Avoiding Communication in Linear Algebra

Jim Demmel
UC Berkeley

bebop.cs.berkeley.edu

Motivation

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

communication

Motivation

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

  communication

- Exponentially growing gaps between
  - Time_per_flop << 1/Network BW << Network Latency
    - **Improving 59%/year vs 26%/year vs 15%/year**
  - Time_per_flop << 1/Memory BW << Memory Latency
    - **Improving 59%/year vs 23%/year vs 5.5%/year**

Motivation

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

  (the last two terms: communication)

- Exponentially growing gaps between
  - Time_per_flop << 1/Network BW << Network Latency
    - **Improving 59%/year vs 26%/year vs 15%/year**
  - Time_per_flop << 1/Memory BW << Memory Latency
    - **Improving 59%/year vs 23%/year vs 5.5%/year**
- Goal : reorganize linear algebra to *avoid* communication
  - Not just *hiding* communication (speedup $\leq$ 2x)
  - Arbitrary speedups possible

Outline

- Motivation
- Avoiding Communication in Dense Linear Algebra
- Avoiding Communication in Sparse Linear Algebra

Outline

- Motivation
- Avoiding Communication in Dense Linear Algebra
- Avoiding Communication in Sparse Linear Algebra
- A poem in memory of Gene Golub (separate file)

Collaborators (so far)

- UC Berkeley
  - Kathy Yelick, Ming Gu
  - Mark Hoemmen, Marghoob Mohiyuddin, Kaushik Datta, George Petropoulos, Sam Williams, BeBOp group
  - Lenny Oliker, John Shalf
- CU Denver
  - Julien Langou
- INRIA
  - Laura Grigori, Hua Xiang

- Much related work
  - Complete references in technical reports

Why all our problems are solved for dense linear algebra– in theory

- (Talk by Ioana Dumitriu on Monday)
- Thm (D., Dumitriu, Holtz, Kleinberg) (Numer.Math. 2007)
  - Given any matmul running in O(n^{ω}) ops for some ω>2, it can be made stable and still run in O($n^{\omega+\varepsilon}$) ops, for any ε>0.
    - Current record: $\omega \approx 2.38$
- Thm (D., Dumitriu, Holtz) (Numer. Math. 2008)
  - Given any stable matmul running in O($n^{\omega+\varepsilon}$) ops, it is possible to do backward stable dense linear algebra in O($n^{\omega+\varepsilon}$) ops:
    - GEPP, QR
    - rank revealing QR (randomized)
    - (Generalized) Schur decomposition, SVD (randomized)
- Also reduces communication to O($n^{\omega+\varepsilon}$)
- But constants?

Summary (1) – Avoiding Communication in Dense Linear Algebra

- QR or LU decomposition of m x n matrix, m >> n
  - Parallel implementation
    - Conventional: O(n log p) messages
    - **"New"**: O(log p) messages - optimal
  - Serial implementation with fast memory of size F
    - Conventional: O(mn/F) moves of data from slow to fast memory
      - mn/F = how many times larger matrix is than fast memory
    - **"New"**: O(1) moves of data - optimal
  - Lots of speed up possible (measured and modeled)
    - Price: some redundant computation, stability?
- Extends to square case, with optimality results
- Extends to other architectures (eg multicore)
- (Talk by Julien Langou Monday, on QR)

Minimizing Comm. in Parallel QR

- QR decomposition of m x n matrix W, m >> n
  - TSQR = “Tall Skinny QR”
  - P processors, block row layout
- Usual Parallel Algorithm
  - Compute Householder vector for each column
  - Number of messages $\propto$ n log P
- Communication Avoiding Algorithm
  - Reduction operation, with QR as operator
  - Number of messages $\propto$ log P

$$W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \begin{matrix} \to \\ \to \\ \to \\ \to \end{matrix} \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \begin{matrix} \searrow \\ \nearrow \\ \searrow \\ \nearrow \end{matrix} \begin{matrix} R_{01} \\ \\ \\ R_{11} \end{matrix} \begin{matrix} \searrow \\ \\ \nearrow \end{matrix} R_{02}$$

TSQR in more detail

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

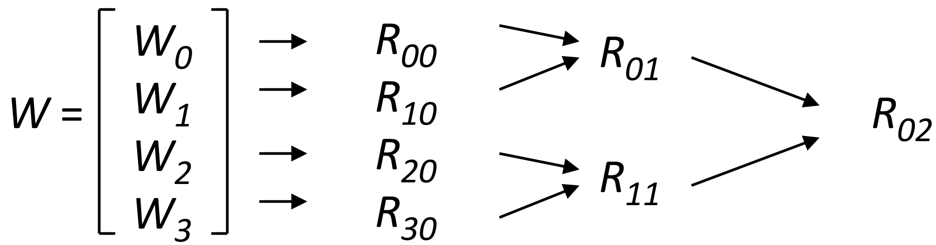
$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix}$$

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

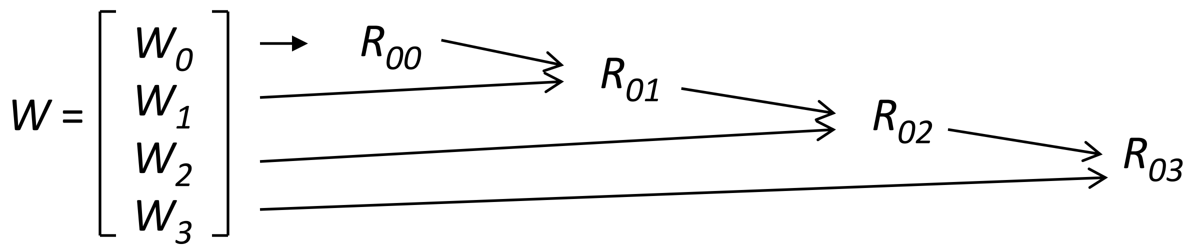
Q is represented implicitly as a product
(tree of factors)

Minimizing Communication in TSQR

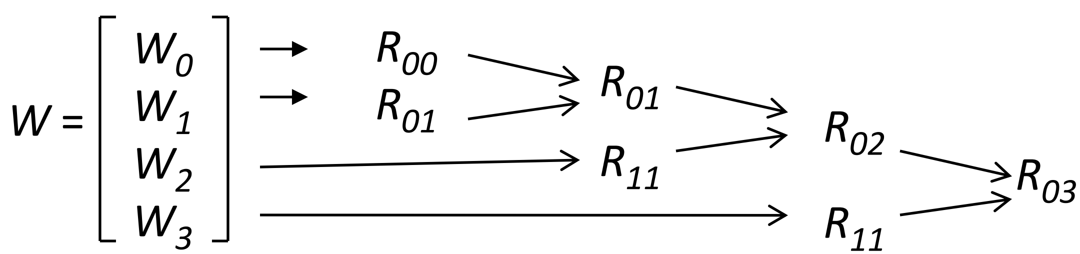
Parallel:



Sequential:



Dual Core:



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

Choose reduction tree dynamically

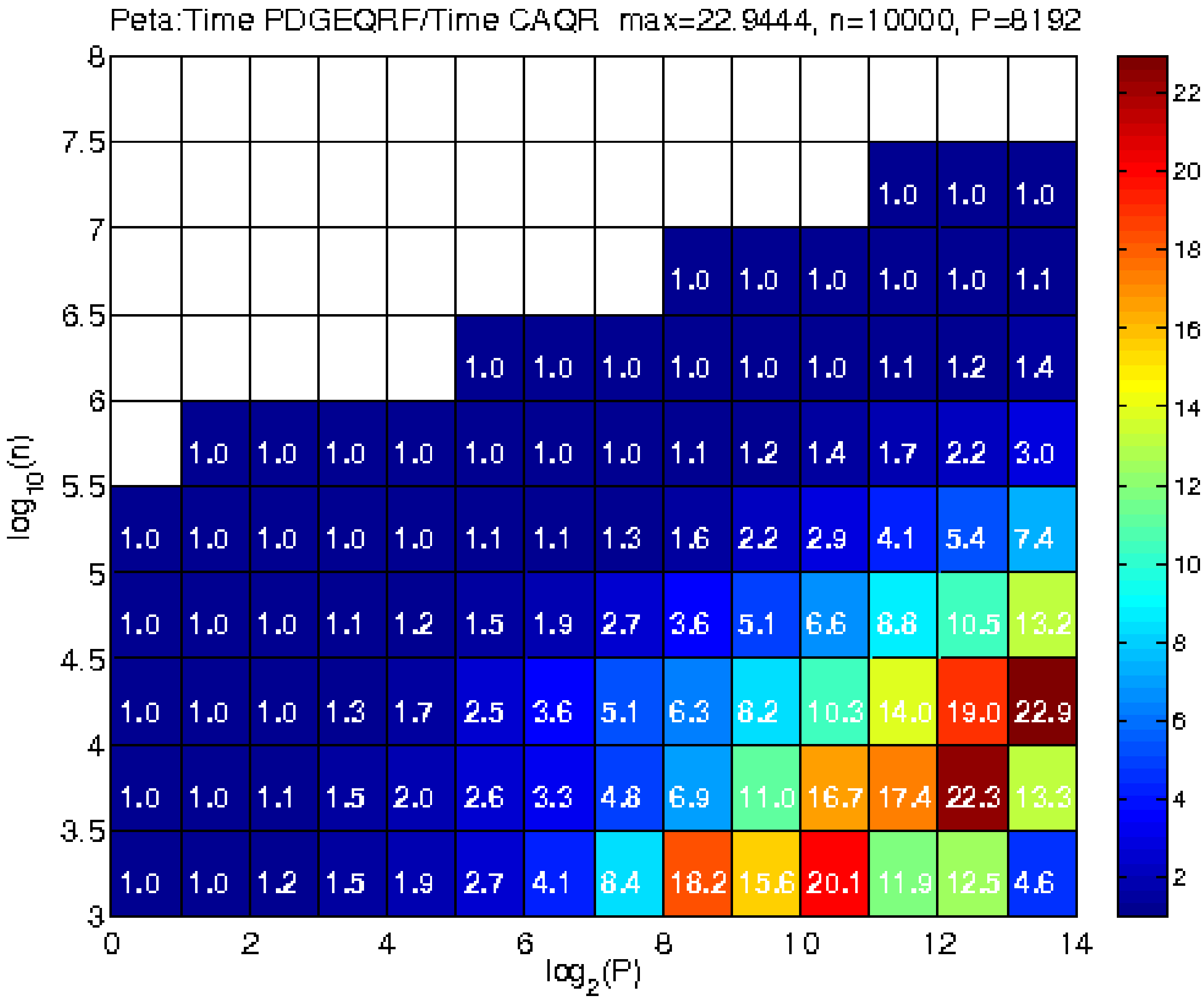
Performance of TSQR vs Sca/LAPACK

- Parallel
  - 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)
  - Both use Elmroth-Gustavson locally – enabled by TSQR
- Sequential
  - OOC on PowerPC laptop
    - As little as **2x slowdown vs (predicted) infinite DRAM**
- See UC Berkeley EECS Tech Report 2008-74

