Avoiding communication in the Lanczos bidiagonalization routine and associated Least Squares QR solver

Erin Carson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-15
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-15.html

April 12, 2015

Acknowledgement

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0004938, DE-SC0003959, and DE-SC0010200; by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, X-Stack program under Award Numbers DE-SC0005136, DE-SC0008699, DE-SC0008700, and AC02-05CH11231; by DARPA Award Number HR0011-12-2-0016, as well as contributions from Intel, Oracle, and MathWorks.

AVOIDING COMMUNICATION IN THE LANCZOS BIDIAGONALIZATION ROUTINE AND ASSOCIATED LEAST SQUARES QR SOLVER

ERIN CARSON

Abstract. Communication – the movement of data between levels of memory hierarchy or between processors over a network – is the most expensive operation in terms of both time and energy at all scales of computing. Achieving scalable performance in terms of time and energy thus requires a dramatic shift in the field of algorithmic design. Solvers for sparse linear algebra problems, ubiquitous throughout scientific codes, are often the bottlenecks in application performance due to a low computation/communication ratio. In this paper we develop three potential implementations of communication-avoiding Lanczos bidiagonalization algorithms and discuss their different computational requirements. Based on these new algorithms, we also show how to obtain a communication-avoiding LSQR least squares solver.

1. Introduction. Classical implementations of Krylov methods, Lanczos bidiagonalization methods included, require one or more sparse matrix-vector multiplications (SpMVs) and one or more inner product operations in each iteration. These computational kernels are both communication-bound on modern computer architectures. To perform an SpMV, each processor must communicate entries of the source vector it owns to other processors in the parallel algorithm, and in the sequential algorithm the matrix A must be read from slow memory (when it is too large to fit in cache, the most interesting case). Inner products involve a global reduction (see [21, §11.4]) 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).

Thus, many efforts have focused on communication-avoiding Krylov subspace methods (CA-KSMs), or s-step Krylov methods, which can perform s iterations with a factor of $O(s)$ less communication than classical KSMs; see, e.g., [3, 4, 6, 8, 9, 11, 14, 15, 22, 23]. In practice, this can translate into significant speedups for many problems [18, 24]. In this paper, we will use the terminology 's-step methods', which was introduced in [5]. The reader should note this use of the term differs from other works, e.g., [7, 16] and [13, §9.2.7], in which the term 's-step methods' is used to refer to a type of restarted Lanczos procedure.

In this manuscript we present three approaches to developing communication-avoiding variants of Lanczos bidiagonalization. Each of the three approaches are used to give both communication-avoiding upper and lower bidiagonalization routines. The LSQR least squares solver of Paige and Saunders [19] is based on the Lanczos lower bidiagonalization method, and we use this to derive two potential CA-LSQR methods based on two of our approaches to communication-avoiding lower bidiagonalization.

The rest of this manuscript is organized as follows. Section 2 gives a brief overview of the communication-avoiding approach and motivates such methods from a performance perspective. In Section 3, we review the upper and lower Lanczos bidiagonalization procedures. In Section 4 we demonstrate three approaches to developing communication-avoiding versions of the algorithms in Section 3. Section 5 first reviews the LSQR method and then gives algorithms for two possible communication-avoiding variants. Section 6 briefly discusses future work, namely, convergence and performance studies to compare these new methods and determine guidelines for when one should be used over another.

2. Background on communication-avoiding Krylov methods.

The basic idea behind the CA-KSMs introduced by Hoemmen, Mohiyuddin, and others (see [15]), is to unroll the iteration loop by a factor of $s > 1$. One first builds bases for the Krylov subspaces known to contain the iteration vectors to be computed in the next s iterations, computes Gram matrices to store dot products between these basis vectors, and then subsequently performs s iterations, updating the coordinates of the iteration vectors in the precomputed bases rather than the iteration vectors themselves. For a thorough treatment of communication-avoiding Krylov methods, see [1].

This algorithmic change allows use of communication-avoiding kernels which can asymptotically reduce communication cost. The matrix powers kernel optimization fuses together a sequence of s SpMV operations into one kernel invocation. This kernel is used to compute the $O(s)$-dimensional Krylov bases, which are denoted by calligraphic letters in this and future sections. Depending on the nonzero structure of A (or whatever matrix we must compute a basis for), this enables communication-avoidance in both serial and parallel implementations, as described in paragraphs below. For an in-depth treatment of the matrix powers kernel implementation, see [10].

Serial. In serial, the matrix powers kernel reorganizes the $O(s)$ SpMVs to maximize reuse of A and the vector(s). This means ideally reading A and the starting vector only once and writing the s output vectors spanning the Krylov subspace only once. When the communication cost of reading A dominates that of reading/writing the vectors (a common situation), this results in an s-fold decrease in both latency (the number of messages sent) and bandwidth (the number of words moved).

Parallel. In a parallel implementation, the matrix powers kernel reorganizes the computation in a similar way but with a slightly different goal. In a parallel SpMV operation, only entries of the vectors need to be communicated. The parallel matrix powers kernel avoids interprocessor synchronization by initially storing some redundant elements of A and the starting vector on different processors and performing redundant computation to compute the s Krylov basis vectors without further synchronization in between SpMVs. Provided the additional bandwidth and latency cost to distribute the starting vector is a lower-order term (equivalently, A^s is *well partitioned*; see [10]) this gives an s-fold savings in latency cost.

Serial and parallel variants of the matrix powers kernel, for both structured and general sparse matrices, are described in [17] and [1], which summarize most of [10] and elaborate on the implementation in [18]. Within [17], we refer the reader to the complexity analysis in Tables 2.3-4, the performance modeling in §2.6, and the performance results in §2.10.3 and §2.11.3, which demonstrate that this optimization leads to speedups in practice.

For example, for a 2D five-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 of $1 + (s/2)\sqrt{p/n}$ [17]. Therefore since the additional arithmetic operations and additional words moved are lower order terms, we expect to see a $\Theta(s)$ speedup when latency is the dominant cost. We note that matrix powers kernel performance is sensitive to matrix structure and hardware parameters, making it a good candidate for inclusion in auto-tuning libraries and specializers.

Besides SpMV operations, classical KSMs also must compute inner products in each iteration, which incur a costly global synchronization on parallel computers. For Lanczos-based methods like the ones in this paper, inner products are computed as a block operation producing Gram matrices which are later used to compute dot products without additional communication. In parallel, this can lead to an s-fold decrease in latency.

These communication-avoiding variants can lead to speedups in practice. We direct the reader to recent performance results in [24], which demonstrate speedups up to 4.2 for a communication-avoiding BICGSTAB implementation with $s = 4$.

3. The bidiagonalization algorithm.

We first review classical algorithms for reduction of a matrix to both upper bidiagonal and lower bidiagonal form. The original procedure given by Golub and Kahan gives the procedure as a reduction to upper bidiagonal form [12]. With some slight modifications, Paige and Saunders [19] showed that a similar procedure could be used to produce a reduction to lower bidiagonal form, and that this formulation was more amenable to solving the full-rank least squares problem $\min \|Ax - b\|_2$. This observation forms the basis for the LSQR algorithm. Both methods are connected in that they both produce the same sequence of vectors V_k that would be produced by the symmetric Lanczos method applied to $A^T A$.

Let A be an m-by-n matrix and b be a length-m vector. After k iterations, the Lanczos upper bidiagonalization procedure produces the m-by-k matrix $P_k \equiv [p_1, p_2, \ldots, p_k]$ and the n-by-k matrix $V_k \equiv [v_1, v_2, \ldots, v_k]$ such that

$$\begin{aligned} V_k(\theta_1 e_1) &= A^T b \\ A V_k &= P_k R_k \\ A^T P_k &= V_k R_k^T + \theta_{k+1} v_{k+1} e_k^T, \end{aligned}$$

where

$$R_k = \begin{bmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \rho_{k-1} & \theta_k \\ & & & & \rho_k \end{bmatrix},$$

and in exact arithmetic, $P_k^T P_k = I$ and $V_k^T V_k = I$. The algorithm of Golub and Kahan for reduction to upper bidiagonal form is shown in Algorithm 1. Note that here and in the remainder of this paper bars over variables denote intermediate quantities which are yet to be normalized. We note that one can formulate the upper bidiagonalization algorithm as the Lanczos reduction to tridiagonal form. Letting

$$Z = [z_1, z_2, \ldots, z_{2k}] \equiv \begin{bmatrix} 0 & p_1 & 0 & p_2 & \ldots & 0 & p_k \\ v_1 & 0 & v_2 & 0 & \ldots & v_k & 0 \end{bmatrix},$$

$$\tilde{A} \equiv \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \quad \text{and} \quad \tilde{T} \equiv \begin{bmatrix} 0 & \rho_1 & & & & & \\ \rho_1 & 0 & \theta_2 & & & & \\ & \theta_2 & 0 & \rho_2 & & & \\ & & \rho_2 & 0 & \ddots & & \\ & & & \ddots & \ddots & \theta_k & \\ & & & & \theta_k & 0 & \rho_k \\ & & & & & \rho_k & 0 \end{bmatrix},$$

the procedure in Algorithm 1 is mathematically equivalent to

$$\tilde{A}Z = Z\tilde{T} + \theta_{k+1} z_{2k+1} e_{2k}^T,$$

with $Z^H Z = I_{2k}$ and $Z^H z_{2k+1} = 0$. This means that, in exact arithmetic, k steps of the upper bidiagonalization procedure applied to A with starting vector v_1 produces the same information as $2k$ steps of symmetric Lanczos applied to cyclic matrix $\tilde{A}$ with starting vector z_1 as defined above.

