Minimizing Communication in Linear Algebra

James Demmel

15 June 2010

www.cs.berkeley.edu/~demmel

Outline

- What is “communication” and why is it important to avoid?
- “Direct” Linear Algebra
  - Lower bounds on how much data must be moved to solve linear algebra problems like Ax=b, Ax = λx, etc
  - Algorithms that attain these lower bounds
    - *Not* in standard libraries like Sca/LAPACK (yet!)
    - Large speed-ups possible
- “Iterative” Linear Algebra (Krylov Subspace Methods)
  - Ditto
- Extensions, open problems

Collaborators

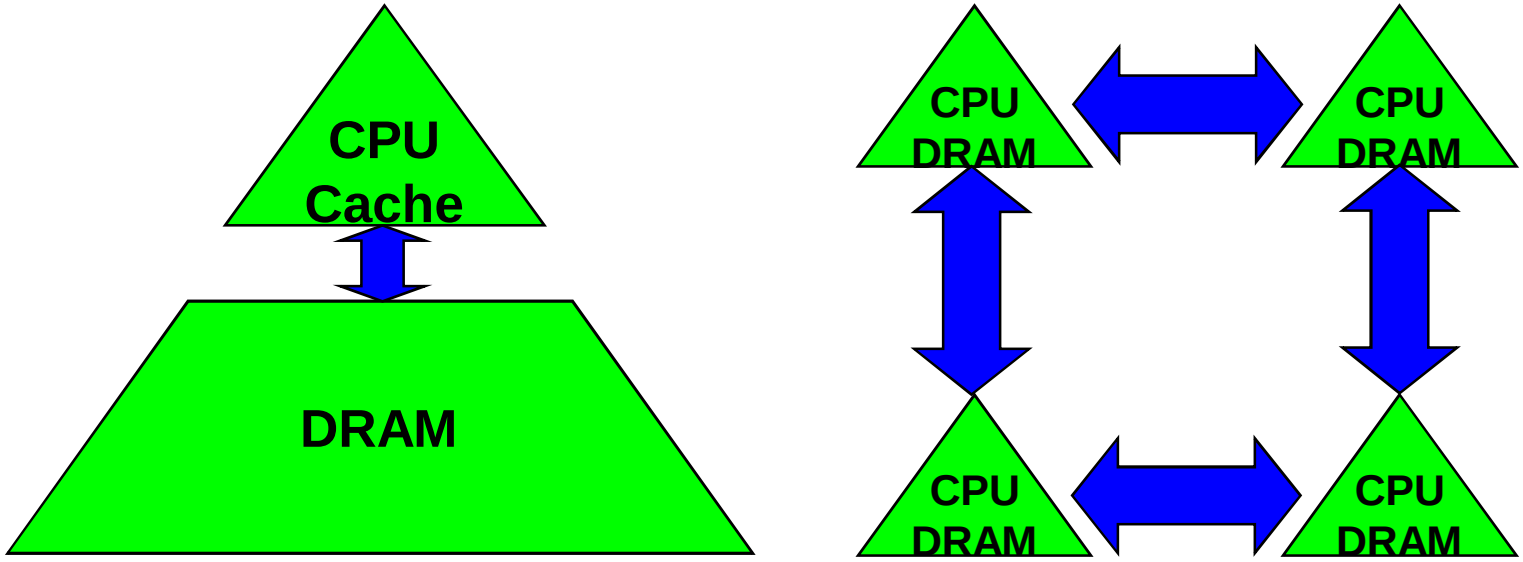
- Grey Ballard, UCB EECS
- Jack Dongarra, UTK
- Ioana Dumitriu, U. Washington
- Laura Grigori, INRIA
- Ming Gu, UCB Math
- Mark Hoemmen, Sandia NL
- Olga Holtz, UCB Math & TU Berlin
- Julien Langou, U. Colorado Denver
- Marghoob Mohiyuddin, UCB EECS
- Oded Schwartz , TU Berlin
- Hua Xiang, INRIA
- Kathy Yelick, UCB EECS & NERSC
- BeBOP group at Berkeley

Thanks to Intel, Microsoft, UC Discovery, NSF, DOE, …

Motivation (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).



Motivation (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

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 $\longleftrightarrow$ L2 $\longleftrightarrow$ DRAM $\longleftrightarrow$ network, etc
  - Not just *hiding* communication (speedup $\leq$ 2x)
  - Arbitrary speedups possible

Direct linear algebra: Prior Work on Matmul

- Assume n^3 algorithm (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" algorithms

- Parallel case on P processors:
  - Let NNZ be total memory needed; assume load balanced
  - Lower bound on #words communicated
    $= \Omega\ (n^3 / (P \cdot NNZ)^{1/2})$ [Irony, Tiskin, Toledo, 04]

NNZ	Lower bound on #words	Attained by
$3n^2$ ("2D alg")	$\Omega\ (n^2 / P^{1/2})$	[Cannon, 69]
$3n^2 P^{1/3}$ ("3D alg")	$\Omega\ (n^2 / P^{2/3})$	[Johnson,93]

Lower bound for all "direct" linear algebra

Let M = "fast" memory size per processor

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

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

- Holds for
  - BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg computing A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)
  - See [BDHS09]

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

- Goals for algorithms:
  - Minimize #words_moved = Ω (#flops/ $M^{1/2}$)
  - Minimize #messages = Ω (#flops/ $M^{3/2}$)
    - Need new data structures: (recursive) blocked
  - Minimize for multiple memory hierarchy levels
    - Cache-oblivious algorithms would be simplest
  - Fewest flops when matrix fits in fastest memory
    - Cache-oblivious algorithms don't always attain this

- Only a few sparse algorithms so far (eg Cholesky)

Summary of dense *sequential* algorithms attaining communication lower bounds

- Algorithms shown minimizing # Messages use (recursive) block layout
  - Not possible with columnwise or rowwise layouts
- *Many* references (see reports), only some shown, plus ours
- Cache-oblivious are underlined, Green are ours, ? is unknown/future work

Algorithm	2 Levels of Memory		Multiple Levels of Memory	
	#Words Moved	and # Messages	#Words Moved	and #Messages
BLAS-3	Usual blocked or recursive algorithms		Usual blocked algorithms (nested), or recursive [Gustavson,97]	
Cholesky	LAPACK (with b = $M^{1/2}$) [Gustavson 97] [BDHS09]	[Gustavson,97] [Ahmed,Pingali,00] [BDHS09]	(←same)	(←same)
LU with pivoting	LAPACK (rarely) [Toledo,97] , [GDX 08]	[GDX 08] *not* partial pivoting	[Toledo, 97] [GDX 08]?	[GDX 08]?
QR Rank-revealing	LAPACK (rarely) [Elmroth,Gustavson,98] [DGHL08]	[Frens,Wise,03] but 3x flops [DGHL08]	[Elmroth, Gustavson,98] [DGHL08] ?	[Frens,Wise,03] [DGHL08] ?
Eig, SVD	*Not* LAPACK [BDD10] randomized, but more flops		[BDD10]	[BDD10]

Summary of dense *2D 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 $P^{1/2} \cdot \log P$	$\log^3 P$ $N \cdot \log P$

