Ma221: Matrix Computations

Lecture 28:

Communication-Avoiding Sparse Iterative Methods

James Demmel
(and many collaborators)

www.cs.berkeley.edu/~demmel

Motivation

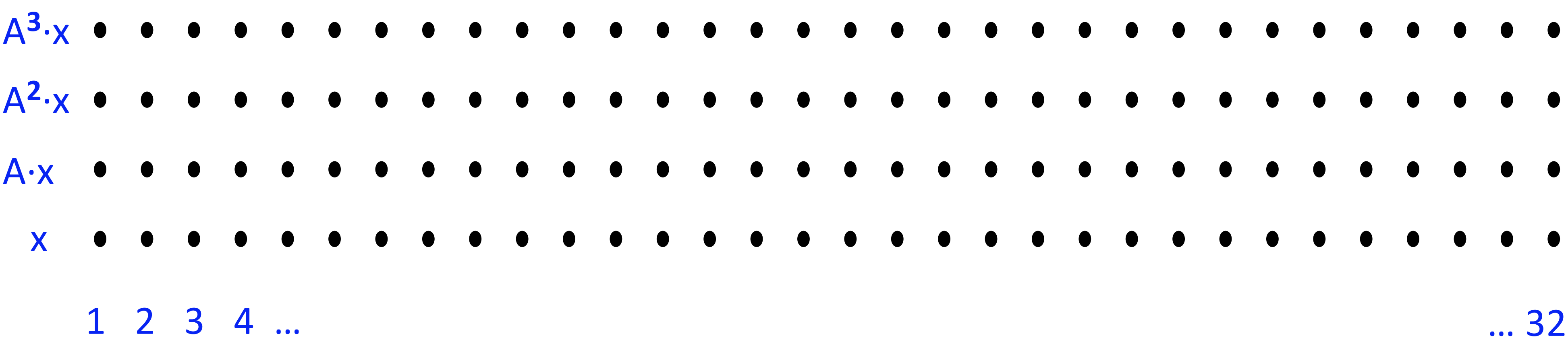
- Iterative Methods (both Splitting Methods and Krylov Subspace Methods) rely on Sparse-Matrix-Vector-Multiplication (SpMV)
  - If A too large to fit in fast memory, need to reread A at each iteration, do only 2 flops per matrix entry, so limited by communication
  - On a parallel machine, typically need to do an operation like a dot product (eg to compute $||Ax-b||_2$) after each iteration, also a bottleneck

Goal

- Reorganize iterative algorithms to take k steps for the communication cost of 1 step
  - k will be a tuning parameter to optimize speedup
- Challenges:
  - A needs to be sparse enough
    - Tradeoff avoiding communication with redundant flops
  - Need to maintain numerical stability
    - Simplest approaches not stable
  - Preconditioning

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

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

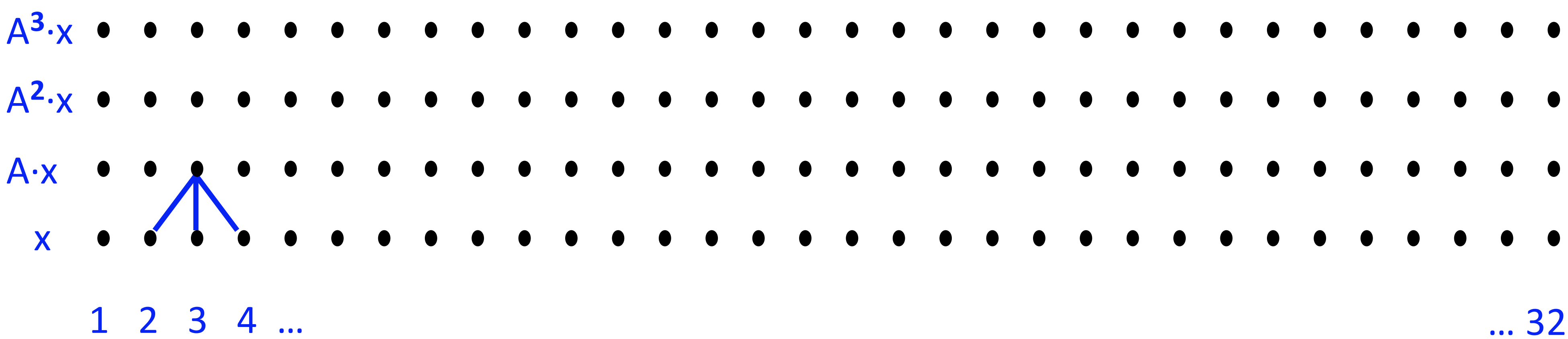


- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:

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

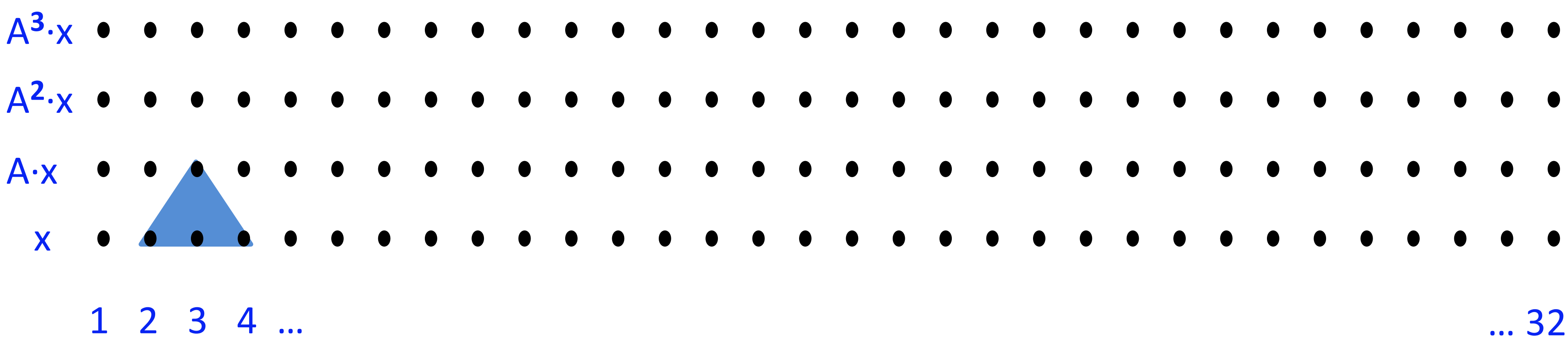
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



