Avoiding Communication in Successive Band Reduction

Grey Ballard
James Demmel
Nicholas Knight

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-131
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-131.html

July 11, 2013

Acknowledgement

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Also supported by U.S. DOE grants DE-SC0003959, DE-AC02-05-CH11231, and by NSF SDCI under Grant Number OCI-1032639.

Z

Avoiding Communication in Successive Band Reduction

GREY BALLARD, University of California at Berkeley
JAMES DEMMEL, University of California at Berkeley
NICHOLAS KNIGHT, University of California at Berkeley

The running time of an algorithm depends on both arithmetic and communication (i.e., data movement) costs, and the relative costs of communication are growing over time. In this work, we present sequential and parallel algorithms for tridiagonalizing a symmetric band matrix that asymptotically reduce communication compared to previous approaches.

The tridiagonalization of a symmetric band matrix is a key kernel in solving the symmetric eigenvalue problem for both full and band matrices. In order to preserve sparsity, tridiagonalization routines use annihilate-and-chase procedures that previously have suffered from poor data locality. We improve data locality by reorganizing the computation and obtain asymptotic improvements. We consider the cases of computing eigenvalues only and of computing eigenvalues and all eigenvectors.

1. INTRODUCTION

The running time of an algorithm depends on both the number of floating point operations performed (*arithmetic*) and the amount of data moved (*communication*) through the memory hierarchy of a single processor and, in the parallel case, across a network between processors. The cost of moving data on today's machines already greatly exceeds the cost of performing floating point operations on it, and architectural trends indicate that this processor-memory gap is growing exponentially over time [Fuller and Millett 2011]. Thus, we are interested in new algorithms which reduce the communication costs of existing ones, even at the expense of doing more arithmetic.

In this work, we present new sequential and parallel algorithms for tridiagonalizing a symmetric band matrix in order to compute its eigendecomposition. Our algorithms reduce the communication costs compared to previous approaches. Although no communication lower bound is known for this problem, we demonstrate that previous approaches communicate asymptotically more than necessary.

While the symmetric band problem is interesting in its own right, this work is motivated by the high communication costs of the standard algorithms for solving the full (dense) symmetric eigenproblem via tridiagonalization. Greater efficiency than the standard approach can be obtained if the tridiagonalization procedure is split into two steps: reducing the full matrix to band form and then reducing the band matrix to tridiagonal form. Thus, by reducing the communication and improving the algorithm for tridiagonalizing a band matrix, we can also improve algorithms for tridiagonalizing a full matrix. While we focus on symmetric matrices in this work, the ideas here can be readily applied to tridiagonalization of Hermitian matrices as well as bidiagonalization of general matrices (for singular value problems).

In order to preserve band structure, band reduction algorithms based on orthogonal similarity transformations proceed by an annihilate-and-chase approach. Annihilating entries within the band creates fill-in (bulges); to preserve sparsity, these bulges are chased off the band before annihilating subsequent entries. The most general framework for this procedure, known as successive band reduction (SBR), appears in [Bischof et al. 2000b].

The main contributions of this work are the following:

New Techniques for Avoiding Communication. In Section 3, we extend the band reduction algorithm design space with new techniques for avoiding communication. The main novel contribution is the idea of chasing multiple bulges in the context of SBR.

New/Improved Sequential Algorithms. In Section 4, we give an asymptotic complexity analysis of previous approaches, and show how our new techniques can be used to improve their communication costs. We also introduce a new algorithm, CA-SBR, which communicates asymptotically less than all other approaches.

New Parallel Algorithm. In Section 5, we extend CA-SBR to a distributed-memory parallel algorithm which communicates asymptotically fewer messages than previous approaches.

A preliminary version of this work appeared in [Ballard et al. 2012]. The multiple bulge chasing approach and sequential CA-SBR algorithm (for eigenvalues only) first appeared in that paper. We also showed how to extend the sequential algorithm to a shared-memory parallel environment. Our implementations obtained $2-6\times$ speedups over state-of-the-art library implementations. This paper extends those results in two ways: we discuss distributed-memory algorithms and consider computing both eigenvalues and eigenvectors. However, we do not give implementation details or performance results in this work.

2. PRELIMINARIES

2.1. Communication Model

In order to quantify the communication costs of an algorithm, we model a sequential machine with two levels of memory hierarchy (fast and slow) and count the number of words moved between these two levels during the execution of the algorithm; this we call the *bandwidth cost*. This model is sometimes referred to as the two-level I/O or disk access model (see, e.g., [Aggarwal and Vitter 1988]) and the number of words moved is also known as the I/O-complexity of the algorithm. We use M to denote the size of the fast memory in words. If words are stored contiguously in slow memory, then they can be read or written together as a *message*. We are also interested in the number of messages transferred between fast and slow memory, which we call the *latency cost*. In our model, messages may range in size from one word to M words.

In the distributed-memory parallel case, we model the machine as a collection of p processors, each with a limited local memory of size M, connected over a network. We assume processors can communicate via point-to-point messages, and each processor can send or receive only one message at a time. The network topology is assumed to have all-to-all connectivity, so we do not model contention or the number of hops a message would travel on a more realistic physical network. Again, we are interested in both the number of words (bandwidth cost) and messages (latency cost), and we count these costs along the critical path of the algorithm. That is, if two processors each send a message to separate processors simultaneously, the cost along the critical path is that of one message.

2.2. Eigendecomposition of Band Matrices

In this paper, we are interested in computing the eigenvalues (and possibly the eigenvectors) of a symmetric band matrix via tridiagonalization. Let $A \in \mathbb{R}^{n\times n}$ be a symmetric band matrix with *bandwidth* b (i.e., having $2b+1$ nonzero diagonals). Because we preserve symmetry, it is sufficient to store and operate on only the lower $b+1$ diagonals of A. We reduce A to a symmetric tridiagonal matrix T via orthogonal similarity transformations which comprise an orthogonal matrix Q such that $Q^TAQ = T$. We refer to this process as the *band reduction* phase. We assume the eigendecomposition of the tridiagonal matrix T is computed via an efficient algorithm such as Bisection/Inverse Iteration, MRRR, Divide-and-Conquer, or QR Iteration (see, e.g., [Demmel et al. 2008]), and we ignore the computation and communication costs of this phase.

If only eigenvalues are desired, the eigenvalues of T are the eigenvalues of A, so no extra computation is required and Q need not be computed or stored. If eigenvectors are also

desired, then a *back-transformation* phase is needed to reconstruct the eigenvectors of A from the eigenvectors of T. That is, if the eigendecomposition of T is given by $T = V\Lambda V^T$, then the eigendecomposition of A is $A = (QV)\Lambda(QV)^T$, so to compute the eigenvectors of A, we must compute QV. There are a range of possibilities for computing QV: if we form Q and V explicitly, then this can be done with matrix multiplication; if we store Q implicitly (e.g., as a set of Householder vectors), then it can be applied to V after V is formed explicitly; if QR Iteration is used to compute the eigendecomposition of T, then Q should be formed explicitly so that V can be applied implicitly to Q from the right as it is computed; or Q and V can be left implicit, allowing us to multiply by them when needed.

In many applications only a subset of eigenpairs are desired. The cost of the back-transformation can be made proportional to the number of eigenpairs desired; this can significantly improve the runtime. Here, we consider only the case of computing all n eigenpairs.

One of the most important applications of solving the symmetric band eigenproblem is when solving the full symmetric eigenproblem. An efficient alternative to direct tridiagonalization [Wilkinson 1962] is a two-step approach [Bischof et al. 2000b]: (1) reducing the full matrix to a band matrix, and (2) reducing the band matrix to tridiagonal form. Both direct and two-step tridiagonalization approaches use orthogonal similarity transformations. The remainder of this work concerns step (2); we discuss step (1) in Section 6.

2.3. SBR Notation

We follow notation from [Bischof et al. 2000b] and the authors' related papers to describe the terminology associated with successive band reduction (SBR), our approach for reducing A to T. While we do not give a complete description of SBR here, Figure 2 in [Bischof et al. 2000b] is particularly helpful for visualizing the framework.

To exploit symmetry, we store and operate on only the lower triangle of the band matrix, though analogous algorithms apply to the upper triangle. When we refer to a column of the band, we mean the entries of the column on and below the diagonal.

In a given *sweep*, SBR eliminates d subdiagonals in sets of c columns,[1] using an annihilate-and-chase approach. We assume Householder transformations are used; each set of transformations eliminates a d-by-c *parallelogram* of nonzeros but creates trapezoidal-shaped fill (a *bulge*). Using analogous orthogonal similarities, SBR chases each bulge off the end of the band, translating the bulge b columns to the right with each *bulge chase*. Figure 1 shows the data access pattern of a single bulge chase. A QR decomposition of the $(d+1+c)$-by-c matrix (QR region in Figure 1) containing the parallelogram computes the orthogonal matrix that annihilates the parallelogram; the corresponding rows (PRE region) are updated with a premultiplication of the orthogonal matrix; the corresponding columns (POST region) are updated with a postmultiplication by the transpose of the orthogonal matrix, creating the next bulge; and the lower half of the corresponding symmetric submatrix on the diagonal (SYM region) is updated from both the left and right.

We define the *working bandwidth* $b + d + 1$ to be the number of subdiagonals necessary to store the $b + 1$ diagonals of the matrix as well as to store the d diagonals that hold temporary fill-in during the course of a sweep. As observed in [Murata and Horikoshi 1975], we note that an entire bulge need not be eliminated; only the first c columns of the bulge must be annihilated to prevent subsequent bulges from introducing nonzeros beyond the working bandwidth. This results in temporary *triangular fill*.

We index sweeps with an integer i, where $i = 1$ is the first sweep, so $b_1 = b$ is the initial bandwidth. We index the parallelograms which initiate each bulge chase by j and the sequence of following bulges by the ordered pairs (j, k): j is the parallelogram index and k is the bulge index, as in [Bischof et al. 1994].

[1] We depart from the LAPACK-style notation `nb` of [Bischof et al. 2000b].

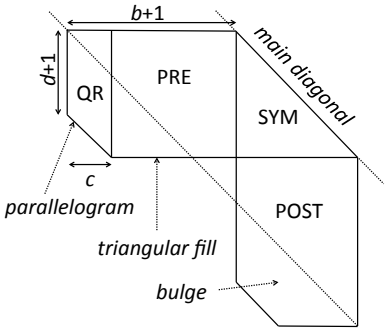


Fig. 1. Following the notation of [Bischof et al. 2000b], the bulge chasing operation based on an orthogonal similarity transformation can be decomposed into four parts. There are d diagonals in each bulge and c is the number of columns annihilated during a bulge chase which leaves behind triangular fill.

2.4. Related Work

Sequential Algorithms. The two papers of Bischof, Lang, and Sun [Bischof et al. 2000b; 2000a] provide a general framework of sequential SBR algorithms. Their approach first appeared in [Bischof and Sun 1992] and generalizes most of the related work described in this section.

The annihilate-and-chase strategy began with Rutishauser and Schwarz in 1963. Rutishauser [Rutishauser 1963] identified two extreme points in the SBR algorithm design space: (1) a Givens rotation-based approach with b sweeps and $c_i = d_i = 1$ for each i and (2) a column-based approach with one sweep where $c_1 = 1$ and $d_1 = b-1$. Rutishauser's first approach considered only pentadiagonal matrices; Schwarz [Schwarz 1963] generalized the algorithm to arbitrary bandwidths. Later, Schwarz [Schwarz 1968] proposed a different algorithm based on Givens rotations which does not fit in the SBR framework. This algorithm eliminates entries by column rather than by diagonal and does not generalize to parallelograms.

