Minisymposia 9 and 34:

Avoiding Communication in Linear Algebra

Jim Demmel

UC Berkeley

bebop.cs.berkeley.edu

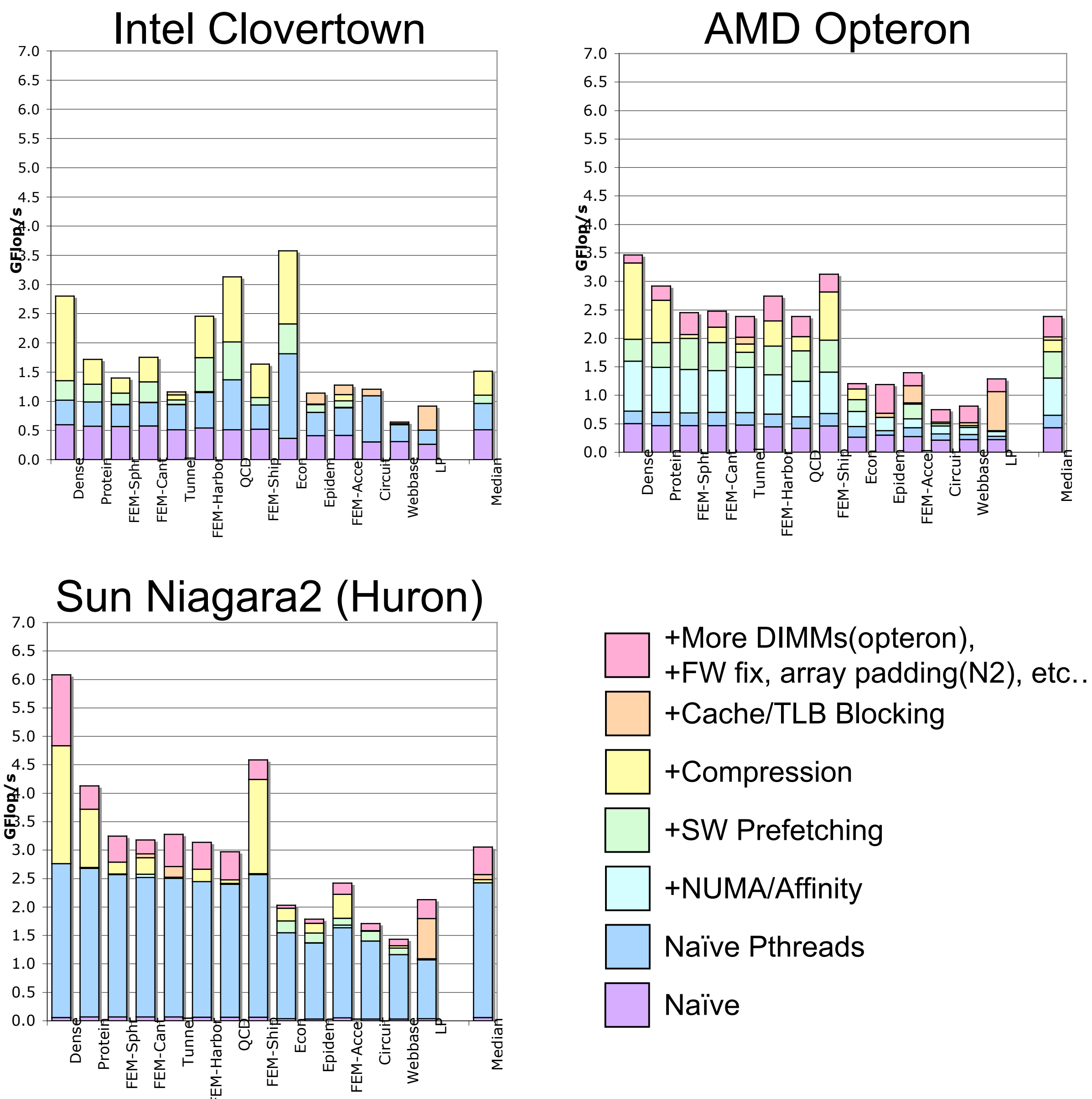
Motivation (1)

- Increasing parallelism to exploit
  - From Top500 to multicores in your laptop
- Exponentially growing gaps between
  - Floating point time << 1/Network BW << Network Latency
    - **Improving 59%/year vs 26%/year vs 15%/year**
  - Floating point time << 1/Memory BW << Memory Latency
    - **Improving 59%/year vs 23%/year vs 5.5%/year**
- Goal 1: reorganize linear algebra to *avoid* communication
  - Not just *hiding* communication (speedup ≤ 2x)
  - Arbitrary speedups possible

Motivation (2)

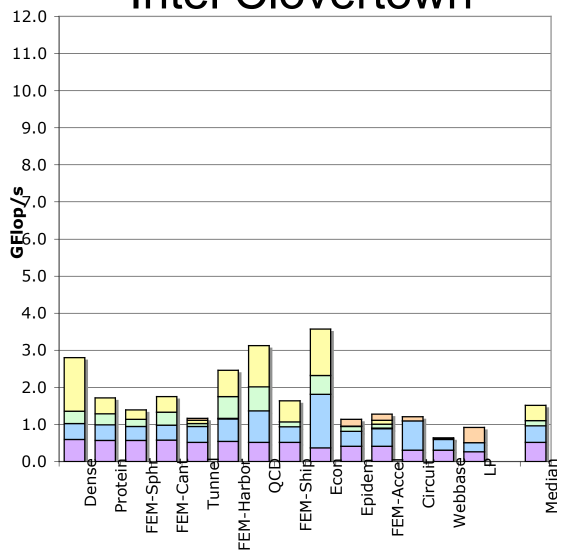
- Algorithms and architectures getting more complex
- Performance harder to understand
- Can’t count on conventional compiler optimizations
- Goal 2: Automate algorithm reorganization
  - “Autotuning”
  - Emulate success of PHiPAC, ATLAS, FFTW, OSKI etc.
- Example:
  - Sparse-matrix-vector-multiply (SpMV) on multicore, Cell
  - Sam Williams, Rich Vuduc, Lenny Oliker, John Shalf, Kathy Yelick

