Avoiding Communication in Two-Sided Krylov Subspace Methods

Erin Carson
Nicholas Knight
James Demmel

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-93
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-93.html

August 16, 2011

Acknowledgement

This paper was researched with Government support under and awarded by the Department of Defense, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. This research was also supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung, U.S. DOE grants DE-SC0003959, DE-AC02-05-CH11231, Lawrence Berkeley National Laboratory, and NSF SDCI under Grant Number OCI-1032639.

Avoiding Communication in Two-Sided Krylov Subspace Methods

Erin Carson, Nicholas Knight, and James Demmel

Department of Computer Science
University of California at Berkeley, Berkeley, CA, USA
ecc2z@eecs.berkeley.edu, knight@eecs.berkeley.edu, demmel@eecs.berkeley.edu

Contents

1 Introduction

The cost of an algorithm is a function of both *computation*, the number of arithmetic operations performed, and *communication*, the amount of data movement. Communication cost encapsulates both data movement between levels of the memory hierarchy and between processors, and the number of messages in which the data is sent. In terms of performance, communication costs are much greater than computation costs on modern computer architectures, and the gap is only expected to widen in future systems. Therefore, in order to increase the performance of an algorithm, we must turn to strategies to minimize communication rather than try to decrease the number of arithmetic operations. We call this a "communication-avoiding" (CA) approach to algorithmic design.

Many scientific applications require codes which solve linear systems. Methods for solving linear systems can be classified either as *direct* methods, which perform a fixed number of steps to find a solution, or *iterative* methods, which repeatedly refine a candidate solution until acceptable convergence is reached. Such iterative methods are commonly used when the matrix is too large to be solved with direct methods, when the matrix is sparse, or when an error tolerance greater than machine precision is acceptable. The most general and flexible class of iterative methods are Krylov Subspace Methods (KSMs). These methods are based on projection onto expanding subspaces, where, in each iteration s, the "best" solution is chosen from the expanding Krylov subspace, $\mathcal{K}(s, A, v) = \text{span}\{v, Av, A^2v, ..., A^{s-1}v\}$. Numerous variants of Krylov subspace methods exist, each with different properties, different storage requirements, and different methods of selecting the "best" new solution.

Standard implementations of KSMs require one or more Sparse Matrix-Vector Multiplications (SpMVs) and one or more vector operations in each iteration. These are both communication-bound operations. To perform an SpMV, each processor must communicate entries of the source vector to other processors in the parallel algorithm, and A must be read from slow memory in the sequential algorithm. Vector operations, such as dot products, involve a global reduction in the parallel algorithm, and a number of reads and writes to slow memory in the sequential algorithm (depending on the size of the vectors and the size of the fast memory). The goal of Communication-Avoiding KSMs (CA-KSMs) is to break the dependency between SpMV and vector operations in each iteration, enabling us to perform s iterations for the communication cost of $1 + O(1)$ iterations.

Hoemmen, et al. have previously derived and tested algorithms for communication-avoiding implementations of both the Generalized Minimal Residual Method (CA-GMRES) and the Conjugate Gradient Method (CA-CG) [20, 27]. Communication-avoiding direct methods have also been implemented and analyzed - for a good overview, see [3]. In our work, we focus specifically on two-sided KSMs, which, implicitly or explicitly, use two different Krylov subspaces - one for the "right-hand" space, $\text{span}\{v, Av, A^2v, ...\}$, and one for the "left-hand" space, $\text{span}\{w, A^Hw, (A^H)^2w, ...\}$, where A^H denotes the conjugate transpose of A. Two-sided KSMs are specifically suited for nonsymmetric linear systems, which arise frequently in many scientific domains.

The approach to avoiding communication in KSMs involves two communication-avoiding kernels - the matrix powers kernel (`Akx`), and the Tall-Skinny QR kernel (TSQR). By using communication-avoiding kernels, we are able to reduce parallel latency, and sequential bandwidth and latency costs in a KSM by a factor of $O(s)$. If we are communication-bound, this

suggests up to a factor of s speedup, which is desirable even for small s.

Given a matrix A, a vector v, and a basis length s, the matrix powers kernel computes the basis vectors $[p_0(A)v, p_1(A)v, p_2(A)v, ..., p_{s-1}(A)v]$, where p_j is a polynomial of degree j. If A is well-partitioned (i.e., A has a low surface-to-volume ratio, see [12]), the $s-$dimensional basis can be computed only reading A (or communicating entries of v in the parallel case) $1+O(1)$ times. Therefore, to perform s iterations of the KSM, we only need to communicate $1+O(1)$ times as opposed to s times (s SpMV operations) in the standard implementation. This idea has been discussed in the literature for the stencil case, dating back to Hong and Kung's red-blue pebble game [21]. Our implementation uses the same approach, but also extends to general graphs, with the only requirement that they remain well-partitioned.

The TSQR kernel is useful for avoiding communication in KSMs which require explicit orthogonalization (e.g., GMRES). The TSQR operation is performed on the output of the matrix powers kernel, which is a tall-skinny matrix ($O(n) \times O(s)$), to orthogonalize the basis. This replaces the Modified Gram-Schmidt operation in standard GMRES, reducing communication by a factor of s (see [12, 20, 27] for details).

In addition to the matrix powers kernel and TSQR, our communication-avoiding implementations also require an additional kernel to compute the Gram-like matrix, $\tilde{V}^T V$, where V and $\tilde{V}$ are $O(n) \times O(s)$ matrices of Krylov basis vectors (the output of the matrix powers kernel). We do not discuss implementation of this kernel further, as this operation can be performed in a straightforward block-wise fashion so as to minimize communication.

Our primary contributions are as follows:

- We have derived three new two-sided Communication-Avoiding Krylov Subspace Methods (CA-KSMs): Biconjugate Gradient (CA-BICG), Conjugate Gradient Squared (CA-CGS), and Biconjugate Gradient Stabilized (CA-BICGSTAB). These are mathematically, but not numerically, equivalent to the standard implementations, in the sense that after every s steps, they produce an identical solution as the conventional algorithm in exact arithmetic. We give algorithms for two-term recurrence versions of each method. We also comment on 3-term variants for these methods, which are more susceptible to round off error, but may be attractive if computation costs are too high or storage is limited. We give an example derivation for BICG (CA-BICG3).

- Our CA-KSMs handle complex inputs, and preconditioning in the s-step basis. In order to remain communication-avoiding, the preconditioned matrix $M_L^{-1} A M_R^{-1}$ must be well-partitioned. Since preconditioning serves to reduce the condition number of A, we only expect this to improve the quality of our s-step bases.

- We provide convergence results for our methods on a set of small test matrices. The monomial basis quickly becomes ill-conditioned for larger basis sizes, which can slow down or prevent convergence. We derive the necessary recurrences and change-of-basis matrices for both the Newton basis and the scaled and shifted Chebyshev basis. Both of these recurrences are known to produce bases which can be much better conditioned than the monomial basis, thus preserving convergence for higher s values. We describe methods for obtaining good eigenvalue estimates in practice, used for construction of both these bases. Using these more stable bases, we are able to maintain stability (compared against the standard implementation) for basis lengths as high as $s =$

20. Note that if the algorithm is communication-bound, even basis length $s = 2$ theoretically results in 2 times less communication than the standard algorithm, and thus should yield a $2\times$ speedup. We also explore techniques such as orthogonalization of the basis, restarting, and deflation.

- We discuss implementation details for these methods, specifically for variants of the matrix powers kernel. Our algorithms can exploit either multiple right-hand-sides (RHSs) or computations with A^H as well as A, which both present opportunities for data reuse in certain KSMs. We also discuss future opportunities for auto-tuning, such as choosing a basis and a basis size, which must be selected to achieve both performance and stability.

In all subsequent sections of this paper, we use BICG, CGS, and BICGSTAB to refer to the standard implementations of these KSMs (as described in [33, 35, 39], respectively), and CA-BICG, CA-CGS, and CA-BICGSTAB to refer to our communication-avoiding algorithms.

2 Related Work

There is a wealth of work described in the literature related to s-step KSMs and the idea of avoiding communication. A thorough overview is given in [20], which we summarize here.

2.1 Related Work in $s-$step Methods

The first instance of an $s-$step method in the literature is Van Rosendale's conjugate gradient method [31]. Van Rosendale's implementation was motivated by exposing more parallelism using the PRAM model. Chronopoulous and Gear later created an $s-$step GMRES method with the goal of exposing more parallel optimizations [9]. Walker looked into s-step bases as a method for improving stability in GMRES by replacing the modified Gram-Schmidt orthogonalization process with Householder QR [41]. All these authors used the monomial basis, and found that convergence often could not be guaranteed for $s > 5$. It was later discovered that this behavior was due to the inherent instability of the monomial basis, which motivated research into the use of other bases for the Krylov Subspace.

Hindmarsh and Walker used a scaled (normalized) monomial basis to improve convergence [19], but only saw minimal improvement. Joubert and Carey implemented a scaled and shifted Chebyshev basis which provided more accurate results [23]. Bai et al. also saw improved convergence using a Newton basis [1]. Constructing other bases for the Krylov Subspace will be covered more thoroughly in Section 4.

Although successively scaling the basis vectors serves to lower the condition number of the basis matrix, hopefully yielding convergence closer to that of the standard method, this computation reintroduces the dependency we sought to remove, hindering communication-avoidance. Hoemmen resolves this problem using a novel matrix equilibration and balancing approach as a preprocessing step, which eliminates the need for scaled basis vectors [20].

2.2 Related Work in Avoiding Communication in KSMs

Our work is most closely related to that of [12, 20, 27]. Although their efforts focused on Lanczos, Arnoldi, CG and GMRES, the method by which we have derived our communication-avoiding algorithms is closely related. In addition to working with different Krylov Subspace Methods (BICG, CGS, and BICGSTAB), we also make contributions in improving convergence and identify further optimizations for, and variants of, the matrix powers kernel.

There have been many efforts to avoid communication in Krylov Subspace Methods in the past which differ from our approach. These can be categorized as follows:

- Reducing synchronization cost
  - Replacing Modified Gram-Schmidt with Classical Gram-Schmidt, although considered to be less stable, requires fewer synchronization points in methods which require explicit orthogonalization [18].
  - Asynchronous iterations relax synchronization constraints to reduce data dependencies, but lead to nondeterministic behavior (see, e.g., [8, 5, 6, 13, 42]).
- Increasing/Exploiting Temporal Locality
  - Block Krylov Methods can be used to solve many systems at a time (multiple RHSs for the same matrix A) (see, e.g., [10, 38, 15, 28, 26, 2]).
  - Blocking Covers can be used to reformulate the KSM to exploit temporal locality by blocking dot products. This work is a direct precursor to our methods. The same approach can be applied to avoid communication in multigrid [37, 25].
- Altering the Krylov Method
  - Chebyshev Iteration is an iterative method (but not an $s-$step method) which requires no inner products, and only one SpMV per iteration. Removing the global communication requirement has advantages in performance, but disadvantages in terms of convergence, as information can't travel as quickly (see, e.g., [4]).

3 Derivation of Algorithms

We will reorganize three BICG-like methods - BICG, CGS, and BICGSTAB - to avoid communication. In exact arithmetic, our communication-avoiding variants will get the same answer as the original versions. In finite precision, our algorithms accumulate different rounding errors, and the iterates may diverge significantly. Our communication-avoiding implementation is based on an s-step formulation of the original algorithm. This means we explicitly extend each of the underlying Krylov spaces by s dimensions and then perform s steps of Lanczos biorthogonalization, or something closely related.

Each of these steps normally requires $O(s)$ parallel synchronization points (rounds of messages), but we will use communication-avoiding kernels: the matrix powers kernel instead of s SpMVs and computing a Gram-like matrix to replace inner products. One step of the s-step

method now involves only two synchronizations. We say this approach is communication-avoiding because we reduce the latency cost asymptotically, by a factor of $\Theta(s)$. However, the ability to avoid communication in the sparse portion is constrained by the structure of A^s: for example, if A^s, $s > 1$ has diameter $s - 1$, one parallel processor or cache block will need a copy of the entire matrix and source vector. Except for block diagonal matrices and assuming, as always, no cancellation in powers of A, we expect s to remain modest, say less than 30. Thus in practice, the constants in our big-O notation may be relevant. The additional costs of the communication-avoiding biorthogonalization kernel grow slowly with s, exceeding the standard algorithm by factors of $O(s)$ more flops and words moved.

The main concern with s-step methods is how to compute the s-step basis stably in practice. The sequence of vectors $(x, Ax, A^2x, \ldots)$, which is the monomial basis of a Krylov space, typically becomes linearly dependent in finite precision faster than in exact arithmetic (when it does so always before $A^{n+1}x$). This happens when the sequence converges to an eigenvector of A, as in the power method. We cannot hope to find s basis vectors of a space of dimension less than s, yet this is exactly what we may ask our algorithm to find.

If A is normal, we can use spectral information to find Newton and Chebyshev polynomial bases that are linearly independent for greater s values. In practice, we expect matrices that are roughly normal, i.e., relatively few defective eigenvalues, to also benefit from these polynomials. Unfortunately, when A is highly nonnormal, the analogy with polynomial interpolation breaks down and we can no longer bound the condition number of our s-step bases by the condition number of a polynomial defined on the spectrum of A (in fact, convergence theory for the standard algorithms breaks down as well). Attempts to instead use the field of values of A or the convex hull of the spectrum have been unsuccessful in general. Although we only discuss the Newton and Chebyshev polynomials, our presentation assumes only that a polynomial satisfy a three-term recurrence, which is general enough for all the classical orthogonal polynomials.

All algorithms we present assume complex-valued inputs, and we will not discuss the straightforward simplifications that arise for real data.

3.1 Two-term vs. three-term recurrences

All of the BICG-like methods we consider originate with the nonsymmetric Lanczos process, and so the vector iterates always satisfy two- or three-term recurrences. In the case of the BICG algorithm, we consider both the BIOMIN (two pairs of coupled two-term recurrences) and the BIORES (one pair of uncoupled three-term recurrences) forms.

In finite precision, the BICG-like methods we consider demonstrate a deviation between the computed residual and true residual. This is because the candidate solution and residual are updated independently, with different rounding errors. Usually this destroys convergence: as the computed residual norm decreases, the corrections to the true residual become smaller, causing the true residual to stagnate or worsen. This limits the maximum attainable accuracy of the method, which we want to be independent of the data. It is demonstrated in [17] that this instability can be worse for three-term recurrences than for two- term recurrences. We did not generalize this error analysis to our communication-avoiding formulations, but conjecture that the same conclusion holds.

We note two algorithmic approaches to cope with this problem of round off accumulation.

The first approach is to substitute the true residual $r_m = b - Ax_m$ on iteration m, in place of the computed residual. The authors of [34] have formalized the problem of dynamically selecting m and minimizing the impact of such a substitution on the underlying Lanczos process. For our convergence studies, we monitored the true residual for stagnation, chose the corresponding iterations m, and reran the method with these statically selected restart indices m. A dynamic restarting approach for CA-KSMs is future work.

The second algorithmic approach is based on the observation that BICG-like methods unnecessarily restrict the Lanczos vectors to be the residuals; this is called enforcing consistency. An inconsistent formulation [16] computes scalar multiples of the residuals, and the flexibility to arbitrarily choose these scaling factors gives more control over the sizes of the iterates and scalar coefficients, and thus the bounds on round off error. Relaxing consistency also enables us to avoid pivot breakdowns in three-term recurrence variants of BICG and CGS. The inconsistent formulations of classical BICG and CGS incur no more communication or computation asymptotically and involve only slight modifications to the consistent versions. The same is true for communication-avoiding inconsistent formulations and we give an example for inconsistent BIORES, below. This preventative measure does not eliminate the possibility of stagnation - it only attempts to postpone it - thus restarting might still be necessary. For this reason, we only considered restarting in our convergence studies and consider inconsistent formulations a future direction.

Although they have different stability properties, two-term and three-term recurrence variants have similar computation communication, and storage costs. This might be surprising: in the case of BICG, the two-term variant BIOMIN produces four Krylov bases while the three-term variant BIORES produces only two, yet both versions perform the same number of SpMV operations per iteration. The BIOMIN variant couples the search directions and the residuals in order to advance four Krylov spaces with two SpMV operations. Unfortunately, in order to avoid communication, we must break the data dependency introduced by this coupling. Thus, our communication-avoiding BIOMIN implementation explicitly constructs all four Krylov bases, at the cost of doubling sparse flops and words moved in parallel. Our communication-avoiding BIORES implementation avoids doubling the sparse flops, but instead requires maintaining a number of n-vectors proportional to the number of steps we wish to take without communication (s).

We chose to implement and experiment with two-term recurrence, consistent formulations of BICG, CGS, and BICGSTAB, in order to compare convergence with MATLAB BICG, CGS, and BICGSTAB. We discuss three-term and inconsistent formulations to suggest that the communication-avoiding algorithm design space offers flexibility to trade off stability and performance.

3.2 Communication-avoiding biconjugate gradient method (CA-BICG)

We present derivations for four variants of communication-avoiding BICG (CA-BICG):

- CA-BIOMIN (consistent version), Alg. 3
- CA-BIOMIN (inconsistent version), Alg. 4
- CA-BIORES (consistent version), Alg. 7

- CA-BIORES (inconsistent version), Alg. 8

The names BIOMIN and BIORES refer to the two-term- and three-term-recurrence variants of the classical BICG method, which we present in Algs. 1, 2, 5, and 6. We referenced [16] for these algorithms, as well as the naming conventions.

Our numerical experiments were performed with consistent CA-BIOMIN, Alg. 3, which is closest to what is implemented in MATLAB BICG. The other variants have different stability properties and performance characteristics, and we present them as directions for future work.

3.2.1 Consistent and inconsistent BIOMIN

We present two versions of two-term classical BICG. [16] refers to this as algorithm as BIOMIN, and we will do the same. The standard version, as in MATLAB BICG is due to [14], and is consistent - see Alg. 1 . We also present an inconsistent variant in Alg. 2.