QR for General Matrices

- CAQR – Communication Avoiding QR for general A
  - Use TSQR for panel factorizations
  - Apply to rest of matrix
- Cost of CAQR vs ScaLAPACK's PDGEQRF
  - n x n matrix on $P^{1/2}$ x $P^{1/2}$ processor grid, block size b
  - Flops: $(4/3)n^3/P + (3/4)n^2 b \log P/P^{1/2}$ vs $(4/3)n^3/P$
  - Bandwidth: $(3/4)n^2 \log P/P^{1/2}$ vs same
  - Latency: $2.5\ n \log P / \mathbf{b}$ vs $1.5\ n \log P$
- Close to optimal (modulo log P factors)
  - Assume: $O(n^2/P)$ memory/processor, $O(n^3)$ algorithm,
  - Choose b near $n / P^{1/2}$ (its upper bound)
  - Bandwidth lower bound: $\Omega(n^2 / P^{1/2})$ – just log(P) smaller
  - Latency lower bound: $\Omega(P^{1/2})$ – just polylog(P) smaller
  - Extension of Irony/Toledo/Tishkin (2004)
- Implementation – Julien's summer project

Modeled Speedups of CAQR vs ScaLAPACK



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.

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

TSLU: LU factorization of a tall skinny matrix

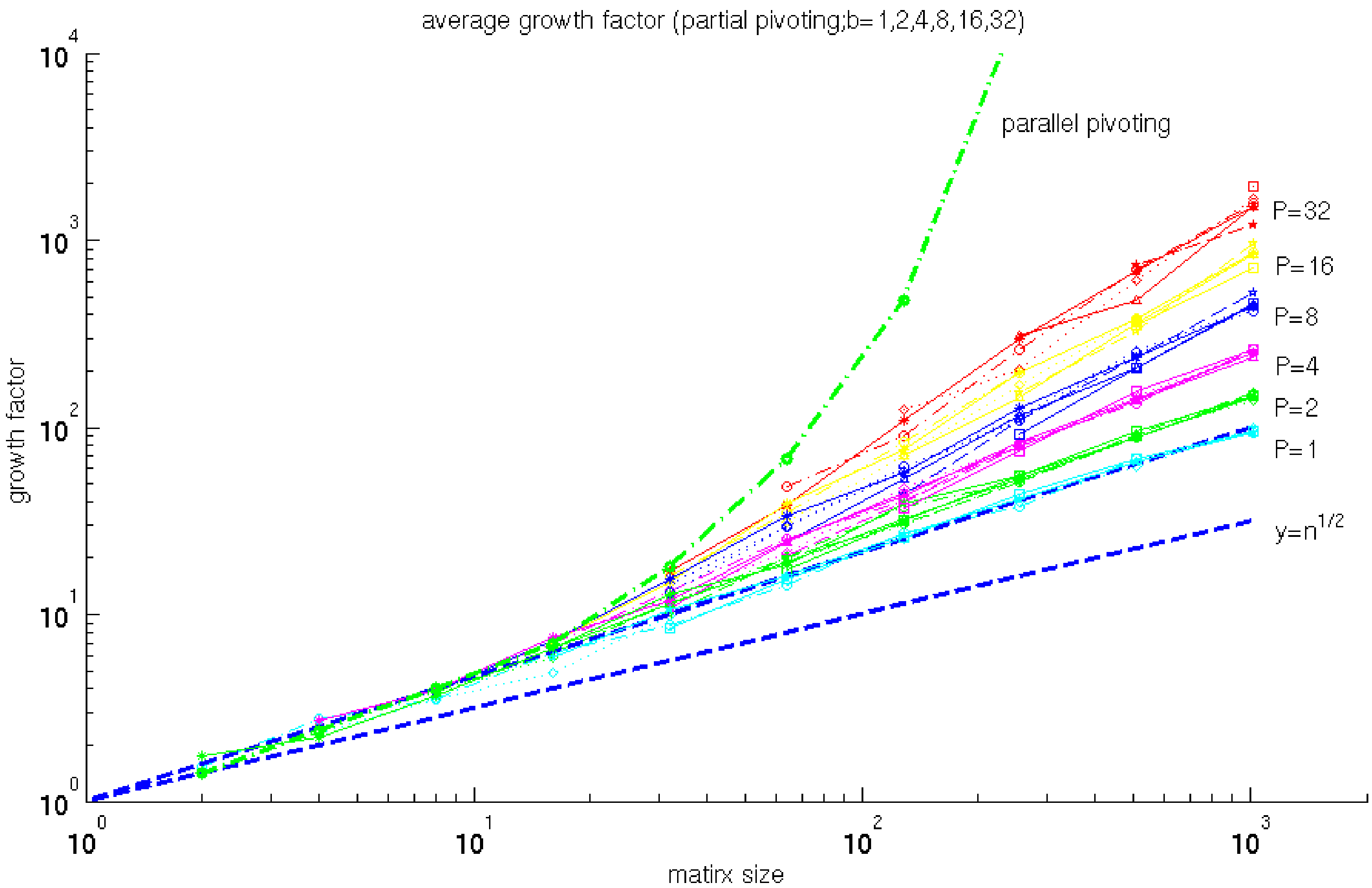
First try the obvious generalization of TSQR:

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix} = \underbrace{\begin{pmatrix} \Pi_{00} & & & \\ \hline & \Pi_{10} & & \\ \hline & & \Pi_{20} & \\ \hline & & & \Pi_{30} \end{pmatrix}}_{\Pi_0} \cdot \begin{pmatrix} L_{00} & & & \\ \hline & L_{10} & & \\ \hline & & L_{20} & \\ \hline & & & L_{30} \end{pmatrix} \cdot \begin{pmatrix} U_{00} \\ \hline U_{10} \\ \hline U_{20} \\ \hline U_{30} \end{pmatrix}$$

$$\begin{pmatrix} U_{00} \\ U_{10} \\ \hline U_{20} \\ U_{30} \end{pmatrix} = \underbrace{\begin{pmatrix} \Pi_{01} & \\ \hline & \Pi_{11} \end{pmatrix}}_{\Pi_1} \cdot \begin{pmatrix} L_{01} & \\ \hline & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{01} \\ \hline U_{11} \end{pmatrix}$$

$$\begin{pmatrix} U_{01} \\ U_{11} \end{pmatrix} = \underbrace{\Pi_{02}}_{\Pi_2} L_{02} U_{02}$$

Growth factor for TSLU based factorization



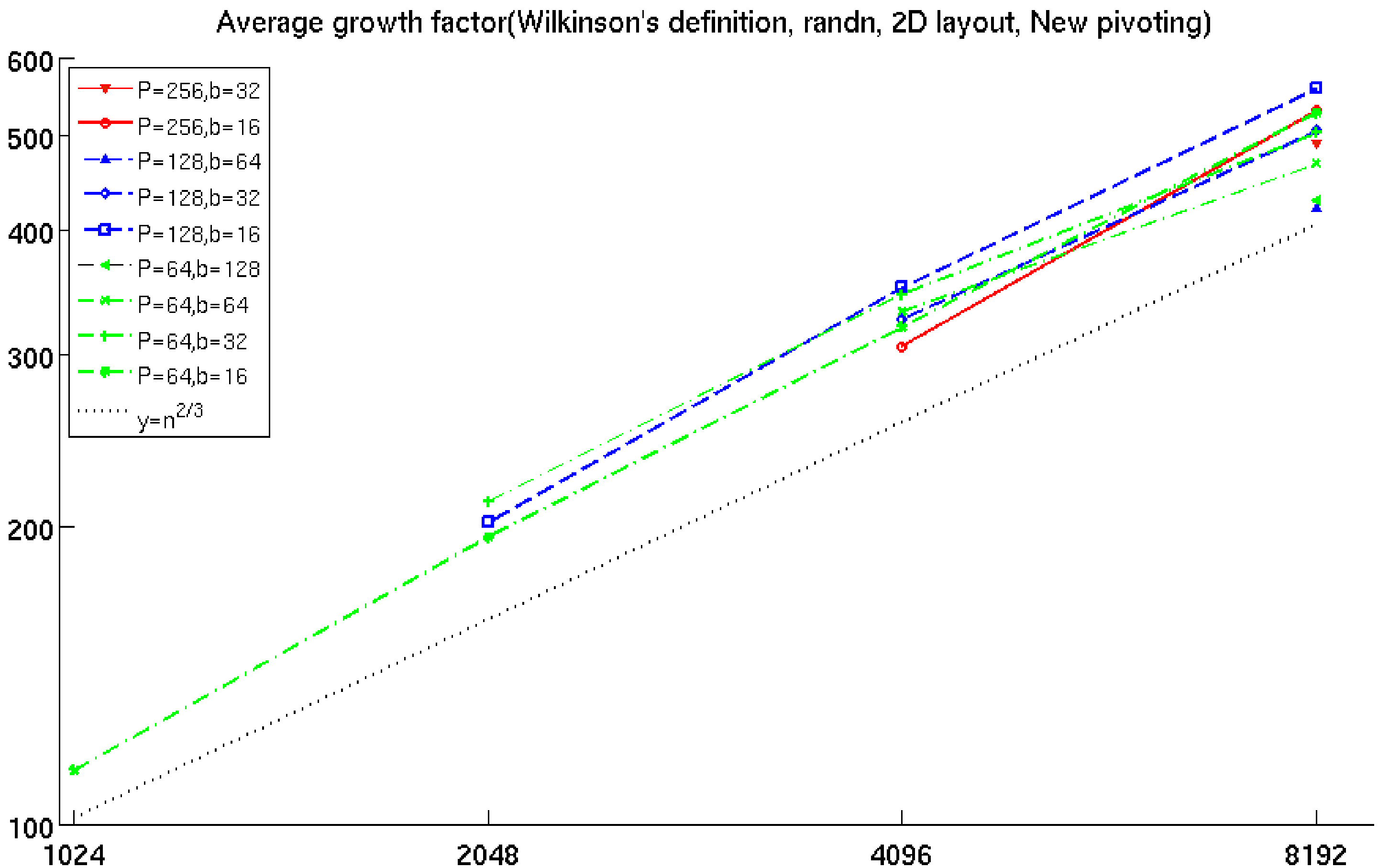
Unstable for large P and large matrices.
When P = # rows, TSLU is equivalent to parallel pivoting.