Algorithm 1 Lanczos reduction to upper bidiagonal form

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ vector $b$
1: $\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
2: for $i = 1, 2, \ldots$ until convergence do
3:     $\bar{v}_{i+1} = A^T p_i - \rho_i v_i$
4:     $\theta_{i+1} = \|\bar{v}_{i+1}\|_2$
5:     $v_{i+1} = \bar{v}_{i+1}/\theta_{i+1}$
6:     $\bar{p}_{i+1} = Av_{i+1} - \theta_{i+1} p_i$
7:     $\rho_{i+1} = \|\bar{p}_{i+1}\|_2$
8:     $p_{i+1} = \bar{p}_{i+1}/\rho_{i+1}$
9: end for
```

We will also consider reduction to *lower* bidiagonal form for the purpose of easy connection to the LSQR method of Paige and Saunders [19]. Again, A is an m-by-n matrix and b is a length-m vector. After k iterations, the Lanczos lower bidiagonalization procedure produces the m-by-$(k+1)$ matrix $U_{k+1} \equiv [u_1, u_2, \ldots, u_{k+1}]$ and the n-by-k matrix $V_k \equiv [v_1, v_2, \ldots, v_k]$ such that

$$\begin{aligned} U_{k+1}(\beta_1 e_1) &= b \\ AV_k &= U_{k+1} B_k \\ A^T U_{k+1} &= V_k B_k^T + \alpha_{k+1} u_{k+1} e_{k+1}^T, \end{aligned}$$

where

$$B_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix},$$

and in exact arithmetic, $U_{k+1}^T U_{k+1} = I$ and $V_k^T V_k = I$.

Again in this case, we can formulate the lower bidiagonalization algorithm as the Lanczos reduction to tridiagonal form. Here we define

$$Z = [z_1, z_2, \ldots, z_{2k}] \equiv \begin{bmatrix} 0 & v_1 & 0 & v_2 & \ldots & 0 & v_k \\ u_1 & 0 & u_2 & 0 & \ldots & u_k & 0 \end{bmatrix},$$

$$\tilde{A} \equiv \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}, \quad \text{and} \quad \tilde{T} \equiv \begin{bmatrix} 0 & \alpha_1 & & & & & \\ \alpha_1 & 0 & \beta_2 & & & & \\ & \beta_2 & 0 & \alpha_2 & & & \\ & & \alpha_2 & 0 & \ddots & & \\ & & & \ddots & \ddots & \beta_k & \\ & & & & \beta_k & 0 & \alpha_k \\ & & & & & \alpha_k & 0 \end{bmatrix},$$

and then Algorithm 2 is mathematically equivalent to

$$\tilde{A}Z = Z\tilde{T} + \beta_{k+1} z_{2k+1} e_{2k}^T,$$

with $Z^H Z = I_{2k}$ and $Z^H z_{2k+1} = 0$. Then in exact arithmetic, k steps of the lower bidiagonalization procedure applied to A with starting vector u_1 produces the same information as $2k$ steps of symmetric Lanczos applied to cyclic matrix $\tilde{A}$ with starting vector z_1 as defined above.

Algorithm 2 Lanczos reduction to lower bidiagonal form

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
2: for $i = 1, 2, \ldots$ until convergence do
3:     $\bar{u}_{i+1} = Av_i - \alpha_i u_i$
4:     $\beta_{i+1} = \|\bar{u}_{i+1}\|_2$
5:     $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
6:     $\bar{v}_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$
7:     $\alpha_{i+1} = \|\bar{v}_{i+1}\|_2$
8:     $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
9: end for
```

4. Communication-avoiding Lanczos bidiagonalization.

There are at least three ways to derive communication-avoiding variants of Algorithms 1 and 2, each with associated pros and cons. The correct method to choose will depend on the structure and conditioning of the matrix, the requirements of the particular application, and machine-specific parameters such as cache size and relative latency/bandwidth cost. We describe the three potential communication-avoiding variants in subsections below.

4.1. Equivalent form of CA-Lanczos.

As discussed in Section 3, k steps of either bidiagonalization procedure in Algorithm 1 or 2 will produce the same information as $2k$ steps of symmetric Lanczos applied to the appropriately defined cyclic matrix $\tilde{A}$ and appropriately chosen starting vector z_1. Therefore one can simply use an existing version of CA-Lanczos (available in, e.g., [15, 1, 2]) run on input $\tilde{A}$ and z_1, and recover the bidiagonalization matrices, either P_k, V_k, and R_k for upper bidiagonalization, or U_{k+1}, V_k, and B_k for lower bidiagonalization, from Z and $\tilde{T}$.

This method is simple and allows us to use an existing communication-avoiding method. The drawback is that the system is now twice the size, and extra work and storage will be required unless the Lanczos method is modified to optimize for the block non-zero structure of the matrix/vectors.

4.2. Forming Krylov bases.

By introducing auxiliary quantities, yet another version can be derived that works by building s-step Krylov bases with AA^T and

A^TA. The benefit here is that other polynomial bases can be used in order to improve numerical properties (e.g., Newton or Chebyshev). The drawbacks are that this requires computing bases with powers of AA^T and A^TA, which squares the condition number of A. Also, in order to satisfy the recurrences, we need $4s+1$ basis vectors in each iteration, which doubles the number of SpMVs per s iterations versus the classical method (this is assuming we form and store A^TA and AA^T offline; otherwise the number of SpMVs required is $8s+2$). We note that this is equivalent (in exact arithmetic) to the method described in the previous subsection, but takes nonzero blocks into account and uses auxiliary quantities.

We derive this method below for both upper and lower bidiagonalization procedures. Note that in communication-avoiding algorithms, we will switch from indexing iterations by i to indexing iterations by $sk+j$, where s is the iteration blocking parameter, k is the outer iteration index, and j in the inner iteration index.

4.2.1. Reduction to upper bidiagonal form. Assume we are beginning iteration $sk+1$ of Algorithm 1, where $k \in \mathbb{N}$ and $0 < s \in \mathbb{N}$, so that v_{sk+1} and p_{sk+1} have just been computed. Recall that

$$\begin{aligned} p_{sk+j+1} &\in \mathcal{K}_{s+1}(AA^T, p_{sk+1}) + \mathcal{K}_s(AA^T, Av_{sk+1}) \quad \text{and} \\ v_{sk+j+1} &\in \mathcal{K}_s(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tp_{sk+1}), \end{aligned}$$

for $j \in \{0, \dots, s\}$.

We define basis matrices whose columns span these subspaces as follows. Let $\mathcal{V}_k$ be a basis for $\mathcal{K}_s(A^TA, v_{sk+1})$, $\tilde{\mathcal{V}}_k$ a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{P}_k$ a basis for $\mathcal{K}_{s+1}(AA^T, p_{sk+1})$ and $\tilde{\mathcal{P}}_k$ a basis for $\mathcal{K}_s(A^TA, A^Tp_{sk+1})$. Assuming these polynomial bases are generated using a three-term recurrence, we can write the recurrence relations

$$\begin{aligned} (AA^T)[\underline{\mathcal{P}}_k, 0, \underline{\tilde{\mathcal{V}}}_k, 0] &= [\mathcal{P}_k, \tilde{\mathcal{V}}_k] \begin{bmatrix} [T_k^{(\mathcal{P})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{V}})}, 0] \end{bmatrix} \quad \text{and} \\ (A^TA)[\underline{\mathcal{V}}_k, 0, \underline{\tilde{\mathcal{P}}}_k, 0] &= [\mathcal{V}_k, \tilde{\mathcal{P}}_k] \begin{bmatrix} [T_k^{(\mathcal{V})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{P}})}, 0] \end{bmatrix}, \end{aligned}$$

where $\underline{\mathcal{P}}_k$, $\underline{\tilde{\mathcal{V}}}_k$, $\underline{\mathcal{V}}_k$, and $\underline{\tilde{\mathcal{P}}}_k$ are the same as $\mathcal{P}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{V}_k$, and $\tilde{\mathcal{P}}_k$, resp., but with the last column removed, and the T_k matrices are tridiagonal matrices of the form

$$\begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & \\ \hat{\gamma}_1 & \ddots & \ddots & \\ & \ddots & \ddots & \hat{\beta}_{i-1} \\ & & \ddots & \hat{\alpha}_i \\ & & & \hat{\gamma}_i \end{bmatrix}, \tag{4.1}$$

where above $i = s$ for $T_k^{(\mathcal{P})}$ and $i = s-1$ for $T_k^{(\mathcal{V})}$, $T_k^{(\tilde{\mathcal{V}})}$, and $T_k^{(\tilde{\mathcal{P}})}$. Note that the entries $\hat{\alpha}_j$, $\hat{\gamma}_j$, and $\hat{\beta}_j$ can be set differently depending on whether we are constructing polynomials in AA^T or A^TA. Also note that these recurrence coefficients could be refined with each new outer loop. See Philippe and Reichel [20] for guidelines on setting these entries such that the basis condition number is improved.

