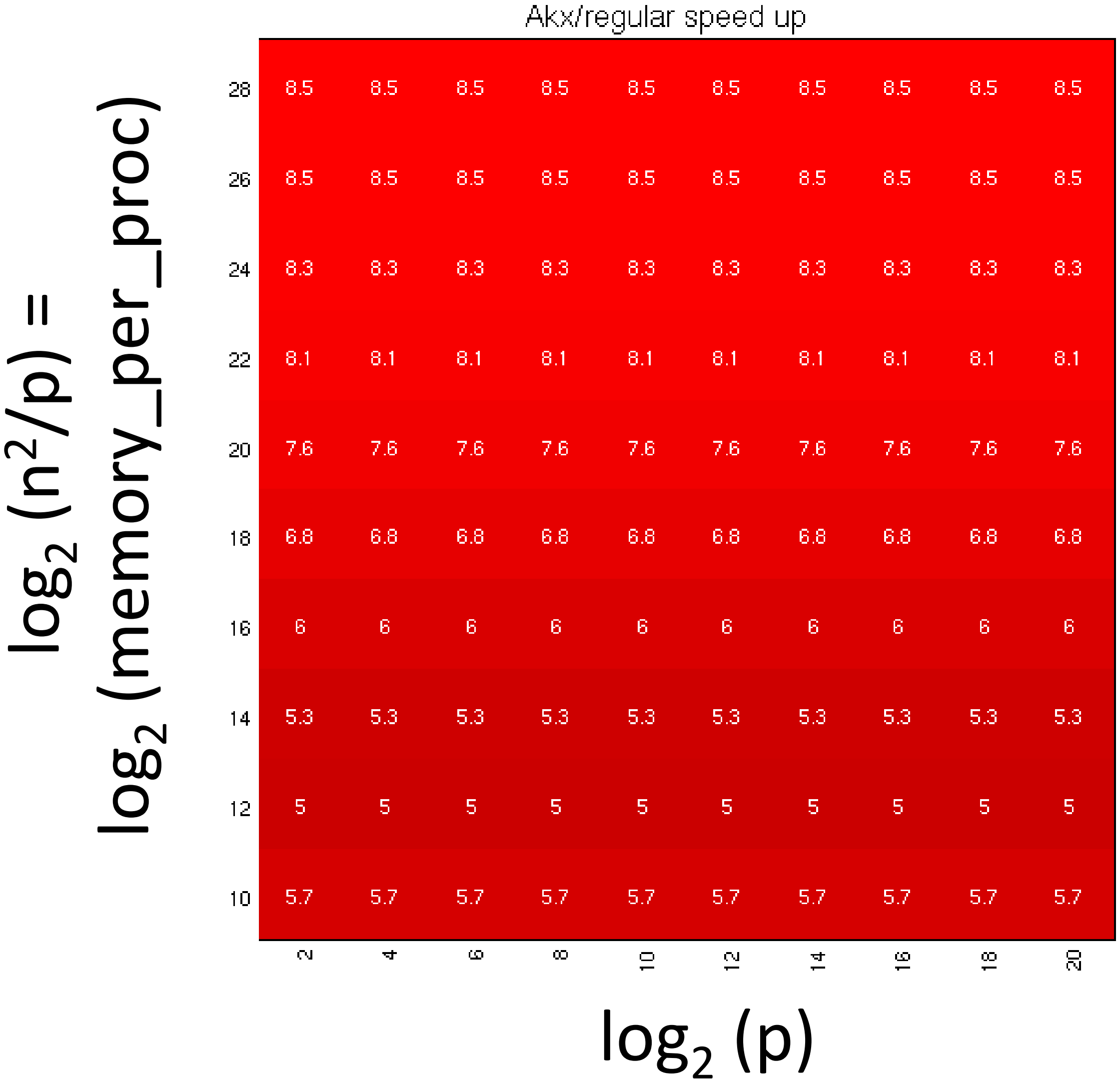
Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



Preliminary Exascale predicted speedups forMatrix Powers Kernel over SpMV for 2D Poisson (5 point stencil)



	2	4	6	8	10	12	14	16	18	20
28	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
26	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5	8.5
24	8.3	8.3	8.3	8.3	8.3	8.3	8.3	8.3	8.3	8.3
22	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1	8.1
20	7.6	7.6	7.6	7.6	7.6	7.6	7.6	7.6	7.6	7.6
18	6.8	6.8	6.8	6.8	6.8	6.8	6.8	6.8	6.8	6.8
16	6	6	6	6	6	6	6	6	6	6
14	5.3	5.3	5.3	5.3	5.3	5.3	5.3	5.3	5.3	5.3
12	5	5	5	5	5	5	5	5	5	5
10	5.7	5.7	5.7	5.7	5.7	5.7	5.7	5.7	5.7	5.7

Summary of Iterative Linear Algebra

- New Lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - GMRES, CG, BiCGStab, Arnoldi, Lanczos reorganized
  - Other Krylov methods?
  - Recognizing stable variants more easily?
  - Avoiding communication with preconditioning harder
    - "Hierarchically semi-separable" preconditioners work
  - Autotuning

For further information

- www.cs.berkeley.edu/~demmel
- Papers
  - bebop.cs.berkeley.edu
    - Mark Hoemmen's PhD thesis for Krylov methods
  - www.netlib.org/lapack/lawns
- 1-week-short course – slides and video
  - www.ba.cnr.it/ISSNLA2010

Collaborators and Supporters

- Collaborators
  - Michael Anderson (UCB), Grey Ballard (UCB), Erin Carson (UCB), Jack Dongarra (UTK), Ioana Dumitriu (U. Wash), Laura Grigori (INRIA), Ming Gu (UCB), Mark Hoemmen (Sandia NL), Olga Holtz (UCB & TU Berlin), Nick Knight (UCB), Julien Langou, (U Colo. Denver), Marghoob Mohiyuddin (UCB), Oded Schwartz (TU Berlin), Edgar Solomonik (UCB), Michelle Strout (Colo. SU), Vasily Volkov (UCB), Sam Williams (LBNL), Hua Xiang (INRIA), Kathy Yelick (UCB & LBNL)
  - Other members of the ParLab, BEBOP, CACHE, EASI, MAGMA, PLASMA, TOPS projects
- Supporters
  - NSF, DOE, UC Discovery
  - Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Sun

We’re hiring!

- Seeking software engineers to help develop the next versions of LAPACK and ScaLAPACK
- Locations: UC Berkeley and UC Denver
- Please send applications to julie@cs.utk.edu

Summary

Time to redesign all linear algebra algorithms and software

Don’t Communic…