

Communication-Avoiding Iterative Methods

Erin Carson
UC Berkeley Parallel Computing Lab
BeBop Group

Discovery 2015: HPC and Cloud Computing
Workshop, June 2011

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, et al.)

Talk Outline

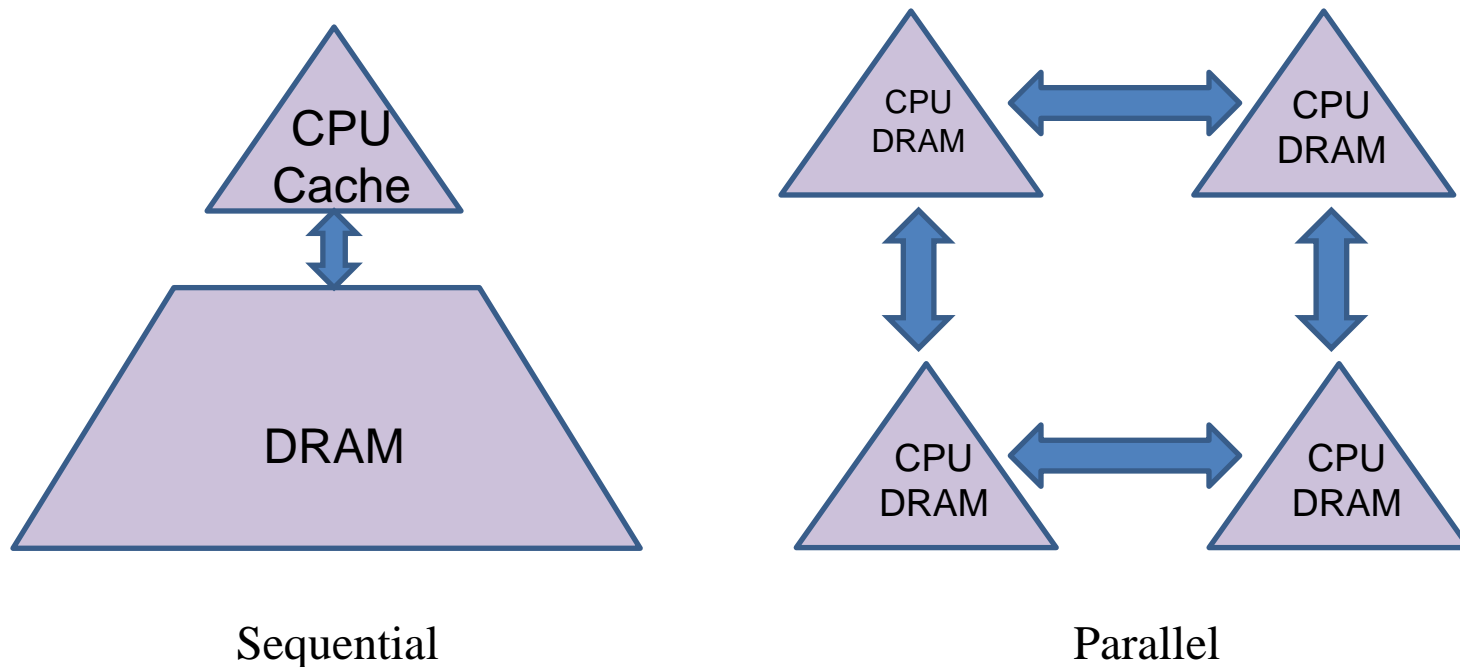
- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- Future Work

Talk Outline

- **What is communication?**
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- Future Work

What is “Communication”?

- Algorithms have 2 costs:
 - Arithmetic (FLOPS)
 - Movement of data
 - Two parameters: α – Latency, β – Reciprocal Bandwidth
 - Time to move n words of data is $\alpha + n\beta$



Communication in the future...

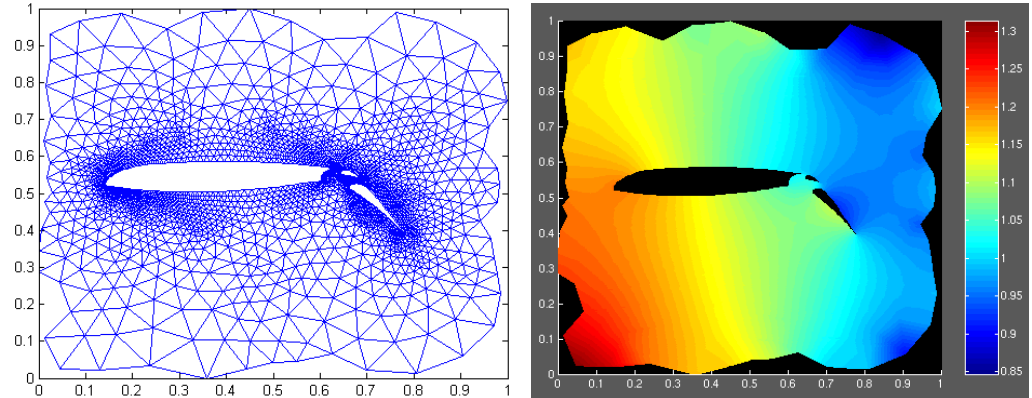
- Gaps growing exponentially...
 - Floating point time $\ll 1/\text{Network BW} \ll \text{Network Latency}$
 - **Improving 59%/year vs. 26%/year vs. 15%/year**
 - Floating point time $\ll 1/\text{Memory BW} \ll \text{Memory Latency}$
 - **Improving 59%/year vs. 23%/year vs. 5.5%/year**
- We want more than just “hiding” communication
 - Arbitrary speedups possible, vs. at most 2x speedup

Talk Outline

- What is communication?
- **What are Krylov Subspace Methods?**
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- Future Work

Motivation: Sparse Matrices

- Many algorithms for scientific applications require solving linear systems of equations: $Ax = b$



- In many cases, the matrix A is sparse
 - Sparse matrix: a matrix with enough zero entries to be worth taking advantage of
 - This means that information is “local” instead of “global”. A given variable only depends on some of the other variables.
 - Example: Simulating Pressure around Airfoil

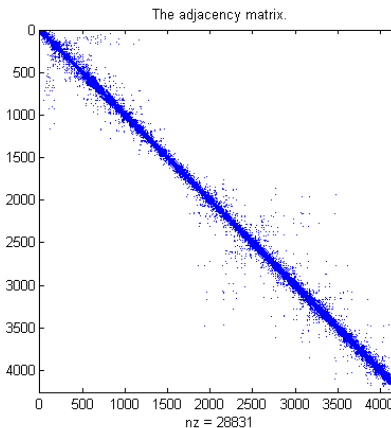
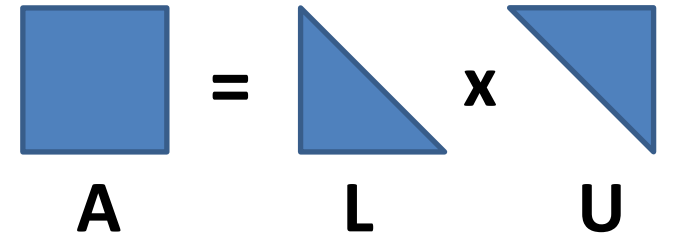


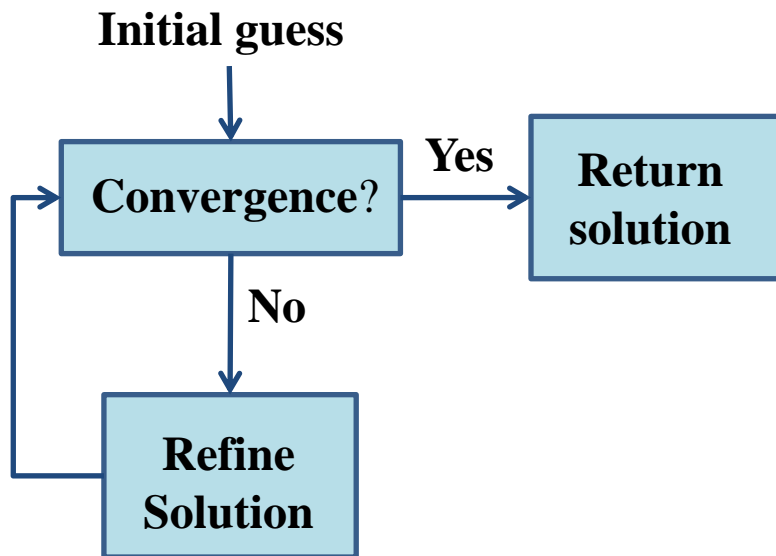
Figure: Simulating Pressure over Airfoil.
Source: <http://www.nada.kth.se>

Solving a Sparse Linear System

- **Direct** methods solve a linear system in a finite sequence of operations
 - Often used to solve **dense** problems
 - Ex: Gaussian Elimination



Direct Method for Solving $Ax = b$



Iterative Method for Solving $Ax = b$

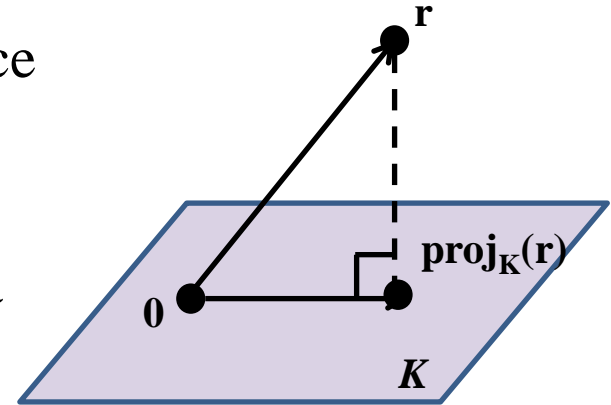
- **Iterative** methods iteratively refine an approximate solution to the system
 - Used when
 - System is large and sparse – direct method too expensive
 - We only need an approximation – don't need to solve exactly, so less operations needed
 - A is not explicitly stored
 - Ex: Krylov Subspace Methods (KSMs)

How do Krylov Subspace Methods Work?

- A Krylov Subspace is defined as:

$$\mathcal{K}_k(A, v) = \text{span}\{v, Av, A^2v, \dots, A^{k-1}v\}$$

- In each iteration,
 - Sparse matrix-vector multiplication (SpMV) with A to create new basis vector
 - Adds a dimension to the Krylov Subspace
 - Use vector operations to choose the “best” approximation of the solution in the expanding Krylov Subspace (projection of a vector onto a subspace)
 - How “best” is defined distinguishes different methods
- Examples: Conjugate Gradient (CG), Generalized Minimum Residual Methods (GMRES), Biconjugate Gradient (BiCG)



A Few Applications of Krylov Subspace Methods

- Image Processing Applications
 - Ex: Image segmentation, Contour detection

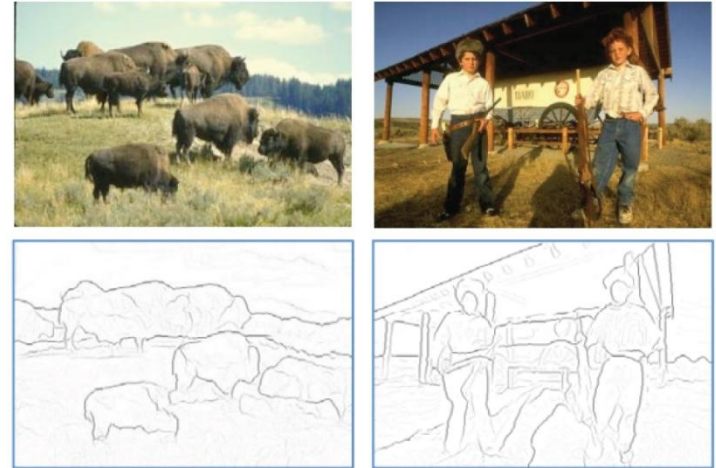


Figure: Contour Detection [CSNYMK10]