Courtesy of H. Xiang

Making TSLU Stable

- At each node in tree, TSLU selects b pivot rows from 2b candidates from its 2 child nodes
- At each node, do LU on 2b *original* rows selected by child nodes, not U factors from child nodes
- When TSLU done, permute b selected rows to top of original matrix, redo b steps of LU without pivoting
- CALU – Communication Avoiding LU for general A
  - Use TSLU for panel factorizations
  - Apply to rest of matrix
  - Cost: redundant panel factorizations
- Benefit:
  - Stable in practice, but *not* same pivot choice as GEPP
  - b times fewer messages overall - faster

Growth factor for better CALU approach



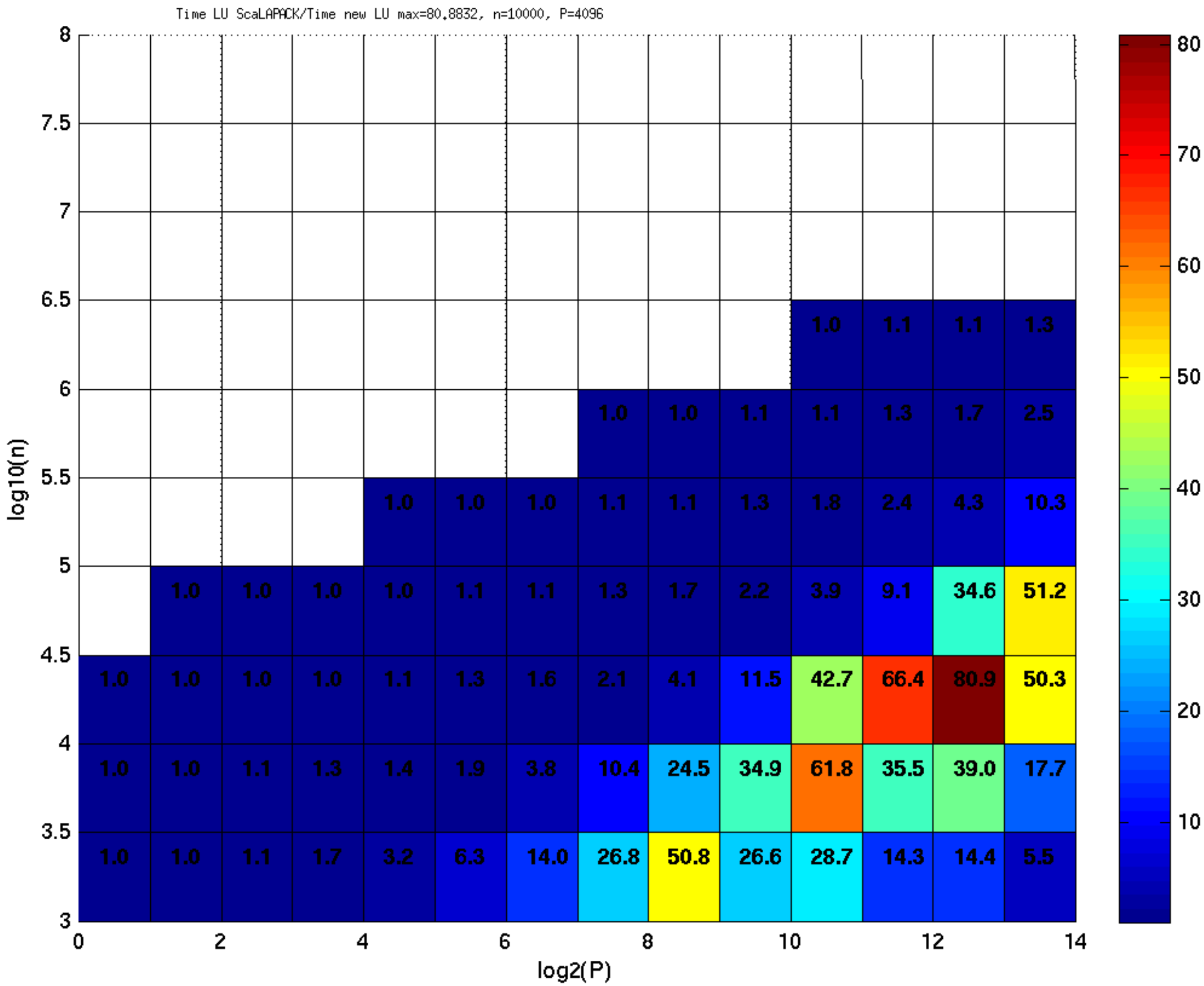
Like threshold pivoting with worst case threshold = .33 , so |L| <= 3

Testing shows about same residual as GEPP

Performance vs ScaLAPACK

- TSLU
  - IBM Power 5
    - Up to **4.37x** faster (16 procs, 1M x 150)
  - Cray XT4
    - Up to **5.52x** faster (8 procs, 1M x 150)
- CALU
  - IBM Power 5
    - Up to **2.29x** faster (64 procs, 1000 x 1000)
  - Cray XT4
    - Up to **1.81x** faster (64 procs, 1000 x 1000)
- Optimality analysis analogous to QR
- See INRIA Tech Report 6523 (2008)

Speedup prediction for a Petascale machine - up to 81x faster



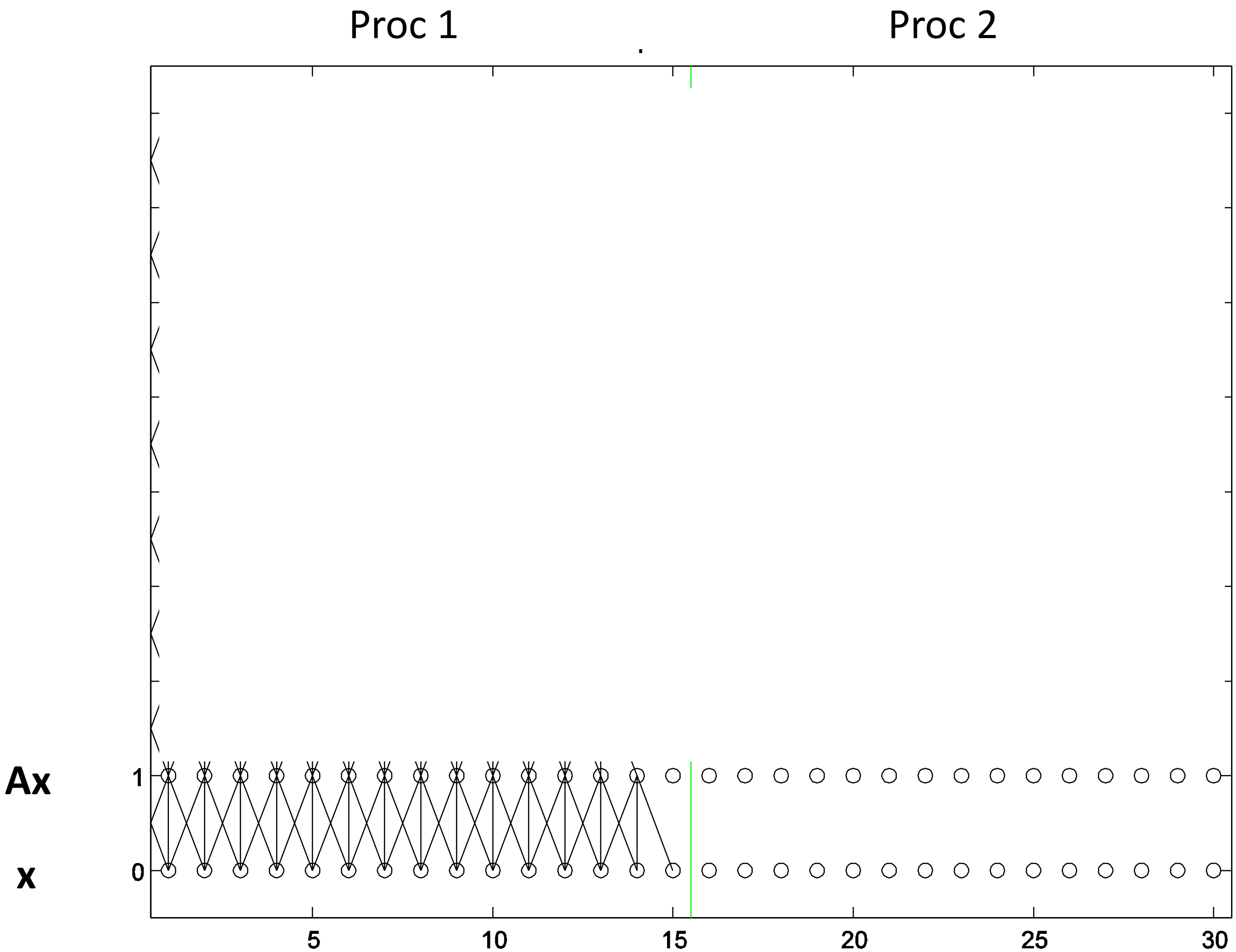
Petascale machine with 8192 procs, each at 500 GFlops/s, a bandwidth of 4 GB/s.

