Parallel Tridiagonalization through Two-Step Band Reduction

Christian Bischof
Xiaobai Sun

Mathematics & Computer Science Div.
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439-4844
U.S.A.

Bruno Lang

FB Mathematik
Universität Wuppertal
Gauss-Straße 20
42097 Wuppertal
Germany

Argonne Preprint ANL-MCS-P412-0194
to appear in Proc. of the 1994 Scalable High-Performance Computing Conference, Knoxville, May 1994

Abstract

We present a two-step variant of the "successive band reduction" paradigm for the tridiagonalization of symmetric matrices. Here we reduce a full matrix first to narrow-banded form and then to tridiagonal form. The first step allows easy exploitation of block orthogonal transformations. In the second step, we employ a new blocked version of a banded matrix tridiagonalization algorithm by Lang. In particular, we are able to express the update of the orthogonal transformation matrix in terms of block transformations. This expression leads to an algorithm that is almost entirely based on BLAS-3 kernels and has greatly improved data movement and communication characteristics. We also present some performance results on the Intel Touchstone DELTA and the IBM SP1.

1 Introduction

Reduction to tridiagonal form is a major step in eigenvalue computations for symmetric matrices. If the matrix is full, the conventional Householder tridiagonalization approach [9, p. 276] or a block variant thereof [8] is the method of choice. These two approaches also underlie the parallel implementations described, for example, in [12] and [6].

The approach described in this paper, on the other hand, is a two-step instantiation of the "successive band reduction" framework suggested by Bischof and Sun [4]. We first reduce the dense matrix to bandwidth nb using block orthogonal transformations employing the so-called WY representation [5]; this is described in Section 2. The remaining narrow-banded matrix is then reduced to tridiagonal form using a new variant of an algorithm originally suggested in [13]. In particular, we have devised a way of blocking the orthogonal transformations. The new algorithm is described in Section 3. The reason for considering this two-step approach is that in our experience the reduction of A is a computational bottleneck in the traditional approaches for tridiagonalization. By accomplishing the tridiagonalization in two parts, we are able to work on A more efficiently in either of the two parts. We have also devised ways for blocking the aggregation of orthogonal transformations in either of the two parts, resulting in good efficiency here as well.

2 Reduction of a Full Matrix to Narrow Banded Form

Block orthogonal transformations express the product $H_1 \cdots H_k$ of k Householder transformations $H_i = I - u_i u_i^T, u_i \in \mathbf{R}^n$ as a rank-k update of the form $I - WY^T$[5] or $I - WSW^T$[14], where $W, Y \in \mathbf{R}^{n \times k}$

This work was supported by the Applied and Computational Mathematics Program, Advanced Research Projects Agency, under contract DM28E04120, and by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

and $S \in \mathbf{R}^{k \times k}$. As a result, the application of these transformations now involves BLAS-3 operations, such as matrix-matrix multiplication, which perform efficiently on contemporary high-performance architectures [7]. What complicates the formulation of block algorithms for tridiagonalization is the fact that, if one wishes to reduce a matrix directly to tridiagonal form, the update of the next column must involve the full remainder of the matrix to be reduced, not just a part of it. To illustrate, partition

$$A = \begin{pmatrix} \alpha & b^T \\ b & C \end{pmatrix},$$

where b is a vector, and C is a matrix. To reduce b to a multiple of the canonical unit vector, one employs a Householder transformation

$$H = I - \begin{pmatrix} 0 \\ u \end{pmatrix} \begin{pmatrix} 0 \ , & u^T \end{pmatrix},$$

for properly chosen u. If one now considers the symmetric application of H to A, one sees that the update of the second column of A requires knowledge of the matrix-vector product $u^T C$. As a result, any reduction approach that directly goes to tridiagonal form must access all of the remaining data. Therefore, even block approaches to implementing tridiagonalization [8, 6] can at most halve the data transfer cost compared with the traditional approach.

On the other hand, if one reduces a full matrix to bandwidth nb, one can employ a standard one-sided QR factorization algorithm to reduce $A(nb+1:n, 1:nb)$ to upper triangular form without the need to access any other part of A, accumulate the nb Householder transformations in a block orthogonal transformation, and only then access the remainder of A. This approach reduces data accesses by a factor of nb and has been shown to perform efficiently on parallel machines [3, 2] and in out-of-core factorization approaches [10].

We have implemented this algorithm using a two-dimensional block torus wrapping. The block size nb of the block torus wrapping is also the block size used for orthogonal transformations, which are expressed with the WY representation [5]. To develop a portable code, and to allow a maintainable implementation, we chose to base our implementation on the Chameleon parallel programming tools [11].

