SIAM J. SCI. COMPUT.
Vol. 35, No. 5, pp. S42–S61

AVOIDING COMMUNICATION IN NONSYMMETRIC LANCZOS-BASED KRYLOV SUBSPACE METHODS[*]

ERIN CARSON[†], NICHOLAS KNIGHT[†], AND JAMES DEMMEL[‡]

Abstract. Krylov subspace methods are iterative methods for solving large, sparse linear systems and eigenvalue problems in a variety of scientific domains. On modern computer architectures, communication, or movement of data, takes much longer than the equivalent amount of computation. Classical formulations of Krylov subspace methods require data movement in each iteration, creating a performance bottleneck, and thus increasing runtime. This motivated s-step, or communication-avoiding, Krylov subspace methods, which only perform data movement every $O(s)$ iterations. We present new communication-avoiding Krylov subspace methods, CA-BICG and CA-BICGSTAB. We are the first to provide derivations of these methods. For both sequential and parallel implementations, our methods reduce data movement by a factor of $O(s)$ versus the classical algorithms. We implement various polynomial bases and perform convergence experiments to enable comparison with the classical algorithm. We discuss recent results in improving both numerical behavior and performance in communication-avoiding Krylov subspace methods.

Key words. Krylov subspace methods, iterative methods, minimizing communication, BICG, BICGSTAB, sparse matrix

AMS subject classifications. 65F10, 65F50, 65N22, 65Y05, 65Y20

DOI. 10.1137/120881191

1. Introduction.

The runtime of an algorithm is a function of both *computation*, the number of arithmetic operations performed, and *communication*, the amount of data movement. Communication encapsulates both bandwidth, the amount of data moved between levels of the memory hierarchy and between processors, and latency, the number of messages in which the data are sent. On modern computer architectures, communicating a word of data takes much longer than one floating point operation. This gap is only expected to increase in future systems. Therefore, to increase the performance of algorithms, we must turn to strategies to minimize communication rather than the traditional approach of decreasing 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 $Ax = b$ for an n-by-n matrix A. Iterative methods are commonly used when the coefficient matrix is large and sparse. The most general and flexible class of iterative methods is Krylov subspace methods (KSMs). These methods are based on projection onto expanding subspaces where, in each iteration m, the approximate solution is chosen from the expanding Krylov subspace, $\mathcal{K}_m(A, v) = \text{span}(v, Av, \ldots, A^{m-1}v)$.

Classical 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 on modern machines. To perform an SpMV, each processor must communicate entries of the source vector to other processors in the parallel algorithm, and A and the vectors must be read from slow memory in the

[*]Received by the editors June 15, 2012; accepted for publication (in revised form) January 22, 2013; published electronically October 28, 2013.
http://www.siam.org/journals/sisc/35-5/88119.html

[†]Computer Science Division, University of California Berkeley, Berkeley, CA 94704 (ecc2z@eecs.berkeley.edu, knight@cs.berkeley.edu).

[‡]Mathematics and EECS Department, University of California Berkeley, Berkeley, CA 94720 (demmel@eecs.berkeley.edu).

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). This motivated communication-avoiding KSMs (CA-KSMs), which break the dependency between SpMV and vector operations in each iteration by unrolling the iteration loop s times, allowing an $O(s)$ reduction in communication cost.

Hoemmen et al. have derived algorithms for communication-avoiding generalized minimal residual method (CA-GMRES) and conjugate gradient method (CA-CG) [19, 24]. Additionally, CA-GMRES has been shown to speed up over the classical algorithm on a shared memory platform [24]. In our work, we focus specifically on two-sided (i.e., nonsymmetric Lanczos-based) KSMs, which, implicitly or explicitly, use two Krylov subspaces—one for the "right-hand" space, span$\{v, Av, A^2v, \dots\}$, and one for the "left-hand" space, span$\{w, A^H w, (A^H)^2 w, \dots\}$, where A^H denotes the conjugate transpose of A. Two-sided KSMs are specifically suited for sparse nonsymmetric linear systems, which arise frequently in many scientific domains. Our primary contributions are the following.

We derive new two-sided CA-KSMs: biconjugate gradient (CA-BICG) and biconjugate gradient stabilized (CA-BICGSTAB). These are mathematically, but not numerically, equivalent to the classical methods, in that after every s steps, they produce an identical solution in exact arithmetic. We give algorithms for two-term recurrence versions of each method. We provide convergence results for our methods on test matrices from two scientific domains.

We address numerical stability concerns, and derive the necessary change-of-basis matrices for general three-term recurrence polynomials which can be used to replace the monomial basis in the matrix powers computation with more stable bases. We give examples for both Newton and Chebyshev polynomials. These recurrences produce polynomials which are much better conditioned than the monomials, thus preserving numerical stability 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 convergence (compared with the classical implementation) for basis lengths as high as $s = 16$, which is likely the largest value needed in many practical situations. Note that if the algorithm is communication bound, even basis length $s = 2$ theoretically results in $2\times$ less communication than the classical algorithm, and could thus yield a $2\times$ speedup.

Although here we focus on derivations and numerical stability of these new algorithms, we discuss future work related to implementation choices in two-sided CA-KSMs, which are critical to obtaining good performance. We comment on preconditioning and methods to improve numerical stability in CA-KSMs. We discuss future opportunities for autotuning, which allows for automatic selection of parameters and appropriate code variant to satisfy both performance and stability constraints.

In all subsequent sections of this paper, we use BICG and BICGSTAB to refer to the classical implementations of the KSMs (as described in [30, 33]), and CA-BICG and CA-BICGSTAB to refer to our communication-avoiding variants.

2. Related work.

There is a wealth of literature related to s-step KSMs and the idea of avoiding communication. Space constraints prohibit an in-depth discussion; we direct the reader to the thorough overview given in [19, sections 1.5 and 1.6]. We highlight a few results in the sections below.

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 [35]. Van Rosendale's implementation was motivated by exposing more parallelism. Chronopoulous and Gear later created an s-step GMRES method with the goal of exposing more parallel optimizations [8]. With a similar goal, Kim and Chronopoulous [21] derived an s-step nonsymmetric Lanczos method. Walker used s-step bases as a method for improving stability in GMRES by replacing the modified Gram–Schmidt orthogonalization process with Householder QR [36]. All 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 [18], but only saw minimal improvement. Joubert and Carey implemented a scaled and shifted Chebyshev basis which provided more accurate results [20]. Bai, Hu, and Reichel also saw improved convergence using a Newton basis [3]. Constructing other bases for the Krylov subspace will be covered more thoroughly in section 4.1.

These previously derived methods did not exploit the potential communication avoidance in computing the s Krylov basis vectors. Hoemmen et al. (see, e.g., [13, 19, 24]) were the first to make use of the matrix powers kernel optimization, which reduces communication cost by a factor of $O(s)$ for well-partitioned matrices; our derivations most closely follow their work.

There are many alternative approaches to reducing communication in KSMs which differ from this approach, including reducing synchronizations, allowing asynchronous iterations, using block Krylov methods to exploit locality, and using alternative methods such as Chebyshev iteration. For a good overview, see [19, section 1.6].

2.2. Communication-avoiding kernels. The CA-KSMs introduced by Hoemmen, Mohiyuddin, and others (see [19]), are designed to exploit the matrix powers optimization. This optimization fuses together a sequence of s SpMV operations into one kernel invocation. This kernel is used in the CA-KSM to compute an $(s+1)$-dimensional Krylov basis $\mathcal{K}_{s+1}(A, v)$. Depending on the nonzero structure of A (more precisely, of $\{A^j\}_{j=1}^{s}$), this enables communication avoidance in both serial and parallel implementations, as described below.

Serial. The serial matrix powers kernel reorganizes the s SpMVs to maximize reuse of A and the $s+1$ vectors, ideally reading A and v once, and writing the s output vectors spanning the Krylov subspace only once. When reading A is the dominant communication cost versus reading/writing the vectors (a common situation), this implies an s-fold decrease in both latency and bandwidth.

Parallel. In parallel, the matrix powers kernel reorganizes the computation in a similar way but with a slightly different goal; in a parallel SpMV operation, A is not communicated, only the vectors. The parallel matrix powers optimization avoids interprocessor synchronization by storing some redundant elements of A and v on different processors, and performing redundant computation to compute the s matrix powers without further synchronization. Provided the additional bandwidth and latency cost to distribute v is a lower-order term (equivalently, A^s is *well partitioned*), this gives an s-fold savings in latency.

We note that this discussion applies even if A is not given explicitly. However, if A can be represented in $o(n)$ words, then our serial CA-KSMs no longer asymptotically avoid communication (see section 6.3 for optimizations for this case).

Serial and parallel variants of the matrix powers kernel, for both structured and general sparse matrices, are described in [23], which summarizes most of [13] and elaborates on the implementation in [24]. We refer the reader to the complexity analysis in Tables 2.3–2.4, the performance modeling in section 2.6, and the performance results in section 2.10.3 and section 2.11.3 of [23], which demonstrate that this optimization leads to speedups in practice. For example, for a two-dimenisonal 5-point stencil on a $\sqrt{n} \times \sqrt{n}$ mesh with P processors, assuming $s \ll \sqrt{n/P}$, the number of arithmetic operations grows by a factor $1 + 2s\sqrt{P/n}$, the number of messages decreases by a factor of $s/2$, and the number of words moved grows by a factor [23] of $1 + (s/2)\sqrt{P/n}$. Therefore since the additional arithmetic operations and additional words moved are lower-order terms, we expect to see a $\Theta(s)$ speedup when the problem is latency bound.