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

Summary (2) – Avoiding Communication in Sparse Linear Algebra

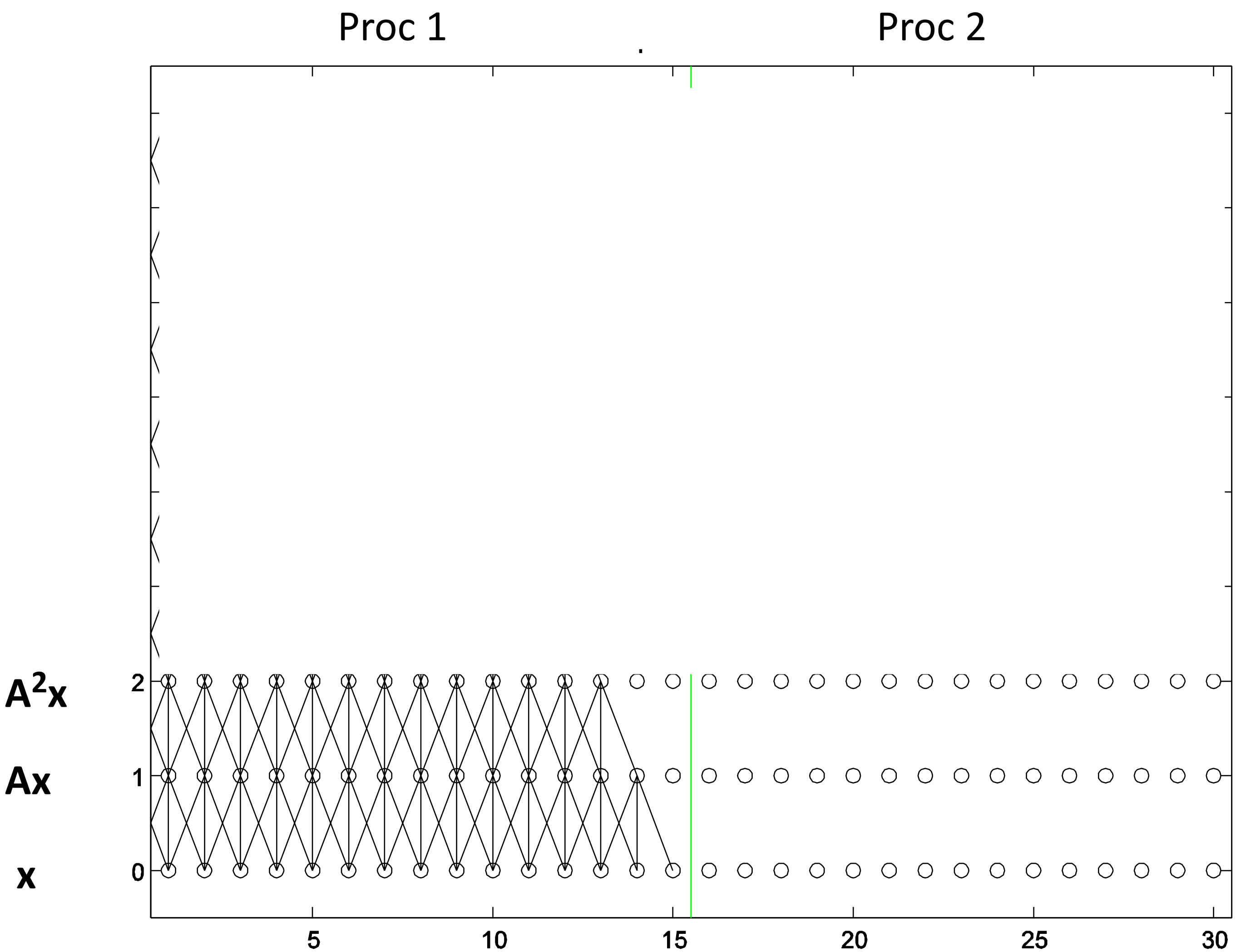
- Take *k* steps of Krylov subspace method
  - GMRES, CG, Lanczos, Arnoldi
  - Assume matrix "well-partitioned," with modest surface-to-volume ratio
  - Parallel implementation
    - Conventional: O(k log p) messages
    - **"New"**: O(log p) messages - optimal
  - Serial implementation
    - Conventional: O(k) moves of data from slow to fast memory
    - **"New"**: O(1) moves of data – optimal
- Can incorporate some preconditioners
  - Hierarchical, semiseparable matrices …
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation

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



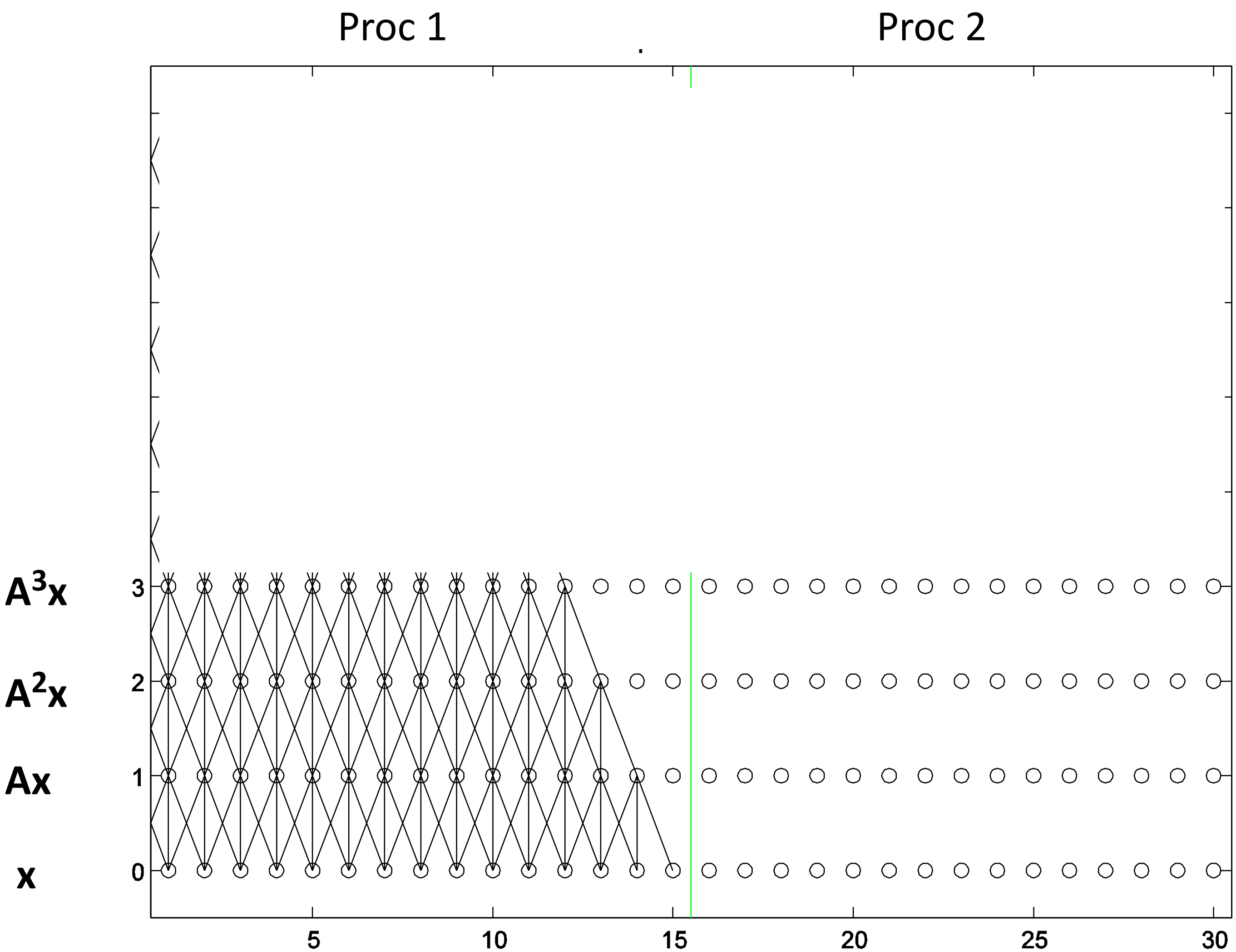
Can be computed without communication

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



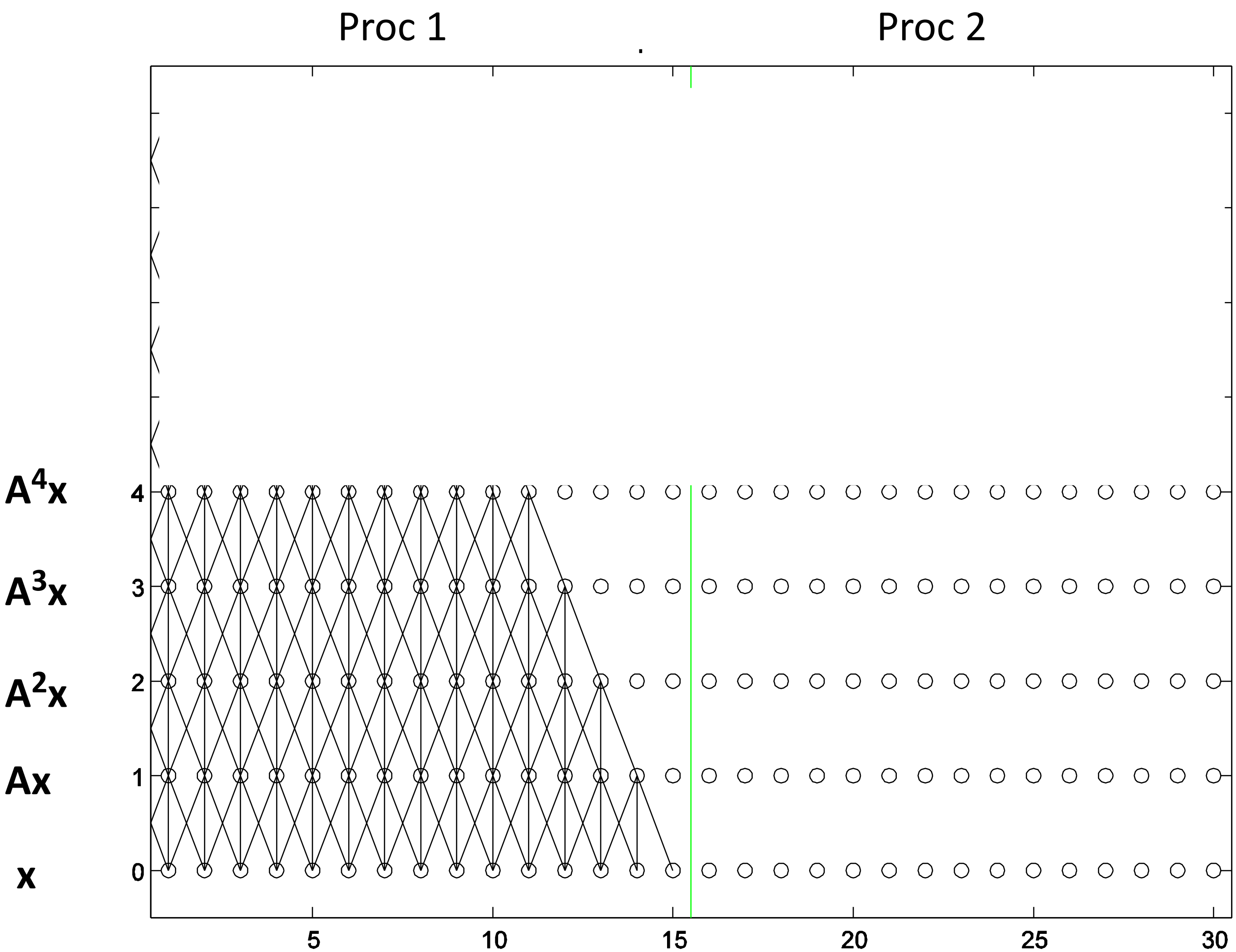
Can be computed without communication

Locally Dependent Entries for $[x, Ax, \ldots, A^3x]$, A tridiagonal, 2 processors



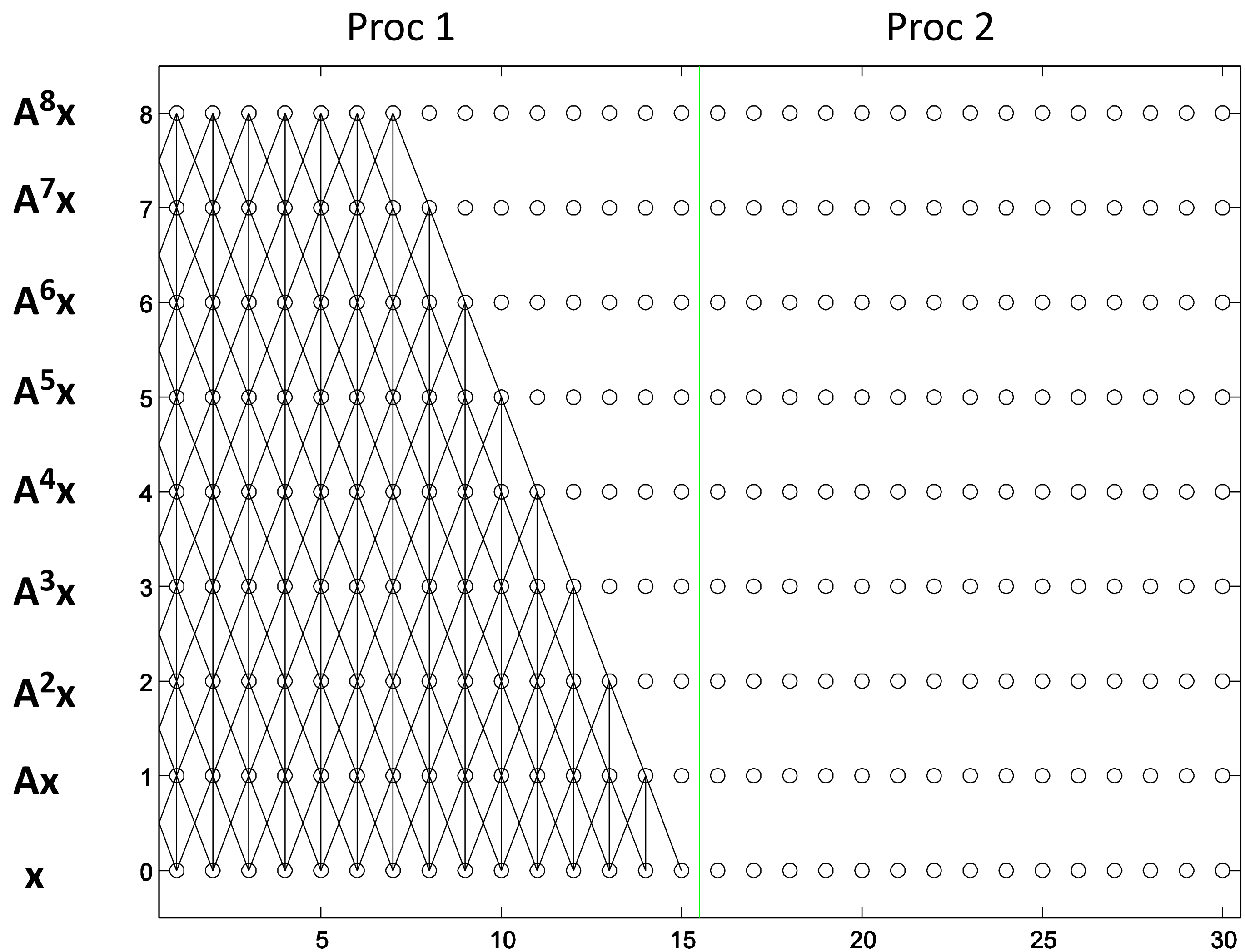
Can be computed without communication

Locally Dependent Entries for [x,Ax,…,A⁴x], A tridiagonal, 2 processors



Can be computed without communication

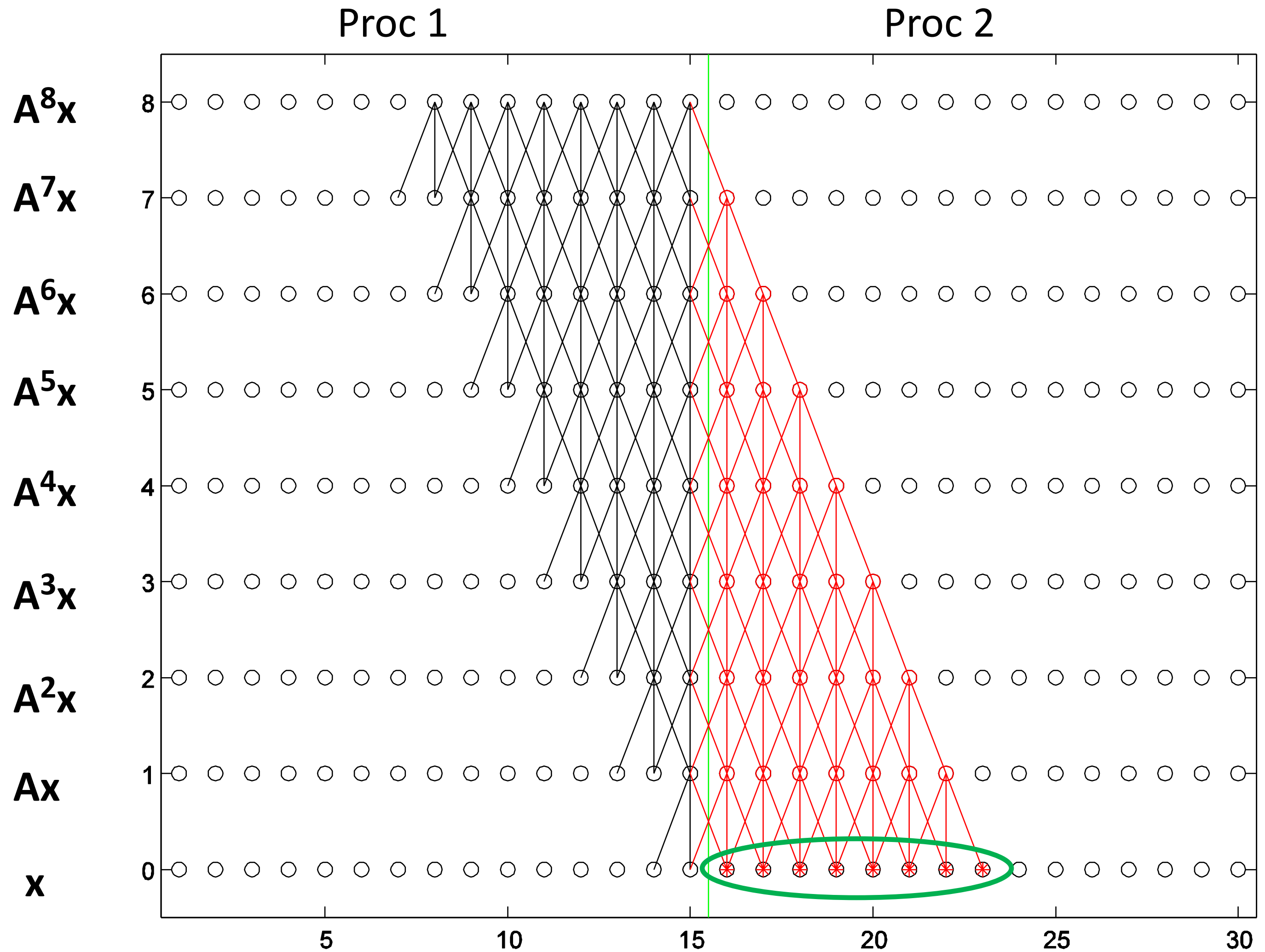
Locally Dependent Entries for [x,Ax,…,A⁸x], A tridiagonal, 2 processors