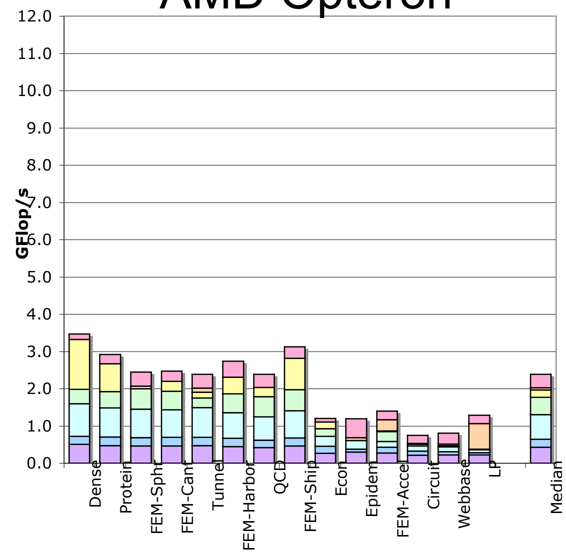
Autotuned Performance of SpMV(1)



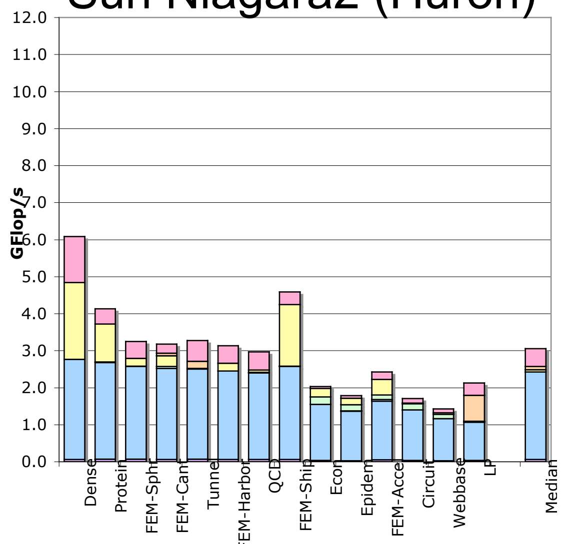
- Clovertown was already fully populated with DIMMs
- Gave Opteron as many DIMMs as Clovertown
- Firmware update for Niagara2
- Array padding to avoid inter-thread conflict misses
- PPE's use ~1/3 of Cell chip area

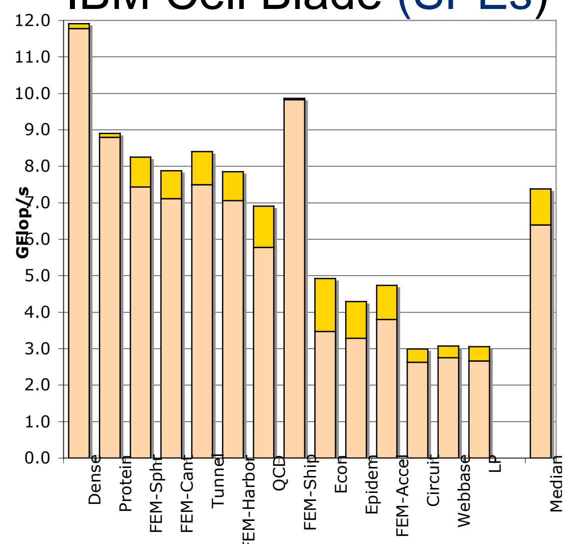
Autotuned Performance of SpMV(2)

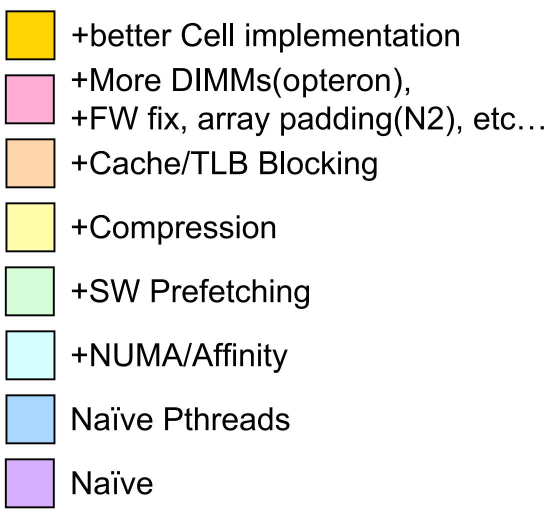




- Model faster cores by commenting out the inner kernel calls, but still performing all DMAs
- Enabled 1x1 BCOO
- ~16% improvement



