Parallel Bandreduction and Tridiagonalization*

Christian Bischof† Mercedes Marques‡ Xiaobai Sun†

To appear in the Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientfic Computing

Abstract

This paper presents a parallel implementation of a blocked band reduction algorithm for symmetric matrices suggested by Bischof and Sun. The reduction to tridiagonal or block tridiagonal form is a special case of this algorithm. A blocked double torus wrap mapping is used as the underlying data distribution and the so-called WY representation is employed to represent block orthogonal transformations. Preliminary performance results on the Intel Delta indicate that the algorithm is well-suited to a MIMD computing environment and that the use of a block approach significantly improves performance.

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 [13, p. 276] or block variants thereof [12] is the method of choice. These two approaches also underlie the parallel implementations described for example in [15] and [10].

The approach described in this paper, on the other hand, follows the band reduction framework suggested by Bischof and Sun [7]. The standard approach, which eliminates all subdiagonals at one time, is a special case, but "piecemeal" approaches are possible as well, as illustrated in Figure 1. The "piecemeal" approach was shown to be attractive in comparison to previously suggested band reduction schemes [20, 21, 18, 19] in that it allows tradeoffs between flops and storage.

*This work was supported by the Applied and Computational Mathematics Program, Defense 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.

†Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439

‡Department of Computer Science, University of Texas at Austin, Austin, TX 78712. This work was performed while the author was visiting the Mathematics and Computer Science Division of Argonne National Laboratory

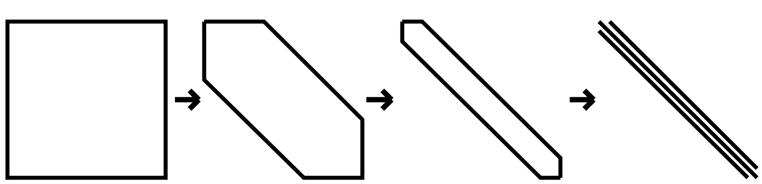

FIG. 1. *Reduction to Tridiagonal Form by a Sequence of Bandreductions*

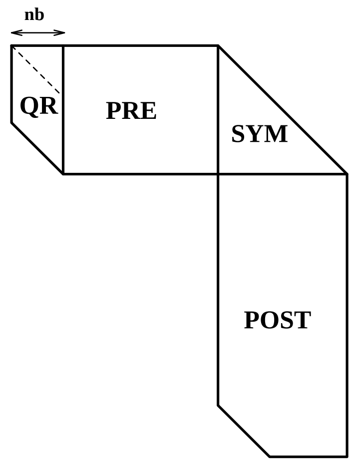

FIG. 2. *Band Reduction Primitive*

In the next section, we will briefly review the idea behind this band reduction approach, and in particular show that it allows in a very natural fashion for the use of so-called "block algorithms." In Section 3, we briefly outline our parallel implementation and in Section 4 we present preliminary performance results. Lastly, we summarize our findings.

2 A Blocked Bandreduction Algorithm

This section gives a brief introduction into the algorithmic framework suggested in [7]. For reasons of brevity, we omit tedious detail and try to convey a general understanding of the nature of the algorithm, in particular with respect to the various degrees of parallelism that it embodies.

Our algorithm employs essentially only one primitive, namely an update acting upon a submatrix of the form shown in Figure 2. It involves the following steps:

QR: The reduction of a trapezoidal matrix T (say) with nb columns to upper triangular form $\begin{pmatrix} R \\ 0 \end{pmatrix}$ via Householder reductions and the accumulation of transformations W and Y such that

$$T = (I - WY)\begin{pmatrix} R \\ 0 \end{pmatrix}.$$

W and Y are of the same size as T and are accumulated according to what is called "method 1" in [8].

PRE: The application of the block orthogonal transformation $I - WY$ to a matrix B (say) from the left, i.e. $B \leftarrow (I - WY)^T B$.

SYM: The application of $I - WY$ to a matrix C (say) from the left and the right, i.e. $C \leftarrow (I - WY)^T C (I - WY)$.

POST: The application of $I - WY$ to a matrix D (say) from the right, i.e. $D \leftarrow D(I - WY)$.