Can be computed without communication

k=8 fold reuse of A

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

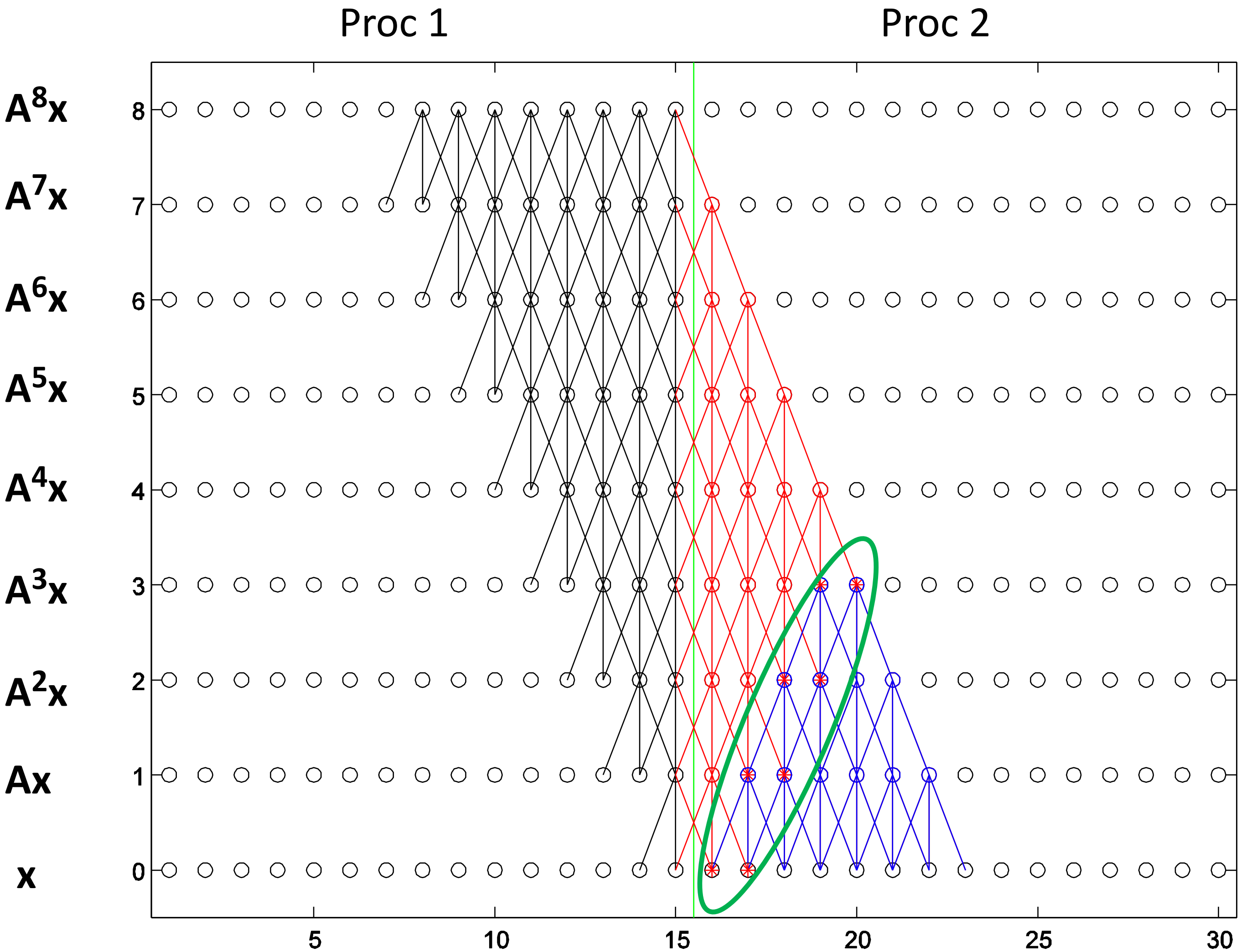


One message to get data needed to compute remotely dependent entries, ***not k=8***

Minimizes number of messages = latency cost

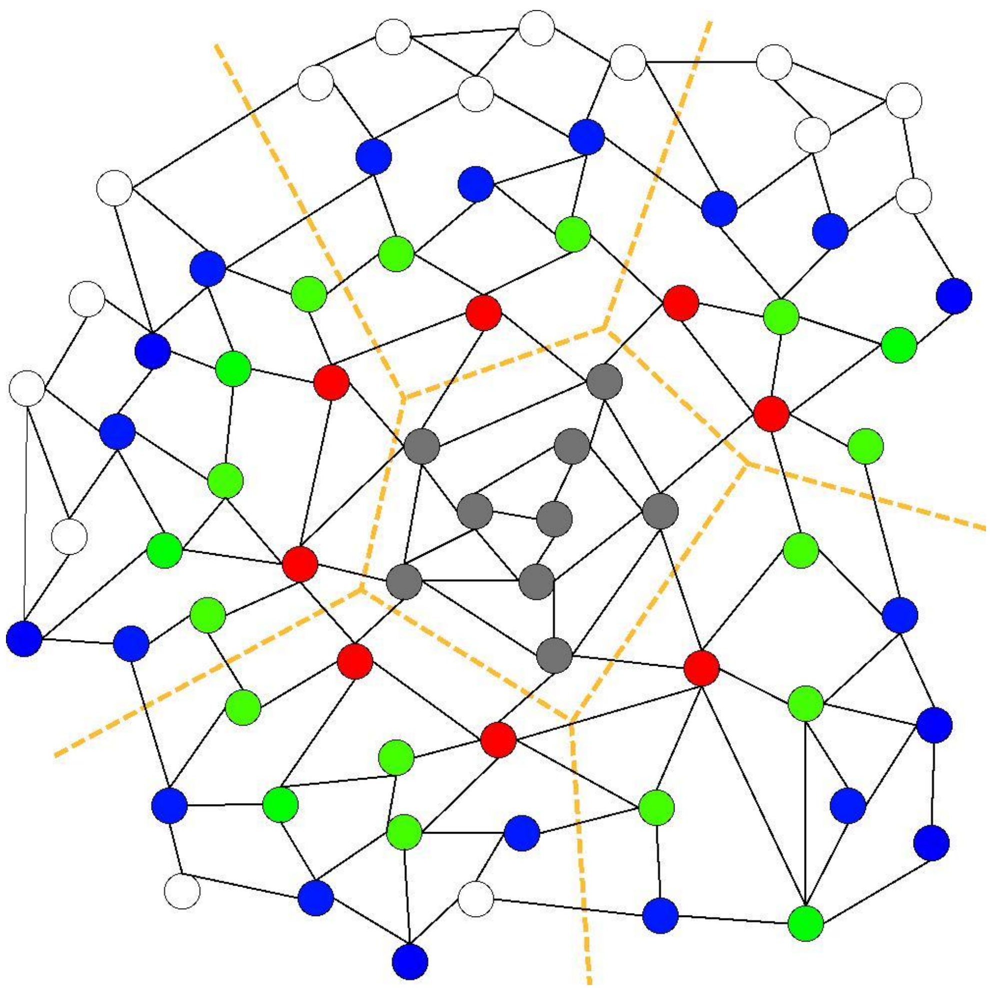
Price: redundant work ∝ “surface/volume ratio”

Fewer Remotely Dependent Entries for $[x, Ax, \ldots, A^8x]$, A tridiagonal, 2 processors