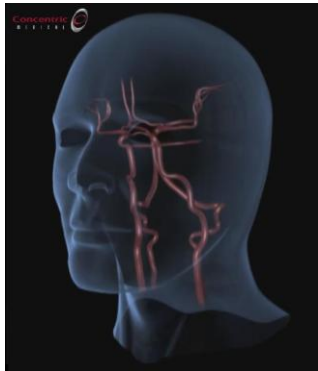


Figure: ParLab Health App:
Modeling Blood Flow in the
Brain

- Physical simulations
 - Solving PDEs
 - Often used in combination with Multigrid as bottom-solve
 - Ex: Simulating blood flow (Parlab's Health App)
- Mobile/Cloud applications
 - Even more important where bandwidth is very limited, latency is long (or if these parameters are variable between machines!)
 - Auto-tuning becomes more important if we don't know our hardware

Krylov Subspace Methods are Communication-Bound

- Problem: Calls to communication-bound kernels every iteration

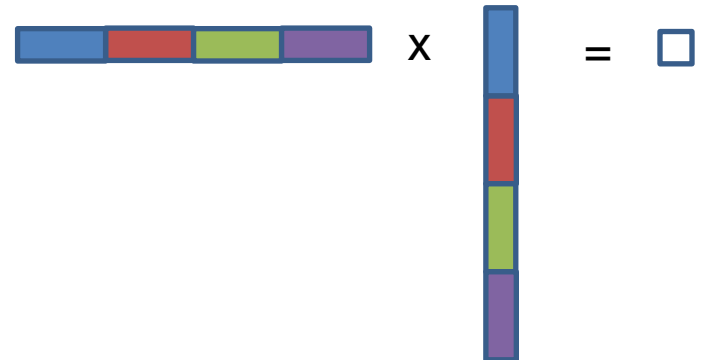
- SpMV (computing $A*v$)

- Parallel: share/communicate source vector with neighbors
- Sequential: read A (and vectors) from slow memory

- Vector operations

- Orthogonalization

- » Dot products
- » Vector addition and scalar multiplication



- Solution:

- Replace Communication-bound kernels by Communication-Avoiding ones
- Reformulate KSMs to use these kernels

Example: GMRES

Pseudocode to perform s steps of original algorithm:

- 1: **for** $k = 1$ to s **do**
- 2: $w = Av_{k-1}$
- 3: Orthogonalize w against v_0, \dots, v_{k-1} using Modified Gram-Schmidt
- 4: **end for**
- 5: Compute solution using H

SpMV operation in every iteration:
requires communication of current entries of v (parallel) / reading A and vectors from slow memory (sequential)

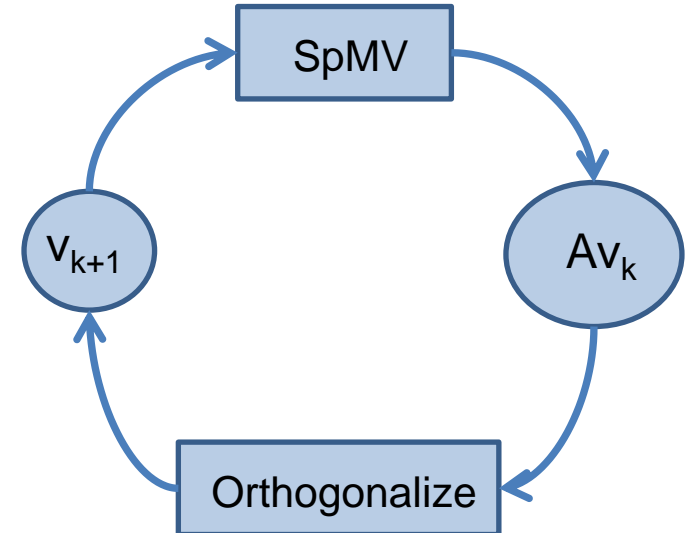
Vector operations in every iteration:
requires global communication (parallel) / reading $O(n)$ words from slow memory (sequential)

Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- **Communication-Avoiding Krylov Subspace Methods**
 - **New Communication-Avoiding Kernels**
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- Future Work

Communication-Avoiding KSMs

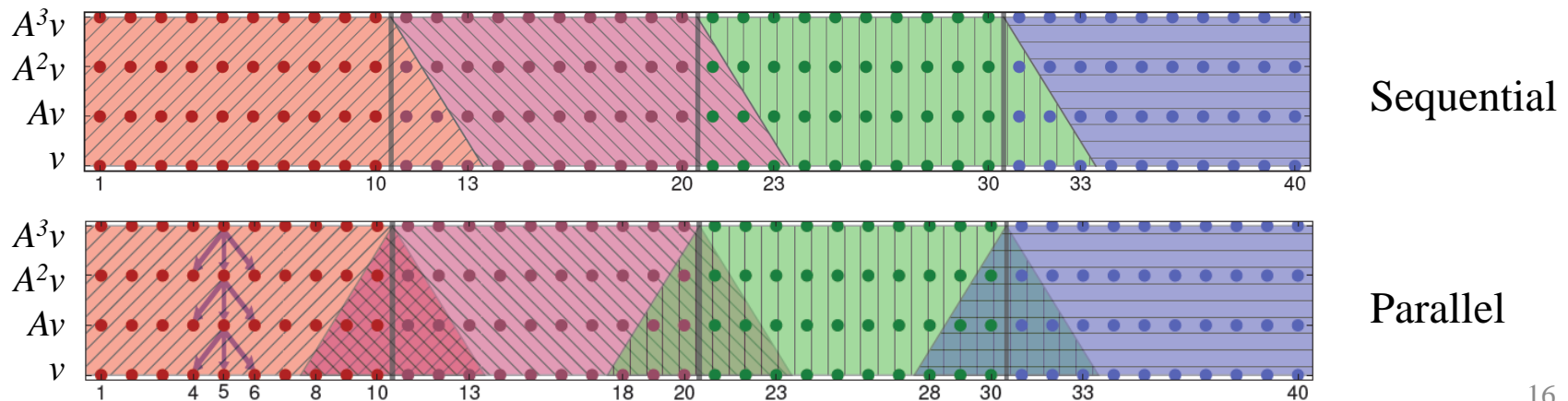
- We need to break the dependency between communication bound kernels and KSM iterations
- Idea: Expand the subspace s dimensions (s SpMVs with A), then do s steps of refinement
 - unrolling the loop s times
- To do this we need two new Communication-Avoiding kernels
 - “Matrix Powers Kernel” replaces SpMV
 - “Tall Skinny QR” (TSQR) replaces orthogonalization operations



The Matrix Powers Kernel

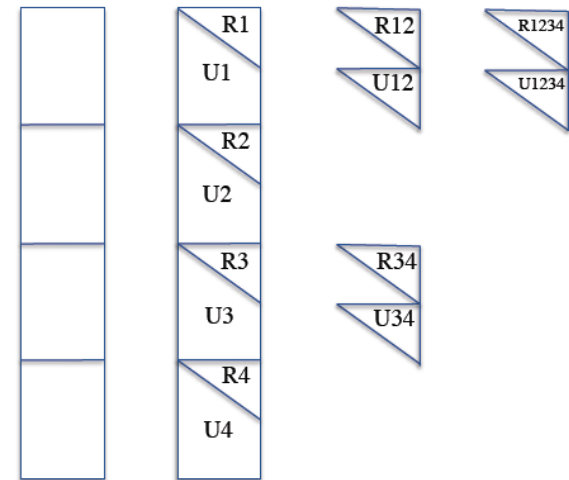
- Given A , v , and s , Matrix powers kernel computes

$$\{v, Av, A^2v, \dots, A^{s-1}v\}$$
- If we figure out dependencies beforehand, we can do all the communication for s steps of the algorithm only reading/communicating A $o(1)$ times!
 - Parallel case: Reduces latency by a factor of s at the cost of redundant computations
 - Sequential case: reduces latency and bandwidth by a factor of s , no redundant computation
- Simple example: a tridiagonal matrix



Communication Avoiding Kernels: TSQR

- TSQR = Tall Skinny QR ($\#rows \gg \#cols$)
 - QR: factors a matrix A into the product of
 - An orthogonal matrix (Q)
 - An upper triangular matrix (R)
 - Here, A is the matrix of the Krylov Subspace Basis Vectors
 - output of the matrix powers kernel
 - Q and R allow us to easily expand the dimension of the Krylov Subspace
- Usual Algorithm
 - Compute Householder vector for each column $O(n \log P)$ messages
- Communication Avoiding Algorithm
 - Reduction operation, with QR as operator $O(\log P)$ messages



- Shape of reduction tree depends on architecture
 - Parallel: use “deep” tree, saves messages/latency
 - Sequential: use flat tree, saves words/bandwidth
 - Multicore: use mixture

Example: CA-GMRES

s steps of original algorithm:

- 1: **for** $k = 1$ to s **do**
- 2: $w = Av_{k-1}$
- 3: Orthogonalize w against v_0, \dots, v_{k-1} using Modified Gram-Schmidt
- 4: **end for**
- 5: Compute solution using H

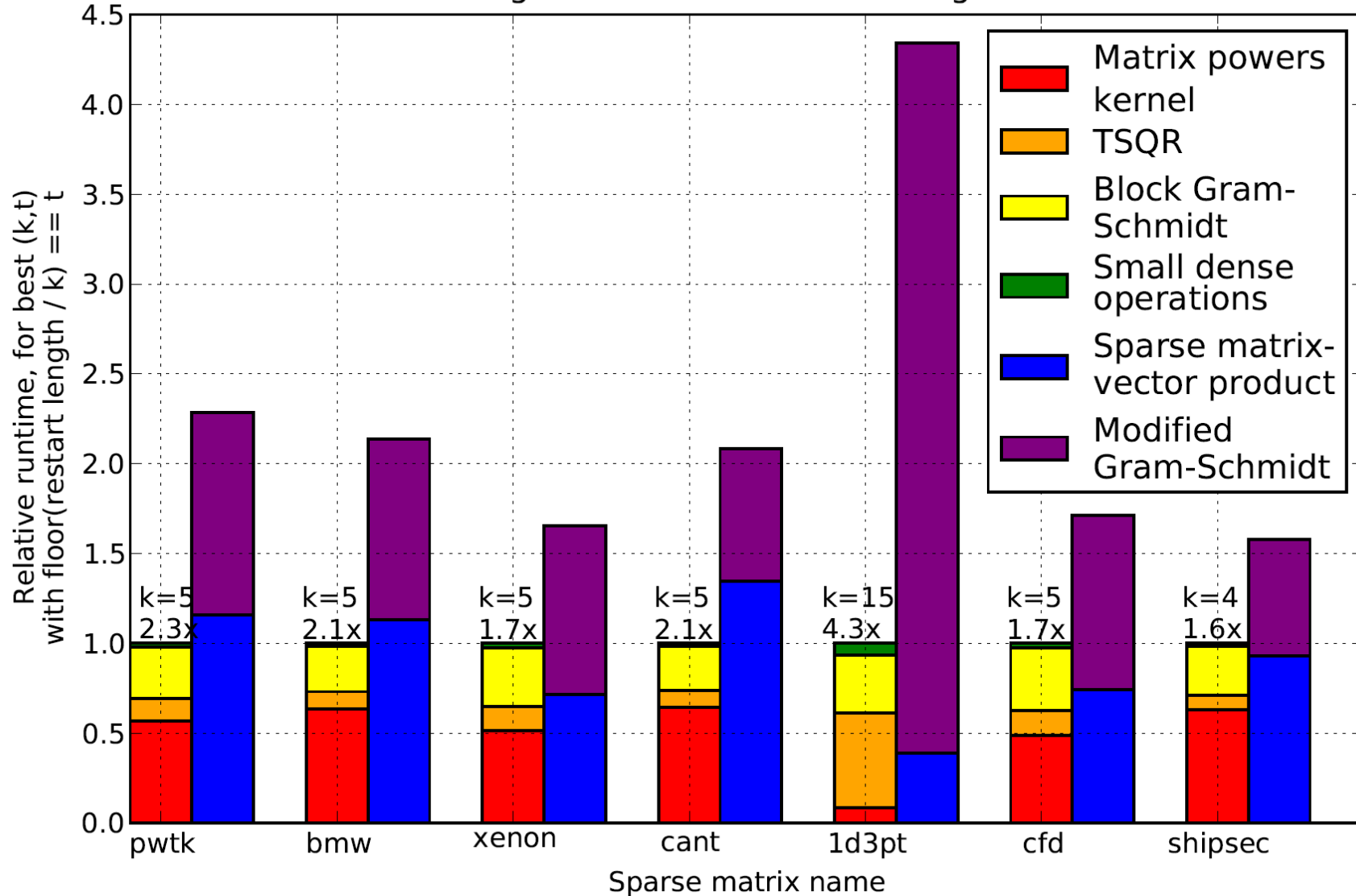
s steps of CA algorithm:

- 1: $W = [v_0, Av_0, A^2v_0, \dots, A^s v_0]$
- 2: $[Q, R] = TSQR(W)$
- 3: Compute H using R
- 4: Compute solution using H

s powers of A for no extra
latency cost

s steps of QR for one step of
latency

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60



Current CA Krylov Subspace Methods

- CG, Lanczos, Arnoldi (Hoemmen, 2010),
- GMRES (Hoemmen, Mohiyuddin, Demmel, Yelick, 2009)
- BiCG, CGS, BiCGStab (Carson, Knight, Demmel, 2011).