The so-called "WY representation" of a series of Householder transformations was chosen to allow for the use of matrix-matrix operations instead of matrix-vector operations in applying the orthogonal transformation. This has a very beneficial effect on architectures

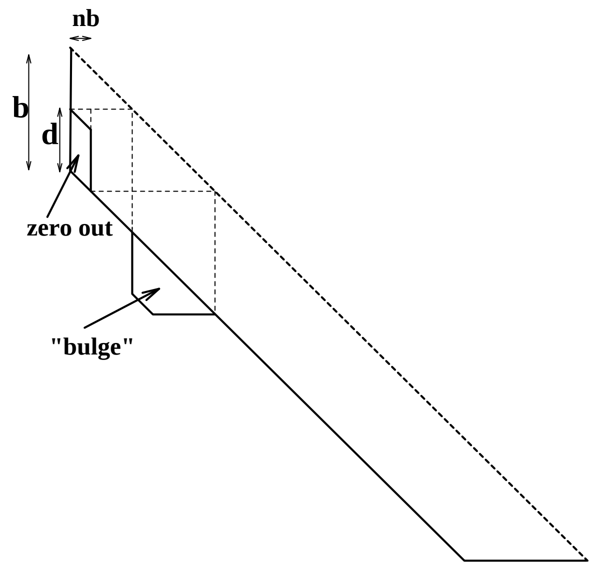

FIG. 3. *First Block Bandreduction Step.*

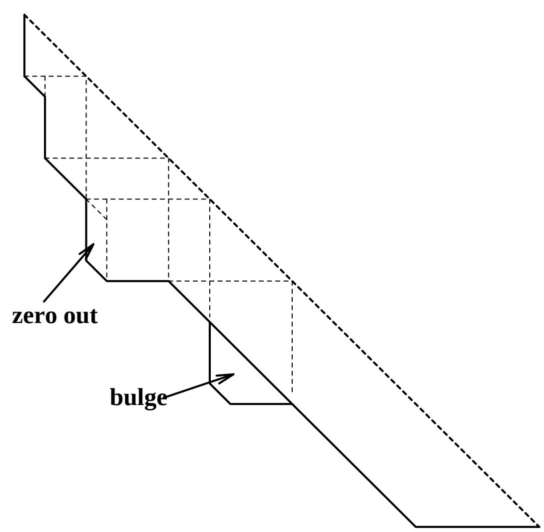

FIG. 4. *Chasing the Bulge.*

employing a memory hierarchy, since it greatly reduces the amount of data movement required [8, 4, 5, 1].

To see how this algorithmic primitive is used, let us consider an example. Figure 3 shows the reduction of d subdiagonals in nb columns of a matrix with initial bandwidth b. In this figure, as in the following ones, only the update on the lower triangle is shown. We see that in zeroing out the $d \times nb$ trapezoid, we generate a "bulge". To avoid more fill-in when zeroing out the next $d \times nb$ subdiagonal block, we now first must zero out the first nb columns of the bulge. This is shown in Figure 4. We see that in removing the first $d \times nb$ columns of the bulge, we create another bulge further down. The first $d \times nb$ columns of this bulge are then removed in the same fashion.

We see that the band reduction algorithm has two main stages, namely *band reduction* and *bulge chasing* and that the computational primitive shown in Figure 2 takes care of

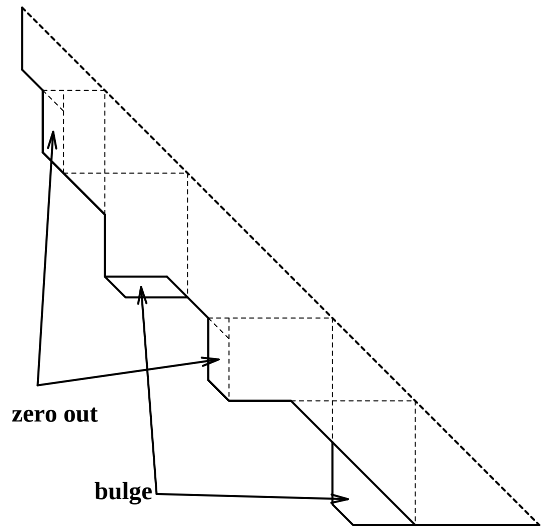

FIG. 5. *Parallelism Within one Bandreduction*

both of them.

It is also worth pointing out that the band reduction of the next $d \times nb$ subdiagonal matrix can begin before the bulge chasing associated with the first band reduction has been completed. This is shown in Figure 5. Hence, depending on the matrix size n, the initial bandwidth b, the number d of subdiagonals to be reduced, and the block size nb, there is a fair amount of potential overlap between the bulge chasing steps associated with the reduction of different $d \times nb$ subdiagonal blocks. We also note that this paradigm implicitly requires that $nb \leq b - d$, so if $d = b - 1$ (that is, we reduce the matrix directly to tridiagonal form) we are restricted to $nb = 1$.