Algorithm 1 Consistent BIOMIN

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)$.
Choose $\tilde{p}_0 = \tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{p}_0^H A p_0 \neq 0$.
repeat for $m = 0, 1, \ldots,$
    Set $\delta_m' = \tilde{p}_m^H A p_m$. If $\delta_m' = 0$, declare *pivot breakdown* and STOP.
    Take steps along search directions:
        $\omega_m = \delta_m / \delta_m'$.
        $r_{m+1} = r_m - \omega_m A p_m$.
        $\tilde{r}_{m+1} = \tilde{r}_m - \overline{\omega_m} A^H \tilde{p}_m$.
    Update candidate solution: $x_{m+1} = x_m + \omega_m p_m$.
    Set $\delta_{m+1} = \tilde{r}_{m+1}^H r_{m+1}$. If $\delta_{m+1} = 0$, STOP.
        If $r_{m+1} = 0$, terminate with $x_{\text{ex}} = x_{m+1}$.
        Otherwise, declare *Lanczos breakdown.*
    Update search directions:
        $\psi_m = -\delta_{m+1} / \delta_m$.
        $p_{m+1} = r_{m+1} - \overline{\psi_m} p_m$.
        $\tilde{p}_{m+1} = \tilde{r}_{m+1} - \overline{\psi_m} \tilde{p}_m$.
until $||r_{m+1}||$ is small.
Terminate with approximate solution $x \approx x_{m+1}$

3.2.2 CA-BIOMIN

We first consider the consistent BIOMIN algorithm in Alg. 1. Starting at iteration m, we must identify the data dependencies through the end of iteration $m + s - 1$. These data dependencies are the four s-step bases

$$[P, R] = [p_m, Ap_m, \ldots, A^s p_m, r_m, Ar_m, \ldots, A^s r_m]$$
$$\left[\tilde{P}, \tilde{R}\right] = \left[\tilde{p}_m, A^H \tilde{p}_m, \ldots, \left(A^H\right)^s \tilde{p}_m, \tilde{r}_m, A^H \tilde{r}_m, \ldots, \left(A^H\right)^s \tilde{r}_m\right]$$

Algorithm 2 Inconsistent BIOMIN

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)/\gamma_{-1}$ for some $\gamma_{-1} \neq 0$, e.g. so $||r_0|| = 1$.
Redefine $x_0 = x_0/\gamma_{-1}$.
Choose $\tilde{p}_0 = \tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{p}_0^H A p_0 \neq 0$.
repeat for $m = 0, 1, \ldots,$
    Set $\delta_m' = \tilde{p}_m^H A p_m$. If $\delta_m' = 0$, declare *pivot breakdown* and STOP.
    Take steps along search directions:
        $\phi_m = \delta_m'/\delta_m$.
        Choose $\gamma_m \neq 0$ and $\tilde{\gamma}_m \neq 0$ arbitrarily.
        $r_{m+1} = (r_m - \phi_m A p_m)/\gamma_m$.
        $\tilde{r}_{m+1} = \left(\tilde{r}_m - \overline{\phi_m} A^H \tilde{p}_m\right)/\tilde{\gamma}_m$.
        $\pi_{m+1} = -\pi_m \phi_m/\gamma_m$, with $\pi_0 = 1/\gamma_{-1}$.
    Update candidate solution: $x_{m+1} = -(\phi_m x_m + p_m)/\gamma_m$.
    Set $\delta_{m+1} = \tilde{r}_{m+1}^H r_{m+1}$. If $\delta_{m+1} = 0$, STOP.
        If $r_{m+1} = 0$, terminate with $x_{\text{ex}} = x_{m+1}/\pi_{m+1}$.
        Otherwise, declare *Lanczos breakdown*.
    Update search directions:
        $\psi_m = \overline{\tilde{\gamma}_m} \delta_{m+1}/\delta_m'$.
        $\tilde{\psi}_m = \gamma_m \delta_{m+1}/\delta_m'$.
        $p_{m+1} = r_{m+1} - \psi_m p_m$.
        $\tilde{p}_{m+1} = \tilde{r}_{m+1} - \tilde{\psi}_m \tilde{p}_m$.
until $||r_{m+1}||\,/\,|\pi_{m+1}|$ is small.
Terminate with approximate solution $x \approx x_{m+1}/\pi_{m+1}$.

and the scalar coefficient δ_m. The columns of these matrices span Krylov spaces. We compute s-step bases of these Krylov spaces, $\left[\hat{P}, \hat{R}, \hat{\tilde{P}}, \hat{\tilde{R}}\right]$, calling the matrix powers kernel. As previously mentioned, we might choose to use a different polynomial basis besides the monomial basis for stability reasons - we have decorated the `Akx` bases with carats. For ease of presentation, assume we use the same polynomials $\rho_j(z), 0 \leq j \leq s$ for all four bases, and that ρ_j has nonzero leading coefficient so that $\deg\rho_j = j$. For example, $\hat{P}(:, j) = \rho_{j+1}(A)\, p_m$ and $\hat{\tilde{R}}(:, j) = \rho_{j+1}\left(A^H\right) \tilde{r}_m$. Then we can find an upper triangular $s+1 \times s+1$ change of basis matrix B, to recover the monomial (or Krylov) bases, according to

$$\left[\hat{P}, \hat{R}, \hat{\tilde{P}}, \hat{\tilde{R}}\right] \cdot I_{4,4} \otimes B = \left[P, R, \tilde{P}, \tilde{R}\right]$$

We discuss the choice of polynomials and computing such a matrix B later.

Since all iterates in iterations $m : m+s$ can be expressed as a linear combination of the vectors in the s-step bases, we can run nonsymmetric Lanczos symbolically with the coefficient vectors, for $j = 0 : s$ and $k = 0 : s - j$,

$$\left[\hat{P}, \hat{R}\right] c_j^k = A^k r_{m+j} \qquad \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] d_j^k = \left(A^H\right)^k \tilde{r}_{m+j}$$
$$\left[\hat{P}, \hat{R}\right] a_j^k = A^k p_{m+j} \qquad \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] b_j^k = \left(A^H\right)^k \tilde{p}_{m+j}$$

to represent the iterates locally/in fast memory. This also lets us compute the inner products, although not symbolically - the inner products are encoded in the entries of the Gram-like matrix $G = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right]^H \left[\hat{P}, \hat{R}\right]$ and recoverable using the coefficient vectors.

At this point we simply state the algorithm, in Alg. 3. It is a purely mechanical process to generalize this result to the inconsistent BIOMIN formulation (Alg. 2), which we present in Alg. 4.

3.2.3 Consistent and inconsistent BIORES

We present two versions of three-term BICG in Algs. 5 and 6. Gutknecht, in [16] refers to this as algorithm as BIORES, and we will do the same. The difference between the consistent and inconsistent versions is the same as with BIOMIN, except additionally, the consistent version is able to avoid the *pivot breakdown* condition, which means a division by zero when γ_m vanishes or perhaps overflow when γ_m nearly vanishes. Pivot breakdown is due to a zero pivot, when (implicitly) performing Gaussian elimination on the (implicit) tridiagonal matrix, and is easily avoided. As before, the additional cost is negligible - we simply maintain an extra scalar π_m.

[16] also presents BIODIR, another 3-term variant, which constructs a set of biconjugate rather than biorthogonal bases. Biconjugate means biorthogonal with respect to the A inner product - because A is not necessarily SPD, this is a formal inner product. This variant has the advantage of avoiding Lanczos breakdown (by stalling, perhaps indefinitely, in the case of an incurable breakdown), but is still susceptible to pivot breakdown unless an inconsistent formulation is applied. BIODIR is part of a large class of look-ahead Lanczos methods, which are designed to overcome Lanczos breakdowns and near breakdowns. A

Algorithm 3 Communication-avoiding consistent BIOMIN

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)$.

Choose $\tilde{p}_0 = \tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{p}_0^H A p_0 \neq 0$.

repeat for $m = 0, 1, \ldots,$

  Extend the four Krylov bases by s dimensions, using the matrix powers kernel (`Akx`).

  $\left[\left[\hat{P}, \hat{R}\right], B\right] = \texttt{Akx}\left(A,\ s,\ [p_m, r_m]\right),\ \left[\left[\hat{\tilde{P}}, \hat{\tilde{R}}\right], \tilde{B}\right] = \texttt{Akx}\left(A^H,\ s,\ [\tilde{p}_m, \tilde{r}_m]\right).$

  Compute $G = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right]^H \left[\hat{P}, \hat{R}\right]$.

  Initialize coefficient vectors

  $[c_0^0, c_0^1, \ldots, c_0^s] = \begin{bmatrix} 0_{s+1,s+1} \\ B \end{bmatrix},\ [a_0^0, a_0^1, \ldots, a_0^s] = \begin{bmatrix} B \\ 0_{s+1,s+1} \end{bmatrix},$

  $[d_0^0, d_0^1, \ldots, d_0^s] = \begin{bmatrix} 0_{s+1,s+1} \\ \tilde{B} \end{bmatrix},\ [b_0^0, b_0^1, \ldots, b_0^s] = \begin{bmatrix} \tilde{B} \\ 0_{s+1,s+1} \end{bmatrix},\ e_0 = \begin{bmatrix} 0_{2s+2,1} \\ 1 \end{bmatrix}.$

  for $j = 0 : s - 1$, **do**

    $\delta_{m+j}' = \left(b_j^0\right)^H G a_j^1$. If $\delta_{m+j}' = 0$, declare *pivot breakdown* and STOP.

    Take steps along search directions:

      $\omega_{m+j} = \delta_{m+j} / \delta_{m+j}'$.

      $c_{j+1}^k = c_j^k - \omega_{m+j} a_j^{k+1}$, $d_{j+1}^k = d_j^k - \overline{\omega_{m+j}} b_j^{k+1}$, for $k = 0 : s - j - 1$.

    Update candidate solution: $e_{j+1} = e_j + \omega_{m+j} \begin{bmatrix} a_j^0 \\ 0 \end{bmatrix}$.

    Set $\delta_{m+j+1} = \left(d_{j+1}^0\right)^H G c_{j+1}^0$. If $\delta_{m+j+1} = 0$, STOP.

      If $\left[\hat{P}, \hat{R}\right] c_{j+1}^0 = 0$, terminate with $x_{\text{ex}} = \left[\hat{P}, \hat{R}, x_m\right] e_{j+1}$.

      Otherwise, declare *Lanczos breakdown*.

    Update search directions:

      $\psi_{m+j} = -\delta_{m+j+1} / \delta_{m+j}$.

      $a_{j+1}^k = c_{j+1}^k - \psi_{m+j} a_j^k$, $b_{j+1}^k = d_{j+1}^k - \overline{\psi_{m+j}} b_j^k$, for $k = 0 : s - j - 1$.

  end for

  Recover iterates from last inner iteration.

  $r_{m+s} = \left[\hat{P}, \hat{R}\right] c_s^0,\ p_{m+s} = \left[\hat{P}, \hat{R}\right] a_s^0,$

  $\tilde{r}_{m+s} = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] d_s^0,\ \tilde{p}_{m+s} = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] b_s^0,\ x_{m+s} = \left[\hat{P}, \hat{R}, x_m\right] e_s.$

until $||r_{m+s}||$ is small.

Terminate with approximate solution $x \approx x_{m+s}$.

Algorithm 4 Communication-avoiding inconsistent BIOMIN

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)/\gamma_{-1}$ for some $\gamma_{-1} \neq 0$, e.g., so $||r_0|| = 1$.

Redefine $x_0 = x_0/\gamma_{-1}$.

Choose $\tilde{p}_0 = \tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{p}_0^H A p_0 \neq 0$.

repeat for $m = 0, 1, \ldots,$

Extend the four Krylov bases by s dimensions, using the matrix powers kernel (`Akx`).

$\left[\left[\hat{P}, \hat{R}\right], B\right] = \texttt{Akx}\left(A,\, s,\, [p_m, r_m]\right),\ \left[\left[\hat{\tilde{P}}, \hat{\tilde{R}}\right], \tilde{B}\right] = \texttt{Akx}\left(A^H,\, s,\, [\tilde{p}_m, \tilde{r}_m]\right).$

Compute $G = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right]^H \left[\hat{P}, \hat{R}\right]$.

Initialize coefficient vectors

$[c_0^0, c_0^1, \ldots, c_0^s] = \begin{bmatrix} 0_{s+1,s+1} \\ B \end{bmatrix},\ [a_0^0, a_0^1, \ldots, a_0^s] = \begin{bmatrix} B \\ 0_{s+1,s+1} \end{bmatrix},$

$[d_0^0, d_0^1, \ldots, d_0^s] = \begin{bmatrix} 0_{s+1,s+1} \\ \tilde{B} \end{bmatrix},\ [b_0^0, b_0^1, \ldots, b_0^s] = \begin{bmatrix} \tilde{B} \\ 0_{s+1,s+1} \end{bmatrix},\ e_0 = \begin{bmatrix} 0_{2s+2,1} \\ 1 \end{bmatrix}.$

for $j = 0 : s - 1$, **do**

$\delta_{m+j}' = \left(b_j^0\right)^H G a_j^1$. If $\delta_{m+j}' = 0$, declare *pivot breakdown* and STOP.

Take steps along search directions:

$\phi_{m+j} = \delta_{m+j}'/\delta_{m+j}$.

Choose $\gamma_{m+j} \neq 0$ and $\tilde{\gamma}_{m+j} \neq 0$ arbitrarily.

$c_{j+1}^k = \left(c_j^k - \phi_{m+j} a_j^{k+1}\right)/\gamma_{m+j},\ d_{j+1}^k = \left(d_j^k - \overline{\phi_{m+j}} b_j^{k+1}\right)/\tilde{\gamma}_{m+j},$ for $k = 0 : s - j - 1$.

$\pi_{m+j+1} = -\pi_{m+j}\phi_{m+j}/\gamma_{m+j}$, with $\pi_0 = 1/\gamma_{-1}$.

Update candidate solution: $e_{j+1} = -\left(\phi_{m+j} e_j + \begin{bmatrix} a_j^0 \\ 0 \end{bmatrix}\right)/\gamma_{m+j}$.

Set $\delta_{m+j+1} = \left(d_{j+1}^0\right)^H G c_{j+1}^0$. If $\delta_{m+j+1} = 0$, STOP.

If $\left[\hat{P}, \hat{R}\right] c_{j+1}^0 = 0$, terminate with $x_{\text{ex}} = \left(\left[\hat{P}, \hat{R}, x_m\right] e_{j+1}\right)/\pi_{m+j+1}$.

Otherwise, declare *Lanczos breakdown*.

Update search directions:

$\psi_{m+j} = \overline{\tilde{\gamma}_{m+j}}\delta_{m+j+1}/\delta_{m+j}',\ \tilde{\psi}_{m+j} = \overline{\gamma_{m+j}\delta_{m+j+1}/\delta_{m+j}'}$.

$a_{j+1}^k = c_{j+1}^k - \psi_{m+j} a_j^k,\ b_{j+1}^k = d_{j+1}^k - \tilde{\psi}_{m+j} b_j^k,$ for $k = 0 : s - j - 1$.

end for

Recover iterates from last inner iteration.

$r_{m+s} = \left[\hat{P}, \hat{R}\right] c_s^0,\ p_{m+s} = \left[\hat{P}, \hat{R}\right] a_s^0,$

$\tilde{r}_{m+s} = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] d_s^0,\ \tilde{p}_{m+s} = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right] b_s^0,\ x_{m+s} = \left[\hat{P}, \hat{R}, x_m\right] e_s.$

until $||r_{m+s}||\,/\,|\pi_{m+s}|$ is small.

Terminate with approximate solution $x \approx x_{m+s}/\pi_{m+s}$.

communication-avoiding BIODIR formulation can be derived in a similar way to BIORES and has similar costs except requires the storage of an extra iterate. We conjecture that our communication-avoiding approach will generalize to other look-ahead Lanczos methods; this is future work.

Algorithm 5 Consistent BIORES

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $r_0 = (b - Ax_0)$.
Choose $\tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$.
repeat for $m = 0, 1, \ldots,$
  Set $\alpha_m = \tilde{r}_m^H A r_m / \delta_m$ and $\tilde{\alpha}_m = \overline{\alpha_m}$.
  Set $\beta_{m-1} = \tilde{\gamma}_{m-1}\delta_m/\delta_{m-1}$ and $\tilde{\beta}_{m-1} = \overline{\gamma_{m-1}\delta_m/\delta_{m-1}} = \overline{\beta_{m-1}\gamma_{m-1}}/\tilde{\gamma}_{m-1}$, with $\beta_{-1} = \tilde{\beta}_{-1} = 0$.
  Set $\gamma_m = \alpha_m - \beta_{m-1}$. If $\gamma_m = 0$, declare *pivot breakdown* and STOP.
  Choose $\tilde{\gamma}_m \neq 0$ arbitrarily.
  $r_{m+1} = \left(Ar_m - \alpha_m r_m - \beta_{m-1} r_{m-1}\right)/\gamma_m$
  $\tilde{r}_{m+1} = \left(A^H \tilde{r}_m - \tilde{\alpha}_m \tilde{r}_m - \tilde{\beta}_{m-1}\tilde{r}_{m-1}\right)/\tilde{\gamma}_m$
  $x_{m+1} = -\left(r_m + \alpha_m x_m + \beta_{m-1} x_{m-1}\right)/\gamma_m$.
  $\delta_{m+1} = \tilde{r}_{m+1}^H r_{m+1}$. If $\delta_{m+1} = 0$, STOP.
    If $r_{m+1} = 0$, terminate with $x_{\text{ex}} = x_{m+1}$. Otherwise,
    If $\tilde{r}_{m+1} \neq 0$, declare *Lanczos breakdown*;
    If $\tilde{r}_{m+1} = 0$, declare *left termination*.
until $||r_{m+1}||$ is small.
Terminate with approximate solution $x \approx x_{m+1}$.

3.2.4 CA-BIORES

The derivation here seems more complicated than CA-BIOMIN but the difference is superficial, purely due to the fact that we represent the iterates partially in terms of the s-step history

$$R_{\ell-1} = [r_{m-s+1}, r_{m-s+2}, \ldots, r_m] \quad \text{and} \quad \tilde{R}_{\ell-1} = [\tilde{r}_{m-s+1}, \tilde{r}_{m-s+2}, \ldots, \tilde{r}_m]$$

where ℓ indexes the outer iterations, and $m = \ell s + j$, where j indexes the inner iterations for the current outer iterations. We combine the s-step history with the s-step bases generated from r_m and $\tilde{r}_m$, as

$$\hat{R} := [R_{\ell-1}(:, 1:s), \rho_0(A)\, r_m, \rho_1(A)\, r_m, \ldots, \rho_s(A)\, r_m]$$
$$\hat{\tilde{R}} := \left[\tilde{R}_{\ell-1}(:, 1:s), \tilde{\rho}_0\left(A^H\right)\tilde{r}_m, \tilde{\rho}_1\left(A^H\right)\tilde{r}_m, \ldots, \tilde{\rho}_s\left(A^H\right)\tilde{r}_m\right]$$

Algorithm 6 Inconsistent BIORES

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $r_0 = (b - Ax_0)/\gamma_{-1}$ for some $\gamma_{-1} \neq 0$, e.g., so $||r_0|| = 1$.
Redefine $x_0 = x_0/\gamma_{-1}$.
Choose $\tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$.
repeat for $m = 0, 1, \ldots,$
    Set $\alpha_m = \tilde{r}_m^H A r_m / \delta_m$ and $\tilde{\alpha}_m = \overline{\alpha_m}$.
    Set $\beta_{m-1} = \overline{\tilde{\gamma}_{m-1}} \delta_m / \delta_{m-1}$ and $\tilde{\beta}_{m-1} = \overline{\gamma_{m-1} \delta_m / \delta_{m-1}} = \overline{\beta_{m-1} \gamma_{m-1}} / \tilde{\gamma}_{m-1}$, with $\beta_{-1} = \tilde{\beta}_{-1} = 0$.
    Choose $\gamma_m \neq 0$ and $\tilde{\gamma}_m \neq 0$ arbitrarily.
    $\pi_{m+1} = -(\alpha_m \pi_m + \beta_{m-1} \pi_{m-1})/\gamma_m$ with $\pi_0 = 1/\gamma_{-1}$.
    $r_{m+1} = (A r_m - \alpha_m r_m - \beta_{m-1} r_{m-1})/\gamma_m$
    $\tilde{r}_{m+1} = \left(A^H \tilde{r}_m - \tilde{\alpha}_m \tilde{r}_m - \tilde{\beta}_{m-1} \tilde{r}_{m-1}\right)/\tilde{\gamma}_m$.
    $x_{m+1} = -(r_m + \alpha_m x_m + \beta_{m-1} x_{m-1})/\gamma_m$.
    $\delta_{m+1} = \tilde{r}_{m+1}^H r_{m+1}$. If $\delta_{m+1} = 0$, STOP.
        If $r_{m+1} = 0$, terminate with $x_{\text{ex}} = x_{m+1}/\pi_{m+1}$. Otherwise,
        If $\tilde{r}_{m+1} \neq 0$, declare *Lanczos breakdown*;
        If $\tilde{r}_{m+1} = 0$, declare *left termination*.
until $||r_{m+1}|| / |\pi_{m+1}|$ is small.
Terminate with approximate solution $x \approx x_{m+1}/\pi_{m+1}$.

where $\rho_j(z)$ and $\tilde{\rho}_j(z)$ are polynomials of (exact) degree j. The matrix powers kernel also provides change-of-basis matrices B and $\tilde{B}$ to perform the transformations

$$R := \hat{R} \begin{bmatrix} I_{s,s} & 0_{s,s+1} \\ 0_{s+1,s} & B \end{bmatrix} = [r_{m-s+1}, \ldots, r_{m-1}, r_m, A r_m, \ldots, A^s r_m]$$

$$\tilde{R} := \hat{\tilde{R}} \begin{bmatrix} I_{s,s} & 0_{s,s+1} \\ 0_{s+1,s} & \tilde{B} \end{bmatrix} = \left[\tilde{r}_{m-s+1}, \ldots, \tilde{r}_{m-1}, \tilde{r}_m, A^H \tilde{r}_m, \ldots, \left(A^H\right)^s \tilde{r}_m\right]$$

We will apply these transformations implicitly. Note that the initial conditions $\beta_{-1} = \tilde{\beta}_{-1} = 0$ truncate the three-term recurrence before $m = 0$ and allow us to define $r_i = \tilde{r}_i = 0_{n,1}$ when $i < 0$, and r_0 and $\tilde{r}_0$ are readily available from the input data. We assume $m \geq 0$.