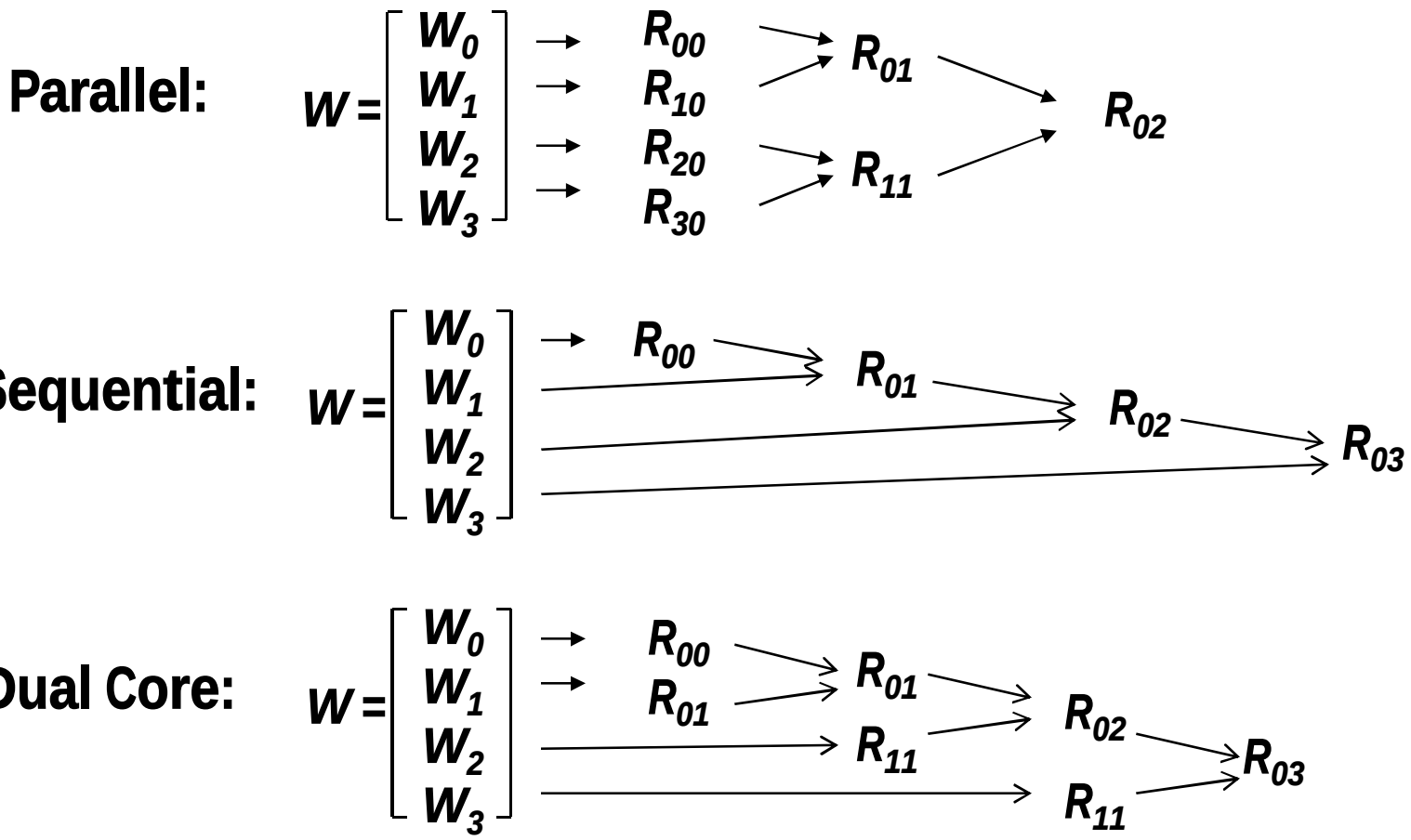
QR of a Tall, Skinny matrix is bottleneck; Use TSQR instead:

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

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

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

Minimizing Communication in TSQR



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

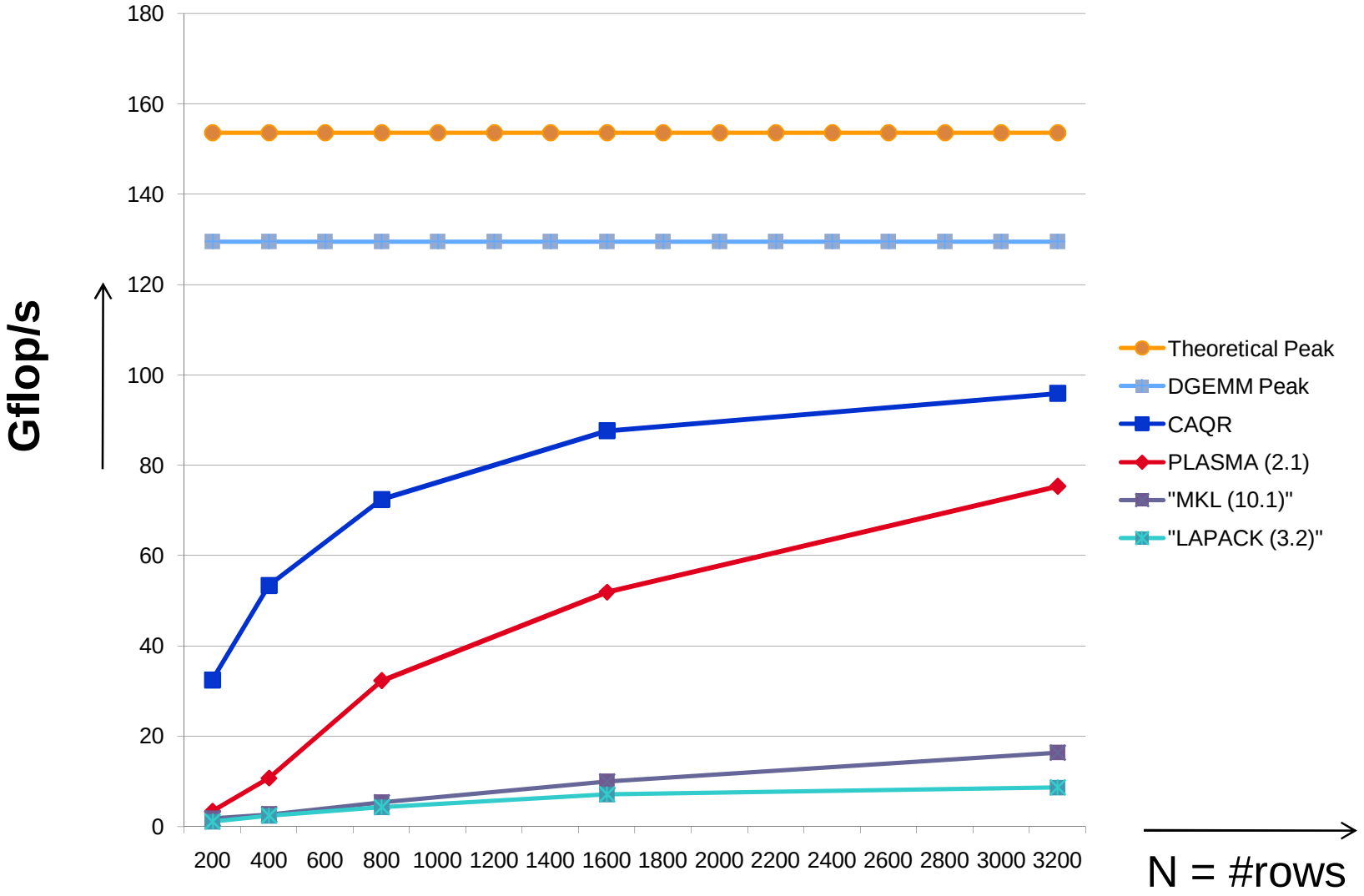
Can Choose reduction tree dynamically

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 – early results
    - Up to 2.5x speedup (vs LAPACK on Nehalem)
    - Better: 10x on Fermi with standard algorithm and better SGEMV
  - Grid – 4x on 4 cities (Dongarra et al)
  - Cloud – early result – up and running using Nexus
- Sequential
  - Out-of-Core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished

QR Factorization Intel 16 cores
Tall Skinny Matrices

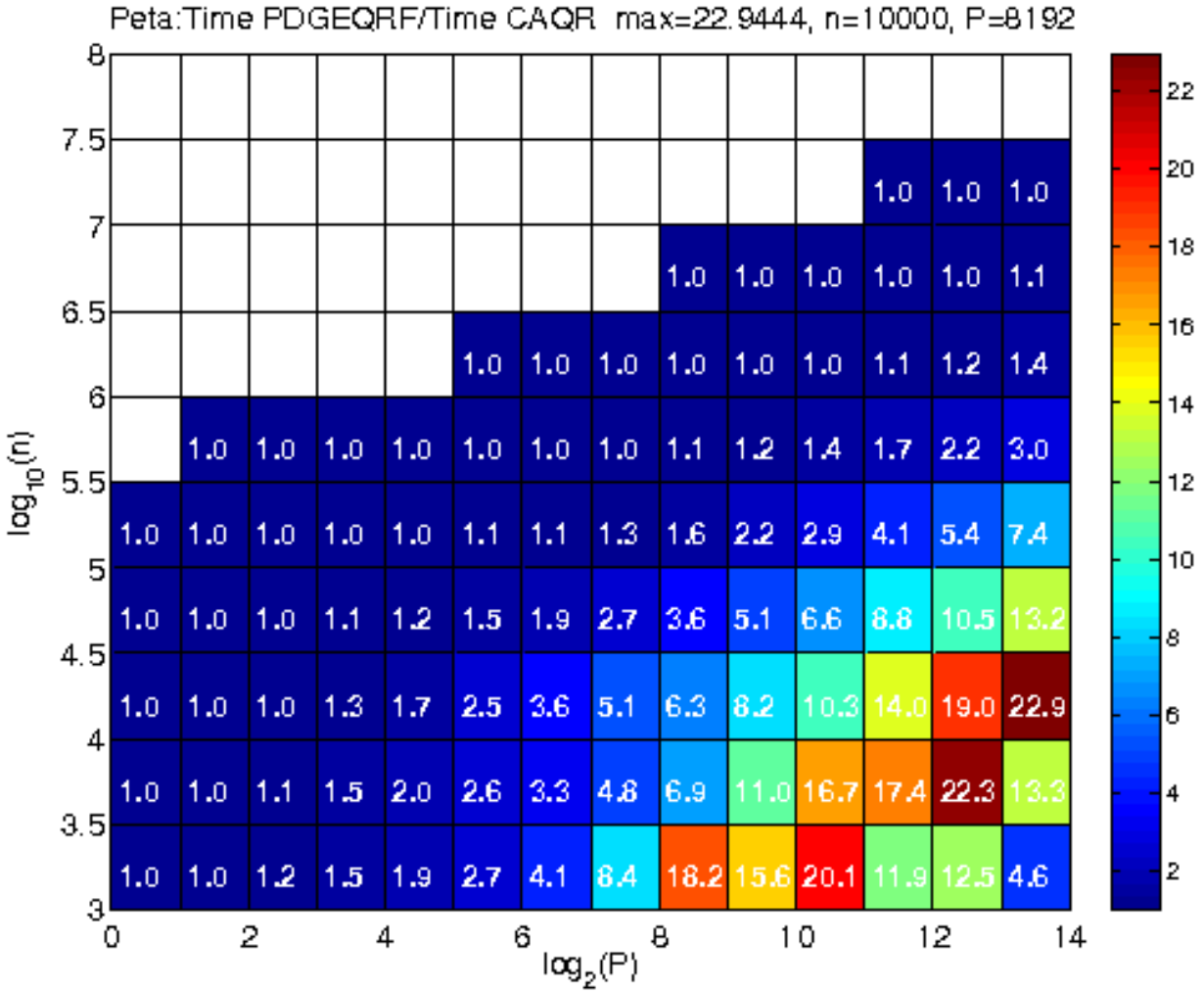
Quad Socket/Quad Core Intel Xeon
See LAPACK Working Note 222
Source: Jack Dongarra



#cols = *51200* x #rows

Modeled Speedups of CAQR vs ScaLAPACK

(on n x n matrices; uses TSQR for panel factorization)



Petascale
up to 22.9x

IBM Power 5
up to 9.7x

"Grid"
up to 11x

Petascale machine with 8192 procs, each at 500 GFlops/s, a bandwidth of 4 GB/s.

$$\gamma = 2 \cdot 10^{-12}\, s, \alpha = 10^{-5}\, s, \beta = 2 \cdot 10^{-9}\, s / word.$$

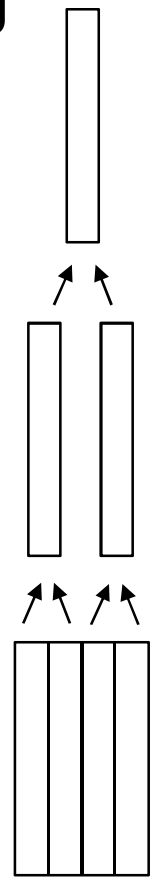
Rank Revealing CAQR Decompositions

Randomized URV

- [V,R'] = caqr(randn(n,n))
  [U,R] = caqr($A \cdot V^T$)
- A = URV reveals the rank of A with high probability
- Applies to $A_1^{\pm 1} \cdot A_2^{\pm 1} \cdot A_3^{\pm 1} \cdot \cdot \cdot$
- Used for eigensolver/SVD

QR with column pivoting

- "Tournament Pivoting" to pick b best columns of 2 groups of b each
- Only works on single matrix



- Both cases: #words_moves = $O(mn^2/\sqrt{M})$,
  or #passes_over_data = $O(n/\sqrt{M})$

- Other CA decompositions with pivoting:
  - LU (tournament, not partial pivoting, but stable!)
  - Cholesky with diagonal pivoting
  - LU with complete pivoting
  - LDL^T ?

Back to LU: 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)

Making TSLU Numerically Stable

- Details matter
  - Going up the tree, we could do LU either on original rows of A (tournament pivoting), or computed rows of U
  - Only tournament pivoting stable

- Thm: New scheme as stable as Partial Pivoting (GEPP) in following sense: Get same Schur complements as GEPP applied to different input matrix whose entries are blocks taken from input A

What about algorithms like Strassen?

- Results for sequential case only
- Restatement of result so far for $O(n^3)$ matmul:

  $\#words\_moved = \Omega(n^3 / M^{3/2-1})$
- Lower bound for Strassen's method:

  $\#words\_moved = \Omega(n^\omega / M^{\omega/2-1})$ where $\omega = \log_2 7$
- Proof very different than before!
- Attained by usual recursive (cache-oblivious) implementation
- Also attained by new algorithms for LU, QR, eig, SVD that use Strassen
  - But not clear how to extend lower bound
- Also attained by fast matmul, LU, QR, eig, SVD *for any* ω
  - Can be made numerically stable
  - True for any matrix multiplication algorithm *that will ever be invented*
  - Conjecture: same lower bound as above
- What about parallelism?

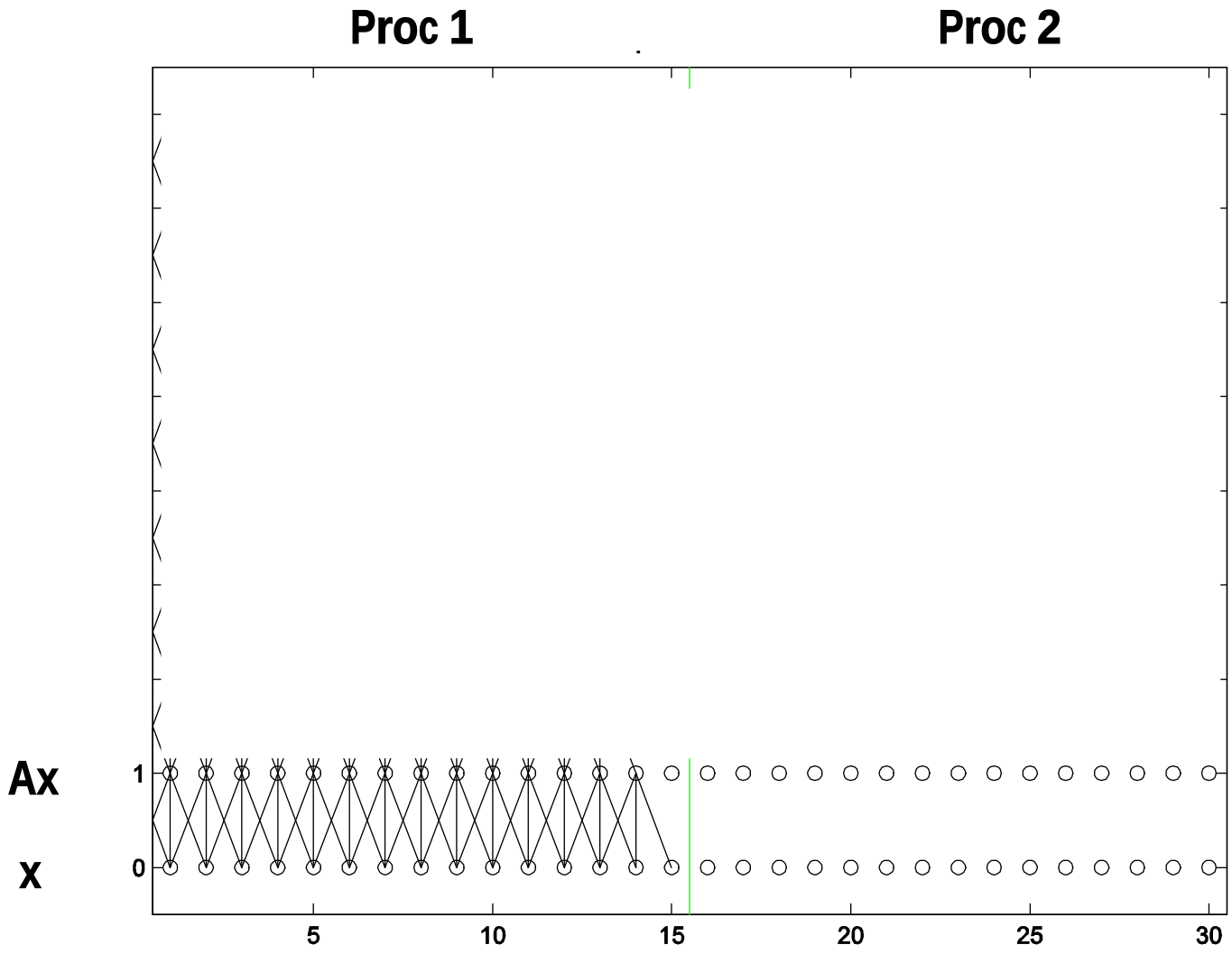
Direct Linear Algebra: summary and future work