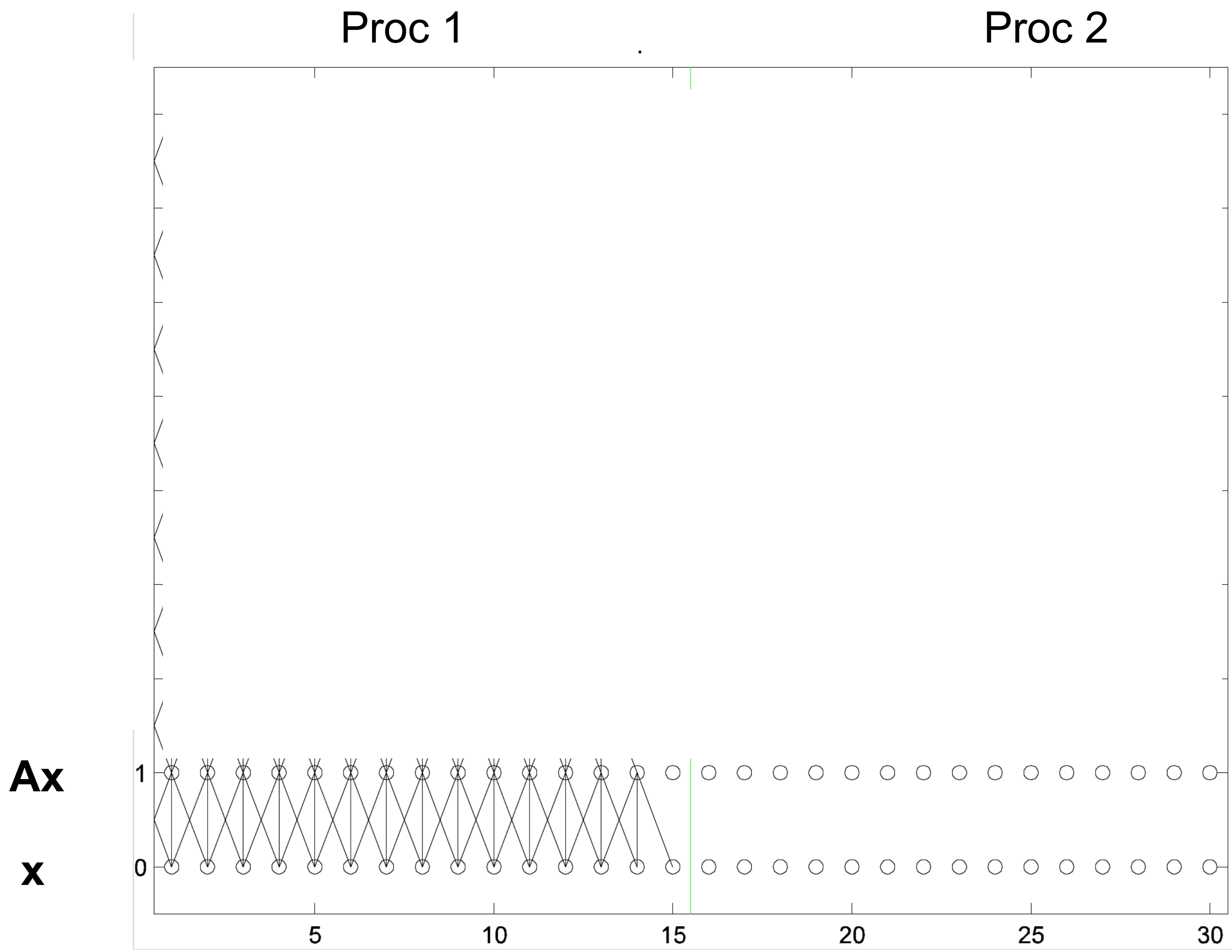




Outline of Minisymposia 9 & 34

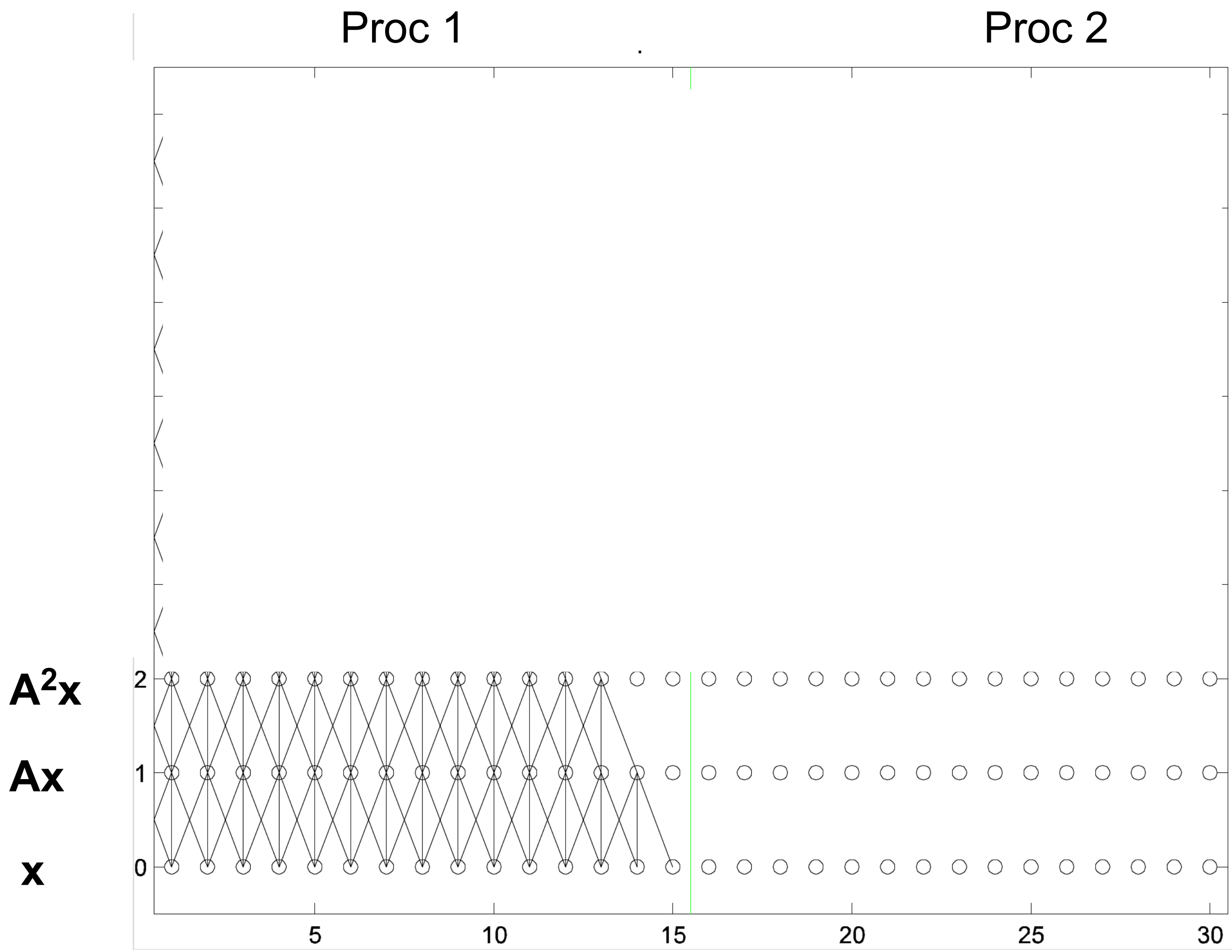
- Minimize communication in linear algebra, autotuning
- MS9: Direct Methods (now)
  - Dense LU: Laura Grigori
  - Dense QR: Julien Langou
  - Sparse LU: Hua Xiang
- MS34: Iterative methods (Thursday, 4-6pm)
  - Jacobi iteration with Stencils: Kaushik Datta
  - Gauss-Seidel iteration: Michelle Strout
  - Bases for Krylov Subspace Methods: Marghoob Mohiyuddin
  - Stable Krylov Subspace Methods: Mark Hoemmen