Murata and Horikoshi [Murata and Horikoshi 1975] improved on Rutishauser's column-based algorithm by noting that computation can be saved by eliminating only the first column of the triangular bulge rather than the entire triangle. If eigenvectors are desired, Bischof, Lang, and Sun [Bischof et al. 1994] showed that, with this approach, the Householder vectors comprising Q can be stored in a lower triangular n-by-n matrix and applied to V in a different order than they were computed, yielding higher performance during the back-transformation phase.

Kaufman [Kaufman 1984] vectorized the Rutishauser/Schwarz algorithm [Rutishauser 1963; Schwarz 1963], chasing multiple single-element bulges in each vector operation. Her motivation for chasing multiple bulges was not locality but rather to increase the length of the vector operation beyond the bandwidth b. Several years later, Kaufman [Kaufman 2000] took the approach of [Schwarz 1968] in order to maximize the vector operation length (especially in the case of large b) and make use of a BLAS subroutine when appropriate. When eigenvectors are requested, the Q matrix is formed explicitly by applying the updates to an identity matrix. By exploiting sparsity, the flop cost of constructing Q is about $(4/3)n^3$, compared with $2n^3$ if sparsity is ignored. The current LAPACK [Anderson et al. 1992] reference code for band reduction (`sbtrd`) is based on [Kaufman 2000].

More recently, Rajamanickam [Rajamanickam 2009] proposed and implemented a different way of eliminating a parallelogram and chasing its fill. His algorithm uses Givens rotations to eliminate the individual entries of a parallelogram, and instead of creating a large bulge, the update rotations are pipelined such that as soon as an element is filled in outside the band, it is immediately annihilated. The rotations are carefully ordered to

obtain temporal and sequential locality. By avoiding the fill-in, this algorithm does up to 50% fewer flops than the Householder-based elimination of parallelograms within SBR and requires minimal working bandwidth.

Parallel Algorithms. Lang [Lang 1991; 1993] implemented a distributed-memory parallel version of the band reduction algorithm in [Murata and Horikoshi 1975], although he did not consider computing Q. Bichof et al. [Bischof et al. 1993] implemented a distributed-memory parallel instance of the SBR framework in the context of tridiagonalizing a full matrix. A subsequent paper [Bischof et al. 1994] extended this implementation to reorganize and block the orthogonal updates comprising Q.

Luszczek et al. [Luszczek et al. 2011] implemented the band reduction algorithm from [Murata and Horikoshi 1975] as part of a two-step shared-memory tridiagonalization algorithm in the PLASMA library [Agullo et al. 2009], using dynamic DAG-scheduling of tile-based tasks. They distinguished between "right-looking" and "left-looking" variants: right-looking algorithms chase a bulge entirely off the band before eliminating the next parallelogram, left-looking algorithms chase bulges only far enough to allow for the next bulge to be created (see Constraint 3.2). For example, the SBR framework [Bischof et al. 2000b] is right-looking while Kaufman's algorithm [Kaufman 2000] is left-looking. In [Luszczek et al. 2011], they found improved performance with a left-looking variant. Later, Haidar et al. [Haidar et al. 2011] reduced the runtime of [Luszczek et al. 2011]; the improvements in the band-to-tridiagonal step include using an algorithm-specific (static) scheduler, "grouping" related tasks, and avoiding fill-in using pipelined Givens rotations (a single-sweep version of the approach in [Rajamanickam 2009]). While PLASMA 2.4.6 has support for computing eigenvectors, these results have not yet been published.

Auckenthaler et al. [Auckenthaler et al. 2011a; Auckenthaler et al. 2011b; Auckenthaler 2012] have implemented a two-step distributed-memory tridiagonalization algorithm as part of a solver for the generalized symmetric eigenproblem. Their band-to-tridiagonal step uses an improved version of Lang's algorithm [Lang 1993], which performs one sweep. They give a new algorithm for orthogonal updates which uses a 2D processor layout instead of a 1D layout. Their implementation also supports taking multiple sweeps when eigenvectors are not requested; however, this algorithm is not given explicitly.

2.5. Related Lower Bounds

No communication lower bound has been established for annihilate-and-chase band reduction algorithms, so we cannot conclude that our new algorithms are communication-optimal in an asymptotic sense. In fact, the general communication lower bound result of [Ballard et al. 2011b], which applies to many algorithms in numerical linear algebra including matrix multiplication and many QR decomposition algorithms, does not apply to SBR or its variants because they fail to satisfy "forward progress" [Ballard et al. 2011b, Definition 4.3]. That is, the lower bound proof there requires that an orthogonal transformation algorithm not fill in a previously created zero — this occurs frequently in SBR, unlike QR decomposition.

The main result of [Ballard et al. 2011b] states that an applicable algorithm that performs G flops must move $\Omega(G/\sqrt{M})$ words and send $\Omega(G/M^{3/2})$ messages for sufficiently large problems. For most dense matrix algorithms, the number of flops is $G = O(n^3/p)$, where $p = 1$ for the sequential case. In the parallel case, if we assume minimal local memory is used (i.e., $M = \Theta(n^2/p)$, or just enough to store the input and output matrices), the the lower bounds simplify to $\Omega(n^2/\sqrt{p})$ words and $\Omega(\sqrt{p})$ messages.

Since our new sequential algorithm (see Algorithm 1 and Table I) performs $O(n^2 b)$ flops, moves $O(n^2 b^2/M)$ words, and sends $O(n^2 b^2/M^2)$ messages, its bandwidth and latency costs drop below the lower bounds by a factor of $O(\sqrt{M}/b)$ for $2 \leq b \leq \sqrt{M}/3$. For small b and large n (such that the band does not fit entirely in fast memory), this discrepancy is as

much as $O(\sqrt{M})$. Similarly, our new parallel algorithm (see Algorithm 3 and Table III) also beats the lower bounds for bandwidth and latency costs, and the discrepancy is largest for small bandwidths. Thus, our algorithms show that not only does the lower bound proof technique not apply to annihilate-and-chase algorithms, the bound itself must not apply.

3. AVOIDING COMMUNICATION IN SUCCESSIVE BAND REDUCTION

The goal of our algorithms is to avoid communication by reorganizing computation, extending the SBR framework to obtain greater data locality. In the sequential case, we can asymptotically reduce the number of words and messages that must be moved between fast and slow memory during the execution of the algorithm; in the parallel case, we can asymptotically reduce the number of messages sent between processors. We achieve data locality (i.e., avoid communication) using two techniques described in Sections 3.1 and 3.2. We navigate the constraints and tradeoffs that arise using a successive halving approach, described in Section 3.3.

3.1. Applying Multiple Householder Transformations

The first means of achieving data locality is within a single bulge chase (see Figure 1). Since c Householder vectors are computed to eliminate the first c columns of the bulge (QR region), every entry in the PRE, SYM, and POST regions is updated by c left and/or right Householder transformations. These transformations may be applied one at a time or blocked (e.g., via [Schreiber and Van Loan 1989]). Assuming all the data involved in a single bulge chase reside in fast or local memory, $O(c)$ flops are performed for every entry read from slow memory.

We identify the following algorithmic constraint. If it is violated, then the parallelogram annihilated by the left update will be (partially) refilled by the right update (i.e., the SYM and POST regions overlap the QR region) — this implies wasted computation.

CONSTRAINT 3.1. *To annihilate a parallelogram within the SBR framework, the dimensions of the parallelogram must satisfy*

$$c + d \leq b.$$

While increasing c improves data locality, it limits the size of d due to Constraint 3.1. Because d is the number of diagonals eliminated in a sweep, this constraint creates a tradeoff between locality and progress towards tridiagonal form.

3.2. Chasing Multiple Bulges

The second means of achieving data locality is across bulge chases. If ω bulges can be chased through the same set of columns without data movement, then we have achieved $O(\omega)$ reuse of those columns. Recall that we refer to columns as the subset of column entries on and below the diagonal. We first establish the following constraint.

CONSTRAINT 3.2. *No bulge may be chased into a set of columns still occupied by a previously created bulge.*

If this constraint is violated, then the fill will expand beyond the working bandwidth of the sweep. While it is possible to eliminate this extra fill, we wish to avoid the extra computation and storage necessary to do so. Chasing the first c columns of a bulge and leaving behind the triangular fill is the least amount of work required to prevent the fill from exceeding the working bandwidth.

We state the following lemmas regarding parallelograms, bulges, sets of bulges, and the working set (measured in columns) for chasing a set of bulges. We assume in both cases that Constraints 3.1 and 3.2 are satisfied.

LEMMA 3.3. *Given a sweep of SBR with parameters b, c, and d,*

(a) the j^{th} parallelogram occupies columns $1+(j-1)c$ through jc,
(b) bulge (j,k) occupies columns $1+(j-1)c+kb-d$ through $jc+kb$,
(c) bulges (j,k) and $(j+1,k-2)$ do not overlap.[2]

LEMMA 3.4. *Chasing the set of ω bulges*

$$\{(j,k),(j+1,k-2),\ldots,(j+\omega-1,k-2(\omega-1))\}$$

each ℓ times requires a working set of $(\omega-1)(2b-c)+c+d+b\ell$ columns.

PROOF. By Lemma 3.3(c), this set of bulges is nonoverlapping. If the bulges are chased in turn ℓ times each, starting with the right-most bulge (j,k) and ending with the left-most bulge $(j+\omega-1,k-2(\omega-1))$, then there is no violation of Constraint 3.2. The conclusion follows from Lemma 3.3(b). □

Figure 2 demonstrates the working set of 44 columns with $\omega=2$ bulges chased $\ell=3$ times each on a matrix with bandwidth $b=8$ with $c=d=4$.

Our motivation for defining a working set is to ensure that the operation of chasing ω bulges ℓ times can be done entirely in fast memory (in the sequential case) or local memory (in the parallel case). We will specify the constraints in each case when we present our algorithms below.

3.3. Successive Halving

We will navigate the tradeoff imposed by Constraint 3.1 by setting $c_i=d_i=b_i/2$ at each sweep i, reducing to tridiagonal form after $\log b$ sweeps. We call this a *successive halving* approach. We will pick the number of bulges in a set (ω_i) and the number of times each bulge is chased (ℓ_i) such that on each sweep (as the bandwidth is successively halved) we double the number of bulges that we chase in a set, and chase each bulge twice as many times, compared to the previous sweep. While the successive halving approach (and doubling ω_i and ℓ_i) simplifies our asymptotic analysis, in practice the parameters $\{c_i,d_i,\omega_i,\ell_i\}$ should be tuned independently for best performance — we previously suggested a framework for automatically tuning these parameters in a shared-memory implementation [Ballard et al. 2012, Section 5].