Our goal is to compute

$$R_\ell = [r_{m+1}, r_{m+2}, \ldots, r_{m+s}] \quad \text{and} \quad \tilde{R}_\ell = [\tilde{r}_{m+1}, \tilde{r}_{m+2}, \ldots, \tilde{r}_{m+s}]$$

from the s-step history and s-step bases. Examine the residual update formulas:

$$r_{m+1} = (A r_m - \alpha_m r_m - \beta_{m-1} r_{m-1})/\gamma_m \quad \text{and} \quad \tilde{r}_{m+1} = \left(A^H \tilde{r}_m - \tilde{\alpha}_m \tilde{r}_m - \tilde{\beta}_{m-1} \tilde{r}_{m-1}\right)/\tilde{\gamma}_m$$

manipulated to become

$$A^k r_{m+j+1} = \begin{cases} \left(A^{k+1} r_{m+j} - A^k \left[r_{m+j-1}, r_{m+j} \right] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix} \right) / \gamma_{m+j} \\ A^{k-1} \left[r_{m+j}, r_{m+j+1}, r_{m+j+2} \right] \begin{bmatrix} -\beta_{m+j} \\ -\alpha_{m+j+1} \\ \gamma_{m+j+1} \end{bmatrix} \end{cases}$$

$$\left(A^H\right)^k \tilde{r}_{m+j+1} = \begin{cases} \left(\left(A^H\right)^{k+1} \tilde{r}_{m+j} - \left(A^H\right)^k \left[\tilde{r}_{m+j-1}, \tilde{r}_{m+j} \right] \begin{bmatrix} \tilde{\beta}_{m+j-1} \\ \tilde{\alpha}_{m+j} \end{bmatrix} \right) / \tilde{\gamma}_{m+j} \\ \left(A^H\right)^{k-1} \left[\tilde{r}_{m+j}, \tilde{r}_{m+j+1}, \tilde{r}_{m+j+2} \right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix} \end{cases}$$

and introduce the coefficient vectors

$$\hat{R} c_j^k = A^k r_{m+j} \quad \text{and} \quad \hat{\tilde{R}} d_j^k = \left(A^H\right)^k \tilde{r}_{m+j}$$

As with BIOMIN, we substitute these coefficient vectors into the recurrences for the left and right residuals. This step is more involved here so we provide an intermediate result before stating the algorithm. Provided the Lanczos coefficients are available (we will show how to compute them shortly), the constraints $-s \leq j \leq s$ and $0 \leq k \leq s$ allow us to express the left and right residuals as

$$c_{j+1}^k = \begin{cases} \left(c_j^{k+1} - \left[c_{j-1}^k, c_j^k \right] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix} \right) / \gamma_{m+j} & j > -1 \\ \begin{bmatrix} 0_{s,1} \\ B(:, k+1) \end{bmatrix} & j = -1 \\ \left[c_j^{k-1}, c_{j+1}^{k-1}, c_{j+2}^{k-1} \right] \begin{bmatrix} -\beta_{m+j} \\ -\alpha_{m+j+1} \\ \gamma_{m+j+1} \end{bmatrix} & j < -1, k > 0 \\ \begin{bmatrix} I_{s,s}(:, s+j+1) \\ 0_{s+1,1} \end{bmatrix} & j < -1, k = 0 \end{cases}$$

$$d_{j+1}^k = \begin{cases} \left(d_j^{k+1} - \left[d_{j-1}^k, d_j^k \right] \begin{bmatrix} \tilde{\beta}_{m+j-1} \\ \tilde{\alpha}_{m+j} \end{bmatrix} \right) / \tilde{\gamma}_{m+j} & j > -1 \\ \begin{bmatrix} 0_{s,1} \\ \tilde{B}(:, k+1) \end{bmatrix} & j = -1 \\ \left[d_j^{k-1}, d_{j+1}^{k-1}, d_{j+2}^{k-1} \right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix} & j < -1, k > 0 \\ \begin{bmatrix} I_{s,s}(:, s+j+1) \\ 0_{s+1,1} \end{bmatrix} & j < -1, k = 0 \end{cases}$$

We avoid the inner products in the same way as CA-BIOMIN, by the use of the Gram-like matrix $G = \hat{\tilde{R}}^H \hat{R}$. The rest of the computations in BIORES require no communication. We present both consistent and inconsistent versions of communication-avoiding BIORES together in Algs. 7 and 8.

3.3 Communication-avoiding conjugate gradient squared method (CA-CGS)

The multiplication by A^H and corresponding left Krylov vectors (residuals $\tilde{r}$) only contribute to the BICG solution of $Ax = b$ through the scalar coefficients. BICG does not exploit the reduction in magnitude of the left residuals unless we solve a dual system. In an effort to get faster convergence, or when A^H might not be available, for instance when the linear system A is the Jacobian in a (matrix-free) Newton-Krylov method, one might demand a transpose-free method such as CGS (conjugate gradient squared, [35]), a QOR (quasi-orthogonal residual) method derived from BICG (another QOR method), that avoids the multiplication by A^H. CGS respects the mutual biorthogonality of the two Krylov spaces (and so the Lanczos coefficients are the same as BICG, in exact arithmetic), however, the polynomials representing the CGS residuals are the squares of those in BICG. These squared residuals are required to avoid keeping track of the left Lanczos vectors. However, CGS actually interprets the squared BICG residuals as the true residuals, and updates the solution accordingly. This heuristic decision was motivated by the observation that the BICG residual polynomials typically reduce the norm of the starting vector, and so one would hope that applying the BICG polynomial again (to an already-reduced residual) might reduce it further. But because it squares the polynomials, CGS might have more irregular convergence than BICG. As a side effect, larger intermediate quantities in CGS could worsen local round off, leading to a (faster) deviation between computed and true residuals.

3.3.1 BIOMINS

We consider the BIOMINS form of CGS, the original from variant [35], presented in Alg. 9. Alg. 9 has been slightly reorganized to absorb his auxiliary quantities u and v. Some notation changed from his: $(\alpha, \beta, \rho, \sigma, q) \to (\omega, \psi, \delta, \delta', s)$. Our notation also borrows from §7.4.1 of [33] and §14 of [16].

3.3.2 CA-BIOMINS

Of the four vector iterates in Alg. 9, three are explicitly multiplied by A. The standard version in [35] only performs two actual SpMV operations per iteration. Our CA implementations will involve a call to `Akx` with three right-hand sides, and to the $2s$ power, to replace s iterations. This is a $3\times$ increase in sparse flops, more depending on redundant flops. We note that a CA-BIORESS (three-term CGS) implementation would only involve two right-hand sides, in exchange for maintaining a $2s$-step history. Furthermore, we can derive an inconsistent version of CA-BIORESS with the same advantages as inconsistent BIORES. However, we believe that CA-BICGSTAB is a better alternative than any of these

Algorithm 7 Communication-avoiding consistent BIORES

Require: Initial approximation x_0 for solving $Ax = b$. Compute $r_0 = (b - Ax_0)$.

Choose $\tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$.

Base case: $R_{-1} = 0_{n,s}$ and $\tilde{R}_{-1} = 0_{n,s}$, $x_{-1} = 0_{n,1}$

Base case: $\beta_{i-1} = \tilde{\beta}_{i-1} = \alpha_i = \tilde{\alpha}_i = \delta_i = \gamma_i = \tilde{\gamma}_i = 0$ for $i = -s : -1$

repeat for $\ell = 0, 1, 2 \ldots$ and $m = s\ell$,

  Set $\hat{R} = [R_{\ell-1}, 0_{n,s+1}]$ and $\hat{\tilde{R}} = \left[\tilde{R}_{\ell-1}, 0_{n,s+1}\right]$.

  $\left[\hat{R}(:, s+1 : 2s+1), B\right] = \texttt{Akx}\,(A,\, s,\, r_m)$, $\left[\hat{\tilde{R}}(:, s+1 : 2s+1), \tilde{B}\right] = \texttt{Akx}\left(A^H,\, s,\, \tilde{r}_m\right)$.

  Compute $G = \hat{\tilde{R}}^H \hat{R}$.

  $\left[c_{-s}^0, c_{-s+1}^0, \ldots, c_{-1}^0\right] = \begin{bmatrix} I_{s,s} \\ 0_{s+1,s} \end{bmatrix}$, $\left[d_{-s}^0, d_{-s+1}^0, \ldots, d_{-1}^0\right] = \begin{bmatrix} I_{s,s} \\ 0_{s+1,s} \end{bmatrix}$.

  for $j = -s : -2$, **do**

    for $k = 1 : s + j + 1$ **do**

      $$c_{j+1}^k = \left[c_j^{k-1}, c_{j+1}^{k-1}, c_{j+2}^{k-1}\right] \begin{bmatrix} -\beta_{m+j} \\ -\alpha_{m+j+1} \\ \gamma_{m+j+1} \end{bmatrix}, \quad d_{j+1}^k = \left[d_j^{k-1}, d_{j+1}^{k-1}, d_{j+2}^{k-1}\right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix}.$$

    end for

  end for

  $[c_0^0, c_0^1, \ldots, c_0^s] = \begin{bmatrix} 0_{s,s+1} \\ B \end{bmatrix}$, $[d_0^0, d_0^1, \ldots, d_0^s] = \begin{bmatrix} 0_{s,s+1} \\ \tilde{B} \end{bmatrix}$, $e_{-1} = \begin{bmatrix} 0_{2s+2,1} \\ 1 \end{bmatrix}$, $e_0 = \begin{bmatrix} 0_{2s+1,1} \\ 1 \\ 0 \end{bmatrix}$.

  for $j = 0 : s - 1$, **do**

    Set $\alpha_{m+j} = \left(d_j^0\right)^H G c_j^1$ and $\tilde{\alpha}_{m+j} = \overline{\alpha_{m+j}}$.

    Set $\beta_{m+j-1} = \overline{\tilde{\gamma}_{m+j-1}} \delta_{m+j} / \delta_{m+j-1}$, with $\beta_{-1} = 0$.

    Set $\tilde{\beta}_{m+j-1} = \overline{\gamma_{m+j-1} \delta_{m+j} / \delta_{m+j-1}} = \overline{\beta_{m+j-1} \gamma_{m+j-1}} / \tilde{\gamma}_{m+j-1}$, with $\tilde{\beta}_{-1} = 0$.

    Set $\gamma_{m+j} = \alpha_{m+j} - \beta_{m+j-1}$. If $\gamma_{m+j} = 0$, declare *pivot breakdown* and STOP.

    Choose $\tilde{\gamma}_{m+j} \neq 0$ arbitrarily.

    $$c_{j+1}^k = \left(c_j^{k+1} - \left[c_{j-1}^k, c_j^k\right] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix} \right) / \gamma_{m+j}, \text{ for } k = 0 : s - j - 1.$$

    $$d_{j+1}^k = \left[d_j^{k-1}, d_{j+1}^{k-1}, d_{j+2}^{k-1}\right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix}, \text{ for } k = 0 : s - j - 1.$$

    $$e_{j+1} = -\left(\begin{bmatrix} c_j^0 \\ 0_{2,1} \end{bmatrix} + [e_{j-1}, e_j] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix} \right) / \gamma_{m+j}.$$

    Set $\delta_{m+j+1} = \left(d_{j+1}^0\right)^H G c_{j+1}^0$. If $\delta_{m+j+1} = 0$, STOP.

      If $\hat{R} c_{j+1}^0 = 0$, terminate with $x_{\text{ex}} = \left[\hat{R}, x_{m-1}, x_m\right] e_{j+1}$.

      If $\hat{\tilde{R}} d_{j+1} \neq 0$, declare *Lanczos breakdown*;

      If $\hat{\tilde{R}} d_{j+1} = 0$, declare *left termination.*

  end for

  $r_{m+s} = \hat{R} c_s^0$, $\tilde{r}_{m+s} = \hat{\tilde{R}} d_s^0$, $x_{m+s} = \left[\hat{R}, x_{m-1}, x_m\right] e_s$.

until $||r_{m+s}||$ is small.

Terminate with approximate solution $x \approx x_{m+s}$.

Algorithm 8 Communication-avoiding inconsistent BIORES

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $r_0 = (b - Ax_0)/\gamma_{-1}$ for some $\gamma_{-1} \neq 0$, e.g., so $||r_0|| = 1$.

Redefine $x_0 = x_0/\gamma_{-1}$. Choose $\tilde{r}_0$ so that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$.

Base case: $R_{-1} = 0_{n,s}$ and $\tilde{R}_{-1} = 0_{n,s}$,$x_{-1} = 0_{n,1}$

Base case: $\beta_{i-1} = \tilde{\beta}_{i-1} = \alpha_i = \tilde{\alpha}_i = \delta_i = \gamma_{i-1} = \tilde{\gamma}_{i-1} = \pi_i = 0$ for $i = -s : -1$

repeat for $\ell = 0, 1, 2 \ldots$ and $m = s\ell$,

  Set $\hat{R} = [R_{\ell-1}, 0_{n,s+1}]$ and $\hat{\tilde{R}} = \left[\tilde{R}_{\ell-1}, 0_{n,s+1}\right]$.

  $\left[\hat{R}(:, s+1:2s+1), B\right] = \texttt{Akx}\,(A,\, s,\, r_m)$, $\left[\hat{\tilde{R}}(:, s+1:2s+1), \tilde{B}\right] = \texttt{Akx}\left(A^H,\, s,\, \tilde{r}_m\right)$.

  Compute $G = \hat{\tilde{R}}^H \hat{R}$.

  $\left[c_{-s}^0, c_{-s+1}^0, \ldots, c_{-1}^0\right] = \begin{bmatrix} I_{s,s} \\ 0_{s+1,s} \end{bmatrix}$, $\left[d_{-s}^0, d_{-s+1}^0, \ldots, d_{-1}^0\right] = \begin{bmatrix} I_{s,s} \\ 0_{s+1,s} \end{bmatrix}$.

  for $j = -s : -2$, **do**

    for $k = 1 : s + j + 1$ **do**

      $c_{j+1}^k = \left[c_j^{k-1}, c_{j+1}^{k-1}, c_{j+2}^{k-1}\right] \begin{bmatrix} -\beta_{m+j} \\ -\alpha_{m+j+1} \\ \gamma_{m+j+1} \end{bmatrix}$, $d_{j+1}^k = \left[d_j^{k-1}, d_{j+1}^{k-1}, d_{j+2}^{k-1}\right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix}$.

    end for

  end for

  $[c_0^0, c_0^1, \ldots, c_0^s] = \begin{bmatrix} 0_{s,s+1} \\ B \end{bmatrix}$, $[d_0^0, d_0^1, \ldots, d_0^s] = \begin{bmatrix} 0_{s,s+1} \\ \tilde{B} \end{bmatrix}$, $e_{-1} = \begin{bmatrix} 0_{2s+2,1} \\ 1 \end{bmatrix}$, $e_0 = \begin{bmatrix} 0_{2s+1,1} \\ 0 \\ 1 \end{bmatrix}$.

  for $j = 0 : s - 1$, **do**

    $\alpha_{m+j} = \left(d_j^0\right)^H G c_j^1$, $\tilde{\alpha}_{m+j} = \overline{\alpha_{m+j}}$, $\beta_{m+j-1} = \overline{\tilde{\gamma}_{m+j-1}}\delta_{m+j}/\delta_{m+j-1}$, with $\beta_{-1} = 0$.

    Set $\tilde{\beta}_{m+j-1} = \overline{\gamma_{m+j-1}\delta_m/\delta_{m+j-1}} = \overline{\beta_{m+j-1}\gamma_{m+j-1}}/\tilde{\gamma}_{m+j-1}$, with $\tilde{\beta}_{-1} = 0$.

    Choose $\gamma_{m+j} \neq 0$ and $\tilde{\gamma}_{m+j} \neq 0$ arbitrarily.

    $\pi_{m+j+1} = -\left(\alpha_{m+j}\pi_{m+j} + \beta_{m+j-1}\pi_{m+j-1}\right)/\gamma_{m+j}$ with $\pi_0 = 1/\gamma_{-1}$.

    $c_{j+1}^k = \left(c_j^{k+1} - \left[c_{j-1}^k, c_j^k\right] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix}\right)/\gamma_{m+j}$, for $k = 0 : s - j - 1$.

    $d_{j+1}^k = \left[d_j^{k-1}, d_{j+1}^{k-1}, d_{j+2}^{k-1}\right] \begin{bmatrix} -\tilde{\beta}_{m+j} \\ -\tilde{\alpha}_{m+j+1} \\ \tilde{\gamma}_{m+j+1} \end{bmatrix}$, for $k = 0 : s - j - 1$.

    $e_{j+1} = -\left(\begin{bmatrix} c_j^0 \\ 0_{2,1} \end{bmatrix} + [e_{j-1}, e_j] \begin{bmatrix} \beta_{m+j-1} \\ \alpha_{m+j} \end{bmatrix}\right)/\gamma_{m+j}$.

    Set $\delta_{m+j+1} = \left(d_{j+1}^0\right)^H G c_{j+1}^0$. If $\delta_{m+j+1} = 0$, STOP.

      If $\hat{R}c_{j+1}^0 = 0$, terminate with $x_{\text{ex}} = \left(\left[\hat{R}, x_{m-1}, x_m\right] e_{j+1}\right)/\pi_{m+j+1}$.

      If $\hat{\tilde{R}}d_{j+1} \neq 0$, declare *Lanczos breakdown*;

      If $\hat{\tilde{R}}d_{j+1} = 0$, declare *left termination.*

  end for

  $r_{m+s} = \hat{R}c_s^0$, $\tilde{r}_{m+s} = \hat{\tilde{R}}d_s^0$, $x_{m+s} = \left[\hat{R}, x_{m-1}, x_m\right] e_s$.

until $||r_{m+s}||$ is small.

Terminate with approximate solution $x \approx x_{m+s}/\pi_{m+s}$.

Algorithm 9 BIOMINS

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $s_0 = p_0 = r_0 = b - Ax_0$.
Choose $\tilde{r}_0$ arbitrary such that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{r}_0^H A p_0 \neq 0$.
Set $\psi_{-1} = 0$ and $s_{-1} = 0_{n,s}$.
repeat for $m = 0, 1, \ldots,$
    Set $\delta_m' = \tilde{r}_0^H A p_m$. If $\delta_m' = 0$, STOP and declare *pivot breakdown*.
    Take steps along search directions:
        $\omega_m = \delta_m / \delta_m'$.
        $s_m = r_m + \psi_{m-1} s_{m-1} - \omega_m A p_m$.
        $r_{m+1} = r_m - 2\omega_m A r_m - 2\omega_m \psi_{m-1} A s_{m-1} - \omega_m^2 A^2 p_m$.
    Update candidate solution:
        $x_{m+1} = x_m + 2\omega_m r_m + 2\omega_m \psi_{m-1} s_{m-1} + \omega_m^2 A p_m$.
    Set $\delta_{m+1} = \tilde{r}_0^H r_{m+1}$. If $\delta_{m+1} = 0$, STOP.
        If $r_{m+1} = 0_{n,1}$, terminate with $x_{\text{ex}} = x_{m+1}$.
        Otherwise, declare *Lanczos breakdown*.
    Update search direction:
        $\psi_m = -\delta_{m+1} / \delta_m$.
        $p_{m+1} = r_{m+1} + 2\psi_m s_m + \psi_m^2 p_m$.
until $||r_{m+1}||$ is small.
Terminate with approximate solution $x \approx x_{m+1}$.

methods, in terms of stability and performance, and will not discuss any CGS variants other than consistent BIOMINS.

We start with the iterates (r_m, p_m, s_{m-1}) given by the initial data ($m = 0$) or from a previous iteration ($m > 0$). We call the matrix powers kernel to compute

$$\begin{aligned}
\hat{P} &= \left[\rho_0(A)\, p_m, \rho_1(A)\, p_m, \ldots, \rho_s(A)\, p_m\right] \\
\hat{R} &= \left[\rho_0(A)\, r_m, \rho_1(A)\, r_m, \ldots, \rho_s(A)\, r_m\right] \\
\hat{S} &= \left[\rho_0(A)\, s_{m-1}, \rho_1(A)\, s_{m-1}, \ldots, \rho_s(A)\, s_{m-1}\right]
\end{aligned}$$

where $\rho_j(z)$ and $\tilde{\rho}_j(z)$ are polynomials of (exact) degree j. (In principle, we might use a different polynomial for each basis matrix.) `Akx` also provides a change-of-basis matrix B that allows us to convert back to the Krylov basis,

$$\begin{aligned}
\left[\hat{P}, \hat{R}, \hat{S}\right] \begin{bmatrix} B \\ O_{4s+2,2s+1} \end{bmatrix} &= \left[p_m, Ap_m, \ldots, A^{2s} p_m\right] \\
\left[\hat{P}, \hat{R}, \hat{S}\right] \begin{bmatrix} 0_{2s+1,2s+1} \\ B \\ 0_{2s+1,2s+1} \end{bmatrix} &= \left[r_m, Ar_m, \ldots, A^{2s} r_m\right] \\
\left[\hat{P}, \hat{R}, \hat{S}\right] \begin{bmatrix} 0_{4s+2,2s+1} \\ B \end{bmatrix} &= \left[s_{m-1}, As_{m-1}, \ldots, A^{2s} s_{m-1}\right]
\end{aligned}$$

although we apply this transformation implicitly.

