Rethinking Algorithms for Future Architectures: Communication-Avoiding Algorithms

Jim Demmel

EECS & Math Departments

UC Berkeley

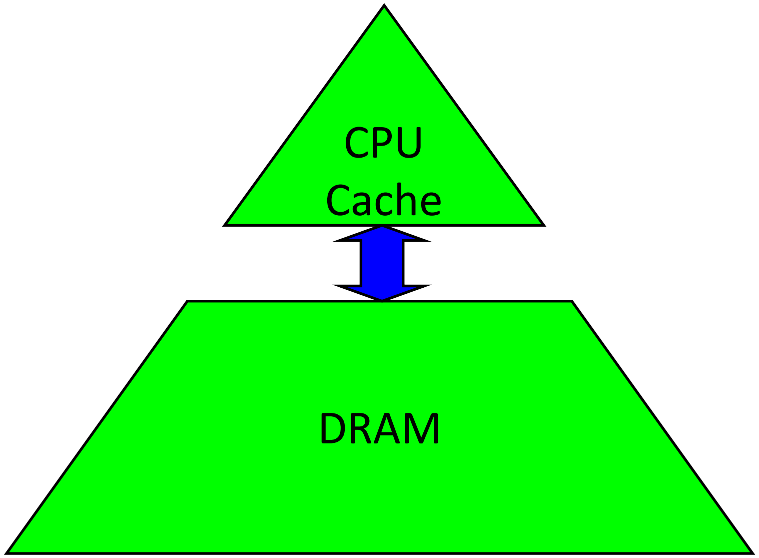
Collaborators and Supporters

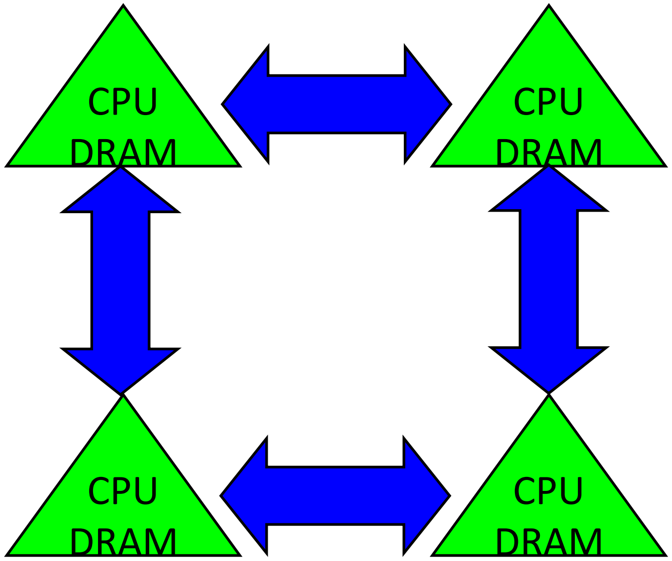
- Collaborators
  - Katherine Yelick (UCB & LBNL) Michael Anderson (UCB), Grey Ballard (UCB), Jong-Ho Byun (UCB), Erin Carson (UCB), Jack Dongarra (UTK), Ioana Dumitriu (U. Wash), Laura Grigori (INRIA), Ming Gu (UCB), Mark Hoemmen (Sandia NL), Olga Holtz (UCB & TU Berlin),   Kurt Keutzer (UCB), Nick Knight (UCB), Julien Langou, (U Colo. Denver), Marghoob Mohiyuddin (UCB), Hong Diep Nguyen (UCB), Oded Schwartz (TU Berlin), Edgar Solomonik (UCB), Michelle Strout (Colo. SU), Vasily Volkov (UCB), Sam Williams (LBNL), Hua Xiang (INRIA)
  - 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

Why avoid communication? (1/2)

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? (2/2)

- 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]

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

- Goal : reorganize algorithms to *avoid* communication
  - Between all memory hierarchy levels
    - L1⟷ L2⟷ DRAM ⟷ network, etc
  - Very large speedups possible
  - Energy savings too!

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

- “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
  - Implications for architectural scaling
    - How flop rate, bandwidths, latencies, memory sizes need to scale to maintain balance
- Ditto for “Iterative” Linear Algebra

Lower bound for all "direct" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul

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 ≥ #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 “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
  - 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)

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 much of dense linear algebra
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations
- Only a few sparse algorithms so far
- Lots of work in progress…

TSQR: QR of a Tall, Skinny matrix

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

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

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

TSQR: QR of a Tall, Skinny matrix

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

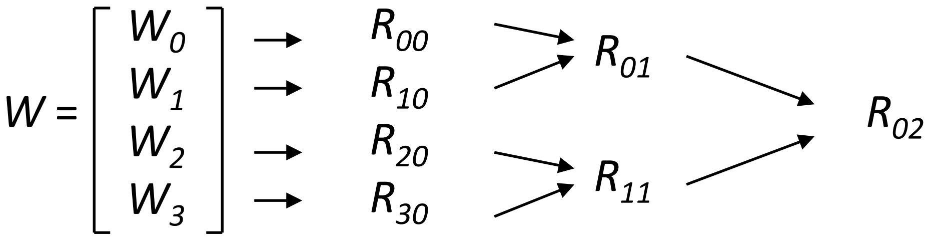
$$\left(\begin{array}{c} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{array}\right) = \left(\begin{array}{c} Q_{01}\,R_{01} \\ \hline Q_{11}\,R_{11} \end{array}\right) = \left(\begin{array}{cc} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot \left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right)$$

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

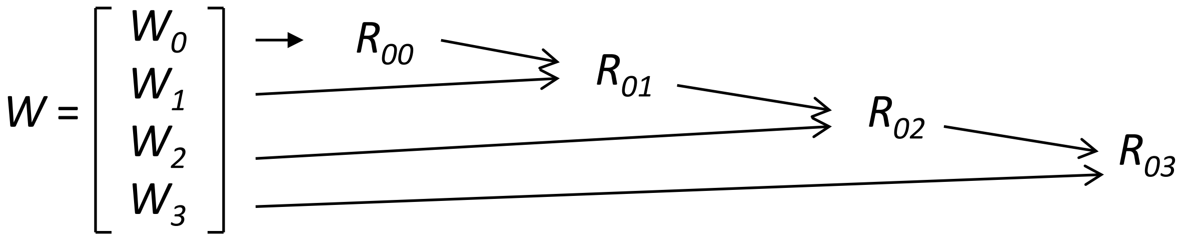
Output = $\{ Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} \}$

TSQR: An Architecture-Dependent Algorithm

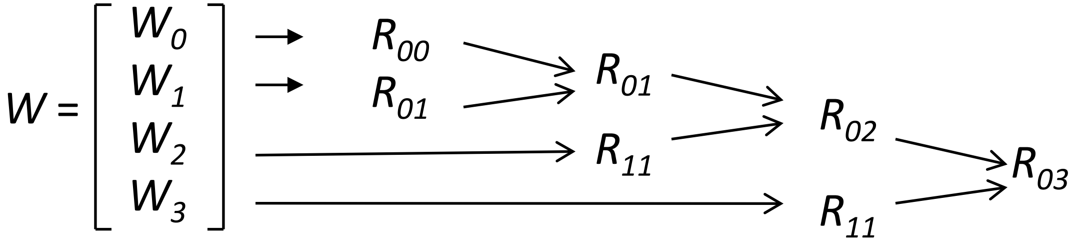
Parallel:



Sequential:



Dual Core:



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

Can choose reduction tree dynamically

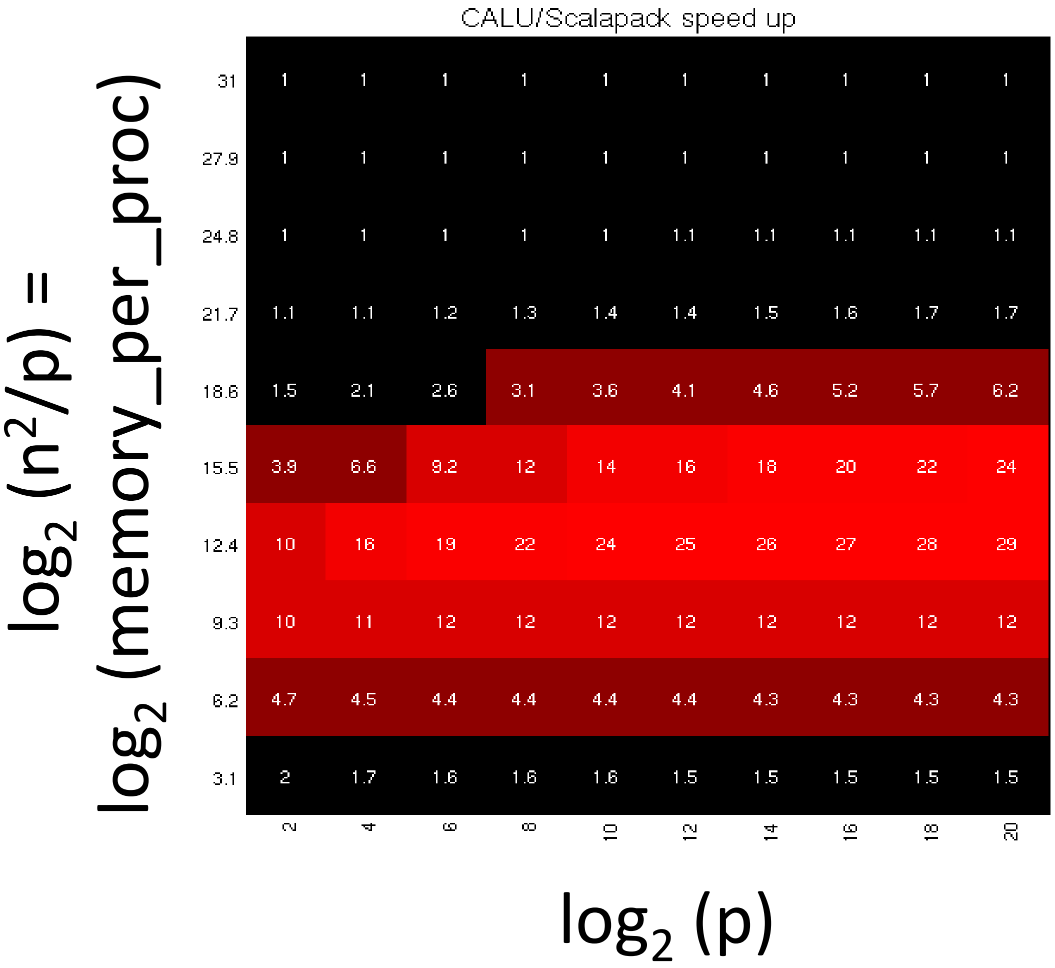
TSQR Performance Results

- 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 (Dongarra et al)
  - Cloud – early result – up and running
- 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

Exascale Machine Parameters Source: DOE Exascale Workshop

- 2^20 ≈ 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

Exascale predicted speedups for Gaussian Elimination: CA-LU vs ScaLAPACK-LU



	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

Are we using all the hardware resources?

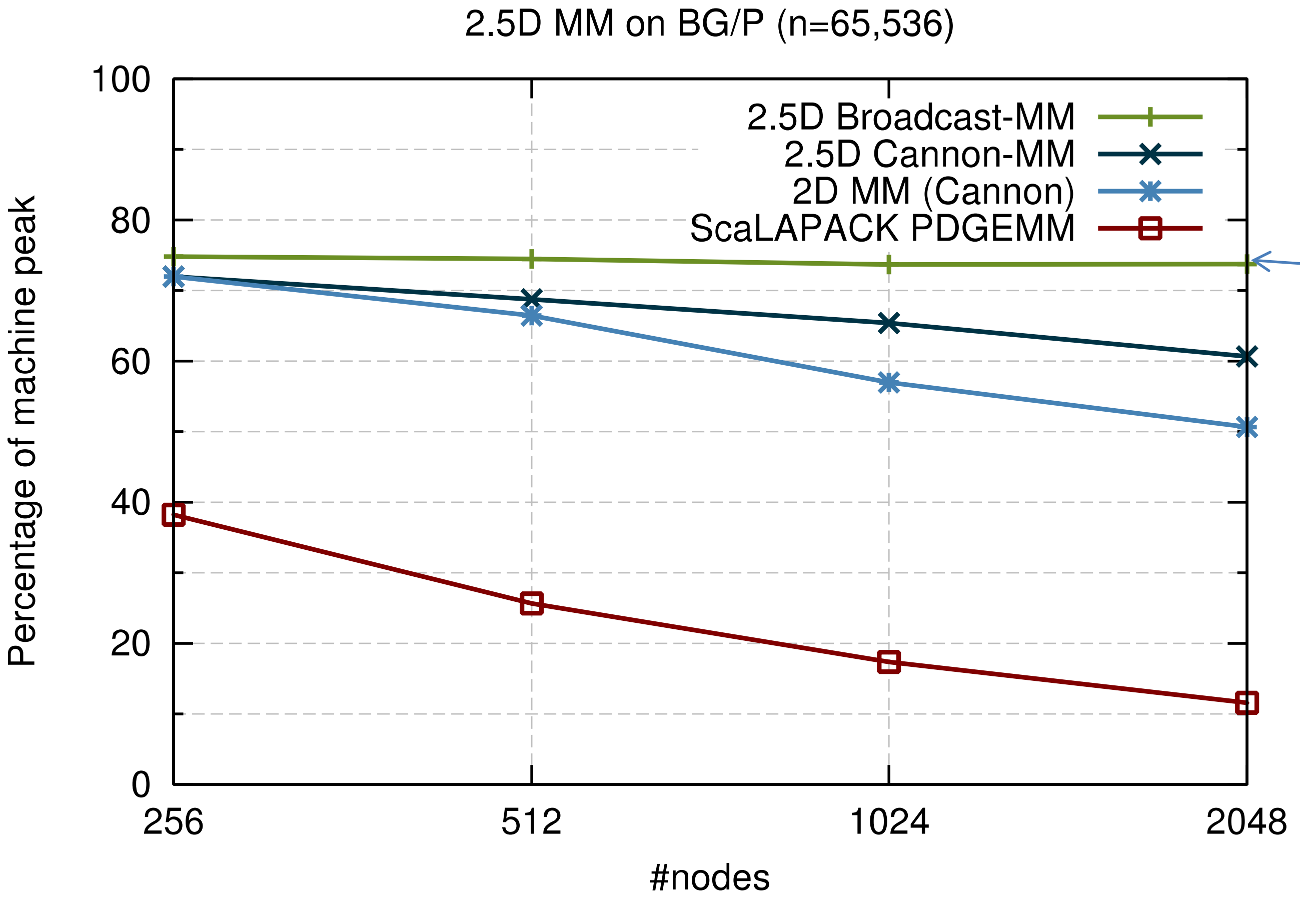
- Assume nxn dense matrices on P processors
- Usual approach
  - 1 copy of data $\Rightarrow$ Memory per processor = $M \approx n^2 / P$
  - Recall lower bounds:

    #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

    #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$
  - Attained by **2D algorithms** (many examples)
    - P processors connected in $P^{1/2}$ x $P^{1/2}$ mesh
    - Each processor owns, computes on a square submatrix
- New approach
  - **Use all available memory**
  - c>1 copies of data $\Rightarrow$ Memory per processor = $M \approx c\, n^2 / P$
  - Lower bounds get smaller
  - New **2.5D algorithms** can attain new lower bounds
    - P processors in $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c mesh

2.5D Matmul versus ScaLAPACK

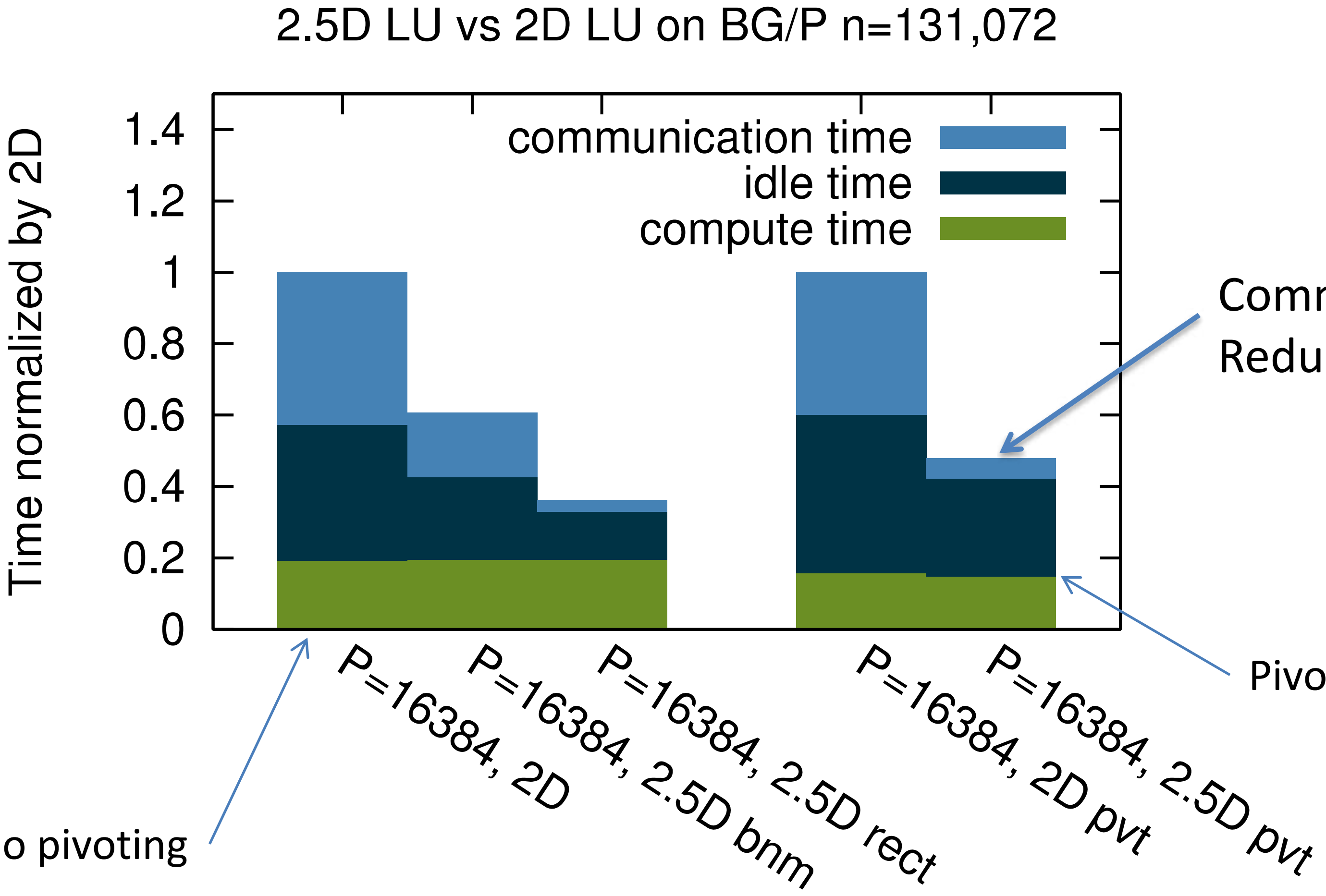
- 2D algorithms use $P^{1/2}$ x $P^{1/2}$ mesh and minimal memory
- 2.5D algorithms use $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x $c^{1/2}$ mesh and c-fold memory
  - Matmul sends $c^{1/2}$ times fewer words – lower bound
  - Matmul sends $c^{3/2}$ times fewer messages – lower bound



