The Past, Present and *Future* of High Performance Numerical Linear Algebra: *Minimizing Communication*

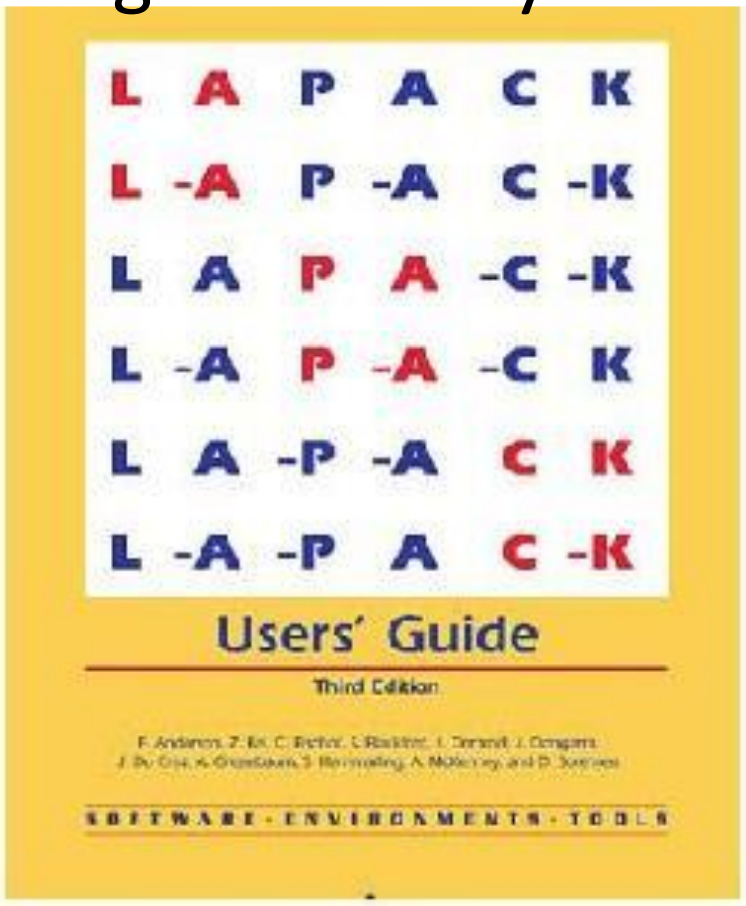
Jim Demmel

EECS & Math Departments

UC Berkeley

The Past, the Present …

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

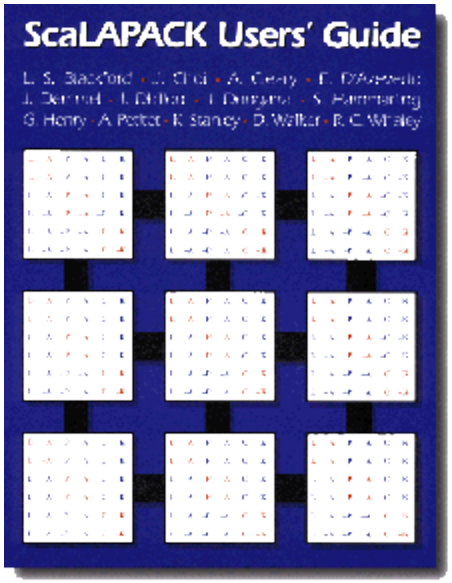


The Past, the Present …

- LAPACK contributors
  - Jack Dongarra, Julien Langou, Julie Langou, Ed Anderson, Zhaojun Bai, David Bailey, David Bindel, 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

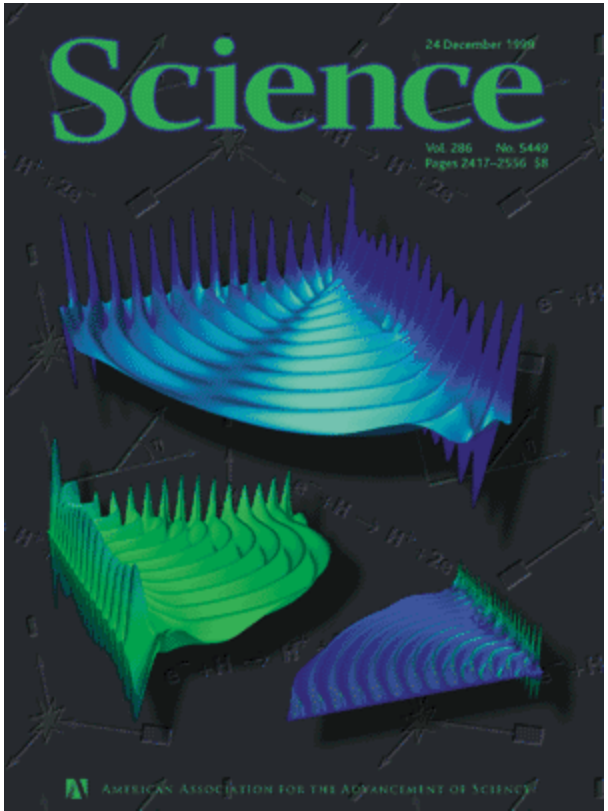
The Past, the Present …

- 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



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

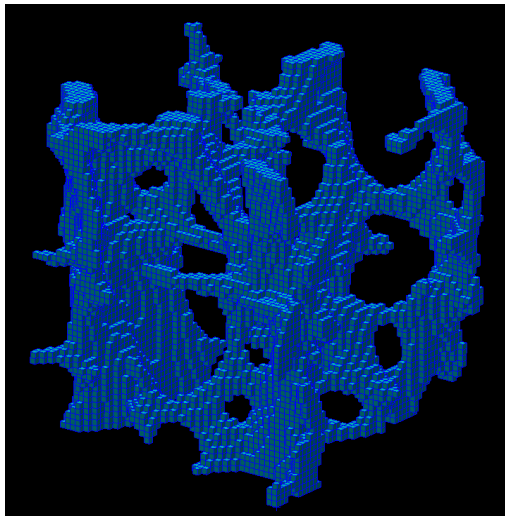


The Past, the Present …

- 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



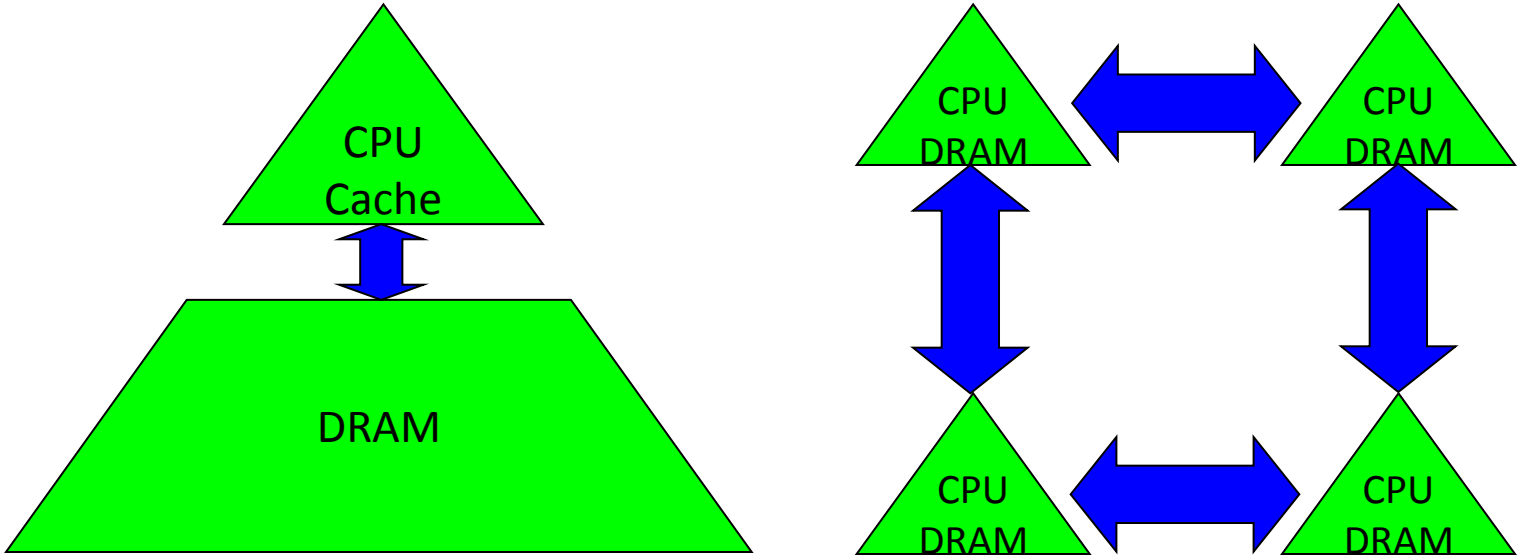
… The Future

- 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 ≤ 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)

#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
  - 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)

One-sided Factorizations (LU, QR), so far

- Classical Approach

  for i=1 to n

    update column i

    update trailing matrix

- #words_moved = $O(n^3)$

- Blocked Approach (LAPACK)

  for i=1 to n/b

    update block i of b columns

    update trailing matrix

- #words moved = $O(n^3/M^{1/3})$

- Recursive Approach

  func factor(A)

    if A has 1 column, update it

  else

    factor(left half of A)

    update right half of A

    factor(right half of A)

- #words moved = $O(n^3/M^{1/2})$

- None of these approaches minimizes #messages
- Need another idea

TSQR: QR of a Tall, Skinny matrix

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

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

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

TSQR: QR of a Tall, Skinny matrix

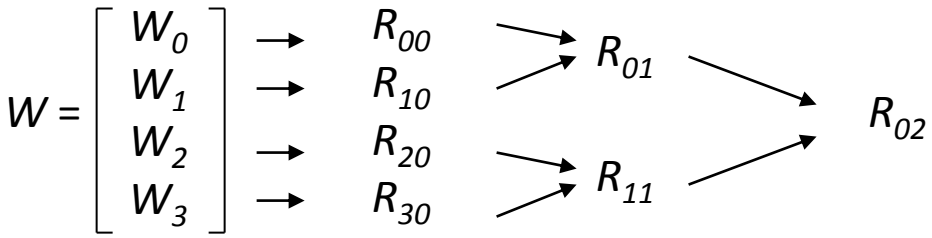
$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} Q_{00}\,R_{00} \\ Q_{10}\,R_{10} \\ Q_{20}\,R_{20} \\ Q_{30}\,R_{30} \end{pmatrix} = \begin{pmatrix} Q_{00} & & & \\ & Q_{10} & & \\ & & Q_{20} & \\ & & & Q_{30} \end{pmatrix} \cdot \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20}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