4. SEQUENTIAL BAND TRIDIAGONALIZATION ALGORITHMS

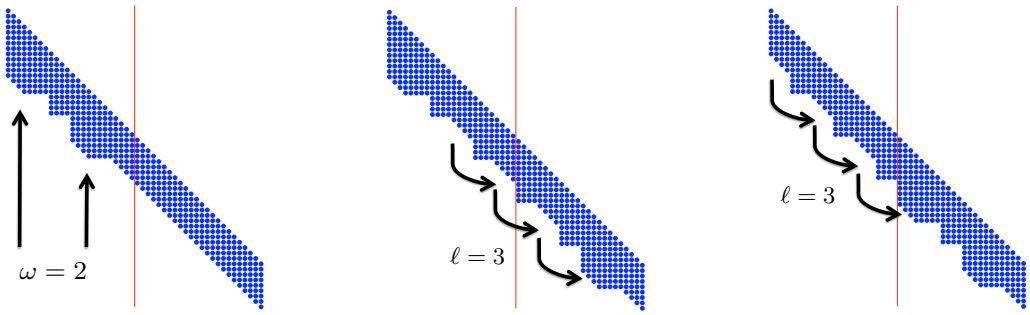
Recall our sequential machine model, where communication is moving data between slow memory of unbounded capacity and a fast memory with a capacity of M words. We will first consider the case of computing eigenvalues only and then extend to the case of computing both eigenvalues and eigenvectors. We will not analyze the solution of the tridiagonal eigenproblem. In each case, we discuss existing approaches, apply our techniques to improve them, and then present our communication-avoiding approach.

For our sequential algorithms, we will assume the initial bandwidth b is bounded above by $\sqrt{M}/3$. As mentioned in Section 2.2, this is a reasonable assumption if the band reduction is used as the second step of a two-step reduction of a full symmetric matrix to tridiagonal form. For larger bandwidths, another approach must be taken to avoid communication (see Section 6). We also assume that $nb \gg M$ (the band does not fit in fast memory).

4.1. Computing Eigenvalues Only

When only eigenvalues are desired, the runtime is dominated by the band reduction. Computing the eigenvalues of a tridiagonal matrix involves only $O(n)$ data and less computation than the band reduction — $O(n^2)$ as opposed to $O(n^2b)$. While there is a large design space for band reduction, the computational cost ranges from $4n^2b$ to $6n^2b$, a difference of only

[2]Note that if $2c+d \le b$, bulges (j,k) and $(j+1,k-1)$ also do not overlap.



(a) The $\omega = 2$ bulges occupy 20 of the 24 columns on the left.

(b) The right-most bulge is chased $\ell = 3$ times.

(c) The second bulge is chased $\ell = 3$ times.

Fig. 2. This figure demonstrates chasing a set of bulges. We store and operate on only the lower triangle of the band matrix. The parameters shown are $b = 8$, $c = 4$, and $d = 4$; $\omega = 2$ bulges are chased $\ell = 3$ times each. Only $2b\ell = 48$ columns of the band are shown. The working bandwidth includes the diagonals which contain bulges and triangular fill. Note that the triangular fill left behind by the first bulge does not cause any increase in the working bandwidth as the second bulge is chased.

Table I. We compare previous sequential algorithms for tridiagonalization (for eigenvalues only) with our improvements, for symmetric band matrices of n columns and $b+1$ subdiagonals on a machine with fast memory of size M. The table assumes that $nb \gg M$ and that $2 \leq b \leq \sqrt{M}/3$. The analysis for all algorithms is given in Section 4. In the fourth and fifth rows, s is the number of sweeps performed and $t \leq s$ is the smallest sweep index such that the subsequent sweeps can be performed in fast memory, or $t = s$ otherwise.

Algorithm	Flops	Words	Messages
LAPACK [Kaufman 2000]	$4n^2b$	$O(n^2b)$	$O(n^2b)$
MH [Murata and Horikoshi 1975]	$6n^2b$	$O(n^2b)$	$O\left(\frac{n^2b}{M}\right)$
Improved MH	$6n^2b$	$O\left(\frac{n^2b^3}{M}\right)$	$O\left(\frac{n^2b^3}{M^2}\right)$
SBR [Bischof et al. 2000b]	$\sum_{i=1}^{s}\left(4d_i + 2\frac{d_i^2}{b_i}\right)n^2$	$O\left(\sum_{i=1}^{t}\left(1+\frac{d_i}{b_i}\right)n^2\right)$	$O\left(\sum_{i=1}^{t}\left(1+\frac{d_i}{b_i}\right)\frac{n^2}{M}\right)$
SBR ($c_i = d_i = b_i/2$)	$5n^2b$	$O(n^2t)$	$O\left(\frac{n^2t}{M}\right)$
CA-SBR	$5n^2b$	$O\left(\frac{n^2b^2}{M}\right)$	$O\left(\frac{n^2b^2}{M^2}\right)$

50% (as long as a bulge-chasing procedure is used to prevent unnecessary fill). However, the communication cost (and expected performance) has a much larger range.

Under the assumption above, the matrix does not fit in fast memory (otherwise, the communication costs are the same for all algorithms: $O(nb)$). In the case that $n < M$ (i.e., one or more diagonals fit in fast memory), when the bandwidth is reduced such that the remaining band matrix fits in fast memory, the communication cost of remaining sweeps is that of reading the band into fast memory once and writing the tridiagonal output.

Table I summarizes the computation and communication costs of various algorithms for tridiagonalizing a band matrix (for computing eigenvalues only). Our new approach, CA-SBR, improves the communication costs compared to the previous approaches. For example, CA-SBR moves a factor of M/b fewer words than LAPACK or MH, which is at least $\sqrt{M}$ in the range of b considered, and near M for $b = O(1)$. In the context of two-step tridiagonalization of a dense matrix, CA-SBR is the only approach that always attains (or beats) the lower bounds discussed in Section 2.5. See Section 4.2 for more discussion.

4.1.1. Alternative Algorithms. We first consider Kaufman's algorithm [Kaufman 2000], which is implemented in the current LAPACK reference code [Anderson et al. 1992], given in the

first row of Table I. The algorithm uses Givens rotations and performs $4n^2b$ flops. It is left-looking and chases multiple single-element bulges in order to maximize the vector operation length, but it does not limit the size of the working set to fit in fast memory. As a result, the algorithm has to read (from slow memory) at least one of each pair of entries to be updated by a Givens rotation. Thus, the data reuse is $O(1)$ and the total number of words transferred between fast and slow memory is proportional to the number of flops: $O(n^2b)$. Since fine-grained data access occurs along both rows and columns, the latency cost is on the same order as the bandwidth cost, assuming LAPACK's column-major layout.

Next, we consider the Householder-based approach of Murata and Horikoshi [Murata and Horikoshi 1975], given as MH in the second row of Table I. In this algorithm, each column is eliminated all at once, and the bulge is chased completely off the band before the next column is eliminated. Because of operations on the triangular fill, the number of flops required increases to $6n^2b$ compared to Givens-based algorithms. Since each bulge is chased entirely off the band, the entire band must be read from slow memory for every column eliminated, a total of $O(n^2b)$ words moved. Assuming column-major layout, the sequence of bulge chases for each column (i.e., bulges (j, k) for fixed j) is executed on contiguous data, and the latency cost is a factor of $O(M)$ less than the bandwidth cost.

In order to reduce communication costs for the MH algorithm it is possible to apply one of the optimizations described in Section 3: chasing multiple bulges. From Lemma 3.4, we can chase $O(M/b^2)$ bulges at a time and maintain a working set which fits in fast memory. This results in a reduction of both bandwidth and latency costs by a factor of $O(M/b^2)$. We call this algorithm "Improved MH," given in the third row of Table I.

Consider an algorithm within the SBR framework with parameters $\{(b_i, c_i, d_i), i = 1, 2, \ldots, s\}$, which does not chase multiple bulges at a time (i.e., $\omega_i = 1$ for every i). This corresponds to the fourth row of Table I. The flop count is given by [Bischof et al. 2000b, Equation (3)] (and Lemma 4.6 below). Note that the approach of [Rajamanickam 2009] allows the computational cost to be reduced to $4n^2b$ for all parameter choices. Since the SBR framework is right-looking, the trailing band must be read for each parallelogram eliminated. During the i^{th} sweep, there are $O(n/c_i)$ parallelograms and each parallelogram is chased $O(n/b_i)$ times. The amount of data accessed during one bulge chase is $O(b_i(c_i + d_i))$ words — for example, b_i columns are accessed during the left update and each bulge occupies $c_i + d_i$ rows. Thus, the number of words read during the i^{th} sweep is $O(n^2(1 + d_i/c_i))$. In the best case, the latency cost is a factor of M smaller than the bandwidth cost.

If we apply the successive halving approach ($c_i = d_i = b_i/2$) but do not chase multiple bulges, then the costs of SBR simplify to $O(n^2t)$ words (where $t \leq \log b$ is the smallest sweep index such that $n(b_t + d_t + 1) \leq M$, or $t = \log b$ otherwise) and $O(n^2t/M)$ messages, in the best case. These costs appear in the fourth row of Table I.

4.1.2. CA-SBR. The communication avoiding sequential algorithm, shown in Algorithm 1, is based on the framework given in [Bischof et al. 2000b], using the successive halving approach (see Section 3.3). Our main deviation from the original SBR framework is chasing multiple bulges at a time, as described in Section 3.2. Recall that ω_i denotes the number of bulges chased at a time, and ℓ_i the number of times each bulge is chased, during sweep i. We would like to maximize ω_i so that for some $\ell_i \geq 1$, this working set fits in a fast memory of size M words. We ignore the sparsity below the b_i^{th} subdiagonal by assuming each column has $b_i + d_i + 1$ nonzeros (i.e., the working bandwidth). It follows from Lemma 3.4 that we would like to pick positive integers ω_i and ℓ_i such that ω_i is maximized and

$$((\omega_i - 1)(2b_i - c_i) + c_i + d_i + b_i\ell_i)(b_i + d_i + 1) \leq M. \tag{1}$$

We use a successive halving approach, as mentioned above. That is, at each sweep i, we cut the remaining bandwidth b_i in half by setting $d_i = b_i/2$. We also set $c_i = b_i/2$ (which

satisfies Constraint 3.1). To simplify the analysis, we assume that the initial bandwidth $b = b_1$ is a power of two.

As in Lemma 3.4, when chasing a set of ω_i bulges, we work right-to-left, chasing each bulge $\ell_i = (3/2)\omega_i$ times in turn. In this way, after all bulges in the set are chased, the set does not overlap the previous columns occupied, and the relative positions of the bulges are maintained. This process is shown in Figure 2 and corresponds to line 9 in Algorithm 1. Fixing ℓ_i in terms of ω_i also has the benefit of decreasing the latency cost on successive sweeps. While the constant ratio between ω_i and ℓ_i simplifies theoretical analysis, these parameters can be tuned independently in practice.

With these parameter choices and assumptions, inequality (1) simplifies, as given in the following constraint.

CONSTRAINT 4.1. *Assuming b and ω are even, $c = d = b/2$, $\ell = (3/2)\omega$, and $b \leq \sqrt{M}/3$, then the number of bulges chased at a time must not exceed $\omega \leq 4M/(9(b+1)^2)$.*

By satisfying Constraint 4.1, we ensure that the entire operation can be performed on columns which all fit in fast memory simultaneously.

We include explicit memory operations within the algorithm in order to determine the communication costs: *writes* imply moving data from fast memory to slow memory and *reads* imply moving data from slow memory to fast memory.