Perfect Strong Scaling

Critical to use all links of BG/P's 3D torus interconnect

Timing Breakdown for 2D vs 2.5D Gaussian Elimination: How much communication can we avoid?



Communication Reduced 86%

Pivoting

No pivoting

Distinguished Paper Award, EuroPar'11

Implications for Architectural Scaling

- Machine parameters:
  - γ = seconds per flop (multiply or add)
  - β = reciprocal bandwidth (in seconds)
  - α = latency (in seconds)
  - M = local (fast) memory size
  - P = number of processors
- Goal: relationships among these parameters that guarantees that communication is not the bottleneck for direct linear algebra

Implications for Architectural Scaling Sequential Case:

- Requirements so that "most" time is spent doing arithmetic on n x n dense matrices, $n^2 > M$
  - $\gamma M^{1/2} > \approx \beta$ $\gamma M^{1/3} > \approx \beta$ for old algorithms
    - In other words, time to add two rows of largest locally storable square matrix exceeds reciprocal bandwidth
  - $\gamma M^{3/2} > \approx \alpha$ $\gamma M > \approx \alpha$ for old algorithms
    - In other words, time to multiply 2 largest locally storable square matrices exceeds latency
- Applies to *every* level of memory hierarchy

Stricter requirements on architecture for old algorithms

Implications for Architectural Scaling Parallel Case:

- Requirements so that "most" time is spent doing arithmetic on n x n dense matrices
  - $\gamma\,(n/p^{1/2}) > \approx\ \beta$ $\gamma\, M^{1/2} > \approx \beta$
    - In other words, time to add two rows of locally stored square matrix exceeds reciprocal bandwidth
  - $\gamma\,(n/p^{1/2})^3 > \approx \alpha$ $\gamma\,(n/p^{1/2})^2 > \approx \alpha$ $\gamma\, M^{3/2} > \approx \alpha$
    - In other words, time to multiply 2 locally stored square matrices exceeds latency

Stricter requirements on architecture for old algorithms

Looser requirements on architecture for 2.5D algorithms

Summary of Direct Linear Algebra

- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - Heterogeneous architectures
    - Extends to case where each processor and link has a different speed (SPAA'11)
  - More dense and sparse algorithms done, underway
  - Extensions to Strassen-like algorithms
    - **Best Paper Award, SPAA'11**
  - Need 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"
- 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

Minimizing Communication of GMRES to solve Ax=b

- GMRES: find x in span{b,Ab,…,A^kb} minimizing || Ax-b $||_2$

Standard GMRES

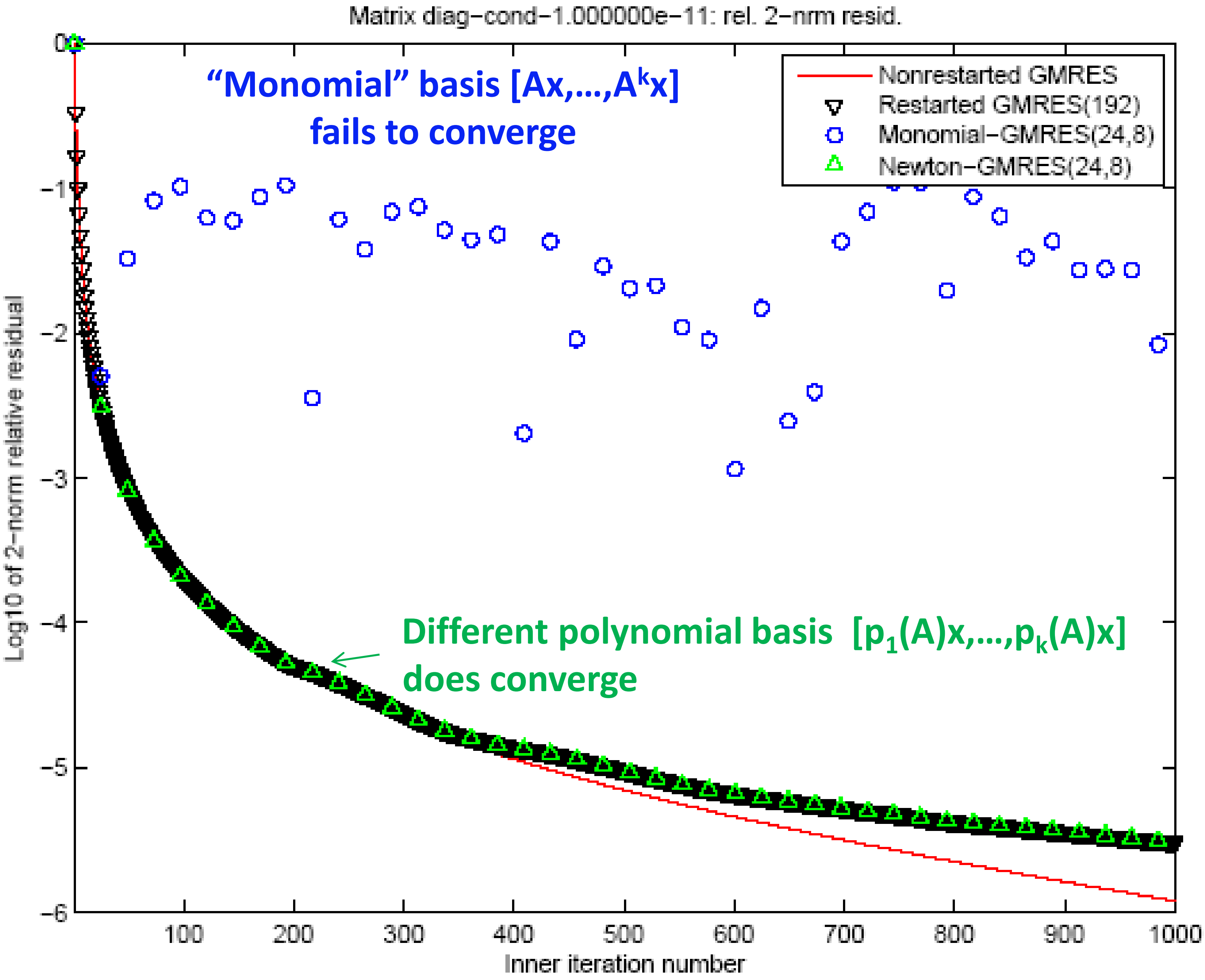
```
for i=1 to k
   w = A · v(i-1)   … SpMV
   MGS(w, v(0),…,v(i-1))
   update v(i), H
endfor
solve LSQ problem with H
```