- Communication lower bounds on #words_moved and #messages
  - BLAS, LU, Cholesky, QR, eig, SVD, tensor contractions, …
  - Some whole programs (compositions of these operations, no matter how individual ops are interleaved, eg computing A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)
- Algorithms that attain these lower bounds
  - Nearly all dense problems (some open problems, eg LDL')
  - A few sparse problems
- Speed ups in theory and practice
- Extensions to Strassen-like algorithms
- Future work
  - Lots to implement, autotune
    - Next generation of Sca/LAPACK on heterogeneous architectures (MAGMA)
  - Few algorithms in sparse case (just Cholesky)
  - 3D Algorithms (only for Matmul so far), may be important for scaling

Avoiding Communication in Iterative Linear Algebra

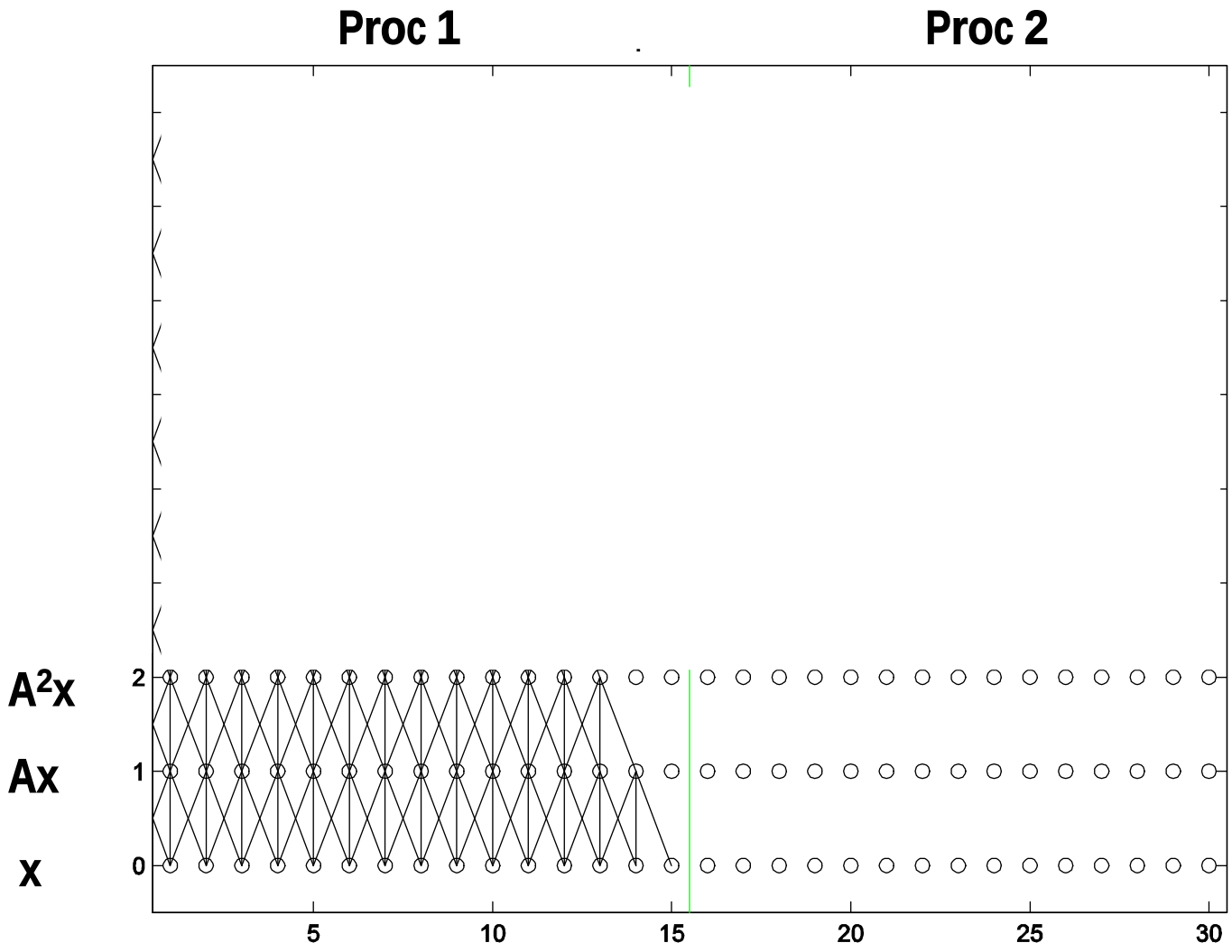
- k-steps of typical iterative solver for sparse Ax=b or Ax=λx
  - Does k SpMVs with starting vector
  - Finds "best" solution among all linear combinations of these k+1 vectors
  - Many such "Krylov Subspace Methods"
    - Conjugate Gradients, GMRES, Lanczos, Arnoldi, …
- Goal: minimize communication in Krylov Subspace Methods
  - Assume matrix "well-partitioned," with modest surface-to-volume ratio
  - Parallel implementation
    - Conventional: O(k log p) messages, because k calls to SpMV
    - **New: O(log p) messages - optimal**
  - Serial implementation
    - Conventional: O(k) moves of data from slow to fast memory
    - **New**: **O(1) moves of data – optimal**
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation
- Much prior work: See PhD Thesis by Mark Hoemmen
  - See bebop.cs.berkeley.edu
  - CG: [van Rosendale, 83], [Chronopoulos and Gear, 89]
  - GMRES: [Walker, 88], [Joubert and Carey, 92], [Bai et al., 94]