We remark that there is another degree of parallelism if we employ this scheme in a "piecemeal" bandreduction approach as shown for example in Figure 1. To see this, note that we could start a further reduction of the band as soon as the previous bandreduction has progressed far enough to be ahead of the bulge generated by the reduction sequence initiated by the narrower band.

Lastly, we note that if the accumulation of the orthogonal transformation matrix Q is required, every orthogonal reduction of width nb necessitates the update of a $n \times nb$ strip of Q from the right. This is the same kernel as in the **POST** step.

3 A Parallel Implementation

To distribute the matrix across a distributed-memory machine, we chose a blocked two-dimensional torus wrapping (see, for example [17, 11, 10]). A scalar wrap mapping and one-dimensional (i.e. row or column oriented distributions) are special cases of this mapping. This mapping also has been selected in other efforts to develop linear algebra basis software for massively parallel machines, for example the ScaLAPACK project [9, 11]. We also assume that our underlying hardware is logically configured as a $p \times p$ mesh, and that only one process is active on every processor.

While the code relies heavily on the key primitive shown in Figure 2, a parallel implementation of this primitive for a block 2-D torus wrapped data distribution is by no means a trivial task. In particular it turns out that if one wants to exploit symmetry, the implementation of the **SYM** step may involve a noncontiguous group of processors

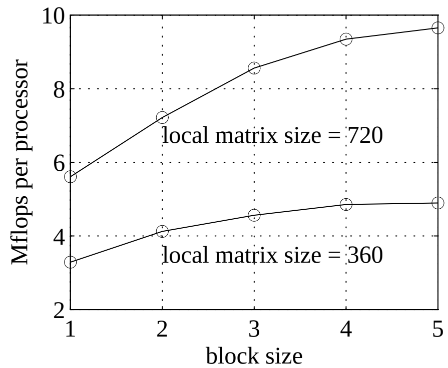

FIG. 6. *Influence of Block Size on (block) Tridiagonalization.*

on the same row or column of the mesh. Hence, to develop a portable code, and to allow a maintainable implementation of this code, we chose to base our implementation on the Chamelon parallel programming tools [14]. Chamelon's primitives (such as broadcast or global summation) support arbitrary process groups, and several such "computational contexts" may be active at any given point in time. This greatly simplifies programming, since, for example, a "broadcast" will automatically involve only the members of this computing context.

Our code is still under development, and at the moment the code for unblocked Householder tridiagonalization, blocked reduction of dense matrix to block tridiagonal form (i.e. nb subdiagonals), and blocked and unblocked band reduction are functional. We do not yet exploit symmetry, we are not yet able to reduce a block tridiagonal matrix to scalar tridiagonal form, and for simplicity we currently assume that the matrix size divides evenly by the block size. We also note that the Chamelon tools currently provide only unoptimized "fan-in/fan-out" broadcast and global sum primitives (see, for example [22, 3]) which are substantially slower than primitives that are optimized for the Intel Delta (e.g. [2, 16]). These issues will be addressed in future versions of our code.

4 Preliminary Performance Results

In this section, we present preliminary performance results that we have obtained with a double-precision version of our code running on 64 processors of the Intel Delta. The purpose of these experiments was to gain insight into the behavior of our implementation and to validate the algorithmic choices that we had made.

First, let us consider the effect that blocking has on the execution of the algorithm. Figure 6 shows the sustained per-processor execution rate that is obtained when a full matrix is reduced to banded form with nb subdiagonals, and the transformation matrix Q is not accumulated. All $n - nb$ subdiagonals are reduced at the same time, so we do not employ a "piecemeal" approach here. We see that the block approach is significantly faster, in particular for larger local matrix sizes. Hence we expect this approach to be superior, even when we add the final bandreduction from nb to 1 subdiagonals which we currently have not implemented yet. In our experience, choosing a bandwidth larger than five did not result in a further performance increase.