Communication-avoiding GMRES

```
W = [ v, Av, $A^2$v, … , $A^k$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!



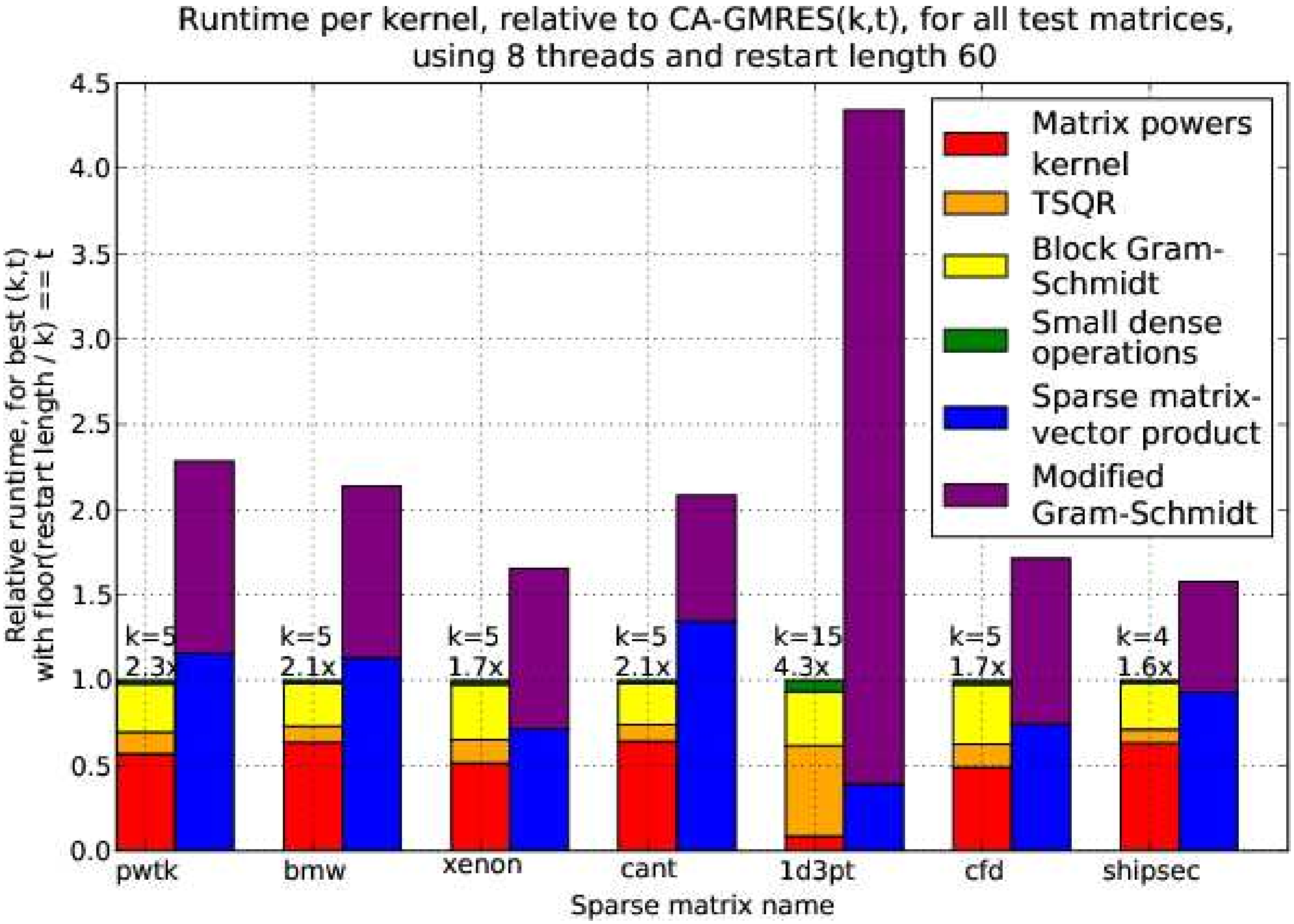
“Monomial” basis [Ax,…,A^{k}x] fails to converge

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

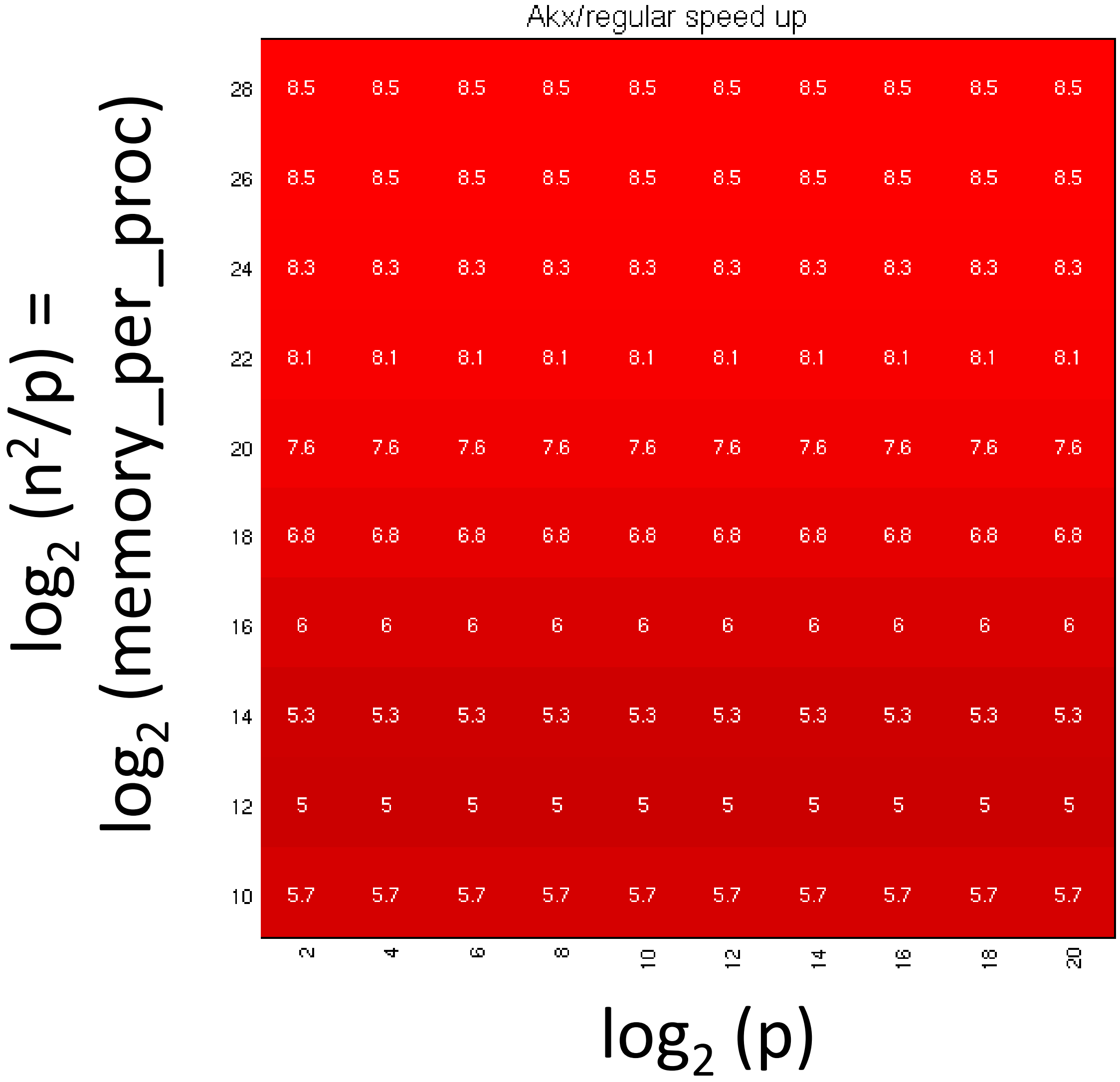
Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



Exascale predicted speedups for Matrix 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
  - Many different algorithms reorganized
    - More underway
  - Architectural scaling rules (as for direct case)
    - Sparse matrices $\Rightarrow$ stricter conditions for scaling
  - Need to recognize stable variants more easily
  - Need 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
- Google “parallel computing course”

Summary

Time to redesign all linear algebra algorithms and software

And eventually the rest of applied mathematics

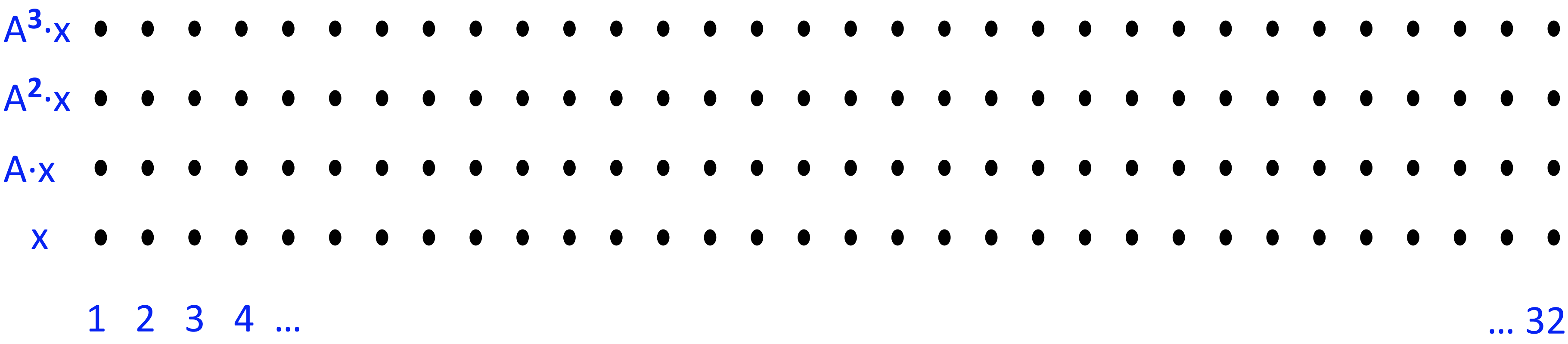
Don’t Communic…

EXTRA SLIDES

Communication Avoiding Kernels:

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

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

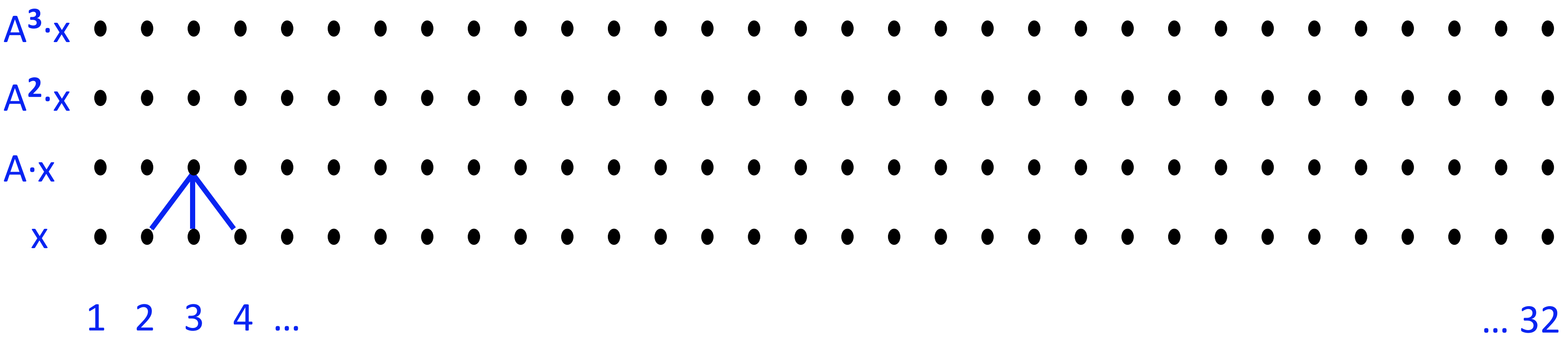


- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:

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

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

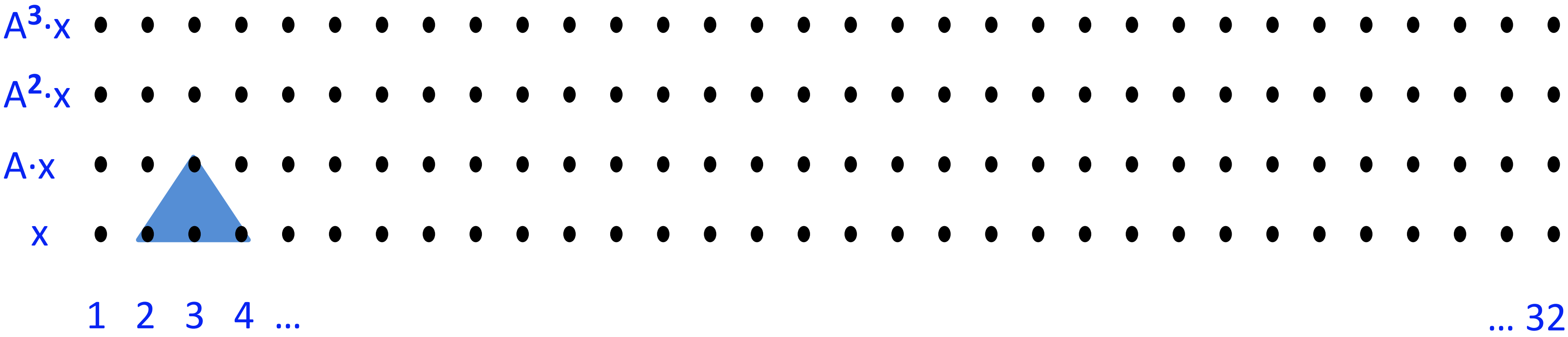


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

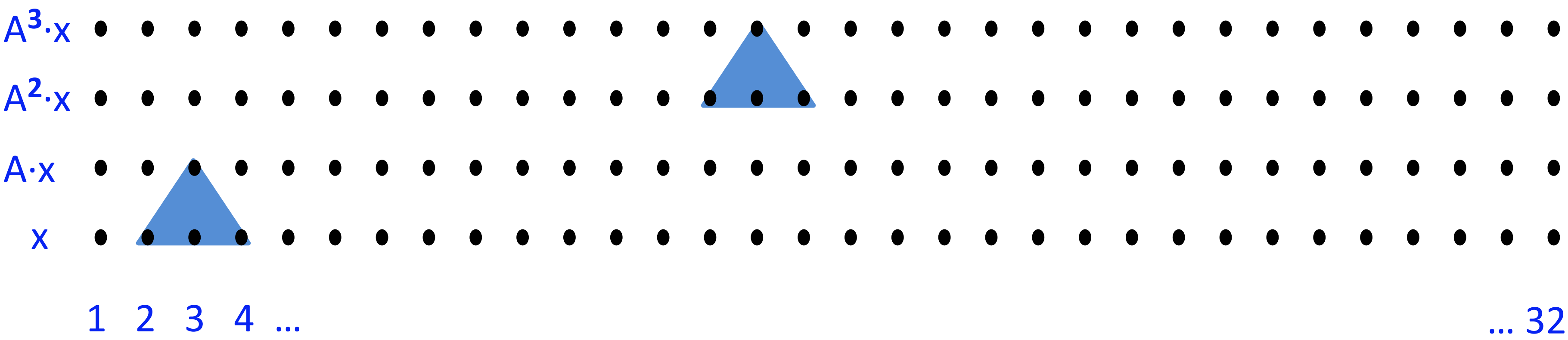


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

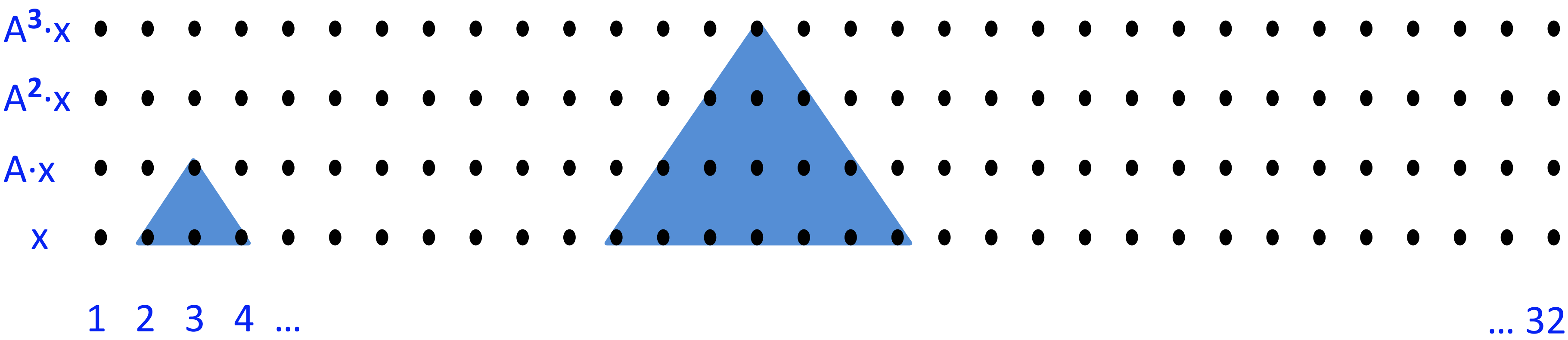


- Example: A tridiagonal, n=32, k=3

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]

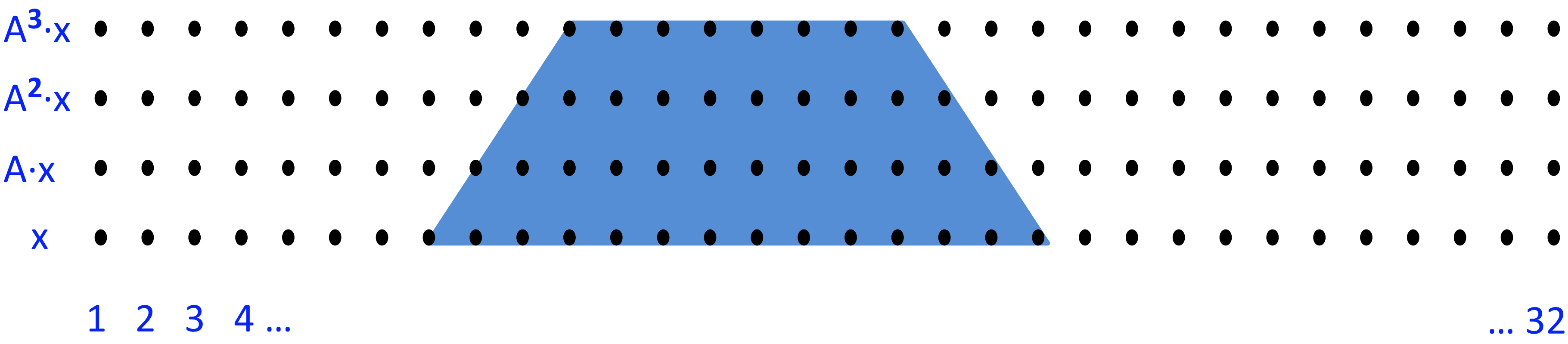


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

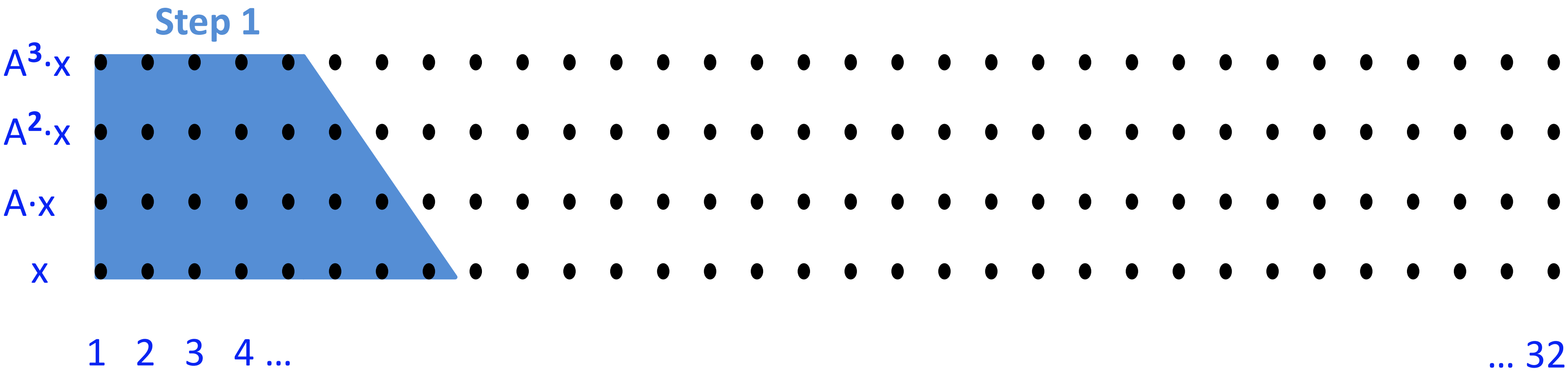


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

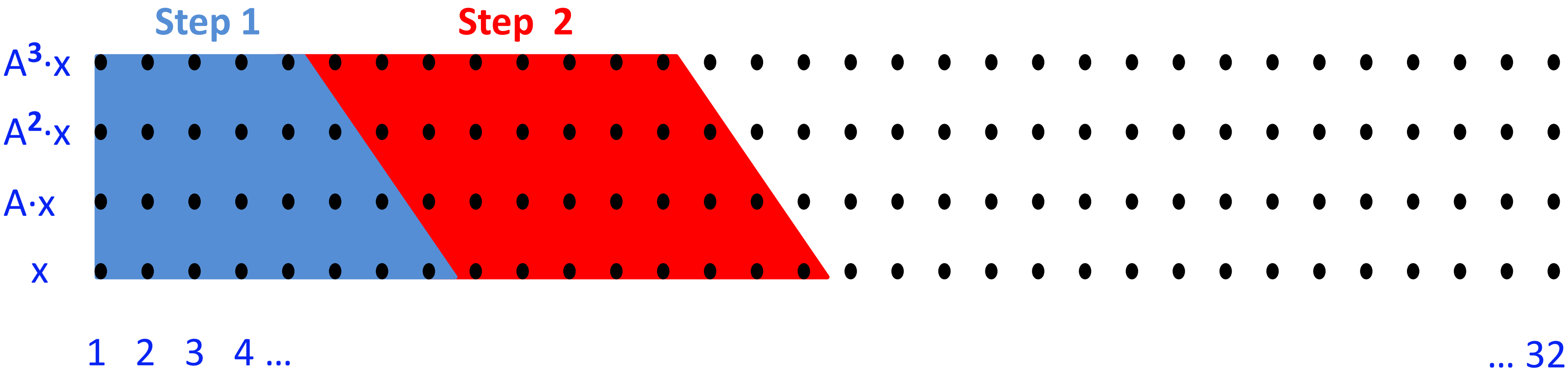
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



- 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

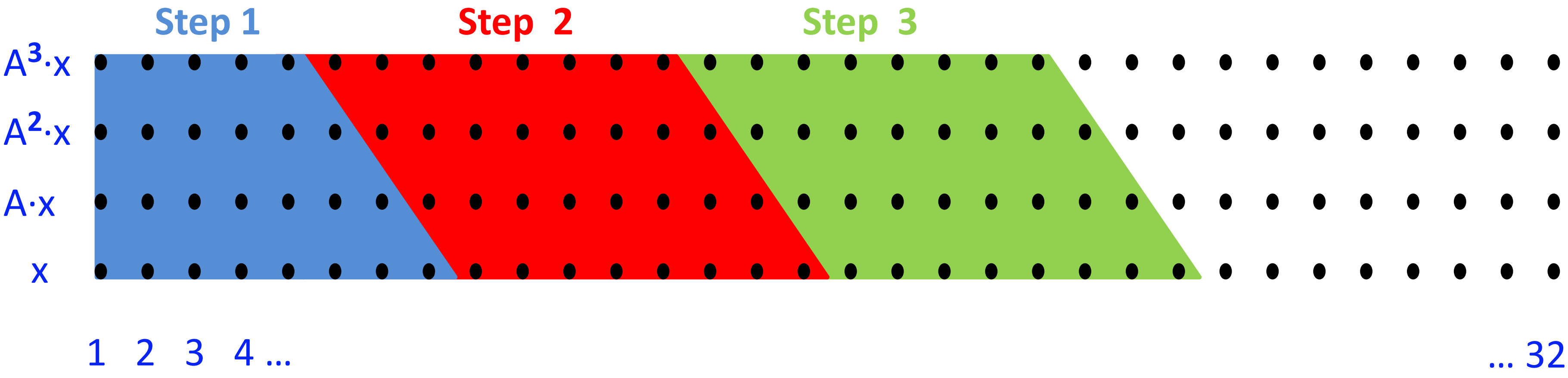


- Example: A tridiagonal, n=32, k=3

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