Using these $2s$-step Krylov bases, substituting the coefficient vectors, defined for $0 \leq k \leq 2s$,

$$\begin{aligned}\left[\hat{P}, \hat{R}, \hat{S}\right] a_j^k &= A^k p_{m+j} \\ \left[\hat{P}, \hat{R}, \hat{S}\right] b_j^k &= \left(A^H\right)^k r_{m+j} \\ \left[\hat{P}, \hat{R}, \hat{S}\right] c_j^k &= A^k s_{m+j}\end{aligned}$$

and row vector $g = \tilde{r}_0^H \left[\hat{P}, \hat{R}, \hat{S}\right]$, we arrive at a communication-avoiding version in Alg. 10.

3.4 Communication-avoiding biconjugate gradient-stabilized method (CA-BICGSTAB)

The CGS method reinforced the idea that the BICG algorithm only exploits the fact that the right Krylov basis is biorthogonal to the left Krylov basis - the biorthogonality need not be mutual. Lanczos-type product methods (LTPMs) use a different polynomial recurrence for the left basis and combine this with the CGS strategy, applying this second polynomial to the right basis in order to avoid computing the left basis at all. The hope is that the left polynomial further reduces the residual. Many LTPMs, including BICGSTAB and its variants, choose the left polynomial to have smoothing or stabilizing properties. As with Lanczos (versus, *e.g.,* Arnoldi), a shorter polynomial recurrence is preferable for performance reasons.

In the case of BICGSTAB, the left polynomial takes a two-term recurrence, and amounts to extending the Krylov space by one dimension (a new basis vector), and taking a steepest descent step in that direction (line search). This is a local one-dimensional minimization, which should result in a smoother convergence curve and avoid possible overflow conditions in CGS. However, an issue with BICGSTAB is that if the input data is all real, the stabilizing polynomial will have only real zeros. Such a polynomial will not reduce error components in the direction of eigenvectors corresponding to eigenvalues with large imaginary components (relative to their real components). Matrices with such a spectrum are also more susceptible to a *minimization breakdown* in BICGSTAB, a new breakdown condition.

These two drawbacks to BICGSTAB are addressed in the literature by many newer LTPMs by using (at least) two-dimensional residual smoothing. Other smoothers, like Chebyshev polynomials, have also been considered. In this work, we only demonstrate our communication-avoiding approach on the simplest LTPMs (CGS and BICGSTAB). We conjecture that our approach generalizes to all LTPMs with short polynomial recurrences, and that the flexibility to choose a polynomial basis in `Akx` could accelerate the computation of the left polynomials, when the recurrence coefficients are known in advance.

We also note that BICGSTAB(ℓ) is a promising direction for future work, especially once combined with the communication-avoiding tall-skinny QR kernel, as described in [20].

We present the version of BICGSTAB from [33] in Alg. 11. Note that the vector iterates s_m are unrelated to the scalar s; no ambiguity will arise since the former is always subscripted while the latter is never.

Algorithm 10 Communication-avoiding BIOMINS

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $s_0 = p_0 = r_0 = b - Ax_0$.
Choose $\tilde{r}_0$ arbitrary such that $\delta_0 = \tilde{r}_0^H r_0 \neq 0$ and $\delta_0' = \tilde{r}_0^H A p_0 \neq 0$.
Set $\psi_{-1} = 0$ and $s_{-1} = 0_{n,s}$.
repeat for $m = 0, 1, \ldots,$
    Compute three $2s$-step Krylov bases using the matrix powers kernel (`Akx`):
        $\left[\left[\hat{P}, \hat{R}, \hat{S}\right], B\right] = \texttt{akx}\left(A, 2s, [p_m, r_m, s_{m-1}]\right)$.
    Compute $g = \tilde{r}_0^H \left[\hat{P}, \hat{R}, \hat{S}\right]$.
    Initialize coefficient vectors
        $[a_0^0, a_0^1, \ldots, a_0^s] = \begin{bmatrix} B \\ O_{4s+2,2s+1} \end{bmatrix}$,
        $[b_0^0, b_0^1, \ldots, b_0^s] = \begin{bmatrix} 0_{2s+2,2s+1} \\ B \\ O_{2s+2,2s+1} \end{bmatrix}$, and
        $\left[c_{-1}^0, c_{-1}^1, \ldots, c_{-1}^s\right] = \begin{bmatrix} O_{4s+2,2s+1} \\ B \end{bmatrix}$, and
        $e_0 = \begin{bmatrix} 0_{6s+3,1} \\ 1 \end{bmatrix}$.
    for $j = 0 : s-1$, **do**
        $\delta_{m+j}' = g a_j^1$. If $\delta_{m+j}' = 0$, declare *pivot breakdown* and STOP.
        Take steps along search direction:
            $\omega_{m+j} = \delta_{m+j} / \delta_{m+j}'$.
            $c_j^k = b_j^k + \psi_{m+j-1} c_{j-1}^k - \omega_{m+j} a_j^{k+1}$, for $k = 0 : 2(s-j-1)$.
            $b_{j+1}^k = b_j^k - 2\omega_{m+j} b_j^{k+1} - 2\omega_{m+j}\psi_{m+j-1} c_{m+j-1}^{k+1} - \omega_{m+j}^2 a_m^{k+1}$, for $k = 0 : 2(s-j-1)$.
        Update candidate solution: $e_{j+1} = e_j + 2\omega_m \begin{bmatrix} b_j^0 \\ 0 \end{bmatrix} + 2\omega_m \psi_{m-1} \begin{bmatrix} c_{j-1}^0 \\ 0 \end{bmatrix} + \omega_m^2 \begin{bmatrix} a_j^1 \\ 0 \end{bmatrix}$.
        Set $\delta_{m+j+1} = g b_{j+1}^0$. If $\delta_{m+j+1} = 0$, STOP.
            If $\left[\hat{P}, \hat{R}, \hat{S}\right] b_{j+1}^0 = 0_{n,1}$, terminate with $x_{\text{ex}} = \left[\hat{P}, \hat{R}, \hat{S}, x_m\right] e_{j+1}$.
            Otherwise, declare *Lanczos breakdown.*
        Update search direction:
            $\psi_{m+j} = -\delta_{m+j+1} / \delta_{m+j}$.
            $a_{j+1}^k = b_{j+1}^k + 2\psi_{m+j} c_j^k + \psi_{m+j}^2 a_{m+j}$, for $k = 0 : 2(s-j-1)$.
    end for
    Recover iterates from last inner iteration.
        $r_{m+s} = \left[\hat{P}, \hat{R}, \hat{S}\right] b_s^0$, $p_{m+s} = \left[\hat{P}, \hat{R}, \hat{S}\right] a_s^0$,
        $s_{m+s-1} = \left[\hat{P}, \hat{R}, \hat{S}\right] c_{s-1}^0$, and $x_{m+s} = \left[\hat{P}, \hat{R}, \hat{S}, x_m\right] e_s$.
until $||r_{m+s}||$ is small.
Terminate with approximate solution $x \approx x_{m+s}$.

Algorithm 11 BICGSTAB

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)$.
Choose $\tilde{r}_0$ arbitrary.
repeat for $m = 0, 1, \ldots,$
    $\alpha_m = \left(\tilde{r}_0^H r_m\right) / \left(\tilde{r}_0^H A p_m\right)$.
    $s_m = r_m - \alpha_m A p_m$
    $\omega_m = \left(s_m^H A s_m\right) / \left(\left(A s_m\right)^H A s_m\right)$.
    $x_{m+1} = x_m + \alpha_m p_m + \omega_m s_m$.
    $r_{m+1} = s_m - \omega_m A s_m$.
    $\beta_m = \left(\tilde{r}_0^H r_{m+1}\right) / \left(\tilde{r}_0^H r_m\right) \times (\alpha_m / \omega_m)$.
    $p_{m+1} = r_{m+1} + \beta_m \left(p_m - \omega_m A p_m\right)$.
until $||r_{m+1}||$ is small.
Terminate with approximate solution $x \approx x_{m+1}$.

3.4.1 CA-BICGSTAB

First, we note that the vector iterates s_m are auxiliary quantities introduced to simplify the underlying coupled two-term recurrences

$$\begin{aligned} r_{m+1} &= (I - \omega_m A)\left(r_m - \alpha_m A p_m\right) \\ p_{m+1} &= r_{m+1} + \beta_m \left(I - \omega_m A\right) p_m \end{aligned}$$

and we do not need them. We substitute the definition $s_m = r_m - \alpha_m A p_m$ to recover these recurrences, and this also gives us new expressions for

$$\begin{aligned} \omega_m &= \frac{r_m^H A r_m - \alpha_m r_m^H A^2 p_m - \overline{\alpha_m} p_m^H A^H A r_m + \overline{\alpha_m} \alpha_m p_m^H A^H A^2 p_m}{r_m^H A^H A r_m - \alpha_m r_m^H A^H A^2 p_m - \overline{\alpha_m} p_m^H \left(A^H\right)^2 A r_m + \overline{\alpha_m} \alpha_m \left(A^H\right)^2 A^2 p_m} \\ x_{m+1} &= x_m + \alpha_m p_m + \omega_m \left(r_m - \alpha_m A p_m\right) \end{aligned}$$

Since BICGSTAB avoids multiplications by A^H, while applying A twice per iteration, the underlying Krylov space is extended by two dimensions per iteration, as opposed to BICG. Thus, in order to take s steps of BICGSTAB in a communication-avoiding manner, we need to explicitly extend the Krylov spaces by $2s$ dimensions with `Akx`. Also, we compute inner products involving the shadow residual vector $\tilde{r}_0$ as well as iterates $\{r, p\}$. We will replace the former using a "Gram vector" and the latter with a "Gram matrix:"

$$\begin{aligned} g &= \tilde{r}_0^H \left[\hat{P}, \hat{R}\right] \\ G &= \left[\hat{P}, \hat{R}\right]^H \left[\hat{P}, \hat{R}\right] \end{aligned}$$

Our call to `Akx` takes the two source vectors $[p_m, r_m]]$ and produce the $2s$-step bases $[\hat{P}, \hat{R}]$, similar to CA-BICG. Lastly we introduce the coefficient vectors

$$\begin{aligned} \left[\hat{P}, \hat{R}\right] b_j^k &= A^k r_{m+j} \\ \left[\hat{P}, \hat{R}\right] a_j^k &= A^k p_{m+j} \end{aligned}$$

and use these vectors as well as G and g to reach the communication-avoiding version, in Alg. 12.

A more costly version of CA-BICGSTAB can be derived using the auxiliary vector iterates s_m in 11. In this case, we call `Akx` with three source vectors to compute three s-step bases, as well as introduce a third coefficient vector. We do not present this algorithm here, but it can be easily derived like Alg. 12. When we performed our numerical experiments, we used this three-iterate version in order to compare with MATLAB BICGSTAB (which also uses three iterates). It is future work to see if and how the floating-point properties of the two-iterate version (Alg. 12) differ from this three-iterate version. We note that the three-iterate version does more flops than Alg. 12, and so conjecture that Alg. 12 will have no worse round off. That is, we conjecture our numerical experiments will produce similar (if not better) results when we implement Alg. 12.

3.5 Preconditioning

Preconditioning is a technique frequently used in practice to accelerate the convergence of KSMs by replacing $Ax = b$ by an equivalent linear system $\tilde{A}y = c$, where $\tilde{A} = M_L^{-1}AM_R^{-1}$, $y = M_R x$, and $c = M_L^{-1}b$. We call M_L and M_R left and right preconditioners, respectively; these preconditioners are chosen so that $\tilde{A}$ has a smaller condition number than A. When A and $\tilde{A}$ are normal, this lowers theoretic upper bounds on the worst-case rate of convergence for Krylov methods. Selecting an appropriate preconditioner is a wide field of study itself, which we do not discuss further here.

Our concern is primarily the extension of our methods to the preconditioned case, such that communication is still avoided. Foremost, we require that the preconditioned matrix $\tilde{A}$ can be well-partitioned or have a special structure we can exploit - otherwise, we cannot avoid communication with the matrix powers kernel. However, even if communication can not be avoided by use of the matrix powers kernel, we could compute the s SpMVs in a straightforward way in the outer loop and still save in communication costs by blocking the dot products in the inner loop, as discussed in [37]. When $\tilde{A}$ can be well-partitioned (and is explicitly available), our usual communication-avoiding matrix powers kernel can still be used. The simplest case for which this holds is (block) diagonal preconditioning, used in many scientific applications. A Krylov subspace for $\tilde{A}$ can be computed using the same dependencies required for A , as application of the diagonal preconditioner matrix requires only local work. If the matrix is dense, but has a special structure we can exploit (i.e., the low-rank structure in Hierarchical Semiseparable (HSS) Matrices), we can still avoid communication with a different variant of the matrix powers kernel, which will be discussed in future work.

The preconditioned variants of our CA-KSMs require a few additional changes from the unpreconditioned versions. In the case of left- or split-preconditioning, our algorithms will report the preconditioned residuals, z_m , instead of the unpreconditioned residuals, r_m. This means that we must compute $||r_m||_2 = ||M^{-1}z_m||_2$ in order to check convergence, where M is the left preconditioner, or left part of the split preconditioner. The communication cost incurred by the preconditioner solve is a lower order term (compared to the preconditioned matrix powers kernel invocation), assuming M^{-1} is well-partitioned.

Others have discussed the use of polynomial preconditioners [32]. Polynomial precondi-

Algorithm 12 Communication-avoiding BICGSTAB

Require: Initial approximation x_0 for solving $Ax = b$.

Compute $p_0 = r_0 = (b - Ax_0)$.
Choose $\tilde{r}_0$ arbitrary.
repeat for $m = 0, 1, \ldots,$
    Extend the two Krylov bases by $2s$ dimensions, using the matrix powers kernel (`Akx`).
        $\left[\left[\hat{P}, \hat{R}\right], B\right] = \texttt{Akx}\left(A,\, 2s,\, [p_m, r_m]\right)$.
    Compute $g = \tilde{r}_0^H \left[\hat{P}, \hat{R}\right]$.
    Compute $G = \left[\hat{\tilde{P}}, \hat{\tilde{R}}\right]^H \left[\hat{P}, \hat{R}\right]$.
    Initialize coefficient vectors
        $[b_0^0, b_0^1, \ldots, b_0^s] = \begin{bmatrix} 0_{s+1,s+1} \\ B \end{bmatrix}$, $[a_0^0, a_0^1, \ldots, a_0^s] = \begin{bmatrix} B \\ 0_{s+1,s+1} \end{bmatrix}$. $e_0 = \begin{bmatrix} 0_{2s+2,1} \\ 1 \end{bmatrix}$.
    for $j = 0 : s - 1$, **do**
        $\alpha_{m+j} = g b_j^0 / g a_j^1$.
        $\omega_{m+j} = \frac{\left(b_j^0\right)^H G b_j^1 - \alpha_{m+j}\left(b_j^0\right)^H G a_j^2 - \overline{\alpha_{m+j}}\left(a_j^1\right)^H G b_j^1 + \overline{\alpha_{m+j}}\alpha_{m+j}\left(a_j^1\right)^H G a_j^2}{\left(b_j^1\right)^H G b_j^1 - \alpha_{m+j}\left(b_j^1\right)^H G a_j^2 - \overline{\alpha_{m+j}}\left(a_j^2\right)^H G b_j^1 + \overline{\alpha_{m+j}}\alpha_{m+j}\left(a_j^2\right)^H G a_j^2}$.
        Update candidate solution:
            $e_{j+1} = e_j + \alpha_{m+j} \begin{bmatrix} a_j^0 \\ 0 \end{bmatrix} + \omega_{m+j} \begin{bmatrix} b_j^0 \\ 0 \end{bmatrix} - \alpha_{m+j}\omega_{m+j} \begin{bmatrix} a_j^1 \\ 0 \end{bmatrix}$.
        Update residual:
            $b_{j+1}^k = b_j^k - \alpha_{m+j} a_j^{k+1} - \omega_{m+j} b_j^{k+1} + \alpha_{m+j}\omega_{m+j} a_j^{k+2}$, for $k = 0 : 2(s - j - 1)$.
        Update search direction:
            $\beta_{m+j} = \left(g b_{j+1}^0 / g b_j^0\right) \times (\alpha_{m+j} / \omega_{m+j})$.
            $a_{j+1}^k = b_{j+1}^k + \beta_{m+j} a_j^k - \beta_{m+j}\omega_{m+j} a_j^{k+1}$, for $k = 0 : 2(s - j - 1)$.
    end for
    Recover iterates from last inner iteration.
        $r_{m+s} = \left[\hat{P}, \hat{R}\right] b_s^0$, $p_{m+s} = \left[\hat{P}, \hat{R}\right] a_s^0$,
        $x_{m+s} = \left[\hat{P}, \hat{R}, x_m\right] e_s$.
until $||r_{m+s}||$ is small.
Terminate with approximate solution $x \approx x_{m+s}$.

tioning could be easily incorporated in our methods, as the general implementation of the matrix powers kernel computes polynomials of A. Hoemmen et al. have also devised an algorithm to avoid communication in the case of semiseparable, hierarchical, and hierarchical semiseparable matrices (matrices with low-rank off-diagonal blocks) [20, 12]. Such matrices arise when, for example, A is tridiagonal: the inverse of a tridiagonal, although dense, has the property that any submatrix strictly above or strictly below the diagonal has rank 1. Hoemmen exploits such low-rank off-diagonal blocks in order to avoid communication when applying $\tilde{A}$, although he notes that his algorithm introduces significantly additional computational requirements. In future work, we will investigate Hoemmen's approach and determine its practicality, as well as investigate communication-avoiding approaches for other classes of preconditioners.