Locally Dependent Entries for [x,Ax], A tridiagonal, 2 processors



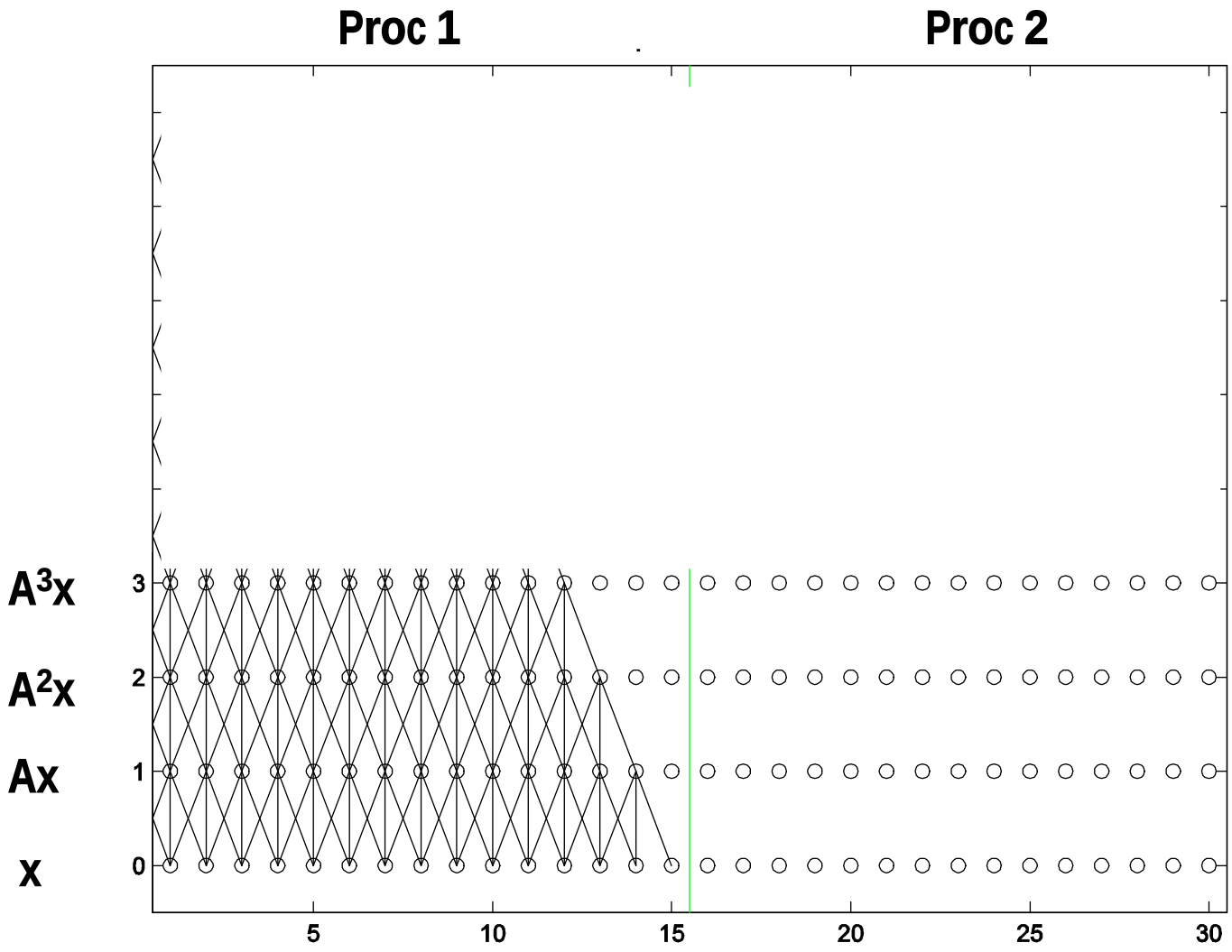
Can be computed without communication

Locally Dependent Entries for [x,Ax,A^2x], A tridiagonal, 2 processors



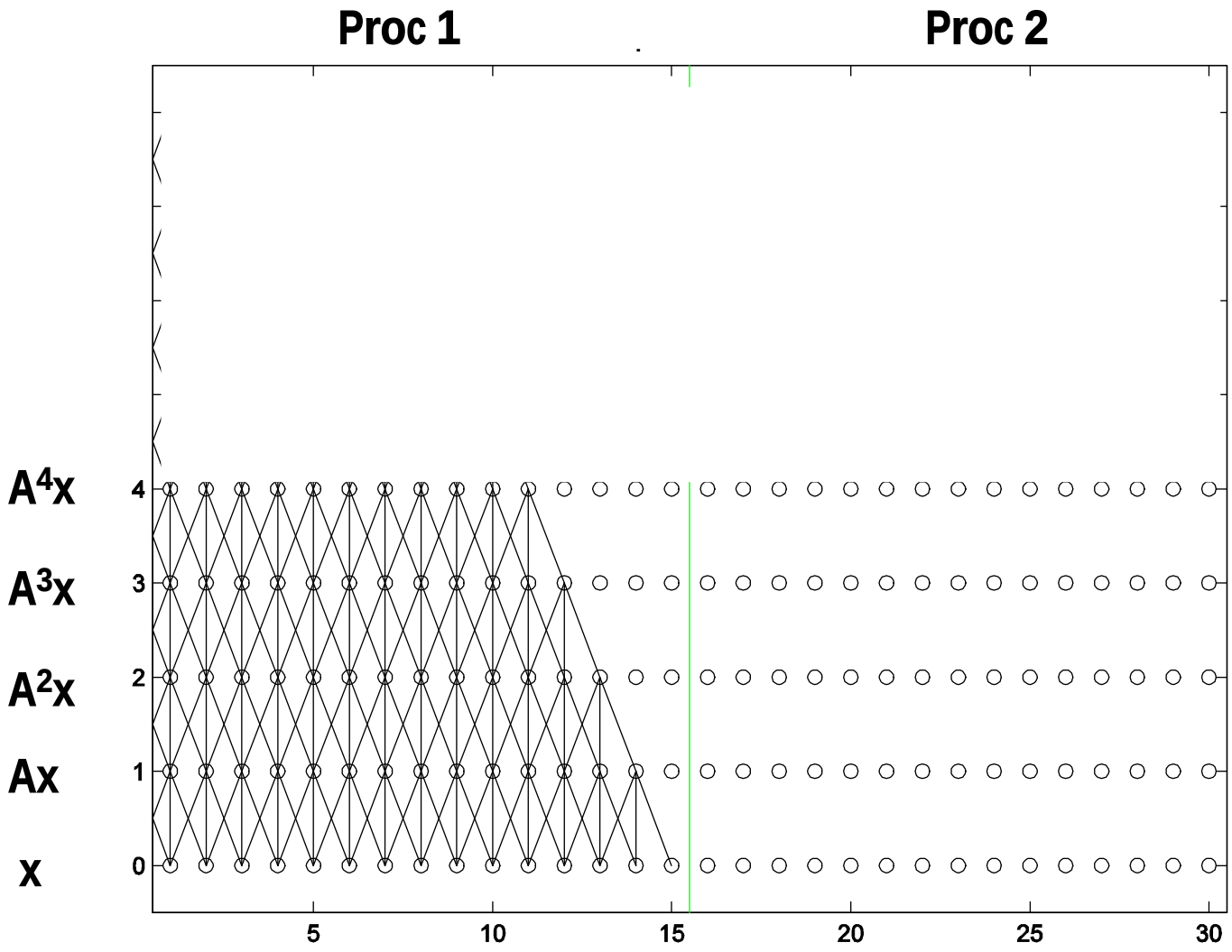
Can be computed without communication

Locally Dependent Entries for [x,Ax,...,A^3x], A tridiagonal, 2 processors



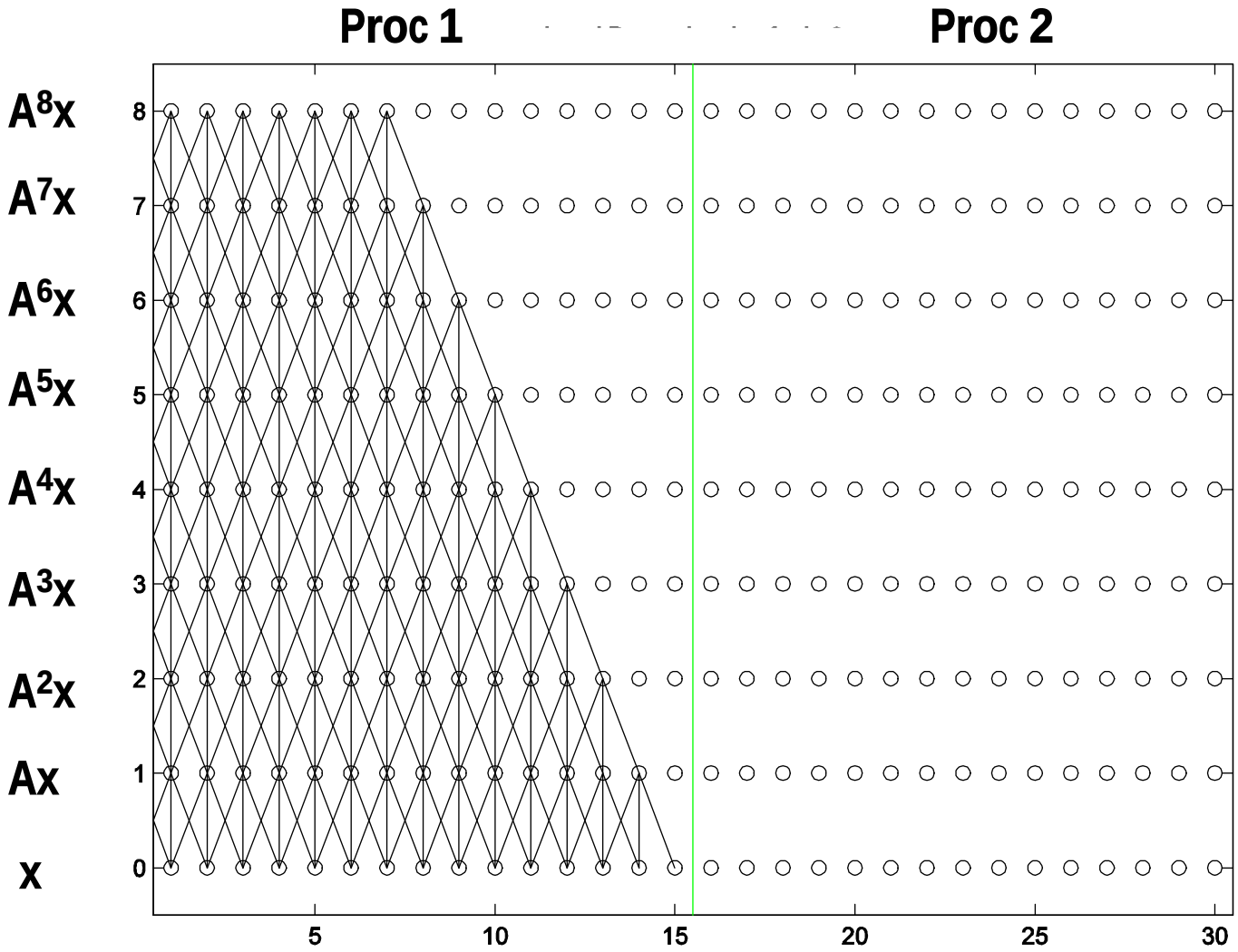
Can be computed without communication