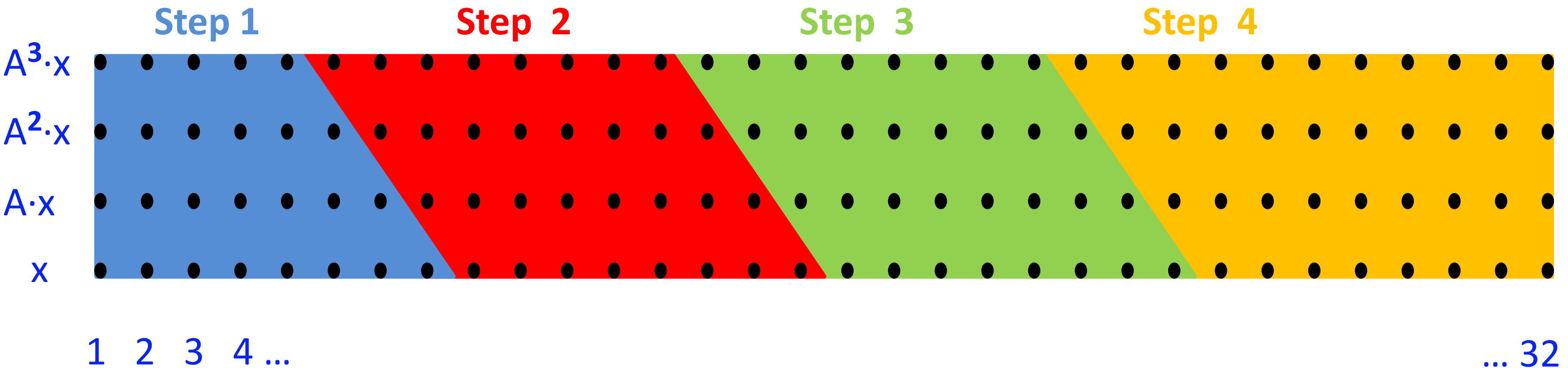


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

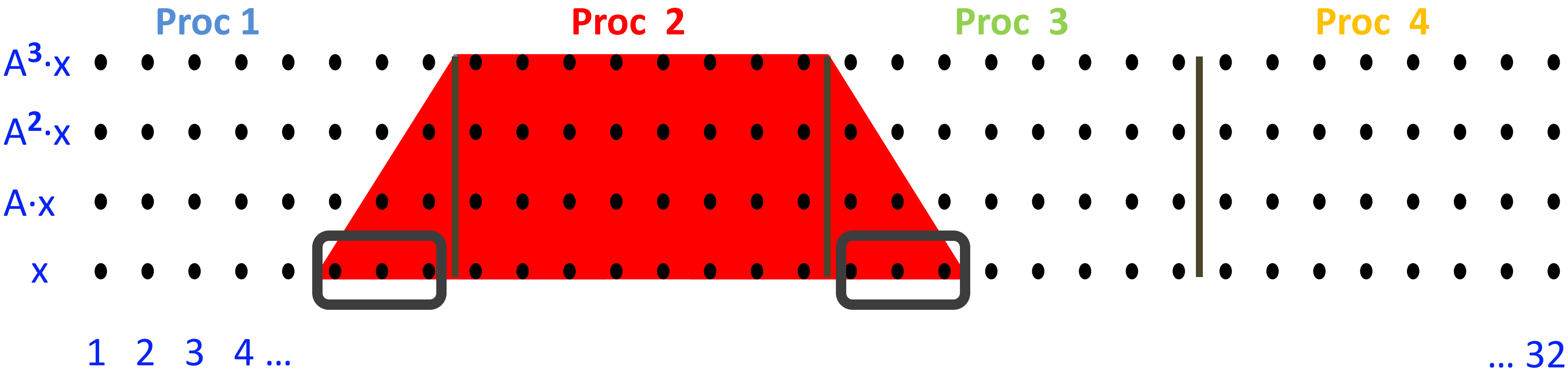


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

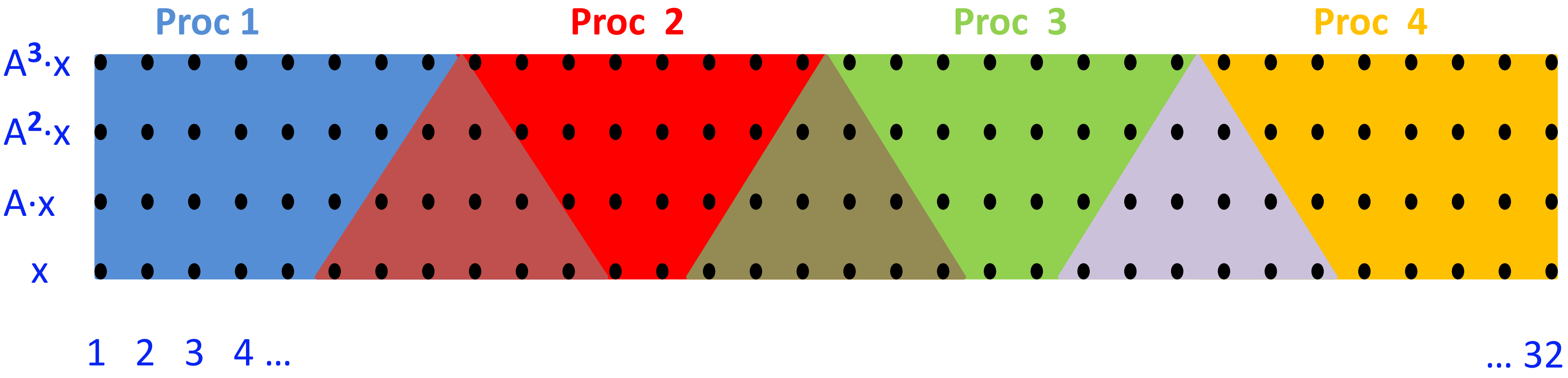


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

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

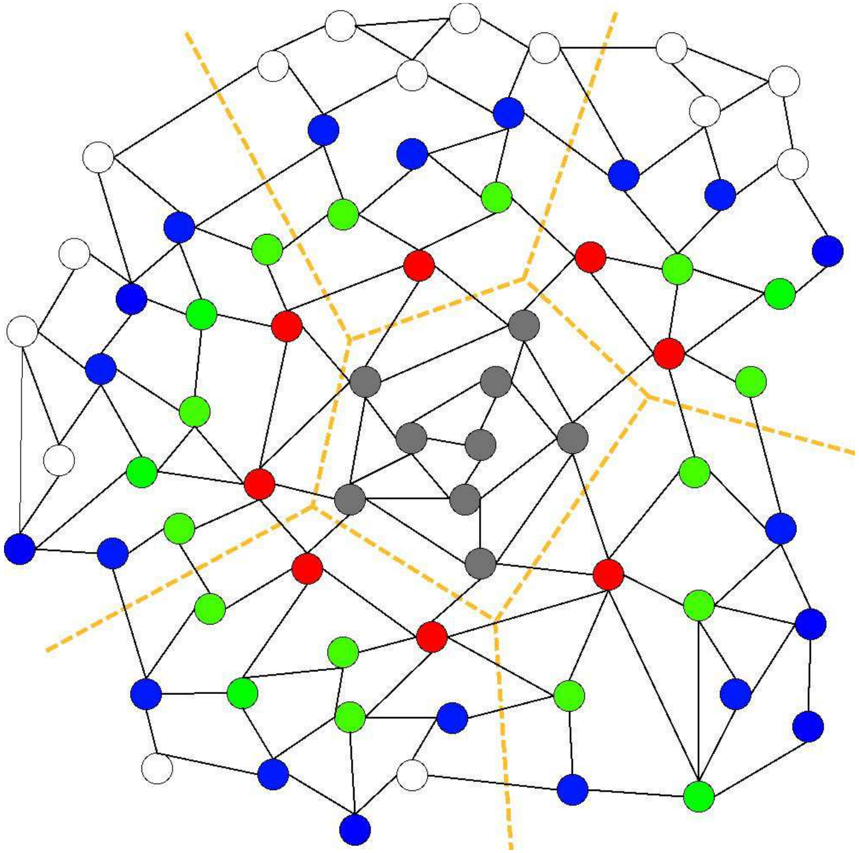


- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

Communication Avoiding Kernels:

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

Same idea works for general sparse matrices



We’re hiring!

- Seeking a postdoc to help develop the next versions of LAPACK and ScaLAPACK

7 Dwarfs of High Performance Computing (HPC)

Structured Grid

Dense Matrix

Sparse Matrix

Spectral (FFT)

Particle Methods

Unstructured Grid

Monte Carlo

7 Dwarfs – Are they enough?

	Embed	SPEC	DB	Games	ML	CAD	HPC
Structured Grid							
Dense Matrix							
Sparse Matrix							
Spectral (FFT)							
Particle Methods							
Unstructured Grid							
Monte Carlo							

13 Motifs (nee "Dwarf") of Parallel Computing

Popularity: (Red Hot / Blue Cool)

	Embed	SPEC	DB	Games	ML	CAD	HPC
Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟨	🟦
Circuits	🟥	🟦	🟩	🟦	🟩	🟦	🟦
Graph Algorithms	🟥	🟨	🟨	🟨	🟥	🟥	🟦
Structured Grid	🟥	🟥	🟦	🟨	🟦	🟦	🟥
Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	🟥
Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥	🟥
Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟨	🟥
Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟥	🟦
Particle Methods	🟦	🟨	🟦	🟨	🟦	🟦	🟥
Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟥	🟦
Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦	🟦
Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟨	🟥
Monte Carlo	🟦	🟩	🟥	🟦	🟥	🟦	🟥

Motifs in ParLab Applications

(Red Hot / Blue Cool)

		Embed	SPEC	DB	Games	ML	CAD	HPC	Health	Image	Speech	Music	Browser
1	Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟨	🟦	🟦	🟦	🟦	🟦	🟥
2	Circuits	🟥	🟦	🟩	🟦	🟩	🟦	🟦	🟦	🟦	🟦	🟦	🟥
3	Graph Algorithms	🟥	🟨	🟨	🟨	🟥	🟥	🟦	🟥	🟦	🟥	🟩	🟩
4	Structured Grid	🟥	🟥	🟦	🟨	🟦	🟦	🟥	🟦	🟥	🟦	🟦	🟦
5	Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	🟥	🟦	🟥	🟥	🟥	🟦