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



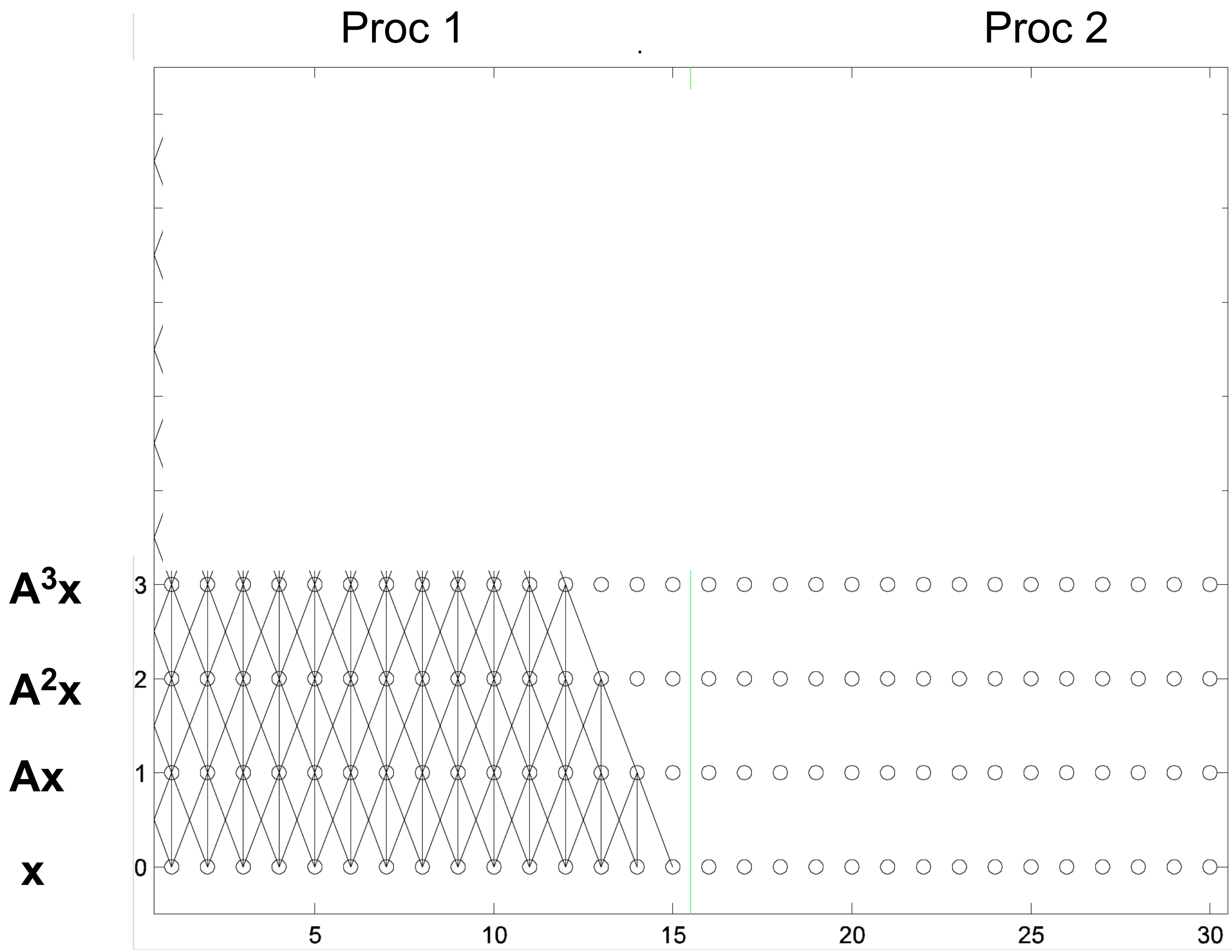
Can be computed without communication

Locally Dependent Entries for [x,Ax,A²x], A tridiagonal
2 processors



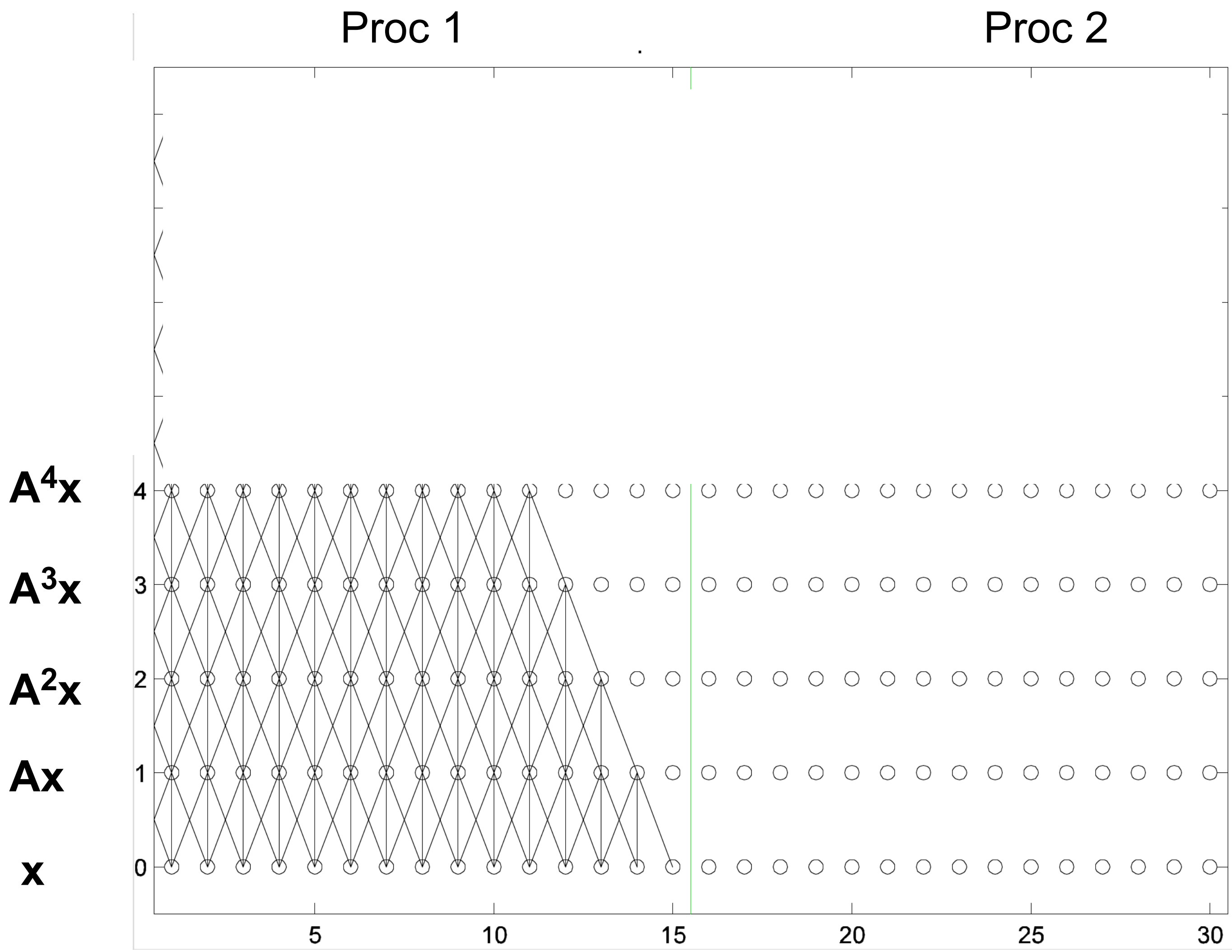