- Factor of s less communication than standard version.

- General approach for CG-like methods:
 - In each outer loop, compute s basis vectors from previous iteration's residual vectors
 - Perform s inner loop iterations
 - Compute current recurrence coefficients
 - Replace SpMVs with local basis vector operations
 - Replace dot products with shorter, local dot products
 - continue until convergence....

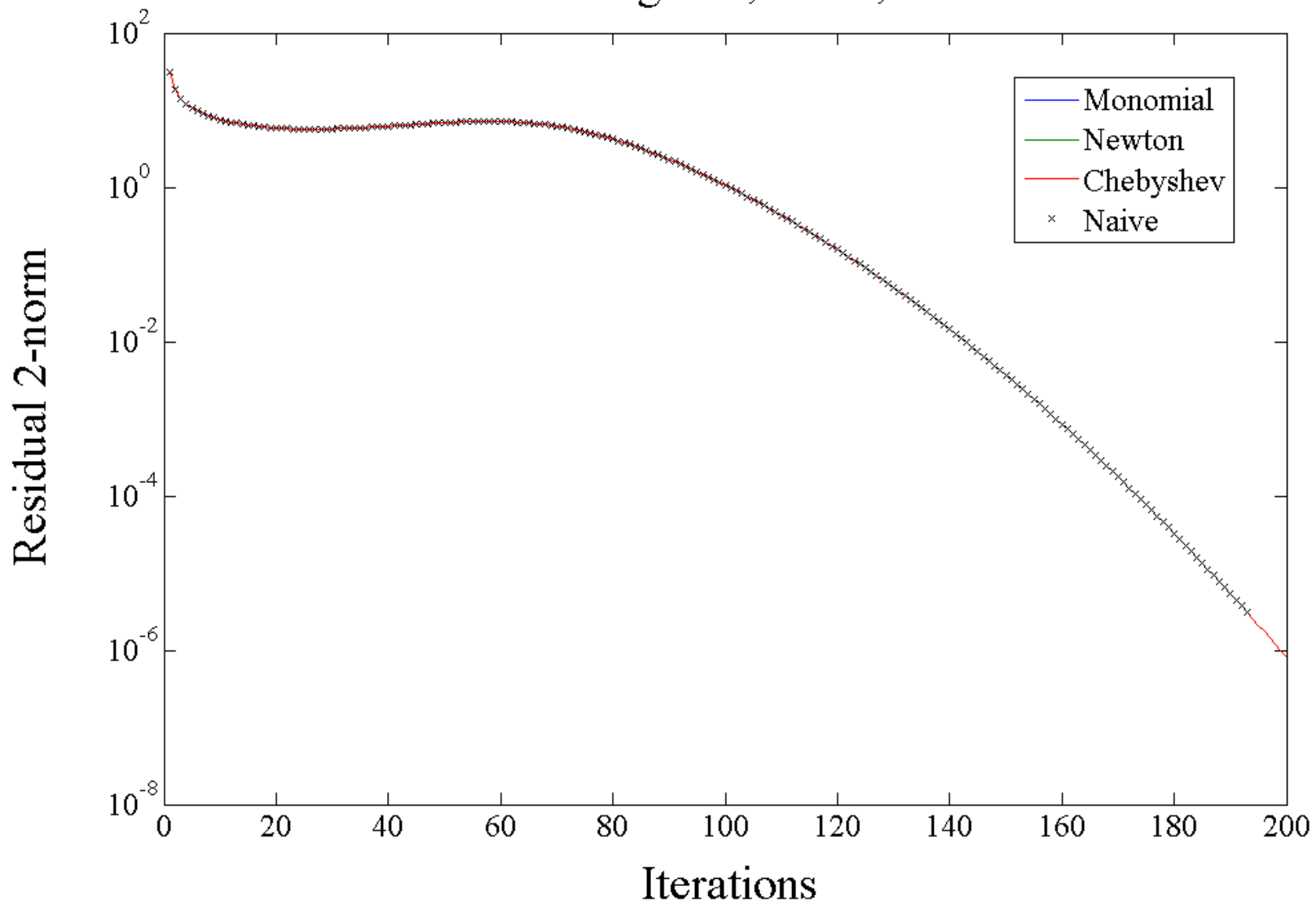
Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- **Challenges in Communication-Avoiding Krylov Subspace Methods**
 - **Stability and Convergence**
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- Future Work

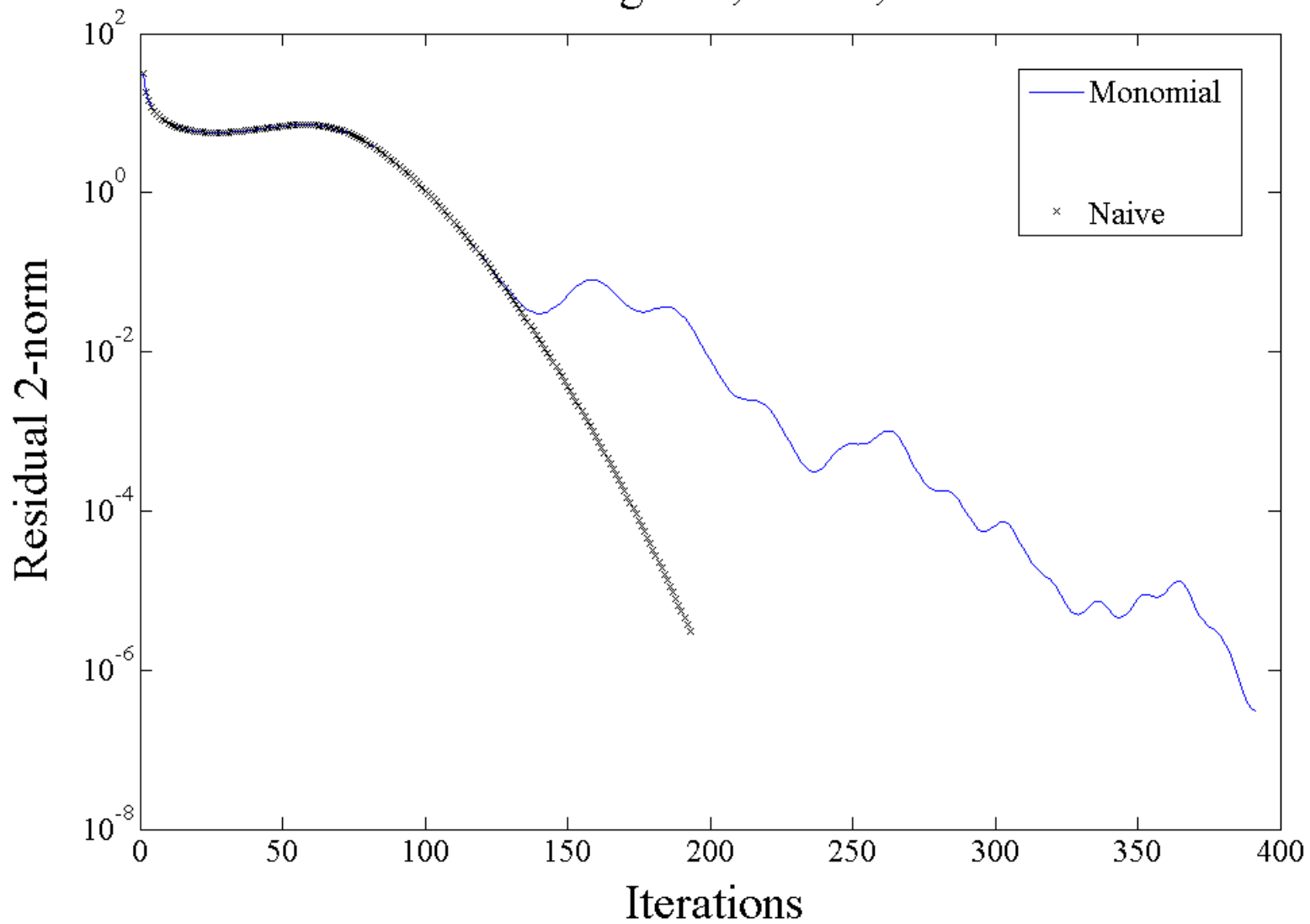
Challenges: Stability and Convergence

- **Stability** of Communication-Avoiding Krylov Subspace Methods **depends on s**
- Does v, Av, A^2v, \dots look familiar?
 - Power Method! Converges to principle eigenvector
 - Expected linear dependence of basis vectors
 - Means the Krylov Subspace can't expand any more – method breaks down, convergence stalls
- Can we remedy this problem to remain stable for larger s values?
 - Yes! Other possible basis choices:
 - Newton $[v, (A - \theta_1 I)v, (A - \theta_2 I)(A - \theta_1 I)v, \dots]$
 - Chebyshev $[v, T_1(v), T_2(v), \dots]$