Next, we show the per-processor sustained performance of the (block)tridiagonal reduction of a full matrix with and without accumulating Q for various matrix sizes. We

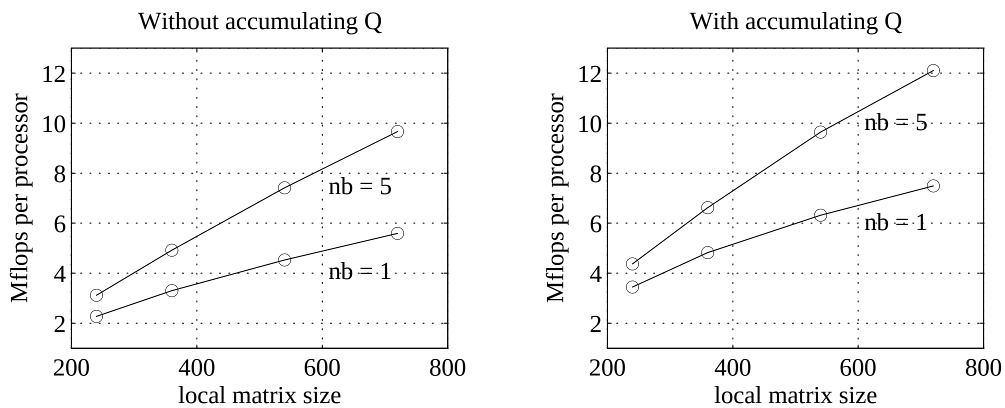

FIG. 7. *Influence of Problem Size on (block) Tridiagonalization on* 8×8 *Partition.*

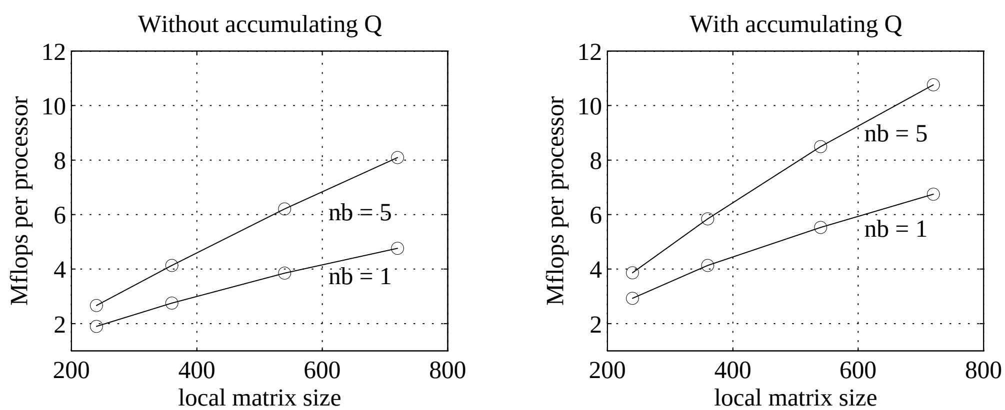

FIG. 8. *Influence of Problem Size on (block) Tridiagonalization on* 16×4 *Partition.*

see again that the blocked approach is always superior to the unblocked approach. We also note that in particular for smaller matrix sizes the accumulation of Q leads to a significant increase in the Mflop rate. This is not surprising since the accumulation of Q requires much less communication in relation to computation than the tridiagonal reduction.

Exploiting the freedom that the generality of the Chamelon system provided us, we also performed the same suite of tests on a physical 16×4 partition, which we viewed as an 8×8 logical mesh. These results are shown in Figure 8. Performance degrades somewhat, but not dramatically. These results confirm us in our decision to base our implementation on programming tools that directly support our logical view of the machine, and hide possible machine-specific improvements in that layer instead of directly embedding them in our code.

Lastly, Figure 9 shows the per-processor performance that we obtain for a "true" band reduction. In these experiments, 240 subdiagonals are removed from a matrix of order 5760 with initially 720 subdiagonals, and a matrix of order 2880 with initially 360 subdiagonals, respectively. The transformation matrix Q is not accumulated. In contrast to the previous experiments, these reductions execute the "bulge chasing" that was described in Section 2. We see that, not surprisingly, a true band reduction performs much slower than a (block) tridiagonalization, but again the choice of a blocked approach pays off.

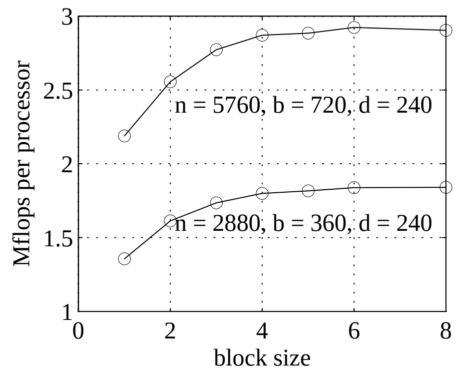

FIG. 9. *Influence of Block Size on Band Reduction*

5 Conclusions

This paper presented an implementation of a blocked band reduction framework for symmetric matrices. A blocked double torus wrap mapping is used as the underlying data distribution and the so-called WY representation is employed for the block orthogonal transformations. Preliminary performance results on the Intel Delta indicate that the algorithm is well-suited to a MIMD computing environment and that the use of a block approach significantly improves performance.