Can be computed without communication

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



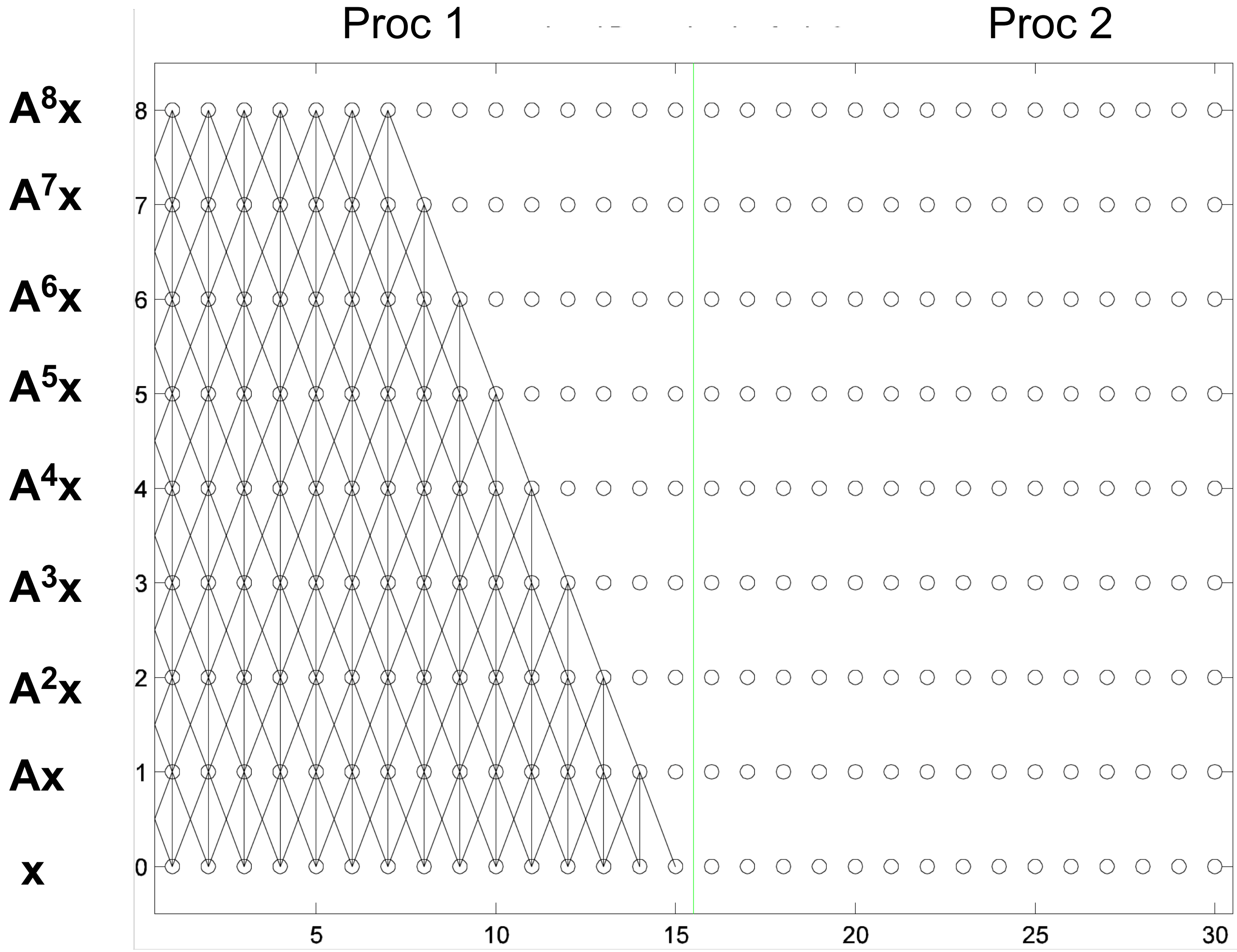
Can be computed without communication

Locally Dependent Entries for [x,Ax,…,A^{4}x], A tridiagonal
2 processors



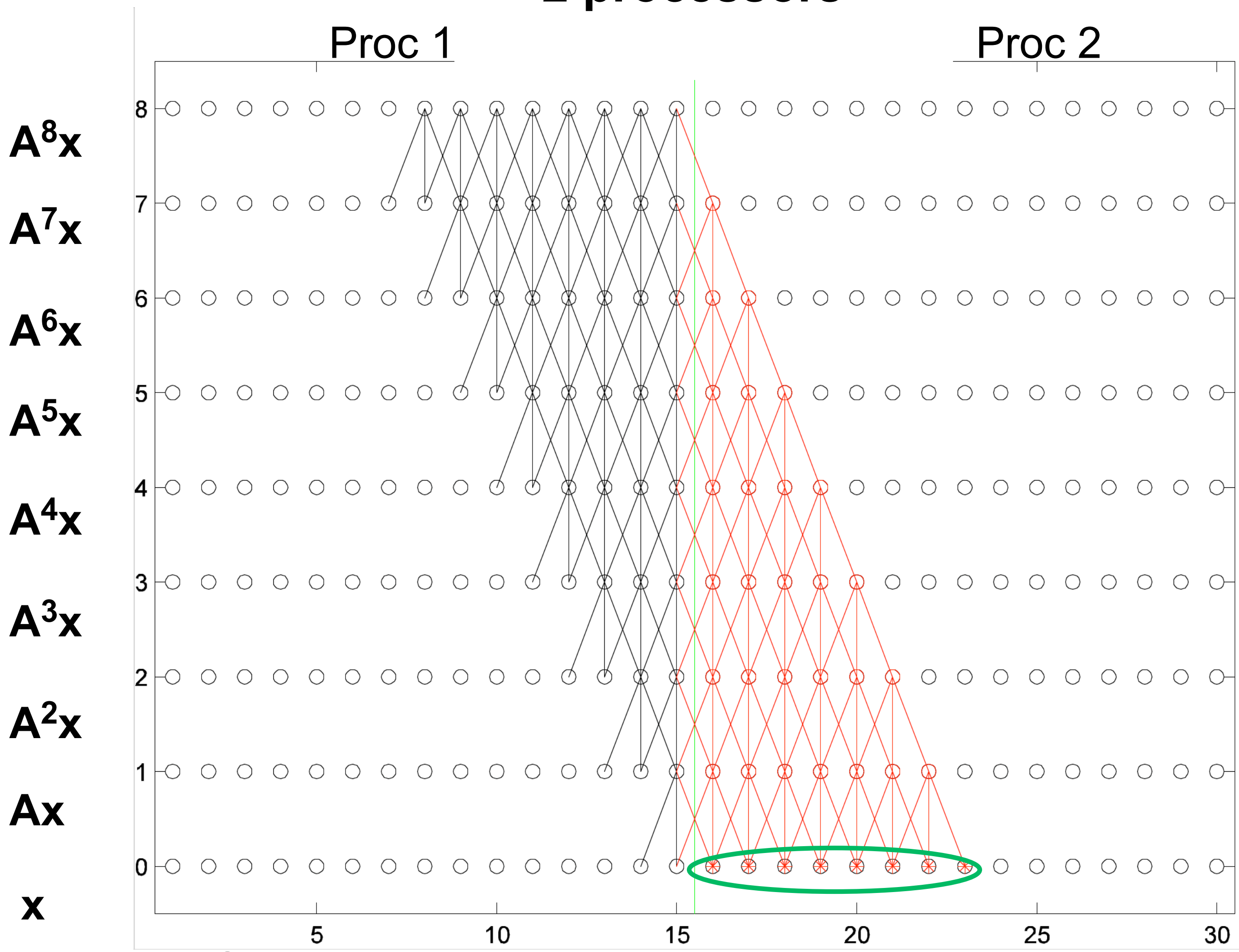