Matrix powers kernel performance is extremely sensitive to matrix structure and hardware parameters, thus making it a good candidate for inclusion in autotuning libraries and specializers. Preliminary efforts are promising; in [25], the Python interpreter is modified to selectively apply the matrix powers optimization by choosing an autotuned code variant of the matrix powers kernel based on runtime information.

Besides SpMV operations, KSMs also incur communication costs due to orthogonalization performed in each iteration. The orthogonalization step involves a series of dot products, which incur a costly global synchronization on parallel computers. For Lanczos-based KSMs (like those in this work), the strategy is to block together dot products using a Gram matrix—in parallel, this can lead to an s-fold decrease in latency. We have adapted this approach to two-sided KSMs, where we compute a Gram-like matrix to achieve similar savings.

In the case of Arnoldi-based KSMs (like GMRES), the orthogonalization operations can be blocked by computing a (thin) QR factorization of a tall, skinny matrix. Using the tall-skinny QR algorithm in [12], this leads to an s-fold decrease in latency in the parallel case, as well as an s-fold decrease in latency and bandwidth in the sequential case (see [19]).

3. Derivation of communication-avoiding variants.

We derive our CA-KSMs in two steps. First, we reformulate the classical KSM as an s-step method, breaking the interiteration data dependencies between SpMV and inner product operations within every block of s loop iterations. We then apply communication-avoiding optimizations: we replace $\Theta(s)$ SpMV calls with $\Theta(1)$ calls to the matrix powers kernel, and $\Theta(s)$ inner product calls with $\Theta(1)$ calls to dense matrix-matrix multiply.

We will later see that this enables $\Theta(s)$-fold savings in serial and parallel latency and serial bandwidth. The savings in practice depend on the size and nonzero structure of the linear system A, the machine parameters for each level of the memory/network hierarchy, and the value of s.

3.1. Communication-avoiding BICG.

The first step towards communication-avoiding BICG is to convert classical BICG (Algorithm 3.1) to an s-step method.

The jth Krylov subspace generated by A from v is

$$\mathcal{K}_j(A, v) \coloneqq \operatorname{span}\{v, Av \ldots, A^{j-1}v\}, \tag{3.1}$$

and when $j \le 0$, we set $\mathcal{K}_j \coloneqq \{0\}$. By induction on lines {4, 5, 6, 9, 10} of classical BICG, it can be shown that, given $m \ge 0, j > 0$, the five vector iterates of classical BICG satisfy

$$\begin{aligned} &p_{m+j}, r_{m+j} \in \mathcal{K}_{j+1}(A, p_m) + \mathcal{K}_j(A, r_m), \\ &\tilde{p}_{m+j}, \tilde{r}_{m+j} \in \mathcal{K}_{j+1}(A^H, \tilde{p}_m) + \mathcal{K}_j(A^H, \tilde{r}_m), \\ &x_{m+j} - x_m \in \mathcal{K}_j(A, p_m) + \mathcal{K}_{j-1}(A, r_m). \end{aligned} \tag{3.2}$$

Algorithm 3.1. Classical BICG.

```
Require: Initial approximation $x_0$ for solving $Ax = b$, let $p_0 \coloneqq r_0 \coloneqq b - Ax_0$
 1: Choose $\tilde{r}_0$ arbitrarily s.t. $\delta_0 \coloneqq (\tilde{r}_0, r_0) \neq 0$, and let $\tilde{p}_0 \coloneqq \tilde{r}_0$
 2: for $m \coloneqq 0, 1, \ldots,$ until convergence do
 3:     $\alpha_m \coloneqq \delta_m / (\tilde{p}_m, Ap_m)$
 4:     $x_{m+1} \coloneqq x_m + \alpha_m p_m$
 5:     $r_{m+1} \coloneqq r_m - \alpha_m A p_m$
 6:     $\tilde{r}_{m+1} \coloneqq \tilde{r}_m - \overline{\alpha_m} A^H \tilde{p}_m$
 7:     $\delta_{m+1} \coloneqq (\tilde{r}_{m+1}, r_{m+1})$
 8:     $\beta_m \coloneqq \delta_{m+1} / \delta_m$
 9:     $p_{m+1} \coloneqq r_{m+1} + \beta_m p_m$
10:     $\tilde{p}_{m+1} \coloneqq \tilde{r}_{m+1} + \overline{\beta_m} \tilde{p}_m$
11: end for
```

Now, we perform a block of s BICG iterations at once. That is, we calculate

$$\begin{matrix} [p_{m+1}, \ldots, p_{m+s}], & [r_{m+1}, \ldots, r_{m+s}], \\ [\tilde{p}_{m+1}, \ldots, \tilde{p}_{m+s}], & [\tilde{r}_{m+1}, \ldots, \tilde{r}_{m+s}], \end{matrix} \quad [x_{m+1}, \ldots, x_{m+s}],$$

given $\{p_m, \tilde{p}_m, r_m, \tilde{r}_m, x_m\}$.

Suppose $s > 0$ and $j \leq s$. Since the Krylov bases are nested, i.e., $\mathcal{K}_j(A, v) \subseteq \mathcal{K}_{j+1}(A, v)$, (3.2) tells us that

$$\begin{aligned} p_{m+j}, r_{m+j} &\in \mathcal{K}_{s+1}(A, p_m) + \mathcal{K}_s(A, r_m), \\ \tilde{p}_{m+j}, \tilde{r}_{m+j} &\in \mathcal{K}_{s+1}(A^H, \tilde{p}_m) + \mathcal{K}_s(A^H, \tilde{r}_m), \\ x_{m+j} - x_m &\in \mathcal{K}_{s+1}(A, p_m) + \mathcal{K}_s(A, r_m). \end{aligned}$$

Then to perform iterations m to $m+s$, we use the following Krylov matrices,

$$\begin{aligned} P_j &\coloneqq [\rho_0(A)p_m, \rho_1(A)p_m, \ldots, \rho_{j-1}(A)p_m], & \operatorname{span}(P_j) &= \mathcal{K}_j(A, p_m), \\ \tilde{P}_j &\coloneqq [\rho_0(A^H)\tilde{p}_m, \rho_1(A^H)\tilde{p}_m, \ldots, \rho_{j-1}(A^H)\tilde{p}_m], & \operatorname{span}(\tilde{P}_j) &= \mathcal{K}_j(A^H, \tilde{p}_m), \\ R_j &\coloneqq [\rho_0(A)r_m, \rho_1(A)r_m, \ldots, \rho_{j-1}(A)r_m], & \operatorname{span}(R_j) &= \mathcal{K}_j(A, r_m), \\ \tilde{R}_j &\coloneqq [\rho_0(A^H)\tilde{r}_m, \rho_1(A^H)\tilde{r}_m, \ldots, \rho_{j-1}(A^H)\tilde{r}_m], & \operatorname{span}(\tilde{R}_j) &= \mathcal{K}_j(A^H, \tilde{r}_m), \end{aligned} \tag{3.3}$$

where $\rho_j(z)$ is a polynomial of degree j, satisfying a three-term recurrence

$$\begin{aligned} &\rho_0(z) \coloneqq 1, \quad \rho_1(z) \coloneqq (z - \hat{\alpha}_0)\rho_0(z)/\hat{\gamma}_0, \quad \text{and} \\ &\rho_j(z) \coloneqq ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z) - \hat{\beta}_{j-2}\rho_{j-2}(z))/\hat{\gamma}_{j-1} \quad \text{for } j > 1. \end{aligned} \tag{3.4}$$

This derivation generalizes to polynomials satisfying longer recurrences; we explain our choice of three-term recurrences in section 4.1.

Using the Krylov matrices (3.3) and equations in (3.2), we represent components of the BICG iterates in $\mathbb{C}^n$ by their coordinates in the Krylov bases, that is, subspaces of $\mathbb{C}^n$ of dimension at most $2s+1$. We introduce vectors $\{a_j, \tilde{a}_j, c_j, \tilde{c}_j, e_j\}$ each of length $2s+1$ to represent vectors $\{p_{m+j}, \tilde{p}_{m+j}, r_{m+j}, \tilde{r}_{m+j}, x_{m+j} - x_m\}$ of length n:

(3.5)
$$\begin{matrix} p_{m+j} \eqqcolon [P_{s+1}, R_s]a_j, & r_{m+j} \eqqcolon [P_{s+1}, R_s]c_j, \\ \tilde{p}_{m+j} \eqqcolon [\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_j, & \tilde{r}_{m+j} \eqqcolon [\tilde{P}_{s+1}, \tilde{R}_s]\tilde{c}_j, \end{matrix} \quad x_{m+j} - x_m \eqqcolon [P_{s+1}, R_s]e_j,$$

where the base cases for these recurrences are given by

$$a_0 := \tilde{a}_0 := [1, 0_{1,2s}]^T, \quad c_0 := \tilde{c}_0 := [0_{1,s+1}, 1, 0_{1,s-1}]^T, \quad e_0 := 0_{2s+1,1}. \tag{3.6}$$