To simplify notation, we will define $\mathcal{Y}_k \equiv [\mathcal{P}_k, \tilde{\mathcal{V}}_k]$, $\underline{\mathcal{Y}}_k \equiv [\underline{\mathcal{P}}_k, 0, \underline{\tilde{\mathcal{V}}}_k, 0]$, $\mathcal{Z}_k \equiv [\mathcal{V}_k, \tilde{\mathcal{P}}_k]$, $\underline{\mathcal{Z}}_k \equiv [\underline{\mathcal{V}}_k, 0, \underline{\tilde{\mathcal{P}}}_k, 0]$, and

$$T_k^{(\mathcal{Y})} \equiv \begin{bmatrix} [T_k^{(\mathcal{P})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{V}})}, 0] \end{bmatrix}, \qquad T_k^{(\mathcal{Z})} \equiv \begin{bmatrix} [T_k^{(\mathcal{V})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{P}})}, 0] \end{bmatrix}.$$

This lets us rewrite the recurrences as

$$(AA^T)\underline{\mathcal{Y}}_k = \mathcal{Y}_k T_k^{(\mathcal{Y})} \quad \text{and} \quad (A^TA)\underline{\mathcal{Z}}_k = \mathcal{Z}_k T_k^{(\mathcal{Z})}.$$

The recurrences do not give us a way to represent multiplication by A and A^T in these new bases, which are necessary to perform updates to the coordinate vectors v'_{j+1} and p'_{j+1}. The recurrences do however give ways to multiply by AA^T and A^TA, and we introduce auxiliary quantities to make use of this. Let

$$\begin{aligned} \tilde{p}_{sk+j+1} \equiv A^T p_{sk+j+1} &= A^T(Av_{sk+j+1} - \theta_{sk+j+1}p_{sk+j})/\rho_{sk+j+1} \\ &= \big((A^TA)v_{sk+j+1} - \theta_{sk+j+1}\tilde{p}_{sk+j}\big)/\rho_{sk+j+1}, \quad \text{and} \\ \tilde{v}_{sk+j+1} \equiv Av_{sk+j+1} &= A(A^Tp_{sk+j} - \rho_{sk+j}v_{sk+j})/\theta_{sk+j+1} \\ &= \big((AA^T)p_{sk+j} - \rho_{sk+j}\tilde{v}_{sk+j}\big)/\theta_{sk+j+1}. \end{aligned}$$

Then vector updates can then be written

$$\begin{aligned} \bar{v}_{sk+j+1} &= \tilde{p}_{sk+j} - \rho_{sk+j}v_{sk+j}, \\ \tilde{v}_{sk+j+1} &= \big((AA^T)p_{sk+j} - \rho_{sk+j}\tilde{v}_{sk+j}\big)/\theta_{sk+j+1}, \quad \text{and} \\ \bar{p}_{sk+j+1} &= \tilde{v}_{sk+j+1} - \theta_{sk+j+1}p_{sk+j} \end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{p}_{sk+j+1} = \big((A^TA)v_{sk+j+1} - \theta_{sk+j+1}\tilde{p}_{sk+j}\big)/\rho_{sk+j+1}$$

for $j \in \{1, \ldots, s-1\}$ ($\tilde{p}_{sk+s+1}$ is not needed). As before, $v_{sk+j+1} = \bar{v}_{sk+j+1}/\theta_{sk+j+1}$ and $p_{sk+j+1} = \bar{p}_{sk+j+1}/\rho_{sk+j+1}$. Note that $\tilde{v}_{sk+j+1} \in \mathcal{Y}_k$ for $j \in \{1, \ldots, s\}$ and $\tilde{p}_{sk+j+1} \in \mathcal{Z}_k$ for $j \in \{1, \ldots, s-1\}$, so no additional basis vectors are required to represent updates to these auxiliary quantities. The classical version of this modified upper bidiagonalization algorithm is given in Alg. 3.

We can then represent v_{sk+j+1}, $\tilde{v}_{sk+j+1}$, p_{sk+j+1}, and $\tilde{p}_{sk+j+1}$ by their coordinates v'_{j+1}, $\tilde{v}_{j+1}$, p_{j+1}, and $\tilde{p}_{j+1}$, resp., in $\mathcal{Y}_k$ and $\mathcal{Z}_k$, i.e.,

$$\begin{aligned} v_{sk+j+1} &= \mathcal{Z}_k v'_{j+1}, \\ \tilde{v}_{sk+j+1} &= \mathcal{Y}_k \tilde{v}'_{j+1}, \quad \text{and} \\ p_{sk+j+1} &= \mathcal{Y}_k p'_{j+1}, \quad \text{for } j \in \{1, \ldots, s\}, \quad \text{and} \\ \tilde{p}_{sk+j+1} &= \mathcal{Z}_k \tilde{p}'_{j+1} \quad \text{for } j \in \{1, \ldots, s-1\}. \end{aligned} \tag{4.2}$$

Note that using (4.2), in each new outer loop we initialize the coordinate vectors to $p'_1 = e_1$, $v'_1 = e_1$, $\tilde{p}'_1 = e_{s+1}$, and $\tilde{v}'_1 = e_{s+2}$, and update them in each iteration by the

Algorithm 3 Lanczos upper bidiagonalization with auxiliary quantities

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ vector $b$
1: $\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
2: $\tilde{v}_1 = Av_1$, $\tilde{p}_1 = A^T p_1$
3: for $i = 1, 2, \ldots$ until convergence do
4:     $\bar{v}_{i+1} = \tilde{p}_i - \rho_i v_i$
5:     $\theta_{i+1} = \|\bar{v}_{i+1}\|_2$
6:     $v_{i+1} = \bar{v}_{i+1}/\theta_{i+1}$
7:     $\tilde{v}_{i+1} = (AA^T p_i - \rho_i \tilde{v}_i)/\theta_{i+1}$
8:     $\bar{p}_{i+1} = \tilde{v}_{i+1} - \theta_{i+1} p_i$
9:     $\rho_{i+1} = \|\bar{p}_{i+1}\|_2$
10:    $p_{i+1} = \bar{p}_{i+1}/\rho_{i+1}$
11:    $\tilde{p}_{i+1} = (A^T A v_{i+1} - \theta_{i+1}\tilde{p}_i)/\rho_{i+1}$
12: end for
```

formulas

$$\begin{aligned}
\bar{v}'_{j+1} &= \tilde{p}'_j - \rho_{sk+j} v'_j, \\
v'_{j+1} &= \bar{v}'_{j+1}/\theta_{sk+j+1}, \\
\tilde{v}'_{j+1} &= \big(T_k^{(\mathcal{Y})} p'_j - \rho_{sk+j}\tilde{v}'_j\big)/\theta_{sk+j+1}, \\
\bar{p}'_{j+1} &= \tilde{v}'_{j+1} - \theta_{sk+j+1} p'_j, \quad \text{and} \\
p'_{j+1} &= \bar{p}'_{j+1}/\rho_{sk+j+1},
\end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{p}'_{j+1} = \big(T_k^{(\mathcal{Z})} v'_{j+1} - \theta_{sk+j+1}\tilde{p}'_j\big)/\rho_{sk+j+1},$$

for $j \in \{1, \ldots, s-1\}$.

Now, it remains to determine how to compute the inner products θ_{sk+j+1} and ρ_{sk+j+1}. We can write

$$\begin{aligned}
\theta_{sk+j+1} &= (\bar{v}^T_{sk+j+1}\bar{v}_{sk+j+1})^{1/2} \\
&= ((\mathcal{Z}_k \bar{v}'_{j+1})^T(\mathcal{Z}_k \bar{v}'_{j+1}))^{1/2} \\
&= (\bar{v}'^T_{j+1}\mathcal{Z}_k^T\mathcal{Z}_k\bar{v}'_{j+1})^{1/2} \qquad (4.3)
\end{aligned}$$

and

$$\begin{aligned}
\rho_{sk+j+1} &= (\bar{p}^T_{sk+j+1}\bar{p}_{sk+j+1})^{1/2} \\
&= ((\mathcal{Y}_k \bar{p}'_{j+1})^T(\mathcal{Y}_k \bar{p}'_{j+1}))^{1/2} \\
&= (\bar{p}'^T_{j+1}\mathcal{Y}_k^T\mathcal{Y}_k\bar{p}'_{j+1})^{1/2}. \qquad (4.4)
\end{aligned}$$

Defining the Gram matrices

$$G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T\mathcal{Y}_k \qquad \text{and} \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T\mathcal{Z}_k,$$

we can compute (4.3) and (4.4) by the formulas

$$\theta_{sk+j+1} = (\bar{v}'^T_{j+1}G_k^{(\mathcal{Z})}\bar{v}'_{j+1})^{1/2} \quad \text{and} \quad \rho_{sk+j+1} = (\bar{p}'^T_{j+1}G_k^{(\mathcal{Y})}\bar{p}'_{j+1})^{1/2}.$$

The resulting communication-avoiding version of Algorithm 3 is shown in Algorithm 4. Note that in lines 19 and 20 of Algorithm 3, we have shown how to recover all vectors that would be computed in the s iterations. For correctness of the algorithm as shown, only the vectors for the most recent iteration need be recovered for use in the next outer loop.

Algorithm 4 Communication-avoiding Lanczos upper bidiagonalization with auxiliary quantities

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ vector $b$
 1: $\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
 2: $\tilde{v}_1 = Av_1$, $\tilde{p}_1 = A^T p_1$
 3: for $k = 0, 1, \dots$ until convergence do
 4:     Compute $\mathcal{V}_k$, a basis for $\mathcal{K}_s(A^TA, v_{sk+1})$, $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{P}_k$, a basis for $\mathcal{K}_{s+1}(AA^T, p_{sk+1})$, and $\tilde{\mathcal{P}}_k$, a basis for $\mathcal{K}_s(A^TA, A^Tp_{sk+1})$. Let $\mathcal{Y}_k = [\mathcal{P}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{P}}_k]$.
 5:     $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$
 6:     $v_1' = e_1$, $p_1' = e_1$, $\tilde{v}_1' = e_{s+2}$, $\tilde{p}_1' = e_{s+1}$.
 7:     for $j = 1, \dots, s$ do
 8:         $\bar{v}_{j+1}' = \tilde{p}_j' - \rho_{sk+j} v_j'$
 9:         $\theta_{sk+j+1} = \left(\bar{v}_{j+1}'^T G_k^{(\mathcal{Z})} \bar{v}_{j+1}'\right)^{1/2}$
10:         $v_{j+1}' = \bar{v}_{j+1}'/\theta_{sk+j+1}$
11:         $\tilde{v}_{j+1}' = (T_k^{(\mathcal{Y})} p_j' - \rho_{sk+j}\tilde{v}_j')/\theta_{sk+j+1}$
12:         $\bar{p}_{j+1}' = \tilde{v}_{j+1}' - \theta_{sk+j+1} p_j'$
13:         $\rho_{sk+j+1} = \left(\bar{p}_{j+1}'^T G_k^{(\mathcal{Y})} \bar{p}_{j+1}'\right)^{1/2}$
14:         $p_{j+1}' = \bar{p}_{j+1}'/\rho_{sk+j+1}$
15:         if $j < s$ then
16:             $\tilde{p}_{j+1}' = (T_k^{(\mathcal{Z})} v_{j+1}' - \theta_{sk+j+1}\tilde{p}_j')/\rho_{sk+j+1}$
17:         end if
18:     end for
19:     $[v_{sk+2}, \dots, v_{sk+s+1}] = \mathcal{Z}_k[v_2', \dots, v_{s+1}']$
20:     $[p_{sk+2}, \dots, p_{sk+s+1}] = \mathcal{Y}_k[p_2', \dots, p_{s+1}']$
21: end for
```

4.2.2. Reduction to lower bidiagonal form. Now assume we are beginning iteration $sk+1$ of Algorithm 2, where $k \in \mathbb{N}$ and $0 < s \in \mathbb{N}$, so that u_{sk+1} and v_{sk+1} have just been computed. Recall that

$$\begin{aligned} u_{sk+j+1} &\in \mathcal{K}_s(AA^T, Av_{sk+1}) + \mathcal{K}_s(AA^T, u_{sk+1}) \quad \text{and} \\ v_{sk+j+1} &\in \mathcal{K}_{s+1}(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tu_{sk+1}), \end{aligned}$$

for $j \in \{0, \dots, s\}$.