4 Convergence

4.1 Choice of Basis

As discussed in [12, 27], the matrix powers kernel extends the Krylov spaces using polynomials ρ which may not be simple monomials. For simplicity, we only consider polynomials that may be computed by a three-term recurrence. Given a starting vector v, we compute a sequence of vectors

$$v_j = \begin{cases} \gamma v & j = 0 \\ \alpha_1 A v_0 + \beta_1 v_0 & j = 1 \\ (\alpha_j A + \beta_j I)\, v_{j-1} + \gamma_j v_{j-2} & j > 1 \end{cases}$$

Note that α_j and β_j are defined when $j \geq 1$ and γ_j is defined when $j \geq 2$. The first vector is scaled by some scalar γ. We rearrange terms to uncover the identity

$$Av_j = \begin{cases} -\frac{\beta_1}{\alpha_1} v_0 + \frac{1}{\alpha_1} v_1 & j = 0 \\ -\frac{\gamma_{j+1}}{\alpha_{j+1}} v_{j-1} - \frac{\beta_{j+1}}{\alpha_{j+1}} v_j + \frac{1}{\alpha_{j+1}} v_{j+1} & j > 0 \end{cases}$$

or written in matrix form,

$$A \begin{bmatrix} v_0 & \cdots & v_j \end{bmatrix} = \begin{bmatrix} v_0 & \cdots & v_{j+1} \end{bmatrix} \begin{pmatrix} -\frac{\beta_1}{\alpha_1} & -\frac{\gamma_2}{\alpha_2} & & \\ \frac{1}{\alpha_1} & -\frac{\beta_2}{\alpha_2} & \ddots & \\ & \frac{1}{\alpha_2} & \ddots & -\frac{\gamma_{j+1}}{\alpha_{j+1}} \\ & & \ddots & -\frac{\beta_{j+1}}{\alpha_{j+1}} \\ & & & \frac{1}{\alpha_{j+1}} \end{pmatrix}$$

$$AV_j = V_{j+1} \hat{B}_{j+1}$$

Now we derive a change-of-basis matrix B_j such that we transform from our polynomial basis to the Krylov basis:

$$V_j B_j = \begin{bmatrix} v_0 & \cdots & v_j \end{bmatrix} B_j = \begin{bmatrix} v & \cdots & A^j v \end{bmatrix} = K_j$$

where K_j is a Krylov matrix. The first column of B_j is obvious: $B_j(:,1) = \frac{1}{\gamma}e_1$. Now we observe that, when $1 < k \leq j$

$$\begin{aligned} V_j B_j(:,k) &= K_j(:,k) \\ AV_j B_j(:,k) &= AK_j(:,k) \\ &= K_j(:,k+1) \\ &= V_j B_j(:,k+1) \\ V_{j+1}\hat{B}_{j+1} B_j(:,k) &= V_j B_j(:,k+1) \end{aligned}$$

Now, we note that B_j has a nonzero diagonal and is upper triangular in general, since $\text{span}(V_i) = \text{span}(K_i)\ \forall i \geq 0$, based on the assumption that the polynomials p^i are of degree i exactly. This means that the column vector $B_j(:,k)$ is zero in rows $(k+1 : j+1)$. Furthermore, $\hat{B}_{j+1}$ is upper Hessenberg, so $\hat{B}_{j+1}(:,1:k) = \begin{bmatrix} \hat{B}_{j+1}(1:k+1,1:k) \\ 0_{j-k+1,k} \end{bmatrix}$. We will exploit these zeros by splitting the matrices blockwise:

$$\begin{aligned} &V_{j+1}\hat{B}_{j+1}B_j(:,k) \\ &\quad = \begin{bmatrix} V_{j+1}(:,1:k+1) & V_{j+1}(:,k+2:j+2) \end{bmatrix} \\ &\quad \times \begin{bmatrix} \hat{B}_{j+1}(1:k+1,1:k) & \hat{B}(1:k+1,k+1:j+1) \\ 0_{j-k+1,j} & \hat{B}_{j+1}(k+2:j+2,k+1:j+1) \end{bmatrix} \begin{bmatrix} B_j(1:k,k) \\ 0_{j-k+1,1} \end{bmatrix} \\ &\quad = \begin{bmatrix} V_{j+1}(:,1:k+1) & V_{j+1}(:,k+2:j+2) \end{bmatrix} \begin{bmatrix} \hat{B}_{j+1}(1:k+1,1:k) \\ 0_{j-k+1,j} \end{bmatrix} B_j(1:k,k) \\ &\quad = V_{j+1}(:,1:k+1)\,\hat{B}_{j+1}(1:k+1,1:k)\,B_j(1:k,k) \\ &\quad = V_k\hat{B}_j(1:k+1,1:k)\,B_j(1:k,k) \end{aligned}$$

For the right hand side, we have

$$\begin{aligned} V_j B_j(:,k+1) &= \begin{bmatrix} V_j(:,1:k+1) & V_j(:,k+2:j+1) \end{bmatrix} \begin{bmatrix} B_j(1:k+1,k+1) \\ 0_{j-k,1} \end{bmatrix} \\ &= V_j(:,1:k+1)\,B_j(1:k+1,k+1) \\ &= V_k B_j(1:k+1,k+1) \end{aligned}$$

so combining,

$$\begin{aligned} V_k B_j(1:k+1,k+1) &= V_k\hat{B}_j(1:k+1,1:k)\,B_j(1:k,k) \\ B_j(1:k+1,k+1) &= \hat{B}_j(1:k+1,1:k)\,B_j(1:k,k) \end{aligned}$$

This last formula can be used to construct the upper triangular $(j+1)\times(j+1)$ change-of-basis matrix B_j for any three-term polynomial recurrence, starting with $B_j(1,1) = \frac{1}{\gamma}$, given the $(j+1)\times j$ upper Hessenberg matrix $\hat{B}_j$. For an s-step basis, $j = s$.

4.1.1 Monomial Basis

The simplest basis that we can use in our communication-avoiding methods is the monomial basis, $[x,\ Ax,\ A^2x, ...]$.

In the case of the scaled monomial basis, with scaling factors $(\sigma_j)_{j=0}^{s}$, we have $\gamma = \frac{1}{\sigma_0}$ and α_j leading to

$$\hat{B}_s = \begin{pmatrix} 0 & & & & \\ \sigma_1 & 0 & & & \\ & \sigma_2 & 0 & & \\ & & \sigma_3 & \ddots & \\ & & & \ddots & 0 \\ & & & & \sigma_s \end{pmatrix}$$

Note that for the unscaled monomial basis, $\hat{B}_s = [0\,;\, I]$ and $B_s = I$.

However, it is well-known that the monomial basis converges to the principle eigenvector of A, a property that is exploited in the Power Method. Therefore, in finite-precision arithmetic, monomial basis vectors are subject to becoming linearly dependent if we choose s too large. As was previously observed by [9, 31], using a rank-deficient basis leads to convergence failure, as the Krylov subspace can no longer expand. To remedy this problem, one must use a more stable basis. Two options that we consider are the Newton basis and the Chebyshev basis.

4.1.2 Newton Basis

The Newton basis can be written

$$V = [x,\, (A - \theta_1 I)x,\, (A - \theta_2)(A - \theta_1)x,\, ...,\, (A - \theta_s)(A - \theta_{s-1})...(A - \theta_1)x]$$

Here, the shifts, $[\theta_1,\, ..., \theta_s]$, are taken to be Leja points computed using eigenvalue estimates for A, as described in [29]. For the purpose of convergence results in this paper, we use the knowledge of the full spectrum of A to compute s Leja points, rather than eigenvalue estimates. In practice, if the user does not have any information about the spectrum of A, our method can be adapted to perform $O(s)$ steps of Arnoldi and find the eigenvalues of the resulting upper Hessenberg matrix H, giving us eigenvalue estimates for A (Ritz values). The Leja ordering routine is then used to select the s "best" shifts to use. The "best" shifts are defined as being largest in magnitude/furthest away from each other (if two shifts are close together, the resulting basis vectors can again become linearly dependent). We can write the formula for selecting s shifts as follows

$$\begin{aligned} \theta_1 &= \text{argmax}_{z \in K_0} |z| \\ \theta_{j+1} &= \text{argmax}_{z \in K_j} \prod_{k=0}^{j} |z - z_k| \\ &\quad \text{where } K_j \equiv \{\theta_{j+1},\, \theta_{j+2},\, ...,\, \theta_s\} \end{aligned}$$

Using the same notation as for the monomial basis, for the scaled Newton basis, with scaling factors $(\sigma_j)_{j=0}^{s}$ and shifts $(\theta_j)_{j=1}^{s}$, we have $\gamma = \frac{1}{\sigma_0}$ and α_j, which gives

$$\hat{B}_s = \begin{pmatrix} \theta_1 & & & & \\ \sigma_1 & \theta_2 & & & \\ & \sigma_2 & \theta_3 & & \\ & & \sigma_3 & \ddots & \\ & & & \ddots & \theta_s \\ & & & & \sigma_s \end{pmatrix}$$

We can also construct the upper triangular B_s change of basis matrix without constructing $\hat{B}_s$ (although the operations to form B_s are analogous). Start with $B_s = I_s$, and build B according to the following recurrence:

$$B(i,\, j) = B(i-1,\, j-1) + \theta_i \cdot B(i,\, j-1)$$

For example, let's construct B_s for $s = 4$, with eigenvalues estimates $[\theta_1,\, \theta_2,\, \theta_3,\, \theta_4]$. B_s is an $s+1 \times s+1$ matrix. Using the recurrence above, for $s = 4$,

$$B_4 = \begin{bmatrix} 1 & \theta_1 & \theta_1^2 & \theta_1^3 & \theta_1^4 \\ 0 & 1 & \theta_1+\theta_2 & \theta_1^2+(\theta_1+\theta_2)\theta_2 & \theta_1^3+(\theta_1^2+(\theta_1+\theta_2)\theta_2)\lambda\theta_2 \\ 0 & 0 & 1 & \theta_1+\theta_2+\theta_3 & \theta_1^2+(\theta_1+\theta_2)\theta_2+(\theta_1+\theta_2+\theta_3)\theta_3 \\ 0 & 0 & 0 & 1 & \theta_1+\theta_2+\theta_3+\theta_4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

If we have a real matrix with complex eigenvalues and wish to avoid complex arithmetic (for performance reasons), we use instead use the modified Leja ordering [1], which places complex conjugate pairs adjacently so the complex values disappear from the computed basis vectors. The matrix powers kernel for the modified Leja ordering is then used, which is described in [20]. For consistency, if (θ_{j-1}, θ_j) form a complex conjugate pair, we require that θ_{j-1} has the positive imaginary component. We have $\gamma = \frac{1}{\sigma_0}$ and

$$\begin{aligned} \alpha_j &= \frac{1}{\sigma_j} \\ \beta_j &= \begin{cases} -\frac{\theta_j}{\sigma_j} & \theta_j \in \mathbb{R} \\ -\frac{\mathrm{Re}(\theta_j)}{\sigma_j} & \theta_j \text{ is part of a complex conjugate pair} \end{cases} \\ \gamma_j &= \begin{cases} \frac{-\mathrm{Im}(\theta_{j-1})^2}{\sigma_j} & (\theta_{j-1}, \theta_j) \text{ are a complex conjugate pair} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Construction of the change of basis matrix is analogous for the modified Leja ordering. Effects of the modified Leja ordering on stability have not been thoroughly investigated in the literature.

The construction of the Newton basis can result in underflow or overflow in finite precision arithmetic, if the chosen shifts are too close together or too far apart, respectively. Reichel [30] solves this problem by using an estimate for the *capacity* of a subset of the complex plane, defined for the set of shifts as

$$c_k = \prod_{j=1}^{k-1} |\theta_k - \theta_j|^{1/k}$$

The shifts are divided by this quantity before an ordering is chosen. The convergence results shown in this paper do not use this capacity estimate, as we expect that the matrix equilibration routine will sufficiently bound the radius of the spectrum of A - implementation and analysis is considered future work.

4.1.3 Chebyshev Basis

Chebyshev polynomials are another attractive choice for the construction of the Krylov basis. Chebyshev polynomials have the property that over all real polynomials of a degree m on a specified real interval I, the Chebyshev polynomial of degree m minimizes the maximum absolute value on I. This unique property allows for optimal reduction of error in each iteration, making many improvements in KSM convergence and analysis possible.

Joubert and Carey [23] present the recurrence for the scaled and shifted Chebyshev polynomials as

$$\begin{aligned} P_0(z) &= 1 \\ P_1(z) &= \frac{1}{2g}(z-c) \\ P_{k+1}(z) &= \frac{1}{g}\left[(z-c)P_k(z) - \frac{d^2}{4g}P_{k-1}(z)\right] \end{aligned}$$

where the coefficients g, c and d serve to scale and shift the spectrum of A to the unit circle (Note that this only works for real matrices, whose center lies on the real axis). Using the spectrum of A, we select these parameters such that the focus of the ellipse are at $c \pm d$, and $g = \max(|\lambda_i|)$. Here, we have $\gamma = 1$ and

$$\begin{aligned} \alpha_j &= \begin{cases} \frac{1}{2g} & j = 1 \\ \frac{1}{g} & j > 1 \end{cases} \\ \beta_j &= \begin{cases} -\frac{c}{2g} & j = 1 \\ -\frac{c}{g} & j > 1 \end{cases} \\ \gamma_j &= -\frac{d^2}{4g^2} \end{aligned}$$

This gives us the matrix

$$\hat{B}_s = \begin{pmatrix} c & \frac{d^2}{4g} & & & \\ 2g & c & \frac{d^2}{4g} & & \\ & g & c & \ddots & \\ & & g & \ddots & \frac{d^2}{4g} \\ & & & \ddots & c \\ & & & & g \end{pmatrix}$$

We can equivalently derive a formula to construct the change of basis matrix B_s :

$$\begin{aligned} B_s(1,1) &= 1 \\ B_s(i,\, j) &= \begin{cases} 2gB(i-1,\, j-1) + cB(i,j-1) + \frac{d^2}{4g}B(i+1,j-1) & i = 2 \\ gB(i-1,\, j-1) + cB(i,j-1) + \frac{d^2}{4g}B(i+1,j-1) & otherwise \end{cases} \end{aligned}$$

For example, for $s = 4$, we get the change of basis matrix

$$B_4 = \begin{bmatrix} 1 & c & c^2 + \frac{d^2}{2} & c^3 + \frac{3}{2}cd^2 & c^4 + 3c^2d^2 + \frac{3}{8}d^4 \\ 0 & 2g & 2cg & \frac{3}{2}g(4c^2 + d^2) & 2g(4c^3 + 3cd^2) \\ & 0 & 2g^2 & 6cg^2 & 2g^2(6c^2 + d^2) \\ & & 0 & 2g^3 & 8cg^3 \\ & & & 0 & 2g^4 \end{bmatrix}$$

Note that this simplifies to the upper-triangular checkerboard pattern if we take $c = 0$, which means that the ellipse enclosing the eigenvalues of A is already centered around 0 on the real axis.

4.1.4 Basis Scaling vs. Matrix Equilibration

As discussed in [20], successively scaling (e.g., normalizing) the basis vectors reintroduces the global communication requirement between SpMV operations. Even if we wait until the end of the computation and perform the scaling of all basis vectors in one step, this still requires an all-to-all communication in each outer loop iteration (or reading and writing $O(n)$ words to and from slow memory). Although this does not asymptotically increase the communication costs, it does increase the constant factors, and will thus decrease performance (and can also fail to improve stability). In order to avoid this extra cost, we perform matrix equilibration as a preprocessing step before running the CA-KSM. Matrix equilibration is a common technique to improve the spectrum of ill-conditioned matrices. Our results, as well as results from [20], indicate that this technique is often good enough for reducing the condition number of the basis, and thus we do not use the scaled versions of the bases described above (except in the case of Chebyshev, where "scaling" is constant throughout the execution and refers to a scaling of the spectrum of A rather than normalization of the basis vectors in every iteration).

4.2 Convergence Results

In this section, we present convergence results which demonstrate that our Communication-Avoiding methods can maintain stability and convergence similar to the standard implementation for many test matrices. We present convergence results for the two-term variants of CA-BICG and CA-BICGSTAB, as the BICG method is the easiest to analyze, and the BICGSTAB method is most used in practice. All experiments were run with [1, 1, ..., 1] as the RHS and $x_0 = 0$ as the initial guess, which results in a starting residual $r_0 = b$, with $\|r_0\|_2 = \|b\|_2 = \sqrt{n}$.

4.2.1 Diagonal Matrices

We initially show results for diagonal matrices to demonstrate some basic convergence trends. For diagonal matrices, the only significant round off errors in our algorithm stem from the vector and scalar operations (rather than in the matrix powers kernel), making them useful cases for analysis. The diagonal matrices were created with equally spaced eigenvalues on an interval chosen to achieve a given condition number. We first explore how convergence changes for all three bases as a function of s, the basis length, and as a function of the condition number of A. Figure 1 shows results for one matrix with low condition number (~ 10), with s varying, and Figure 2 shows results for one value of s, with the condition number of A varying for the CA-BICG method.

These simple examples demonstrate two basic properties of our CA-KSMs. We can see that if we choose a high s value, the basis vectors can become linearly dependent, resulting in failure to converge. This happens early on for the monomial basis, as is expected. For $s = 10$ (a theoretical $10\times$ speedup), all bases follow the same iterates as the standard implementation. Behavior for other values of s "interpolates" the data shown, i.e., all bases will follow the same iterates as the standard implementation for $2 \leq s \leq 10$. The Newton and Chebyshev bases (or change of basis matrices) eventually become ill-conditioned as well, but are able to reproduce the standard iterates using a larger Krylov subspace than the monomial basis. From Figure 2, it is clear that, like the standard KSMs, convergence of our CA-KSMs is (somewhat) dependent on the condition number of A. It is also important to notice, in Figure 1, the stagnation that occurs in all three bases for $s = 20$. This behavior is due to build up of round-off errors in the computed residual, which causes significantly deviation from the true residual. This behavior worsens (stagnation occurs sooner) with increasing s. We will discuss remedies for this in Section 4.2.3.

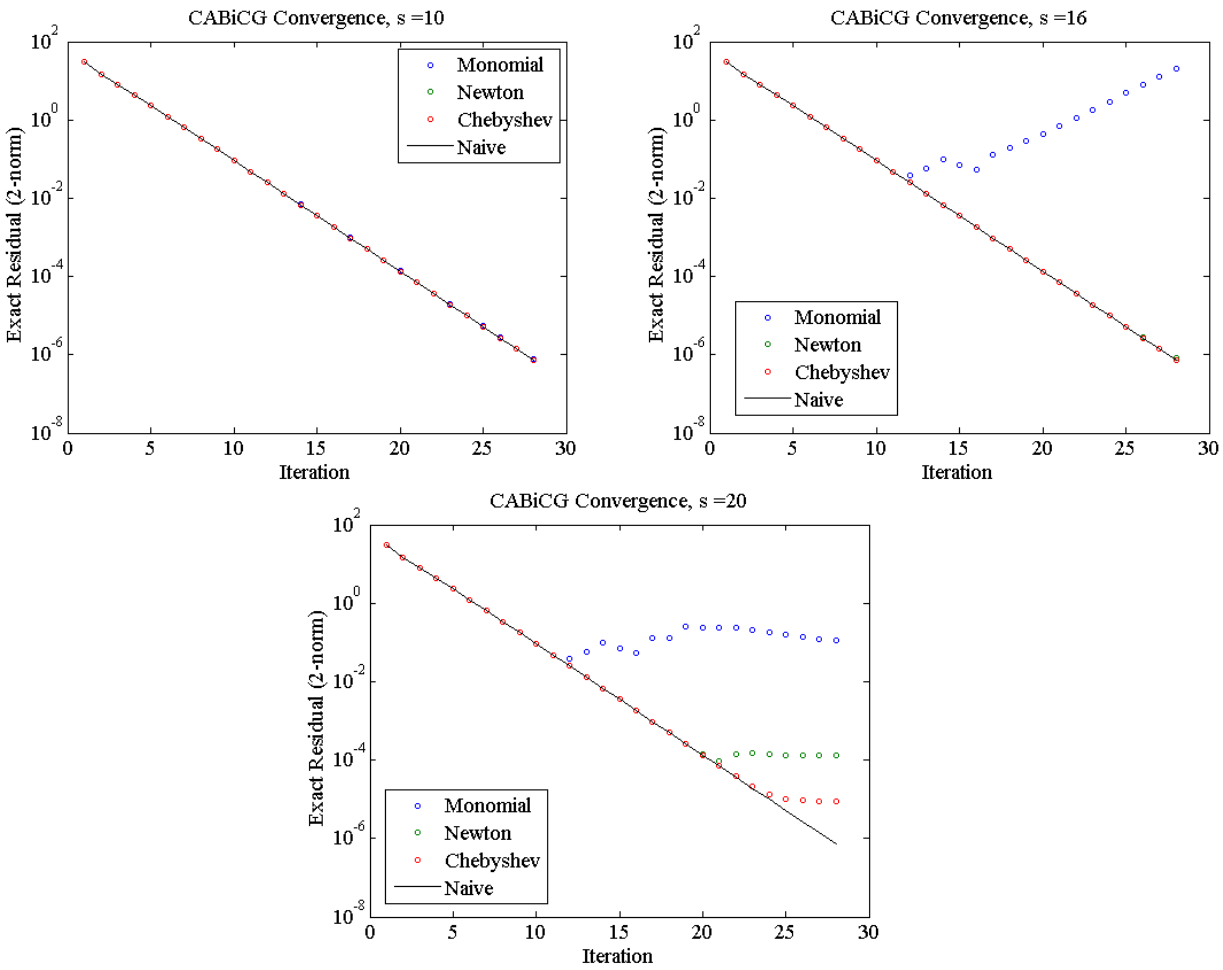

Figure 1: Convergence of Diagonal Matrices for Various Values of s. $N = 1000$, cond~10.