Algorithm 1 Sequential CA-SBR

Require: initial bandwidth $b \leq \sqrt{M}/3$ is a power of 2

1: $t = \min\{\log b, \left\lceil \log \frac{n(b+1)}{M} \right\rceil\}$
2: **for** $i = 1$ to t **do**
3: $b_i = \frac{b}{2^{i-1}}$, $c_i = \frac{b_i}{2}$, $d_i = \frac{b_i}{2}$, $\omega_i = 2\left\lfloor \frac{2M}{9(b_i+1)^2} \right\rfloor$, $\ell_i = \frac{3}{2}\omega_i$
4: **while** not reached end of band **do**
5: create next set of ω_i bulges
6: **while** not reached end of band **do**
7: write previous $\ell_i b_i$ columns of band
8: read next $\ell_i b_i$ columns of band
9: chase ω_i bulges ℓ_i times each
10: **end while**
11: chase ω_i bulges off the end of the band
12: **end while**
13: copy band into data structure with column height $\frac{3}{2}b_{i+1}$
14: **end for**
15: **if** $t < \log b$ **then**
16: read remaining band into fast memory
17: reduce band to tridiagonal
18: write output to slow memory
19: **end if**

We omit the details of creating a set of bulges (line 5) and of chasing bulges at the end of the band (line 11). Both the arithmetic and communication costs of creating ω_i bulges or chasing ω_i bulges off the end of the band are dominated by that of chasing the ω_i bulges ℓ_i times each. Also, since neither operation occurs in the inner loop of the algorithm, they contribute only lower-order terms to the costs of the entire algorithm.

The computation of t in line 1 determines the sweep (if any) after which the remaining band fits entirely in fast memory. Note that if $n > M$, then the band will never fit in fast memory and $t = \log b$. If the band becomes small enough to fit in fast memory, then the algorithm will stop the main loop (lines 2–14) and fall to the clean-up code in lines 15–19

which simply reads the band into fast memory, reduces to tridiagonal form, and writes the result back to slow memory.

Arithmetic cost. In order to count the number of flops required by Algorithm 1, we first establish two lemmas related to the cost of applying Householder transformations.

LEMMA 4.2. *Applying a Householder transformation from the left,* $\text{House}(u) \cdot A$*, costs no more than* $4hc+h-c$ *flops, where* h *is the number of nonzeros in* u *and* A *has* c *columns. Equivalently, applying the transformation from the right,* $A \cdot \text{House}(u)$*, costs no more than* $4hr + h - r$ *flops if* A *has* r *rows.*

PROOF. The first statement is verified by counting the operations in $A \coloneqq A - (\tau u)\left(u^T A\right)$. The second statement is verified by transposing the first transformation. □

LEMMA 4.3. *Applying a Householder transformation symmetrically to an* n*-by-*n *symmetric matrix* A*,* $\text{House}(u) \cdot A \cdot \text{House}(u)^T$*, costs no more than* $(4h-1)n+5h$ *flops, where* h *is the number of nonzeros in* u*.*

PROOF. We perform three steps: $y \coloneqq A(\tau u)$, $v \coloneqq y - (1/2)\left(y^T u\right) u$, and $A \coloneqq A - uv^T - vu^T$. The first step costs $(2h-1)n+h$ operations, the second $4h-1$, and the third $2nh$, if we exploit symmetry. □

Given these lemmas, we can compute the arithmetic cost of a single bulge chase.

LEMMA 4.4. *A single bulge chase costs* $8bcd+4cd^2+O(bc)$ *operations. Creating a bulge, or clearing a bulge (off the end of the band), is less expensive.*

PROOF. We refer to the four operations depicted in Figure 1. Let $1 \le m \le c$ index the (unblocked) Householder transformations that eliminate the parallelogram in the QR region. Transformation m is applied from the left to $c-m$ columns in the QR region and $b-c$ columns in the PRE region, from the right to $b-(c-m)$ rows in the POST region, and symmetrically to a $(d+c)$-by-$(d+c)$ symmetric matrix in the SYM region. Applying Lemmas 4.2 and 4.3, transformation m performs $8bd + 4d^2 + O(b)$ flops, and there are c transformations. Creating a bulge is less expensive because the PRE region includes only $b-c-d$ columns. As a result, transformation m does fewer flops. Clearing a bulge is less expensive because there are fewer rows in the POST region. □

See [Ballard et al. 2012, Section 5.3] for a discussion of different approaches to chasing individual bulges and their implications on performance.

We can also count the number of bulge chases that occur during each sweep.

LEMMA 4.5. *The number of bulges chased during a sweep with parameters* n*,* b*,* c*, and* d *is* $n^2/(2bc) + O(n/b)$*.*

PROOF. For each parallelogram eliminated, the bulge must be chased the length of the trailing band, in increments of b columns. Thus, the total number of bulge chases during a sweep is $\sum_{j=1}^{n/c}(n-jc)/b = n^2/(2bc) + O(n/b)$. □

Lemmas 4.4 and 4.5 together imply the following fact, which agrees with [Bischof et al. 2000b, Equation (3)].

LEMMA 4.6. *The arithmetic cost of eliminating* d *diagonals from a matrix with bandwidth* b *using SBR is* $\left(4d + 2d^2/b\right) n^2 + O(n^2)$*.*

The order of operations specified by the algorithm does not affect the arithmetic count, provided Constraints 3.1 and 3.2 are satisfied. Given the cost of the i^{th} sweep specified by Lemma 4.6, since $d_i = b_i/2$ and $\sum_i d_i = b-1$, the arithmetic cost of Algorithm 1 (ignoring

lower-order terms) is

$$\sum_{i=1}^{\log b} \left(4d_i + 2\frac{d_i^2}{b_i}\right) n^2 = 5n^2 b.$$

Bandwidth cost. In determining the communication costs of Algorithm 1, we must consider two cases. If $n > M$, then $\log b < \lceil \log(n(b+1)/M) \rceil$ and the main loop (lines 2–14) will be executed $\log b$ times, reducing the band to tridiagonal. However, if $n < M$, then at some point the bandwidth will become small enough such that the entire band fits in fast memory. At this point, the algorithm reduces to lines 15–19 and the only communication required to finish the reduction is that of reading the band into fast memory and writing the tridiagonal output back to slow memory for a cost of $O(nb_{t+1})$ words.

We now consider the i^{th} sweep, where we assume the band is too large to fit in fast memory. The dominant communication cost is in the innermost loop (lines 6–10). The number of words in each column is $(3/2)b_i$, so the bandwidth cost of one iteration of the inner loop is $3\ell_i b_i^2 = O(M)$ words. The inner loop is executed $O(n/(\ell_i b_i))$ times for each set of bulges, and there are $O(n/(c_i\omega_i))$ sets of bulges during the sweep. Thus, the bandwidth cost of one sweep is $O(n^2 b_i^2/M)$ words.

The bandwidth cost (i.e., number of words moved) of Algorithm 1 is then

$$\sum_{i=1}^{t} O\left(\frac{n^2 b_i^2}{M}\right) + O(nb_{t+1}) = O\left(\frac{n^2 b^2}{M} + nb\right).$$

Latency cost. We will assume the band matrix is stored in LAPACK symmetric band storage format (column-major with column height equal to the working bandwidth) so that any block of columns of the band will be stored contiguously in slow memory. After each set of subdiagonals is annihilated from a column block, the algorithm packs the remaining diagonals into a smaller data structure (see line 13) to maintain a packed column-major layout for all successive sweeps. This increases the memory footprint by no more than a factor of two and can also be done in place, and it adds only lower-order terms to the bandwidth and latency costs.

As in the previous section, if the band becomes small enough to fit in fast memory, then the communication costs of completing the algorithm are reduced to reading the band and writing the tridiagonal output. In this case, the latency cost is 2 messages. When the band is too large to fit in fast memory, the dominant latency cost is that of the innermost loop. Since consecutive columns are stored contiguously, the latency cost per iteration of the innermost loop is 2 messages. As argued above, the inner loop is executed $O(n/(\ell_i b_i))$ times for each set of bulges, and there are $O(n/(c_i\omega_i))$ sets of bulges during the sweep. Thus, the latency cost of one sweep is $O(n^2 b_i^2/M^2)$ messages.

The latency cost (i.e., number of messages moved) of Algorithm 1 is then

$$\sum_{i=1}^{t} O\left(\frac{n^2 b_i^2}{M^2}\right) + O(1) = O\left(\frac{n^2 b^2}{M^2} + 1\right).$$

4.2. Computing Eigenvalues and Eigenvectors

When only eigenvalues are desired, the orthogonal similarity transformations that reduce the band matrix to tridiagonal form may be discarded. However, when eigenvectors are desired, these transformations must be used to reconstruct the eigenvectors QV of the band matrix from the eigenvectors V of the tridiagonal matrix.

Compared to Section 4.1, the main difference between computing eigenvalues and additionally eigenvectors is that the arithmetic cost of computing QV increases with the number of sweeps taken in the band reduction. While the arithmetic cost of the band reduction in

Table II. We compare previous sequential algorithms for tridiagonalization (for eigenvalues and eigenvectors) with our improvements, for symmetric band matrices of n columns and $b+1$ subdiagonals on a machine with fast memory of size M. We include the cost of the back transformation (but not the cost of the tridiagonal eigendecomposition). The table assumes that $nb \gg M$ and that $2 \leq b \leq \sqrt{M}/3$. The two terms in the communication costs correspond to the band reduction and back transformation, respectively. In the last row, $t = O(\min\{\log b, \log(nb/M)\})$.

Algorithm	**Flops**	**Words**	**Messages**
LAPACK [Kaufman 2000]	$2n^3$	$O(n^2b + n^3)$	$O\left(n^2b + \frac{n^3}{M}\right)$
Improved LAPACK	$2n^3$	$O\left(n^2b + \frac{n^3}{\sqrt{M}}\right)$	$O\left(n^2b + \frac{n^3}{M}\right)$
BLS [Bischof et al. 1994]	$2n^3$	$O\left(n^2b + \frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^2b}{M} + \frac{n^3}{M}\right)$
Improved BLS	$2n^3$	$O\left(\frac{n^2b^3}{M} + \frac{n^3}{\sqrt{M}}\right)$	$O\left(\frac{n^2b^3}{M^2} + \frac{n^3}{M^{3/2}}\right)$
CA-SBR	tn^3	$O\left(\frac{n^2b}{\sqrt{M}} + \frac{tn^3}{\sqrt{M}}\right)$	$O\left(\frac{tn^2}{M} + \frac{tn^3}{M^{3/2}}\right)$

Section 4.1 ranges from $4n^2b$ to $6n^2b$, that of the back-transformation ranges from $2n^3$ up to $n^3 \log b$.

The orthogonal matrix Q can be constructed explicitly by applying the updates from the band reduction to an n-by-n identity matrix. Some flops may be saved when starting from the identity matrix (compared to applying them to a dense matrix, see e.g., [Kaufman 2000]), but the entries fill in quickly after one sweep, and we will ignore this savings in our analysis. Then, the arithmetic cost of computing QV given V is the cost of a matrix multiplication, $2n^3$ flops. However, the cost of this matrix multiplication can be avoided by storing Q implicitly as a set of Householder vectors and applying them to V. While this choice does not affect our theoretical analysis of CA-SBR, it should be considered in practice. Storing the Householder information for each sweep requires extra memory for at most $n^2/2$ entries per sweep.

Table II shows the computation and communication costs for various approaches to tridiagonalizing a band matrix (for computing both eigenvalues and eigenvectors).