The performance of this code is promising. Its performance on the reduction of a full random matrix to bandwidth 10, including the accumulation of orthogonal transformations, on the Intel Touchstone DELTA is shown in Figure 1. In these experiments, the matrix

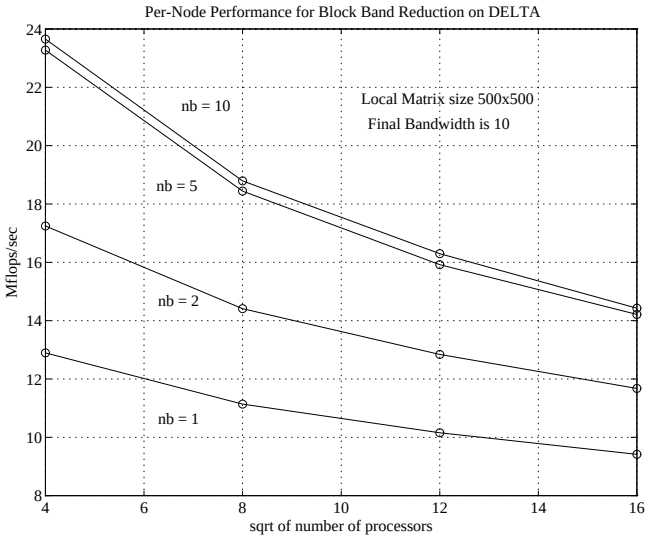

Figure 1: Per-Node Performance of Blocked Band Reduction Code on the Intel DELTA

Table 1: Preliminary Performance Results on 16-Node IBM SP1

n	nb	b	Mflops/proc.	Mflops/total
1000	1	20	10.1	162
1000	15	15	18.7	299
2000	1	20	10.0	160
2000	10	10	26.2	419
3000	1	10	10.9	174
3000	15	15	33.7	539
4000	20	20	38.2	611

size on each processor was kept constant at 500×500. The execution rates are based on the standard symmetric flop count of $(8n^3)/3$. These experiments were performed in double precision.

Some preliminary performance results on an IBM SP1, using the EUI-H transport layer, are shown in Table 1. Here n is the overall matrix size, nb is the block size used, b is the number of sub(super)diagonals the full matrix is reduced to, and the last two columns are per-processor performance and overall performance for a double-precision run, again based on the standard flop count. These results should be taken because a lower bound on the ultimate performance, as the performance of the IBM software and of Chameleon is improving rapidly. We see that the performance of the blocked code is superior to that of

the unblocked code, on both the DELTA and the SP1. For example, on the SP1, in the time that we can reduce a 2000×2000 matrix to tridiagonal form using the unblocked approach, we can reduce a 3000×3000 matrix to bandwidth 10, even though the latter reduction involves roughly 3.5 times more floating-point operations.

We note that the code we have implemented can not only reduce a full matrix to banded form but also narrow the band of a matrix that is already banded. We will report elsewhere on the performance of this transformation.

3 Reduction of a Narrow Banded Matrix to Tridiagonal Form

To complete our tridiagonal reduction, we are now faced with the task of reducing a narrow-banded matrix to tridiagonal form. To this end, Lang [13] suggested the following algorithm, which we illustrate on the 15×15 matrix with bandwidth 3 shown in Figure 2. In the first step, we generate a Householder transformation (labeled (1,1)), which reduces a(2:4,1) to a multiple of the first canonical unit vector e_1*. As indicated by the dashed lines in the band matrix, the application of this transformation generates fill-in in entries a(6,2), a(7,2), and a(7,3). The first column of this so-called bulge is eliminated by Householder transformation (1,2), which reduces a(5:7,2), generating, as indicated by the dotted lines, fill in a(9,5), a(10,5), and a(10,6). Again, we generate a transformation, labeled (1,3), to eliminate the first column of that bulge. After applying transformations (1,4) and (1,5), the first column of the bulge has been chased out. Notice that once transformation (1,2) has been completed, we can also reduce a(3:5,2) via the transformation labeled (2,1), which again generates a bulge that is chased by transformations labeled (2,2), (2,3), and (2,4).

Unlike the traditional approaches, which rely on BLAS-1 operations, this algorithm allows for the use of BLAS-2 kernels. If one desires to accumulate the orthogonal transformations, one can apply the orthogonal transformations in a matrix Q on the fly, using BLAS-2 kernels. If the size n of the matrix is much larger than the semibandwidth b, however, the accumulation of the orthogonal transformations is by far the dominant portion of the work, since a Householder update involving b rows of A requires only $O(b^2)$ flops,

*In the sequel, we assume that we always reduce vectors to multiples of e_1, and omit this fact for brevity.