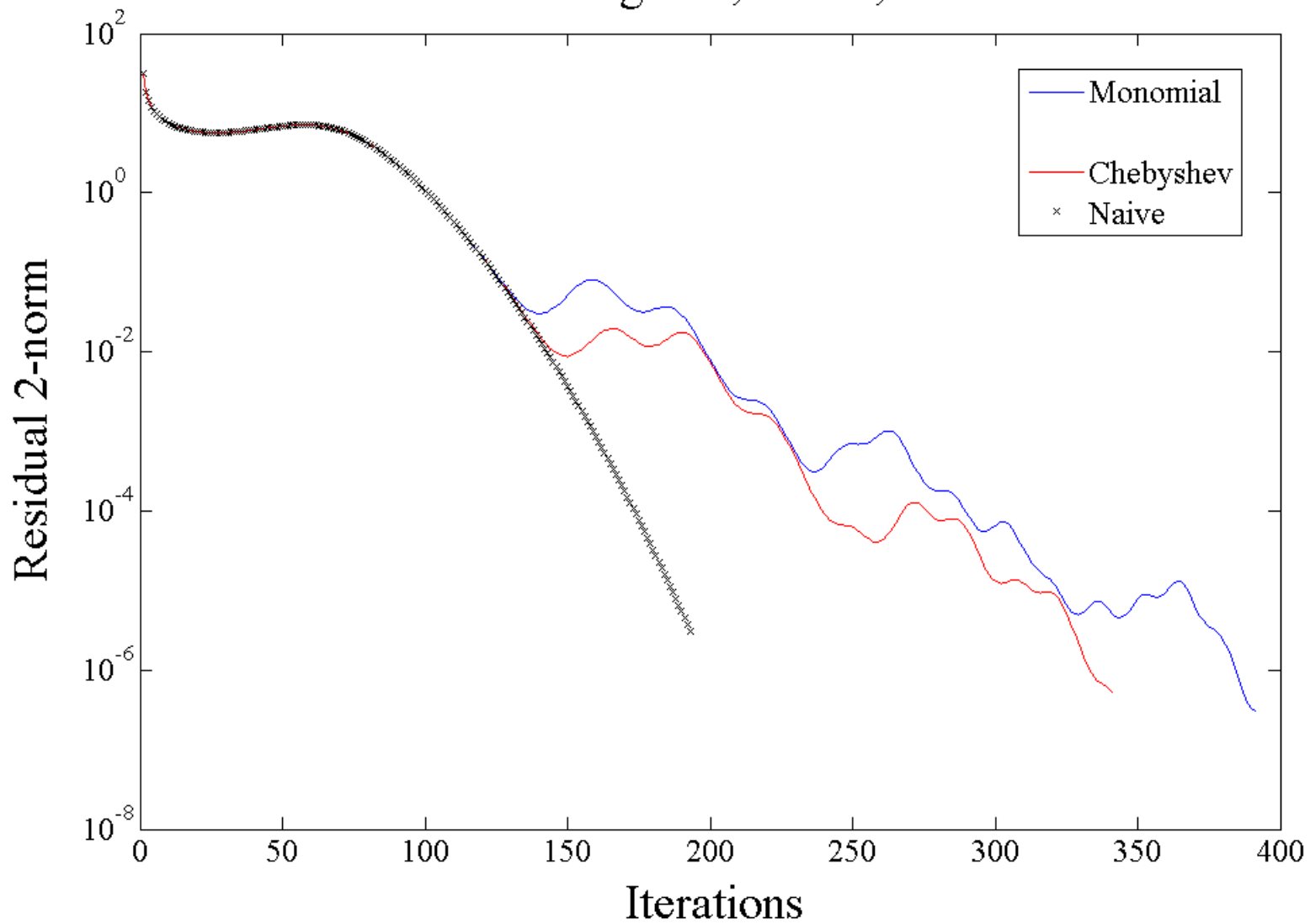
CABiCG Convergence, $s = 4$, Cond# $\sim 1e4$



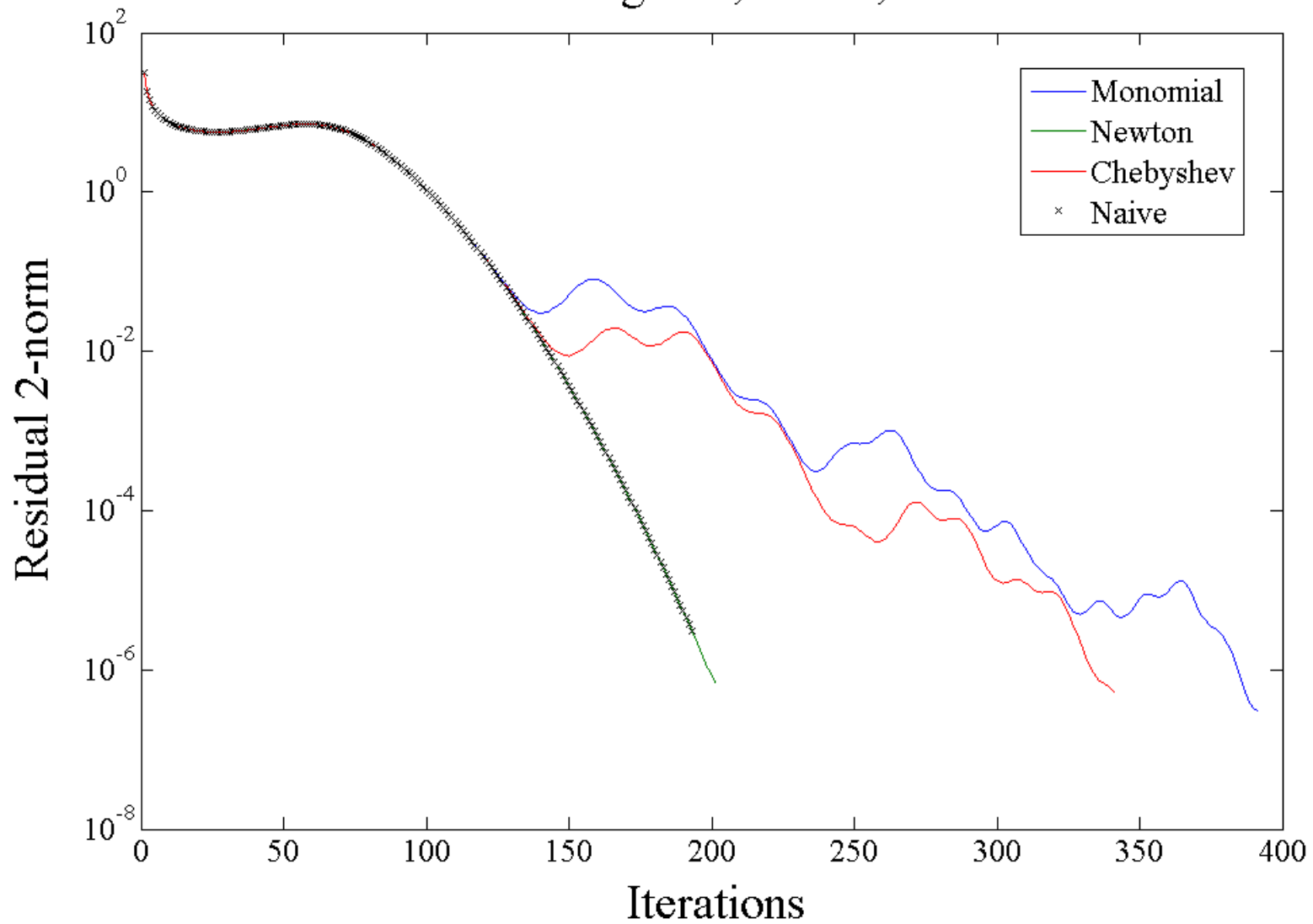
CABiCG Convergence, $s = 10$, $\text{Cond\#} \sim 1e4$



CABiCG Convergence, $s = 10$, $\text{Cond\#} \sim 1e4$



CABiCG Convergence, $s = 10$, $\text{Cond\#} \sim 1e4$



Summary of Preliminary Results

- Our CA variants (generally) maintain stability for s in between 2 and 10
 - Which basis (Monomial, Newton, or Chebyshev) is most effective depends on the specific Krylov method we use and the condition number of A (and other spectral properties of A)
 - Reduces communication costs by a factor of s
 - So, if $s = 10$, possible speedup is 10x!
- In general, as s increases, the number of iterations needed to converge increases, and after a certain point, the method breaks down
 - Could be remedied by preconditioning, extended precision, etc.
- Must choose s to maintain stability

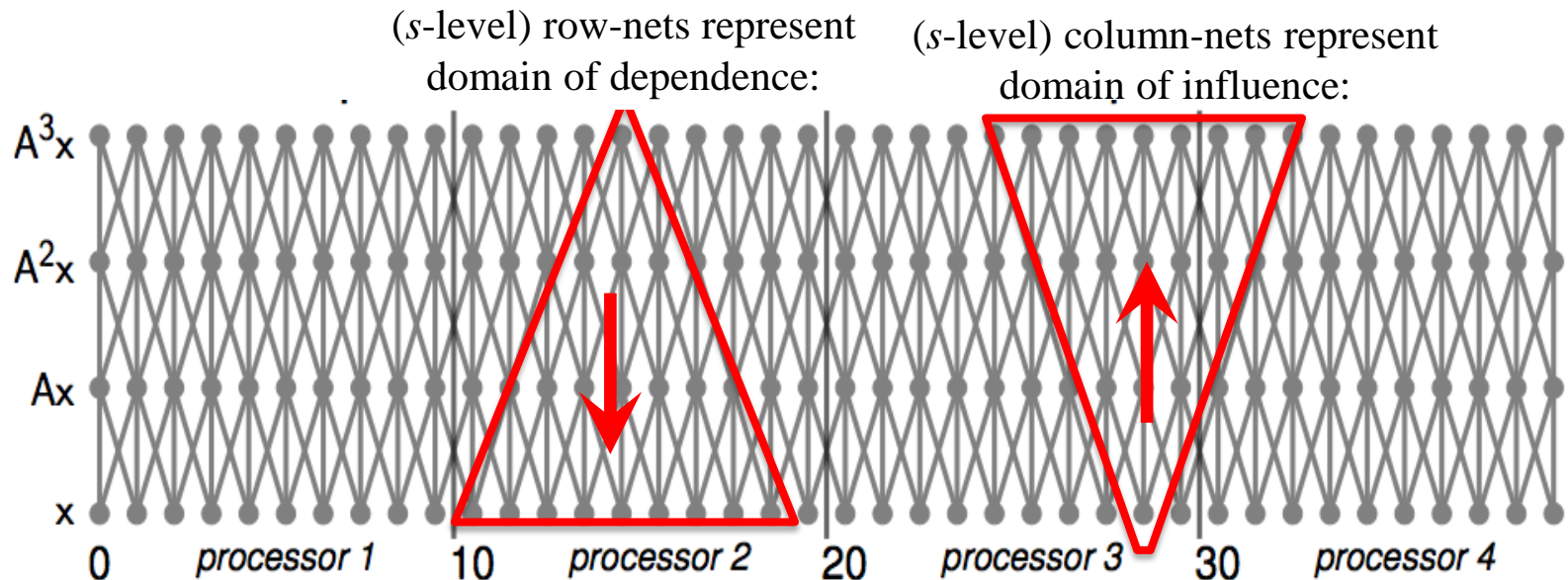
Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- **Challenges in Communication-Avoiding Krylov Subspace Methods**
 - Stability and Convergence
 - **Performance**
- Preconditioning
- Related Work: “s-step methods”
- Future Work

Challenges: Performance

- How to choose s ?
 - Assuming that stability is not an issue...
 - After some value of s , the matrix is too dense to avoid communication using the Matrix Powers Kernel
 - But exactly computing this value of s requires computing the matrix powers!
- How to partition the matrix for $A^s x$?
 - As above, computing dependencies requires computing matrix powers
 - The redundant work (“ghost zones”) are induced by the partition. So how can we achieve load balance?

Partitioning for CA-KSMs



Parallel communication for $A^s x(A, s, x) = [x, Ax, A^2x, \dots, A^s x]$, given overlapping partition of A

=

Parallel communication for $y = A^s x$, given 1D rowwise layout of A^s

(assuming no cancellation and nonzero diagonal)

- Minimizing communication in matrix powers reduces to hypergraph partitioning s-level column-nets.
- **Problem:** Computational and storage cost:
 - $s \times$ Boolean sparse matrix-matrix multiplies!

Partitioning for CA KSMs

- Solution: Use reachability estimation [Cohen '94]
 - $O(\text{nnz})$ time randomized algorithm for estimating size of transitive closure.
 - Calculating transitive closure costs $O(n*\text{nnz})$
- Can be used to estimate nnz-per-column in matrix product A^s in $O(\text{nnz})$ time
 - Can be used to **sparsify** the hypergraph – Drop large nets during construction
 - Reduces size of data structure and computational cost, while still providing a good partition
- Can be used to estimate **overlap** between columns – the number of nonzero rows two column have in common
 - This could allow us to heuristically load balance

Challenges: Performance for Stencil-like Matrices

- What if A is stencil-like (in general, $O(n)$ cost to read)?
 - In the sequential algorithm...
 - Not communication-bound due to reading A , but...
 - Communication bottleneck is now reading Krylov vectors
 - $O(kn)$ cost to read Krylov basis vectors every k steps
- Can we reduce the communication cost of k steps from $O(kn)$ to $O(n)$?