Recall that one context of this work is two-step tridiagonalization; the communication lower bounds referenced in Section 2.5 apply to the first step (full-to-banded), but not the second step. However, note that a lower bound for part of the algorithm gives a valid lower bound for the whole algorithm. So, we will compare the approaches in Table II (the second step) and see which attain the lower bounds of $\Omega(n^3/\sqrt{M})$ words moved and $\Omega(n^3/M^{3/2})$ messages sent; both bounds are attainable by the first step by setting $b = \Theta(\sqrt{M})$. We claim that only CA-SBR attains these expected lower bounds for all ranges of parameters we consider, within a factor of $t = O(\log M)$.

Clearly the costs of LAPACK asymptotically exceed the lower bounds. If $n \ll M$, the bandwidth costs of the band reduction for Improved LAPACK, BLS, and Improved BLS asymptotically exceed the lower bound. If $n \gg M$, then the bandwidth costs of those three approaches match the lower bound, and the latency cost of Improved BLS also matches the lower bound.

FACT 4.7. *The cost of applying all the updates from a single band reduction sweep to a dense n-by-n matrix is $2\frac{d}{b}n^3$, ignoring lower-order terms.*

PROOF. From Lemma 4.5, there are $n^2/(2bc)$ bulge chases, each consisting of c Householder vectors of length $d+1$. From Lemma 4.2, the cost of applying each Householder transformation to an n-by-n matrix is $4(d+1)n$, so the total arithmetic cost is $4dn \cdot (n^2/(2b)) = 2(d/b)n^3$, ignoring lower-order terms. □

4.2.1. Alternative Algorithms. As mentioned in Section 4.1.1, the current LAPACK reference code for band reduction (`sbtrd`) is based on [Kaufman 2000]. When eigenvectors are re-

quested, Q can be either explicitly formed or applied to an input matrix. The LAPACK routine for solving the eigenproblem for a band matrix (`sbevd`) forms Q explicitly and premultiplies V by it. The arithmetic cost of forming Q is approximately $(4/3)n^3$ [Kaufman 2000], and the cost of the matrix multiplication is $2n^3$. In Table II, we do not count the cost of computing Q, because the Givens rotations can be stored, reordered, and later applied to V for a total of $2n^3$ flops, although LAPACK does not offer this functionality.

The communication cost of the band reduction is analyzed in Section 4.1.1. Assuming the stored Givens rotations are applied to the rows of V one at a time (which is how they are accumulated in Q in LAPACK), at least one of the rows must be read from slow memory, and the data reuse is $O(1)$. This implies that the bandwidth cost of the band reduction, which is $O(n^2b)$, is dominated by the cost of the orthogonal updates. In the best case, if V is stored in row-major order and $n > M$, the latency cost is $O(n^3/M)$.

In [Bischof et al. 1994], the authors consider an alternative approach for computing both eigenvalues and eigenvectors of a band matrix, in the context of a 2-step reduction of a full symmetric matrix. The band reduction scheme follows the algorithm of [Murata and Horikoshi 1975] consisting of one sweep (i.e., $d = b - 1$ and $c = 1$). The key idea from [Bischof et al. 1994] is to store all of the Householder vectors and, instead of applying them to V in exactly the reverse order that they were computed, to use a reordering technique that respects the dependency pattern. This reordering allows for the orthogonal updates to be blocked. See Figure 2 in [Bischof et al. 1994] or Figure 2 in [Auckenthaler et al. 2011a] for illustrations of this technique. Since the band reduction is performed in one sweep, the arithmetic cost is $2n^3$. Using the reordering technique with a blocking factor of size $\Theta(\sqrt{M})$, the communication cost of the orthogonal updates is $O(n^3/\sqrt{M})$. While the orthogonal updates are performed efficiently, the data reuse obtained during the band reduction is $O(1)$, as explained in Section 4.1.1. Thus, the bandwidth cost of the band reduction is $O(n^2b)$ which dominates the total bandwidth cost for $b \gg n/\sqrt{M}$. In the best case, the latency cost of the band reduction is $O(n^2b/M)$. In order to determine the latency cost of the orthogonal updates, we assume the matrix V is stored in column-major order and the Householder vectors are written to memory in the order they are computed. Then every application of a block of Householder vectors involves $O(\sqrt{M})$ messages, and so the latency cost is a factor of $O(\sqrt{M})$ less than the bandwidth cost. We refer to this as BLS, given in the third row of Table II.

Note that this same reordering optimization from [Bischof et al. 1994] can be used to improve the LAPACK algorithm. That is, the Givens rotations may be reordered and applied to V in a blocked fashion. For examples of implementations for applying blocks of Givens rotations, see [Rajamanickam 2009; Van Zee et al. 2013]. If the right block size is chosen, the bandwidth cost of the orthogonal updates can be reduced to $O(n^3/\sqrt{M})$. We refer to this algorithm as "Improved LAPACK," given in the second row of Table II. Because of better alternatives, we do not discuss improvements in the latency cost.

We can apply two optimizations to reduce the communication costs of the BLS algorithm. First, as noted in Section 4.1.1, when $b \ll \sqrt{M}$, the communication costs of the band reduction can be improved by chasing $O(M/b^2)$ bulges at a time, reducing both the bandwidth and latency costs by a factor of $O(M/b^2)$.

Second, we can reduce the latency cost in performing the orthogonal updates by storing the eigenvector matrix V in a block-contiguous layout with block size C-by-C with $C = \Theta(\sqrt{M})$ and by performing a data layout transformation of the temporary data structure of Householder vectors. In order to minimize bandwidth cost, the Householder vectors corresponding to eliminating $\Theta(\sqrt{M})$ columns and chasing their respective bulges off the band should be temporarily stored before applying them to Q.

In order to analyze the data layout transformation, we need to consider the temporary storage of Householder vectors. If we let H be the temporary storage matrix, then we can

store each Householder vector associated with the same eliminated column of A in the same column of H. Further, each vector can occupy the rows of H corresponding to the rows of A it updated; in this way, H is an n-by-n lower triangular matrix. If one bulge is chased at a time and Householder vectors are written to H in the order they are computed, then H will have a column-major data layout. However, in order to improve data reuse in applying the vectors to V, we want to apply parallelograms of vectors at a time, so we need those parallelograms to be stored contiguously. The data layout transformation is equivalent to transforming a matrix in column-major layout to a block-contiguous layout. By applying (for example) the `Separate` function given as Algorithm 3 in [Ballard et al. 2013] to each panel of width $\Theta(\sqrt{M})$ a logarithmic number of times, we can convert H from column-major to $\Theta(\sqrt{M})$-by-$\Theta(\sqrt{M})$ block-contiguous layout with total bandwidth cost $O(n^2 \log(n/\sqrt{M}))$ and total latency cost $O((n^2/M)\log(n/\sqrt{M}))$, which are lower-order terms for $n \gg \sqrt{M}$.

Note that these two optimizations cannot both be applied straightforwardly to the approach of [Bischof et al. 1994], as H will not be written in column-major order when multiple bulges are chased at a time. We claim that a more complicated data layout transformation is possible in the case that multiple bulges are chased at a time. This costs of this algorithm are given as "Improved BLS" in the fourth row of Table II. We also claim it is possible to apply the second optimization to the LAPACK algorithm, though the order in which the Givens rotations are computed and the method for temporarily storing them is more complicated.

4.2.2. CA-SBR. Algorithm 2 is a modification of Algorithm 1 which includes the explicit formation of the matrix Q, which we store in a block-contiguous layout with C-by-C blocks. An important difference between the two algorithms is the definition of ω_i, the number of bulges chased at a time. In Algorithm 1, ω_i is maximized under the constraint that the working set of data to chase ω_i bulges ℓ_i times each remains of size $O(M)$. In Algorithm 2, ω_i is further limited so that the working set of data while applying the Householder updates to a block row of the intermediate Q matrix remains of size $O(M)$. This working set of data now consists of three components: a subset of A, Householder transformations (temporarily stored in a data structure $\mathcal{H}$), and blocks of Q. We will pick ω_i to be approximately the square root of the previous choice so that each of these three components occupies no more than a third of fast memory. Reducing ω_i results in more communication cost during the band reduction, but we will see that this cost is always dominated by that of the orthogonal updates. One advantage of this approach is that, assuming the band is too large to fit into fast memory, Householder information is never written to slow memory: it is computed in fast memory, all updates are applied, and then the Householder entries are discarded.

In order to validate the communication pattern described in Algorithm 2, we verify three facts: $2\ell_i b_i$ columns of A fit in one third of fast memory, $\mathcal{H}$ fits in one third of fast memory, and each iteration in the OrthogonalUpdates function involves at most 3 blocks of Q, which fit in one third of fast memory. We will show that this is possible when $\omega_i \leq 2\sqrt{M}/(9(b_i+1))$, $\ell_i = (3/2)\omega_i$, $C = \sqrt{M}/3$, and the assumption from above that $b \leq \sqrt{M}/3$.

Since each column of the band has at most $(3/2)b_i + 1$ entries, the total number of words in $2\ell_i b_i$ columns is $\omega_i((9/2)b_i^2 + 3b_i) < M/3$. The $\mathcal{H}$ data structure needs to store Householder information corresponding to chasing ω_i bulges ℓ_i times each, and each bulge consists of $c_i(d_i + 1)$ entries. Thus $\mathcal{H}$ occupies $(3/8)\omega_i^2(b_i^2 + 2b_i) < M/3$ words. Finally, we must also verify that the number of columns of Q updated by the $\omega_i\ell_i$ bulge chases (which correspond to the rows of the band that are updated) cannot span more than 3 blocks of Q (i.e., one third of fast memory). By Lemma 3.3, the number of columns is $(3/2)\omega_i(b_i + 1) - b_i/2 \leq 2\sqrt{M}/3$; since $C = \sqrt{M}/3$, these columns cannot span more than 3 blocks.

Note that t is defined differently here than in Section 4.1.2. Here, since we will eliminate all subdiagonals at once, we need twice the working bandwidth to fit into fast memory.

Algorithm 2 Sequential CA-SBR with orthogonal updates

Require: initial bandwidth $b \leq \sqrt{M}/3$ is a power of 2, $Q = I_n$ is stored in contiguous C-by-C blocks, $\mathcal{H}$ is a temporary data structure of size $O(M)$ which resides in fast memory

1: $t = \min\{\log b, \left\lceil \log \frac{2n(b+1)}{M} \right\rceil\}$
2: **for** $i = 1$ to t **do**
3: $b_i = \frac{b}{2^{i-1}}$, $c_i = \frac{b_i}{2}$, $d_i = \frac{b_i}{2}$, $\omega_i = 2\left\lfloor \frac{\sqrt{M}}{9(b_i+1)} \right\rfloor$, $\ell_i = \frac{3}{2}\omega_i$
4: **while** not reached end of band **do**
5: create next set of ω_i bulges , storing Householder entries in $\mathcal{H}$
6: OrthogonalUpdates$(Q, \mathcal{H})$
7: **while** not reached end of band **do**
8: write previous $\ell_i b_i$ columns of band
9: read next $\ell_i b_i$ columns of band
10: chase ω_i bulges ℓ_i times each , storing Householder entries in $\mathcal{H}$
11: OrthogonalUpdates$(Q, \mathcal{H})$
12: **end while**
13: chase ω_i bulges off the end of the band , storing Householder entries in $\mathcal{H}$
14: OrthogonalUpdates$(Q, \mathcal{H})$
15: **end while**
16: copy band into data structure with column height $\frac{3}{2}b_{i+1}$
17: **end for**
18: **if** $t < \log b$ **then**
19: read remaining band into fast memory
20: reduce band to tridiagonal in one sweep, updating Q with improved BLS algorithm
21: write output to slow memory
22: **end if**

23: **function** OrthogonalUpdates$(Q, \mathcal{H})$
24: **for** $i = 1$ to $\frac{n}{C}$ **do**
25: read at most 3 blocks from ith block column of Q into fast memory
26: apply Householder updates stored in $\mathcal{H}$ to blocks of Q
27: write blocks of Q back to slow memory
28: **end for**
29: **end function**

Arithmetic cost. From Lemma 4.7, the arithmetic cost of the orthogonal updates is given by $2n^3 \sum_{i=1}^{t} d_i/b_i$, where t is the number of sweeps, and the cost of the band reduction is always a lower-order term. By the definition of t and the fact that $d_i = b_i/2$, the arithmetic cost is then $n^3 \min\{\log b, \lceil \log(2n(b+1)/M) \rceil\}$, ignoring lower-order terms.