We are working to incorporate exploitation of symmetry as well as to separate the "mapping block size" used for the data layout and the "QR block size" used in the orthogonal reductions which currently are assumed to be the same. We are also working on incorporating better tuned communication routines in the Chamelon programming system that our implementation builds on. Lastly we mention that, in the end, we plan to exploit the divide-and-conquer nature of the tridiagonalization of matrices with only 0 and 1 eigenvalues [6] as it arises in the ISDA eigenvalue solver framework [17]. With the 2D torus wrapped data mapping, this approach would not significantly reduce the communication requirements of the code, but would approximately halve the number of arithmetic operations required.

Acknowledgements

The authors would like to thank Bill Gropp for his help and support with the Chamelon programming system. We also thank Steven Huss-Lederman and Anna Tsao for many helpful discussions during the course of this work.

References

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

[2] M. Barnett, R. Littlefield, D. G. Payne, and R. van de Geijn, *Efficient communication primitives on mesh architectures with hardware routing*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientfic Computing, R. Sincovec, ed., Philadelphia, 1993, SIAM.

[3] M. Barnett, D. Payne, and R. van de Geijn, *Optimal broadcasting in mesh-connected architectures*, Tech. Rep. TR-91-38, Department of Computer Science, University of Texas at Austin, 1991.

[4] C. H. Bischof, *Computing the singular value decomposition on a distributed system of vector processors*, Parallel Computing, 11 (1989), pp. 171–186.

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

[6] C. H. Bischof and X. Sun, *A divide-and-conquer method for computing complementary invariant subspaces of symmetric matrices*, Tech. Rep. MCS–P286–0192, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[7] ———, *A framework for band reduction and tridiagonalization of symmetric matrices*, Tech. Rep. MCS–P298–0392, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[8] 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.

[9] J. Demmel, J. Dongarra, R. van de Geijn, and D. Walker, *LAPACK for distributed-memory machines: The next generation*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientfic Computing, R. Sincovec, ed., Philadelphia, 1993, SIAM.

[10] J. Dongarra and R. van de Geijn, *Reduction to condensed form for the eigenvalue problem on distributed-memory architectures*, Tech. Rep. ORNL/TM-12006, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, January 1992.

[11] J. Dongarra, R. van de Geijn, and D. Walker, *A look at scalable dense linear algebra libraries*, Tech. Rep. TR CS-92-155, Computer Science Department, The University of Tennessee, May 1992.

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

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

[14] W. Gropp and B. Smith, *Chamelon: Parallel programming tools user manual*, tech. rep., Mathematics and Computer Science Division, Argonne National Laboratory, 1993. unpublished draft.

[15] 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.

[16] S. Huss-Lederman, E. Jacobson, A. Tsao, and G. Zhang, *Matrix multiplication on the intel touchstone delta*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientfic Computing, R. Sincovec, ed., Philadelphia, 1993, SIAM.

[17] S. Huss-Lederman, A. Tsao, and G. Zhang, *A parallel implementation of the invariant subspace decomposition algorithm for dense symmetric matrices*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientfic Computing, R. Sincovec, ed., Philadelphia, 1993, SIAM.

[18] L. Kaufman, *Banded eigenvalue solvers on vector machines*, ACM Transactions on Mathematical Software, 10 (1984), pp. 73–86.

[19] B. Lang, *Parallele Reduktion symmetrischer Bandmatrizen auf Tridiagonalgestalt*, PhD thesis, Universität Karlsruhe (TH), 1991.

[20] H. Rutishauser, *On Jacobi rotation patterns*, in Proc. of Symposia in Applied Mathematics, Vol. 15, Experimental Arithmetic, High Speed Computing and Mathematics, 1963, pp. 219–239.

[21] H. R. Schwarz, *Tridiagonalization of a symmetric band matrix*, Numerische Mathematik, 12 (1968), pp. 231–241.

[22] R. van de Geijn, *On global combine operations*, Tech. Rep. CS-91-129, Computer Science Department, The University of Tennessee, 1991.