Locally Dependent Entries for [x,Ax,...,A^4x], A tridiagonal, 2 processors



Can be computed without communication

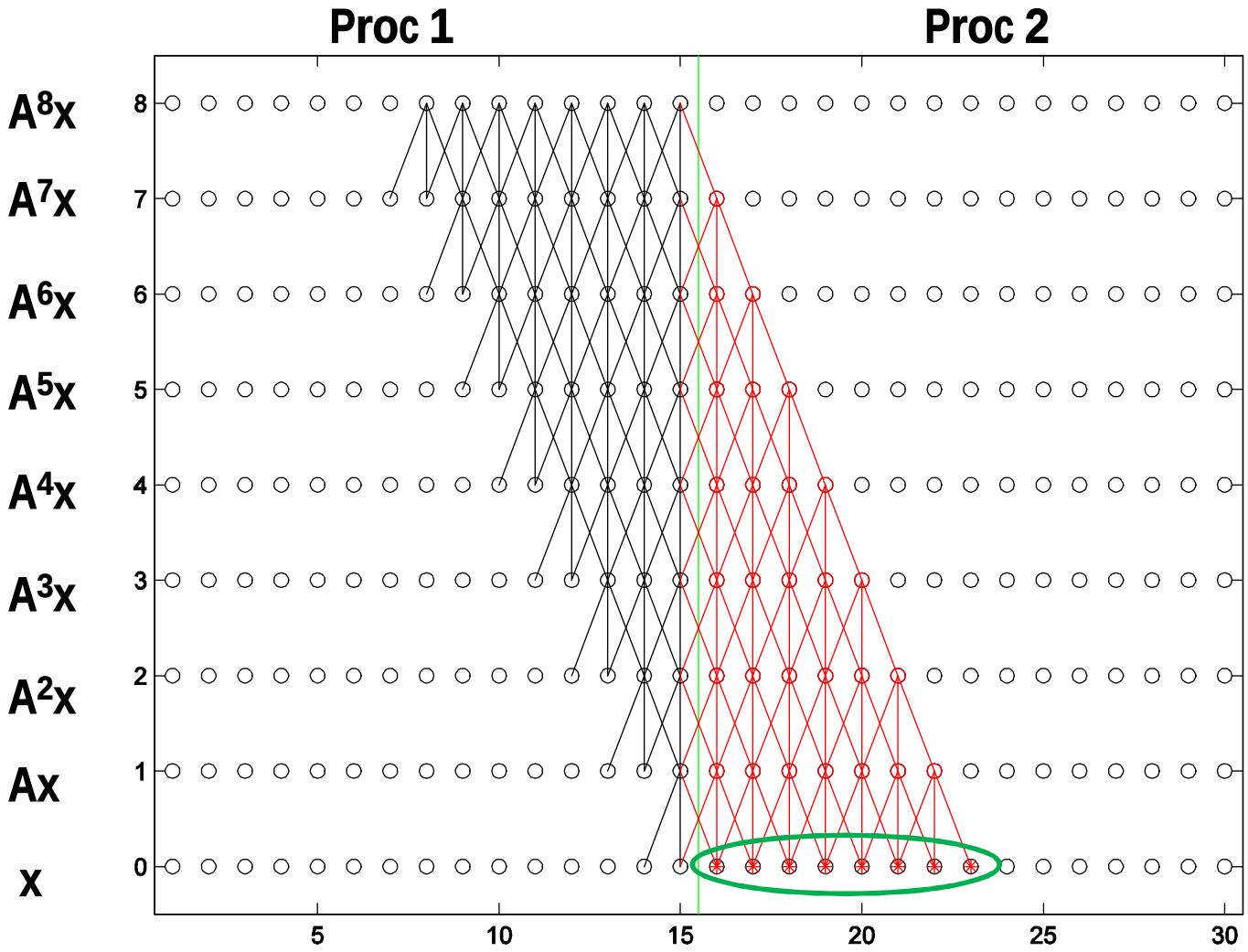
Locally Dependent Entries for [x,Ax,...,A^8x], A tridiagonal, 2 processors



Can be computed without communication

k=8 fold reuse of A

Remotely Dependent Entries for [x,Ax,...,A^8x], A tridiagonal, 2 processors

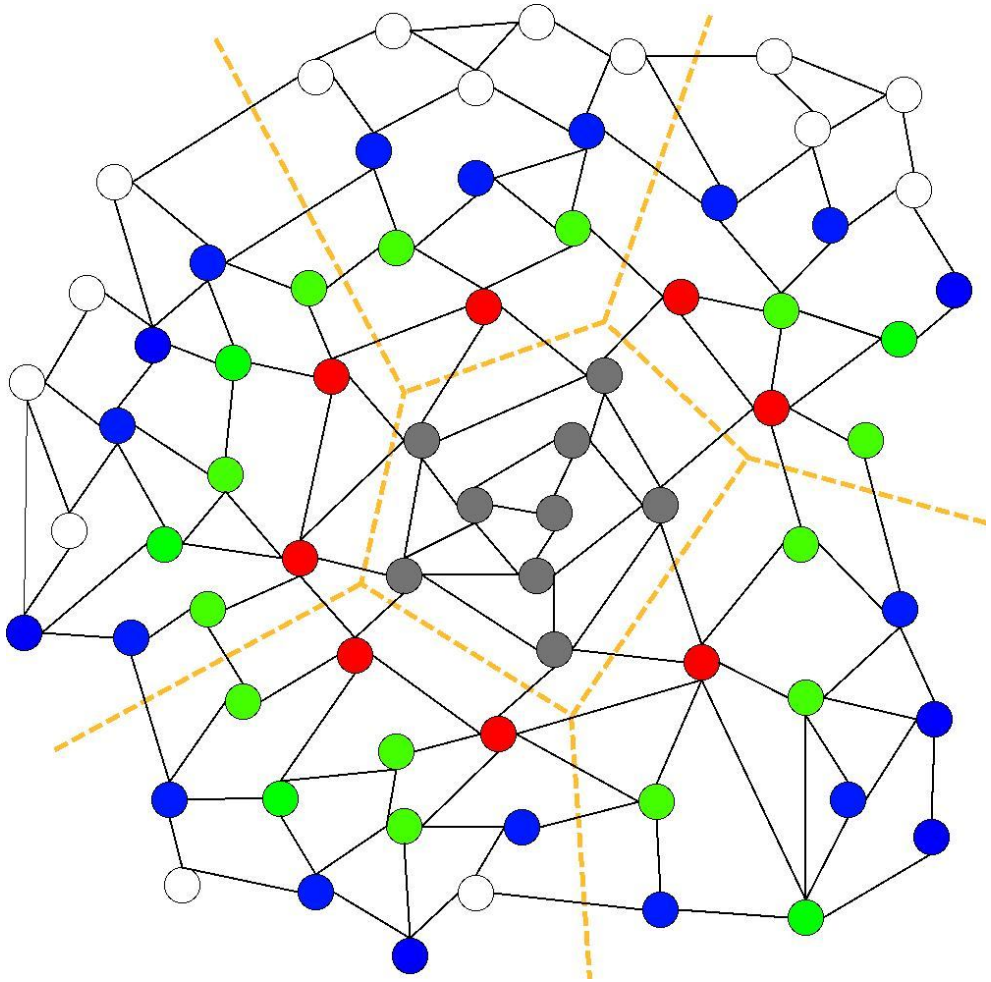


One **message to get data needed to compute remotely dependent entries, *not k=8***