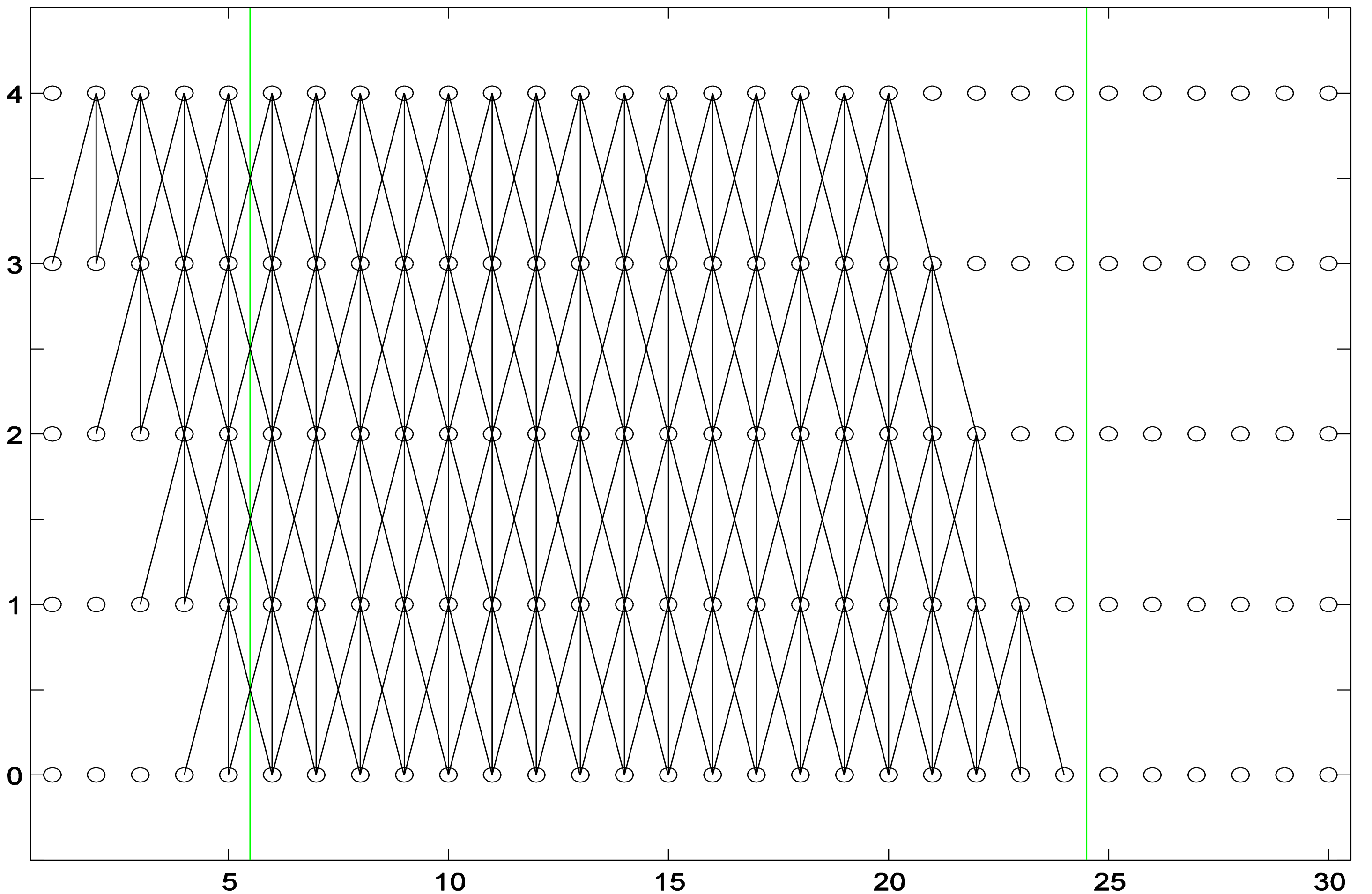


Reduce redundant work by half

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

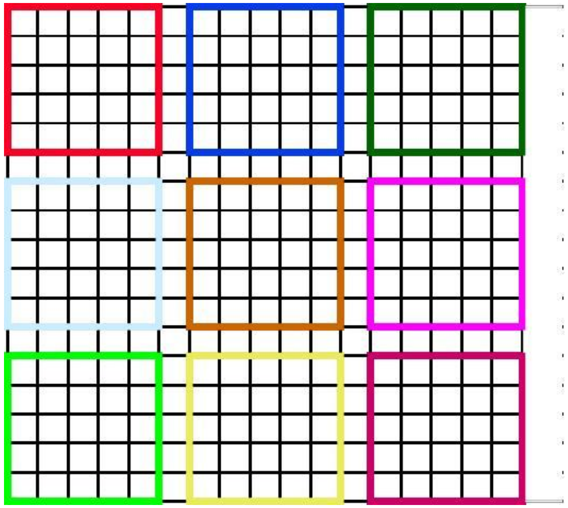
No redundant work

Performance Results

- Measured
  - Sequential/OOC speedup up to **3x**
- Modeled
  - Sequential/multicore speedup up to **2.5x**
  - Parallel/Petascale speedup up to **6.9x**
  - Parallel/Grid speedup up to **22x**
- See bebop.cs.berkeley.edu/#pubs

Optimizing Communication Complexity of Sparse Solvers

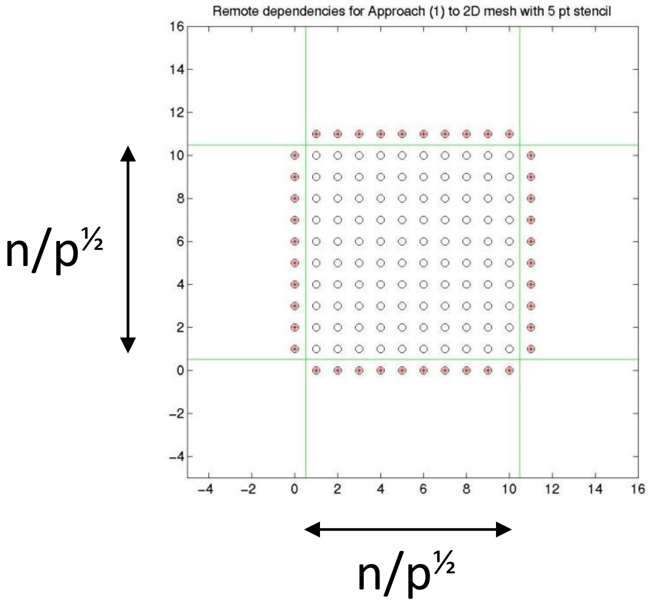
- Example: GMRES for $Ax=b$ on "2D Mesh"
  - x lives on n-by-n mesh
  - Partitioned on $p^{1/2}$ -by- $p^{1/2}$ grid
  - A has "5 point stencil" (Laplacian)
    - $(Ax)(i,j)$ = linear_combination($x(i,j)$, $x(i,j\pm 1)$, $x(i\pm 1,j)$)
  - Ex: 18-by-18 mesh on 3-by-3 grid



Minimizing Communication of GMRES

- What is the cost = (#flops, #words, #mess) of k steps of standard GMRES?

```
GMRES, ver.1:
  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
```



- Cost(A * v) = k * ($9n^2$ /p, 4n / $p^{½}$, 4)
- Cost(MGS) = $k^2/2$ * ($4n^2$ /p , log p , log p)
- Total cost ~ Cost(A * v) + Cost (MGS)
- Can we reduce the latency?

Minimizing Communication of GMRES

- Cost(GMRES, ver.1) = Cost($A*v$) + Cost(MGS)

  = ($9kn^2/p$, $4kn / p^{½}$, 4k) + ($2k^2n^2/p$, $k^2 \log p / 2$, $k^2 \log p / 2$)

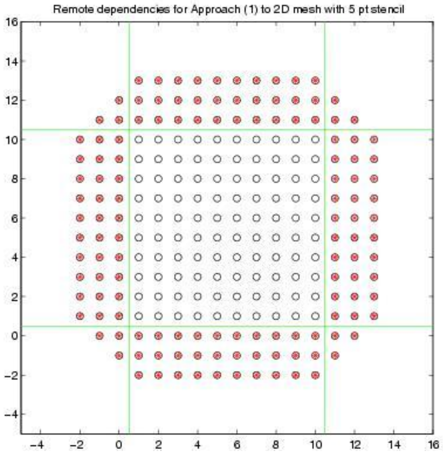
- How much latency cost from A*v can you avoid? Almost all

GMRES, ver. 2:

W = [v, Av, A^2v, … , A^kv]

[Q,R] = MGS(W)

Build H from R, solve LSQ problem



k = 3

- Cost(W) = (~ same, ~ same , 8)
  - Latency cost independent of k – **optimal**
- Cost (MGS) unchanged
- Can we reduce the latency more?

Minimizing Communication of GMRES

- Cost(GMRES, ver. 2) = Cost(W) + Cost(MGS)

  $= (9kn^2/p,\ 4kn / p^{1/2},\ 8) + (2k^2n^2/p,\ k^2 \log p / 2,\ k^2 \log p / 2)$

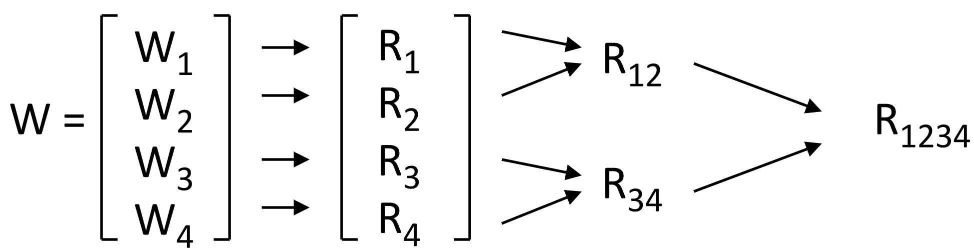
- How much latency cost from MGS can you avoid? Almost all

GMRES, ver. 3:

$W = [\ v, Av, A^2v, \ldots, A^kv\ ]$

$[Q,R] = \mathrm{TSQR}(W)$ … "Tall Skinny QR"

Build H from R, solve LSQ problem



- Cost(TSQR) = (~ same, ~ same , $\log p$)
  - Latency cost independent of s - **optimal**

Minimizing Communication of GMRES

- Cost(GMRES, ver. 2) = Cost(W) + Cost(MGS)

  $= (9kn^2/p,\ 4kn / p^{½} ,\ 8) + (2k^2n^2/p ,\ k^2 \log p / 2 ,\ k^2 \log p / 2)$

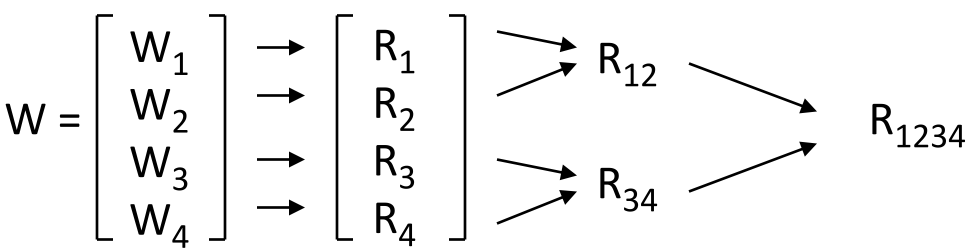
- How much latency cost from MGS can you avoid? Almost all

GMRES, ver. 3:

$W = [v, Av, A^2v, \ldots , A^kv]$

$[Q,R] = \mathrm{TSQR}(W)$ … "Tall Skinny QR"

Build H from R, solve LSQ problem



- Cost(TSQR) = (~ same, ~ same , log p)
- **Oops**

Minimizing Communication of GMRES

- Cost(GMRES, ver. 2) = Cost(*W*) + Cost(MGS)

  $= (9kn^2/p,\ 4kn / p^{1/2},\ 8) + (2k^2n^2/p,\ k^2 \log p / 2,\ k^2 \log p / 2)$