- 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 $[Ax, A^2x, \ldots, A^kx]$

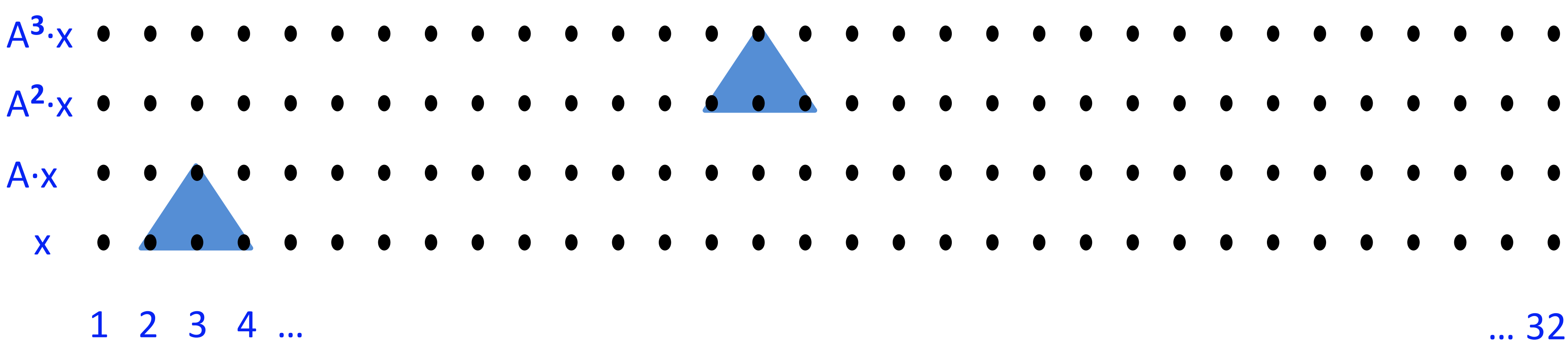


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

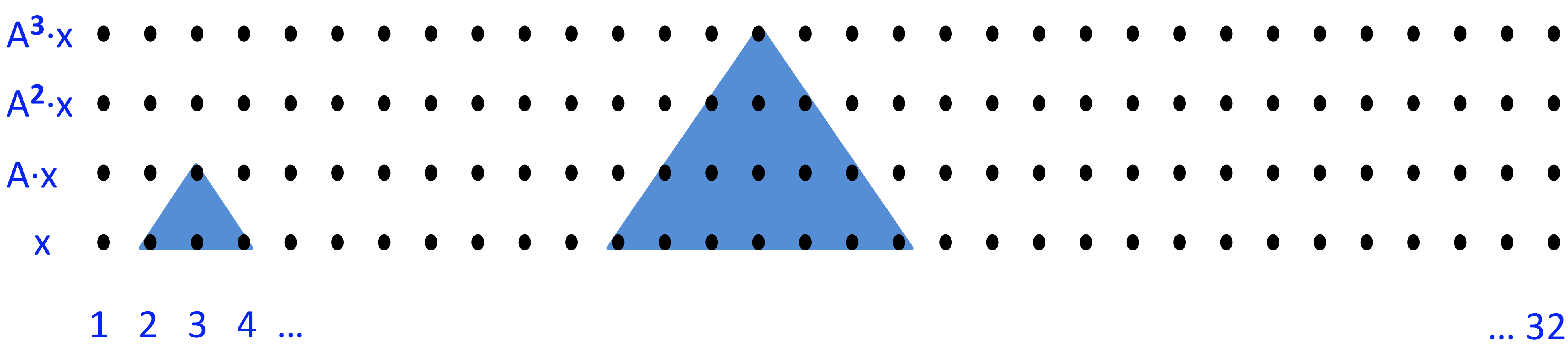
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]



- 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 $[Ax, A^2x, \ldots, A^kx]$