We then define basis matrices whose columns span the desired subspaces as follows. Let $\mathcal{U}_k$ be a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{V}}_k$ a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{V}_k$ a basis for $\mathcal{K}_{s+1}(A^TA, v_{sk+1})$ and $\tilde{\mathcal{U}}_k$ a basis for $\mathcal{K}_s(A^TA, A^Tu_{sk+1})$. Assuming these polynomial bases are generated using a three-term recurrence, we can write the recurrence relations

$$\begin{aligned}(AA^T)[\underline{\mathcal{U}}_k, 0, \underline{\tilde{\mathcal{V}}}_k, 0] &= [\mathcal{U}_k, \tilde{\mathcal{V}}_k] \begin{bmatrix} [T_k^{(\mathcal{U})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{V}})}, 0] \end{bmatrix} \quad \text{and} \\ (A^TA)[\underline{\mathcal{V}}_k, 0, \underline{\tilde{\mathcal{U}}}_k, 0] &= [\mathcal{V}_k, \tilde{\mathcal{U}}_k] \begin{bmatrix} [T_k^{(\mathcal{V})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{U}})}, 0] \end{bmatrix},\end{aligned}$$

where $\underline{\mathcal{U}}_k$, $\underline{\tilde{\mathcal{V}}}_k$, $\underline{\mathcal{V}}_k$, and $\underline{\tilde{\mathcal{U}}}_k$ are the same as $\mathcal{U}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{V}_k$, and $\tilde{\mathcal{U}}_k$, resp., but with the last column removed, and the T_k matrices are tridiagonal matrices of the form given in (4.1) with $i = s$ for $T_k^{(\mathcal{V})}$ and $i = s-1$ for $T_k^{(\mathcal{U})}$, $T_k^{(\tilde{\mathcal{V}})}$, and $T_k^{(\tilde{\mathcal{U}})}$. As before the entries $\hat{\alpha}_j$, $\hat{\gamma}_j$, and $\hat{\beta}_j$ can be different depending on whether we are constructing polynomials in AA^T or A^TA and could be refined with each new outer loop.

To simplify notation, we will define $\mathcal{Y}_k \equiv [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\underline{\mathcal{Y}}_k \equiv [\underline{\mathcal{U}}_k, 0, \underline{\tilde{\mathcal{V}}}_k, 0]$, $\mathcal{Z}_k \equiv [\mathcal{V}_k, \tilde{\mathcal{U}}_Z]$, $\underline{\mathcal{Z}}_k \equiv [\underline{\mathcal{V}}_k, 0, \underline{\tilde{\mathcal{U}}}_k, 0]$, and

$$T_k^{(\mathcal{Y})} \equiv \begin{bmatrix} [T_k^{(\mathcal{U})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{V}})}, 0] \end{bmatrix}, \qquad T_k^{(\mathcal{Z})} \equiv \begin{bmatrix} [T_k^{(\mathcal{V})}, 0] & 0 \\ 0 & [T_k^{(\tilde{\mathcal{U}})}, 0] \end{bmatrix}.$$

This lets us rewrite the recurrences as

$$\begin{aligned}(AA^T)\underline{\mathcal{Y}}_k &= \mathcal{Y}_k T_k^{(\mathcal{Y})} \quad \text{and} \\ (A^TA)\underline{\mathcal{Z}}_k &= \mathcal{Z}_k T_k^{(\mathcal{Z})}.\end{aligned}$$

Note that these are the same recurrences used in the communication-avoiding upper bidiagonalization method of the previous subsection, but with different definitions of $\mathcal{Y}_k$, $\mathcal{Z}_k$, $\tilde{\mathcal{Y}}_k$, and $\tilde{\mathcal{Z}}_k$. Again, we introduce auxiliary quantities. Let

$$\begin{aligned}\tilde{u}_{sk+j+1} \equiv A^T u_{sk+j+1} &= A^T(Av_{sk+j} - \alpha_{sk+j}u_{sk+j})/\beta_{sk+j+1} \\ &= \big((A^TA)v_{sk+j} - \alpha_{sk+j}\tilde{u}_{sk+j}\big)/\beta_{sk+j+1}, \quad \text{and} \\ \tilde{v}_{sk+j+1} \equiv Av_{sk+j+1} &= A(A^T u_{sk+j+1} - \beta_{sk+j+1}v_{sk+j})/\alpha_{sk+j+1} \\ &= \big((AA^T)u_{sk+j+1} - \beta_{sk+j+1}\tilde{v}_{sk+j}\big)/\alpha_{sk+j+1}.\end{aligned}$$

Then the vector updates become

$$\begin{aligned}\bar{u}_{sk+j+1} &= (\tilde{v}_{sk+j} - \alpha_{sk+j}u_{sk+j}), \\ \tilde{u}_{sk+j+1} &= \big((A^TA)v_{sk+j} - \alpha_{sk+j}\tilde{u}_{sk+j}\big)/\beta_{sk+j+1}, \quad \text{and} \\ \bar{v}_{sk+j+1} &= (\tilde{u}_{sk+j} - \beta_{sk+j+1}v_{sk+j}),\end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{v}_{sk+j+1} = \big((AA^T)u_{sk+j+1} - \beta_{sk+j+1}\tilde{v}_{sk+j}\big)/\alpha_{sk+j+1},$$

for $j \in \{1, \ldots, s-1\}$. As before, $u_{sk+j+1} = \bar{u}_{sk+j+1}/\beta_{sk+j+1}$ and $v_{sk+j+1} = \bar{v}_{sk+j+1}/\alpha_{sk+j+1}$. The classical version of this modified lower bidiagonalization algorithm is given in Algorithm 5.

Note that in Algorithm 5, $\tilde{u}_{sk+j+1} \in \mathcal{Z}_k$ for $j \in \{0, \ldots, s\}$, and $\tilde{v}_{sk+j+1} \in \mathcal{Y}_k$ for $j \in \{0, \ldots, s-1\}$. Then we can represent u_{sk+j+1}, $\tilde{u}_{sk+j+1}$, v_{sk+j+1}, and $\tilde{v}_{sk+j+1}$ by