Bandwidth cost. The bandwidth cost can be computed in a similar way to Section 4.1.2, though ω_i is defined slightly differently. The dominant communication cost is the call to the function OrthogonalUpdates within the innermost loop (lines 7-12). During the i^{th} sweep, the number of sets of ω_i bulges is $n/(c_i\omega_i)$, and for each set, the innermost loop is executed $O(n/(\ell_i b_i))$ times. Since $\mathcal{H}$ resides in fast memory, the bandwidth cost of the function OrthogonalUpdates is that of reading and writing the row panels of the Q matrix: $O(nC)$ words. Thus, the total bandwidth cost of Algorithm 2 is

$$\sum_{i=0}^{t} O\left(\frac{n^3}{\sqrt{M}}\right) = O\left(\frac{tn^3}{\sqrt{M}}\right).$$

Table III. We compare previous parallel algorithms for tridiagonalization (for eigenvalues only) with our improvements, for symmetric band matrices of n columns and $b+1$ subdiagonals on a machine with p processors. The first row assumes that $p \leq n/b$, and the second row assumes $p \leq n/(3b)$. The asymptotic arithmetic and communication costs are determined along the critical path.

Algorithm	Flops	Words	Messages
Lang [Lang 1993; Auckenthaler 2012]	$O\left(\frac{n^2 b}{p}\right)$	$O(nb)$	$O(n)$
CA-SBR	$O\left(\frac{n^2 b}{p}\right)$	$O(nb)$	$O(p \log b)$

Note that due to the change in definition of ω_i, the bandwidth cost of the band reduction is increased from $O(n^2b^2/M)$ (from Section 4.1.2) to $O(n^2b/\sqrt{M})$, but this higher cost is still dominated by that of the orthogonal updates.

In the case that $\lceil \log(2n(b+1)/M) \rceil < \log b$, the final step of the algorithm is to read the entire band into memory and reduce all the remaining subdiagonals at once, updating Q using the second technique of improving the BLS algorithm (i.e., transforming the column-major H matrix to block-contiguous layout). In this case, the bandwidth cost of reading A is $O(nb)$, and the cost of the orthogonal updates is $O(n^3/\sqrt{M})$ as explained above. Both of these are lower-order terms.

Latency cost. The latency cost is also dominated by that of the orthogonal updates. Since Q is stored in C-by-C contiguous blocks, the latency cost of the function OrthogonalUpdates is $O(n/C)$. Thus, the latency cost of Algorithm 2 simplifies to $O(tn^3/M^{3/2})$.

Like the bandwidth cost, the latency cost associated with the band reduction is increased by the choice of ω_i, but this higher cost of $O(tn^2/M)$ is still dominated by that of the orthogonal updates. In the case that $\lceil \log(nb/M) \rceil < \log b$, the final step of the algorithm using the improved BLS technique incurs a latency cost which is also a lower-order term.

5. PARALLEL BAND TRIDIAGONALIZATION ALGORITHMS

Recall our distributed-memory parallel model, where we have p processors connected over a network. Again, we will first discuss the case of computing eigenvalues only and then extend to the case of computing both eigenvalues and eigenvectors. The main improvement of our new algorithm over previous approaches is a reduction in latency cost, both in terms of the band reduction and the back-transformation phase (when eigenvectors are desired).

We assume that $b \leq n/(3p)$, where p is the number of processors involved in the band reduction. This is a reasonable assumption in the context of two-step tridiagonalization, in order to minimize the latency cost in the first step. For larger bandwidths, one may use fewer processors on the first sweep(s), or have multiple processors participate in a single bulge chase. The latter approach may incur a higher communication cost — see [Lang 1993].

5.1. Eigenvalues Only

In this section we concern ourselves with the case when only eigenvalues are desired, so the orthogonal updates may be discarded after applying them to the band. We collect the results from the analyses in Sections 5.1.1-5.1.2 in Table III.

5.1.1. Alternate approaches. The 'conventional' distributed memory band tridiagonalization algorithm was introduced in [Lang 1993], and has been extended several times. This is a parallelization of the MH algorithm, discussed in Section 4.1.1, a one-sweep band reduction algorithm (i.e., $d = b - 1$ and $c = 1$). We will refer to this as Lang's algorithm.

For brevity, we will not present this algorithm and its variants, but instead refer the reader to the detailed complexity analysis (and performance modeling) in [Auckenthaler 2012] (summarized in the papers [Auckenthaler et al. 2011a] and [Auckenthaler et al. 2011b]). We present their complexity results in asymptotic notation; the hidden constant factors vary

depending on the optimizations applied, including 'logical blocking,' which eliminates a factor of 2 idle time along the critical path, and using a cyclic layout, which helps alleviate load imbalance between processors. Along the critical path, their algorithm performs $O(n^2b/p)$ flops and moves $O(nb)$ words. Because there is a communication step for every column in the band, the latency cost is $O(n)$ messages.

Unless multiple bulges are chased at a time, the latency cost of $O(n)$ cannot be asymptotically reduced. That is, if a message is sent along the critical path for every parallelogram annihilated, then the last sweep, which has one parallelogram for each column, will incur $O(n)$ latency cost.

5.1.2. CA-SBR. The parallel CA-SBR algorithm begins with a similar data layout as Lang's algorithm. Each of the p processors (indexed 0 to $p-1$) owns a contiguous set of $C = n/p$ columns of the lower half of the symmetric band. We use a similar successive-halving and multiple-bulge-chasing approach to the sequential CA-SBR algorithm. During each sweep, the number of columns per processor stays fixed at $C = n/p$. We assume each of the p processors has $\Omega(nb/p)$ words of memory available, so that the band A can be stored across the machine. To simplify the presentation, we assume that $3bp$ divides n, and that b is a power of 2. This implies that $p \leq n/(3b)$, which is our maximum parallelism. These constant factors will not affect our asymptotic analysis.

Roughly, the parallel algorithm proceeds as each processor chases bulges through its C (local) columns and into the C columns of its right neighbor, and then passes the second set of columns to its right neighbor. This way, each of the p processors accesses only $O(nb/p)$ of A rather than streaming through the entire band. In the algorithm we present below, each processor is active on every other step; we can eliminate this idle time by using logical blocking (as in [Lang 1993]); we ignore this factor of 2 savings for the purposes of our asymptotic analysis.

At the high level, there are four kernels: create_bulges, pass_bulges, clear_bulges, and create_and_clear_bulges. The create_bulges kernel eliminates ω_i parallelograms (each with c_i columns and d_i diagonals) from the local set of C columns of A and chases the resulting bulges ℓ_i times (on average[3]) into the right neighbor's set of C columns. The pass_bulges kernel chases ω_i bulges (created by the left neighbor) from the local set ℓ_i times into the right neighbor's set. The create_and_clear_bulges and clear_bulges kernels are only executed by the last processor[4] and are analogous to create_bulges and pass_bulges, except the 'second set of columns' is off the end of the band. Both create_bulges and pass_bulges require $2C$ columns to pass information from one processor's columns to the next: the left set of C columns is owned by the processor invoking the kernel, and the right set is owned by the right neighbor. The create_and_clear_bulges and clear_bulges kernels require only the last C columns of the band (its local set).

At any time, a processor will have access to and update only its own C columns and the C columns from its right neighbor. For example, the parallel algorithm begins with processor 1 sending its columns to processor 0. After processor 0 executes the create_bulges kernel, it sends the updated 2nd set of C columns (with bulges) back to processor 1. Processor 1 must then also receive processor 2's C columns in order to execute the pass_bulges kernel. The parallel algorithm ends (on sweep $i = \log b$) with processor $p-1$ receiving C columns from the left, clearing all bulges, and finally eliminating the last subdiagonal of its local block (via create_and_clear_bulges).

In order for the pass_bulges kernel to pass the bulges into the right neighbor's column block, and for the bulges to retain their respective positions relative to the column blocks,

[3]Note that some bulges may need to be chased up to $2\ell_i$ times, some less.

[4]Note that processor $p-2$ may chase some bulges (partially or completely) off the end of the band when invoking pass_bulges and create_bulges, depending on the number of columns owned by processor $p-1$ and the current bandwidth.

we set $\ell_i = C/b_i$, which is an integer given the assumptions above. Recall that a bulge chase advances a bulge exactly b_i columns.

Our constraint on ω_i, the maximum number of bulges that fits in $C = n/p$ columns, is given by Lemma 3.4:

$$(\omega_i - 1)(2b_i - c_i) + c_i + d_i \leq C.$$

A little more care must be taken when creating bulges to ensure that they do not cross processor boundaries (adjacent sets of C columns). Consulting Lemma 3.3, for the successive halving approach, we arrive at the following lemma.

LEMMA 5.1. *Assuming b and ω are even, $c = d = b/2$, and $3b$ divides C, then we can create and chase $\omega = 2C/(3b)$ bulges at a time, and chasing them $\ell = C/b$ times each advances them to the next set of C columns.*

As in the sequential case, we fix the parameters to simplify the asymptotic analysis; in practice, the parameters (including the number of processors $p' \leq p$ used and the number of columns C a processor owns) should be tuned independently.

Algorithm 3 Parallel CASBR

Require: $3bp$ divides n, b is a power of 2, processor ranks are between 0 and $p-1$, each processor owns $C = \frac{n}{p}$ columns of A.

for $i = 1$ to $\log b$ **do**
  $b_i = \frac{b}{2^{i-1}}$, $c_i = \frac{b_i}{2}$, $d_i = \frac{b_i}{2}$, $\omega_i = \frac{2C}{3b_i}$, $\ell_i = \frac{3}{2}\omega_i$.
  if myrank > 0 **then**
    send left: block of C columns
  end if
  for $j = 1$ to $3 \cdot$ myrank **do**
    receive from left: block of C columns (includes bulges)
    if myrank $= p - 1$ **then**
      clear_bulges
    else
      receive right: block of C columns
      pass_bulges
      send right: block of C columns (includes bulges)
    end if
    if $j < 3 \cdot$ myrank **then**
      send left: block of C columns
    end if
  end for
  for $j = 1$ to 3 **do**
    if myrank $= p - 1$ **then**
      create_and_clear_bulges
    else
      receive right: block of C columns
      create_bulges
      send right: block of C columns (includes bulges)
    end if
  end for
end for

We analyze the arithmetic, bandwidth, and latency costs along the critical path of the algorithm. That is, we follow the progress of the first ω_1 bulges from processor 0 to processor $p-2$, at which point (exactly) one of processors $p-2$ and $p-1$ is active chasing and/or clearing bulges on every remaining step of every sweep.