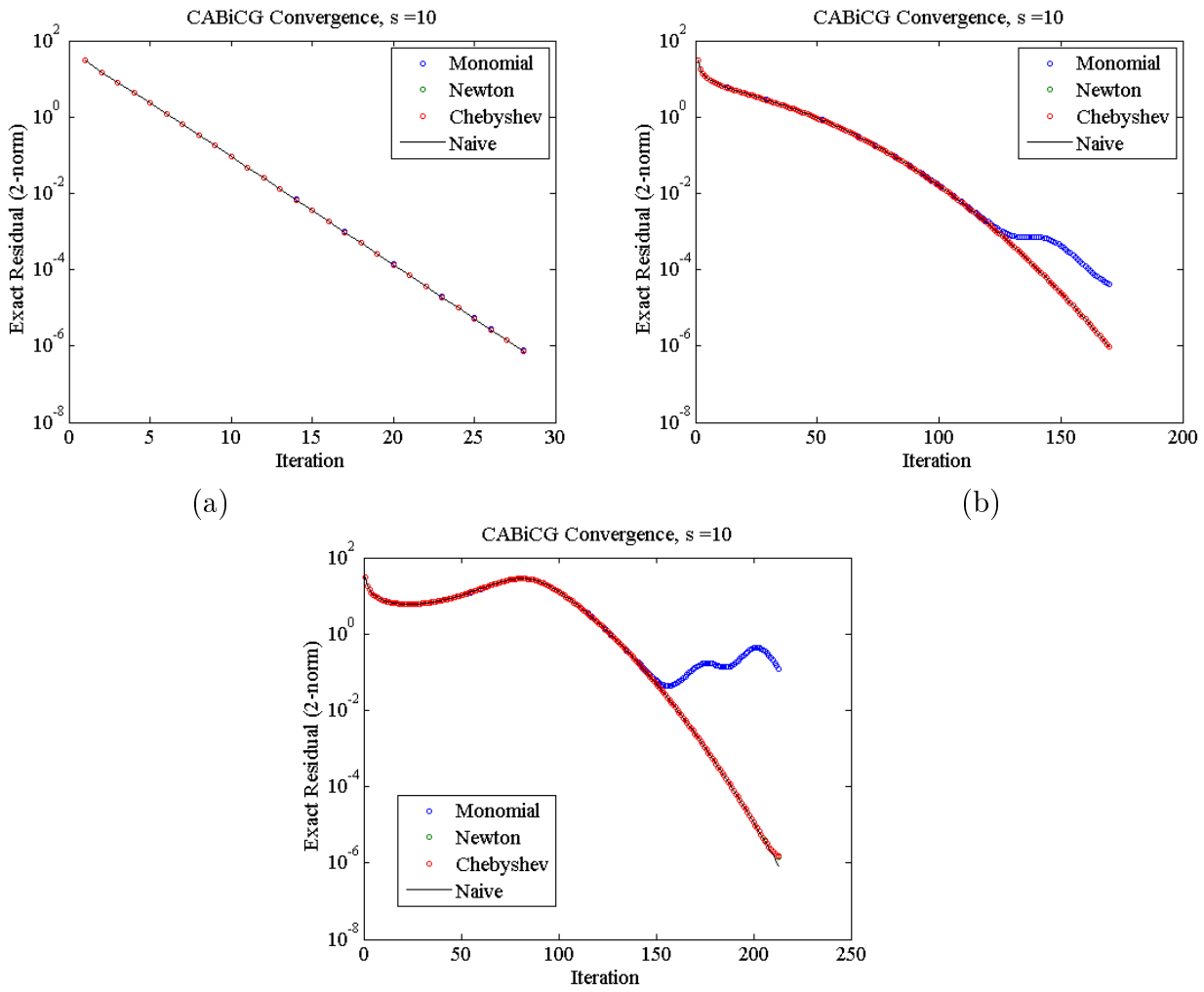

Figure 2: Convergence of Diagonal Matrices for Various Condition Numbers, (a) 10, (b) 10^3, (c) 10^5. $N = 1000$. Eigenvalues are uniformly distributed.

4.2.2 Results for Test Matrices from Various Applications

We have performed convergence tests for both CA-BICG and CA-BICGSTAB for the a small set of test matrices from the University of Florida Sparse Matrix Collection [11]. In these tests, the matrix equilibration routine was used (when necessary). As above, the starting vector ($r_0 = b$) used was a vector of all one's, and the initial guess $x_0 = 0$.

mesh2e1 The matrix mesh2e1 is a discretized structural problem from NASA. The dimension of the matrix is 306, with 2,018 nonzeros. This matrix is symmetric positive definite (SPD), and is well-conditioned (condition number ~ 400). Although BICG and BICGSTAB work for nonsymmetric matrices, we begin with this example as the convergence behavior is smoother and more predictable for SPD matrices.

Figure 3 shows the eigenvalues of this matrix (blue x's) along with the first 30 Leja points (black o's). The plot on the right shows the condition number of the monomial basis,

Newton basis (computed using the shown Leja points), and the Chebyshev basis (where the bounding ellipse was calculated using the exact eigenvalues), versus the size of the basis.

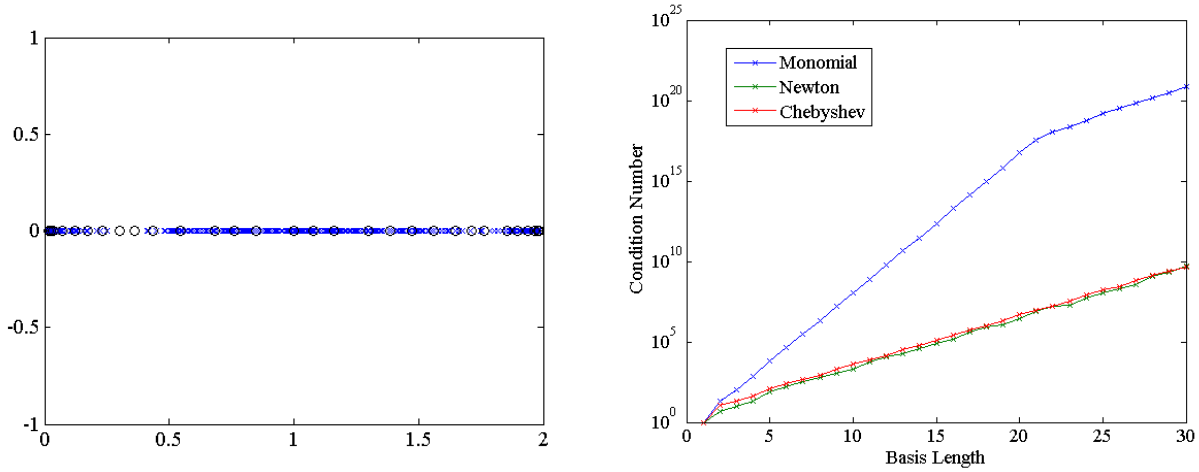

Figure 3: Spectrum and Leja Points for mesh2e1 (left). Basis Condition Number vs. Basis Length (right).

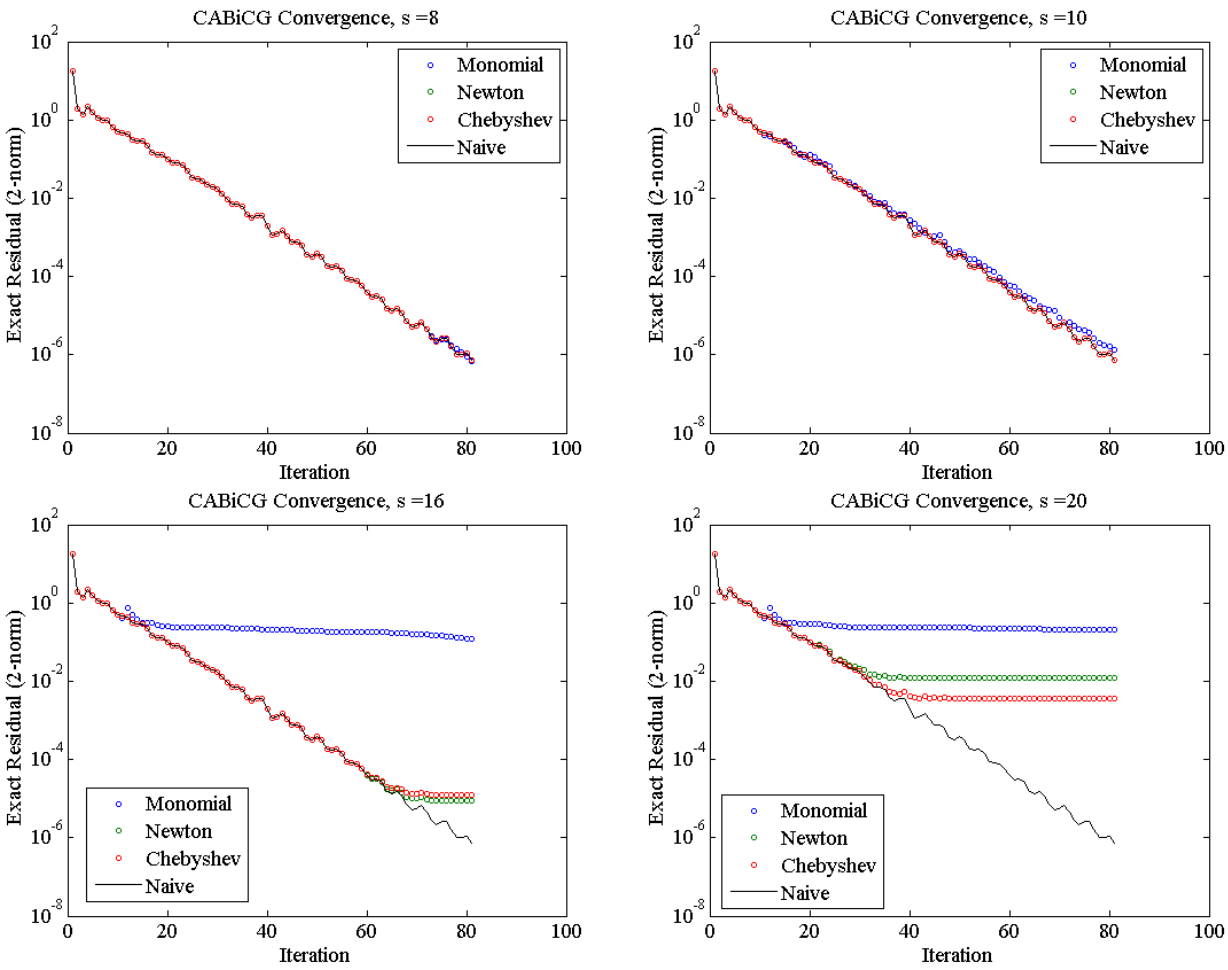

Figure 4: Convergence of CA-BICG for mesh2e1.

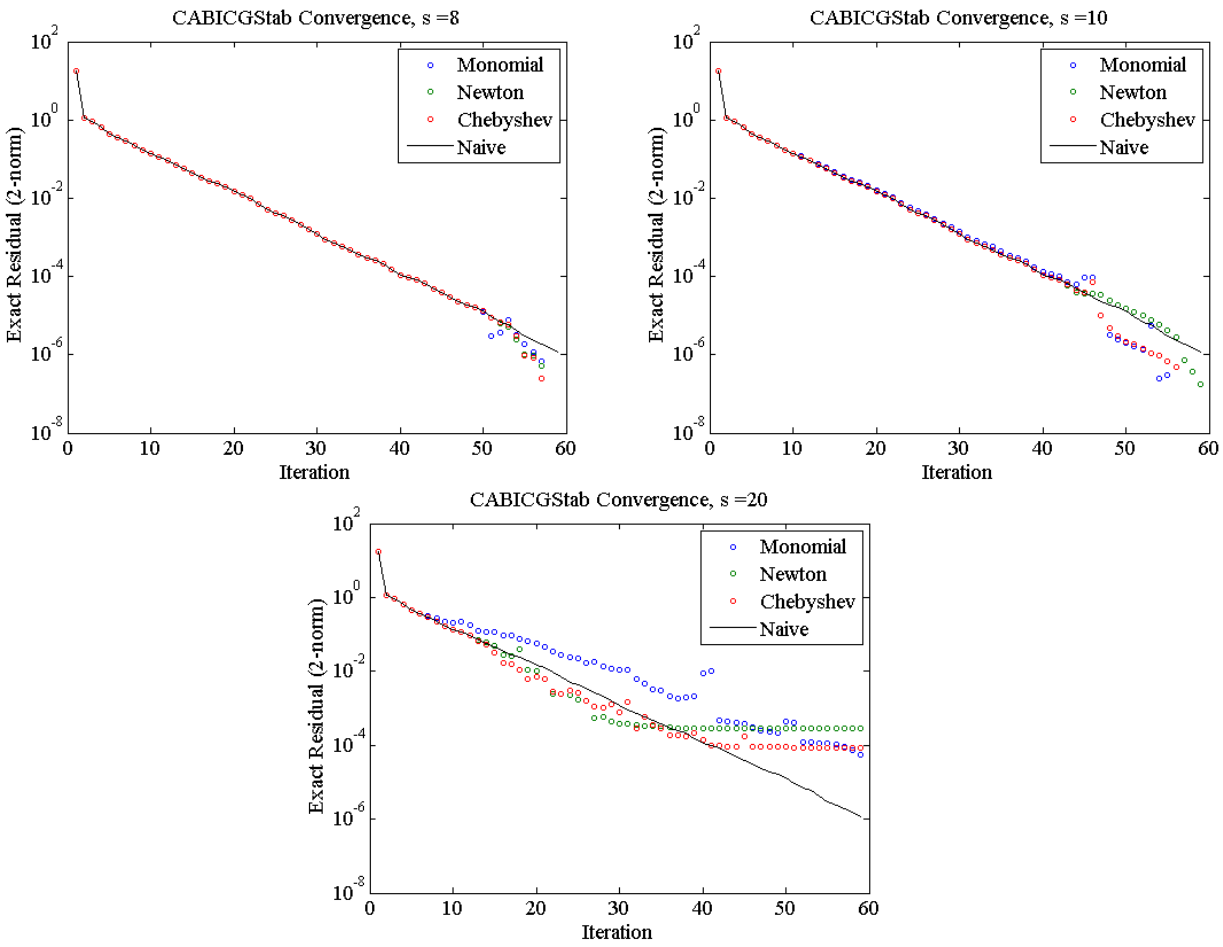

Figure 5: Convergence of CA-BICGSTAB for mesh2e1.

pde900 This matrix is a 5-pt central difference discretization of a 2D variable-coefficient linear elliptic equation on the unit square with Dirichlet boundary conditions, with $NX = NY = 30$. It is well-conditioned, with condition number ~ 300. Here, $N = 900$, and there are $4,380$ nonzeros. This matrix is structurally symmetric, with 50% numeric value symmetry. This matrix is not positive definite. Figure 6 below shows the eigenvalues of this matrix, selected Leja points, and resulting condition number of the basis for various bases and basis lengths. Figures 7 and 8 show convergence results for CA-BICG and CA-BICGSTAB for the different bases, for increasing values of s.

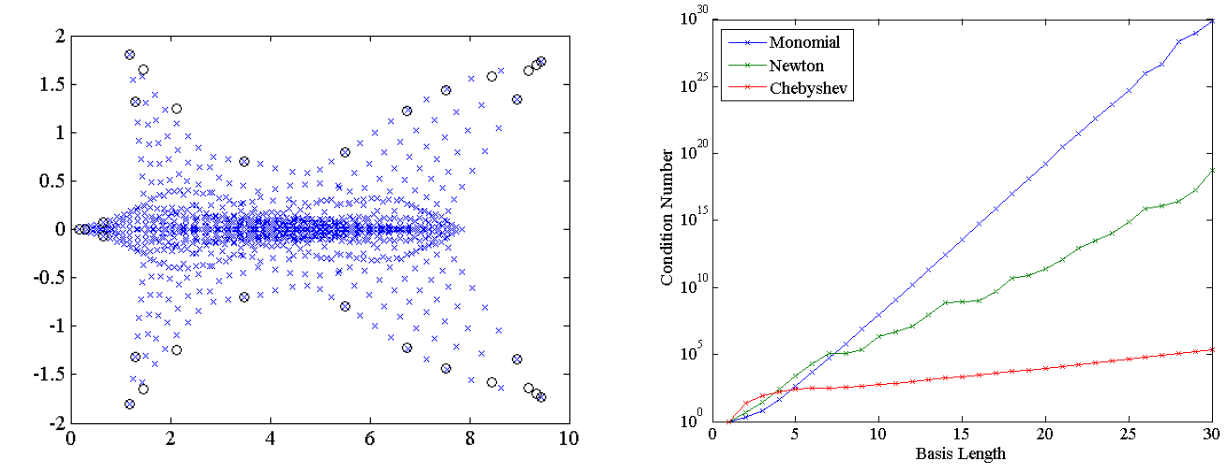

Figure 6: Spectrum and Leja Points for pde900 (left). Basis Condition Number vs. Basis Length (right).

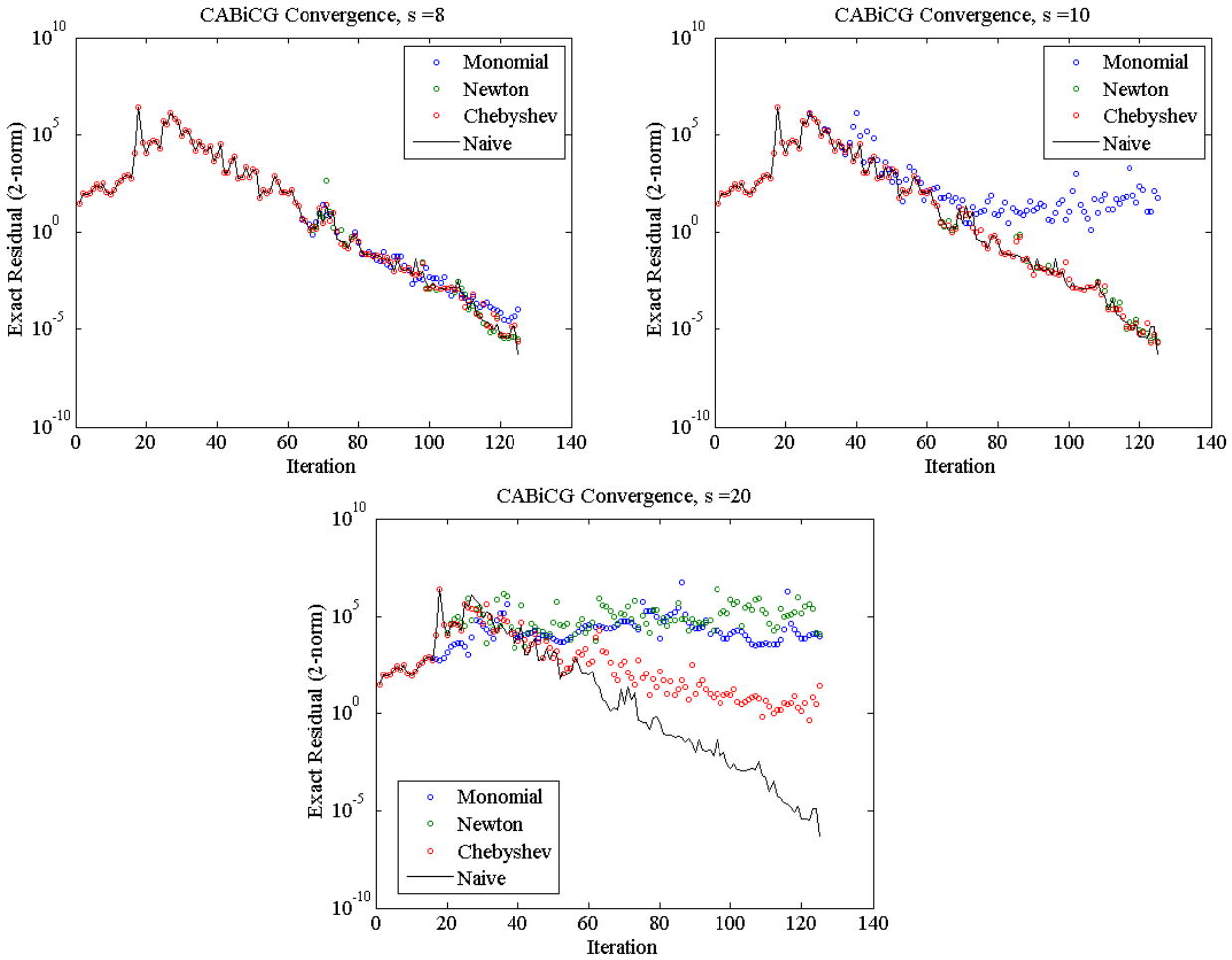

Figure 7: Convergence of CA-BICG for pde900.