Algorithm 5 Lanczos lower bidiagonalization with auxiliary quantities

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
2: $\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = Av_1$
3: for $i = 1, 2, \ldots$ until convergence do
4:     $\bar{u}_{i+1} = \tilde{v}_i - \alpha_i u_i$
5:     $\beta_{i+1} = \|\bar{u}_{i+1}\|_2$
6:     $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
7:     $\tilde{u}_{i+1} = (A^T A v_i - \alpha_i \tilde{u}_i)/\beta_{i+1}$
8:     $\bar{v}_{i+1} = \tilde{u}_{i+1} - \beta_{i+1} v_i$
9:     $\alpha_{i+1} = \|\bar{v}_{i+1}\|_2$
10:    $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
11:    $\tilde{v}_{i+1} = (AA^T u_{i+1} - \beta_{i+1}\tilde{v}_i)/\alpha_{i+1}$
12: end for
```

their coordinates u'_{j+1}, $\tilde{u}_{j+1}$, v_{j+1}, and $\tilde{v}_{j+1}$, resp., in $\mathcal{Y}_k$ and $\mathcal{Z}_k$, i.e.,

$$\begin{aligned} u_{sk+j+1} &= \mathcal{Y}_k u'_{j+1}, \\ \tilde{u}_{sk+j+1} &= \mathcal{Z}_k \tilde{u}'_{j+1}, \quad \text{and} \\ v_{sk+j+1} &= \mathcal{Z}_k v'_{j+1} \quad \text{for } j \in \{1, \ldots, s\}, \quad \text{and} \\ \tilde{v}_{sk+j+1} &= \mathcal{Y}_k \tilde{u}'_{j+1} \quad \text{for } j \in \{1, \ldots, s-1\}. \end{aligned} \tag{4.5}$$

Note that using (4.5), in each new outer loop we initialize the coordinate vectors to $u'_1 = e_1$, $v'_1 = e_1$, $\tilde{u}'_1 = e_{s+2}$, and $\tilde{v}'_1 = e_{s+1}$, and update them in each iteration by the formulas

$$\begin{aligned} \bar{u}'_{j+1} &= \tilde{v}'_j - \alpha_{sk+j} u'_j, \\ u'_{j+1} &= \bar{u}'_{j+1}/\beta_{sk+j+1}, \\ \tilde{u}'_{j+1} &= \big(T_k^{(\mathcal{Z})} v'_j - \alpha_{sk+j}\tilde{u}'_j\big)/\beta_{sk+j+1}, \\ \bar{v}'_{j+1} &= \tilde{u}'_{j+1} - \beta_{sk+j+1} v'_j, \quad \text{and} \\ v'_{j+1} &= \bar{v}'_{j+1}/\alpha_{sk+j+1}, \end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{v}'_{j+1} = \big(T_k^{(\mathcal{Y})} u'_{j+1} - \beta_{sk+j+1}\tilde{v}'_j\big)/\alpha_{sk+j+1},$$

for $j \in \{1, \ldots, s-1\}$.

Now, it only remains to determine how to compute the inner products β_{sk+j+1} and α_{sk+j+1}. We can write

$$\begin{aligned} \beta_{sk+j+1} &= (\bar{u}^T_{sk+j+1}\bar{u}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Y}_k \bar{u}'_{j+1})^T(\mathcal{Y}_k \bar{u}'_{j+1}))^{1/2} \\ &= (\bar{u}'^T_{j+1}\mathcal{Y}_k^T \mathcal{Y}_k \bar{u}'_{j+1})^{1/2} \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} \alpha_{sk+j+1} &= (\bar{v}^T_{sk+j+1}\bar{v}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Z}_k \bar{v}'_{j+1})^T(\mathcal{Z}_k \bar{v}'_{j+1}))^{1/2} \\ &= (\bar{v}'^T_{j+1}\mathcal{Z}_k^T \mathcal{Z}_k \bar{v}'_{j+1})^{1/2}. \end{aligned} \tag{4.7}$$

Defining the Gram matrices

$$G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad \text{and} \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k,$$

equations (4.6) and (4.7) become

$$\beta_{sk+j+1} = (\bar{u}_{j+1}'^T G_k^{(\mathcal{Y})} \bar{u}_{j+1}')^{1/2} \quad \text{and} \quad \alpha_{sk+j+1} = (\bar{v}_{j+1}'^T G_k^{(\mathcal{Z})} \bar{v}_{j+1}')^{1/2}.$$

The resulting communication-avoiding version of Algorithm 5 is shown in Algorithm 6. Again note that in lines 19 and 20 of Algorithm 5, we show recovery of all vectors that would be computed in the s iterations, although only the vectors from the most recent iteration need be recovered for correctness.

Algorithm 6 Communication-avoiding Lanczos lower bidiagonalization with auxiliary quantities

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
 1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
 2: $\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = A v_1$
 3: for $k = 0, 1, \dots$ until convergence do
 4:     Compute $\mathcal{U}_k$, a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$,
        $\mathcal{V}_k$, a basis for $\mathcal{K}_{s+1}(A^T A, v_{sk+1})$, and $\tilde{\mathcal{U}}_k$, a basis for $\mathcal{K}_s(A^T A, A^T u_{sk+1})$. Let
        $\mathcal{Y}_k = [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{U}}_k]$.
 5:     $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$    $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$
 6:     $u_1' = e_1$, $v_1' = e_1$, $\tilde{u}_1' = e_{s+2}$, $\tilde{v}_1' = e_{s+1}$.
 7:     for $j = 1, \dots, s$ do
 8:         $\bar{u}_{j+1}' = \tilde{v}_j' - \alpha_{sk+j} u_j'$
 9:         $\beta_{sk+j+1} = \left(\bar{u}_{j+1}'^T G_k^{(\mathcal{Y})} \bar{u}_{j+1}'\right)^{1/2}$
10:         $u_{j+1}' = \bar{u}_{j+1}' / \beta_{sk+j+1}$
11:         $\tilde{u}_{j+1}' = (T_k^{(\mathcal{Z})} v_j' - \alpha_{sk+j} \tilde{u}_j') / \beta_{sk+j+1}$
12:         $\bar{v}_{j+1}' = \tilde{u}_{j+1}' - \beta_{sk+j+1} v_j'$
13:         $\alpha_{sk+j+1} = \left(\bar{v}_{j+1}'^T G_k^{(\mathcal{Z})} \bar{v}_{j+1}'\right)^{1/2}$
14:         $v_{j+1}' = \bar{v}_{j+1}' / \alpha_{sk+j+1}$
15:         if $j < s$ then
16:             $\tilde{v}_{j+1}' = (T_k^{(\mathcal{Y})} u_{j+1}' - \beta_{sk+j+1} \tilde{v}_j') / \alpha_{sk+j+1}$
17:         end if
18:     end for
19:     $[u_{sk+2}, \dots, p_{sk+s+1}] = \mathcal{Y}_k [u_2', \dots, u_{s+1}']$
20:     $[v_{sk+2}, \dots, v_{sk+s+1}] = \mathcal{Z}_k [v_2', \dots, v_{s+1}']$
21: end for
```

4.3. Alternating matrix powers. Another communication-avoiding variant can be derived which builds two coupled Krylov bases, where basis vectors are computed by alternating between multiplication by A and by A^T. We still need to obtain $4s+1$ basis vectors in order to take s steps of the algorithm, but in this case we do not need to construct or multiply by $A^T A$ and AA^T. It is less clear how to choose polynomial basis parameters (entries of T_k) in this case. Numerical and performance comparisons between these versions and the communication-avoiding versions discussed in Section 4.2 remains future work. Note, as before, in both algorithms derived below we show recovery of all iteration vectors after each inner loop, although only the last vectors are needed to begin the next outer loop.

4.3.1. Reduction to upper bidiagonal form. Another version can be derived by using coupled recurrences to generate bases for v_{sk+j+1} and p_{sk+j+1}. Recall that for $j \in \{0, \dots, s\}$,

$$\begin{aligned} p_{sk+j+1} &\in \mathcal{K}_{s+1}(AA^T, p_{sk+1}) + \mathcal{K}_s(AA^T, Av_{sk+1}) \quad \text{and} \\ v_{sk+j+1} &\in \mathcal{K}_s(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tp_{sk+1}). \end{aligned}$$

Then assume that we have a $2s$ dimensional basis $\mathcal{Z}_k$ such that $v_{sk+j+1} = \mathcal{Z}_k v'_{j+1}$ and a $2s+1$ dimensional basis $\mathcal{Y}_k$ such that $p_{sk+j+1} = \mathcal{Y}_k p'_{j+1}$ for $j \in \{1, \dots, s\}$, and that these bases satisfy the recurrences

$$A\underline{\mathcal{Z}}_k = \mathcal{Y}_k T_k \quad \text{and} \quad A^T \underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{T}_k,$$

where $\underline{\mathcal{Z}}_k$ and $\underline{\mathcal{Y}}_k$ are the same as $\mathcal{Z}_k$ and $\mathcal{Y}_k$, resp., but with the last column removed. The matrices T_k and $\tilde{T}_k$ are tridiagonal matrices of the form given in (4.1). Given $z_1 \equiv v_{sk+1}$ and $y_1 \equiv p_{sk+1}$, the columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$ can be generated by, e.g., computing

$$\begin{aligned} y_2 &= (Az_1 - \hat{\alpha}_1 y_1)/\hat{\gamma}_1, \\ z_2 &= (A^T y_1 - \hat{\alpha}_1 z_1)/\hat{\gamma}_1, \quad \text{and} \\ y_{\ell+1} &= (Az_\ell - \hat{\alpha}_\ell y_\ell - \hat{\beta}_{\ell-1} y_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \dots, 2s+1\}, \\ z_{\ell+1} &= (A^T y_\ell - \hat{\alpha}_\ell z_\ell - \hat{\beta}_{\ell-1} z_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \dots, 2s\}, \end{aligned}$$

where z_ℓ and v_ℓ denote the ℓth columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$, respectively. Note that above, the coefficients $\hat{\alpha}_\ell$, $\hat{\beta}_\ell$, and $\hat{\gamma}_\ell$ could be different for computation of $y_{\ell+1}$ and $z_{\ell+1}$.

Then

$$\begin{aligned} \bar{v}_{sk+j+1} &= A^T p_{sk+j} - \rho_{sk+j} v_{sk+j} \\ \mathcal{Z}_k \bar{v}'_{j+1} &= A^T \underline{\mathcal{Y}}_k p'_j - \rho_{sk+j} \mathcal{Z}_k v'_j \\ &= \mathcal{Z}_k \tilde{T}_k p'_j - \rho_{sk+j} \mathcal{Z}_k v'_j \end{aligned}$$

and

$$\begin{aligned} \bar{p}_{sk+j+1} &= Av_{sk+j+1} - \theta_{sk+j+1} p_{sk+j} \\ \mathcal{Y}_k \bar{p}'_{j+1} &= A\underline{\mathcal{Z}}_k v'_{j+1} - \theta_{sk+j+1} \mathcal{Y}_k p'_j \\ &= \mathcal{Y}_k T_k v'_{j+1} - \theta_{sk+j+1} \mathcal{Y}_k p'_j. \end{aligned}$$

Therefore in the inner loop we can update

$$\begin{aligned} \bar{v}'_{j+1} &= \tilde{T}_k p'_j - \rho_{sk+j} v'_j \qquad \text{and} \\ \bar{p}'_{j+1} &= T_k v'_{j+1} - \theta_{sk+j+1} p'_j, \end{aligned}$$

and recover the iteration vectors by

$$v_{sk+j+1} = \mathcal{Z}_k v'_{j+1} \quad \text{and} \quad p_{sk+j+1} = \mathcal{Y}_k p'_{j+1},$$

for $j \in \{1, \ldots, s\}$.