Arithmetic cost. From Lemma 4.4, the arithmetic cost of chasing one bulge (a single hop), with parameters b, c, and d, is bounded above by $8bcd + 4cd^2 + O(bc)$ flops, while the cost of creating a bulge and the cost of chasing a bulge partially or completely off the band are less. For our choices $b/2^i = b_i/2 = c_i = d_i$, this cost is $(5/2)b_i^3$ flops.

Every kernel call involves at most ω_i bulges; the calls to create_bulges, pass_bulges, clear_bulges, and create_and_clear_bulges costs each chase the bulges about ℓ_i times, so each kernel invocation costs about $\omega_i \ell_i (5b_i^3/2) = O(n^2 b_i/p^2)$ flops. Following the critical path, there are (fewer than) p kernel invocations while the pipeline fills. At this point, processors $p-2$ and $p-1$ are active for the remainder of the execution, each invoking a kernel on alternating steps. There are $3p$ steps (iterations of the inner two for-loops) per sweep, each with one kernel invocation (along the critical path). Altogether, this is

$$O\left(\frac{n^2 b_1}{p}\right) + \sum_{i=1}^{\log b} O\left(\frac{n^2 b_i}{p}\right) = O\left(\frac{n^2 b}{p}\right)$$

flops. The hidden leading constant is about 20; a cyclic layout and logical blocking as in [Lang 1993] can be applied here to reduce this constant to between 5 and 10 (note these same strategies reduced the corresponding constant in Lang's algorithm's arithmetic cost from 24 to between 6 and 12).

Bandwidth cost. Every message in the algorithm consists of C columns of the band; because of bulges and triangular fill stored below the b_i^{th} subdiagonal, each message (during the i^{th} sweep) has size (at most) $C(3b_i/2+1) = O(nb_i/p)$ words. Following the critical path as before, we have the upper bound of

$$O\left(nb_1\right) + \sum_{i=1}^{\log b} O\left(nb_i\right) = O\left(nb\right)$$

words moved.

Latency cost. The latency cost analysis is similar to the bandwidth cost analysis, replacing the $O(nb_i)$ terms by $O(1)$; in total, we have $O(p \log b)$ messages. This is asymptotically smaller than the $O(n)$ messages that Lang's algorithm sends: we save a factor of $O(n/(p \log b))$ messages.

5.2. Eigenvalues and Eigenvectors

Recall our three steps: first, tridiagonalize $A = QTQ^T$; second, compute the eigendecomposition $T = V\Lambda V^T$ with an efficient algorithm; finally, back-transform the matrix V by computing QV. We may either store Q implicitly as a collection of Householder vectors, and apply it using a blocked approach, or compute Q explicitly by applying the orthogonal updates (from the band reduction) to an identity matrix, and then compute QV with a matrix multiplication. As in the sequential case, the computation and communication involved in constructing and/or applying Q dominates the costs of the band reduction.

We assume V is distributed in a 2D blocked fashion to all p processors, and that the bandwidth b of A is (at most) 1/3 of the width of a block row of V, i.e., $b \leq n/(3\sqrt{p})$. This means that we will use only $\sqrt{p}$ of the p available processors to perform the band reduction, and all p for the back-transformation. So, we must assume each processor has $\Omega(n^2/p)$ words of memory.

We collect the results from the analyses in Sections 5.2.1-5.2.2 in Table IV. Under our assumptions, for both algorithms, the arithmetic and bandwidth costs of the back-transformation always dominate those of the band reduction. The asymptotic arithmetic costs decrease linearly (in p) as expected. The first step of two-step tridiagonalization can attain the communication lower bounds for parallel dense linear algebra (without extra

Table IV. We compare previous parallel algorithms for tridiagonalization (for eigenvalues and eigenvectors) with our improvements, for symmetric band matrices of n columns and $b+1$ subdiagonals on a machine with p processors. We include the cost of the back transformation (but not the cost of the tridiagonal eigendecomposition). The first row assumes $\sqrt{p} \leq n/b$, and the second row assumes $\sqrt{p} \leq n/(3b)$. The asymptotic arithmetic and communication costs are determined along the critical path. The two terms in each cost correspond to the band reduction and the back transformation, respectively.

Algorithm	Flops	Words	Messages
Lang [Auckenthaler 2012]	$O\left(\frac{n^2 b}{\sqrt{p}} + \frac{n^3}{p}\right)$	$O\left(nb + \frac{n^2}{\sqrt{p}}\right)$	$O\left(n + \frac{n}{b}\right)$
CA-SBR	$O\left(\frac{n^2 b}{\sqrt{p}} + \frac{n^3}{p}\log b\right)$	$O\left(nb + \frac{n^2}{\sqrt{p}}\log b\right)$	$O(\sqrt{p}\log b + \sqrt{p}\log b)$

memory), i.e., $\Omega(n^2/\sqrt{p})$ words moved and $\Omega(\sqrt{p})$ messages, if $b = \Theta(n/\sqrt{p})$. Asymptotically, both algorithms attain the bandwidth lower bound, up to a factor of $\Theta(\log(n/\sqrt{p}))$ in the case of CA-SBR. However, only CA-SBR attains the latency lower bound of $\Omega(\sqrt{p})$, again up to a factor of $\Theta(\log(n/\sqrt{p}))$.

5.2.1. Alternate Approaches. The approach in [Auckenthaler et al. 2011a] stores Q implicitly (as a sequence of Householder transformations) and then applies Q to V in a blocked fashion. The authors give three algorithms to compute QV, with different parallel layouts of the matrix V — we consider only their best approach, based on a 2D layout which is dynamically rebalanced. Before computing QV, we assume each processor owns a $(n/\sqrt{p})$-by-$(n/\sqrt{p})$ block of V. Again, we refer the reader to the detailed analysis in [Auckenthaler 2012]. Along the critical path, the additional costs for the back-transformation are $O(n^3/p)$ flops, $O(n^2/\sqrt{p})$ words moved, and $O(n/b)$ messages.

5.2.2. CA-SBR. As in the sequential case (Section 4.2.2), we construct Q explicitly rather than storing it implicitly. The extra cost of the matrix multiplication QV is dominated by the cost of constructing Q and thus will not affect our asymptotic analysis. Again, in practice, this cost can be avoided by storing and applying Q to V as a sequence of Householder transformations.

By the assumption $\sqrt{p} \leq n/(3b)$, we can involve all $\sqrt{p}$ processors in each processor row in a band reduction. Since the arithmetic cost for the band reduction is a lower-order term, we can afford to perform the band reduction $\sqrt{p}$ times redundantly (or once, but only on a subset of $\sqrt{p}$ processors). We distribute the band A to each row of the given $\sqrt{p}$-by-$\sqrt{p}$ processor grid; each row performs the band reduction once. Note that each processor owns $C = n/\sqrt{p}$ columns of A, rather than n/p (as before).

We use Algorithm 4, a modification of Algorithm 3, which simultaneously computes the band reduction and the n-by-n matrix Q. That is, we postmultiply an n-by-n identity matrix I_n by each orthogonal matrix $Q_1, Q_2, \ldots$, generated by the bulge chasing procedure. (To simplify the presentation, we will again refer to the intermediate products $I_n \cdot Q_1 \cdot Q_2 \cdots$ also as Q, and the intermediate band matrices all as A.) These orthogonal updates combine columns of Q (but not rows); thus, each processor row may work independently. Each processor row is assigned C contiguous rows of Q; the columns of this block row are distributed according to the distribution of the band matrix. That is, if processor i (indexed within a given processor row) owns the first element of the j^{th} row of A, then processor i will own the j^{th} column of the corresponding block row of Q. In this way, the communication pattern of the blocks of Q between neighboring processors will exactly match the communication pattern of the blocks of the band. Whenever a processor performs a local kernel on $2C$ columns of the band, it will also apply all of those updates to $2C$ columns of (its block row of) Q. This implies that in sweep i, within each processor row, the first processor owns the first $C+b_i$ columns of the corresponding block row of Q, each subsequent processor owns the next C columns, and the last processor owns the last $C - b_i$ columns. (Note that the first processor does not touch the first $b_i/2$ columns, but rather stores them

to be updated in the next sweep.) This distribution also implies that between sweeps i and $i+1$, the Q matrix must be shifted to maintain the relationship between the ownership of rows of the band and the columns of Q. To simplify the presentation, we assume that on each sweep i, Q is padded with b_i zero columns, and that the first processor in each row always sends its rightmost C columns; under these assumptions, each processor always sends/receives C-by-C blocks of Q, avoiding fringe cases for the first and last processors (within each processor row).

We introduce four kernels — create_bulges_update, pass_bulges_update, create_and_clear_bulges_update, and clear_bulges_update — which apply the *right* orthogonal updates (as sets of Householder transformations) from the corresponding band reduction kernels to the local blocks of Q.

Again, we do not analyze computing the eigendecomposition of T, but we assume that this step terminates with V distributed across the processor grid with each processor owning a C-by-C block of V. We then compute QV using matrix-matrix multiplication.

In the following complexity analysis, we count only the additional work and communication done for the orthogonal updates. To obtain the results for CA-SBR in Table IV, we simply add the the band reduction costs (Section 5.1.2), substituting $\sqrt{p}$ for p (since now we run the band reduction redundantly). Then we add the cost of multiplying QV with Cannon's algorithm [Cannon 1969], which costs $2n^3/p$ flops, $O(n^2/\sqrt{p})$ words moved, and $O(\sqrt{p})$ messages. These are all lower-order terms, due to the logarithmic factors in the other costs.

Arithmetic cost. As argued in Section 5.1.2, there are at most $\omega_i \ell_i$ bulges chased in the pass_bulges, clear_bulges, and create_and_clear_bulges kernels, and at most $2\omega_i \ell_i$ bulges chased in the create_bulges kernel. Since the number of Householder entries in each bulge chase is $c_i d_i = b_i^2/4$, from Lemma 4.2, the cost of applying the updates from one kernel invocation to $n/\sqrt{p}$ rows of the Q matrix is at most

$$4 \cdot \frac{b_i^2}{4} \cdot \frac{n}{\sqrt{p}} \cdot \omega_i \ell_i = \frac{2n^3}{3p^{3/2}} = O\left(\frac{n^3}{p^{3/2}}\right)$$

flops (and up to 2 times more for create_bulges).

Following the analysis in Section 5.1.2, we can upper bound the additional arithmetic performed along the critical path by

$$O\left(\frac{n^3}{p}\right) + \sum_{i=1}^{\log b} O\left(\frac{n^3}{p}\right) = O\left(\frac{n^3 \log b}{p}\right)$$

flops. The costs of the band reduction and multiplication QV are lower-order terms.

Bandwidth cost. The communication costs of the orthogonal updates are also analogous to band reduction. As shown in Algorithm 4, for every message sent/received containing a block of A, there is a second message containing a block of Q. (The additional message every sweep to shift the block row of Q amounts to a lower-order term.) However, while the size of the A messages decreases with the bandwidth, the size of the Q messages remains the same (n^2/p words). The additional bandwidth cost, following the analysis in Section 5.1.2, is bounded by

$$O\left(\frac{n^2}{\sqrt{p}}\right) + \sum_{i=1}^{\log b} O\left(\frac{n^2}{\sqrt{p}}\right) = O\left(\frac{n^2 \log b}{\sqrt{p}}\right)$$

words moved. Again, the cost of the band reduction and multiplication QV are lower-order terms.