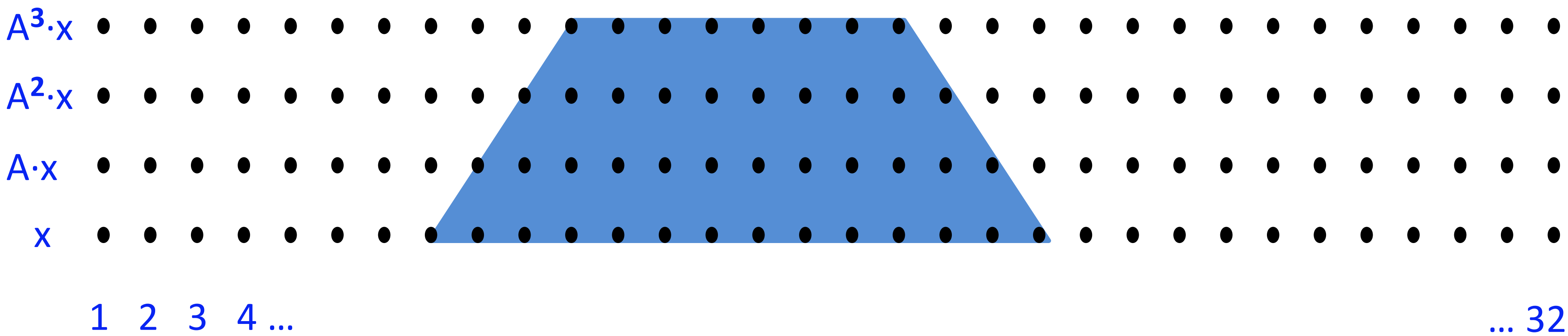


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

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

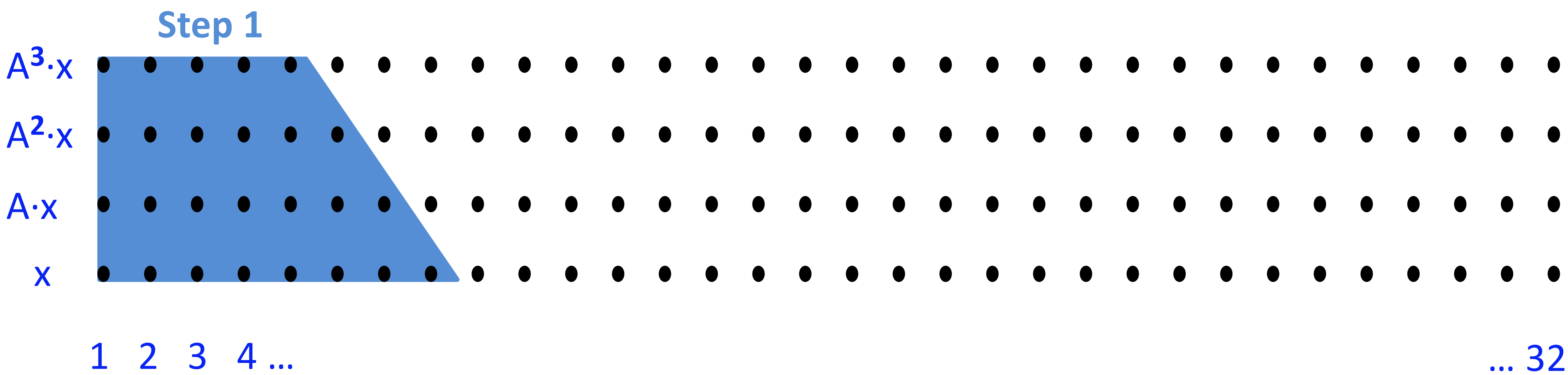


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

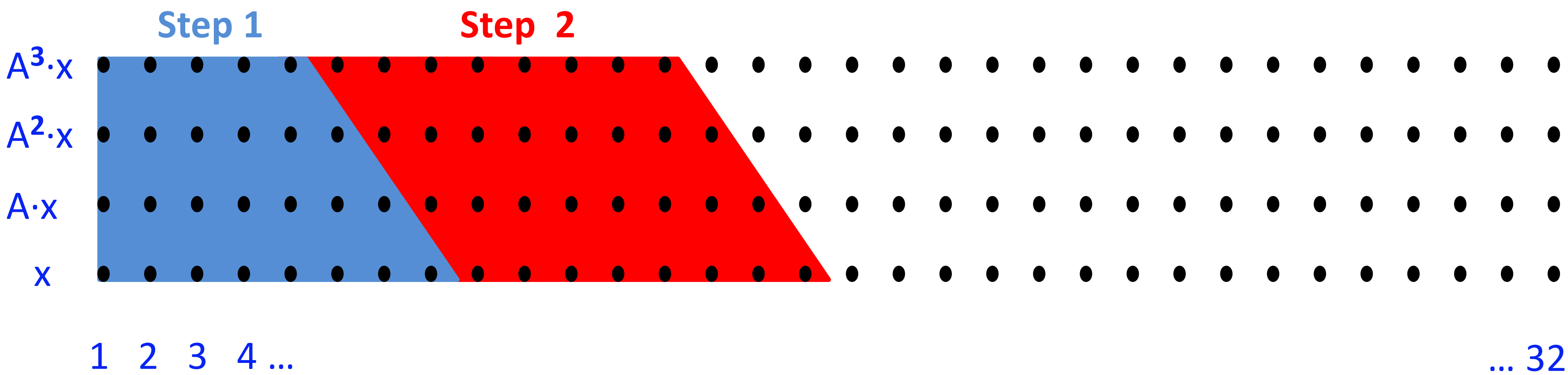
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Sequential Algorithm



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

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

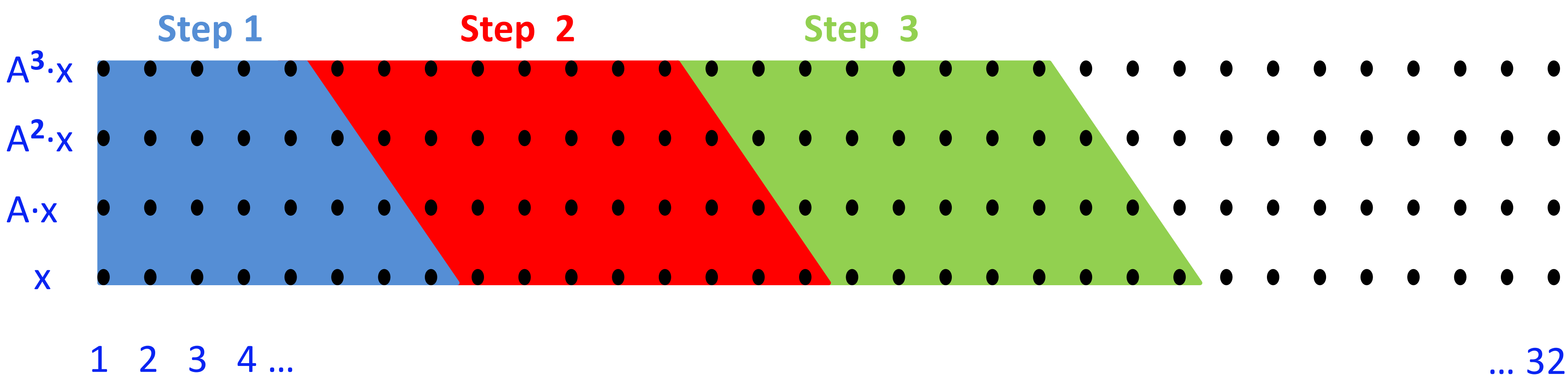
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