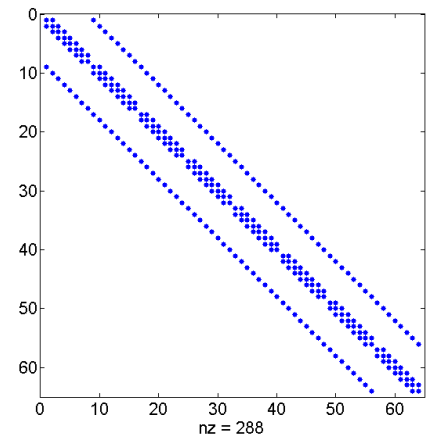
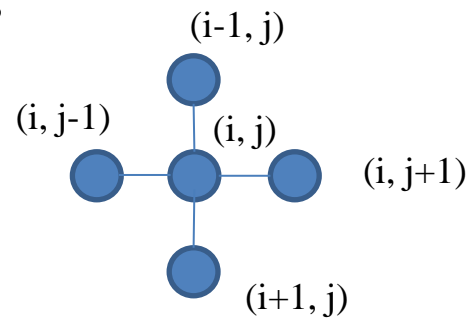
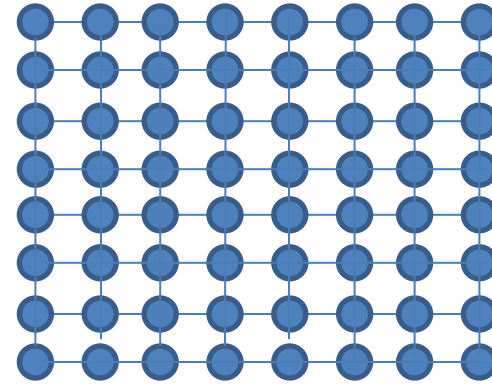
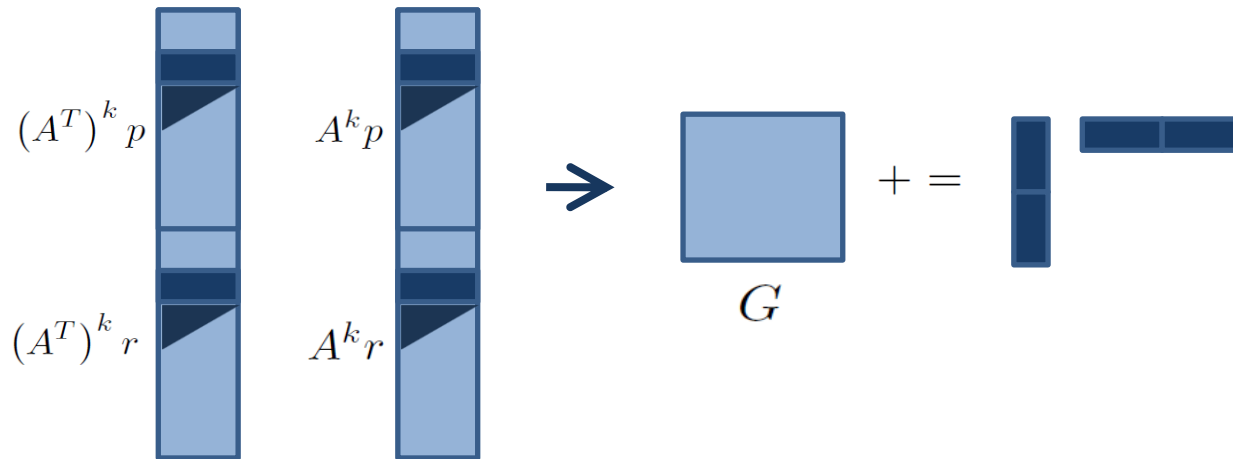


Figure: 2D 5-point stencil. Each grid-point is updated at each time-step using only nearest neighbor values

Streaming Matrix Powers Computation

- Idea: Don't explicitly store basis vectors
 - *Streaming* Matrix Powers: Interleave matrix powers computation and construction of the Gram Matrix G
 - Part i computes $G += V_i^T V_i$, discards V_i



- Tradeoff: requires two matrix powers invocations, but bandwidth reduced by a factor of k
 - OK if reading and applying A is inexpensive (e.g., stencil, AMR base case, others?)
- Overall communication reduced from $O(kn)$ to $O(n)$!

Auto-tuning for CA-KSMs

- Auto-tuning for stability
 - Choice of basis to use
 - Depends on s , condition number of A , method, etc.
- Auto-tuning for performance
 - Partitioning A amongst parallel processors to minimize communication
 - Partitioning for cache blocking to maximize cache reuse
 - Determine which variant of the matrix powers kernel to use
 - E.g., “streaming” if A is stencil-like
 - Many other standard parallel and sequential optimizations...
- Eventually will be built into pOSKI (Parallel Optimized Sparse Kernel Interface), an auto-tuning library for sparse matrix computations

Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- **Preconditioning**
- Related Work: “s-step methods”
- Future Work

What is preconditioning?

- The number of iterations a KSM takes to converge depends on the “condition number” of A
 - Condition number is a property of a matrix/system (not of the algorithm or precision used)
 - For $Ax=b$, roughly denotes how error in b affects error in x
 - The lower the condition number, the fewer iterations needed for convergence
- Preconditioning: Instead of solving $Ax=b$, solve $(MA)x = Mb$, where the matrix MA has a lower condition number than A
 - Many methods exist for finding a matrix M which has this property
 - “Sparse Approximate Inverse”, “Incomplete LU”, “Polynomial Preconditioning”, etc.
- This technique is used in almost all practical applications of KSMs

What About Preconditioning in CA-KSMs?

- Problem: CA preconditioning approach requires a different approach/implementation for each type of preconditioner!
- Existing algorithms
 - Polynomial preconditioners (Saad, Toledo)
 - M is polynomial in A – easily incorporated into Matrix Powers Kernel
 - CA-Left and Right preconditioning (Hoemmen, 2010)
 - For 2 non-trivial classes of preconditioners
 - $1 + o(1)$ more messages than single SpMV, 1 preconditioner solve
 - Tradeoff: computation cost increases significantly
 - Can require twice as many flops as s SPMVs!

Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- **Related Work: “s-step methods”**
- Future Work

Related Work: s-step Methods

Author	Algorithm	Basis	Preconditioning	Matrix Powers?	TSQR?
Van Rosendale, 1983	CG	Monomial	Polynomial	No	-
Leland, 1989	CG	Monomial	Polynomial	No	-
Walker, 1988	GMRES	Monomial	None	No	No
Chronopoulos and Gear, 1989	CG	Monomial	None	No	-
Chronopoulos and Kim, 1990	Orthomin, GMRES	Monomial	None	No	No
Chronopoulos, 1991	MINRES	Monomial	None	No	No
Kim and Chronopoulos, 1991	Symm. Lanczos, Arnoldi	Monomial	None	No	No
Sturler, 1991	GMRES	Chebyshev	None	No	No

Related Work, contd.

Author	Algorithm	Basis	Preconditioning	Matrix Powers?	TSQR?
Joubert and Carey, 1992	GMRES	Chebyshev	None	Yes (stencil only)	No
Chronopoulos and Kim, 1992	Nonsymm . Lanczos	Monomial	None	No	-
Bai, Hu, and Reichel, 1991	GMRES	Newton	None	No	No
Erhel, 1995	GMRES	Newton	None	No	No
De Sturler and van der Vorst, 2005	GMRES	Chebyshev	General	No	No
Toledo, 1995	CG	Monomial	Polynomial	Yes (stencil only)	-
Chronopoulos and Swanson, 1990	CGR, Orthomin	Monomial	None	No	-
Chronopoulos and Kinkaid, 2001	Orthodir	Monomial	None	No	-

Talk Outline

- What is communication?
- What are Krylov Subspace Methods?
- Communication-Avoiding Krylov Subspace Methods
 - New Communication-Avoiding Kernels
- Challenges in Communication-Avoiding Krylov Subspace Methods
 - Stability and Convergence
 - Performance
- Preconditioning
- Related Work: “s-step methods”
- **Future Work**

Future Work

- Other CA Krylov Subspace methods?
- Evaluate current preconditioning methods
 - and extend CA approach to other classes of preconditioners
- Parallel Implementations
 - Performance tests
- Improving stability
 - Extended precision
- Auto-tuning work
 - Incorporation of Matrix Powers into pOSKI (Jong-Ho Byun, et al., UCB)
 - Code generation for Matrix Powers (collaborating with Ras Bodik, Michelle Strout)
 - Exploring co-tuning for CA-KSMS (i.e., Matrix Powers and TSQR)
- Looking forward: how do Communication-Avoiding algorithms relate to energy efficiency?

Thank you!

Questions?

Extra Slides

CA-BiCG

For $k = 0, 1, \dots$, until convergence, Do

Compute four $N \times (s+1)$ matrices via 2×2 matrix powers kernel, and change-of-basis matrix B

$$\begin{aligned} P &= [p_{sk}, Ap_{sk}, \dots, A^s p_{sk}] \\ R &= [r_{sk}, Ar_{sk}, \dots, A^s r_{sk}] \\ P^* &= [p_{sk}^*, A^T p_{sk}^*, \dots, (A^T)^s p_{sk}^*] \\ R^* &= [r_{sk}^*, A^T r_{sk}^*, \dots, (A^T)^s r_{sk}^*] \end{aligned}$$

Compute the $(2s+2) \times (2s+2)$ Gram matrix

$$G = \begin{bmatrix} (P^*)^T \\ (R^*)^T \end{bmatrix} [P, R]$$

Compute the $(2s+2) \times (s+1)$ coefficient matrices

$$c_{sk} = \begin{bmatrix} 1 & B \\ 0_{s \times 1} & \\ 0_{(s+1) \times (s+1)} & \end{bmatrix}, \quad d_{sk} = \begin{bmatrix} 0_{(s+1) \times (s+1)} \\ 1 \\ 0_{s \times 1} & B \end{bmatrix}$$

For $j = 0$ to $s-1$, Do

$$\begin{aligned} \alpha_{sk+j} &= \frac{(d_{sk+j}^0)^T G d_{sk+j}^0}{(c_{sk+j}^0)^T G c_{sk+j}^1} \\ x_{sk+j+1} &= x_{sk+j} + \alpha_{sk+j} p_{sk+j} \\ r_{sk+j+1} &= r_{sk+j} - \alpha_{sk+j} [P, R] c_{sk+j}^1 \\ r_{sk+j+1}^* &= r_{sk+j}^* - \alpha_{sk+j} [P^*, R^*] c_{sk+j}^1 \\ d_{sk+j+1} &= d_{sk+j}^{0:(s-j-1)} - \alpha_{sk+j} c_{sk+j}^{1:(s-j)} \\ \beta_{sk+j} &= \frac{(d_{sk+j+1}^0)^T G d_{sk+j+1}^0}{(d_{sk+j}^0)^T G d_{sk+j}^0} \\ p_{sk+j+1} &= r_{sk+j+1} + \beta_{sk+j} p_{sk+j} \\ p_{sk+j+1}^* &= r_{sk+j+1}^* + \beta_{sk+j} p_{sk+j}^* \\ c_{sk+j+1} &= d_{sk+j+1} + \beta_{sk+j} c_{sk+j}^{0:(s-j-1)} \end{aligned}$$

EndDo

EndDo

For $j = 0, 1, \dots$, until convergence, Do

$$\begin{aligned} \alpha_j &= \frac{\langle r_j, r_j^* \rangle}{\langle Ap_j, p_j^* \rangle} \\ x_{j+1} &= x_j + \alpha_j p_j \\ r_{j+1} &= r_j - \alpha_j Ap_j \\ r_{j+1}^* &= r_j^* - \alpha_j A^T p_j^* \\ \beta_j &= \frac{\langle r_{j+1}, r_{j+1}^* \rangle}{\langle r_j, r_j^* \rangle} \\ p_{j+1} &= r_{j+1} + \beta_j p_j \\ p_{j+1}^* &= r_{j+1}^* + \beta_j p_j^* \end{aligned}$$

EndDo

ALGORITHM 7.3: Biconjugate Gradient (BCG)