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

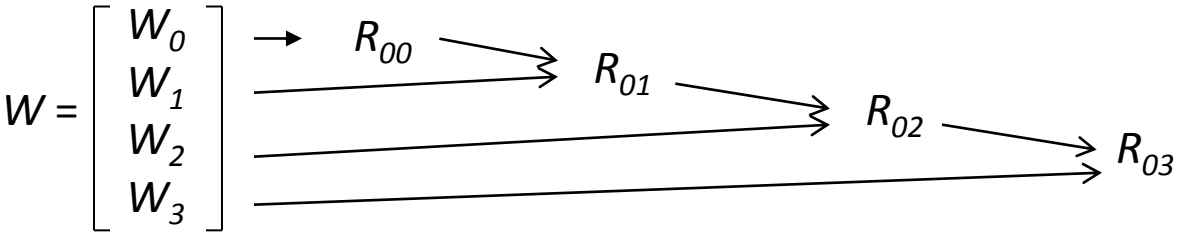
$$\begin{pmatrix} R_{01} \\ R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02}\,R_{02} \end{pmatrix}$$

Minimizing Communication in TSQR

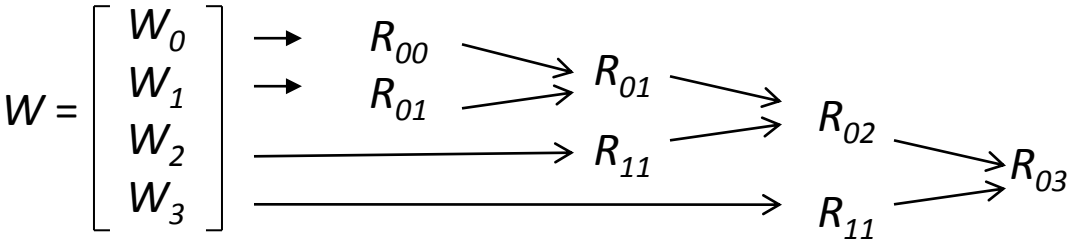
Parallel:



Sequential:



Dual Core:



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

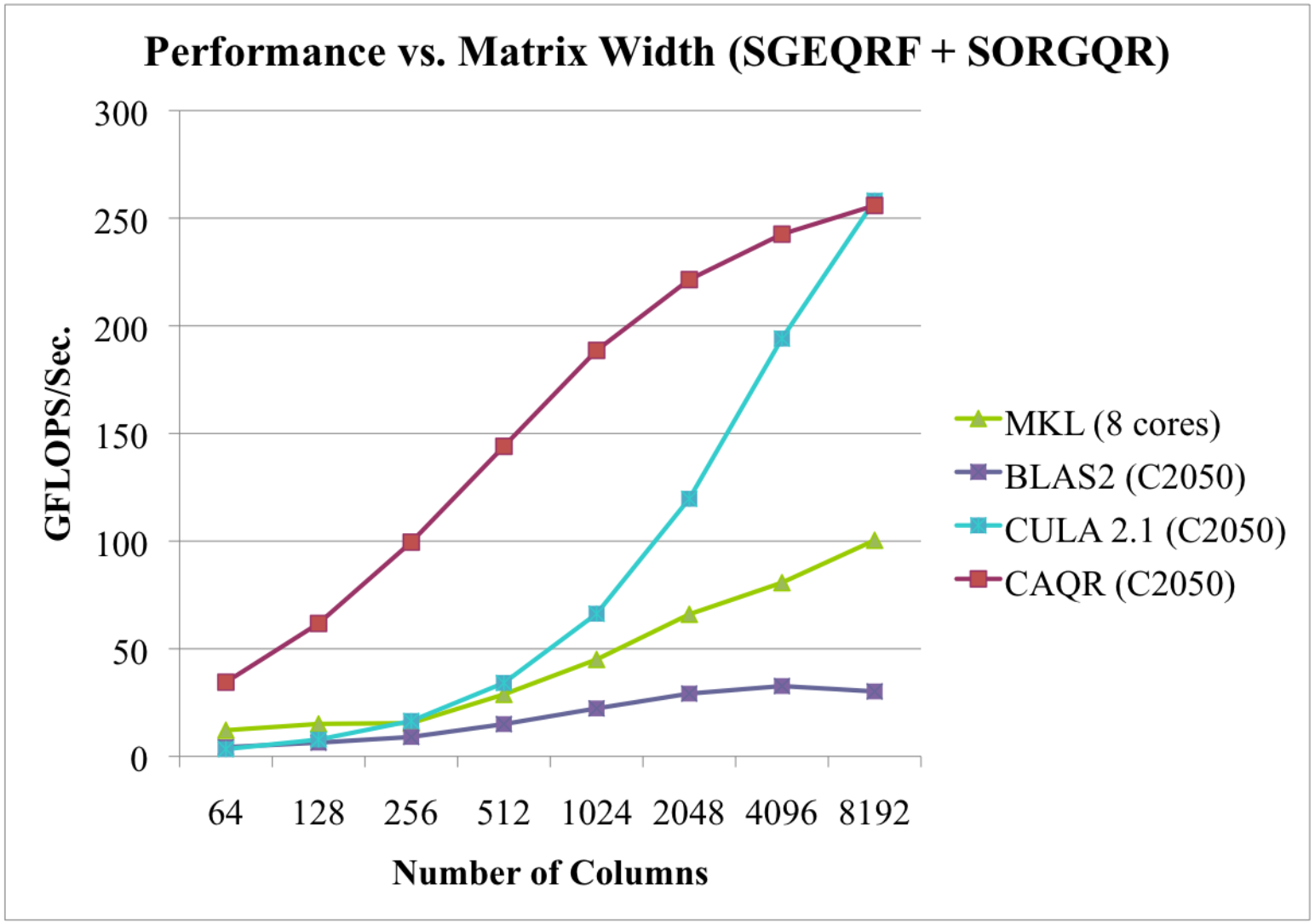
Can choose reduction tree dynamically

TSQR Performance Results

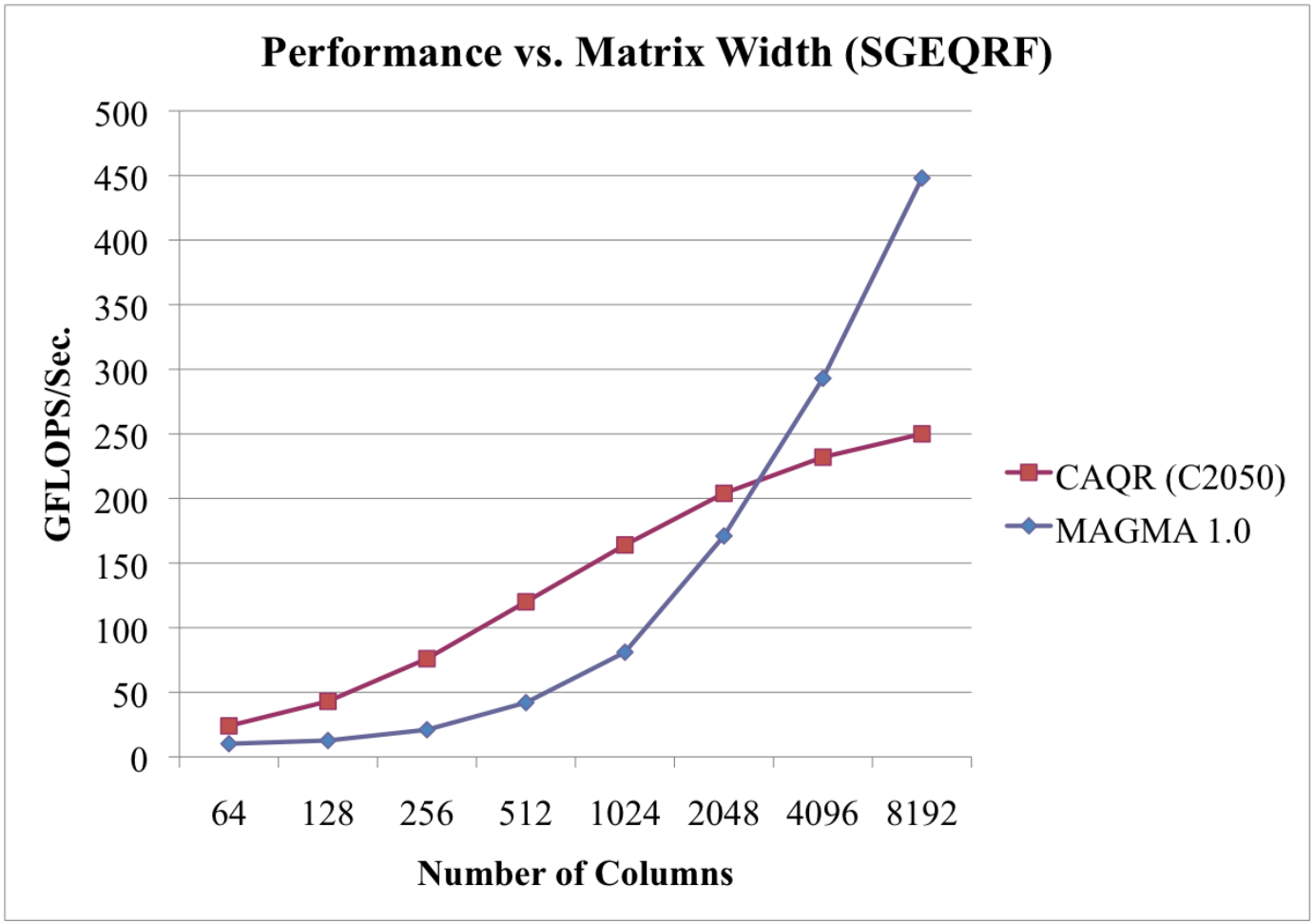
- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Grid – **4x** on 4 cities (Dongarra et 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

Partial History of dense *sequential QR* algorithms attaining communication lower bounds

- Algorithms shown minimizing # Messages assume (recursive) block layout
- *Many* references (see reports), only some shown, plus ours
  - Oldest reference may or may not include analysis
- Cache-oblivious are underlined, Green are ours, ? is unknown/future work
- b = block size, M = fast memory size, n = dimension

Algorithm	2 Levels of Memory		Multiple Levels of Memory	
	#Words Moved	and # Messages	#Words Moved	and # Messages
QR	[Elmroth, Gustavson,98] Recursive, may do more flops LAPACK (only for $n \geq M$ and $b=M^{1/2}$, or $n^2 \leq M$) [DGHL08]	[Frens,Wise,03] explicit Q only, 3x more flops [DGHL08] implicit Q, but represented differently, ~ same flop count	[Elmroth, Gustavson,98] [DGHL08]?	[Frens,Wise,03] [DGHL08]? Can we get the same flop count?