Can be computed without communication

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,…,A⁸x], A tridiagonal 2 processors

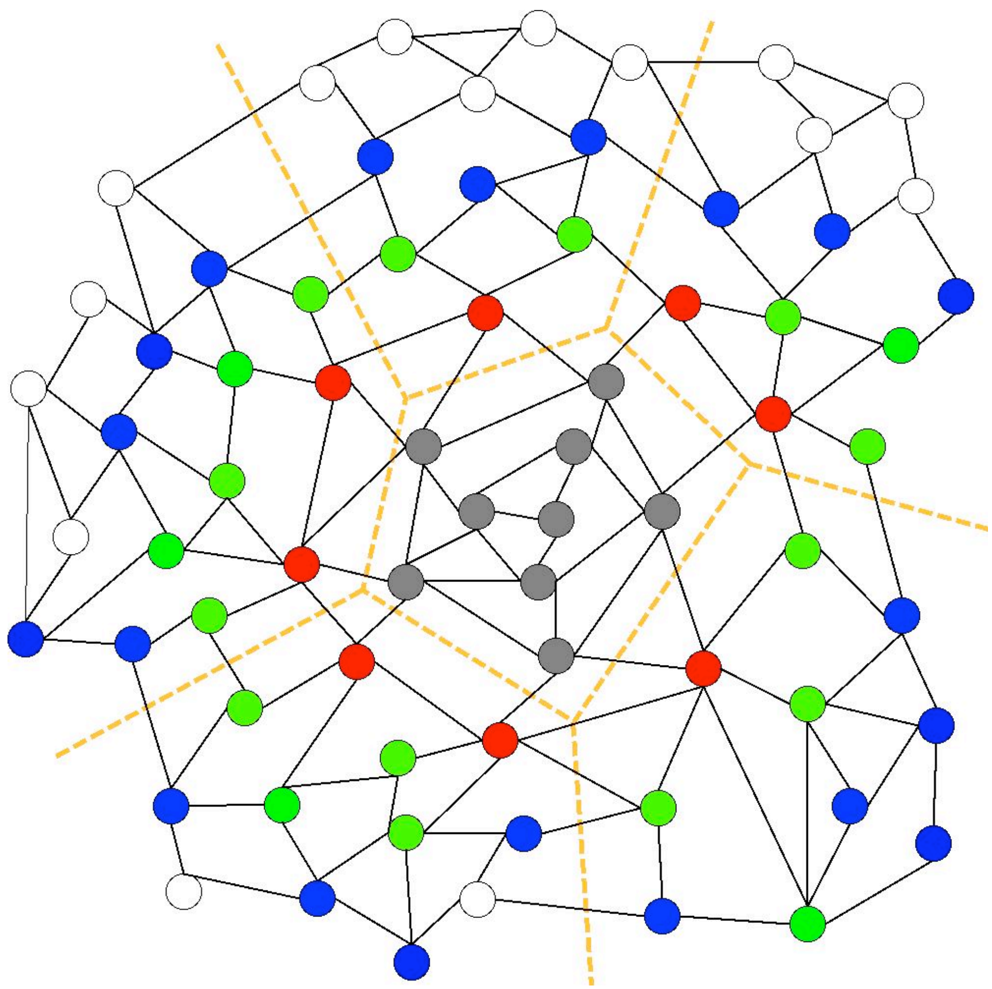


One message to get data needed to compute remotely dependent entries, ***not k=8***