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

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

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

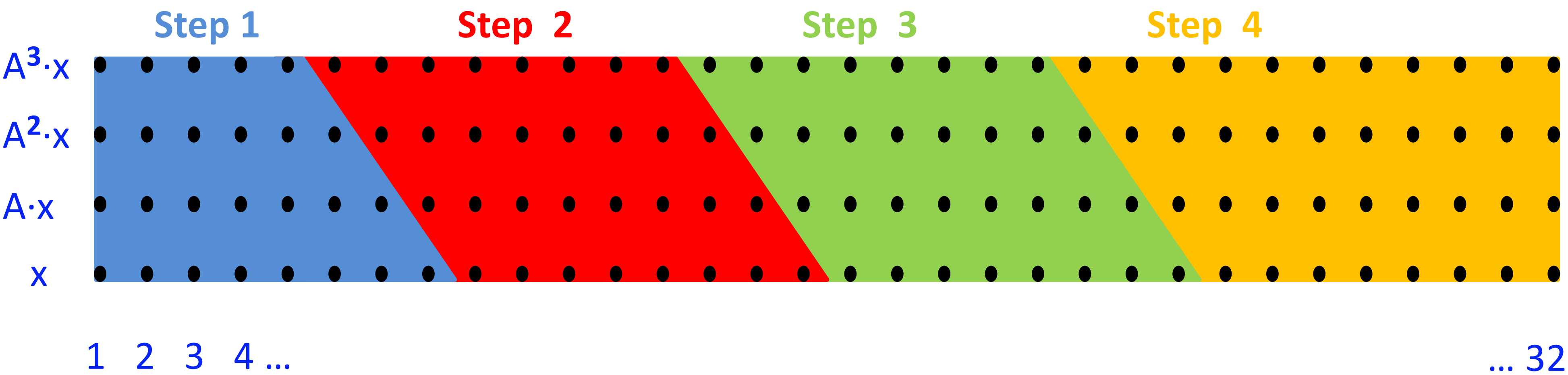


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

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

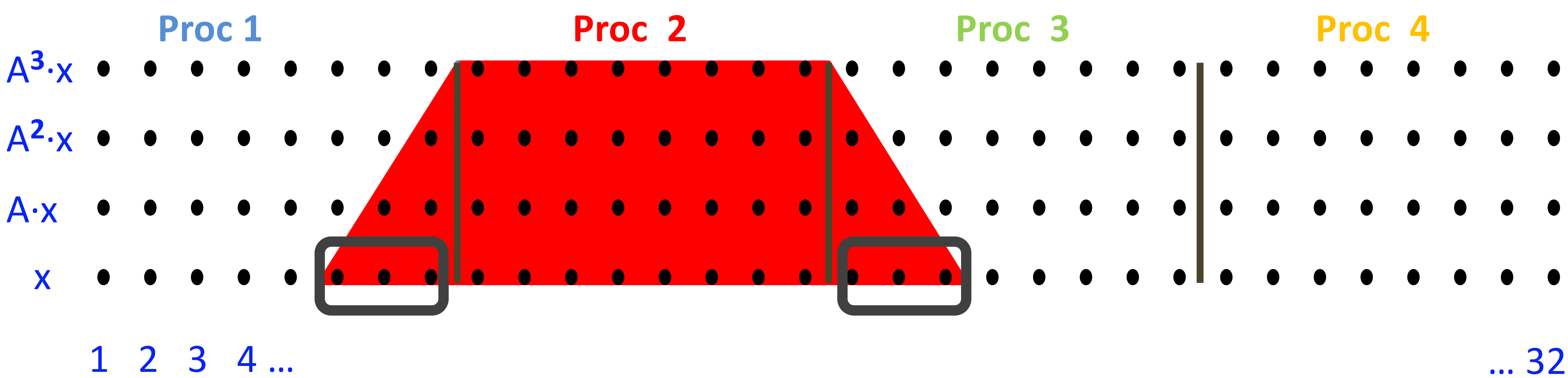
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



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

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

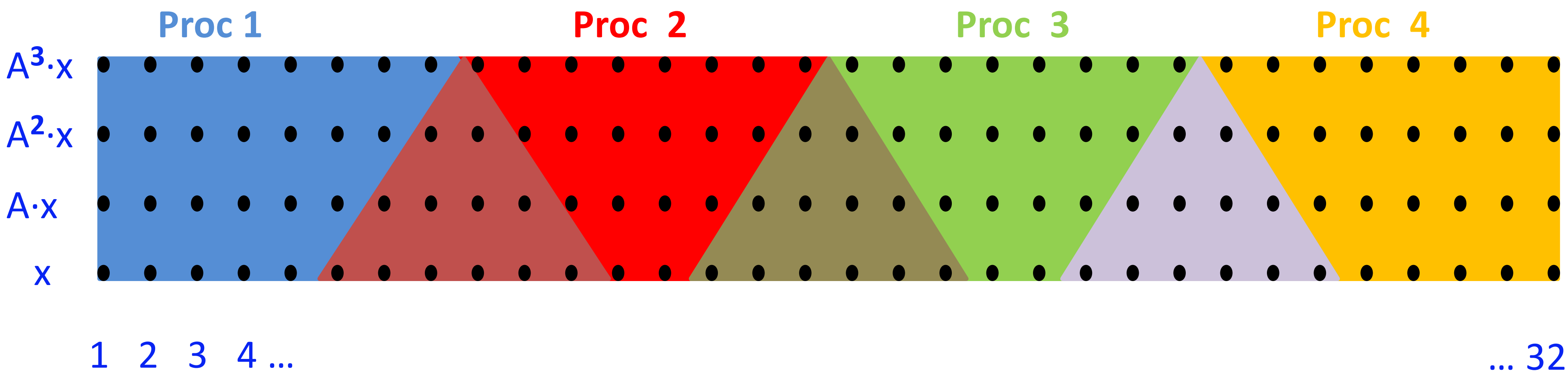
- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]
- Parallel Algorithm