1. Compute $r_0 := b - Ax_0$. Choose r_0^* such that $(r_0, r_0^*) \neq 0$.
2. Set, $p_0 := r_0, p_0^* := r_0^*$
3. For $j = 0, 1, \dots$, until convergence Do:
 4. $\alpha_j := (r_j, r_j^*) / (Ap_j, p_j^*)$
 5. $x_{j+1} := x_j + \alpha_j p_j$
 6. $r_{j+1} := r_j - \alpha_j Ap_j$
 7. $r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
 8. $\beta_j := (r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$
 9. $p_{j+1} := r_{j+1} + \beta_j p_j$
 10. $p_{j+1}^* := r_{j+1}^* + \beta_j p_j^*$
11. EndDo

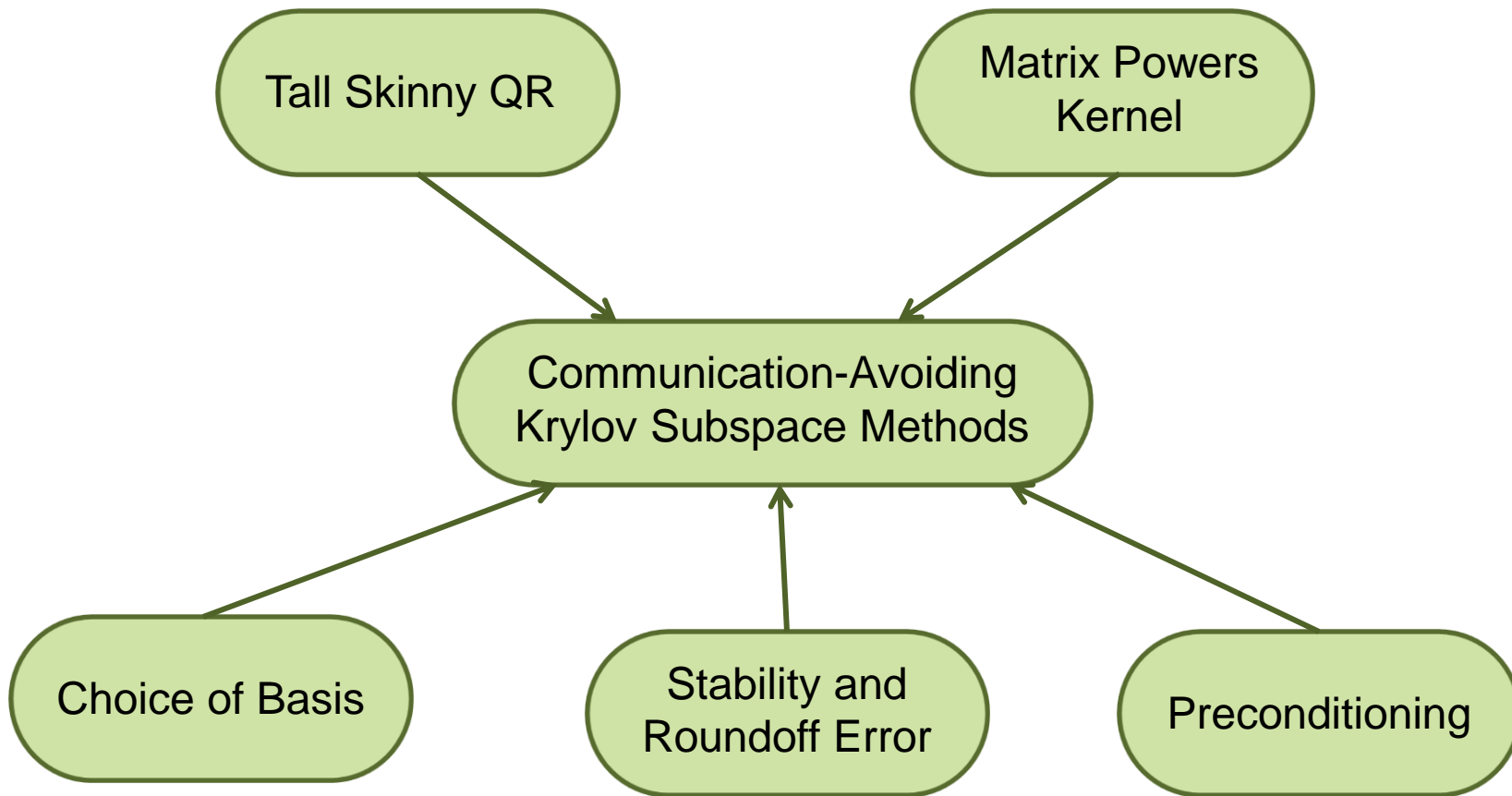
ALGORITHM 7.5: Conjugate Gradient Squared

1. Compute $r_0 := b - Ax_0$; r_0^* arbitrary.
2. Set $p_0 := u_0 := r_0$.
3. For $j = 0, 1, 2, \dots$, until convergence Do:
4. $\alpha_j = (r_j, r_0^*) / (Ap_j, r_0^*)$
5. $q_j = u_j - \alpha_j Ap_j$
6. $x_{j+1} = x_j + \alpha_j (u_j + q_j)$
7. $r_{j+1} = r_j - \alpha_j A(u_j + q_j)$
8. $\beta_j = (r_{j+1}, r_0^*) / (r_j, r_0^*)$
9. $u_{j+1} = r_{j+1} + \beta_j q_j$
10. $p_{j+1} = u_{j+1} + \beta_j (q_j + \beta_j p_j)$
11. EndDo

ALGORITHM 7.6: BICGSTAB

1. Compute $r_0 := b - Ax_0$; r_0^* arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \dots$, 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

(Algorithms)



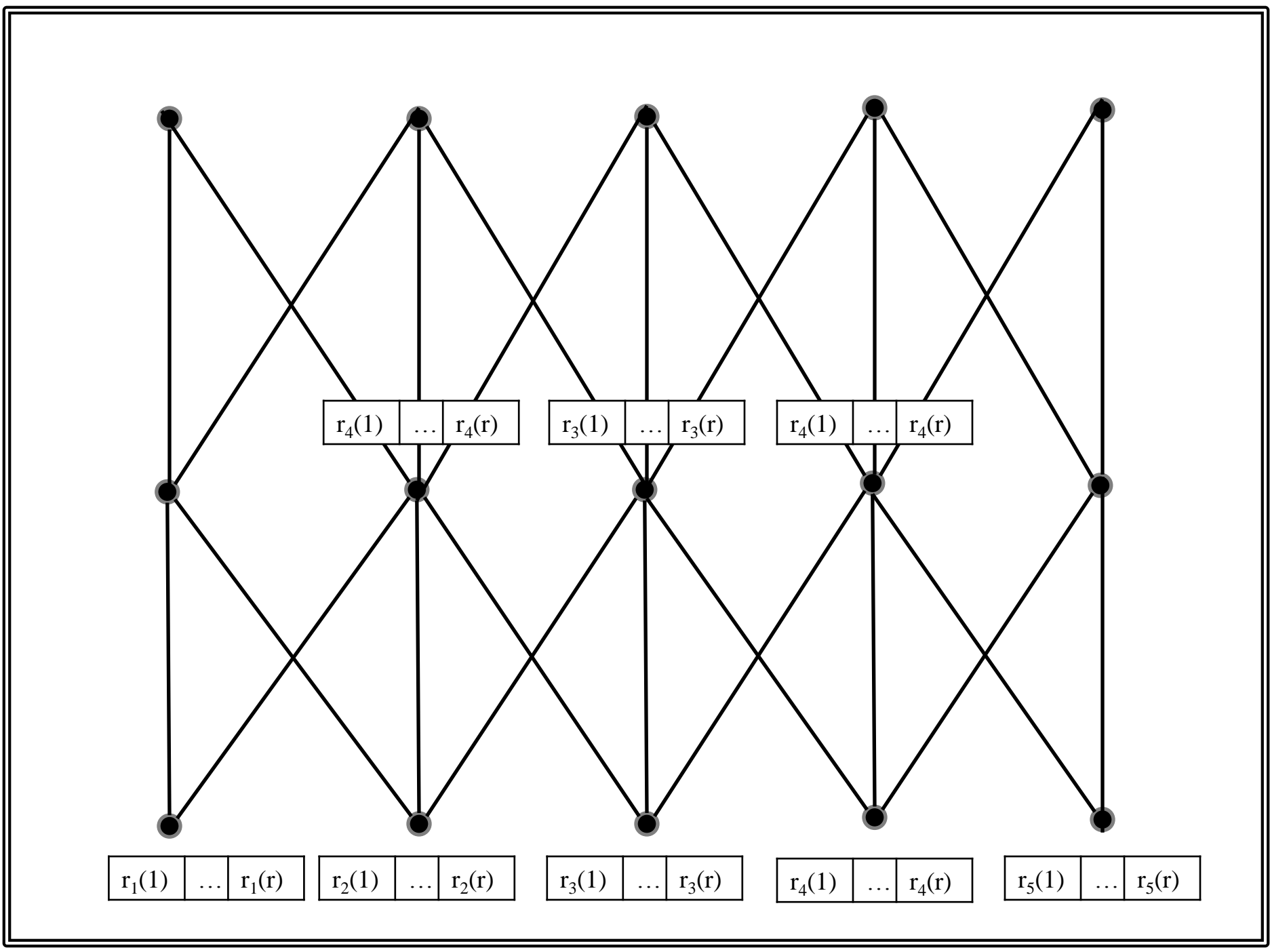
(Numerical Analysis)