Algorithm 4 Communication-Avoiding Parallel SBR (with orthogonal updates)

Require: $3b\sqrt{p}$ divides n, b is a power of 2. Processor ranks are with respect to the processor row (i.e., between 0 and $\sqrt{p}-1$). Within each processor row, each processor stores $C = \frac{n}{\sqrt{p}}$ columns of A, and C-by-C (or C-by-$(C \pm b_i)$) block of Q, whose column indices correspond to the indices of the local rows of A whose first (leftmost) nonzero is stored locally.

1: **for** $i = 1$ to $\log b$ **do**
2: $b_i = \frac{b}{2^{i-1}}$, $c_i = \frac{b_i}{2}$, $d_i = \frac{b_i}{2}$, $\omega_i = \frac{2C}{3b_i}$, $\ell_i = \frac{3}{2}\omega_i$.
3: **if** myrank > 0 **then**
4: send left: block of C columns of A
5: send left: block of C columns and rows of Q
6: **end if**
7: **for** $j = 1$ to $3 \cdot$ myrank **do**
8: receive from left: block of C columns of A (includes bulges)
9: receive from left: block of C columns and rows of Q
10: **if** myrank $= \sqrt{p} - 1$ **then**
11: clear_bulges
12: clear_bulges_update
13: **else**
14: receive from right: block of C columns of A
15: receive from right: block of C columns and rows of Q
16: pass_bulges
17: pass_bulges_update
18: send right: block of C columns of A (includes bulges)
19: send right: block of C columns and rows of Q
20: **end if**
21: **if** $j < 3 \cdot$ myrank **then**
22: send left: block of C columns of A
23: send left: block of C columns and rows of Q
24: **end if**
25: **end for**
26: **for** $q = 1$ to 3 **do**
27: **if** myrank $= \sqrt{p} - 1$ **then**
28: create_and_clear_bulges
29: create_and_clear_bulges_update
30: **else**
31: receive from right: block of C columns of A
32: receive from right: block of C columns and C rows of Q
33: create_bulges
34: create_bulges_update
35: send right: block of C columns of A (includes bulges)
36: send right: block of C columns and rows of Q
37: **end if**
38: **end for**
39: **if** myrank $< \sqrt{p} - 1$ **then**
40: send right: block of $b_i/2$ columns and C rows of Q.
41: **else if** myrank > 0 **then**
42: receive left: block of $b_i/2$ columns and C rows of Q.
43: **end if**
44: **end for**

Latency cost. The additional latency cost is the same as that for the band reduction (see Section 5.1.2) plus the shift (a lower-order term), i.e., $O(\sqrt{p}\log b)$ messages. In the more restrictive case $\sqrt{p} \ll n/(b\log b)$, this is an asymptotic improvement compared to Lang's algorithm for just the back-transformation phase; considering also the cost of the band reduction, we always have an asymptotic improvement.

6. CONCLUSIONS

In theory, both band reduction and dense matrix-matrix multiplication have $O(n)$ possible data reuse in the sequential case, given by the ratio of total flops to size of inputs and outputs. When the problem does not fit in fast memory (of size M words), matrix multiplication can attain only $O(\sqrt{M})$ data reuse [Hong and Kung 1981], while our CA-SBR algorithm achieves $O(M/b)$ reuse, provided $b \leq \sqrt{M}/3$. This constraint on b also ensures that the reuse is always asymptotically at least as large as that of matrix multiplication, and when $b \ll \sqrt{M}$, we can actually attain much better reuse.

Indeed, improved data reuse often translates to better performance. In [Ballard et al. 2012], we observed that using the techniques of reducing communication (even at the expense of some extra arithmetic), as well as a framework that automatically tuned the algorithmic parameters, led to speedups of $2-6\times$ on sequential and shared-memory parallel machines. We believe that these benefits will extend to the distributed-memory case, particularly when performance is latency-bound.

The performance results in [Ballard et al. 2012] focused on the case of computing eigenvalues only and did not include the cost of the back-transformation phase. In that case, the arithmetic cost increased by no more than 50%. As we have seen, the cost of the back-transformation, which dominates that of the band reduction when eigenvectors are requested, increases with the number of sweeps. For example, for the successive halving approach, the increase in arithmetic was a factor of $O(\log b)$. Thus, there exists an important tradeoff between reducing communication in the band reduction phase and the resulting increased costs in the back-transformation phase. Note that when computing partial eigensystems, the costs of the back-transformation can be reduced to be proportional to the number of eigenvectors desired, improving this tradeoff.

W also do not give algorithms or complexity analysis for taking more than 1 and less than $\log b$ sweeps and using the technique of chasing multiple bulges. Indeed, we fixed many parameters in this work with the sole intention of simplifying the theoretical analysis. In practice, parameters such as the number of sweeps and the number of bulges chased at a time should be autotuned for the target architecture to navigate the tradeoffs mentioned above.

Recall our application of two-step tridiagonalization for the symmetric eigenproblem. The first step (full-to-banded) and its corresponding back-transformation phase, can be performed efficiently [Ballard et al. 2011a; Luszczek et al. 2011]. Combined with the approaches here for the second step (and an efficient tridiagonal eigensolver), we have sequential and parallel algorithms for the symmetric eigenproblem that attain the communication lower bounds for dense linear algebra in [Ballard et al. 2011b] up to $O(\log b)$ factors: in sequential, $\Omega(n^3/\sqrt{M})$ words moved and $\Omega(n^3/M^{3/2})$ messages, in parallel (if minimal memory is used), $\Omega(n^2/\sqrt{p})$ words moved and $\Omega(\sqrt{p})$ messages. Even though these lower bounds formally apply to only the first step, they are still valid lower bounds for any algorithm that performs this step.

We also remark that, in the sequential case, similar techniques to those used in a communication-optimal first step can also be applied in the case $b > \sqrt{M}/3$ (violating an assumption in Section 4). In fact, for any $b+1 \leq n$, we can reduce to tridiagonal form with communication costs that attain (or beat) the aforementioned bounds.

ACKNOWLEDGMENTS

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Also supported by U.S. DOE grants DE-SC0003959, DE-AC02-05-CH11231, and by NSF SDCI under Grant Number OCI-1032639.

REFERENCES

Aggarwal, A. and Vitter, J. 1988. The input/output complexity of sorting and related problems. *Comm. ACM 31,* 9, 1116–1127.

Agullo, E., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Langou, J., Ltaief, H., Luszczek, P., and YarKhan, A. 2009. PLASMA users' guide. http://icl.cs.utk.edu/plasma/.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1992. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, USA.

Auckenthaler, T. 2012. Highly scalable eigensolvers for petaflop applications. Ph.D. thesis, Fakultät für Informatik, Technische Universität München.

Auckenthaler, T., Blum, V., Bungartz, H.-J., Huckle, T., Johanni, R., Krämer, L., Lang, B., Lederer, H., and Willems, P. 2011a. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Computing 37,* 12, 783–794.

Auckenthaler, T., Bungartz, H.-J., Huckle, T., Krämer, L., Lang, B., and Willems, P. 2011b. Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem. *Journal of Computational Science 2,* 3, 272–278.

Ballard, G., Demmel, J., and Dumitriu, I. 2011a. Communication-optimal parallel and sequential eigenvalue and singular value algorithms. EECS Technical Report EECS-2011-14, UC Berkeley. Feb.

Ballard, G., Demmel, J., Holtz, O., and Schwartz, O. 2011b. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications 32,* 3, 866–901.

Ballard, G., Demmel, J., and Knight, N. 2012. Communication avoiding successive band reduction. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '12. ACM, New York, NY, USA, 35–44.

Ballard, G., Demmel, J., Lipshitz, B., Schwartz, O., and Toledo, S. 2013. Communication efficient gaussian elimination with partial pivoting using a shape morphing data layout. Tech. Rep. UCB/EECS-2013-12, EECS Department, University of California, Berkeley. Feb.

Bischof, C., Lang, B., and Sun, X. 2000a. Algorithm 807: The SBR Toolbox—software for successive band reduction. *ACM Trans. Math. Soft. 26,* 4, 602–616.

Bischof, C., Lang, B., and Sun, X. 2000b. A framework for symmetric band reduction. *ACM Trans. Math. Soft. 26,* 4, 581–601.

Bischof, C., Marques, M., and Sun, X. 1993. Parallel bandreduction and tridiagonalization. In *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*. Vol. 1. SIAM, 383–390.

Bischof, C. and Sun, X. 1992. A framework for symmetric band reduction and tridiagonalization. Technical Report MCS-P298-0392, Argonne National Laboratory.

Bischof, C., Sun, X., and Lang, B. 1994. Parallel tridiagonalization through two-step band reduction. In *Proceedings of the Scalable High-Performance Computing Conference*. IEEE Computer Society Press, 23–27.

Cannon, L. 1969. A cellular computer to implement the Kalman filter algorithm. Ph.D. thesis, Montana State University, Bozeman, MT.

Demmel, J., Marques, O., Parlett, B., and Vömel, C. 2008. Performance and accuracy of lapack's symmetric tridiagonal eigensolvers. *SIAM J. Sci. Comput. 30,* 3, 1508–1526.

Fuller, S. and Millett, L., Eds. 2011. *The Future of Computing Performance: Game Over or Next Level?* The National Academies Press, Washington, D.C.

Haidar, A., Ltaief, H., and Dongarra, J. 2011. Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 8.

Hong, J. and Kung, H. 1981. I/O complexity: The red-blue pebble game. In *STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, USA, 326–333.

Kaufman, L. 1984. Banded eigenvalue solvers on vector machines. *ACM Trans. Math. Soft. 10*, 73–86.

Kaufman, L. 2000. Band reduction algorithms revisited. *ACM Trans. Math. Soft. 26*, 551–567.

Lang, B. 1991. Parallele reduktion symmetrischer bandmatrizen auf tridiagonalgestalt. Ph.D. thesis, Fakultät für Mathematik, Universität Karlsruhe.

Lang, B. 1993. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM J. Sci. Comput. 14,* 6, 1320–1338.

Luszczek, P., Ltaief, H., and Dongarra, J. 2011. Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. IPDPS '11. IEEE Computer Society, Washington, DC, USA, 944–955.

Murata, K. and Horikoshi, K. 1975. A new method for the tridiagonalization of the symmetric band matrix. *Information Processing in Japan 15*, 108–112.

Rajamanickam, S. 2009. Efficient algorithms for sparse singular value decomposition. Ph.D. thesis, University of Florida.

Rutishauser, H. 1963. On Jacobi rotation patterns. In *Proceedings of Symposia in Applied Mathematics*. Vol. 15. AMS, 219–239.

Schreiber, R. and Van Loan, C. 1989. A storage-efficient wy representation for products of householder transformations. *SIAM J. Sci. Stat. Comput. 10,* 1, 53–57.

Schwarz, H. 1963. Algorithm 183: reduction of a symmetric bandmatrix to triple diagonal form. *Comm. ACM 6,* 6, 315–316.

Schwarz, H. 1968. Tridiagonalization of a symmetric band matrix. *Numerische Mathematik 12*, 231–241.

Van Zee, F., van de Geijn, R., and Quintana-Orti, G. 2013. Restructuring the qr algorithm for performance. *ACM Trans. Math. Soft.*. Submitted.

Wilkinson, J. 1962. Householder's method for symmetric matrices. *Numerische Mathematik 4,* 1, 354–361.

Received not yet; revised not yet; accepted not yet