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

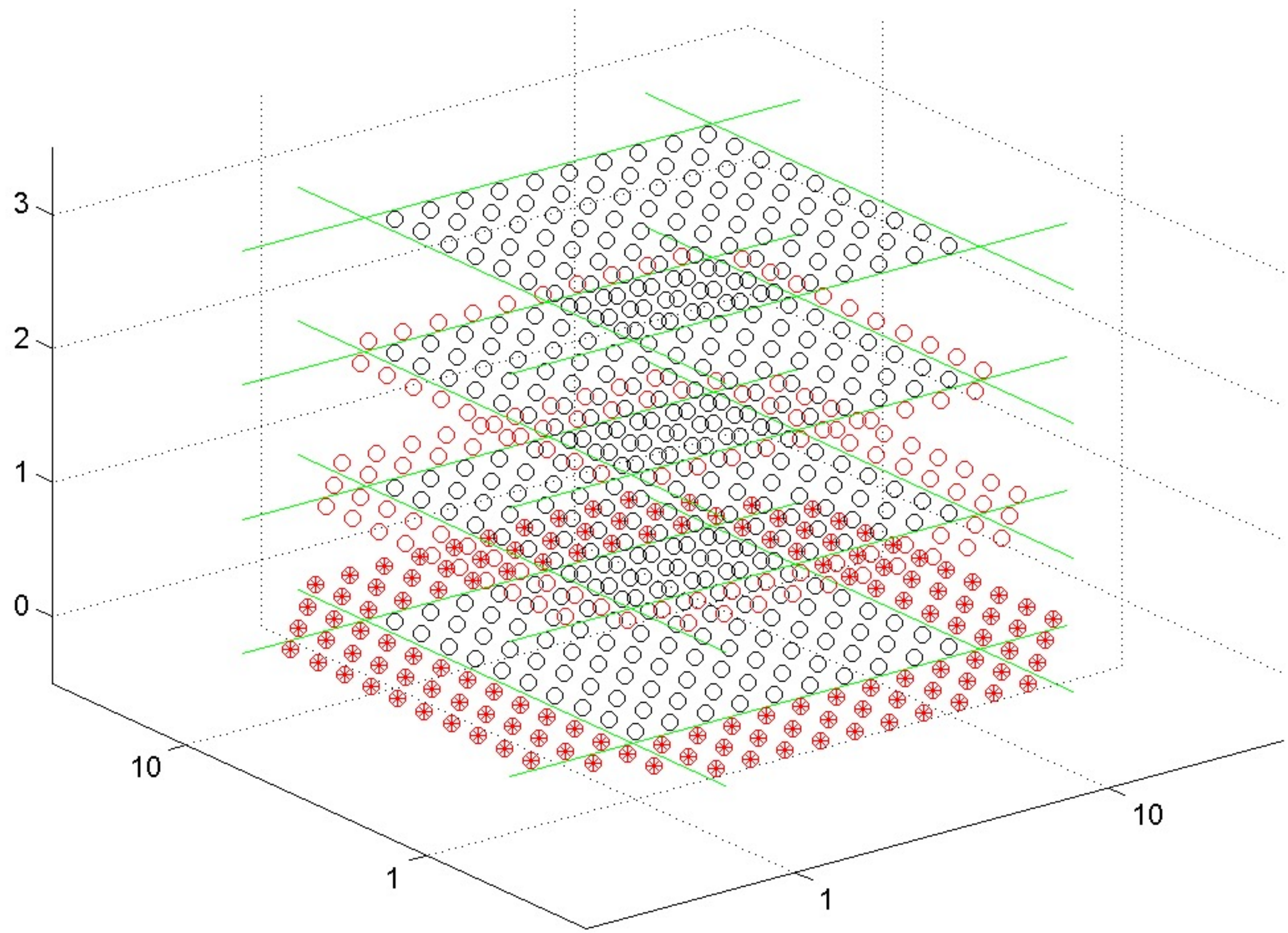
Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