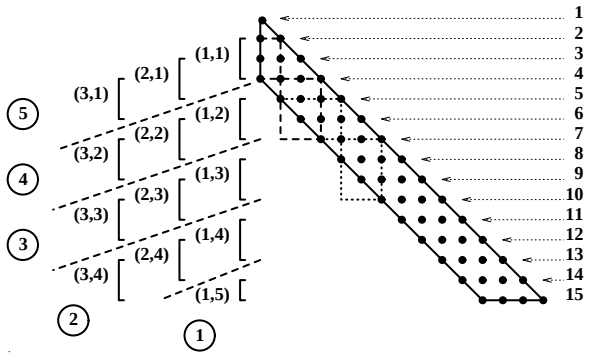

Figure 2: Illustration of the Tridiagonalization Algorithm

but induces an $O(nb)$ flops update for the corresponding columns of Q.

We have developed an algorithm that allows for a blocked update of Q by exploiting the freedom inherent in the partial ordering of the transformations. As in Figure 2, let (k,l) denote the transformation that reduces the lth bulge induced by the reduction of the kth column. In our convention the first bulge consists of the subdiagonal entries of the kth column itself. Also let the term "kth sweep" denote transformations $(k,1),(k,2),\ldots$. If we use "$<$" to denote the logical precedence of a transformation over another, then the order of generation of Householder transformations and their application to A must satisfy the

column order requirement: $(k,l) < (k+1,l)$,

same-sweep chasing requirement: $(k,l) < (k,l+1)$, and the

successive-sweep chasing requirement: $(k,l) < (k+1,l-1)$.

Once the transformations have been generated, however, we have considerably more freedom in applying these transformations to Q. First note that transformations are applied only to Q from one side, so that all transformations generated by one sweep can be applied to Q at the same time, since they involve disjoint sets of rows or columns. That is, we need not be concerned with the "same-sweep chasing requirement" any more. Since transformations must be applied to any particular column of Q in the same order as they were applied to A, we still must satisfy the column order and the successive-sweep chasing requirements. So even if we generate orthogonal transformations in the usual order (1,1), (1,2), (1,3), (1,4), (1,5), (2,1), (2,2), ..., we can apply them to Q in a different order, as long as it is consistent with the column

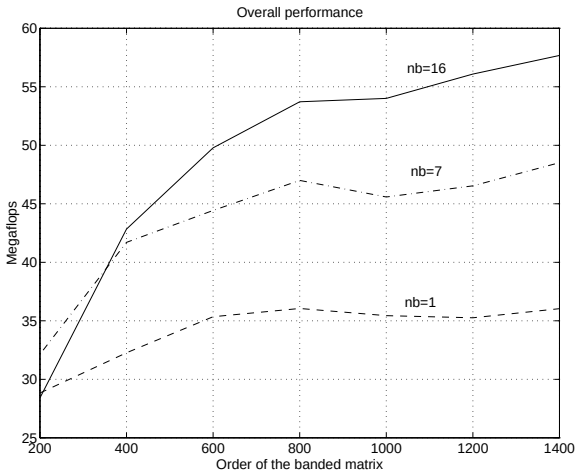

Figure 3: Tridiagonalization Performance (including Accumulation of Q) on a Single Node of the IBM SP1

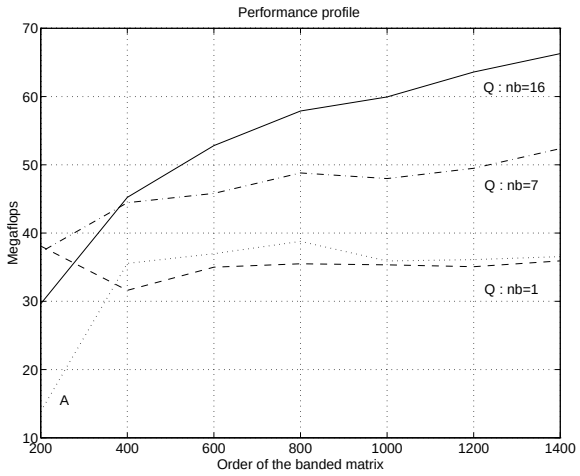

Figure 4: Performance of the Reduction and of the Update on a Single Node of the IBM SP1

order and successive-sweep chasing requirements. For example, if we assume that we reduced the first three columns, and saved the associated transformations, as shown in Figure 2, we can accumulate transformations $(1,k),(2,k)$ and $(3,k)$ into a block transformation [5, 14], and apply them "bottom-up" as indicated by the numbers in circles. The bottom-up approach is mandated by the successive-sweep chasing requirement, and the antidiagonal gathering of transformations then satisfies the column order requirement. In general, if we use $O(nb * n)$ workspace to save the Householder transformations associated with the reduction of nb columns, we can then use orthogonal transformations with blocksize nb for the update of Q. Hence, the overall fraction of work performed in this algorithm via block orthogonal transformations is $(1 - \frac{3b}{n})$, and almost all work is performed using block orthogonal transformations in the case where A is large and the band is small.