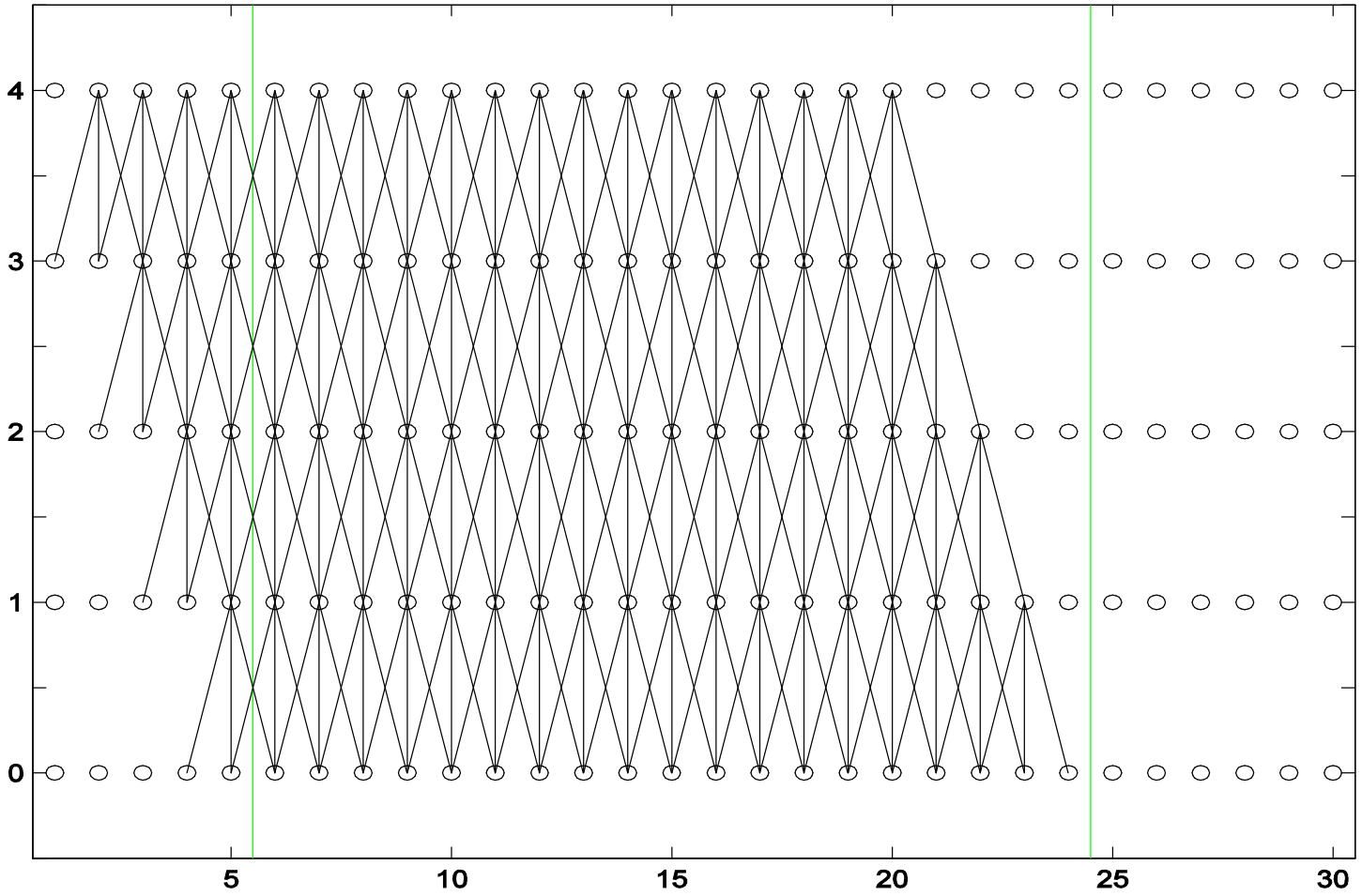
Minimizes number of messages = latency cost

Price: redundant work ∝ "surface/volume ratio"

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



Sequential [x,Ax,…,A^4x], with memory hierarchy



One* read of matrix from slow memory, *not k=4

Minimizes words moved = bandwidth cost

No redundant work

Minimizing Communication of GMRES to solve Ax=b

- GMRES: find x in span{b,Ab,…,A^kb} minimizing || Ax-b ||$_2$
- Cost of k steps of standard GMRES vs new GMRES

Standard GMRES

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

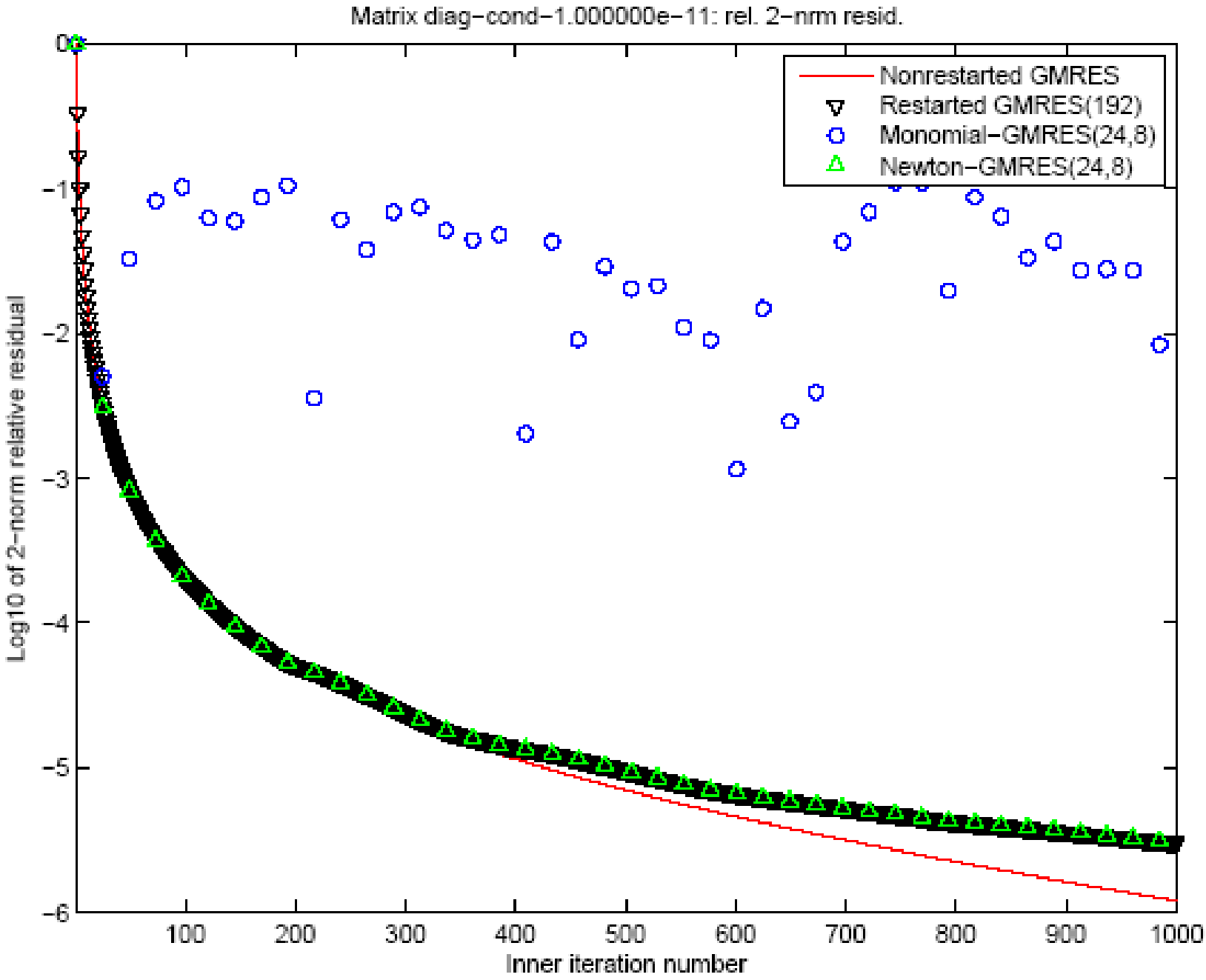
Communication-avoiding GMRES

```
W = [ v, Av, A²v, ... , A^k v ]
[Q,R] = TSQR(W)  ...  "Tall Skinny QR"
Build H from R, solve LSQ problem
```

Sequential: #words_moved =
O(k·nnz) from SpMV
+ O(k^2·n) from MGS
Parallel: #messages =
O(k) from SpMV
+ O(k^2 · log p) from MGS