CALU: Using similar idea for TSLU as TSQR: Use reduction tree, to do "Tournament Pivoting"

$$W^{nxb} = \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{pmatrix} = \begin{pmatrix} P_1 \cdot L_1 \cdot U_1 \\ P_2 \cdot L_2 \cdot U_2 \\ P_3 \cdot L_3 \cdot U_3 \\ P_4 \cdot L_4 \cdot U_4 \end{pmatrix}$$

Choose b pivot rows of W_1, call them W_1'
Choose b pivot rows of W_2, call them W_2'
Choose b pivot rows of W_3, call them W_3'
Choose b pivot rows of W_4, call them W_4'

$$\begin{pmatrix} W_1' \\ W_2' \\ W_3' \\ W_4' \end{pmatrix} = \begin{pmatrix} P_{12} \cdot L_{12} \cdot U_{12} \\ P_{34} \cdot L_{34} \cdot U_{34} \end{pmatrix}$$

Choose b pivot rows, call them W_{12}'

Choose b pivot rows, call them W_{34}'

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

Choose b pivot rows

- Go back to W and use these b pivot rows (move them to top, do LU without pivoting)
- Repeat on each set of b columns to do CALU

Stability of CALU using Tournament Pivoting

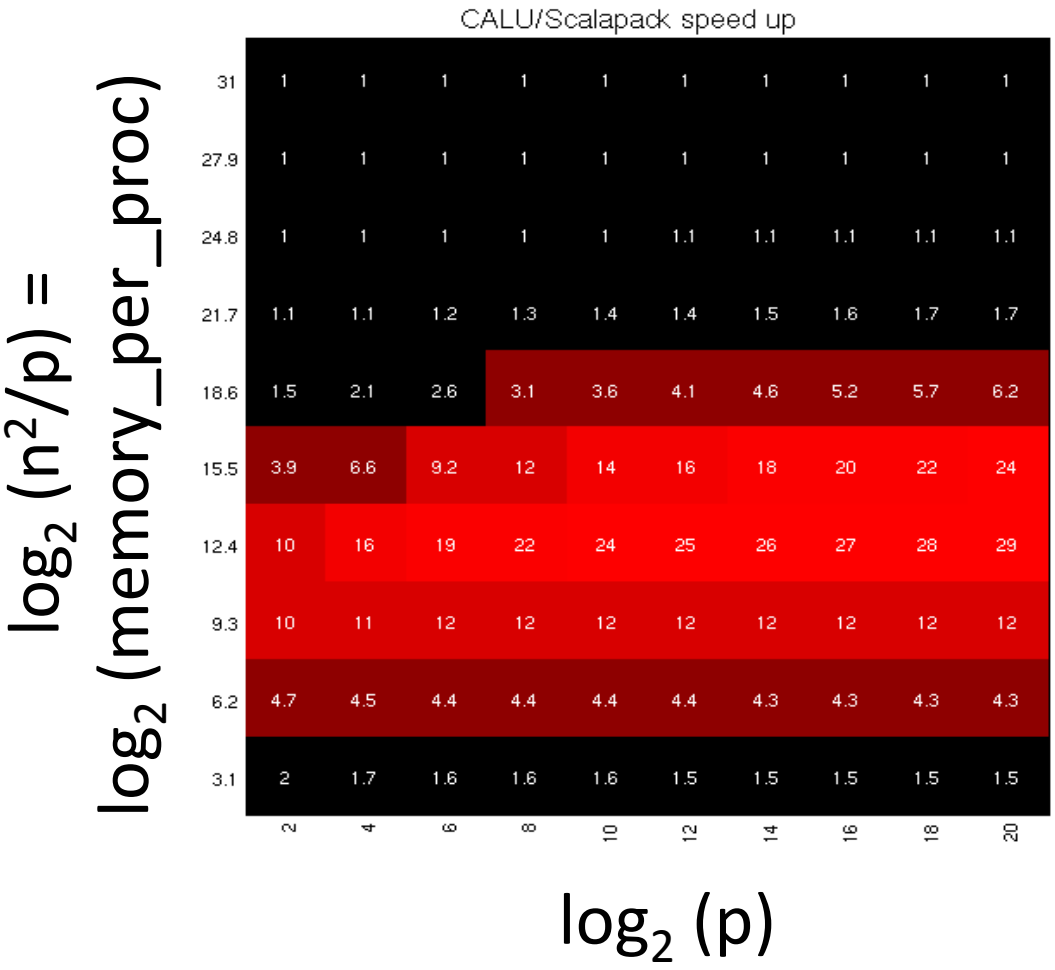
- Thm: Tournament Pivoting (TP) as stable as Partial Pivoting (PP) in following sense: TP gets same results as PP applied to different input matrix whose entries are blocks taken from input A

- Worst case pivot growth factor on n x n matrix
  - Proven: 2^{nH-1} where H = 1 + height of reduction tree
  - Attained: 2^{n-1} , same as PP
- There are examples where PP is exponentially less stable than TP, and vice-versa, so neither is always better than the other
- Extensive numerical testing confirms stability

Exascale Machine Parameters

- 2^30 ≈ 1,000,000 nodes
- 1024 cores/node (a billion cores!)
- 100 GB/sec interconnect bandwidth
- 400 GB/sec DRAM bandwidth
- 1 microsec interconnect latency
- 50 nanosec memory latency
- 32 Petabytes of memory
- 1/2 GB total L1 on a node
- 20 Megawatts !?

Exascale predicted speedups for CA-LU vs ScaLAPACK-LU



History of Spectral Divide-and-Conquer for Eigenvalue problems and the SVD

- Ideas go back to Bulgakov, Godunov, Malyshev [BG88,Mal89]
- Bai, D., Gu [BDG97]
  - Reduced to matmul, QR, generalized QR with pivoting (bug)
- D., Dumitriu, Holtz [DDH07]
  - Uses RURV (randomized URV), not pivoted QR (same bug)
- D., Grigori, Hoemmen, Langou [DGHL08]
  - Communication-avoiding QR (CAQR)
- New Communication-avoiding algorithm [BDD10]
  - Use generalized RURV for better rank-detection (fixes bug)
  - Uses communication-avoiding matmul and QR
  - Uses randomization in divide-and-conquer
  - Ballard, D., Dumitriu

Overview of Spectral Divide-and-Conquer

- Implicitly compute $(I + A^{-2^k})^{-1}$
  - Maps spectrum to 0 and 1 (as $k \to \infty$)
  - Converges to spectral projector of A for $|\lambda| > 1$
  - Rank Revealing QR yields invariant subspace

$$A_{new} = Q^*AQ = \begin{pmatrix} A_{11} & A_{12} \\ \varepsilon & A_{22} \end{pmatrix}$$

- Algorithm:

$A_0 = A$, $B_0 = I$

Repeat

$$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{12} & Q_{22} \end{pmatrix} \cdot \begin{pmatrix} R_k \\ 0 \end{pmatrix} = qr \begin{pmatrix} B_k \\ A_k \end{pmatrix}$$

$A_{k+1} = Q_{12}^* \cdot A_k$

$B_{k+1} = Q_{22}^* \cdot B_k$

until R_k converges

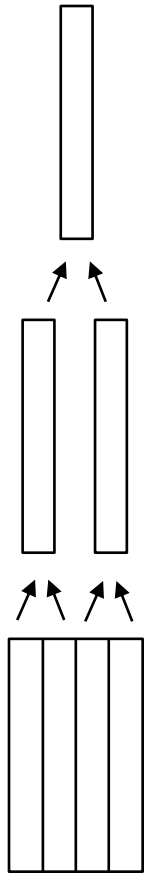
- Fact: $A_k^{-1} B_k = A^{-2^k}$
- So need RRQR of

  $(I + A^{-2^k})^{-1} = (A_k + B_k)^{-1} A_k$

  without forming $(A_k + B_k)^{-1} A_k$

Rank Revealing CA-Decompositions

- Randomized URV of A
  - [V,R'] = caqr(randn(n,n)) … V random orthogonal
    [U,R] = caqr($A \cdot V^T$)
  - Thm: A = URV reveals the rank of A with high probability
- Randomized URV of $(A_k + B_k)^{-1} A_k$
  - [V,R'] = caqr(randn(n,n)) … V random orthogonal
    [Q_1,R_1] = caqr($A_k \cdot V^T$)
    [R_2,U] = carq($Q_1^T \cdot (A_k + B_k)$)
  - Thm: $(A_k + B_k)^{-1} A_k = U^T(R_2^{-1}R_1)V$ reveals rank with high probability
  - Only need U, not $R_2^{-1}R_1$
  - Applies to $M_1^{\pm 1} \cdot M_2^{\pm 1} \cdot M_3^{\pm 1} \cdots$ = URV
- QR with Tournament Pivoting
  - Emulate column pivoting by using a ``tournament" to pick next group of b columns
  - Only applies to single matrix, not $(A_k + B_k)^{-1} A_k$

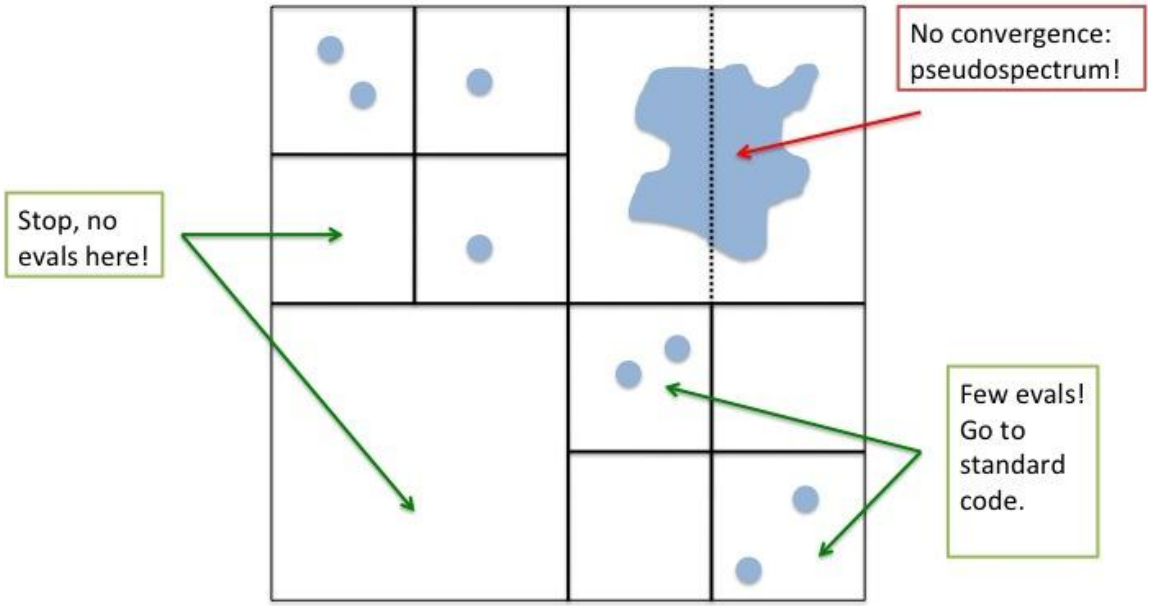


Splitting the spectrum in different places

- Splitting the spectrum along unit circle not enough
- Nonsymmetric case
  - Moebius transformation $z \rightarrow (\alpha z + \beta)/(\gamma z + \delta)$ maps unit circle to any line or circle in complex plane
  - Apply splitting algorithm to $(A_0,B_0) = (\alpha A + \beta I, \gamma A + \delta I)$ instead of (A,I), to split along any desired line or circle
- Symmetric case - easier
  - Just shift A to $A - \sigma I$
  - SVD analogous