6	Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥	🟥	🟥	🟦	🟦	🟥	🟦
7	Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟨	🟥	🟦	🟩	🟥	🟥	🟥
8	Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟥	🟦	🟦	🟦	🟨	🟦	🟥
9	Particle Methods	🟦	🟨	🟦	🟨	🟦	🟦	🟥	🟦	🟦	🟦	🟦	🟦
10	Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟥	🟦	🟦	🟦	🟦	🟨	🟦
11	Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟦	🟥	🟦
12	Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟨	🟥	🟥	🟦	🟦	🟥	🟦

- What happened to Monte Carlo?

One-sided Factorizations (LU, QR)

- 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 or works in parallel
- Need another idea

Communication-Avoiding LU: Use reduction tree, to do “Tournament Pivoting”

$$W^{nxb} = \left(\begin{array}{c} W_1 \\ \hline W_2 \\ \hline W_3 \\ \hline W_4 \end{array}\right) = \left(\begin{array}{c} P_1\cdot L_1\cdot U_1 \\ \hline P_2\cdot L_2\cdot U_2 \\ \hline P_3\cdot L_3\cdot U_3 \\ \hline P_4\cdot L_4\cdot U_4 \end{array}\right)$$

Choose b pivot rows of W_1, call them W_1’
Ditto for W_2, yielding W_2’
Ditto for W_3, yielding W_3’
Ditto for W_4, yielding W_4’

$$\left(\begin{array}{c} W_1' \\ W_2' \\ \hline W_3' \\ W_4' \end{array}\right) = \left(\begin{array}{c} P_{12}\cdot L_{12}\cdot U_{12} \\ \hline P_{34}\cdot L_{34}\cdot U_{34} \end{array}\right)$$

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

Ditto, yielding 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)

Collaborators

- Katherine Yelick, Michael Anderson, Grey Ballard, Erin Carson, Ioana Dumitriu, Laura Grigori, Mark Hoemmen, Olga Holtz, Kurt Keutzer, Nicholas Knight, Julien Langou, Marghoob Mohiyuddin, Oded Schwartz, Edgar Solomonik, Vasily Volkok, Sam Williams, Hua Xiang

Can we do even better?

- Assume nxn matrices on P processors
- Why just one copy of data: $M = O(n^2 / P)$ per processor?
- Recall lower bounds:

  #words_moved $= \Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

  #messages $= \Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$

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	**[GDX10]**	**log P**	**log P**
QR	**[DGHL08]**	**log P**	**$\log^3 P$**
Sym Eig, SVD	**[BDD11]**	**log P**	**$\log^3 P$**