Then, the iterate updates (lines {4, 5, 6, 9, 10}) in the Krylov basis become, for $0 \le j \le s-1$,

$$[P_{s+1}, R_s]e_{j+1} := [P_{s+1}, R_s]e_j + \alpha_{m+j}[P_{s+1}, R_s]a_j, \tag{3.7}$$
$$[P_{s+1}, R_s]c_{j+1} := [P_{s+1}, R_s]c_j - \alpha_{m+j}A[P_{s+1}, R_s]a_j, \tag{3.8}$$
$$[\tilde{P}_{s+1}, \tilde{R}_s]\tilde{c}_{j+1} := [\tilde{P}_{s+1}, \tilde{R}_s]\tilde{c}_j - \overline{\alpha_{m+j}}A^H[\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_j, \tag{3.9}$$
$$[P_{s+1}, R_s]a_{j+1} := [P_{s+1}, R_s]c_{j+1} + \beta_{m+j}[P_{s+1}, R_s]a_j, \tag{3.10}$$
$$[\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_{j+1} := [\tilde{P}_{s+1}, \tilde{R}_s]\tilde{c}_{j+1} + \overline{\beta_{m+j}}[\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_j. \tag{3.11}$$

Next, we represent the multiplications by A and A^H (lines 5 and 6) in the new coordinates, in order to manipulate (3.8) and (3.9). First, note that the scalar coefficients (decorated with hats) in (3.4) can be encoded in the tridiagonal matrices

$$T_{j+1} := \begin{pmatrix} \hat{\alpha}_0 & \hat{\beta}_0 & & \\ \hat{\gamma}_0 & \hat{\alpha}_1 & \ddots & \\ & \hat{\gamma}_1 & \ddots & \hat{\beta}_{j-2} \\ & & \ddots & \hat{\alpha}_{j-1} \\ & & & \hat{\gamma}_{j-1} \end{pmatrix} \in \mathbb{C}^{(j+1)\times j} \tag{3.12}$$

which let us express the recurrence (for $0 < j \le s$) in matrix form

$$\begin{aligned} AP_j = P_{j+1}T_{j+1}, &\quad AR_{j-1} = R_jT_j, \\ A^H\tilde{P}_j = \tilde{P}_{j+1}T_{j+1}, &\quad A^H\tilde{R}_{j-1} = \tilde{R}_jT_j. \end{aligned} \tag{3.13}$$

Note that since we will update for $0 \le j \le s-1$, we only perform multiplication of A with p_{m+j} for these values of j (likewise for A^H and $\tilde{p}_{m+j}$). However, by (3.2), since

$$p_{m+j} \in \mathcal{K}_{j+1}(A, p_m) + \mathcal{K}_j(A, r_m), \qquad \tilde{p}_{m+j} \in \mathcal{K}_{j+1}(A^H, \tilde{p}_m) + \mathcal{K}_j(A^H, \tilde{r}_m)$$

we can write, for $0 \le j \le s-1$,

$$\begin{aligned} Ap_{m+j} = A[P_{s+1}, R_s]a_j = A[P_s, 0_{n,1}, R_{s-1}, 0_{n,1}]a_j = [P_{s+1}, R_s]T'a_j, \\ A^H\tilde{p}_{m+j} = A^H[\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_j = A^H[\tilde{P}_s, 0_{n,1}, \tilde{R}_{s-1}, 0_{n,1}]\tilde{a}_j = [\tilde{P}_{s+1}, \tilde{R}_s]T'\tilde{a}_j, \end{aligned} \tag{3.14}$$

where

$$T' := \begin{bmatrix} \begin{bmatrix} T_{s+1} & 0_{s+1,1} \end{bmatrix} & \\ & \begin{bmatrix} T_s & 0_{s,1} \end{bmatrix} \end{bmatrix}. \tag{3.15}$$

We substitute (3.5) and (3.14) into classical BICG. Each of lines 4, 5, 6, 9, and 10 is now expressed as a linear combination of the columns of the Krylov matrices, and we can match coordinates on the right- and left-hand sides to obtain the recurrences, for

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

$$e_{j+1} \coloneqq e_j + \alpha_{m+j} a_j, \qquad e_0 \coloneqq 0_{2s+1,1}, \tag{3.16}$$
$$c_{j+1} \coloneqq c_j - \alpha_{m+j} T' a_j, \qquad c_0 \coloneqq [0_{1,s+1}, 1, 0_{1,s-1}]^T, \tag{3.17}$$
$$\tilde{c}_{j+1} \coloneqq \tilde{c}_j - \overline{\alpha_{m+j}} T' \tilde{a}_j, \qquad \tilde{c}_0 \coloneqq [0_{1,s+1}, 1, 0_{1,s-1}]^T, \tag{3.18}$$
$$a_{j+1} \coloneqq c_{j+1} + \beta_{m+j} a_j \qquad a_0, \coloneqq [1, 0_{1,2s}]^T, \tag{3.19}$$
$$\tilde{a}_{j+1} \coloneqq \tilde{c}_{j+1} + \overline{\beta_{m+j}} \tilde{a}_j, \qquad \tilde{a}_0 \coloneqq [1, 0_{1,2s}]^T. \tag{3.20}$$

We also need scalar quantities (e.g., α_{m+j}, β_{m+j}, δ_{m+j+1}), which are computed from dot products involving the BICG iterates. We represent these dot products (lines 3 and 7) in the new basis, using the Gram-like matrix

$$G \coloneqq [\tilde{P}_{s+1}, \tilde{R}_s]^H [P_{s+1}, R_s], \tag{3.21}$$

so we have

$$(\tilde{r}_{m+j+1}, r_{m+j+1}) = ([\tilde{P}_{s+1}, \tilde{R}_s]\tilde{c}_{j+1}, [P_{s+1}, R_s]c_{j+1}) = (\tilde{c}_{j+1}, Gc_{j+1}), \tag{3.22}$$
$$(\tilde{p}_{m+j}, Ap_{m+j}) = ([\tilde{P}_{s+1}, \tilde{R}_s]\tilde{a}_j, A[P_{s+1}, R_s]a_j) = (\tilde{a}_j, GT'a_j). \tag{3.23}$$

Now we assemble the CA-BICG method from (3.16)–(3.20), (3.21)–(3.23)—see Algorithm 3.2.

Algorithm 3.2. Communication-Avoiding BICG (CA-BICG).

```
Require: Initial approximation $x_0$ for solving $Ax = b$, let $p_0 \coloneqq r_0 \coloneqq b - Ax_0$
 1: Choose $\tilde{r}_0$ arbitrarily s.t. $\delta_0 \coloneqq (\tilde{r}_0, r_0) \neq 0$, and let $\tilde{p}_0 \coloneqq \tilde{r}_0$
 2: for $m \coloneqq 0, s, 2s, \ldots,$ until convergence do
 3:     Compute $[P_{s+1}, R_s]$ and $[\tilde{P}_{s+1}, \tilde{R}_s]$ according to (3.3)
 4:     Compute $G$ according to (3.21)
 5:     Compute $T'$ according to (3.15) and (3.12)
 6:     Initialize $\{a_0, \tilde{a}_0, c_0, \tilde{c}_0, e_0\}$ according to (3.6)
 7:     for $j \coloneqq 0$ to $s-1$ do
 8:         $\alpha_{m+j} \coloneqq \delta_{m+j}/(\tilde{a}_j, GT'a_j)$
 9:         $e_{j+1} \coloneqq e_i + \alpha_{m+j} a_j$
10:         $c_{j+1} \coloneqq c_j - \alpha_{m+j} T' a_j$
11:         $\tilde{c}_{j+1} \coloneqq \tilde{c}_j - \overline{\alpha_{m+j}} T' \tilde{a}_j$
12:         $\delta_{m+j+1} \coloneqq (\tilde{c}_{j+1}, Gc_{j+1})$
13:         $\beta_{m+j} \coloneqq \delta_{m+j+1}/\delta_{m+j}$
14:         $a_{j+1} \coloneqq c_{j+1} + \beta_{m+j} a_j$
15:         $\tilde{a}_{j+1} \coloneqq \tilde{c}_{j+1} + \overline{\beta_{m+j}} \tilde{a}_j$
16:     end for
17:     Recover iterates $\{p_{m+s}, \tilde{p}_{m+s}, r_{m+s}, \tilde{r}_{m+s}, x_{m+s}\}$ according to (3.5)
18: end for
```

In parallel, the only communication in CA-BICG occurs in the outer loop, in lines 3 and 4; the inner loop lines operate on operands of size $O(ns/P + s^2)$, which we assume fit locally on each of the P processors. Therefore, communication does not occur in the inner loop, and CA-BICG can take s steps per $O(1)$ rounds of messages. In contrast, classical BICG can only take 1 step per $O(1)$ rounds of messages; this is an $\Theta(s)$-fold decrease in latency for well partitioned A.

In order to compute $s > 1$ vectors at one time, each processor requires additional source vector entries (the ghost zones) and must perform redundant computation. If A is well partitioned, then the ghost zones grow slowly with respect to s; thus the additional bandwidth and computation are both lower-order terms. We also note that in CA-BICG, each processor must send $\Theta(s^2)$ words to perform the matrix multiplication in line 4, while the equivalent dot products in s steps of classical BICG only require $\Theta(s)$ words moved per processor. Assuming $s^2 \ll M$, where M is the local memory size, this additional bandwidth cost is unlikely to be a bottleneck. Construction of G brings the computational cost of the dense operations to $O(s^2n/P)$, a factor of $O(s)$ larger than the classical algorithm. Under the reasonable assumption that the number of nonzeros of A per processor is greater than $O(sn/P)$, the SpMV computations dominate dense operations in the classical algorithm, so the additional cost of computing G is not a limiting factor.

In serial, CA-BICG moves data on lines 3, 4, and 17. We are most concerned with the data movement in reading A (line 3), since it is common that A requires far more storage than the vectors on which it operates. If A is well partitioned, then line 3 reads A $2 + o(1)$ times, whereas s iterations of classical BICG read A $2s$ times. This is an s-fold savings in latency and bandwidth for reading A. All three lines (3, 4, and 17) involve reading the vector iterates of length n, asymptotically the same communication cost as the classical algorithm. As in the parallel case, we perform a factor of $O(s)$ more computation in the dense operations, which is a lower-order term if the number of nonzeros in A is greater than $O(sn)$.