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

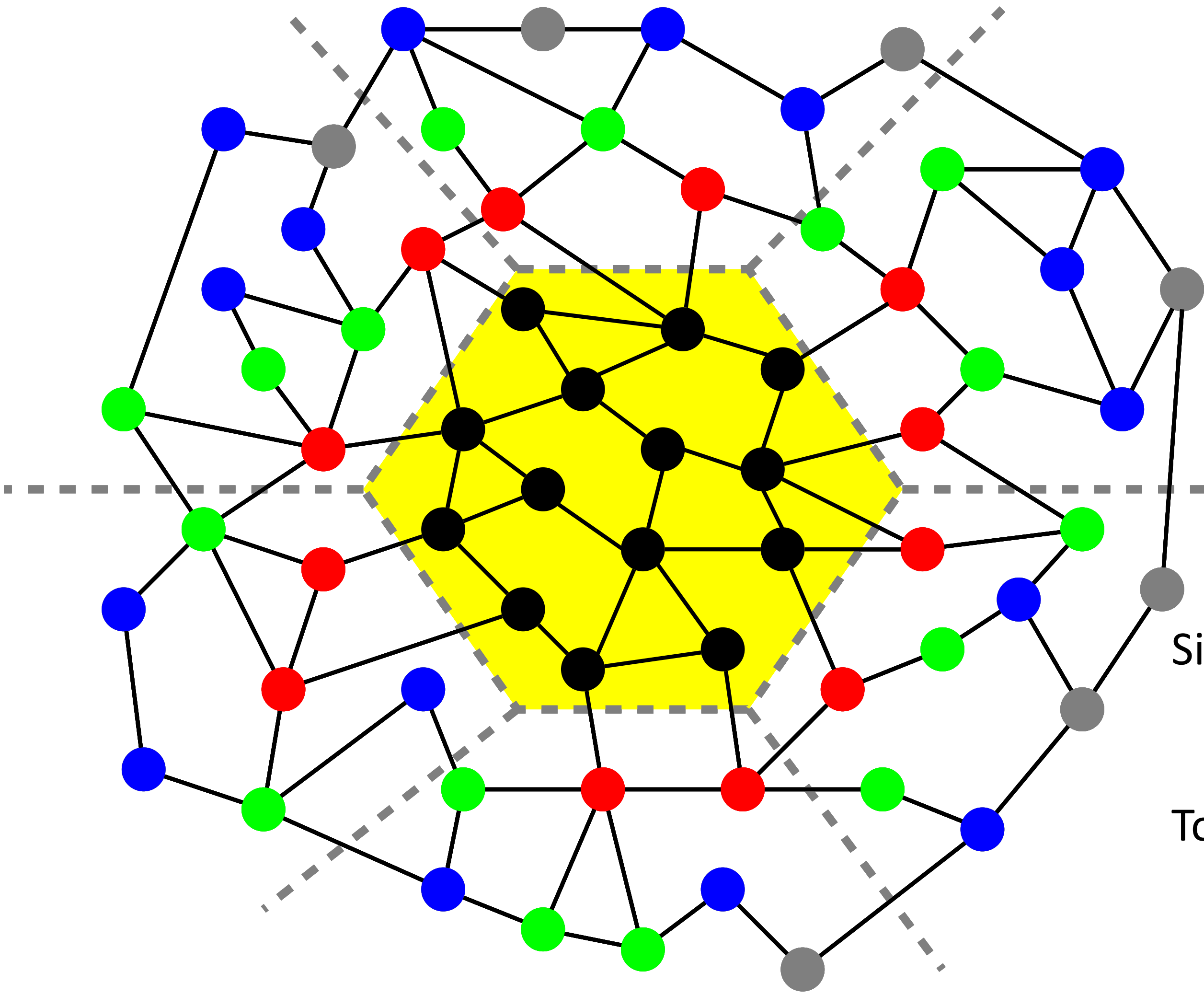


- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

Matrix Power Kernel $[x, Ax, A^2x, A^3x]$ for 2D Poisson



The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$ on a general matrix (nearest k neighbors on a graph)



Simple block-row partitioning ➔ (hyper)graph partitioning

Top-to-bottom processing ➔ Traveling Salesman Problem

Same idea for general sparse matrices: k-wide neighboring region

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
```

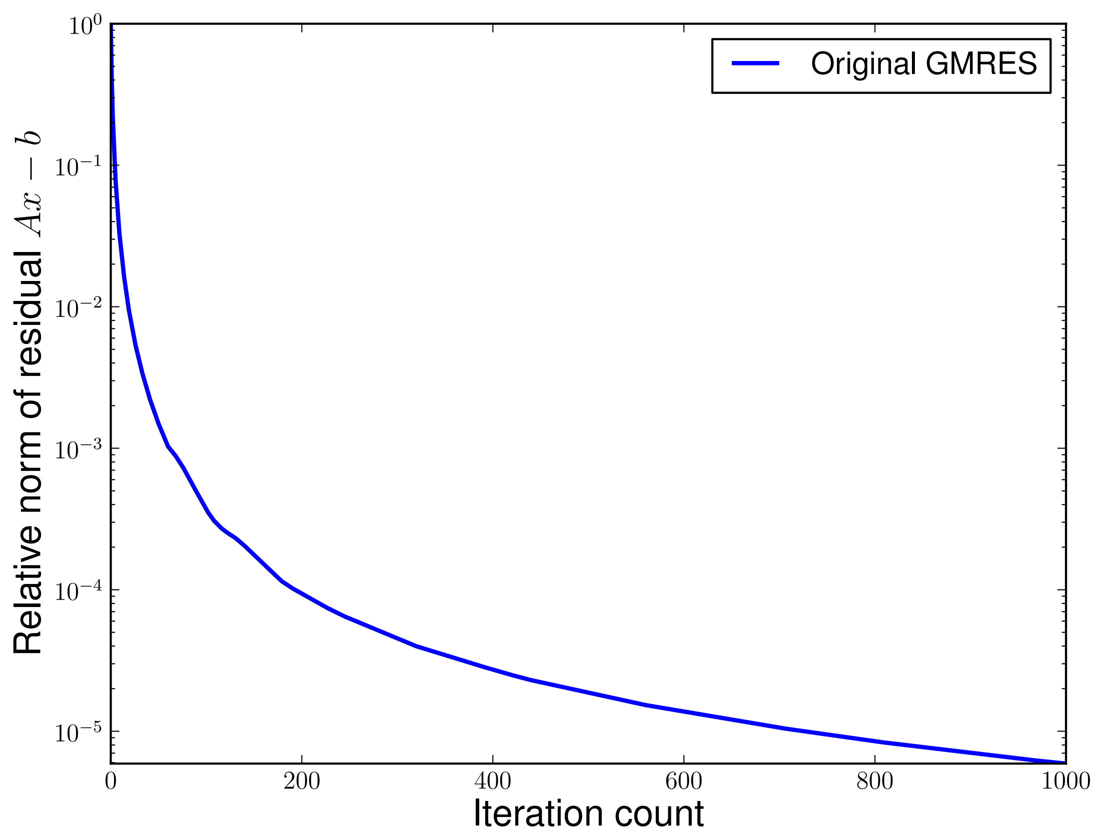
Communication-avoiding GMRES