Randomized Spectral Divide-and-Conquer

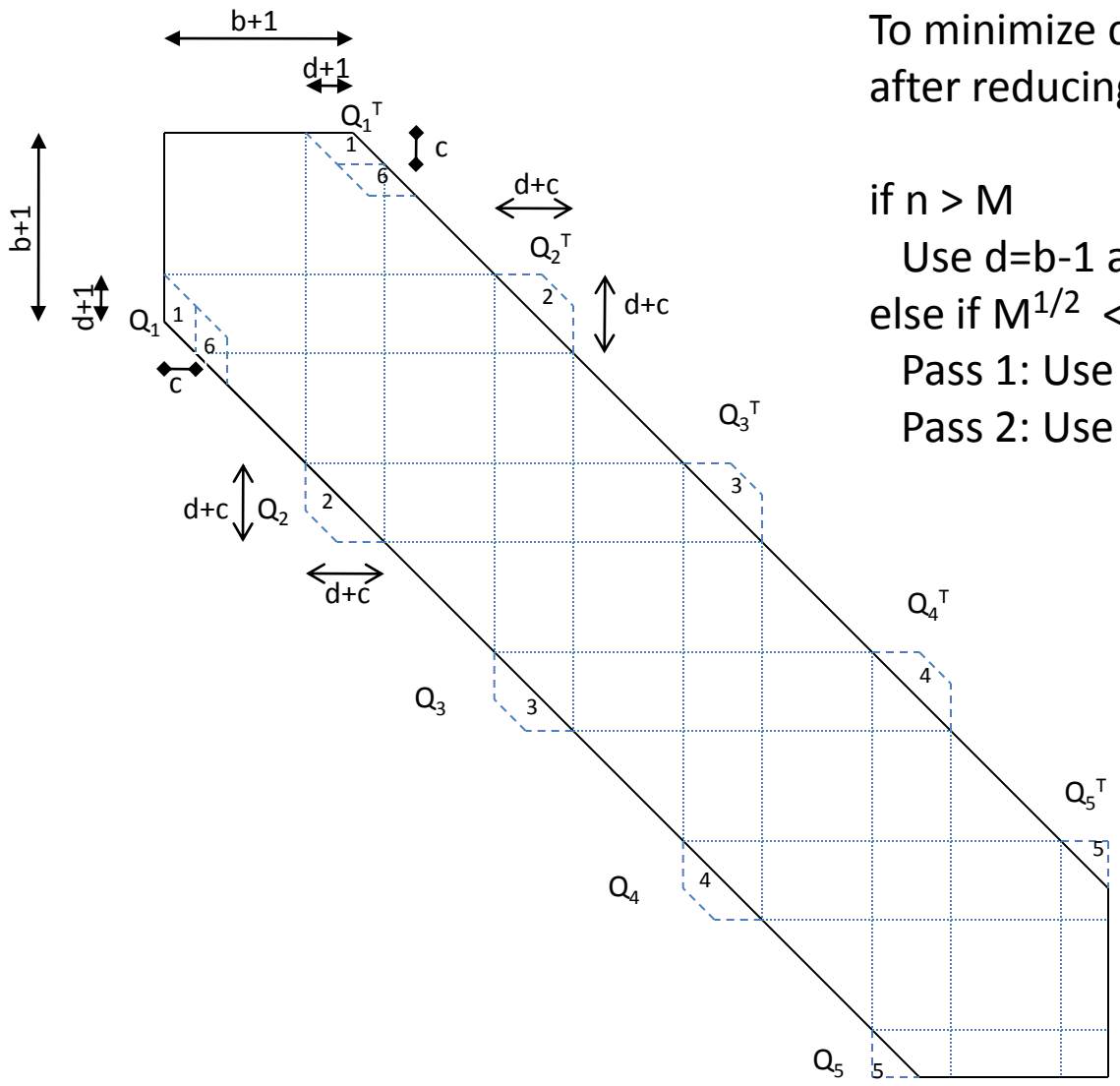
- Pick random place to divide spectrum
- Two progress metrics
  - Spectrum splits, with f·n and (1-f) · n evals per part, with $0 < f < 1$
  - Region splits, with fractions f and 1-f of area in each
  - Progress guaranteed with high probability; always stable
- SymEig and SVD: random point near center of spectrum
- NonsymEig: random line/circle near "center" of spectrum



Successive Band Reduction

- Works just for eig($A=A^T$) or SVD
  - Bischof, Lang Sun
- Step 1: Reduce Dense $\rightarrow$ Narrow Band
  - If bandwidth $\approx M^{1/2}$, can minimize communication
- Step 2: Reduce Narrow Band $\rightarrow$ Tridiagonal
  - Trickier…

Successive Band Reduction



To minimize communication (when $n > M^{1/2}$) after reducing to bandwidth $b = M^{1/2}$:

if $n > M$

Use $d=b-1$ and $c=1$

else if $M^{1/2} < n \leq M$

Pass 1: Use $d=b - M/(4n)$ and $c=M/(4n)$

Pass 2: Use $d=M/(4n)-1$ and $c=1$

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:

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

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	[GDX08] ScaLAPACK	log P log P	log P $(N / P^{1/2}) \cdot \log P$
QR	[DGHL08] ScaLAPACK	log P log P	$\log^3 P$ $(N / P^{1/2}) \cdot \log P$
Sym Eig, SVD	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $N / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $N \cdot \log P$

Can we do better?

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, **memory per processor = $O(n^2 / P)$? Why?**
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:

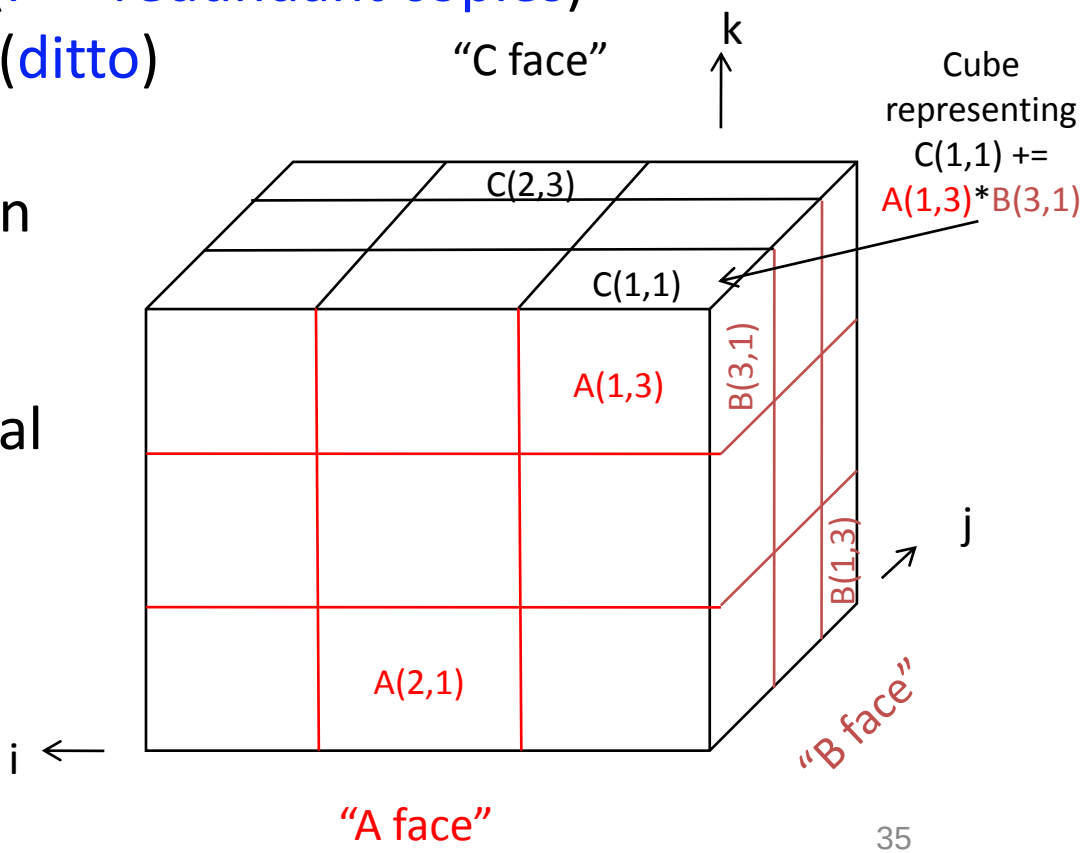
  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	[GDX08] ScaLAPACK	log P log P	log P $(N / P^{1/2}) \cdot \log P$
QR	[DGHL08] ScaLAPACK	log P log P	$\log^3 P$ $(N / P^{1/2}) \cdot \log P$
Sym Eig, SVD	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $N / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $N \cdot \log P$