3.2. Communication-avoiding BICGSTAB.

Like classical BICG, classical BICGSTAB (Algorithm 3.3) has two communication-bound kernels: SpMV and inner products. Again, we replace SpMV with the matrix powers kernel, and inner products with a Gram-like matrix and vector. Because BICGSTAB expands the underlying Krylov space by two dimensions each iteration, we need $2s$ Krylov basis vectors to complete s iterations. First, we convert classical BICGSTAB (Algorithm 3.3) to an s-step method. Similarly to BICG, we can determine the dependencies for steps $m+1$ through $m+s$ by inspection of Algorithm 3.3. By induction on lines {5, 6, 9} of classical BICGSTAB, we can show that, given $m \geq 0, j > 0$, the vector iterates of classical BICGSTAB satisfy

$$\begin{aligned} &p_{m+j}, r_{m+j} \in \mathcal{K}_{2j+1}(A, p_m) + \mathcal{K}_{2j}(A, r_m) \\ &x_{m+j} - x_m \in \mathcal{K}_{2j}(A, p_m) + \mathcal{K}_{2j-1}(A, r_m). \end{aligned} \tag{3.24}$$

ALGORITHM 3.3. CLASSICAL BICGSTAB.

```
Require: Initial approximation $x_0$ for solving $Ax = b$, let $p_0 := r_0 := b - Ax_0$
1: Choose $\tilde{r}$ arbitrarily s.t. $\delta_0 := (\tilde{r}, r_0) \neq 0$
2: for $m := 0, 1, \ldots,$ until convergence do
3:     $\alpha_m := \delta_m / (\tilde{r}, Ap_m)$
4:     $\omega_m := (Ar_m - \alpha_m A^2 p_m, r_m - \alpha_m Ap_m)/(Ar_m - \alpha_m A^2 p_m, Ar_m - \alpha_m A^2 p_m)$
5:     $x_{m+1} := x_m + \alpha_m p_m + \omega_m (r_m - \alpha_m Ap_m)$
6:     $r_{m+1} := (I - \omega_m A)(r_m - \alpha_m Ap_m)$
7:     $\delta_{m+1} := (\tilde{r}, r_{m+1})$
8:     $\beta_m := (\delta_{m+1}/\delta_m) \cdot (\alpha_m/\omega_m)$
9:     $p_{m+1} := r_{m+1} + \beta_m (I - \omega_m A) p_m$
10: end for
```

As with BICG, since the Krylov bases are nested, we can use (3.24) to show that for $0 < j \le s$

$$\begin{aligned} p_{m+j}, r_{m+j} &\in \mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m), \\ x_{m+j} - x_m &\in \mathcal{K}_{2s+1}(A, p_m) + \mathcal{K}_{2s}(A, r_m). \end{aligned}$$

Then to find iterates $m+1$ to $m+s$, we use the Krylov matrices P_{2s+1} and R_{2s}, as defined in (3.3).

Then we can represent the n components of the BICGSTAB iterates by their $4s+1$ coordinates in the Krylov bases defined by P_{2s+1} and R_{2s}. We introduce coordinate vectors $\{a_j, c_j, e_j\}$ to represent vectors $\{p_{m+j}, r_{m+j}, x_{m+j} - x_m\}$, such that

$$\begin{aligned} p_{m+j} &=: [P_{2s+1}, R_{2s}]a_j, \\ r_{m+j} &=: [P_{2s+1}, R_{2s}]c_j, \end{aligned} \qquad x_{m+j} - x_m =: [P_{2s+1}, R_{2s}]e_j, \tag{3.25}$$

where the base cases for these recurrences are given by

$$a_0 \coloneqq [1, 0_{1,4s}]^T, \quad c_0 \coloneqq [0_{1,2s+1}, 1, 0_{1,2s-1}]^T, \quad e_0 \coloneqq 0_{4s+1,1}. \tag{3.26}$$

Then, the iterate updates (lines {5, 6, 9}) in the Krylov basis become, for $0 \le j \le s-1$,

$$\begin{aligned} [P_{2s+1}, R_{2s}]e_{j+1} \coloneqq {} & [P_{2s+1}, R_{2s}]e_j + \alpha_{m+j}[P_{2s+1}, R_{2s}]a_j \\ & + \omega_{m+j}([P_{2s+1}, R_{2s}]c_j - \alpha_{m+j}A[P_{2s+1}, R_{2s}]a_j), \end{aligned} \tag{3.27}$$

$$\begin{aligned} [P_{2s+1}, R_{2s}]c_{j+1} \coloneqq {} & [P_{2s+1}, R_{2s}]c_j - \alpha_{m+j}A[P_{2s+1}, R_{2s}]a_j \\ & - \omega_{m+j}A[P_{2s+1}, R_{2s}]c_j + \omega_{m+j}\alpha_{m+j}A^2[P_{2s+1}, R_{2s}]a_j, \end{aligned} \tag{3.28}$$

$$\begin{aligned} [P_{2s+1}, R_{2s}]a_{j+1} \coloneqq {} & [P_{2s+1}, R_{2s}]c_{j+1} + \beta_{m+j}[P_{2s+1}, R_{2s}]a_j \\ & - \beta_{m+j}\omega_{m+j}A[P_{2s+1}, R_{2s}]a_j. \end{aligned} \tag{3.29}$$

We want to represent multiplication by A and A^2 in the new basis. Let T_{j+1} be defined as in (3.12). We can then write, for $0 \le j \le s-1$,

$$A[p_{m+j}, r_{m+j}] = [P_{2s+1}, R_{2s}]T'[a_j, c_j], \qquad A^2 p_{m+j} = [P_{2s+1}, R_{2s}]T'' a_j, \tag{3.30}$$

where

$$T' \coloneqq \begin{bmatrix} [T_{2s+1} \quad 0_{2s+1,1}] & \\ & [T_{2s} \quad 0_{2s,1}] \end{bmatrix}, \qquad T'' \coloneqq T' \cdot T'. \tag{3.31}$$

We substitute (3.25) and (3.30) into classical BICGSTAB. Each of lines 5, 6, and 9 is now expressed as a linear combination of the columns of the Krylov matrices, and we can match coordinates on the right- and left-hand sides to obtain the recurrences, for $j = 0$ to $s-1$,

$$e_{j+1} \coloneqq e_j + \alpha_{m+j}a_j + \omega_{m+j}c_j - \omega_{m+j}\alpha_{m+j}T'a_j, \tag{3.32}$$

$$a_{j+1} \coloneqq c_{j+1} + \beta_{m+j}a_j - \beta_{m+j}\omega_{m+j}T'a_j, \tag{3.33}$$

$$c_{j+1} \coloneqq c_j - T'(\alpha_{m+j}a_j + \omega_{m+j}c_j) + \omega_{m+j}\alpha_{m+j}T''a_j, \tag{3.34}$$

where

$$a_0 \coloneqq [1, 0_{1,4s}]^T, \quad c_0 \coloneqq [0_{1,2s+1}, 1, 0_{1,2s-1}]^T, \quad e_0 \coloneqq 0_{4s+1,1}.$$

We also need scalar quantities α_{m+j}, ω_{m+j}, β_{m+j}, and δ_{m+j+1}, which are computed from dot products involving the BICGSTAB iterates. We represent these dot products in the new basis, using the Gram-like matrix G and vector g as

$$[G, g] \coloneqq [P_{2s+1}, R_{2s}]^H [[P_{2s+1}, R_{2s}], \tilde{r}]. \tag{3.35}$$

The formulas for the scalar quantities can be easily derived by substitution, as in the derivation for CA-BICG. Now we assemble the CA-BICGSTAB method shown in Algorithm 3.4.

Algorithm 3.4. Communication-Avoiding BICGSTAB (CA-BICGSTAB).

```
Require: Initial approximation $x_0$ for solving $Ax = b$, let $p_0 \coloneqq r_0 \coloneqq b - Ax_0$
 1: Choose $\tilde{r}$ arbitrarily s.t. $\delta_0 \coloneqq (\tilde{r}, r_0) \neq 0$
 2: for $m \coloneqq 0, s, 2s, \ldots,$ until convergence do
 3:     Compute $P_{2s+1}$ and $R_{2s}$ according to (3.3)
 4:     Compute $G$ and $g$ according to (3.35)
 5:     Compute $T'$ and $T''$ according to (3.31)
 6:     Initialize $\{a_0, c_0, e_0\}$ according to (3.26)
 7:     for $j \coloneqq 0$ to $s-1$ do
 8:         $\alpha_{m+j} \coloneqq \delta_{m+j}/(g, T'a_j)$
 9:         $\omega_{m+j} \coloneqq \frac{(T'c_j - \alpha_{m+j}T''a_j, Gc_j - \alpha_{m+j}GT'a_j)}{(T'c_j - \alpha_{m+j}T''a_j, GT'c_j - \alpha_{m+j}GT''a_j)}$
10:         $e_{j+1} \coloneqq e_j + \alpha_{m+j}a_j + \omega_{m+j}c_j - \omega_{m+j}\alpha_{m+j}T'a_j$
11:         $c_{j+1} \coloneqq c_j - \omega_{m+j}T'c_j - \alpha_{m+j}T'a_j + \omega_{m+j}\alpha_{m+j}T''a_j$
12:         $\delta_{m+j+1} \coloneqq (g, c_{j+1})$
13:         $\beta_{m+j} \coloneqq (\delta_{m+j+1}/\delta_{m+j}) \cdot (\alpha_{m+j}/\omega_{m+j})$
14:         $a_{j+1} \coloneqq c_{j+1} + \beta_{m+j}a_j - \beta_{m+j}\omega_{m+j}T'a_j$
15:     end for
16:     Recover iterates $\{p_{m+s}, r_{m+s}, x_{m+s}\}$ according to (3.25)
17: end for
```

Analogous to CA-BICG, the only communication in parallel CA-BICGSTAB occurs in lines 3 and 4. CA-BICGSTAB can thus reduce parallel latency by a factor of $\Theta(s)$, albeit at the cost of increasing bandwidth. We refer to the above discussion for CA-BICG, except noting that CA-BICGSTAB does not require multiplication by A^H, and instead requires $2s$ multiplications by A per s iterations.