```
W = [ v, Av, A^2v, … , A^kv ]
[Q,R] = TSQR(W)
      … "Tall Skinny QR"
build H from R
solve LSQ problem with H
```

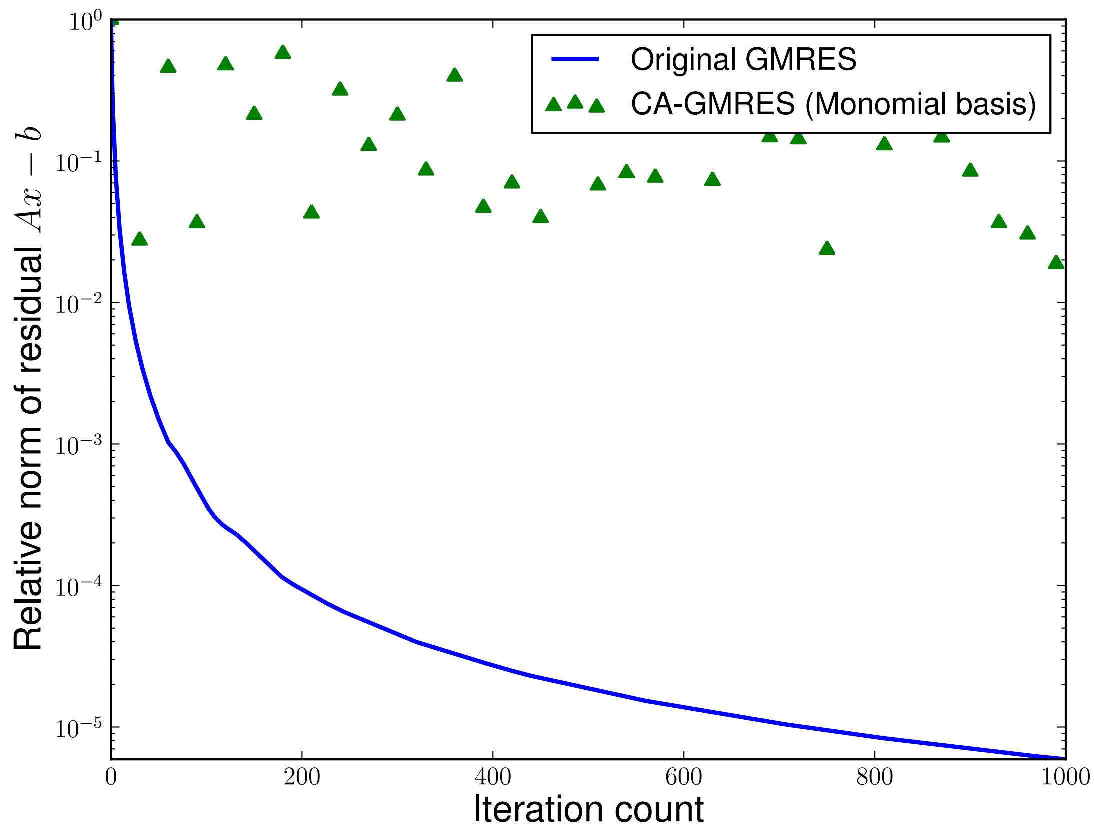
Sequential case: #words moved decreases by a factor of k
Parallel case: #messages decreases by a factor of k

- Oops – W from power method, precision lost!
- Fix: replace W by [v, p_1(A)v, p_2(A)v, …, p_k(A)v]

Convergence of GMRES

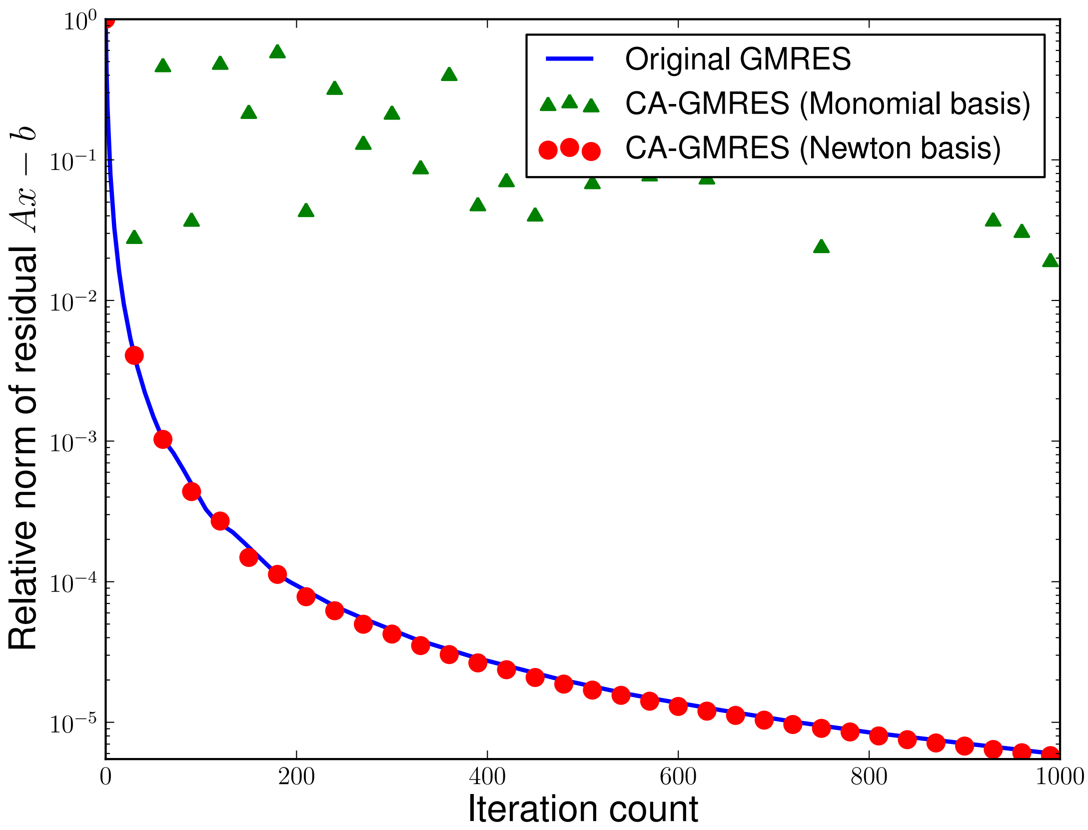


Convergence of GMRES



Convergence of GMRES

Up to **2.3x** speedup on 8 core Intel Clovertown (k=5)



CA-BiCGStab

Up to **4.2x** speedup for BiCGStab on 24K core Cray XE6 (used as coarse grid solver in multigrid)

Compute $r_0 = b - Ax_0$. Choose $r_0^\star$ arbitrary.
Set $p_0 = r_0$, $q_{-1} = 0_{N\times 1}$.
For $k = 0, 1, \ldots,$ until convergence, Do

$P = [p_{sk},\, Ap_{sk},\, \ldots,\, A^s p_{sk}]$
$Q = [q_{sk-1},\, Aq_{sk-1},\, \ldots,\, A^s q_{sk-1}]$
$R = [r_{sk},\, Ar_{sk},\, \ldots,\, A^s r_{sk}]$

//Compute the $1 \times (3s+3)$ Gram vector.
$g = (r_0^\star)^T\, [P,\, Q,\, R]$
//Compute the $(3s+3) \times (3s+3)$ Gram matrix

$$G = \begin{bmatrix} P^T \\ Q^T \\ R^T \end{bmatrix} \begin{bmatrix} P & Q & R \end{bmatrix}$$

For $\ell = 0$ to s,

$$b^\ell_{sk} = \left[B_1(:,\ell)^T,\, 0^T_{s+1},\, 0^T_{s+1}\right]^T$$
$$c^\ell_{sk-1} = \left[0^T_{s+1},\, B_2(:,\ell)^T,\, 0^T_{s+1}\right]^T$$
$$d^\ell_{sk} = \left[0^T_{s+1},\, 0^T_{s+1},\, B_3(:,\ell)^T\right]^T$$

For $j = 0$ to $\lfloor \frac{s}{2} \rfloor - 1$, Do

$$\alpha_{sk+j} = \frac{<g,\, d^0_{sk+j}>}{<g,\, b^1_{sk+j}>}$$
$$q_{sk+j} = r_{sk+j} - \alpha_{sk+j}[P,\, Q,\, R]b^1_{sk+j}$$