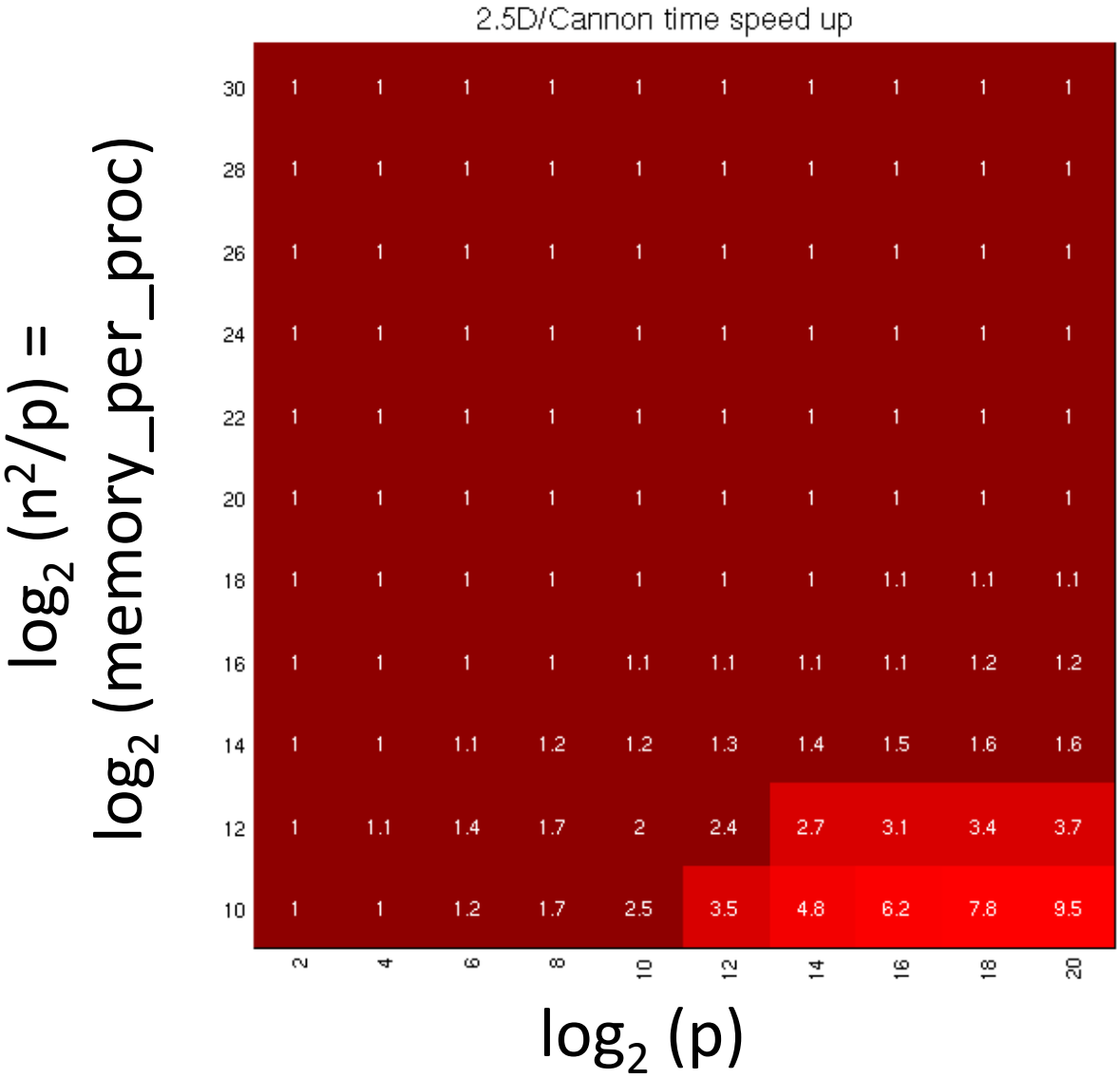
Can we do better?

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

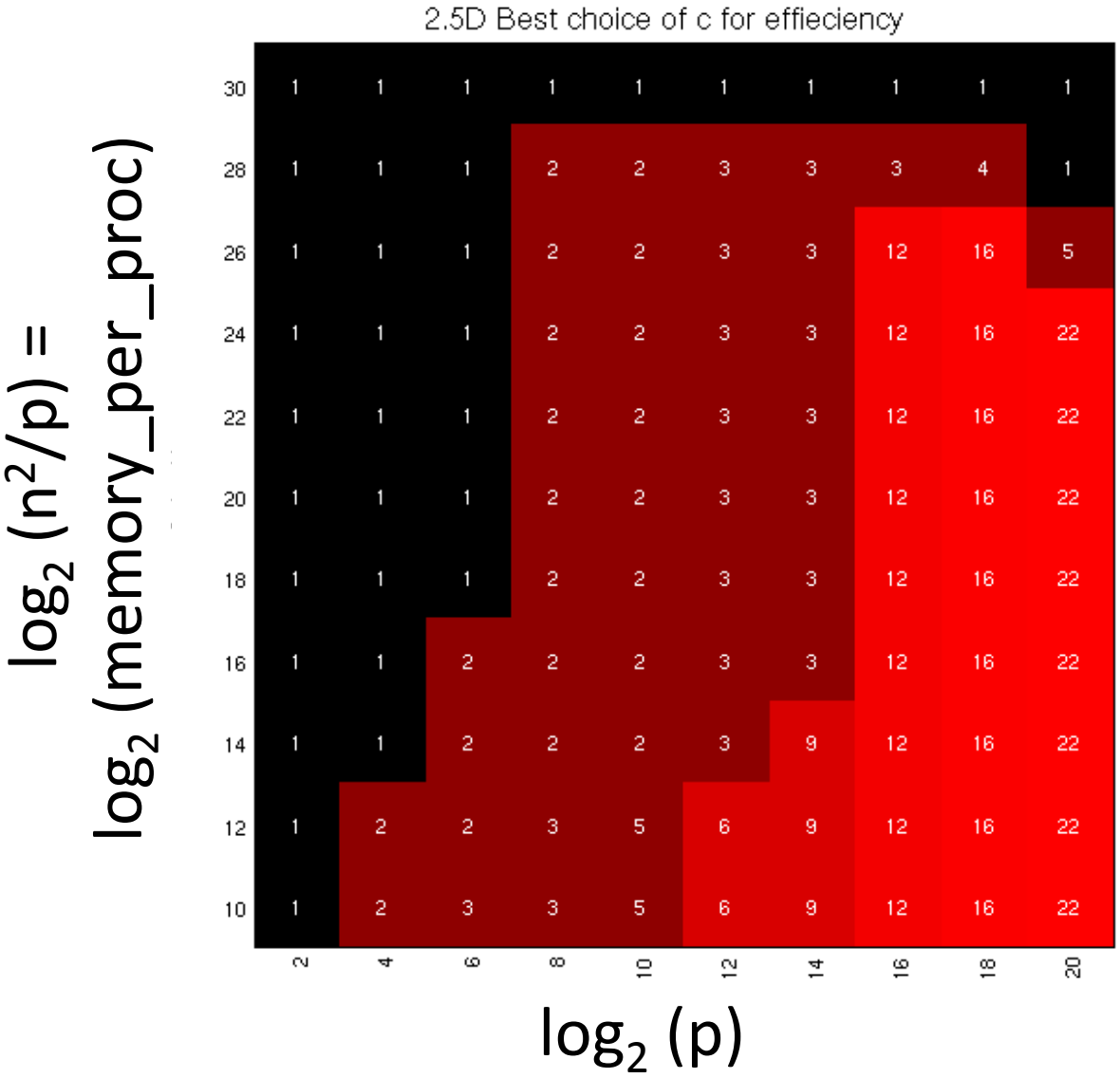
- #words_moved = $\Omega((n^3/P)/local\_mem^{1/2})$
- If one copy of data, local_mem = n^2/P
- Can we use more memory to communicate less?
- Johnson's Matmul Algorithm on $P^{1/3}$ x $P^{1/3}$ x $P^{1/3}$ processor grid
  - Broadcast A in j direction ($P^{1/3}$ redundant copies)
  - Broadcast B in i direction (ditto)
  - Local multiplies
  - Reduce (sum) in k direction
- Communication volume
  = $O((n/P^{1/3})^2 \log(P))$ - optimal
- Number of messages
  = $O(\log(P))$ – optimal
- Extends to $1 \le c \le P^{1/3}$ copies
- What about LU, etc?



Predicted Exascale
2.5D Matmul - Speedup over Cannon



Exascale 2.5D Matmul – optimal # copies



Strassen-like Algorithms

- M = fast memory size
- Conventional matmul
  - #flops = $2n^3$
  - #words_moved = $\Omega(n^3/M^{1/2}) = \Omega(M(n/M^{1/2})^3)$, attainable
- Strassen matmul
  - #flops = $\Theta(n^\omega)$ where $\omega = \log_2(7) \approx 2.81$
  - #words_moved = $\Omega(M(n/M^{1/2})^\omega)$, attainable too
- Applies to other sufficiently "Strassen-like" algorithms
  - How broadly does it apply?
- We know rest of linear algebra can be done in $O(n^\omega)$ flops
  - Also with #words_moved = $\Omega(M(n/M^{1/2})^\omega)$ (sequential)
  - Lower bounds?

Summary of Direct Linear Algebra

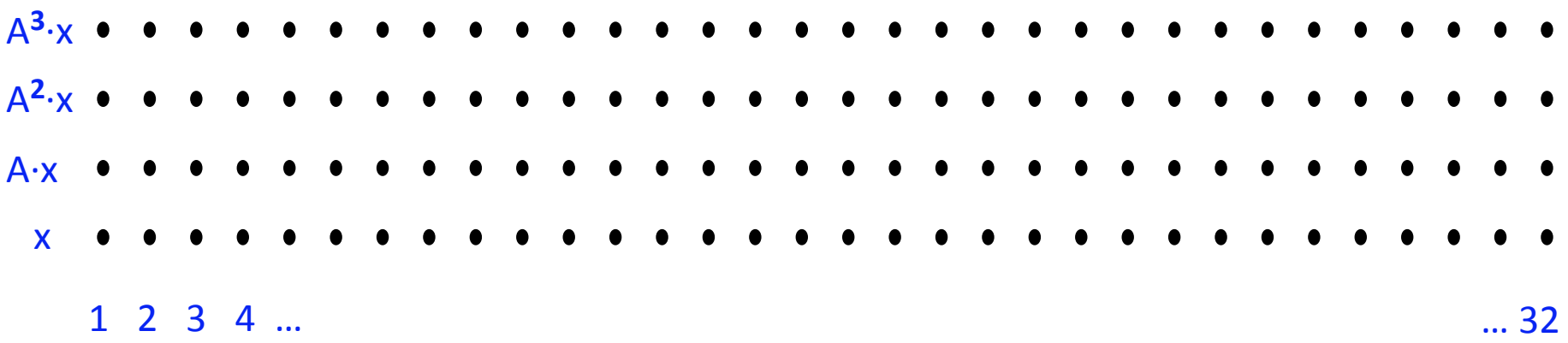
- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - New ways to “pivot”
  - Heterogeneous architectures
  - Some sparse algorithms
  - Autotuning. . .