Nonsym Eig	**[BDD11]**	**log P**	**$\log^3 P$**

Can we do even better?

- Assume nxn matrices on P processors
- Why just one copy of data: $M = O(n^2 / P)$ per processor?
- Increase M to reduce lower bounds:

$$\#\text{words\_moved} = \Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$$

$$\#\text{messages} = \Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$$

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	**[GDX10]**	**log P**	**log P**
QR	**[DGHL08]**	**log P**	**$\log^3$ P**
Sym Eig, SVD	**[BDD11]**	**log P**	**$\log^3$ P**
Nonsym Eig	**[BDD11]**	**log P**	**$\log^3$ P**

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

- #words_moved = $\Omega((n^3/P)/M^{1/2})$
- If c copies of data, $M = c \cdot n^2/P$, bound decreases by factor $c^{1/2}$
- Can we attain it?
- "3D" Matmul Algorithm on $P^{1/3}$ x $P^{1/3}$ x $P^{1/3}$ processor grid
  - $P^{1/3}$ redundant copies of A and B
  - Reduces communication volume to $O((n^2/P^{2/3}) \log(P))$
    - optimal for $P^{1/3}$ copies (more memory can't help)
  - Reduces number of messages to $O(\log(P))$ – also optimal
- "2.5D" Algorithms
  - Extends to $1 \leq c \leq P^{1/3}$ copies on $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid
  - Reduces communication volume of Matmul and LU by $c^{1/2}$
  - Reduces comm 83% on 64K proc BG-P, LU&MM speedup 2.6x
- **Distinguished Paper Prize, Euro-Par'11 (E. Solomonik, JD)**

Lower bound for Strassen's fast matrix multiplication

	Recall O(n^3) case:	For Strassen's:	For Strassen-like:
Sequential:	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} M\right)$
Parallel:	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \frac{M}{P}\right)$

- Parallel lower bounds apply to 2D (1 copy of data) and 2.5D (c copies)
- Attainable
  - Sequential: usual recursive algorithms, also for LU, QR, eig, SVD,…
  - Parallel: just matmul so far …
- Talk by Oded Schwartz, Thursday, 5:30pm
- **Best Paper Award, SPAA'11 (Ballard, JD, Holtz, Schwartz)**

Sequential Strong Scaling

	Standard Alg.	CA-CG with SA1	CA-CG with SA2