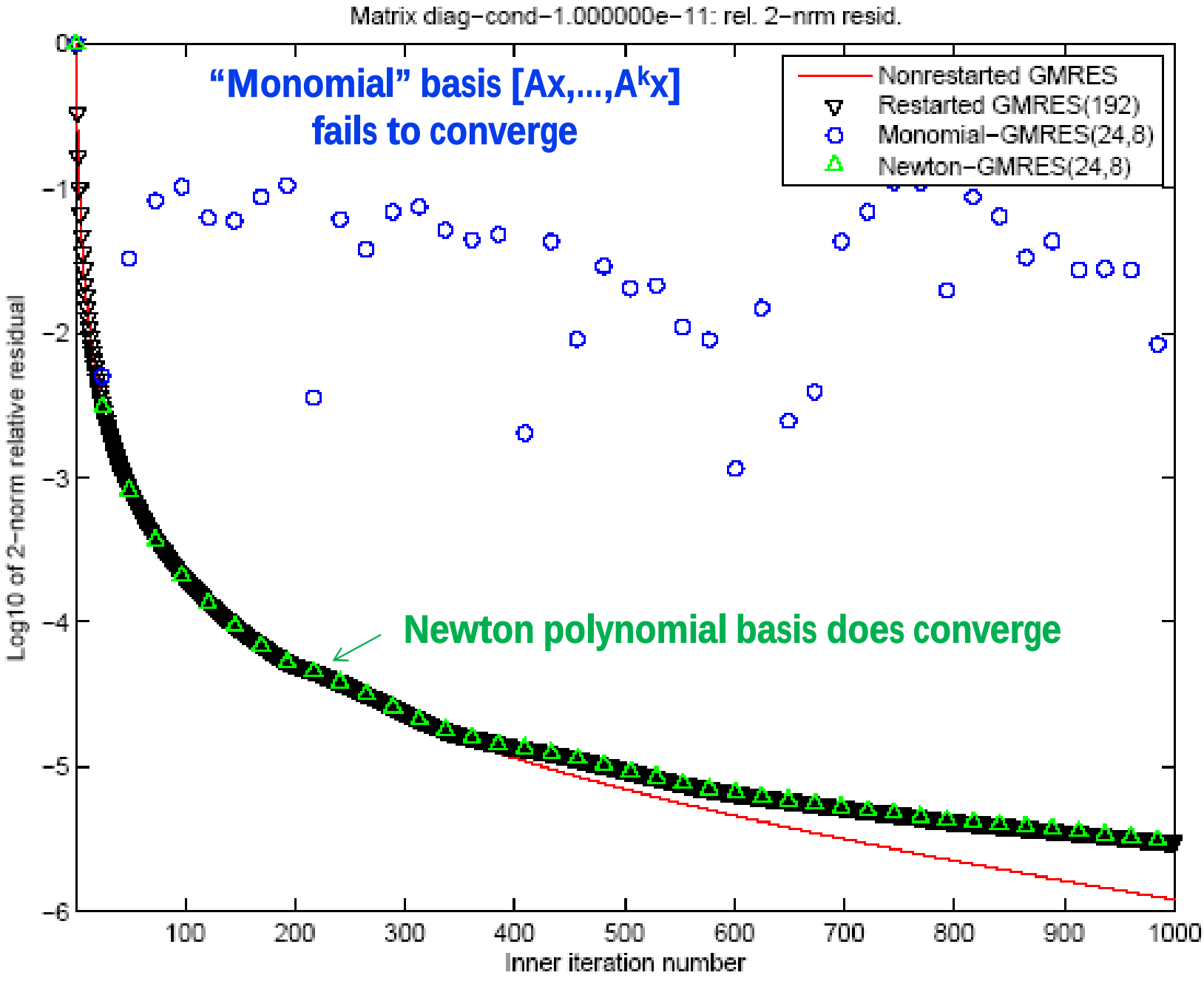
Sequential: #words_moved =
O(nnz) from SpMV
+ O(k·n) from TSQR
Parallel: #messages =
O(1) from computing W
+ O(log p) from TSQR

- **Oops – W from power method, precision lost!**


Matrix diag−cond−1.000000e−11: rel. 2−nrm resid.
Nonrestarted GMRES
Restarted GMRES(192)
Monomial−GMRES(24,8)
Newton−GMRES(24,8)
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

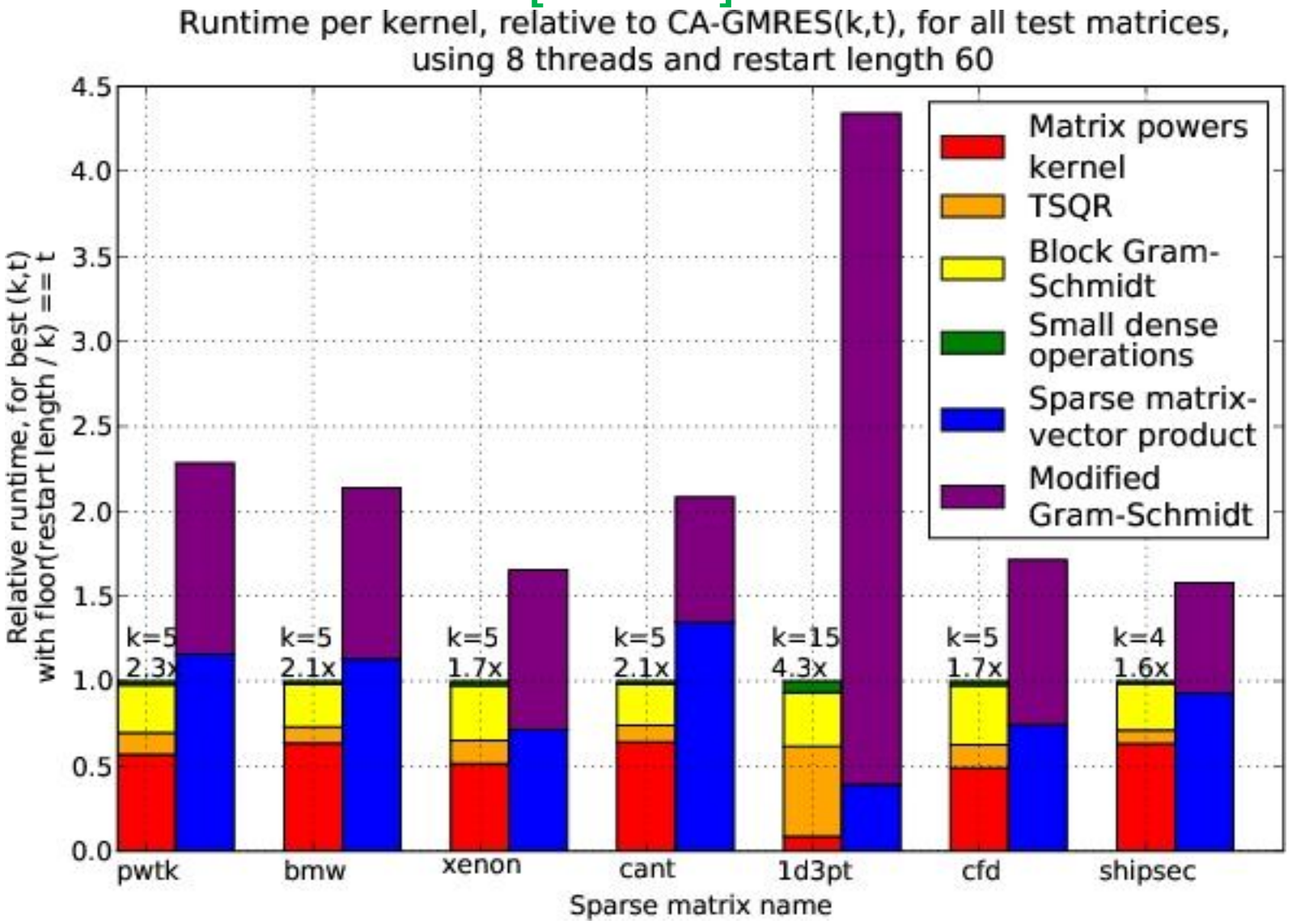
How to make CA-GMRES Stable?

- Use a different polynomial basis for the Krylov subspace
- Not Monomial basis $W = [v, Av, A^2v, \ldots]$, instead $[v, p_1(A)v, p_2(A)v, \ldots]$
- Possible choices:
  - Newton Basis $W_N = [v, (A - \theta_1 I)v, (A - \theta_2 I)(A - \theta_1 I)v, \ldots]$
    - Shifts θ_i chosen as approximate eigenvalues from Arnoldi
    - Using same Krylov subspace, so "free"
  - Chebyshev Basis $W_C = [v, T_1(A)v, T_2(A)v, \ldots]$
    - $T_i(z)$ chosen to be small in region of complex plane containing large eigenvalues


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)
Newton polynomial basis 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]



Communication Avoiding Iterative Linear Algebra: Future Work

- Lots of algorithms to implement, autotune
  - Make widely available via OSKI, Trilinos, PETSc, Python, …
  - Job available…
- Extensions to other Krylov subspace methods
  - So far just Lanczos/CG, Arnoldi/GMRES, BiCG
  - Need BiCGStab, CGS, QMR, …
- Add preconditioning to solve MAx = Mb
  - New kernel [x, Ax, MAx, AMAx, MAMAx, AMAMAx, …]
  - Diagonal M easy
  - Block diagonal M harder
    - (AM)k quickly becomes dense but with low-rank off-diagonal blocks
  - Extends to hierarchical, semiseparable M
    - Theory exists, no implementations yet

For more information

- See papers at bebop.cs.berkeley.edu
- See slides for week-long short course
  - Gene Golub SIAM Summer School in Numerical Linear Algebra
  - www.ba.cnr.it/ISSNLA2010/index.htm

Summary

Time to redesign all dense and sparse linear algebra

Don’t Communic…