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 (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

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

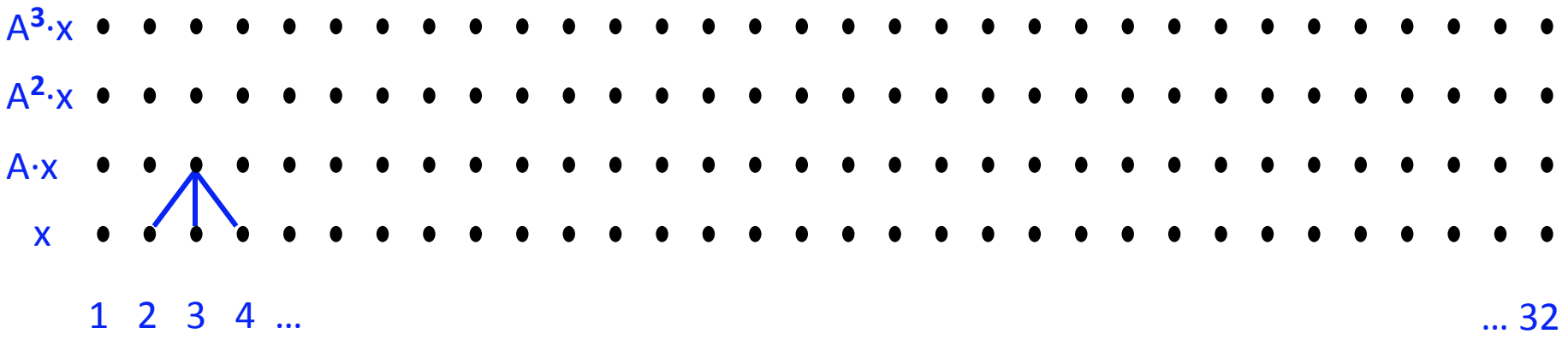
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

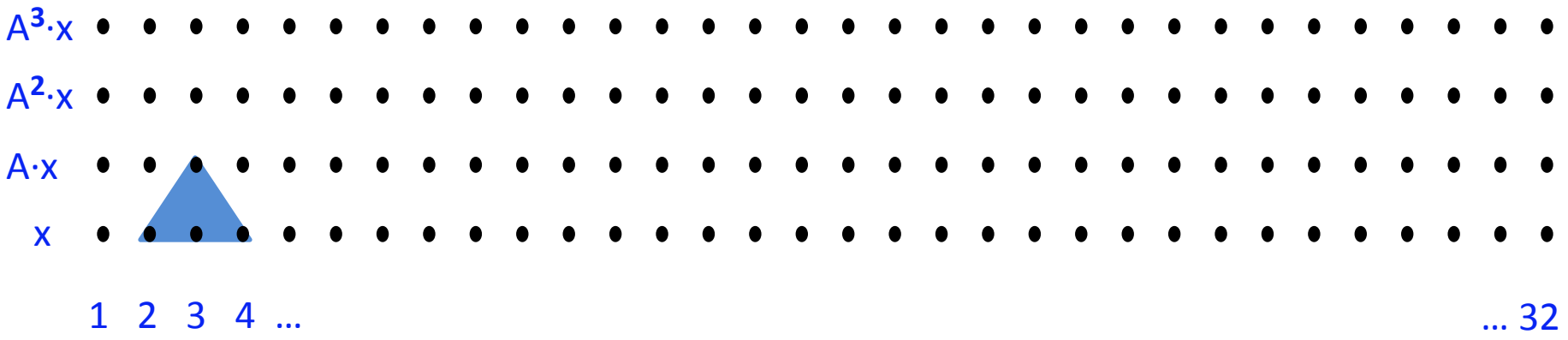
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

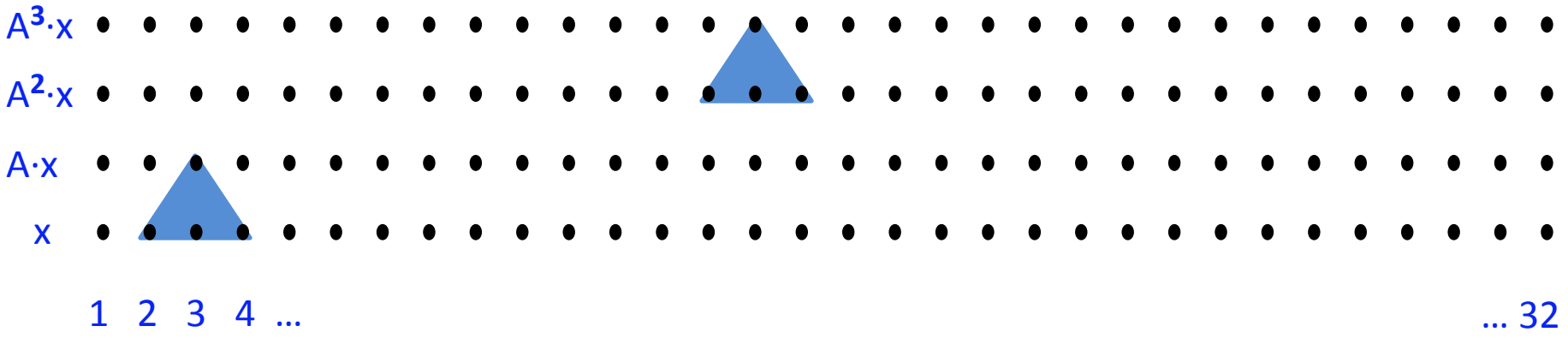
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

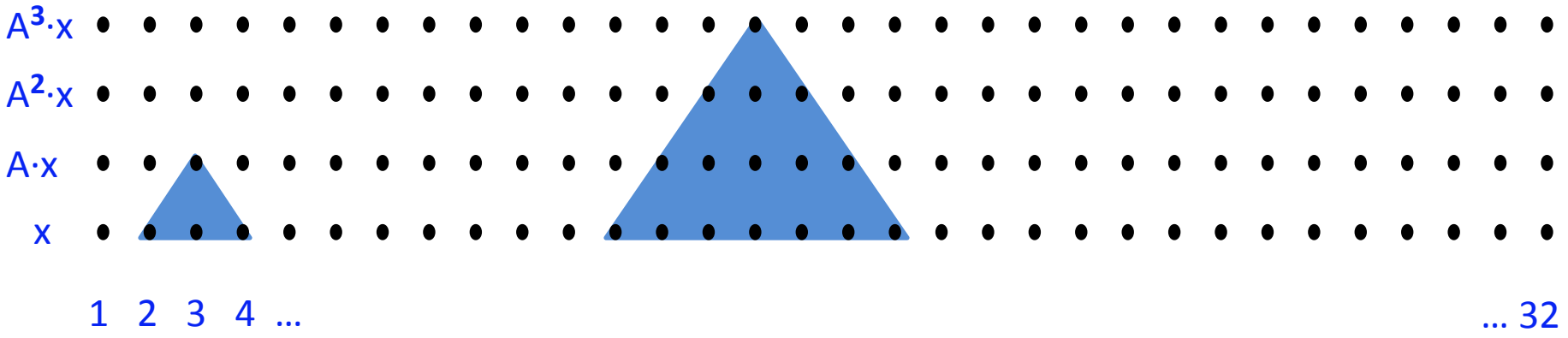
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

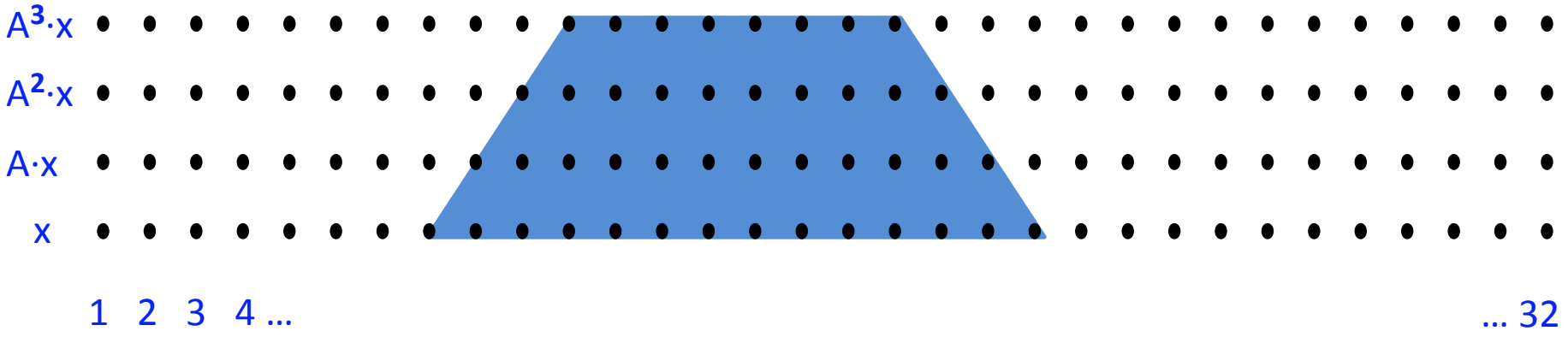
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

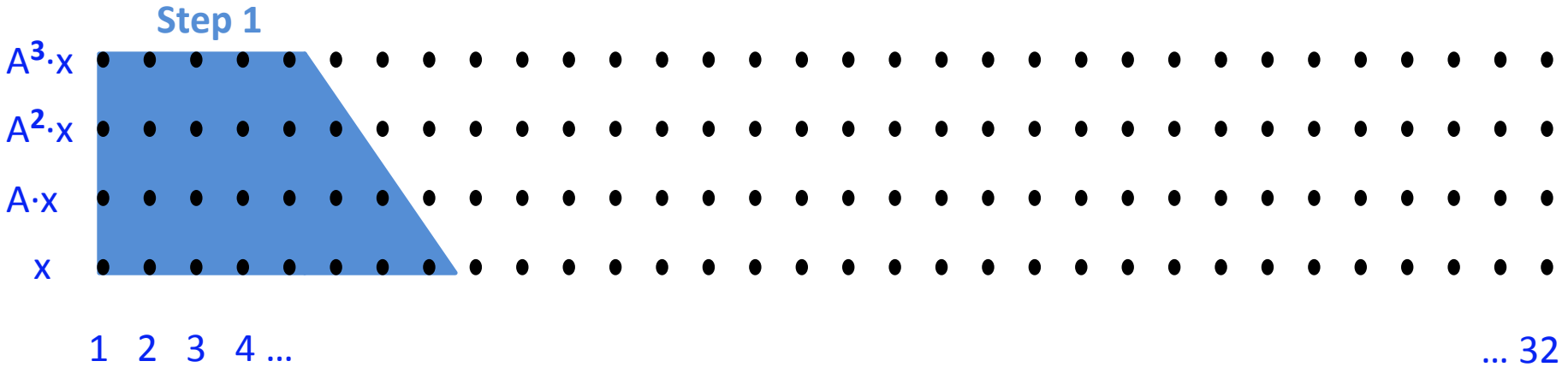
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