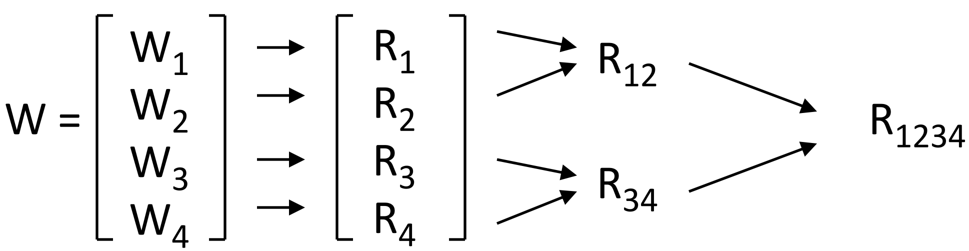
- How much latency cost from MGS can you avoid? Almost all

GMRES, ver. 3:

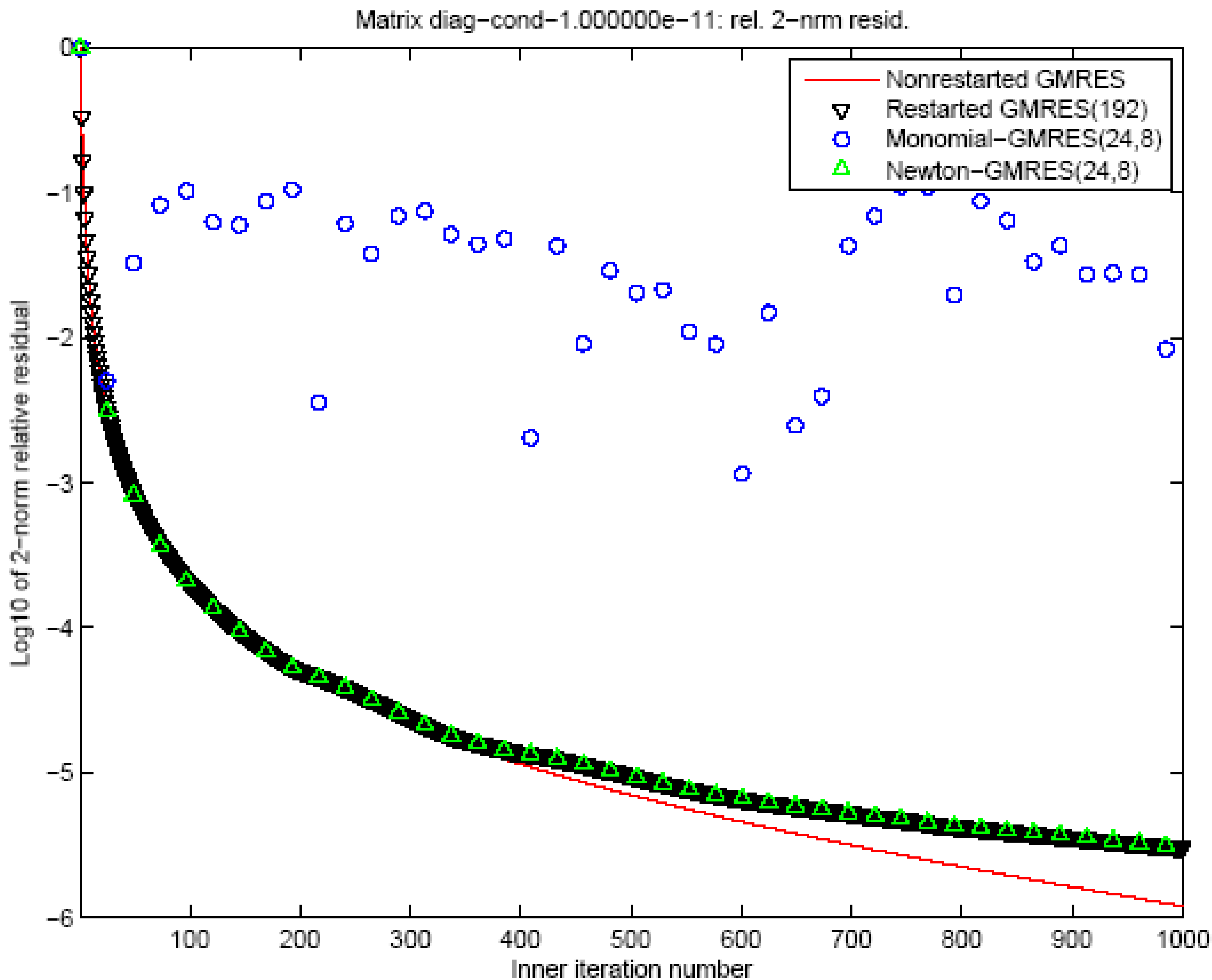
$W = [\ v, Av, A^2v, \ldots, A^kv\ ]$

$[Q,R] = \mathrm{TSQR}(W)$ … "Tall Skinny QR"

Build H from R, solve LSQ problem



- Cost(TSQR) = (~ same, ~ same , log p)
- **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

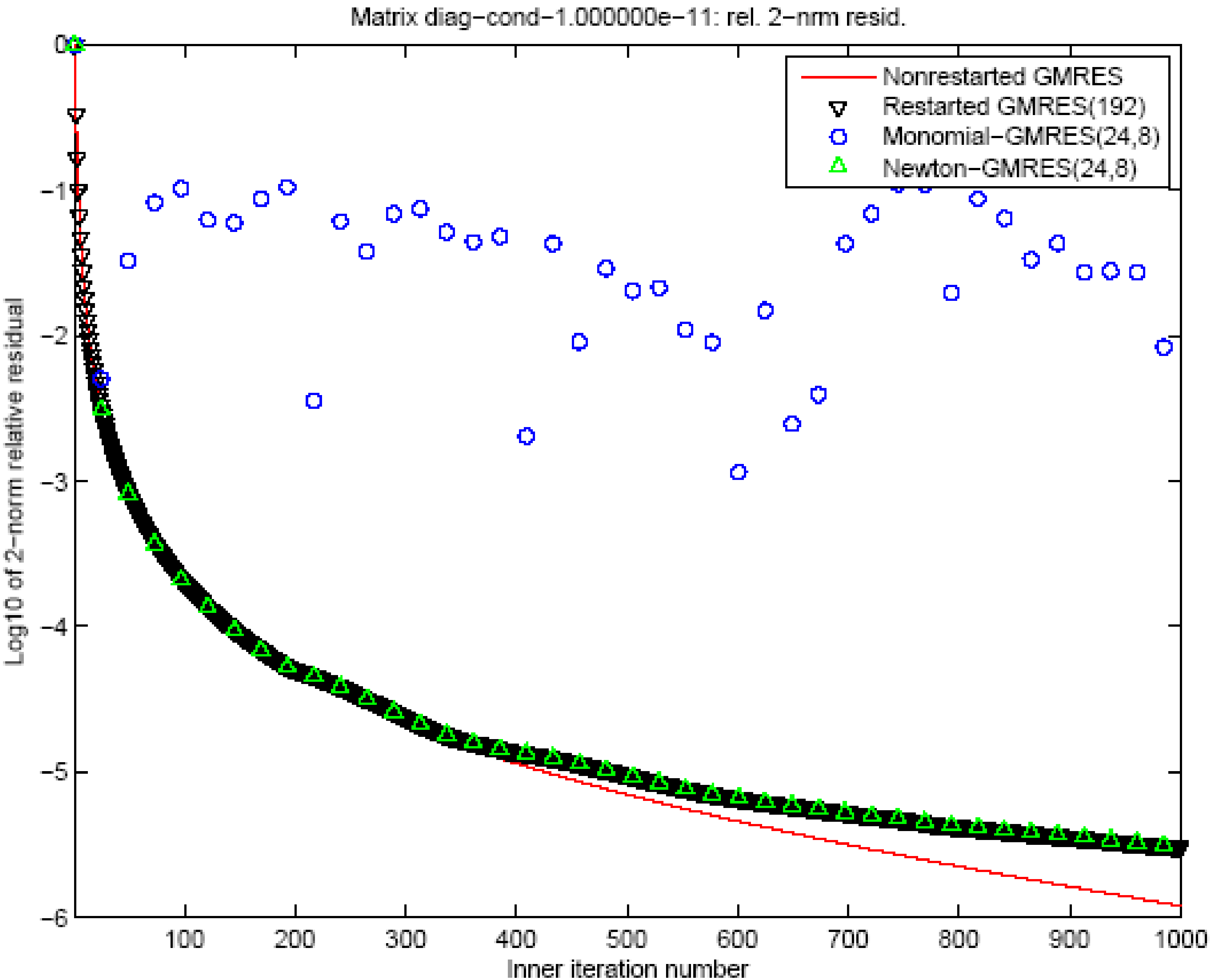
Minimizing Communication of GMRES

- Cost(GMRES, ver. 3) = Cost(W) + Cost(TSQR)

  $= (9kn^2 /p, 4kn / p^{½} , 8) + (2k^2n^2 /p , k^2 \log p / 2 , \log p)$

- Latency cost independent of k, just log p – optimal
- Oops – W from power method, so precision lost – What to do?

- Use a different polynomial basis
- Not Monomial basis $W = [v, Av, A^2v, \ldots]$, instead …
- Newton Basis $W_N = [v, (A - \theta_1 I)v , (A - \theta_2 I)(A - \theta_1 I)v, \ldots]$ or
- Chebyshev Basis $W_C = [v, T_1(v), T_2(v), \ldots]$



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

Summary and Conclusions (1/2)

- Possible to minimize communication complexity of much dense and sparse linear algebra
  - Practical speedups
  - Approaching theoretical lower bounds
- Optimal asymptotic complexity algorithms for dense linear algebra – also lower communication
- Hardware trends mean the time has come to do this
- *Lots* of prior work (see pubs) – and some new

Summary and Conclusions (2/2)

- Many open problems
  - Automatic tuning - build and optimize complicated data structures, communication patterns, code automatically: bebop.cs.berkeley.edu
  - Extend optimality proofs to general architectures
  - Dense eigenvalue problems – SBR or spectral D&C?
  - Sparse direct solvers – CALU or SuperLU?
  - Which preconditioners work?
  - Why stop at linear algebra?