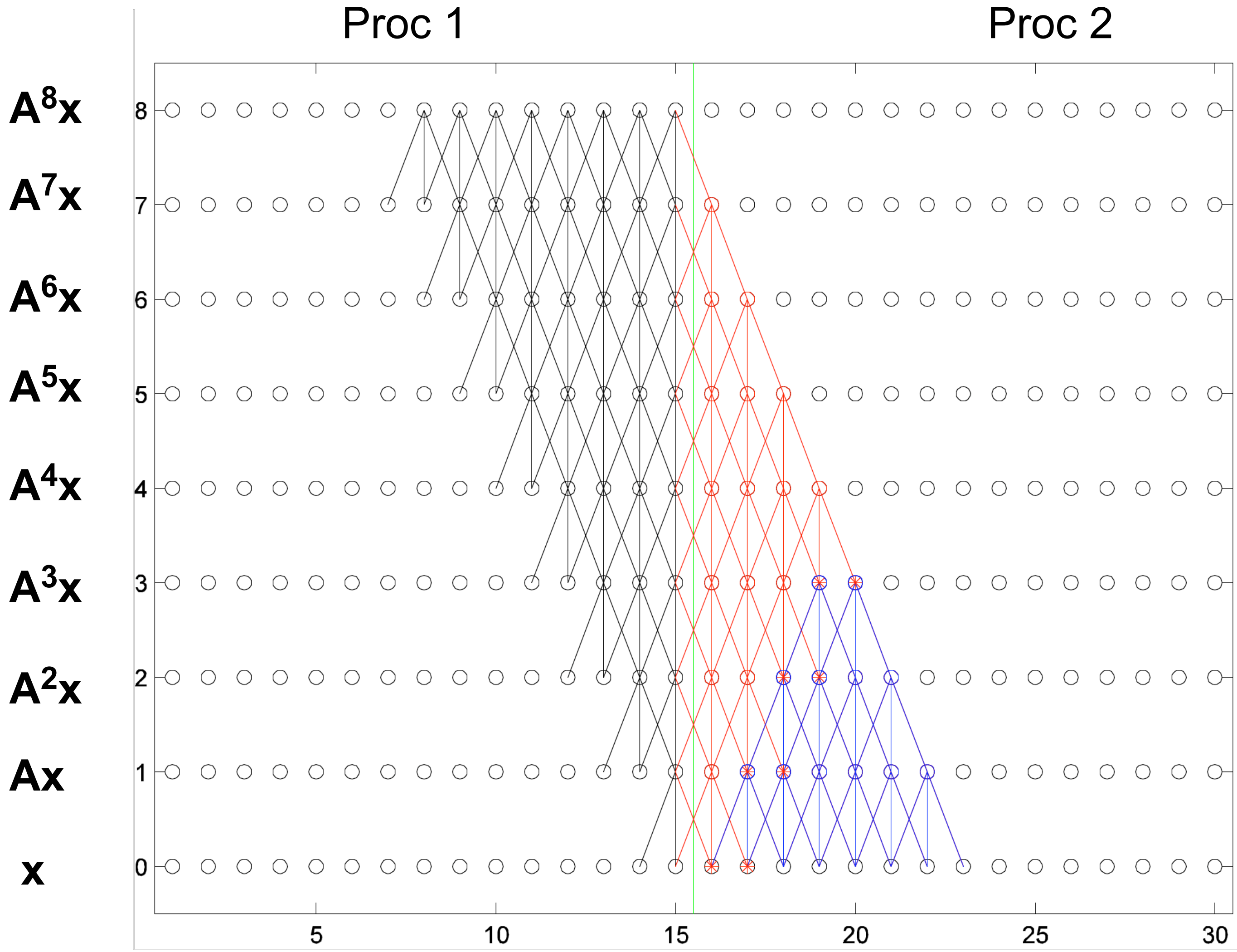
Minimizes number of messages = latency cost

Price: redundant work ∝ "surface/volume ratio"