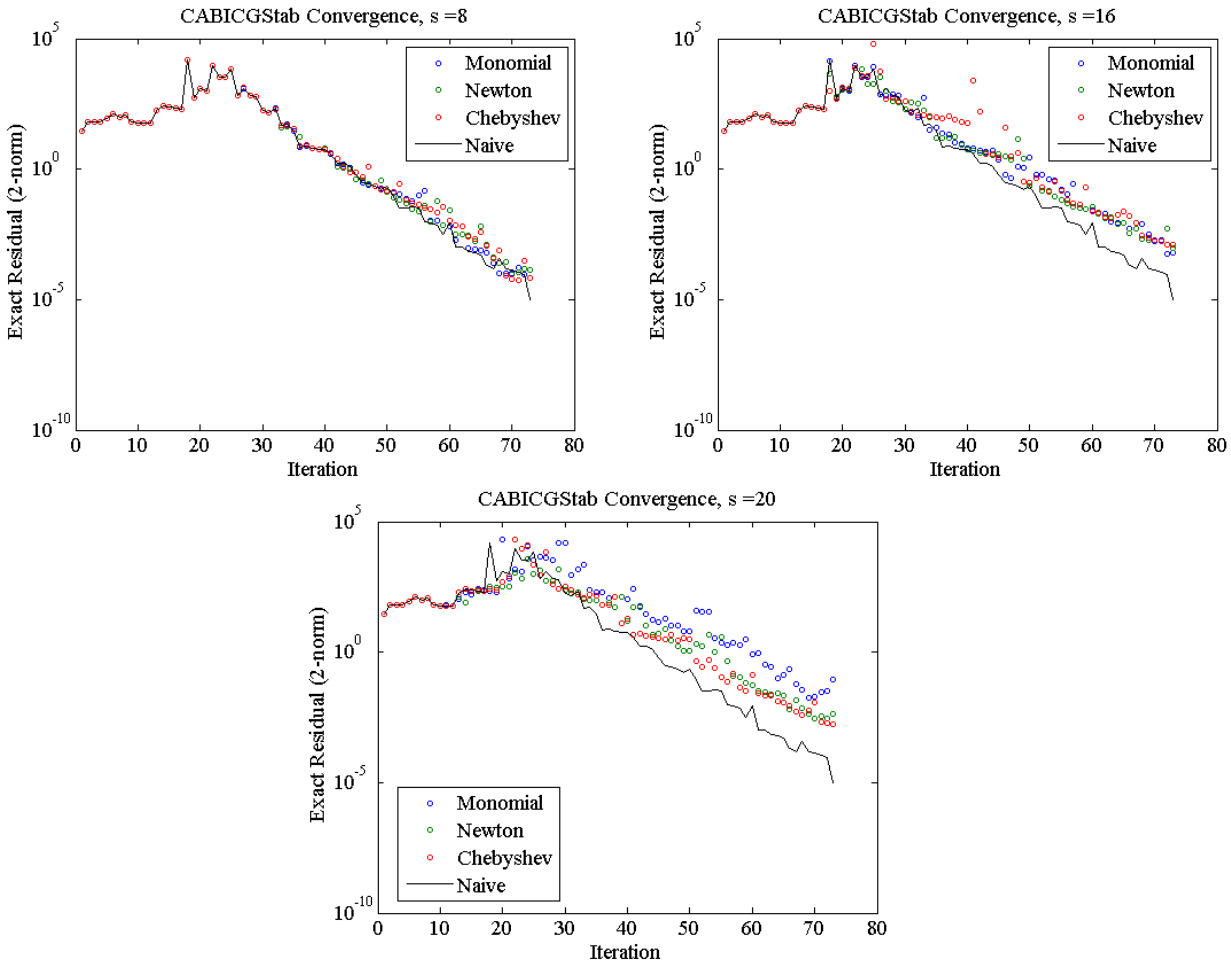

Figure 8: Convergence of CA-BICGSTAB for pde900.

fs_680_1 This matrix is an unsymmetric facsimile convergence matrix. It is slightly less well-conditioned, with condition number $\sim 10^4$. After the matrix equilibration routine, the condition number was lowered to $\sim 10^3$. This matrix is not positive definite, and is not symmetric in structure or in value. Here, $N = 680$, and there are $2,184$ nonzeros. Figure 9 below shows the eigenvalues of this matrix, selected Leja points, and resulting condition number of the basis for various bases and basis lengths. Figures 10 and 11 show convergence results for CA-BICG and CA-BICGSTAB for the different bases, for increasing values of s.

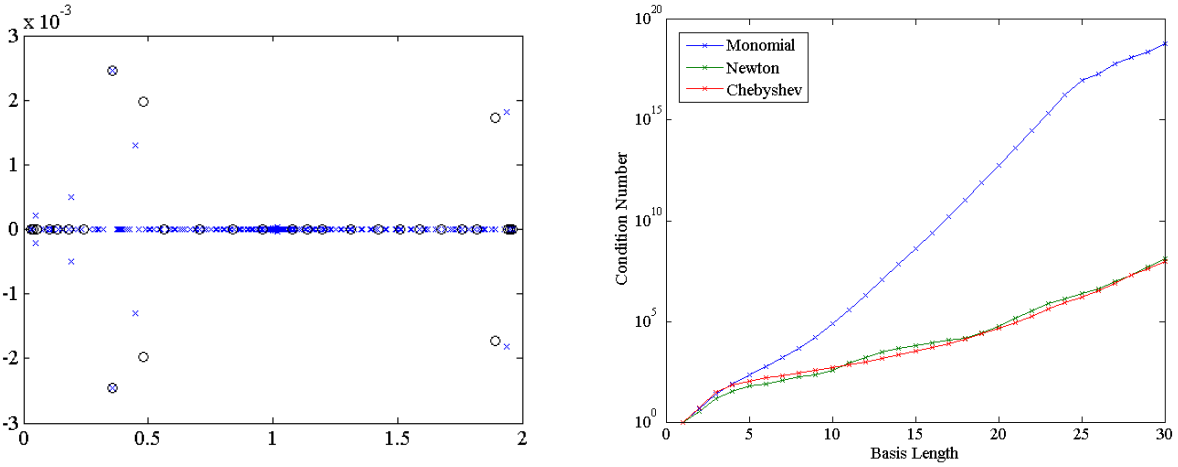

Figure 9: Spectrum and Leja Points for fs_680_1 (left). Basis Condition Number vs. Basis Length (right).

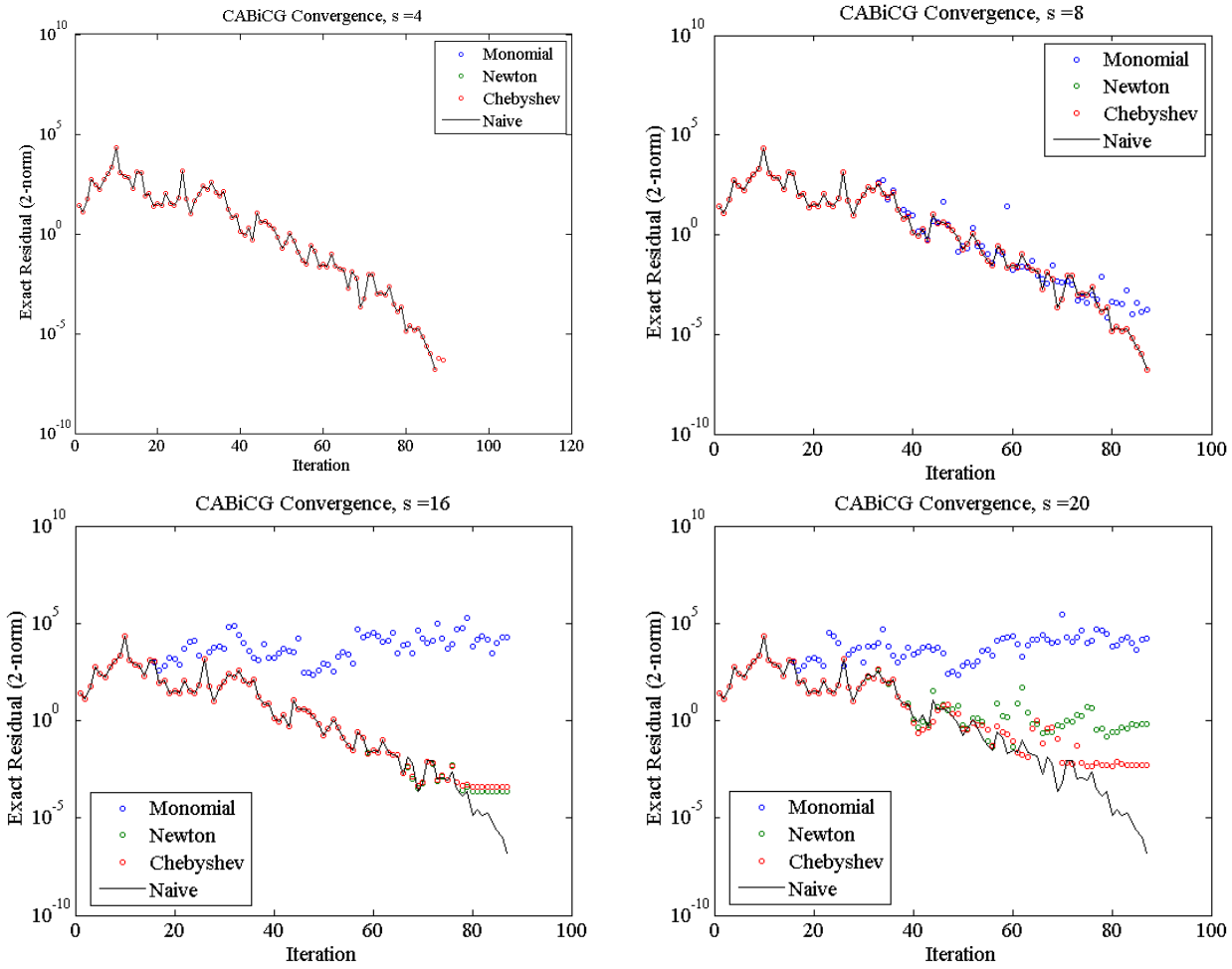

Figure 10: Convergence of CA-BICG for fs_680_1.

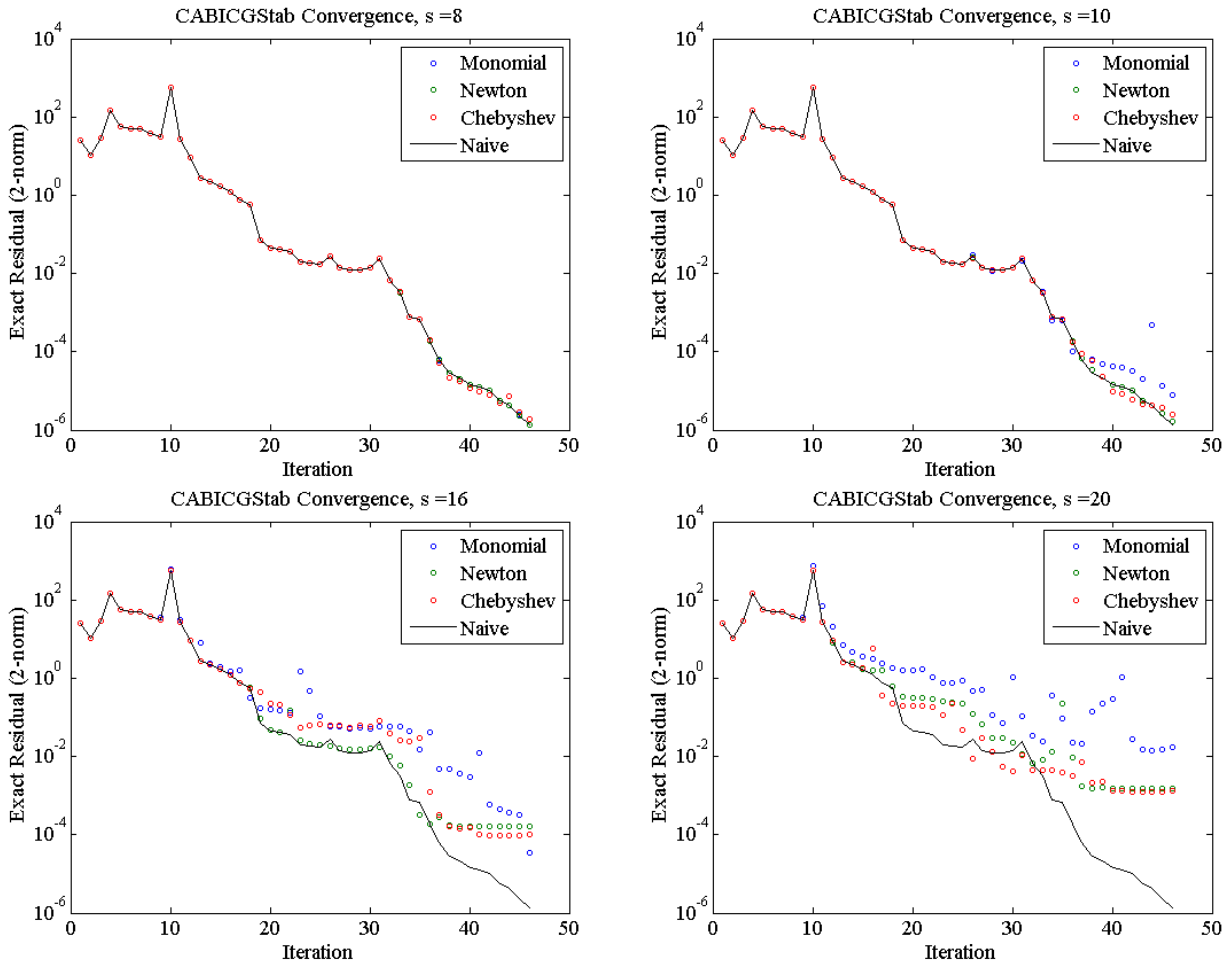

Figure 11: Convergence of CA-BICGSTAB for fs_680_1.

4.2.3 Effect on Maximum Attainable Accuracy

In the previous section, we tested for convergence of the true residual to a precision of 10^{-6} (i.e., $||r_m||_2 \leq 10^{-6}$). In practice, however, higher precision may be necessary. We see from the convergence plots in the previous section that, for high s values, for both CA-BICG and CA-BICGSTAB, our CA method is unable to converge to the same tolerance as the standard implementations because the convergence stagnates. This effect is due to floating point round off error, which causes a discrepancy between the true and computed residuals (similar plots of the computed residual would appear to continue convergence). This problem has been observed before, and is known to especially plague 2-sided KSMs [17]. We observe here that this problem is also present in the CA variants of these methods, and is in fact exacerbated the larger the s value chosen (this behavior is expected to worsen in the 3-term variants of the method [17]). As restarting has been used effectively to combat this problem in the BICG method [34], we extend this technique to our CA-KSMs. In the remainder of this section, we describe the problem of round off in our CA-KSMs, and present results which indicate that restarting prevents buildup of round off error and allows for convergence to a higher precision.

This decrease in maximum attainable accuracy is due to using a basis matrix (or change of basis matrix with a high condition number), which is (more than) squared in the construction of the Gram-like matrix. The presence of very large (and very small) values in the Gram-like matrix leads to floating point round off errors when the recurrence coefficients are used to recover the monomial basis. In the plots for unrestarted CA-BICGSTAB, we see that the monomial basis can sometimes achieve a higher maximum attainable accuracy than the Chebyshev or Newton bases (although it deviates from the standard algorithm iterates sooner in the execution). This is due to increased round off error in the construction of and multiplication by B_s, the change of basis matrix, in both Chebyshev and Newton bases.

If the spectrum of A is large, consecutive shifts used in the Newton basis may differ by orders of magnitude, and ellipse parameters chosen for the Chebyshev basis may be large, leading to an ill-conditioned B_s. By choosing parameters such that the condition number of the basis matrix is as small as possible, we have also chosen parameters which will make the condition number of the change of basis matrix high. We expect results for the Newton basis may be somewhat improved by using Reichel's capacity estimate in computing the (modified) Leja ordering [30]. Further experiments are required.

For the monomial basis, however, $B_s = I_{s+1}$, so the change of basis matrix is always well-conditioned irrespective of the spectrum of A. Here, we have created a change of basis matrix with very low condition number, which results in a basis matrix with a high condition number. This effect is more prevalent in CA-BICGSTAB than in CA-BICG due to the use of three (rather than two) Krylov bases in the construction of the Gram-like matrix, which leads to a significantly higher condition number. We expect that using the variant of CA-BICGSTAB which only requires two RHSs for the matrix powers kernel would lead to behavior closer to that of CA-BICG.

Restarting has been used as a way to combat this problem in the BICG method, with good results [34]. In a similar attempt, we employ restarting of our CA-KSMs. Our results are shown in Figures 12 and 13, which shows both the unrestarted and restarted methods. To restart, we take the current x value and restart the algorithm using this x as the initial guess, which prevents the buildup of floating point errors. Many techniques exist for dynamically choosing when the method should be restarted, e.g., [34], by computing the true residual $b - Ax$ after some number of iterations and monitoring the discrepancy. In our experiments, we manually experimented to find a good restart value for each test case. Implementing a dynamically selected restart length (in a communication-avoiding way) is future work. It should be noted that when the algorithm is restarted, there is an expected plateau before convergence is seen again, as the previous information we had is lost. This can be combated by using deflation techniques which require knowledge of eigenvectors for the smallest eigenvalues. This is, again, considered future work.

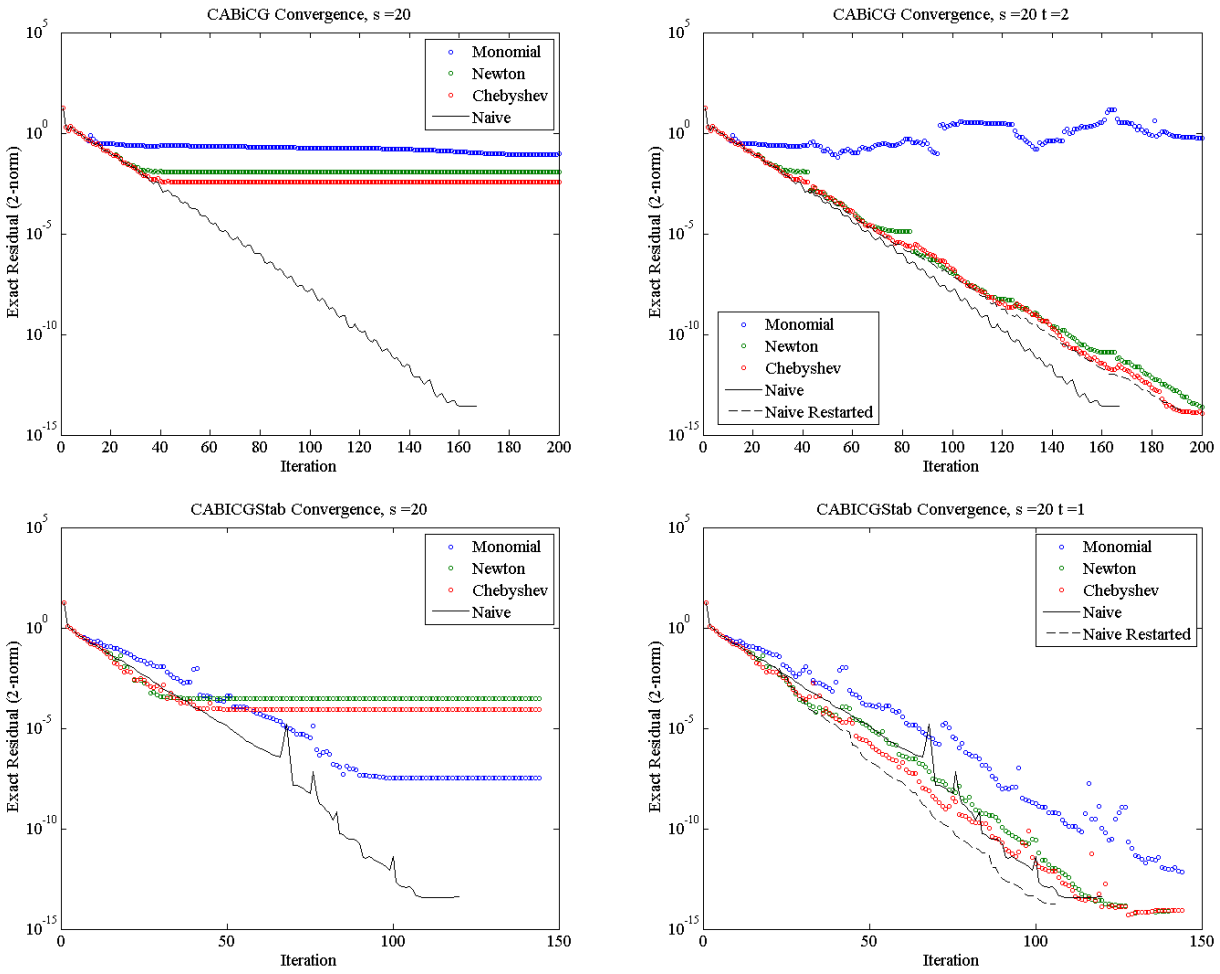

Figure 12: CA-BICG (top) and CA-BICGSTAB Convergence (bottom) for mesh2e1 Matrix, for $s = 20$. $s \times t$ is the chosen restart length. Plots on left show convergence without restarting, plots on right show convergence with restarting technique.