1D 3-pt stencil	$\gamma \gtrsim \beta$ $\gamma \frac{n}{p} \gtrsim \alpha$	$\gamma s^2 \gtrsim \beta$ $\gamma \frac{s^2 n}{p} \gtrsim \alpha$	$\gamma s \gtrsim \beta$ $\gamma \frac{s^2 n}{p} \gtrsim \alpha$
2D 5-pt stencil	$\gamma \gtrsim \beta$ $\gamma \frac{n^2}{p} \gtrsim \alpha$	$\gamma s \gtrsim \beta$ $\gamma \frac{s n^2}{p} \gtrsim \alpha$	$\gamma \gtrsim \beta$ $\gamma \frac{s n^2}{p} \gtrsim \alpha$
3D 7-pt stencil	$\gamma \gtrsim \beta$ $\gamma \frac{n^3}{p} \gtrsim \alpha$	$\gamma s \gtrsim \beta$ $\gamma \frac{s n^3}{p} \gtrsim \alpha$	$\gamma \gtrsim \beta$ $\gamma \frac{s n^3}{p} \gtrsim \alpha$

Parallel Strong Scaling

	Standard Alg.	CA-CG with PA1
1D 3-pt stencil	$\gamma \frac{n}{p} \gtrsim \beta$ $\gamma \frac{n}{p \log p} \gtrsim \alpha$	$\gamma \frac{n}{p} \gtrsim \beta$ $\gamma \frac{s^2 n}{p \log p} \gtrsim \alpha$
2D 5-pt stencil	$\gamma \frac{n}{\sqrt{p}} \gtrsim \beta$ $\gamma \frac{n^2}{p \log p} \gtrsim \alpha$	$\gamma \frac{n}{\sqrt{p}} \gtrsim \beta$ $\gamma \frac{s n^2}{p \log p} \gtrsim \alpha$
3D 7-pt stencil	$\gamma \frac{n}{p^{1/3}} \gtrsim \beta$ $\gamma \frac{n^3}{p \log p} \gtrsim \alpha$	$\gamma \frac{n}{p^{1/3}} \gtrsim \beta$ $\gamma \frac{s n^3}{p \log p} \gtrsim \alpha$

Weak Scaling

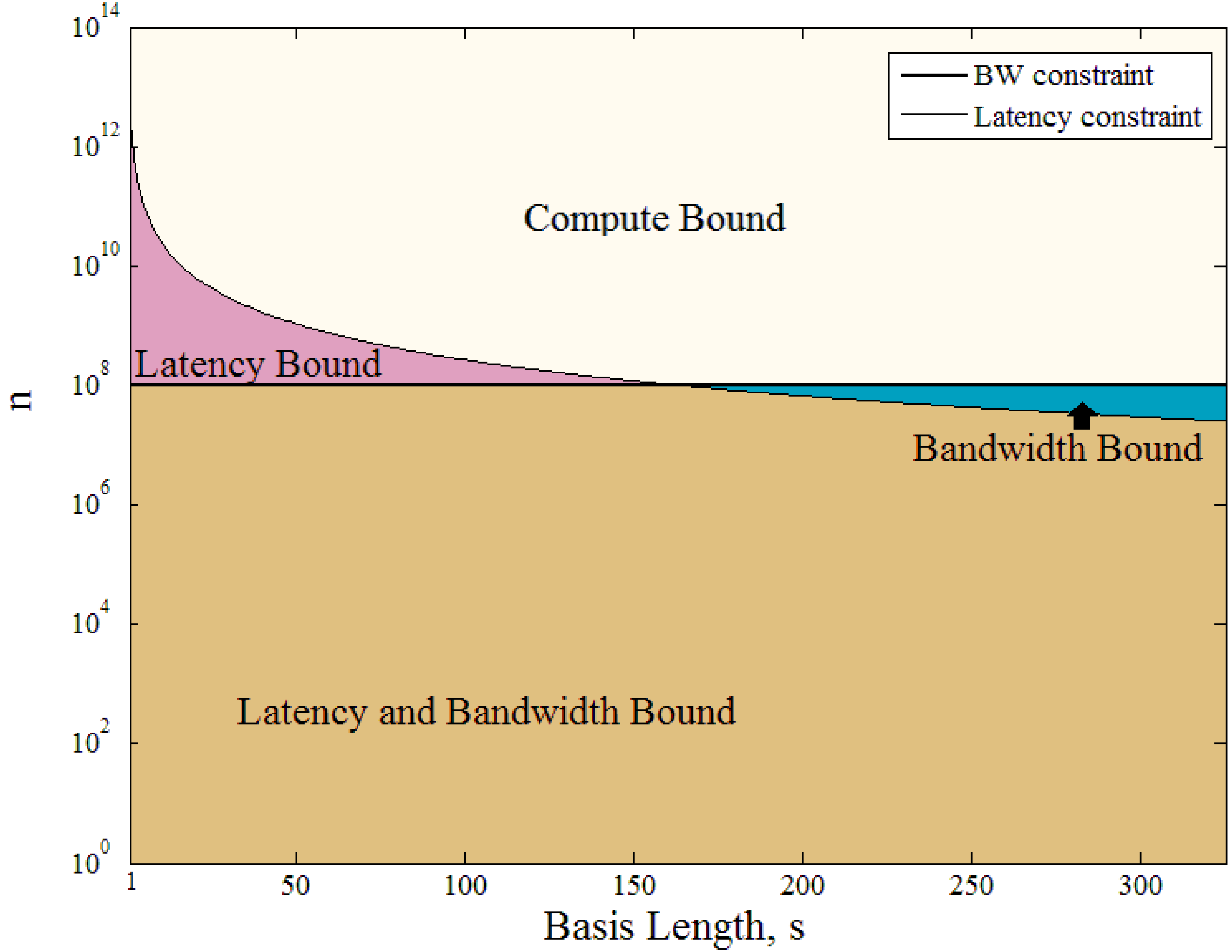
- Change p to x*p, n to x^(1/d)*n
  - d = {1, 2, 3} for 1D, 2D, and 3D mesh

- Bandwidth
  - Perfect weak scaling for 1D, 2D, and 3D
- Latency
  - Perfect weak scaling for 1D, 2D, and 3D if you ignore the log(xp) factor in the denominator
    - Makes constraint on alpha harder to satisfy

Performance Model Assumptions

- Plot for Parallel Algorithm for 1D 3-pt stencil
- Exascale machine parameters:
  - 100 GB/sec interconnect BW
  - 1 microsecond network latency
  - 2^28 cores
  - .1 ns per flop (per core)

Predicted CA-CG Performance on Exascale Machines



Observations

- s =1 are the constraints for the standard algorithm
  - Standard algorithm is communication bound if n <~ 10^{12}
- For 10^{8} <~ n <~ 10^{12}, we can theoretically increase s such that the algorithm is no longer communication bound
  - In practice, high s values have some complications due to stability, but even s ~ 10 can remove communication bottleneck for matrix sizes ~10^{10}