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

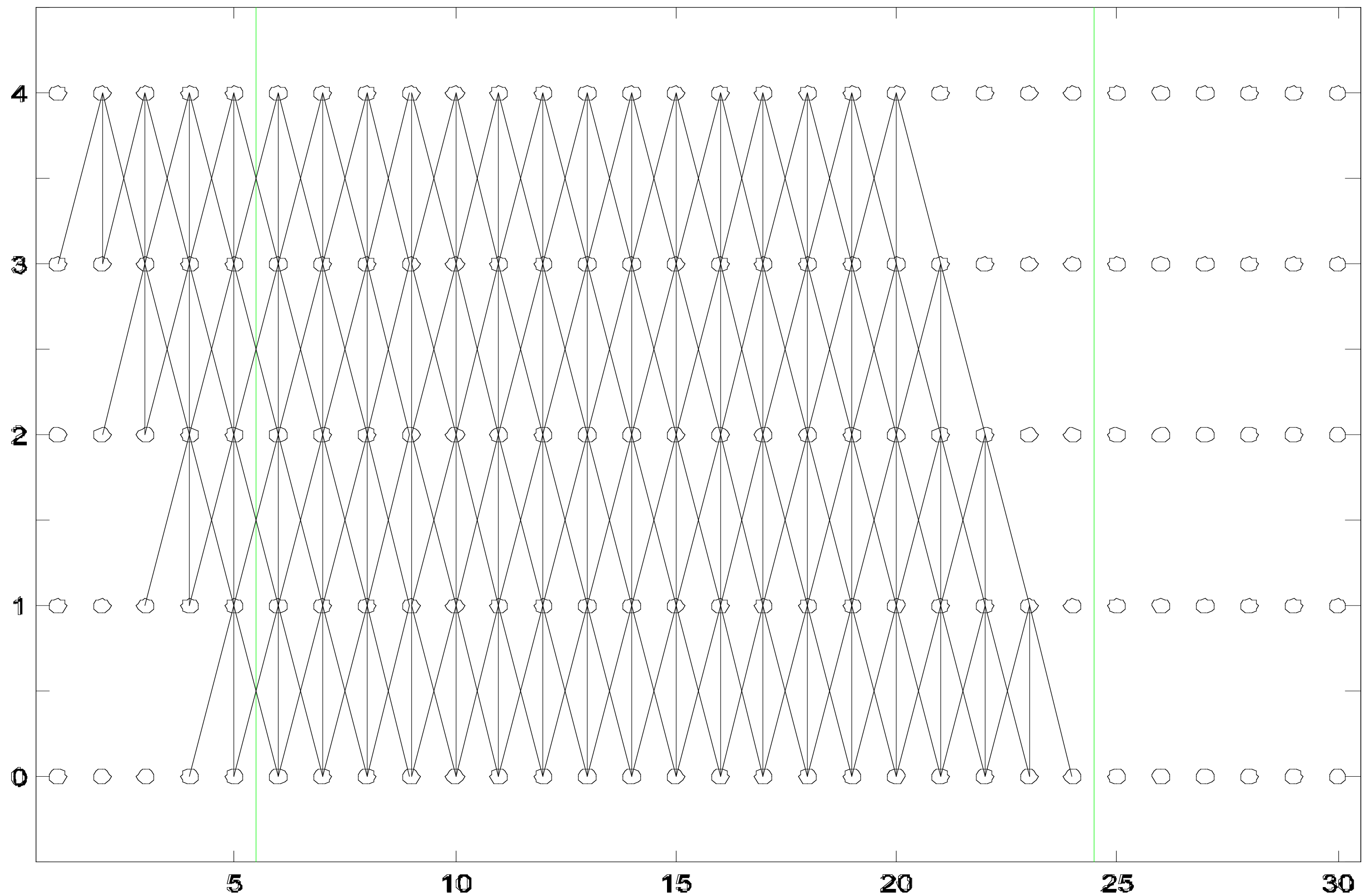


Fewer Remotely Dependent Entries for $[x, Ax, \ldots, A^8x]$, A tridiagonal
2 processors



Reduce redundant work by half

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

Design Goals for [$x, Ax, \ldots, A^k x$]

- Parallel case
  - Goal: Constant number of messages, not $O(k)$
    - Minimizes latency cost
    - Possible price: extra flops and/or extra words sent, amount depends on surface/volume
- Sequential case
  - Goal: Move A, vectors once through memory hierarchy, not k times
    - Minimizes bandwidth cost
    - Possible price: extra flops, amount depends on surface/volume

Design Space for $[x, Ax, \ldots, A^k x]$ (1)

- Mathematical Operation
  - Keep last vector $A^k x$ only
    - Jacobi, Gauss Seidel
  - Keep all vectors
    - Krylov Subspace Methods
  - Preconditioning ($Ay=b \Rightarrow MAy=Mb$)
    - $[x, Ax, MAx, AMAx, MAMAx, \ldots, (MA)^k x]$
  - Improving conditioning of basis
    - $W = [x, p_1(A)x, p_2(A)x, \ldots, p_k(A)x]$
    - $p_i(A)$ = degree i polynomial chosen to reduce cond(W)

Design Space for $[x, Ax, \ldots, A^k x]$ (2)

- Representation of sparse A
  - Zero pattern may be explicit or implicit
  - Nonzero entries may be explicit or implicit
    - Implicit $\Rightarrow$ save memory, communication

	Explicit pattern	**Implicit pattern**
Explicit nonzeros	General sparse matrix	Image segmentation
Implicit nonzeros	Laplacian(graph), for graph partitioning	"Stencil matrix" Ex: tridiag(-1,2,-1)

- Representation of dense preconditioners M
  - Low rank off-diagonal blocks (semiseparable)

Design Space for $[x, Ax, \ldots, A^k x]$ (3)

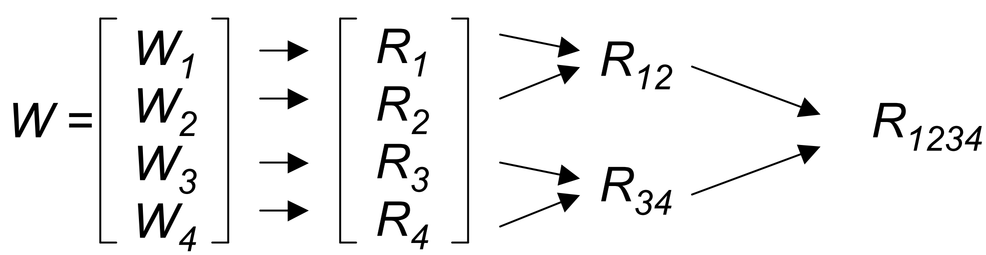
- Parallel implementation
  - From simple indexing,
    with redundant flops $\propto$ surface/volume
  - To complicated indexing, with no redundant flops
    but some extra communication
- Sequential implementation
  - Depends on whether vectors fit in fast memory
- Reordering rows, columns of A
  - Important in parallel and sequential cases
- Plus all the optimizations for one SpMV!

Examples from later talks (MS34)

- Kaushik Datta
  - Autotuning of stencils in parallel case
  - Example: 66 Gflops on Cell (measured)
- Michelle Strout
  - Autotuning of Gauss-Seidel for general sparse A
  - Example speedup: 4.5x (measured)
- Marghoob Mohiyuddin
  - Tuning $[x, Ax, \ldots, A^k x]$ for general sparse A
  - Example speedups:
    - 22x on Petascale machine (modeled)
    - 3x on out-of-core (measured)
- Mark Hoemmen
  - How to use $[x, Ax, \ldots, A^k x]$ *stably* in GMRES, other Krylov methods
  - Requires communication avoiding QR decomposition …

Minimizing Communication in QR

- QR decomposition of m x n matrix W, m >> n
  - P processors, block row layout
- Usual 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



Design space for QR

- TSQR = Tall Skinny QR (m >> n)
  - Shape of reduction tree depends on architecture
    - Parallel: use "deep" tree, saves messages/latency
    - Sequential: use flat tree, saves words/bandwidth
    - Multicore: use mixture
  - QR($\begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$): save half the flops since R_i triangular
  - Recursive QR
- General QR
  - Use TSQR for panel factorizations
- If it works for QR, why not LU?

Examples from later talks (MS9)

- Laura Grigori
  - Dense LU
  - How to pivot stably?
  - 12x speeds (measured)
- Julien Langou
  - Dense QR
  - Speedups up to 5.8x (measured), 23x(modeled)
- Hua Xiang
  - Sparse LU
  - More important to reduce communication

Summary

- Possible to reduce communication to theoretical minimum in various linear algebra computations
  - Parallel: O(1) or O(log p) messages to take k steps, not O(k) or O(k log p)
  - Sequential: move data through memory once, not O(k) times
  - Lots of speed up possible (modeled and measured)
- *Lots* of related work
  - Some ideas go back to 1960s, some new
  - Rising cost of communication forcing us to reorganize linear algebra (among other things!)
- Lots of open questions
  - For which preconditioners M can we avoid communication in $[x, Ax, MAx, AMAx, MAMAx, \ldots, (MA)^k x]$?
  - Can we avoid communication in direct eigensolvers?

bebop.cs.berkeley.edu