In serial, CA-BICGSTAB moves data on lines 3, 4, and 16. Again, this is analogous to CA-BICG; we reduce the serial latency and bandwidth by a factor of $\Theta(s)$.

4. Improving the Krylov basis.

In our numerical experiments (see section 5), we make use of two methods for improving numerical qualities of the generated Krylov bases. First, we give details on implementing general three-term polynomials for the Krylov basis vectors. We give examples for Newton and Chebyshev bases, which we use in our tests (in addition to the monomial basis). We review matrix equilibration techniques, as described by [19], which are effective at reducing the basis norm without incurring additional communication.

4.1. Changing polynomial basis.

A concern with s-step methods is how to compute the s-step Krylov bases stably in practice. The sequence of vectors $(x, Ax, A^2x, \ldots, A^sx)$, i.e., the *monomial basis* of the $(s+1)$-dimensional Krylov space of A with respect to x, may become nearly linearly dependent for $s \ll n$. This happens when the sequence converges to an eigenspace of A. This loss of linear independence can cause the underlying Krylov methods to break down or stagnate, and thus fail to converge. As previous authors have demonstrated (e.g., [19, 3, 27]), we found changing the polynomial basis improved stability and convergence properties. Numerical results are presented in section 5.

The three-term recurrence satisfied by polynomials $\{\rho_j(z)\}_{j=0}^s$ in (3.4) can be represented by a tridiagonal matrix T_{s+1} (3.12) consisting of the recurrence coefficients. In this work we consider the Newton and the Chebyshev polynomials.

Newton. This approach follows that of [3, 19, 27]. The (scaled) Newton polynomials are defined by the recurrence coefficients

$$\hat{\alpha}_j = \theta_j, \qquad \hat{\beta}_j = 0, \qquad \hat{\gamma}_j = \sigma_j, \tag{4.1}$$

where the σ_j are *scaling factors* and the *shifts* θ_j are chosen to be eigenvalue estimates. After choosing s shifts, we permute them according to a Leja ordering (see, e.g., [28]). Informally, the Leja ordering recursively selects each θ_j to maximize a certain measure on the set $\{\theta_i\}_{i\leq j} \subset \mathbb{C}$; this helps avoid repeated or nearly repeated shifts. Real matrices may have complex eigenvalues, so the Newton basis may introduce complex arithmetic; to avoid this, we use the modified Leja ordering [19], which exploits the fact that complex eigenvalues of real matrices occur in complex conjugate pairs. The modified Leja ordering means that for every complex shift $\theta_j \notin \mathbb{R}$ in the sequence, its complex conjugate $\overline{\theta_j}$ is adjacent, and the complex conjugate pairs (θ_j, θ_{j+1}) are permuted so that $\Im(\theta_j) > 0$. This leads to the recurrence coefficients

$$\hat{\alpha}_j = \Re(\theta_j), \qquad \hat{\beta}_j = \begin{cases} -\Im(\theta_j)^2 & \theta_j = \overline{\theta_{j+1}} \wedge \Im(\theta_j) > 0, \\ 0 & \text{otherwise}, \end{cases} \qquad \hat{\gamma}_j = \sigma_j. \tag{4.2}$$

Both Leja orderings can be computed efficiently. For CA-KSMs, choosing σ_j is challenging; we cannot simply scale each basis vector by its Euclidean norm as it is generated since this eliminates our communication savings. In our experiments, we pick all scaling factors $\sigma_j = 1$ (i.e., no scaling), instead relying entirely on matrix equilibration, described in section 4.2, to keep the spectral radius $\rho(A)$ close to 1.

Chebyshev. The Chebyshev basis requires two parameters, complex numbers d and c, where $d \pm c$ are the foci of a bounding ellipse for the spectrum of A. The scaled, shifted, and rotated Chebyshev polynomials $\{\tilde{\tau}_j\}_{j\geq 0}$ can then be written as

$$\tilde{\tau}_j(z) \coloneqq \tau_j((d-z)/c)/\tau_j(d/c) \eqqcolon \tau_j((d-z)/c)/\sigma_j, \tag{4.3}$$

where the Chebyshev polynomials (of the first kind) $\{\tau_j\}_{j\geq 0}$ are

$$\begin{aligned} &\tau_0(z) \coloneqq 1, \quad \tau_1(z) \coloneqq z, \quad \text{and} \\ &\tau_j(z) \coloneqq 2z\tau_{j-1}(z) - \tau_{j-2}(z) \quad \text{for } j > 1; \end{aligned} \tag{4.4}$$

substituting (4.3) into (4.4), we obtain

$$\begin{aligned} &\tilde{\tau}_0(z) = 1, \quad \tilde{\tau}_1(z) = \sigma_0(d-z)/(c\sigma_1), \quad \text{and} \\ &\tilde{\tau}_j(z) = 2\sigma_{j-1}(d-z)\tilde{\tau}_{j-1}(z)/(c\sigma_j) - \sigma_{j-2}\tilde{\tau}_{j-2}(z)/\sigma_j \quad \text{for } j > 1. \end{aligned} \tag{4.5}$$

Extracting coefficients, we obtain

$$\hat{\alpha}_j = d, \qquad \hat{\beta}_j = -c\sigma_j/(2\sigma_{j+1}), \qquad \hat{\gamma}_j = \begin{cases} -c\sigma_1/\sigma_0, & j = 0, \\ -c\sigma_{j+1}/(2\sigma_j), & j > 0. \end{cases} \tag{4.6}$$

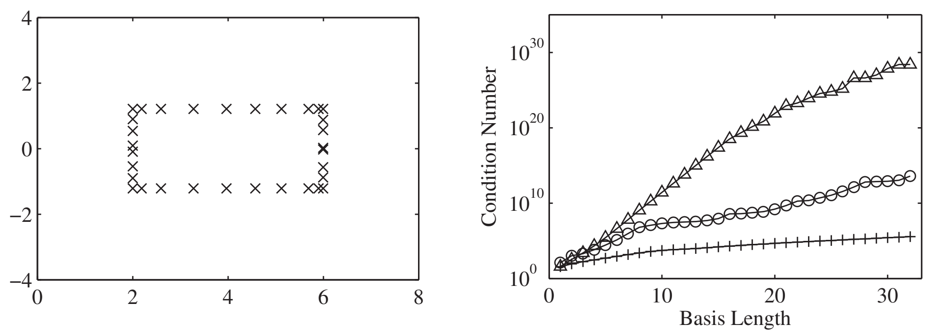

FIG. 5.1. *Basis properties for cdde. Left: computed Leja points* ($\times$) *of cdde, plotted in the complex plane. Right: basis condition number growth rate. The x-axis denotes basis length* s *and the y-axis denotes the basis condition number for monomial* ($\triangle$)*, Newton* ($\circ$)*, and Chebyshev* ($+$) *bases. Leja points on the left were used to generate the bases shown in the right plot.*

Note that the transformation $z \to (d-z)/c$ maps ellipses with foci $f_{1,2} = d \pm c$ to ellipses with foci at ∓ 1, especially the line segment (f_1, f_2) to $(-1, 1)$. If A is real, then the ellipse is centered on the real axis; thus $d \in \mathbb{R}$, so c is either real or imaginary. In the former case, arithmetic will be real. In the latter case ($c \in i\mathbb{R}$), we avoid complex arithmetic by replacing $c \coloneqq c/i$. This is equivalent to rotating the ellipses 90°.

For real matrices, [20] also gives a three-term recurrence, where

$$\hat{\alpha}_j = d, \qquad \hat{\beta}_j = c^2/(4g), \qquad \hat{\gamma}_j = \begin{cases} 2g, & j = 0, \\ g, & j > 0. \end{cases} \tag{4.7}$$

It is assumed that the spectrum is bounded by the rectangle $\{z : |\Re(z) - d| \leq a, |\Im(z)| \leq b\}$ in the complex plane, where $a \geq 0$, $b \geq 0$, and d are real. Here we choose $c = \sqrt{a^2 - b^2}$, $g = \max\{a, b\}$. Note that for real-valued matrices, recurrence (4.7) is usually sufficient for capturing the necessary spectral information.

4.2. Matrix equilibration. Successively scaling (i.e., normalizing) the Krylov vectors as they are generated increases the numerical stability of the basis generation step. As discussed in [19], successively scaling the basis vectors (e.g., by their Euclidean norms) is not possible in the CA variants, as it reintroduces the global communication requirement between SpMV operations. As an alternative, one can perform matrix equilibration. For unsymmetric matrices, this involves applying diagonal row and column scalings, D_r and D_c, such that each row and column of the equilibrated matrix $D_r A D_c$ has norm one. This is performed once offline before running the CA-KSM. Results in [19] indicate that this technique is an effective alternative to the successively scaled versions of the bases.