The scalars required for normalization can be computed by

$$\theta_{sk+j+1} = \|\bar{v}_{sk+j+1}\|_2 = \left(\bar{v}'^T_{j+1}(\mathcal{Z}_k^T \mathcal{Z}_k)\bar{v}'_{j+1}\right)^{1/2} \equiv \left(\bar{v}'^T_{j+1} G_k^{(\mathcal{Z})} \bar{v}'_{j+1}\right)^{1/2} \quad \text{and}$$
$$\rho_{sk+j+1} = \|\bar{p}_{sk+j+1}\|_2 = \left(\bar{p}'^T_{j+1}(\mathcal{Y}_k^T \mathcal{Y}_k)\bar{p}'_{j+1}\right)^{1/2} \equiv \left(\bar{p}'^T_{j+1} G_k^{(\mathcal{Y})} \bar{p}'_{j+1}\right)^{1/2},$$

and then

$$v'_{j+1} = \bar{v}'_{j+1}/\theta_{sk+j+1} \quad \text{and}$$
$$p'_{j+1} = \bar{p}'_{j+1}/\rho_{sk+j+1}.$$

The resulting communication-avoiding upper bidiagonalization algorithm is shown in Algorithm 7.

Algorithm 7 Communication-avoiding upper bidiagonalization with alternating matrix powers

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ vector $b$
 1: $\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
 2: for $k = 0, 1, \ldots$ until convergence do
 3:     Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\underline{\mathcal{Z}}_k = \mathcal{Y}_k T_k$ and $A^T\underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{T}_k$.
 4:     $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$, $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$
 5:     $v'_1 = e_1$, $p'_1 = e_1$
 6:     for $j = 1, \ldots, s$ do
 7:         $\bar{v}'_{j+1} = \tilde{T}_k p'_j - \rho_{sk+j} v'_j$
 8:         $\theta_{sk+j+1} = \left(\bar{v}'^T_{j+1} G_k^{(\mathcal{Z})} \bar{v}'_{j+1}\right)^{1/2}$
 9:         $v'_{j+1} = \bar{v}'_{j+1}/\theta_{sk+j+1}$
10:         $\bar{p}'_{j+1} = T_k v'_{j+1} - \theta_{sk+j+1} p'_j$
11:         $\rho_{sk+j+1} = \left(\bar{p}'^T_{j+1} G_k^{(\mathcal{Y})} \bar{p}'_{j+1}\right)^{1/2}$
12:         $p'_{j+1} = \bar{p}'_{j+1}/\rho_{sk+j+1}$
13:     end for
14:     $[v_{sk+2}, \ldots, v_{sk+s+1}] = \mathcal{Z}_k[v'_2, \ldots, v'_{s+1}]$
15:     $[p_{sk+2}, \ldots, p_{sk+s+1}] = \mathcal{Y}_k[p'_2, \ldots, p'_{s+1}]$
16: end for
```

4.3.2. Reduction to lower bidiagonal form. Now we derive a version of lower bidiagonalization using coupled recurrences to generate bases for u_{sk+j+1} and v_{sk+j+1}. Recall that for $j \in \{0, \ldots, s\}$,

$$u_{sk+j+1} \in \mathcal{K}_s(AA^T, Av_{sk+1}) + \mathcal{K}_s(AA^T, u_{sk+1}) \quad \text{and}$$
$$v_{sk+j+1} \in \mathcal{K}_{s+1}(A^T A, v_{sk+1}) + \mathcal{K}_s(A^T A, A^T u_{sk+1}).$$

Then assume that we have a $2s$ dimensional basis $\mathcal{Y}_k$ such that $u_{sk+j+1} = \mathcal{Y}_k u'_{j+1}$ and a $2s+1$ dimensional basis $\mathcal{Z}_k$ such that $v_{sk+j+1} = \mathcal{Z}_k v'_{j+1}$ for $j \in \{1, \ldots, s\}$, and that these bases satisfy the recurrences

$$A\underline{\mathcal{Z}}_k = \mathcal{Y}_k T_k \quad \text{and} \quad A^T\underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{T}_k,$$

where $\underline{\mathcal{Z}}_k$ and $\underline{\mathcal{Y}}_k$ are the same as $\mathcal{Z}_k$ and $\mathcal{Y}_k$ but with the last column removed. Given u_{sk+1} and v_{sk+1}, the columns of $\mathcal{Y}_k$ and $\mathcal{Z}_k$ can be generated by computing

$$\begin{aligned} y_2 &= (Az_1 - \hat{\alpha}_1 y_1)/\hat{\gamma}_1, \\ z_2 &= (A^T y_1 - \hat{\alpha}_1 z_1)/\hat{\gamma}_1, \quad \text{and} \\ y_{\ell+1} &= (Az_\ell - \hat{\alpha}_\ell y_\ell - \hat{\beta}_{\ell-1} y_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \ldots, 2s\}, \\ z_{\ell+1} &= (A^T y_\ell - \hat{\alpha}_\ell z_\ell - \hat{\beta}_{\ell-1} z_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \ldots, 2s+1\}, \end{aligned}$$

where z_ℓ and v_ℓ denote the ℓth columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$, respectively. Note that above, as in the upper bidiagonalization case, the coefficients $\hat{\alpha}_\ell$, $\hat{\beta}_\ell$, and $\hat{\gamma}_\ell$ can be different for computation of $y_{\ell+1}$ and $z_{\ell+1}$.

Then

$$\begin{aligned} \bar{u}_{sk+j+1} &= Av_{sk+j} - \alpha_{sk+j} u_{sk+j} \\ \mathcal{Y}_k \bar{u}'_{j+1} &= A\underline{\mathcal{Z}}_k v'_j - \alpha_{sk+j} \mathcal{Y}_k u'_j \\ &= \mathcal{Y}_k T_k v'_j - \alpha_{sk+j} \mathcal{Y}_k u'_j \end{aligned}$$

and

$$\begin{aligned} \bar{v}_{sk+j+1} &= A^T u_{sk+j+1} - \beta_{sk+j+1} v_{sk+j} \\ \mathcal{Z}_k \bar{v}'_{j+1} &= A^T \underline{\mathcal{Y}}_k u'_{j+1} - \beta_{sk+j+1} \mathcal{Z}_k v'_j \\ &= \mathcal{Z}_k \tilde{T}_k u'_{j+1} - \beta_{sk+j+1} \mathcal{Z}_k v'_j. \end{aligned}$$

Therefore in the inner loop we can update

$$\begin{aligned} \bar{u}'_{j+1} &= T_k v'_j - \alpha_{sk+j} u'_j \qquad \text{and} \\ \bar{v}'_{j+1} &= \tilde{T}_k u'_{j+1} - \beta_{sk+j+1} v'_j, \end{aligned}$$

and recover the iteration vectors by

$$u_{sk+j+1} = \mathcal{Y}_k u'_{j+1} \quad \text{and} \quad v_{sk+j+1} = \mathcal{Z}_k v'_{j+1},$$

for $j \in \{1, \ldots, s\}$.

The scalars required for normalization can be computed by

$$\begin{aligned} \beta_{sk+j+1} &= \|\bar{u}_{sk+j+1}\|_2 = \left(\bar{u}'^T_{j+1}(\mathcal{Y}_k^T \mathcal{Y}_k)\bar{u}'_{j+1}\right)^{1/2} \equiv \left(\bar{u}'^T_{j+1} G_k^{(\mathcal{Y})} \bar{u}'_{j+1}\right)^{1/2} \quad \text{and} \\ \alpha_{sk+j+1} &= \|\bar{v}_{sk+j+1}\|_2 = \left(\bar{v}'^T_{j+1}(\mathcal{Z}_k^T \mathcal{Z}_k)\bar{v}'_{j+1}\right)^{1/2} \equiv \left(\bar{v}'^T_{j+1} G_k^{(\mathcal{Z})} \bar{v}'_{j+1}\right)^{1/2}, \end{aligned}$$

and then

$$\begin{aligned} u'_{j+1} &= \bar{u}'_{j+1}/\beta_{sk+j+1} \quad \text{and} \\ v'_{j+1} &= \bar{v}'_{j+1}/\alpha_{sk+j+1}. \end{aligned}$$

The resulting communication-avoiding lower bidiagonalization algorithm is shown in Algorithm 8.

Algorithm 8 Communication-avoiding lower bidiagonalization with alternating matrix powers

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
 1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
 2: for $k = 0, 1, \ldots$ until convergence do
 3:     Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\underline{\mathcal{Z}}_k = \mathcal{Y}_k T_k$ and $A^T \underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{T}_k$.
 4:     $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$, $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$
 5:     $v_1' = e_1$, $u_1' = e_1$
 6:     for $j = 1, \ldots, s$ do
 7:         $\bar{u}_{j+1}' = T_k v_j' - \alpha_{sk+j} u_j'$
 8:         $\beta_{sk+j+1} = \left(\bar{u}_{j+1}'^T G_k^{(\mathcal{Y})} \bar{u}_{j+1}'\right)^{1/2}$
 9:         $u_{j+1}' = \bar{u}_{j+1}'/\beta_{sk+j+1}$
10:         $\bar{v}_{j+1}' = \tilde{T}_k u_{j+1}' - \beta_{sk+j+1} v_j'$
11:         $\alpha_{sk+j+1} = \left(\bar{v}_{j+1}'^T G_k^{(\mathcal{Z})} \bar{v}_{j+1}'\right)^{1/2}$
12:         $v_{j+1}' = \bar{v}_{j+1}'/\alpha_{sk+j+1}$
13:     end for
14:     $[u_{sk+2}, \ldots, u_{sk+s+1}] = \mathcal{Y}_k [u_2', \ldots, u_{s+1}']$
15:     $[v_{sk+2}, \ldots, v_{sk+s+1}] = \mathcal{Z}_k [v_2', \ldots, v_{s+1}']$
16: end for
```

5. Communication-avoiding LSQR.

Paige and Saunders [19] showed that the quantities generated by the lower bidiagonalization procedure in Algorithm 2 can be used to solve the least-squares problem $\min \|b - Ax\|_2$. We briefly review the rationale behind the LSQR algorithm given by Paige and Saunders. For some vector y_i, define the quantities

$$\begin{aligned} x_i &= V_i y_i, \\ r_i &= b - Ax_i, \quad \text{and} \\ t_{i+1} &= \beta_1 e_1 - B_i y_i. \end{aligned}$$

Since for the lower bidiagonalization procedure we have $U_{i+1}(\beta_1 e_1) = b$ and $AV_i = U_{i+1} B_i$, it follows that $r_i = U_{i+1} t_{i+1}$, and since U_{i+1} is orthonormal, this suggests choosing y_i such that $\|t_{i+1}\|_2$ is minimized, which gives the least-squares problem $\min \|\beta_1 e_1 - B_i y_i\|_2$.