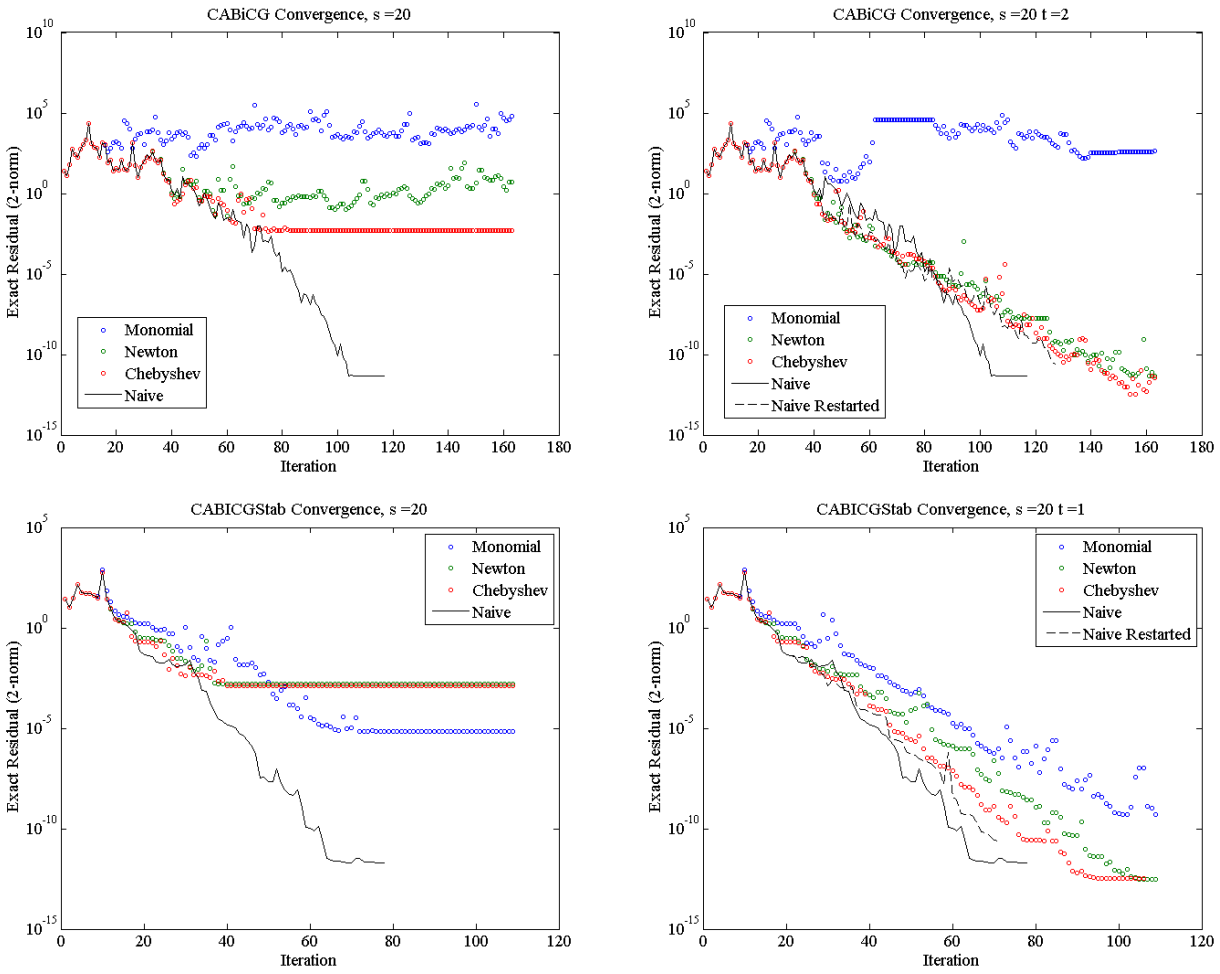

Figure 13: CA-BICG (top) and CA-BICGSTAB Convergence (bottom) for fs_680_1 Matrix, for $s = 20$. $s \times t$ is the chosen restart length. Plots on left show convergence without restarting, plots on right show convergence with restarting technique.

4.2.4 Further Techniques for Improving Convergence: TSQR

Another technique for improving convergence involves the use of TSQR [12]. This kernel has two uses in our algorithms. First, we could use the output of this kernel to construct an orthogonal basis for the Krylov Subspace. To perform this orthogonalization technique, we perform a TSQR operation on the basis vectors after they are generated by the matrix powers kernel, once per outer-loop iteration. As this is a communication-avoiding kernel, we do not asymptotically increase the communication cost, although the constants grow larger, and the computation cost increases. We then construct the Q basis matrix from the output of the kernel, and instead use this orthogonalized basis in the s inner loop iterations. We can easily update the change of basis matrix by pre-multiplication by the R factor.

It should be noted that this approach does not reintroduce dependencies between inner and outer loops in the Krylov Subspace Method, as occurs when computing scaled bases, mentioned in Section 2. In work describing computing a scaled basis, each basis vector

is orthogonalized against all previous basis vectors as they are computed. In using the TSQR kernel for orthogonalization, we first compute all basis vectors, and then perform the orthogonalization. Although this approach can result in more round-off error, it allows us to preserve our communication-avoiding strategy.

Our initial results have indicated that orthogonalization helps convergence only minimally in the case of CA-BICGSTAB, and also only minimally for the Newton and Chebyshev bases in the CA-BICG method. Orthogonalization also does not prevent the stagnation due to round off error. The restarting technique described in the previous section is still necessary. Restarting alone, however, does not allow for convergence when using the monomial basis in the CA-BICG method. In this case, both orthogonalization and restarting are needed, and when both these techniques are used, the monomial basis (which can really no longer be called the "monomial basis") behaves just as well as the Newton and Chebyshev bases. Because the orthogonalization through the TSQR operation requires another communication in each outer loop, we recommend that this technique only be used in the CA-BICG algorithm with the monomial basis. The monomial basis might be an attractive choice if the user does not have good eigenvalue estimates to compute Leja points or a bounding ellipse for the spectrum of A, as are needed for the Newton and Chebyshev bases, respectively. If using CA-BICGSTAB, or CA-BICG with Newton and Chebyshev, our results indicate that restarting the algorithm is sufficient for regaining convergence.

Another potential use of this kernel is dynamically changing the number of inner loop iterations, s, based on the condition number of the R matrix. This prevents the method from attempting to perform s iterations of the Krylov method with a rank-deficient basis matrix, which will occur if the basis length is chosen to be too long (especially for the monomial basis). If the user does not supply an s value, or is unsure of the best s to use for convergence, this added TSQR operation can allow us to dynamically change the s value during each outer loop iteration to ensure convergence. After the TSQR operation, the R factor is distributed to each processor (R is small, $O(s) \times O(s)$). Each processor can then, without communication, compute how many steps of the algorithm it is safe to take by incrementally finding the rank of R (or incrementally estimating the condition number) - when the rank of the first r columns is less than r, we stop and set $s = r - 1$. Therefore we can be sure that our basis vectors are not linearly dependent, and convergence will still be possible.

5 Implementation Details

5.1 Matrix Powers Kernel Variants for Multiple RHSs and A^H

In the CA-CG and CA-GMRES methods studied by [20, 27], only one matrix powers kernel operation is needed with the matrix A. In contrast, our CA-BICG method requires 4 matrix powers kernel operations - two vectors with A and two vectors with A^H (only one RHS is needed for the 3-term recurrence variant), and CA-BICGSTAB requires 3 matrix powers kernel operations - three different vectors with A. In practice, although the storage costs increase by a constant, the communication cost does not increase. We can perform the matrix powers kernel operation with multiple RHSs still reading the matrix A only once.

Here, we perform s Sparse Matrix-Matrix Multiplications (SpMMs) instead of s SpMVs. Vuduc [40] has implemented an optimized version of the SpMM kernel which only requires reading A once to operate on all RHSs. Such an implementation of the matrix powers kernel (one that instead computes $\{B, AB, A^2B, ..., A^sB\}$) would also allow for development of Communication-Avoiding Block Krylov Methods in which many systems are solved simultaneously with the same matrix.

We can also perform the matrix powers kernel operations on both A and A^T reading the matrix A only once (simply by switching the direction of the arrows in the layered graph of dependencies). Because of these further reductions in required communication vs. the standard implementations, we expect to see a greater performance gain for KSMs which require multiple Krylov subspaces using a specialized matrix powers kernel.

Because the required functionality of the matrix powers kernel varies depending on which CA-KSM is used, auto-tuning and code generation is useful. There are currently two active projects dealing with this problem. One approach involves incorporation of the matrix powers kernel into pOSKI, an auto-tuned library for sparse matrix computations [22]. Another involves Selective Embedded Just-In-Time Specialization (SEJITS), in which different code variants are selected dynamically during execution based on the run-time parameters [7].

5.2 Finding Eigenvalue Estimates

In our numerical experiments, we assume full knowledge of the spectrum (as this shows the best we can hope to do using these bases), but in practice the user might not have any information about the eigenvalues of A. In this case, if a Newton or Chebyshev basis is to be used in the computation, we must first find eigenvalue estimates for A.

This is commonly accomplished by taking s (or $O(s)$) steps of Arnoldi to get the upper Hessenberg matrix H. The eigenvalues of H, called Ritz values, are often good estimates for the eigenvalues of A.

Using the Ritz values, we can then either computed the (modified) Leja points for use in the Newton basis or find good bounding ellipse parameters for use in the Chebyshev basis [29]. It should be noted that, to use Joubert and Carey's formula for the Chebyshev basis, we really only need the maximum and minimum real eigenvalues and largest imaginary eigenvalue (which define the ellipse). Here, instead of computing s eigenvalue estimates, we might be able to make due with fewer Arnoldi iterations designed to find these quantities.

While we don't need to be completely accurate in these estimates, problems can occur in extreme cases, depending on the spectrum of A. If Arnoldi gives you the s largest eigenvalues and these are close together, the Newton basis will be ill-conditioned, as the basis vectors will be nearly linearly dependent. Additionally, an ill-conditioned change of basis matrix will result if the chosen shifts vary by many orders of magnitude (here the ill-conditioning of the monomial basis has just been transferred to the change of basis matrix). It is also known that the Chebyshev basis does poorly if the eigenvalues are clustered inside the ellipse.

5.3 Choice of Basis in Practice

Based on observations about the convergence of the CA-KSMs studied in this paper, we offer some preliminary guidance in choosing a basis in practical execution of these methods. In

practice, if $s > 5$ is required, either Newton or Chebyshev basis should be used if the user knows (some of) the eigenvalues of A or can afford to compute estimates, as the monomial basis rarely convergences in this case (unless the matrix is extremely well conditioned). Even though some runtime may be lost computing eigenvalue estimates and performing more floating point operations in recovering the monomial basis, the ability to choose a larger value of s (assuming A^s is still well-partitioned) may still result in a net performance gain.

Another option for higher s values is to use the monomial basis with orthogonalization (by performing a TSQR operation on the basis vectors, described in section 4.2.4), although this requires an additional communication in the outer loop.

If we can only use a small value of s (based on the structure and density of A^s) or if we have a very well-conditioned matrix, the monomial basis will usually perform on par with the Chebyshev and Newton bases. In this case, the monomial basis is the obvious choice, since fewer floating point operations are required in the CA-KSM.

In future implementations of these CA-KSMs, the user will not be burdened with making these decisions. Ideally, an auto-tuner could be used to select an appropriate code variant that achieves both performance and stability based on matrix structure, eigenvalue estimates, and other properties of A. The study and development of heuristics to accomplish such a task is future work.

6 Future Work and Conclusions

6.1 Conclusions and Comments on Convergence

In this work, we have developed three new CA-KSMs: CA-BICG, CA-CGS, and CA-BICGSTAB. We have implemented the monomial, Newton and Chebyshev bases and studied convergence properties for a variety of matrices. We are able to use basis length $2 \leq s \leq 20$, depending on the matrix properties, where using a basis length of s allows for up to an $s\times$ speedup.

Convergence results indicate that, like standard implementations, our methods suffer from round off error. The higher the s value used, the more prominent this effect, which decreases maximum attainable accuracy. To combat this behavior, we have explored restarting in CA-KSMs as a method to reduce stagnation due to deviation of true and computed residuals, with promising results. We have also discussed the use of using the TSQR kernel to orthogonalize the basis and dynamically select s, although whether or not this is practical is yet to be determined.

Based on our initial observations, we have provided details and advice for the practical implementation of CA-KSMs, including necessary preprocessing steps such as selecting a polynomial basis for stability purposes. Additionally, we have shown how to further reduce communication by using a variant of the `Akx` kernel that does s SpMMs instead of s SpMVs if we have a method that requires more than one Krylov Subspace. This kernel can also be used to do CA-Block KSM versions of our methods, to solve multiple systems simultaneously. Decisions such as these will eventually be handled by a combination of auto-tuning and code generation.

6.2 Future Work

Although initial convergence results are promising, there is much remaining work involved in understanding, analyzing and improving convergence, as well as in creating efficient implementations of these methods, benchmarking performance, and developing auto-tuner heuristics. Making CA-KSMs practical, efficient, and available for use will require much time and effort. The three areas of future work listed below are considered to be priorities in order to meet this goal.

Performance Experiments Our next step will be to code parallel, optimized implementations (both sequential and parallel) of our CA-KSMs and required kernels. We will then benchmark performance for a variety of matrices, on a variety of platforms. Although we have shown that communication is theoretically asymptotically reduced in our algorithms, we must also see how our algorithms behave in practice (how much redundant work is required, where the optimal s value is depending on matrix surface-to-volume ratio, etc). Such an analysis will determine the most important areas to work on moving forward.

Auto-tuning and Code Generation for Matrix Powers Many complex optimizations and implementation decisions are required for both matrix powers kernel performance and convergence of CA-KSMs. These optimizations are both machine and matrix dependent, which makes auto-tuning and code generation an attractive approach. Employing such techniques will lift the burden of expertise from the user, making the use of CA-KSMs more accessible. Many approaches and ongoing projects exist for this purpose [7, 22, 24]. We also plan to apply our methods to applications (e.g., as a bottom-solver in multigrid methods), in conjunction with collaborators at the Communication Avoidance and Communication Hiding at the Extreme Scale (CACHE) Institute, an interdisciplinary project funded by the U.S. Department of Energy. We hope that by further study of the convergence and performance properties of our CA-KSMs, we can contribute to the design of auto-tuners and heuristics, and the successful integration of our work into existing frameworks.

Techniques to Further Improve Stability In this work, we presented a qualitative analysis of initial convergence results for a small set of test matrices. A more rigorous analysis of convergence properties for CA-KSMs will be necessary, and helpful in determining new methods for improving stability. Current ideas for improving stability include using techniques such as dispersive Lanczos [36] for finding better eigenvalue estimates, using extended precision in (perhaps only part of) the CA-KSM iterations, and further exploring the use of variable basis length, incremental condition estimation, and orthogonalization techniques.

Acknowledgements

This paper was researched with Government support under and awarded by the Department of Defense, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. This research was also supported by

Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Research supported by U.S. Department of Energy grants under Grant Numbers DE-SC0003959 and DE-AC02-05-CH11231, as well as Lawrence Berkeley National Laboratory. Research supported by NSF SDCI under Grant Number OCI-1032639. The authors would also like to thank Laura Grigori (INRIA, France) and Mark Hoemmen (Sandia National Labs) for their insightful comments.

References

[1] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14(4):563, 1994.

[2] Dennis J.M. Baker, A.H. and E.R. Jessup. On improving linear solver performance: A block variant of GMRES. *SIAM Journal on Scientific Computing*, 27(5):1608–1626, 2006.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIMAX, see also bebop.cs.berkeley.edu*, 2010.

[4] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J.M. Donato, J. Dongarra, V. Eijkhout, R.P.C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. *SIAM, Philadelphia*, 1994.

[5] G.M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM (JACM)*, 25(2):226–244, 1978.

[6] R. Bru, L. Elsner, and M. Neumann. Models of parallel chaotic iteration methods. *Linear Algebra and its Applications*, 103:175–192, 1988.

[7] B. Catanzaro, S. Kamil, Y. Lee, K. Asanovic, J. Demmel, K. Keutzer, J. Shalf, K. Yelick, and A. Fox. SEJITS: Getting productivity and performance with Selective Embedded JIT Specialization. In *First Workshop on Programmable Models for Emerging Architecture at the 18th International Conference on Parallel Architectures and Compilation Techniques*. Citeseer, 2009.

[8] D. Chazan and W. Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2(2):199–222, 1969.

[9] A.T. Chronopoulos and C.W. Gear. S-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics*, 25(2):153–168, 1989.

[10] J. Cullum and W.E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices. In *Decision and Control including the 13th Symposium on Adaptive Processes, 1974 IEEE Conference on*, volume 13, pages 505–509. IEEE, 1974.

[11] T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, 97(23):7, 1997.

[12] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical report, Technical Report UCB/EECS-2007-123, University of California Berkeley EECS, 2007.

[13] W. Deren. On the convergence of the parallel multisplitting AOR algorithm. *Linear algebra and its applications*, 154:473–486, 1991.

[14] R. Fletcher. Conjugate gradient methods for indefinite systems. *Numerical Analysis*, pages 73–89, 1976.

[15] G.H. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. *Mathematical software*, 3:361–377, 1977.

[16] M.H. Gutknecht. Lanczos-type solvers for nonsymmetric linear systems of equations. *Acta Numerica*, 6(1997):271–398, 1997.

[17] M.H. Gutknecht and Z. Strakos. Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM Journal on Matrix Analysis and Applications*, 22:213, 2000.

[18] V. Hernandez, J.E. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Computing*, 33(7-8):521–540, 2007.

[19] A.C. Hindmarsh and H.F. Walker. Note on a Householder implementation of the GMRES method. Technical report, Lawrence Livermore National Lab., CA (USA); Utah State Univ., Logan (USA). Dept. of Mathematics, 1986.

[20] M. Hoemmen. Communication-avoiding Krylov subspace methods. *Thesis. UC Berkeley, Department of Computer Science*, 2010.

[21] J.W. Hong and H.T. Kung. I/O complexity: the red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pages 326–333, 1981.

[22] A. Jain. pOSKI: An extensible autotuning framework to perform optimized SpMVs on multicore architectures. Technical report, MS Report, EECS Department, University of California, Berkeley, 2008.

[23] W.D. Joubert and G.F. Carey. Parallelizable restarted iterative methods for nonsymmetric linear systems. Part I: Theory. *International Journal of Computer Mathematics*, 44(1):243–267, 1992.

[24] A. LaMielle and M. Strout. Enabling code generation within the Sparse Polyhedral Framework. Technical report, Technical Report CS-10-102 Colorado State University, March 2010.

[25] C.E. Leiserson, S. Rao, and S. Toledo. Efficient out-of-core algorithms for linear relaxation using blocking covers. *Journal of Computer and System Sciences*, 54(2):332–344, 1997.

[26] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT Numerical Mathematics*, 24(4):623–633, 1984.

[27] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 36. ACM, 2009.

[28] D.P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 29:293–322, 1980.

[29] B. Philippe and L. Reichel. On the generation of Krylov subspace bases. *Applied Numerical Mathematics*, 2011.

[30] L. Reichel. Newton interpolation at Leja points. *BIT Numerical Mathematics*, 30(2):332–346, 1990.

[31] J.V. Rosendale. Minimizing inner product data dependencies in conjugate gradient iteration. 1983.

[32] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6(4):865–881, 1985.

[33] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial Mathematics, 2003.

[34] G.L.G. Sleijpen and H.A. van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. *Computing*, 56(2):141–163, 1996.

[35] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *Statist. Comput*, 10:36–52, 1989.

[36] S. Toledo. Dispersive Lanczos or (when) does Lanczos converge? In *Combinatorial Scientific Computing 2011, Conference Proceedings*. SIAM, 2011.

[37] S.A. Toledo. *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms*. PhD thesis, Massachusetts Institute of Technology, 1995.

[38] R.R. Underwood. An iterative block Lanczos method for the solution of large sparse symmetric eigenproblems. 1975.

[39] H.A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13:631, 1992.

[40] R. Vuduc, J.W. Demmel, and K.A. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. In *Journal of Physics: Conference Series*, volume 16, page 521. IOP Publishing, 2005.

[41] H.F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9:152, 1988.

[42] B. Zhongzhi, W. Deren, and D.J. Evans. Models of asynchronous parallel matrix multisplitting relaxed iterations. *Parallel computing*, 21(4):565–582, 1995.