5. Numerical experiments.

We present convergence results for two matrices which exemplify the dependence of typical numerical behavior on the chosen basis and s value. Note that for CA-BICGSTAB, the Krylov bases computed in each outer loop are of length $2s$. In all tests, the right-hand side b was constructed such that the true solution x is the n-vector with components $x_i = 1/\sqrt{n}$.

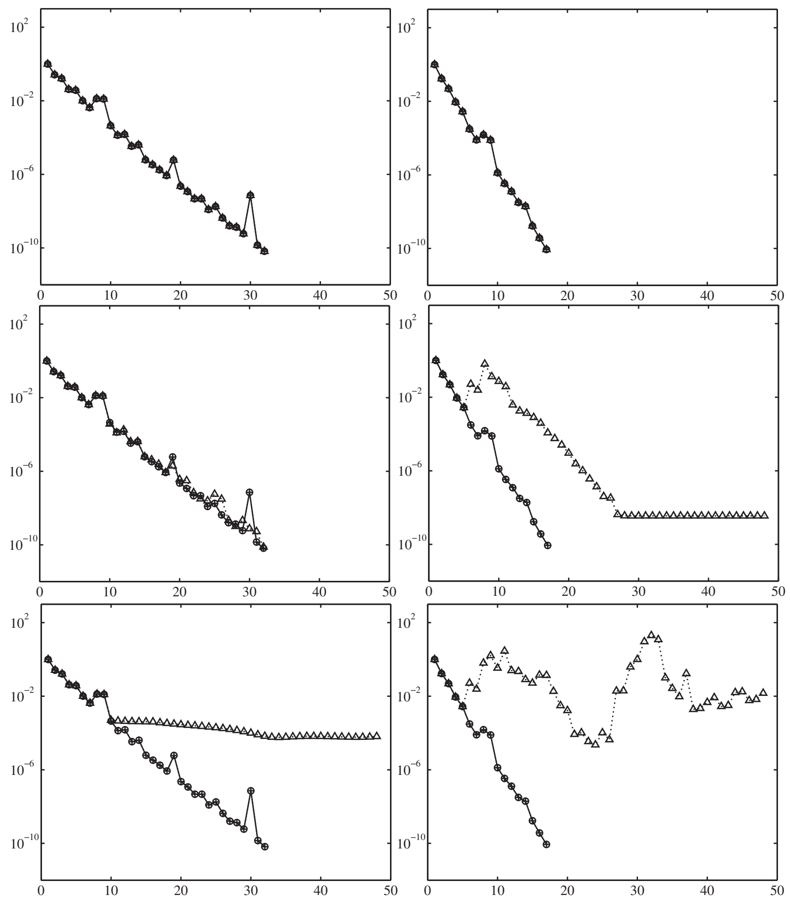

FIG. 5.2. *Convergence for cdde matrix, for (CA)-BICG (left) and (CA)-BICGSTAB (right), and various s values: $s = 4$ (top), $s = 8$ (middle), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the 2-norm of the (normalized) true residual (i.e., $\|b-Ax_m\|_2/\|b\|_2$). Each plot shows convergence for the classical algorithm ($-$), as well as for CA variants ($\cdots$), using the monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases.*

cdde. Our first test problem comes from the constant-coefficient convection diffusion equation

$$
\begin{aligned}
-\Delta u + 2p_1 u_x + 2p_2 u_y - p_3 u_y &= f \qquad \text{in} \qquad [0,1]^2,\\
u &= g \qquad \text{on} \qquad \partial[0,1]^2.
\end{aligned}
$$

This problem is widely used for testing and analyzing numerical solvers [2]. We discretized the PDE using centered finite difference operators on a 512×512 grid with $(p_1, p_2, p_3) = (25, 600, 250)$, resulting in an unsymmetric matrix with $n = 262K$, $\text{nnz}(A) = 1.3M$, and condition number ~ 5.5. To select the Leja points, we used the convex hull of the known spectrum, given in [2]. Figure 5.1 shows the selected Leja points and resulting basis condition number for the monomial, Newton, and Chebyshev bases, for basis lengths up to $s = 32$. Figure 5.2 shows convergence results for CA-BICG and CA-BICGSTAB for the different bases, for $s \in \{4, 8, 16\}$.

xenon1. The xenon1 matrix is an unsymmetric matrix from the University of Florida Sparse Matrix Collection [11]. This matrix is used in analyzing elastic prop-

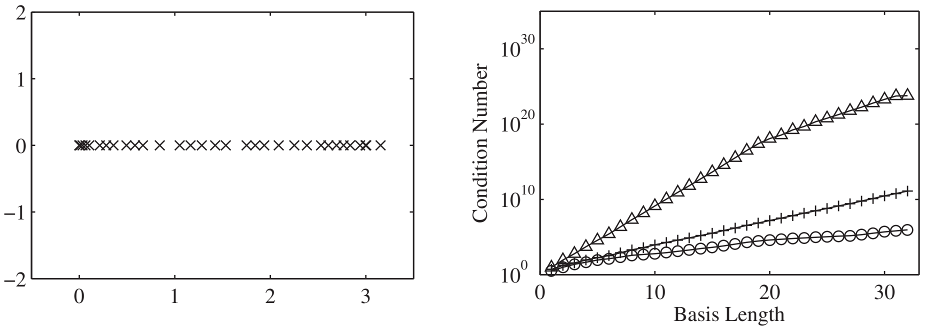

FIG. 5.3. *Basis properties for xenon*1*. Left: computed Leja points* ($\times$) *of xenon*1*, plotted in the complex plane. Right: basis condition number growth rate. The x-axis denotes basis length s and the y-axis denotes the basis condition number for monomial* ($\triangle$)*, Newton* ($\circ$)*, and Chebyshev* ($+$) *bases. Leja points on left were used to generate the bases shown in the right plot.*

erties of crystalline compounds [11]. Here, $n = 48.6K$ and $\text{nnz}(A) = 1.2M$. This test case is less well-conditioned than the first, with condition number $\sim 1.1 \cdot 10^5$. As a preprocessing step, we performed matrix equilibration, as described in [19]. The xenon1 matrix has all real eigenvalues. Therefore, Leja points (and thus basis parameters) were selected based only on estimates of the maximum and minimum eigenvalues obtained from ARPACK (MATLAB `eigs`). Figure 5.3 shows the generated Leja points and resulting basis condition numbers for the monomial, Newton, and Chebyshev bases, for basis lengths up to $s = 32$. Figure 5.4 shows convergence results for CA-BICG and CA-BICGSTAB for the different bases, for $s \in \{4, 8, 16\}$.

5.1. Convergence rate. For CA-KSMs with the monomial basis, we generally see the convergence rate decrease (relative to the classical method) as s grows larger. For example, in Figure 5.2 for CA-BICGSTAB, we see a decrease in convergence rate of the monomial basis from $s = 4$ to $s = 8$. This is due to roundoff error in computation of the basis vectors in the matrix powers kernel, which results in a larger perturbation to the finite precision Lanczos recurrence that determines the rate of convergence (see, e.g., [32]). At some point, when s becomes large, the CA-KSMs with the monomial basis will fail to converge due to (near) numerical rank deficiencies in the generated Krylov bases. For both of our test matrices, both CA-KSMs with the monomial basis fail to converge to the desired tolerance when $s = 16$.

The CA-KSMs with the Newton and Chebyshev bases, however, are able to maintain convergence closer to that of the classical method, even for s as large as 16 (again, note that for CA-BICGSTAB, $s = 16$ requires basis lengths of 32). This is especially evident if we have a well-conditioned matrix, as in Figure 5.2 (cdde). For this matrix, both CA-BICG and CA-BICGSTAB with the Newton and Chebyshev bases converge in the same number of iterations as the classical method, for all tested s values.

As observed in Figure 5.4 (xenon1), for CA-BICGSTAB, the dependence of convergence rate on s is less predictable for the Newton and Chebyshev bases, although we still observe eventual convergence of the true residual to the desired tolerance. This is likely due to the large condition number of xenon1.