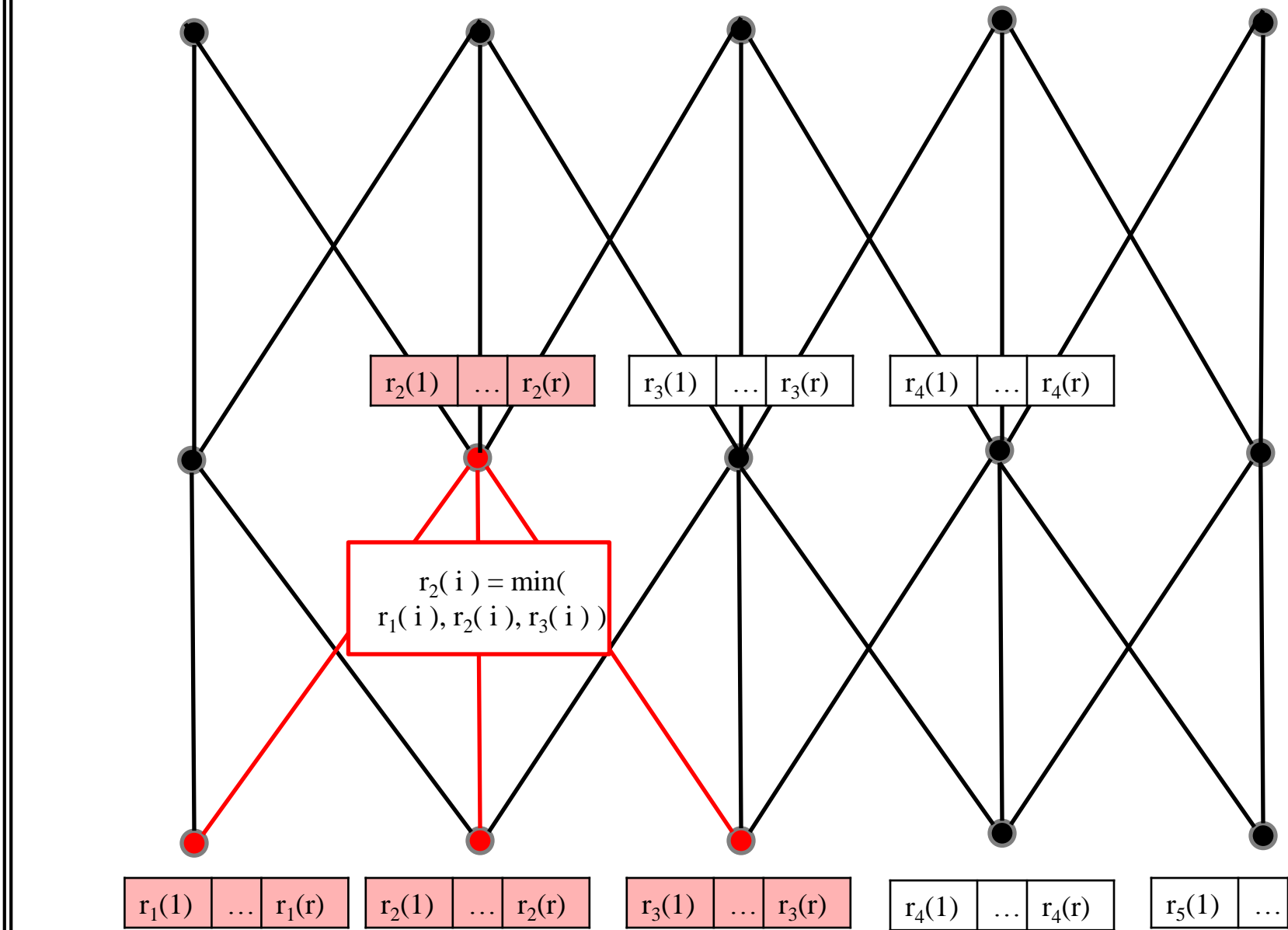
Algorithm Overview

- Initially assign r -vector of rankings (a_1, \dots, a_r) , (sampled from exponential R.V., $\lambda = 1$) to each vertex v
- In each iteration (up to s), for each vertex v , take the coordinate-wise minima of the r -vectors reachable from v (denoted $S(v)$, non-zeros in column of A corresponding to v)
- Apply estimator:
$$\frac{r-1}{\sum_{i=1}^r a_i}$$
- Intuition: lowest-ranked node in $S(v)$ is highly correlated with $|S(v)|$
 - Example: If $S(v)$ contains half the nodes, we expect the lowest rank of nodes in $S(v)$ is very small.

$$\text{Prob}(|\hat{T} - T| \geq \epsilon T) = O\left(\frac{1}{\epsilon\sqrt{r}}\right)$$

where T is the actual size of the transitive closure, r is the number of randomized rankings per vector