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

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

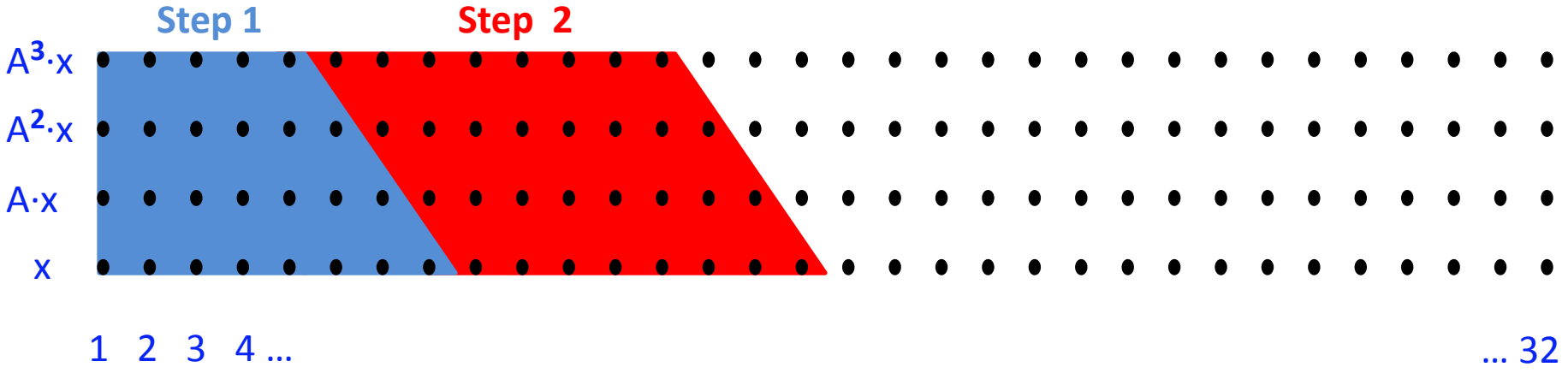
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



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

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

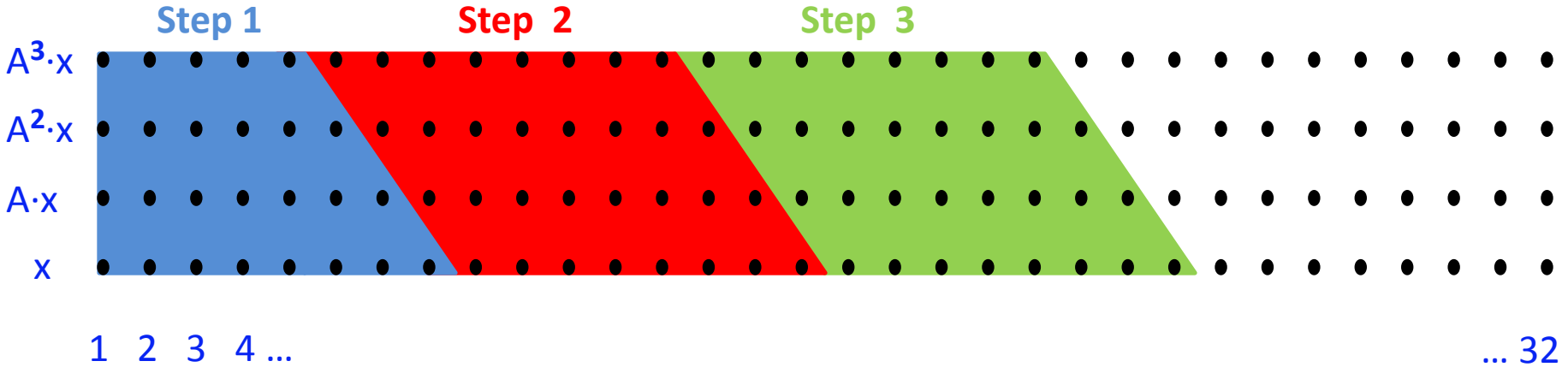
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



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

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

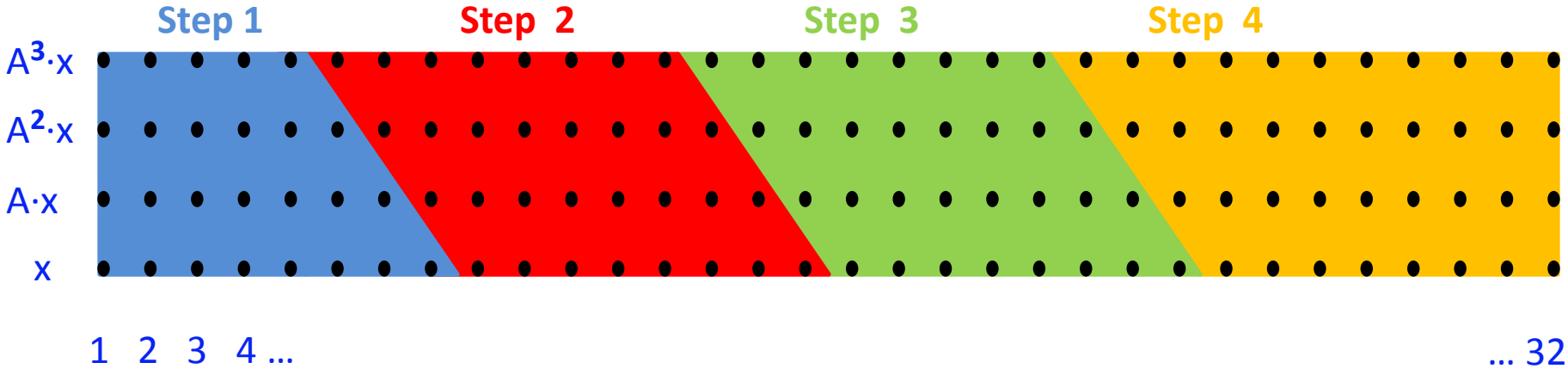
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



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

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

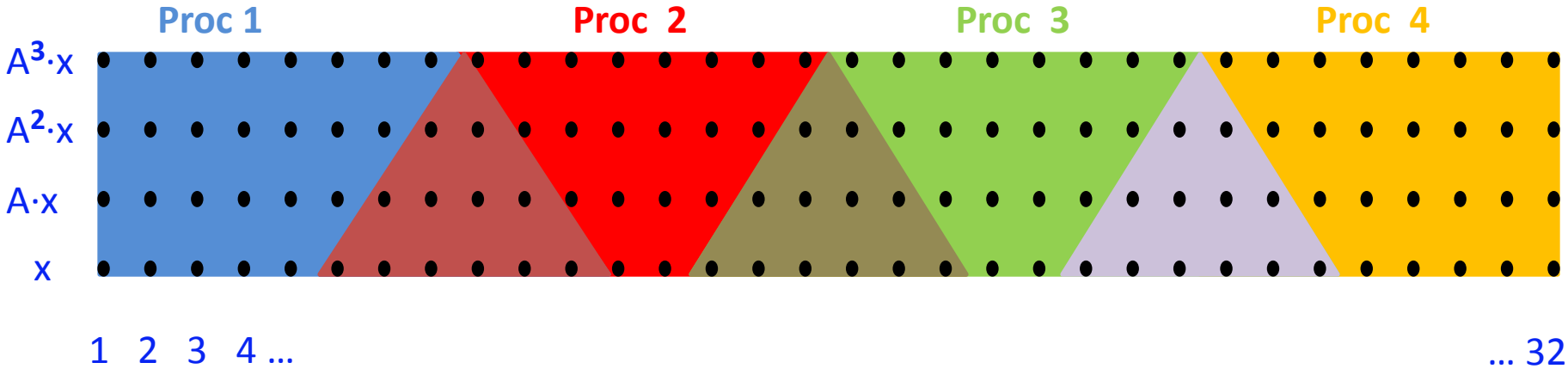
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