The performance of the serial code on a single node of the IBM SP1 is given in Figure 3. For these timings, the "shape" of the matrix A was fixed, or precisely, the ratio of bandwidth b to matrix size n was kept constant at 3/40. For $nb > 1$, the update of Q was done via block transformations, whereas $nb = 1$ refers to updating Q on the fly. All computations were done in double precision, and the performance data are based on the standard flop count of $6bn^2$ for the reduction of A and $2n^3$ for the update of Q. As can readily be seen, the blocking leads to a significant improvement in the overall performance.

In Figure 4 the times for reducing A and updating Q are separated. The former is not affected by the blocking. We observe that the blocked update of Q can almost run at the full speed attainable for matrix multiplication on that machine. This capability is especially important because the update of Q dominates for larger matrices. For example, for $n = 1400$ and $b = 105$, the work on Q takes about 82% of the total time.

A parallel implementation of this algorithm is in progress. In order to enhance data locality and reduce communication requirements, the narrow-banded matrix is replicated across rows of the two-dimensional logical process mesh. The orthogonal transformation matrix Q is stored in the same 2-D block torus wrap mapping as before. We also mention that, unlike the original code used in [13], we employ the same packed storage scheme as in LAPACK [1], which allows us to formulate the packed storage block algorithm more succinctly, and naturally employs BLAS-2 kernels. We also mention that in addition to increasing the computational efficiency on each node, the use of block transformations also decreases the number of communication steps of the algorithm by a factor nb, which reduces the impact of memory latency.

4 Summary

We have presented an approach for reducing a matrix to tridiagonal form through a particular instantiation of the successive band reduction (SBR) framework. By first reducing the matrix to narrow band, and then applying a new blocked variant of Lang's

tridiagonalization algorithm, we arrived at an algorithm that accomplishes the tridiagonalization and the accumulation of the orthogonal transformations almost exclusively with BLAS-3 kernels, and also reduces data access and communication costs by a factor that is proportional to the block size. The techniques presented in this paper are also directly applicable to the reduction of matrices to bidiagonal form, the usual condensed form employed for singular value decomposition algorithms.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, 1992.

[2] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, B. Tourancheau, and R. van de Geijn, *LAPACK for distributed memory architectures: Progress report*, in Parallel Processing for Scientific Computing, SIAM, Philadelphia, 1991.

[3] C. H. Bischof, *A pipelined block QR decomposition algorithm.*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., SIAM, Philadelphia, 1989, pp. 3–7.

[4] C. H. Bischof and X. Sun, *A framework for band reduction and tridiagonalization of symmetric matrices*, Preprint MCS–P298–0392, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[5] C. H. Bischof and C. F. Van Loan, *The WY representation for products of Householder matrices*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. s2–s13.

[6] J. Dongarra and R. van de Geijn, *Reduction to condensed form for the eigenvalue problem on distributed-memory architectures*, Parallel Computing, 18 (1992), pp. 973–982.

[7] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. V. der Vorst, *Solving Linear Systems on Vector and Shared-Memory Computers*, SIAM, Philadelphia, 1991.

[8] J. J. Dongarra, S. J. Hammarling, and D. C. Sorensen, *Block reduction of matrices to condensed form for eigenvalue computations*, Tech. Rep. ANL/MCS–TM–99, Mathematics and Computer Science Division, Argonne National Laboratory, September 1987.

[9] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, 1983.

[10] R. G. Grimes and H. D. Simon, *Solution of large, dense symmetric generalized eigenvalue problems using secondary storage*, ACM Transactions on Mathematical Software, 14 (1988), pp. 241–256.

[11] W. D. Gropp and B. Smith, *Chameleon parallel programming tools users manual*, Tech. Rep. ANL-93/23, Argonne National Laboratory, March 1993.

[12] B. Hendrikson and D. Womble, *The torus-wrap mapping for dense matrix calculations on massively parallel computers*, Tech. Rep. SAND92-0792, Sandia National Laboratories, 1992.

[13] B. Lang, *A parallel algorithm for reducing symmetric matrices banded matrices to tridiagonal form*, SIAM Journal on Scientific Computing, 14 (1993), pp. 1320–1338.

[14] R. Schreiber and C. Van Loan, *A storage efficient WY representation for products of Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57.