This problem is solved by updating the QR factorization of B_i in each iteration, given by

$$Q_i [B_i \quad \beta_1 e_1] = \begin{bmatrix} R_i & f_i \\ & \tilde{\phi}_{i+1} \end{bmatrix},$$

where R_i is the upper bidiagonal matrix produced by Algorithm 1. (Coincidentally, this factorization provides a link between the two bidiagonalization procedures; see [19]). Above, Q_i is the product of a series of plane rotations, i.e., $Q_i \equiv Q_{i,i+1} \cdots Q_{2,3} Q_{1,2}$. We then have

$$x_i = V_i R_i^{-1} f_i \equiv D_i f_i,$$

where the columns of D_i can be found successively by forward substitution on the system $R_i^T D_i^T = V_i^T$. This gives

$$\begin{aligned} d_i &= (1/\rho_i)(v_i - \theta_i d_{i-1}) \quad \text{and} \\ x_i &= x_{i-1} + \phi_i d_i, \end{aligned}$$

where $d_0 = x_0 = 0$.

The QR factorization is determined by constructing the ith plane rotation $Q_{i,i+1}$ to operate on rows i and $i+1$ of the transformed $[B_i \quad \beta_1 e_1]$ and eliminate β_{i+1}. This recurrence relation can be written

$$\begin{bmatrix} c_i & s_i \\ s_i & -c_i \end{bmatrix} \begin{bmatrix} \bar{\rho}_i & 0 & \bar{\phi}_i \\ \beta_{i+1} & \alpha_{i+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_i & \theta_{i+1} & \phi_i \\ 0 & \bar{\rho}_{i+1} & \bar{\phi}_{i+1} \end{bmatrix},$$

where $\bar{\rho}_1 \equiv \alpha_1$, $\bar{\phi}_1 \equiv \beta_1$, and c_i and s_i are the elements of $Q_{i,i+1}$. Note that s without a subscript still denotes the iteration blocking factor. In the algorithm, vectors $w_i \equiv \rho_i d_i$ are computed instead of d_i. As in the previous section, quantities with bars denote intermediate variables.

Thus, the LSQR algorithm proceeds as follows. One begins by setting

$$\bar{\phi}_1 = \beta_1, \quad \bar{\rho}_1 = \alpha_1, \quad w_1 = v_1, \quad \text{and} \quad x_1 = 0_{n,1},$$

and proceeds with the Lanczos lower bidiagonalization process (Algorithm 2). In each iteration, after β_{i+1}, α_{i+1}, and v_{i+1} have been computed via the bidiagonalization process, one updates

$$\begin{aligned} \rho_i &= (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}, \\ c_i &= \bar{\rho}_i/\rho_i, \\ s_i &= \beta_{i+1}/\rho_i, \\ \theta_{i+1} &= s_i \alpha_{i+1}, \\ \bar{\rho}_{i+1} &= -c_i \alpha_{i+1}, \\ \phi_i &= c_i \bar{\phi}_i, \\ \bar{\phi}_{i+1} &= s_i \bar{\phi}_i, \\ x_{i+1} &= x_i + \frac{\phi_i}{\rho_i} w_i, \\ w_{i+1} &= v_{i+1} - \frac{\theta_{i+1}}{\rho_i} w_i, \quad \text{and} \\ r_{i+1} &= \|b - Ax_{i+1}\|_2. \end{aligned}$$

The resulting algorithm is shown in Algorithm 9. Any of the communication-avoiding variants of the lower bidiagonalization algorithm given in Section 4 can be adapted to give a communication-avoiding version of LSQR. In Algorithm 11 we show a CA-LSQR method based on the implementation in Algorithm 6. For reference, we give the intermediate step in obtaining this new method, a classical LSQR algorithm which uses auxiliary quantities, in Algorithm 10. In Algorithm 12, we give a CA-LSQR method using the alternating matrix powers approach of the bidiagonalization in Algorithm 8.

Note that in Algorithm 11 and 12, as in the previous section, although we have shown the recovery of all iteration vectors for all s iterations at the end of each outer loop, only iteration vectors for the last of the s iterations need be computed.

Algorithm 9 LSQR

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
 1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
 2: $\bar{u}_1 = A^T u_1$, $\bar{v}_1 = Av_1$
 3: $\bar{\phi}_1 = \beta_1 \quad \bar{\rho}_1 = \alpha_1 \quad w_1 = v_1 \quad x_1 = 0_{n,1}$
 4: for $i = 1, 2, \dots$ until convergence do
 5:     $\bar{u}_{i+1} = Av_i - \alpha_i u_i$
 6:     $\beta_{i+1} = \left(\bar{u}_{i+1}^T \bar{u}_{i+1}\right)^{1/2}$
 7:     $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
 8:     $\bar{v}_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$
 9:     $\alpha_{i+1} = \left(\bar{v}_{i+1}^T \bar{v}_{i+1}\right)^{1/2}$
10:     $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
11:     $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$
12:     $c_i = \bar{\rho}_i/\rho_i, \qquad s_i = \beta_{i+1}/\rho_i$
13:     $\theta_{i+1} = s_i \alpha_{i+1}, \qquad \bar{\rho}_{i+1} = -c_i \alpha_{i+1}$
14:     $\phi_i = c_i \bar{\phi}_i, \qquad \bar{\phi}_{i+1} = s_i \bar{\phi}_i$
15:     $x_{i+1} = x_i + (\phi_i/\rho_i)\, w_i$
16:     $w_{i+1} = v_{i+1} - (\theta_{i+1}/\rho_i)\, w_i$
17: end for
```

Algorithm 10 LSQR with auxiliary quantities

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
 1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
 2: $\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = Av_1$
 3: $\bar{\phi}_1 = \beta_1, \quad \bar{\rho}_1 = \alpha_1, \quad w_1 = v_1, \quad x_1 = 0_{n,1}$
 4: for $i = 1, 2, \dots$ until convergence do
 5:     $\bar{u}_{i+1} = \tilde{v}_i - \alpha_i u_i$
 6:     $\beta_{i+1} = \left(\bar{u}_{i+1}^T \bar{u}_{i+1}\right)^{1/2}$
 7:     $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
 8:     $\tilde{u}_{i+1} = (A^T A v_i - \alpha_i \tilde{u}_i)/\beta_{i+1}$
 9:     $\bar{v}_{i+1} = \tilde{u}_{i+1} - \beta_{i+1} v_i$
10:     $\alpha_{i+1} = \left(\bar{v}_{i+1}^T \bar{v}_{i+1}\right)^{1/2}$
11:     $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
12:     $\tilde{v}_{i+1} = (AA^T u_{i+1} - \beta_{i+1} \tilde{v}_i)/\alpha_{i+1}$
13:     $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$
14:     $c_i = \bar{\rho}_i/\rho_i, \qquad s_i = \beta_{i+1}/\rho_i$
15:     $\theta_{i+1} = s_i \alpha_{i+1}, \qquad \bar{\rho}_{i+1} = -c_i \alpha_{i+1}$
16:     $\phi_i = c_i \bar{\phi}_i, \qquad \bar{\phi}_{i+1} = s_i \bar{\phi}_i$
17:     $x_{i+1} = x_i + (\phi_i/\rho_i)\, w_i$
18:     $w_{i+1} = v_{i+1} - (\theta_{i+1}/\rho_i)\, w_i$
19: end for
```

6. Future work.

In this manuscript, we have derived three communication-avoiding variants of both upper and lower Lanczos bidiagonalization procedures, and have given two corresponding versions of communication-avoiding LSQR solvers. Future work involves evaluation of both the convergence and stability problems of the various communication-avoiding methods presented here for a variety of different prob-