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

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

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

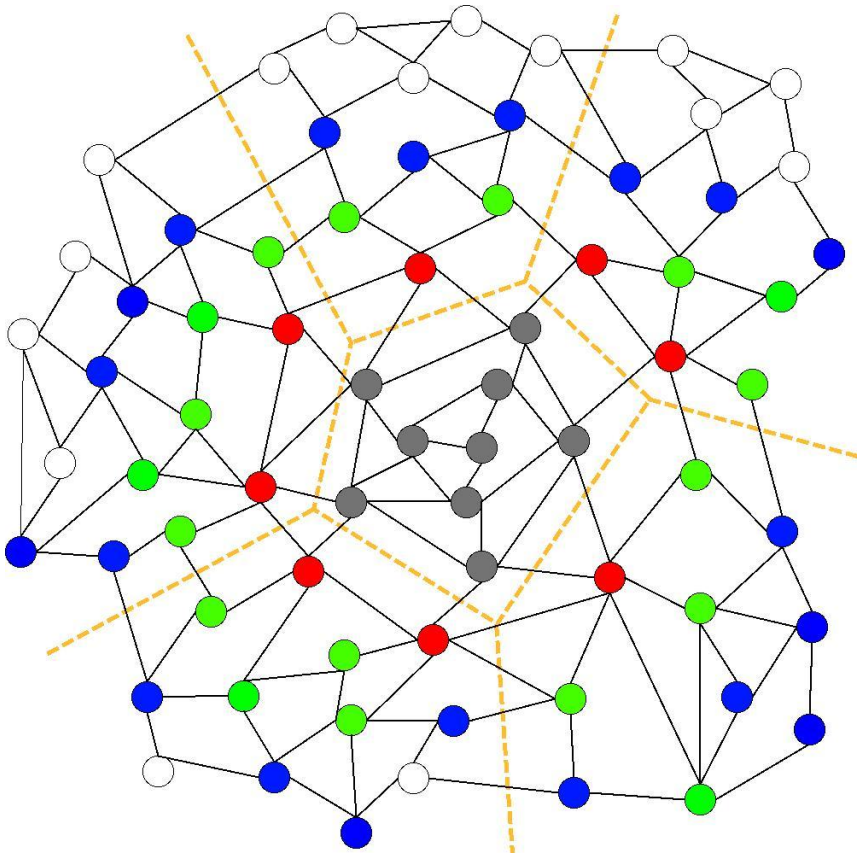


- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

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
```

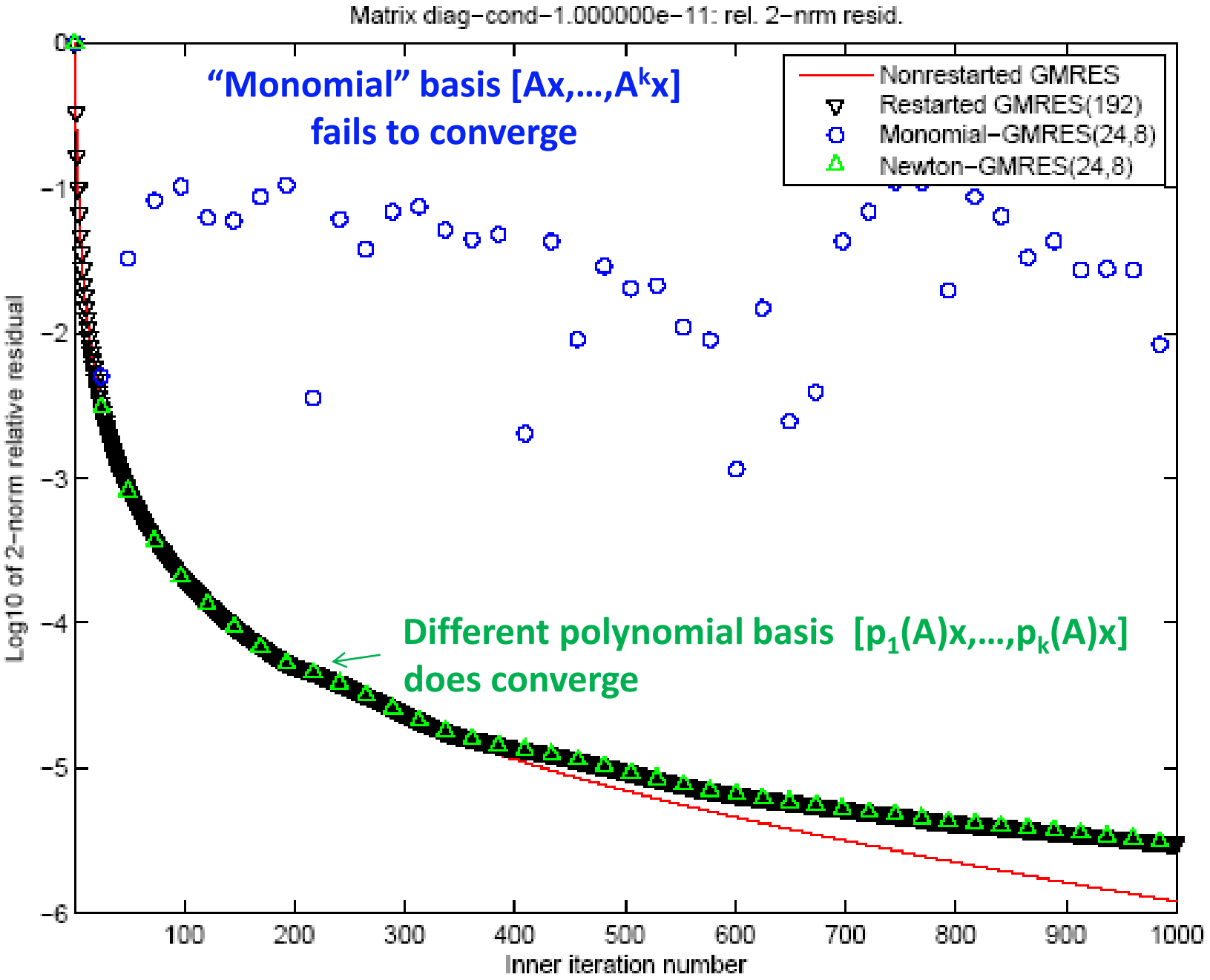
Communication-avoiding GMRES

```
W = [ v, Av, A²v, … , A^k v ]
[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

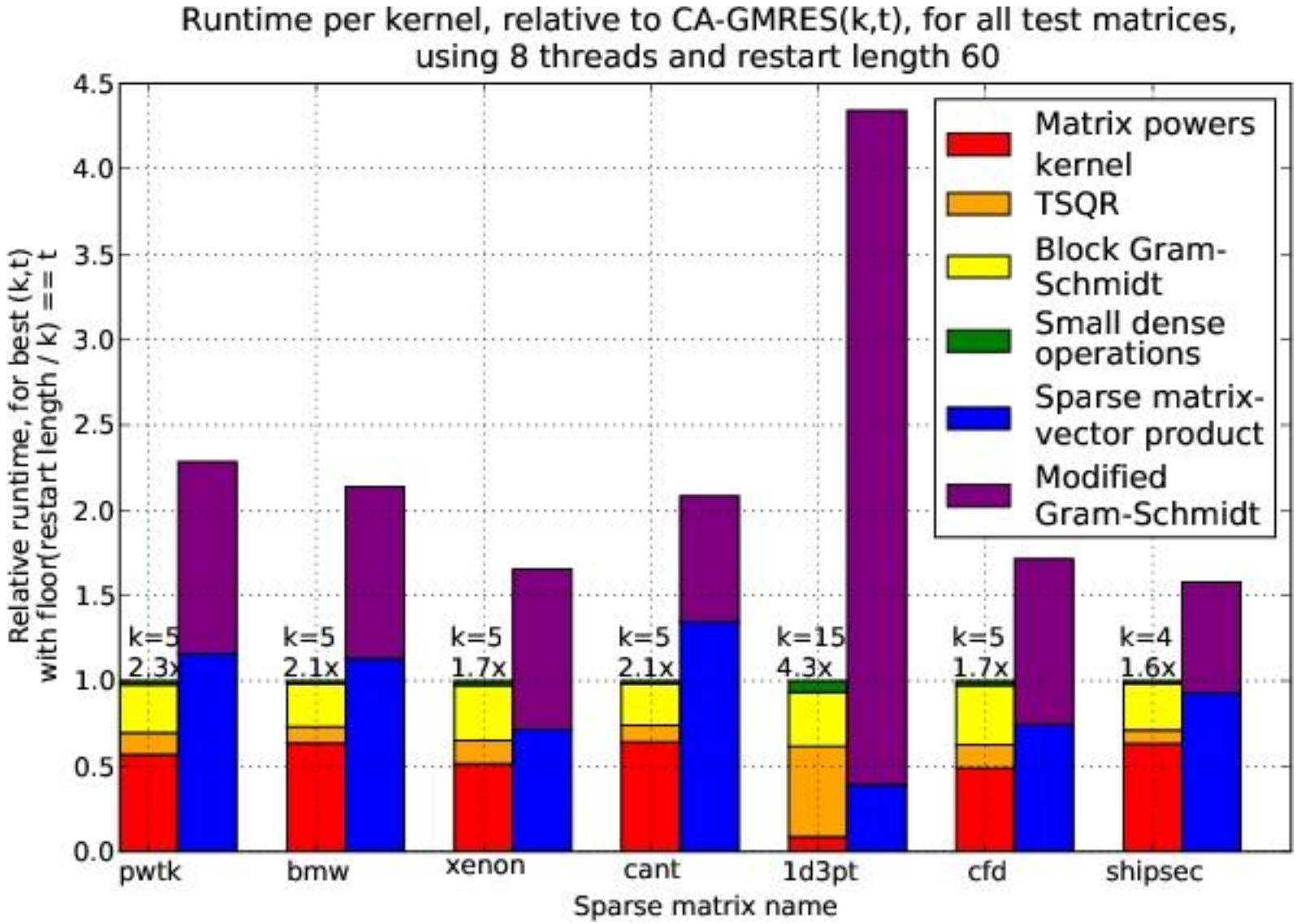
- Oops – W from power method, precision lost!


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

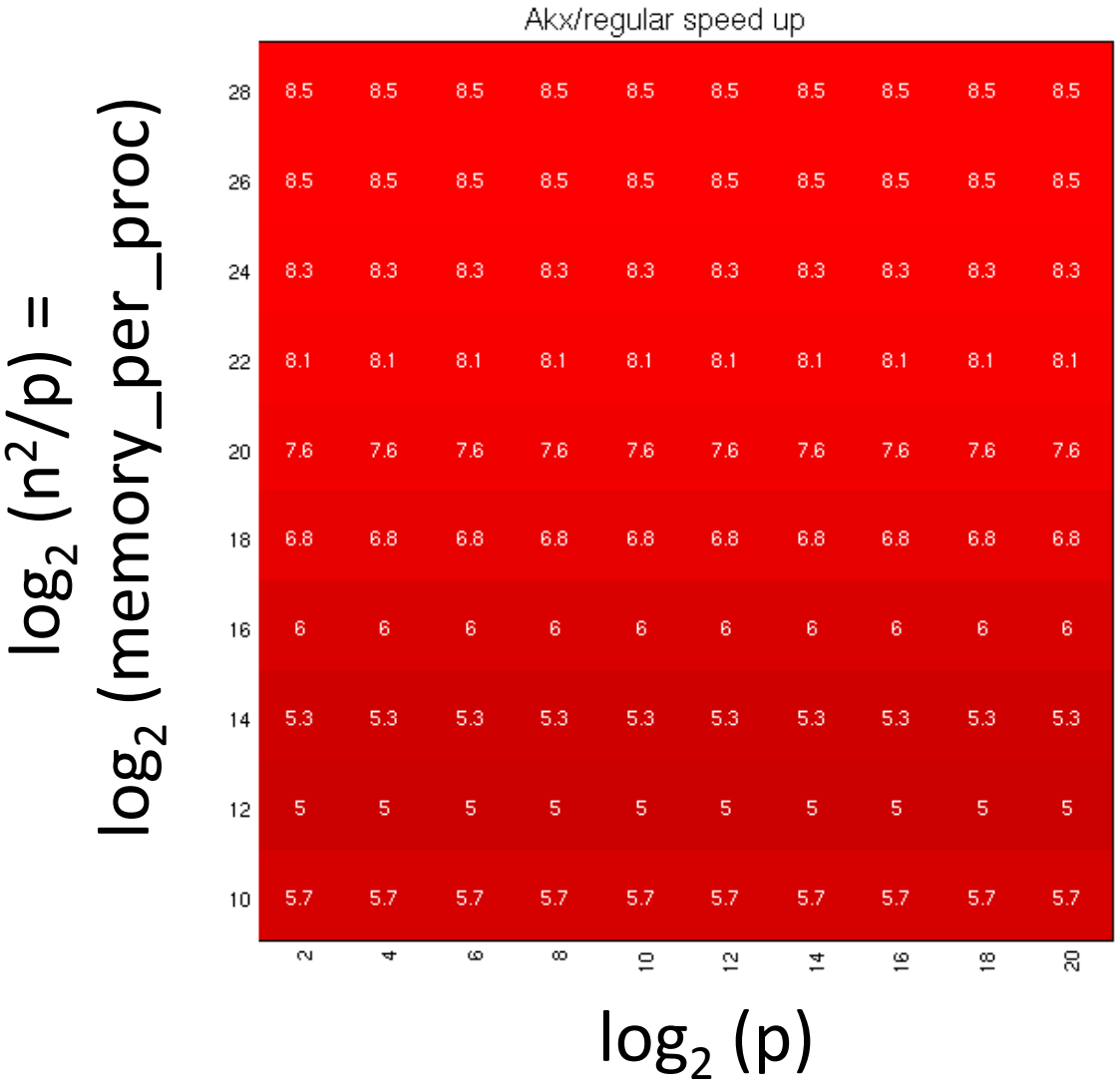
Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



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



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
  - 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…