5.2. Maximum attainable accuracy. We tested for convergence of the true residual to a level of 10^{-10} (i.e., $\|b - Ax_m\|_2 / \|b\|_2 \leq 10^{-10}$). For both test matrices, for CA-BICGSTAB with the monomial basis and $s = 8$, we observe that the norm of the true residual stagnates and never reaches the desired level despite convergence of

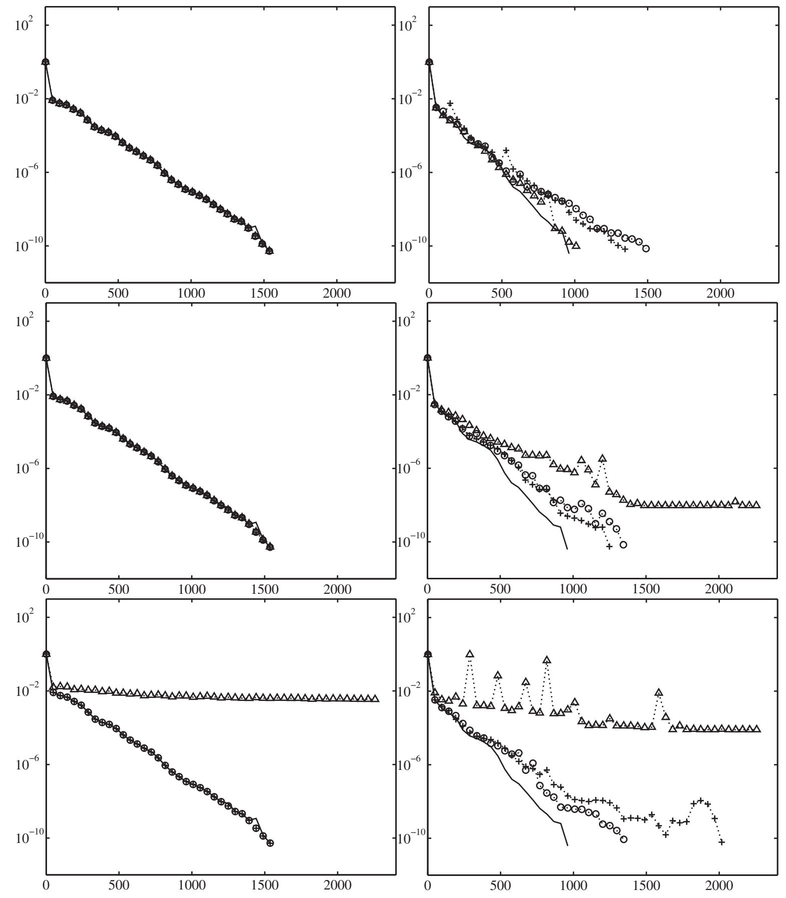

FIG. 5.4. *Convergence for xenon*1 *matrix, for (CA)-BICG (left) and (CA)-BICGSTAB (right), and various s values: $s = 4$ (top), $s = 8$ (middle), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the* 2*-norm of the (normalized) true residual (i.e., $\|b - Ax_m\|_2/\|b\|_2$). Each plot shows convergence for the classical algorithm* $(-)$*, as well as for CA- variants* $(\cdots)$*, using the monomial* $(\triangle)$*, Newton* $(\circ)$*, and Chebyshev* $(+)$ *bases.*

the computed residual. This effect is due to floating point roundoff error, which causes discrepancy between the true and computed residuals. As the computed residual converges to zero, the updates to the true residual rapidly become negligible—thus the method may fail to converge to the desired tolerance. The level where stagnation occurs is the *maximum attainable accuracy*.

This phenomenon has been observed before for the classical implementations, and is known to especially plague two-sided KSMs, which can have arbitrarily large iterates [17]. We observe that this problem is also present in the CA-KSM variants, and is in fact exacerbated the larger the s value. Residual replacement methods have been used effectively to combat this problem in the BICG method [34]. We have performed an equivalent analysis of floating point roundoff error in CA-KSMs and derived an analogous residual replacement strategy. Our analysis and experimental results show that this strategy can significantly improve accuracy without asymptotically affecting the communication or computation cost of CA-KSMs [5].

In classical KSMs, the bound for the deviation of the true and computed residual is provably worse for the three-term variants than the two-term variants [17].

Although omitted in this work, we have also derived a three-term variant of the CA-BICG method. The three-term formulation can be attractive in some situations since it requires computation of only two Krylov bases, as in the classical algorithms. Our empirical results suggest that, similar to the classical methods, the deviation of residuals is indeed larger in the three-term CA-KSM variants.

5.3. Choice of basis in practice. We offer guidance in choosing a basis in practice. If s is limited to small values by performance constraints (based on the structure and density of A^s) or if A is very well-conditioned, the monomial basis will usually perform comparably to the Chebyshev and Newton bases. In this case, the monomial basis is preferable, since fewer floating point operations are required in basis computation and basis change operations. If $s > 8$ is required, Newton or Chebyshev bases should be used, as the monomial basis is often quite ill-conditioned.

In practice, however, the user might not have a priori information about the eigenvalues of A. In this case, if a Newton or Chebyshev basis is desired, we must first find eigenvalue estimates for A. This is commonly accomplished by taking $\Theta(s)$ steps of Lanczos to construct the tridiagonal matrix of Lanczos coefficients. The eigenvalues of the tridiagonal matrix (Ritz or Petrov values) are often good estimates for the eigenvalues of A. Because BICG is based on the nonsymmetric Lanczos process, we can write this tridiagonal matrix in terms of the coefficients in (CA-)BICG. That is, we can use the monomial basis for the first $\Theta(s)$ steps (potentially using the classical algorithm) and compute $\Theta(s)$ Ritz values from the resulting coefficients. We can continue to refine our estimates using the new Ritz values generated after each outer iteration. Using the computed Ritz values, one can either compute (modified) Leja points for the Newton basis or bounding ellipse parameters for the Chebyshev basis [27]. Generating a basis using this approach has been shown to do as well as generating a basis with knowledge of all eigenvalues of A [27].

Note that if A is real we only require estimates for the eigenvalues with maximum and minimum real and imaginary components to compute an acceptable bounding ellipse for the Chebyshev basis. Furthermore, if the eigenvalues of A are real (as with xenon1 in Figure 5.3), we need only find estimates for the maximum and minimum eigenvalues in order to compute parameters for both the Newton and Chebyshev bases. The basis condition number plot in Figure 5.3 demonstrates that this information is sufficient for generating well-conditioned Newton and Chebyshev bases.

6. Future work and conclusions.

We have derived two new CA-KSMs, CA-BICG and CA-BICGSTAB, and studied their convergence behavior for various polynomial bases and basis sizes. Our results indicate that with well-conditioned s-step polynomial bases, the CA-KSMs can achieve convergence rates similar to their classical counterparts for basis lengths as high as 16, suitable for many applications. Recall that even small s values are of interest in practice; a basis length of s allows for up to an $s\times$ speedup for communication-bound problems.

Although our convergence results are promising, there is much remaining work involved in creating stable, high-performance implementations, as well as making these methods practical and available to the scientific computing community. We discuss these primary areas of future work in the subsections below.

6.1. Preconditioning in CA-KSMs. Preconditioning is used to accelerate convergence in KSMs. Our concern is primarily the applicability of our methods to the preconditioned case, such that communication can still be avoided. If preconditioner M^{-1} is too dense in structure, we cannot use our usual communication-avoiding strategy in computing the basis vectors, as the matrix powers kernel requires that the input matrix be well partitioned.[1]

If, however, the preconditioned system $M^{-1}A$ remains well partitioned, our communication-avoiding strategies still apply, with only minor required changes to the algorithms. The simplest case for which this holds is diagonal preconditioning, which is sufficient for many scientific applications. Here, application of M^{-1} to both A and b requires only local work. Others have discussed the use of polynomial preconditioners [29]. These could be easily incorporated in our methods, as the general implementation of the matrix powers kernel computes polynomials of A.

Hoemmen and co-workers devised an algorithm to avoid communication in the case of structured preconditioners, namely hierarchical semiseparable (HSS) matrices [19, 13]. We have improved upon Hoemmen's initial formulation; we show that one can asymptotically reduce communication without significantly increasing the amount of computation (with respect to the cost of computing s HSS matrix vector multiplies, described in [37]). This algorithm will be described in depth in a future publication.

6.2. Improving numerical stability. Understanding how CA-KSMs behave in finite precision is crucial to making CA-KSMs attractive choices in practice. Many methods exist for improving stability in classical two-sided KSMs, including residual replacement strategies [34] and lookahead techniques (see, e.g., [16, 26]). We have extended these strategies to CA-KSMs, as described below.

Residual replacement strategies. Our convergence results indicate that, like classical implementations, CA-KSMs suffer from roundoff error in finite precision, which can decrease the maximum attainable accuracy of the solution. A quantitative analysis of roundoff error and numerical stability in CA-KSMs can be found in [5]. To prevent buildup of error, we have explored implicit residual replacement in CA-KSMs as a method to limit the deviation of true and computed residuals when high accuracy is required, with promising results [5]. In addition to residual replacement strategies, we plan to explore techniques to improve numerical stability such as the use of dispersive Lanczos [1], the use of extended precision in (perhaps only some of) the CA-KSM iterations, and the use of variable basis lengths.

Consistent/inconsistent formulations and lookahead. The versions of classical BICG and BICGSTAB presented in Algorithms 3.1 and 3.3 are called *consistent* formulations, since the Lanczos vectors $\{r_m\}$ are artificially restricted to be the unscaled residuals of the current candidate solution. In an *inconsistent* formulation, the residual vectors may be scaled arbitrarily. This flexibility gives more control over the sizes of the iterates and scalar coefficients, and thus the bounds on roundoff error [15]. Since they only involve a small amount of additional scalar computation, we can derive communication-avoiding formulations of the inconsistent BICG and BICGSTAB variants introduced in [15] and [16] in the same way we obtained Algorithms 3.2 and 3.4.

Inconsistent formulations do not, however, help in the case of a Lanczos breakdown. Lookahead techniques may allow nonsymmetric Lanczos-type solvers, including BICG and BICGSTAB, to continue past a Lanczos breakdown. Kim and Chronopoulous [21] have noted that for their s-step nonsymmetric Lanczos algorithm, in exact arithmetic, if breakdown occurs at iteration m in the classical algorithm, then the same breakdown will only occur in the s-step variant if m is a multiple of s. They

[1]Note that we could compute the basis vectors using s SpMVs in the outer loop and still save communication by blocking dot products between basis vectors, as discussed in [31].

also comment that lookahead in the exact arithmetic s-step method amounts to ensuring that no blocked inner products between basis matrices have determinant zero. Hoemmen, however, has commented that the situation is more complicated in finite precision, as it is difficult to determine which type of breakdown occurred [19].