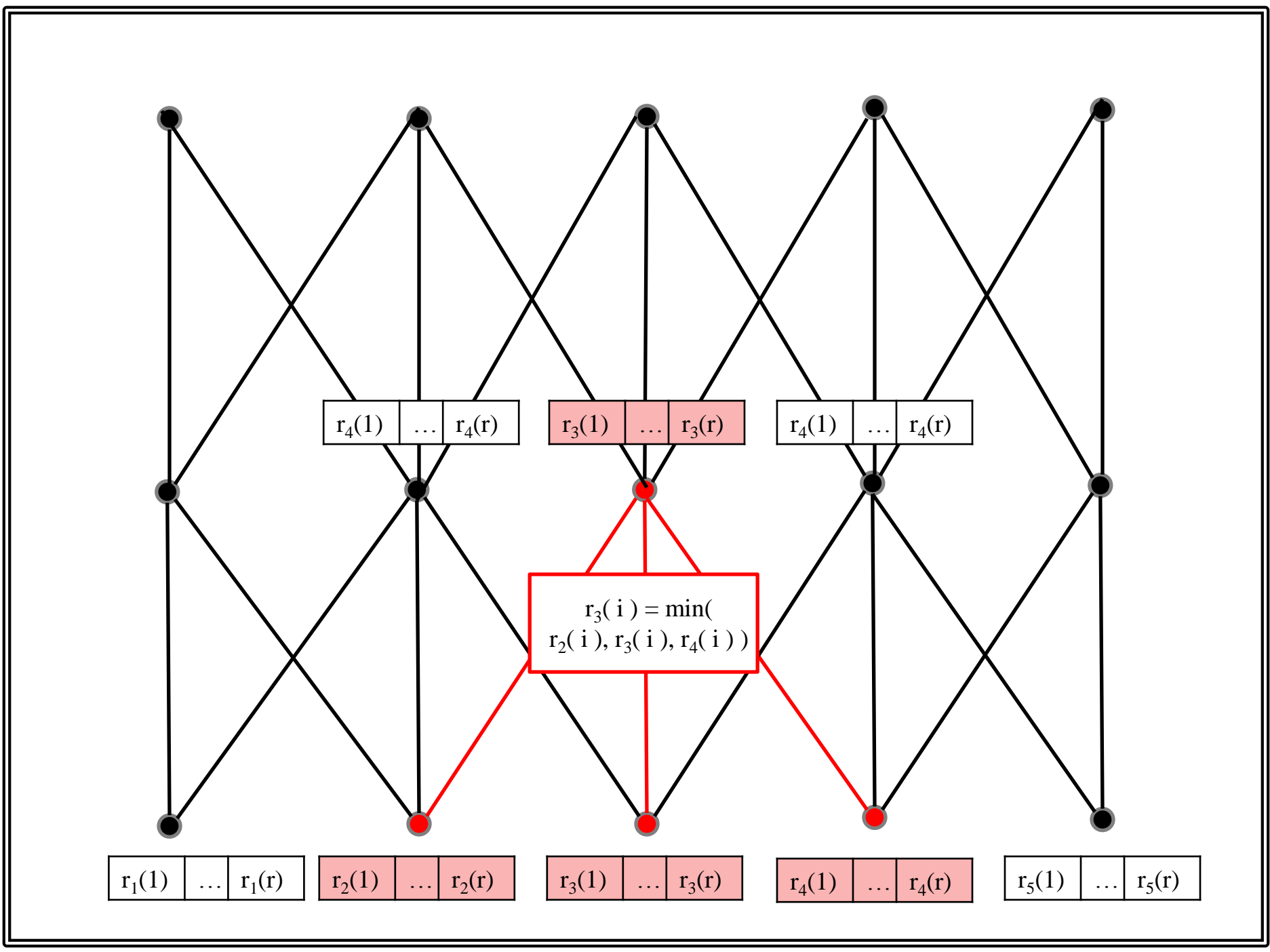
$r_1(1) \quad \dots \quad r_1(r)$

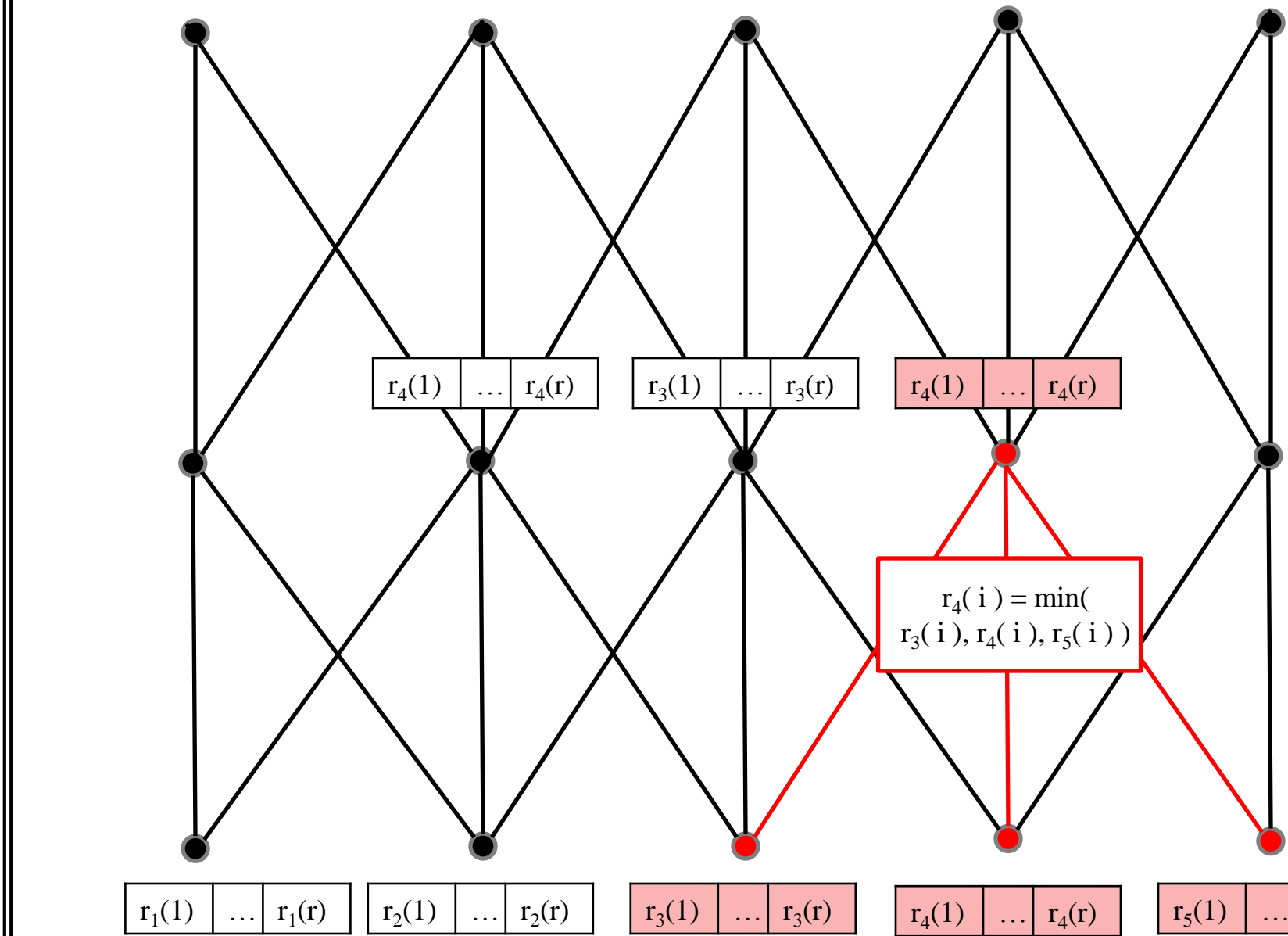
$r_2(1) \quad \dots \quad r_2(r)$

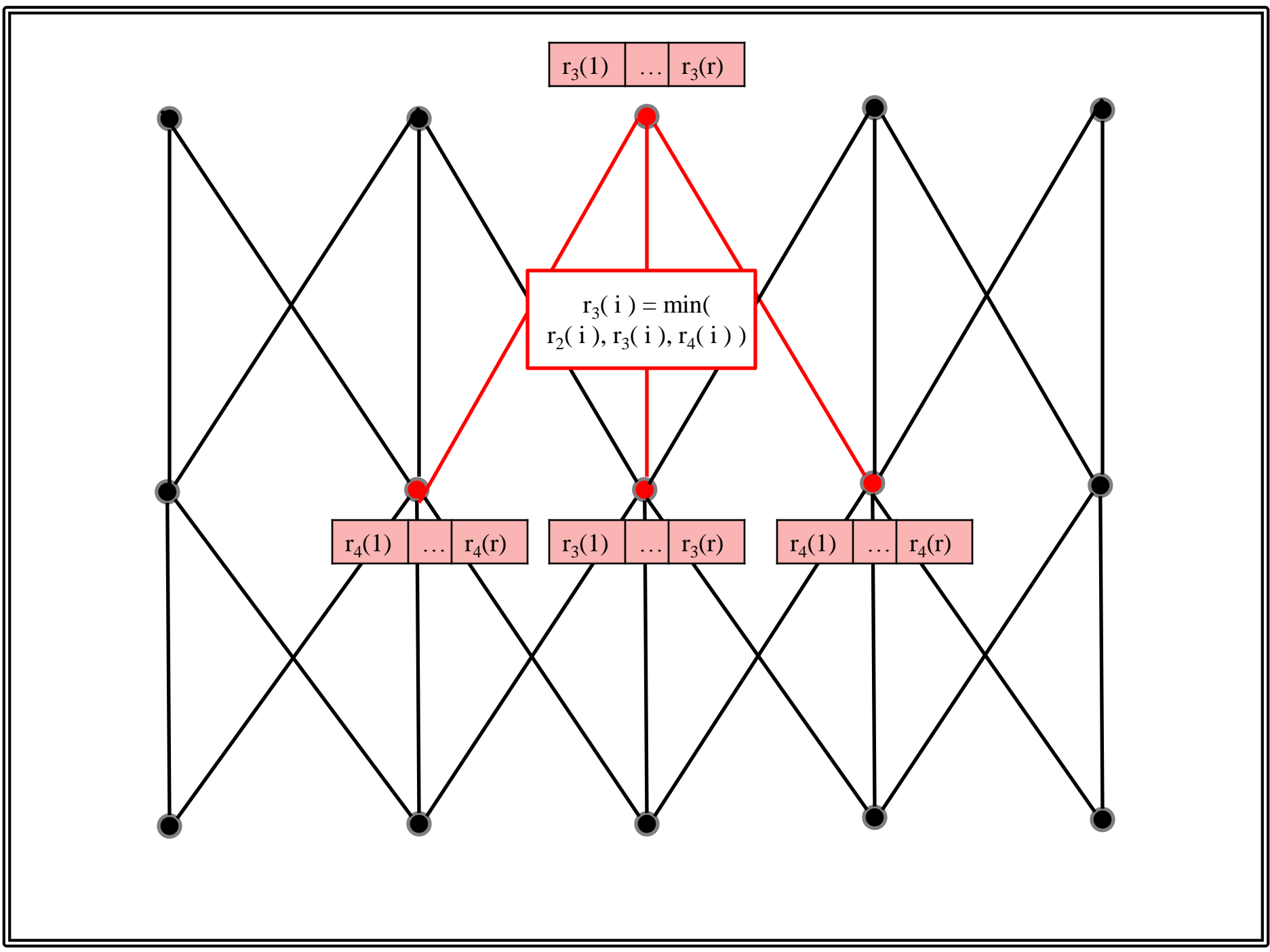
$r_3(1) \quad \dots \quad r_3(r)$

$r_4(1) \quad \dots \quad r_4(r)$

$r_5(1) \quad \dots \quad r_5(r)$







$r_3(1) \dots r_3(r)$

$$r_3(i) = \min(r_2(i), r_3(i), r_4(i))$$

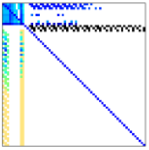
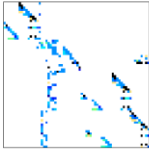
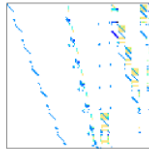
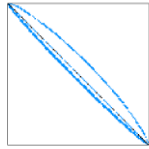
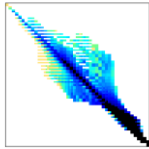
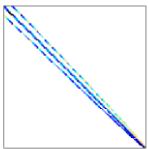
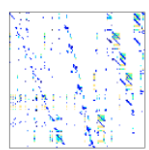
$r_4(1) \dots r_4(r)$

$r_3(1) \dots r_3(r)$

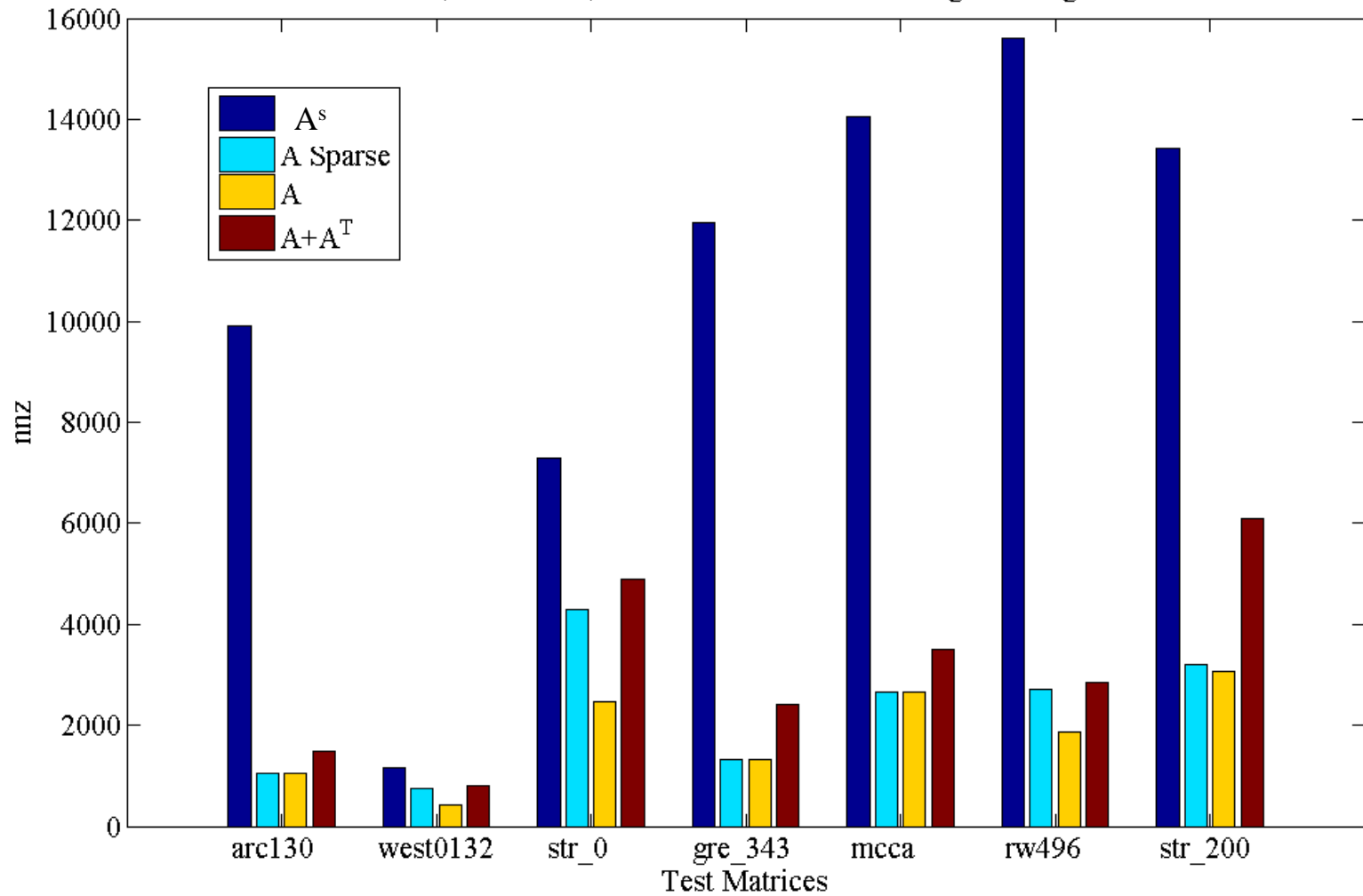
$r_4(1) \dots r_4(r)$

Preliminary Experiments

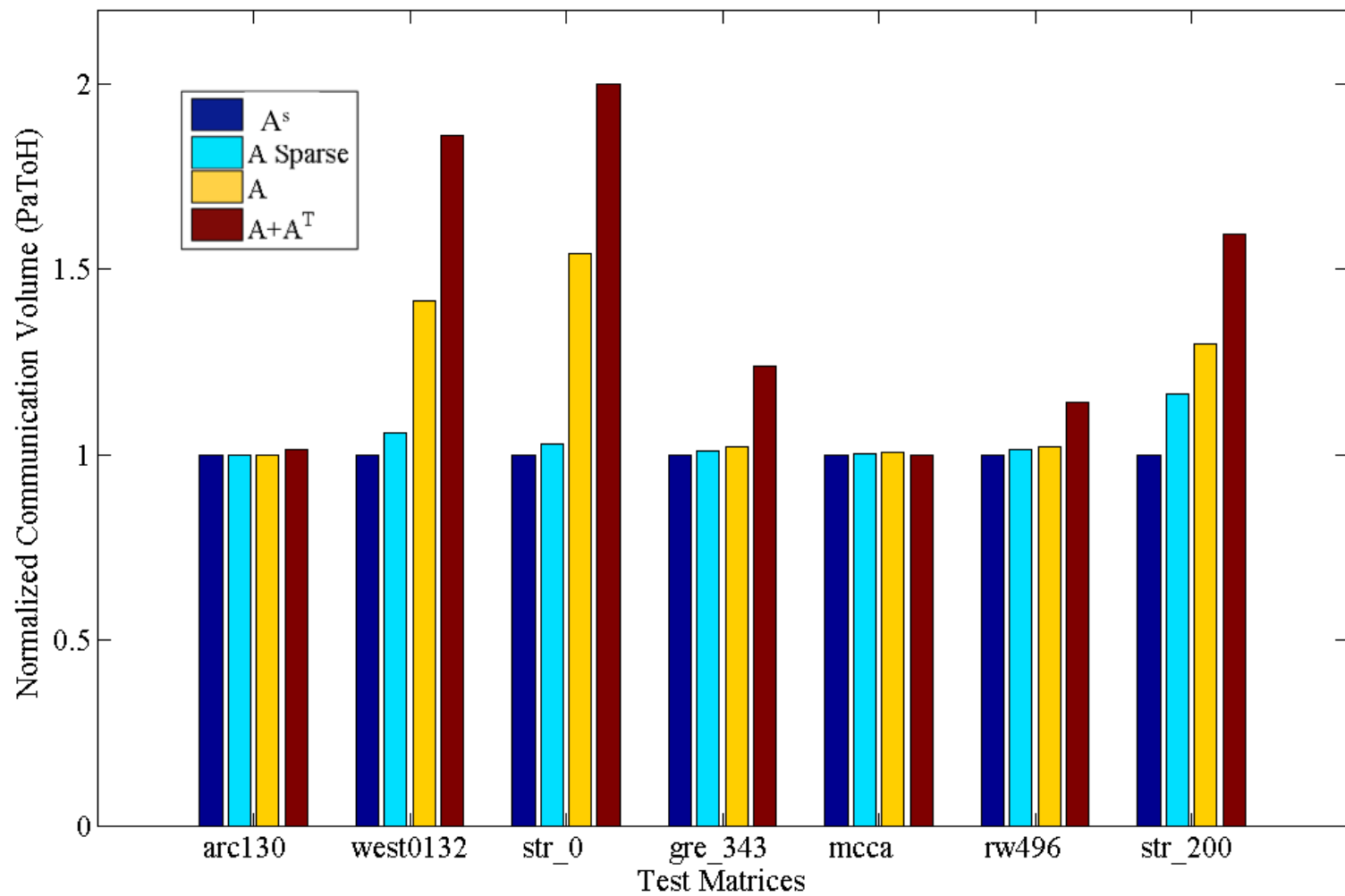
- Set of small test matrices from UFSMC [Davis '94]
- $tol = 0.5$ (half-dense), 4 parts, $s \in \{2, 3, 4\}$ depending on fill in A^s
- Comparison of hypergraph size and communication volume for four strategies:
 - s -level column nets
 - Sparsified column nets (somewhere between s - and 1-level)
 - 1-level column nets
 - Graph partitioning ($A+A^T$)
- Software: PaToH [Catalyurek, Aykanat, '99] and Metis [Karypis, Kumar '98]

Matrix	Application	n	nnz	Spy plot
arc130	Materials Science	130	1037	
west0132	Chemical Engineering	132	413	
str_0	LP	363	2454	
gre_343	Directed graph	343	1032	
mcca	Astrophysics	180	2659	
rw496	Markov Chain Model	496	1859	
str_200	LP	363	3068	

nnz (workload) for Various Partitioning Strategies



Normalized Communication Volume for Various Partitioning Strategies



Results and Observations

- Sparsified nets lead to comparable partition quality for **significantly** reduced hypergraph size
- Tuning parameter *tol* gives flexibility to trade off:
 - Quality of partition
 - Computation and storage costs