Algorithm 11 CA-LSQR

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
2: $\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = Av_1$
3: $\bar{\phi}_1 = \beta_1$,   $\bar{\rho}_1 = \alpha_1$,   $w_1 = v_1$,   $x_1 = 0_{n,1}$
4: for $k = 0, 1, \dots$ until convergence do
5:     Compute $\mathcal{U}_k$, a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{U}}_k$, a basis for $\mathcal{K}_s(A^TA, A^Tu_{sk+1})$, $\mathcal{V}_k$, a basis for $\mathcal{K}_{s+1}(A^TA, v_{sk+1})$, and $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$. Let $\mathcal{Y}_k = [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{U}}_k]$.
6:     $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$,   $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$
7:     $u'_1 = e_1$, $v'_1 = e_1$, $\tilde{u}'_1 = e_{s+2}$, $\tilde{v}'_1 = e_{s+1}$.
8:     $w'_1 = [0_{1,2s+1}, 1]^T$, $x'_1 = 0_{2s+2,1}$
9:     for $j = 1, \dots, s$ do
10:        $\bar{u}'_{j+1} = \tilde{v}'_j - \alpha_{sk+j} u'_j$
11:        $\beta_{sk+j+1} = \left(\bar{u}'^T_{j+1} G_k^{(\mathcal{Y})} \bar{u}'_{j+1}\right)^{1/2}$
12:        $u'_{j+1} = \bar{u}'_{j+1}/\beta_{sk+j+1}$
13:        $\tilde{u}'_{j+1} = (T_k^{(\mathcal{Z})} v'_j - \alpha_{sk+j}\tilde{u}'_j)/\beta_{sk+j+1}$
14:        $\bar{v}'_{j+1} = \tilde{u}'_{j+1} - \beta_{sk+j+1} v'_j$
15:        $\alpha_{sk+j+1} = \left(\bar{v}'^T_{j+1} G_k^{(\mathcal{Z})} \bar{v}'_{j+1}\right)^{1/2}$
16:        $v'_{j+1} = \bar{v}'_{j+1}/\alpha_{sk+j+1}$
17:        $\tilde{v}'_{j+1} = (T_k^{(\mathcal{Y})} u'_{j+1} - \beta_{sk+j+1}\tilde{v}'_j)/\alpha_{sk+j+1}$
18:        $\rho_{sk+j} = (\bar{\rho}^2_{sk+j} + \beta^2_{sk+j+1})^{1/2}$
19:        $c_{sk+j} = \bar{\rho}_{sk+j}/\rho_{sk+j}$,   $s_{sk+j} = \beta_{sk+j+1}/\rho_{sk+j}$
20:        $\theta_{sk+j+1} = s_{sk+j}\alpha_{sk+j+1}$,   $\bar{\rho}_{sk+j+1} = -c_{sk+j}\alpha_{sk+j+1}$
21:        $\phi_{sk+j} = c_{sk+j}\bar{\phi}_{sk+j}$,   $\bar{\phi}_{sk+j+1} = s_{sk+j}\bar{\phi}_{sk+j}$
22:        $x'_{j+1} = x'_j + (\phi_{sk+j}/\rho_{sk+j})\, w'_j$
23:        $w'_{j+1} = [v'^T_{j+1}, 0]^T - (\theta_{sk+j+1}/\rho_{sk+j})\, w'_j$
24:    end for
25:    $[u_{sk+2}, \dots, u_{sk+s+1}] = \mathcal{Y}_k[u'_2, \dots, u'_{s+1}]$
26:    $[v_{sk+2}, \dots, v_{sk+s+1}] = \mathcal{Z}_k[v'_2, \dots, v'_{s+1}]$
27:    $[x_{sk+2}, \dots, x_{sk+s+1}] = [\mathcal{Z}_k, w_{sk+1}][x'_2, \dots, x'_{s+1}] + x_{sk+1}1_{1,s}$
28:    $[w_{sk+2}, \dots, w_{sk+s+1}] = [\mathcal{Z}_k, w_{sk+j}][w'_2, \dots, w'_{s+1}]$
29: end for
```

lems (and different s values), as well as a performance study for both sequential and parallel versions. It is not clear from the derivations whether one method will always win over the others in terms of best speed per iteration or best convergence rate. The correct method to choose will depend on the structure and conditioning of the matrix, the requirements of the particular application, and machine-specific parameters such as cache size and relative latency/bandwidth cost.

Acknowledgements. This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0004938, DE-SC0003959, and DE-SC0010200; by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, X-Stack program under Award

Algorithm 12 CA-LSQR with alternating matrix powers

```
Require: $m$-by-$n$ matrix $A$ and length-$n$ starting vector $b$
 1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
 2: $\bar{\phi}_1 = \beta_1$,   $\bar{\rho}_1 = \alpha_1$,   $w_1 = v_1$,   $x_1 = 0_{n,1}$
 3: for $k = 0, 1, \ldots$ until convergence do
 4:     Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\underline{\mathcal{Z}}_k = \mathcal{Y}_k T_k$ and $A^T \underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{T}_k$.
 5:     $G_k^{(\mathcal{Z})} = \mathcal{Z}^T \mathcal{Z}$,   $G_k^{(\mathcal{Y})} = \mathcal{Y}^T \mathcal{Y}$
 6:     $v'_1 = e_1$, $u'_1 = e_1$
 7:     $w'_1 = [0_{1,2s+1}, 1]^T$, $x'_1 = 0_{2s+2,1}$
 8:     for $j = 1, \ldots, s$ do
 9:         $\bar{u}'_{j+1} = T_k v'_j - \alpha_{sk+j} u'_j$
10:         $\beta_{sk+j+1} = \left(\bar{u}'^T_{j+1} G_k^{(\mathcal{Y})} \bar{u}'_{j+1}\right)^{1/2}$
11:         $u'_{j+1} = \bar{u}'_{j+1}/\beta_{sk+j+1}$
12:         $\bar{v}'_{j+1} = \tilde{T}_k u'_{j+1} - \beta_{sk+j+1} v'_j$
13:         $\alpha_{sk+j+1} = \left(\bar{v}'^T_{j+1} G_k^{(\mathcal{Z})} \bar{v}'_{j+1}\right)^{1/2}$
14:         $v'_{j+1} = \bar{v}'_{j+1}/\alpha_{sk+j+1}$
15:         $\rho_{sk+j} = (\bar{\rho}^2_{sk+j} + \beta^2_{sk+j+1})^{1/2}$
16:         $c_{sk+j} = \bar{\rho}_{sk+j}/\rho_{sk+j}$,      $s_{sk+j} = \beta_{sk+j+1}/\rho_{sk+j}$
17:         $\theta_{sk+j+1} = s_{sk+j}\alpha_{sk+j+1}$,   $\bar{\rho}_{sk+j+1} = -c_{sk+j}\alpha_{sk+j+1}$
18:         $\phi_{sk+j} = c_{sk+j}\bar{\phi}_{sk+j}$,      $\bar{\phi}_{sk+j+1} = s_{sk+j}\bar{\phi}_{sk+j}$
19:         $x'_{j+1} = x'_j + (\phi_{sk+j}/\rho_{sk+j})\, w'_j$
20:         $w'_{j+1} = [v'^T_{j+1}, 0]^T - (\theta_{sk+j+1}/\rho_{sk+j})\, w'_j$
21:     end for
22:     $[u_{sk+2}, \ldots, u_{sk+s+1}] = \mathcal{Y}_k [u'_2, \ldots, u'_{s+1}]$
23:     $[v_{sk+2}, \ldots, v_{sk+s+1}] = \mathcal{Z}_k [v'_2, \ldots, v'_{s+1}]$
24:     $[x_{sk+2}, \ldots, x_{sk+s+1}] = [\mathcal{Z}_k, w_{sk+1}][x'_2, \ldots, x'_{s+1}] + x_{sk+1} 1_{1,s}$
25:     $[w_{sk+2}, \ldots, w_{sk+s+1}] = [\mathcal{Z}_k, w_{sk+j}][w'_2, \ldots, w'_{s+1}]$
26: end for
```

Numbers DE-SC0005136, DE-SC0008699, DE-SC0008700, and AC02-05CH11231; by DARPA Award Number HR0011-12-2-0016, as well as contributions from Intel, Oracle, and MathWorks.

REFERENCES

[1] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, *Communication lower bounds and optimal algorithms for numerical linear algebra*, Acta Numerica, 23 (2014), pp. 1–155.

[2] E. Carson and J. Demmel, *Accuracy of the s-step Lanczos method for the symmetric eigenproblem*, Tech. Report UCB/EECS-2014-165, EECS Dept., U.C. Berkeley, Sep 2014.

[3] E. Carson, N. Knight, and J. Demmel, *Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods*, SIAM J. Sci. Comp., 35 (2013).

[4] A. Chronopoulos and C. Gear, *On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy*, Parallel Comput., 11 (1989), pp. 37–53.

[5] ———, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math, 25 (1989), pp. 153–168.

[6] A. Chronopoulos and C. Swanson, *Parallel iterative s-step methods for unsymmetric linear systems*, Parallel Comput., 22 (1996), pp. 623–641.

[7] J. Cullum and W. Donath, *A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices*, in Proc. of the 1974 IEEE Conf. on Decision and Control, IEEE, 1974, pp. 505–509.

[8] E. de Sturler, *A performance model for Krylov subspace methods on mesh-based parallel computers*, Parallel Comput., 22 (1996), pp. 57–74.

[9] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in computing Krylov subspaces*, Tech. Report UCB/EECS-2007-123, EECS Dept., U.C. Berkeley, Oct 2007.

[10] ———, *Avoiding communication in sparse matrix computations*, in Proc. Int. Parallel Dist. Proc. Symp., IEEE, 2008, pp. 1–12.

[11] D. Gannon and J. Van Rosendale, *On the impact of communication complexity on the design of parallel numerical algorithms*, Trans. Comput., 100 (1984), pp. 1180–1194.

[12] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis, 2 (1965), pp. 205–224.

[13] G. Golub and C. Van Loan, *Matrix computations*, vol. 3, Johns Hopkins University Press, 2012.

[14] A. Hindmarsh and H. Walker, *Note on a Householder implementation of the GMRES method*, Tech. Report UCID-20899, Lawrence Livermore National Lab., CA., 1986.

[15] M. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD thesis, EECS Dept., U.C. Berkeley, 2010.

[16] W. Karush, *An iterative method for finding characteristic vectors of a symmetric matrix*, Pacific J. Math, 1 (1951), pp. 233–248.

[17] M. Mohiyuddin, *Tuning Hardware and Software for Multiprocessors*, PhD thesis, EECS Dept., U.C. Berkeley, May 2012.

[18] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, *Minimizing communication in sparse matrix solvers*, in Proc. ACM/IEEE Conference on Supercomputing, 2009.

[19] C. Paige and M. Saunders, *Lsqr: An algorithm for sparse linear equations and sparse least squares*, ACM Transactions on Mathematical Software (TOMS), 8 (1982), pp. 43–71.

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

[21] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.

[22] S. Toledo, *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms*, PhD thesis, MIT, 1995.

[23] J. Van Rosendale, *Minimizing inner product data dependencies in conjugate gradient iteration*, Tech. Report 172178, ICASE-NASA, 1983.

[24] S. Williams, M. Lijewski, A. Almgren, B. Van Straalen, E. Carson, N. Knight, and J. Demmel, *s-step Krylov subspace methods as bottom solvers for geometric multigrid*, in International Symposium on Parallel and Distributed Processing, IEEE, 2014.