Nonsymmetric Lanczos-based CA-KSMs can thus exploit the matrix powers optimization for two purposes: for increasing performance of the algorithm and increasing stability by use of lookahead techniques to detect breakdown [19]. As the s-step structure of our CA algorithms is similar to the technique used in lookahead algorithms, we can add communication-avoiding lookahead to our CA-KSMs for little additional computational cost. Preliminary experiments are promising.

6.3. High-performance optimizations. We have determined three opportunities for extending the approach in [23, 24] to further optimize high-performance implementations of CA-KSMs. These include a new *streaming matrix powers* optimization, which can reduce communication in the sequential algorithm for stencil-like matrices, improvements to sparse matrix partitioning for the matrix powers computation, and use of multiple source vectors to further increase data reuse.

Streaming matrix powers. Consider the case where A can be represented in $o(n)$ words of data (e.g., a stencil with constant coefficients), where we assume $n > M$, the fast (local) memory size. The Krylov basis vectors require moving $\Theta(sn)$ words of data, so the serial communication bottleneck has shifted from loading A s times ($o(sn)$ data movement) to loading/storing the Krylov basis vectors ($\Theta(sn)$ data movement). In CA-BICG and CA-BICGSTAB, we can reduce the serial bandwidth and latency of moving the Krylov basis vectors by a factor of $\Theta(s)$ at the cost of performing double the sparse flops. We discuss the required kernel for two-term CA-BICG; the generalization to CA-BICGSTAB is straightforward.

In CA-BICG, the Krylov basis vectors are only accessed twice after being computed: when computing G (line 4) and when recovering iterates (line 17). Also, observe that we can execute lines 4 and 17 reading the Krylov bases one time each. We will interleave the computations of lines 3 and 4, and then repeat line 3 just before line 17, and interleave those operations as well. In the first case, the overall memory traffic is loading the $\Theta(n)$ input vector data for the matrix powers kernel—we assume the $\Theta(s^2)$ matrix G fits in fast memory and can be discarded at the end of the $j = s - 1$ iteration of line 12. In the second case, we load the same amount of vector data, plus $O(s^2)$ data (the coefficient vectors (3.5)), and store the five iterates ($\Theta(n)$). Overall, this optimization reduces both bandwidth and latency terms by a factor of $\Theta(s)$. This approach can be generalized to interleaving the matrix powers kernel with other vector operations, such as the tall-skinny QR kernel used in CA-GMRES [24].

Although this approach requires twice as many calls to the matrix powers kernel, there still may be a practical performance gain, especially under the assumption that A can be represented with $o(n)$ words of data: from a communication standpoint, A is inexpensive to apply.

Hypergraph partitioning. Hypergraph models, unlike graph models, can be constructed for the matrix A such that the minimum P-way cut of the hypergraph corresponds exactly to a partition which minimizes communication among P processors in an SpMV operation [6]. We have extended the hypergraph model to partition for s repeated SpMVs, the matrix powers computation. Construction of the hypergraph for this case, however, requires determining the structure of A^s, a costly operation. We have new methods for reducing both the computational cost and storage requirements for the hypergraph model while still obtaining a better partition than can be found using graph partitioning, or hypergraph partitioning of A. This approach uses a probabilistic structure prediction algorithm, based on Cohen's linear time algorithm for estimating the size of the transitive closure [9, 10]. We could also use this algorithm to determine the maximum s value such that A^s is still well partitioned.

Multiple input vectors. In general, simultaneously operating on multiple input vectors improves SpMV performance. We call this operation sparse matrix-matrix multiplication (SpMM). If M is the local memory size, and there are $O(\sqrt{M})$ nonzeros per row of A, we can perform $c = \Theta(\sqrt{M})$ SpMVs while loading A (and storing the output vectors) only once. Thus, we can increase the computational intensity of the matrix powers kernel by a factor of c by performing SpMM instead SpMV. We remark that communication-avoiding variants of *block* Krylov methods (see, e.g., [14]) could easily exploit the combination of matrix powers and SpMM optimizations.

6.4. Using CA-KSMs in practice.

Many complex implementation decisions are required for both matrix powers kernel performance and convergence of the CA-KSMs. Optimizations discussed in this work are both machine and matrix dependent, which makes autotuning and code generation an attractive approach. Many ongoing projects exist for this purpose (see, e.g., [4, 7, 22]). By further analytical and empirical study of the convergence and performance properties of our CA-KSMs, we can design effective autotuners and integrate our work into existing frameworks. These tools will automatically select an appropriate code variant that achieves both performance and stability based on matrix structure, eigenvalue estimates, and other properties of A. Employing such techniques lifts the burden of expertise from the user, making CA-KSMs more accessible to a range of computational scientists.

REFERENCES

[1] H. Avron, A. Druinsky, and S. Toledo, *Dispersive Lanczos or (when) does Lanczos converge?*, in Proceedings of the Fifth SIAM Workshop on Combinatorial Scientific Computing, Darmstadt, Germany, SIAM, 2011, p. 54.

[2] Z. Bai, D. Day, J. Demmel, and J. Dongarra, *A Test Matrix Collection for Non-Hermitian Eigenvalue Problems*, Technical report CS-97-355, Department of Computer Science, University of Tennessee, 1997.

[3] Z. Bai, D. Hu, and L. Reichel, *A Newton basis GMRES implementation*, IMA J. Numer. Anal., 14 (1994), p. 563.

[4] J. Byun, R. Lin, K. Yelick, and J. Demmel, *Autotuning Sparse Matrix-Vector Multiplication for Multicore*, ACM Trans. Math. Software, submitted.

[5] E. Carson and J. Demmel, *A Residual Replacement Strategy for Improving the Maximum Attainable Accuracy of Communication-Avoiding Krylov Subspace Methods*, SIAM J. Matrix Anal. Appl., submitted.

[6] U. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distributed Systems, 10 (1999), pp. 673-693.

[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 Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques, Raleigh, NC, ACM, 2009.

[8] A. Chronopoulos and C.W. Gear, *S-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math, 25 (1989), pp. 153–168.

[9] E. Cohen, *Estimating the size of the transitive closure in linear time*, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, IEEE, 1994, pp. 190–200.

[10] E. Cohen, *Size-estimation framework with applications to transitive closure and reachability*, J. Comput. System Sci., 55 (1997), pp. 441–453.

[11] T. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1.

[12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[13] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding Communication in Computing Krylov Subspaces*, Technical report UCB/EECS-2007-123, EECS Department, University of California, Berkeley, 2007.

[14] G. Golub and R. Underwood, *The block Lanczos method for computing eigenvalues*, ACM Trans. Math. Software, 3 (1977), pp. 361–377.

[15] M. Gutknecht, *Lanczos-type solvers for nonsymmetric linear systems of equations*, Acta Numer., 6 (1997), pp. 271–398.

[16] M. H. Gutknecht and K. J. Ressel, *Look-ahead procedures for Lanczos-type product methods based on three-term Lanczos recurrences*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1051–1078.

[17] M. H. Gutknecht and Z. Strakos, *Accuracy of two three-term and three two-term recurrences for Krylov space solvers*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 213–229.

[18] A. Hindmarsh and H. Walker, *Note on A Householder Implementation of the GMRES Method*, Technical report UCID-20899, Lawrence Livermore National Laboratory, Livermore, CA, 1986.

[19] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, Berkeley, 2010.

[20] W. Joubert and G. Carey, *Parallelizable restarted iterative methods for nonsymmetric linear systems. Part* I: *Theory*, Int. J. Comput. Math., 44 (1992), pp. 243–267.

[21] S. Kim and A. Chronopoulos, *An efficient nonsymmetric Lanczos method on parallel vector computers*, J. Comput. Appl. Math., 42 (1992), pp. 357–374.

[22] A. LaMielle and M. Strout, *Enabling Code Generation Within the Sparse Polyhedral Framework*, Technical report CS-10-102, Colorado State University, Fort Collins, CO, 2010.

[23] M. Mohiyuddin, *Tuning Hardware and Software for Multiprocessors*, Ph.D. thesis, University of Califorina, Berkeley, 2012.

[24] 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, ACM, 2009, p. 36.

[25] J. Morlan, *Auto-Tuning the Matrix Powers Kernel with SEJITS*, Master's thesis, University of California, Berkeley, 2012.

[26] B. Parlett, D. Taylor, and Z. Liu, *A look-ahead Lanczos algorithm for unsymmetric matrices*, Math. Comp., 44 (1985), pp. 105–124.

[27] B. Philippe and L. Reichel, *On the generation of Krylov subspace bases*, Appl. Numer. Math, 62 (2012), pp. 1171–1186.

[28] L. Reichel, *Newton interpolation at Leja points*, BIT, 30 (1990), pp. 332–346.

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

[30] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003.

[31] S. Toledo, *Quantitative Performance Modeling of Scientific Computations and Creating Locality in Numerical Algorithms*, Ph.D. thesis, MIT, Cambridge, MA, 1995.

[32] C. Tong and Q. Ye, *Analysis of the finite precision bi-conjugate gradient algorithm for nonsymmetric linear systems*, Math. Comp., 69 (2000), pp. 1559–1576.

[33] H. A. Van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), p. 631–644.

[34] H. A. Van der Vorst and Q. Ye, *Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals*, SIAM J. Sci. Comput., 22 (2000), pp. 835–852.

[35] J. Van Rosendale, *Minimizing Inner Product Data Dependencies in Conjugate Gradient Iteration*, Technical report 172178, ICASE-NASA, 1983.

[36] H. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), p. 152.

[37] J. Xia, S. Chandrasekaran, M. Gu, and X. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.