For $\ell = 0$ to $s - 2j + 1$, Do

$$c^\ell_{sk+j} = d^\ell_{sk+j} - \alpha_{sk+j} b^{\ell+1}_{sk+j-1}$$

//such that $[P,\, Q,\, R]\, c^\ell_{sk+j} = A^\ell q_{sk+j}$

$$\omega_{sk+j} = \frac{<c^1_{sk+j+1},\, Gc^0_{sk+j+1}>}{<c^1_{sk+j+1},\, Gc^1_{sk+j+1}>}$$
$$x_{sk+j+1} = x_{sk+j} + \alpha_{sk+j} p_{sk+j} + \omega_{sk+j} q_{sk+j}$$
$$r_{sk+j+1} = q_{sk+j} - \omega_{sk+j}[P,\, Q,\, R]c^1_{sk+j+1}$$

For $\ell = 0$ to $s - 2j$, Do

$$d^\ell_{sk+j+1} = c^\ell_{sk+j+1} - \omega_{sk+j} c^{\ell+1}_{sk+j+1}$$

//such that $[P,\, Q,\, R]\, d^\ell_{sk+j+1} = A^\ell r_{sk+j+1}$

$$\beta_{sk+j} = \frac{<g,\, d^0_{sk+j+1}>}{<g,\, d^0_{sk+j}>} \times \frac{\alpha}{\omega}$$
$$p_{sk+j+1} = r_{sk+j+1} + \beta_{sk+j} p_{sk+j} - \beta_{sk+j}\omega_{sk+j}[P,\, Q,\, R]b^1_{sk+j}$$

For $\ell = 0$ to $s - 2j$, Do

$$b^\ell_{sk+j+1} = d^\ell_{sk+j+1} + \beta_{sk+j} b^\ell_{sk+j} - \beta_{sk+j}\omega_{sk+j} b^{\ell+1}_{sk+j}$$

//such that $[P,\, Q,\, R]\, b^\ell_{sk+j+1} = A^\ell p_{sk+j+1}$.

EndDo

EndDo

1. Compute $r_0 := b - Ax_0$; r_0^* arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \ldots,$ until convergence Do:
4. $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $s_j := r_j - \alpha_j Ap_j$
6. $\omega_j := (As_j, s_j)/(As_j, As_j)$
7. $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8. $r_{j+1} := s_j - \omega_j As_j$
9. $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo

Conclusions

- Communication is much more expensive than arithmetic, so avoid it, even if you need to do (a little) more arithmetic!
- For more details, see links on the class webpage under Reference for Communication-Avoiding Algorithms