A Framework for Symmetric Band Reduction

Christian H. Bischof

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439. bischof@mcs.anl.gov

and

Bruno Lang

Fachbereich Mathematik, Bergische Universität GH Wuppertal, Gaußstr. 20, D-42097 Wuppertal, Germany. lang@math.uni-wuppertal.de

and

Xiaobai Sun

Department of Computer Science, Duke University, Durham, NC 27708-0129. xiaobai@cs.duke.edu

---

We develop an algorithmic framework for reducing the bandwidth of symmetric matrices. This framework includes the reduction of full matrices to banded or tridiagonal form and the reduction of banded matrices to narrower banded or tridiagonal form, possibly in multiple steps. Our framework leads to algorithms that require fewer floatingpoint operations than do standard algorithms. In addition, it allows for space-time tradeoffs and enables or increases the use of blocked transformations.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Bandwidth reduction, orthogonal transformations, eigenvalues*; G.4 [**Mathematical Software**]

Additional Key Words and Phrases: symmetric matrices, tridiagonalization, blocked Householder transformations

---

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 (e.g., [Golub and Van Loan 1989]) or a block variant thereof [Dongarra et al. 1989] is usually considered the method of choice.

Hovever, for banded matrices this approach is not optimal if the semibandwidth b (the number of the outmost nonzero off-diagonal) is very small compared with the matrix dimension n, since the matrix being reduced has completely filled in after $n/b-1$ steps. It is well known that the algorithms of Rutishauser [1963] and Schwarz

---

This work was supported by the Advanced Research Projects Agency, under contracts DM28E04120 and P-95006. Bischof also received support from the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under contract W-31-109-Eng-38. Lang also received support from the Deutsche Forschungsgemeinschaft, Geschäftszeichen Fr 755/6-1 and La 734/2-1. This work was partly performed while X. Sun was a postdoctoral associate with the Mathematics and Computer Science Division at Argonne National Laboratory.

Argonne Preprint ANL/MCS-P586-0496, submitted to ACM Trans. Math. Software

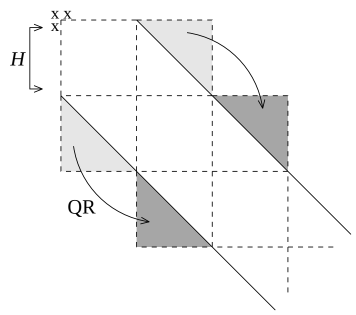

Figure 1. Rutishauser's tridiagonalization with Householder transformations.

[1968] (called RS-algorithms in the rest of the paper) are more economical than the standard approach when $b \ll n$. In these algorithms, elements are annihilated one at a time by Givens rotations; Rutishauser's algorithm annihilates the elements by diagonals, whereas Schwarz's algorithm proceeds by columns. Each Givens rotation generates a fill-in element outside of the current band, and the fill-in is chased out by a sequence of Givens rotations before more fill-in is introduced.

Rutishauser [1963] also suggested another band reduction scheme based on Householder transformations that annihilates all $b-1$ elements of the current column instead of only one. Rutishauser used an analogous scheme to chase the triangular bulge generated by the reduction with a sequence of QR factorizations, as shown in Figure 1. However, because of the significant work involved in chasing the triangular bulges, this algorithm is not competitive with the rotation-based RS-algorithms. The Schwarz algorithm is the basis of the band reduction implementations in EISPACK [Smith et al. 1976; Garbow et al. 1977]. It can be vectorized along the diagonal [Kaufman 1984], and this variant is the basis of the band reduction algorithm in LAPACK [Anderson et al. 1995].

The RS-algorithms require storage for one extra subdiagonal, and Rutishauser's Householder approach requires storage for $b-1$ extra subdiagonals. To assess the storage requirements of various algorithms, we introduce the concept of *working semibandwidth*. The working semibandwidth of an algorithm for a symmetric band matrix is the number of sub(super)diagonals accessed during the reduction. For instance, the working semibandwidth is $b+1$ for the RS-algorithms and $2b-1$ for Rutishauser's second algorithm with Householder transformations.

In both algorithmic approaches, each reduction step has two parts:

—annihilation of one or several elements, and

—bulge chasing to restore the banded form.

Either way, the bulk of computation is spent in bulge chasing. The tridiagonalization algorithm described in [Murata and Horikoshi 1975] and, for parallel computers, in [Lang 1993] (called MHL-algorithm in the following) improves on the bulge chasing strategy. It employs Householder transformations to eliminate all $b-1$ subdiagonal entries in the current column, but instead of chasing out the whole triangular bulge (only to have it reappear the next step) it chases only the first column of the bulges. These are the columns that (if not removed) would increase the working semibandwidth in the next step. The working semibandwidth for this algorithm is also $2b-1$. By leaving the rest of the bulges in place, the algorithm requires roughly the same number of floatingpoint operations (flops) as does the RS-algorithms if the latter are implemented with Givens rotations, and 50% more flops, if the RS-algorithms are based on fast Givens rotations. Using Householder transformations considerably improves data locality, as compared with the rotationbased algorithms. On the other hand, the RS-algorithms require less storage and may be still preferable if storage is tight.

In this paper, we generalize the ideas behind the RS-algorithms and the MHL-algorithm. We develop a band reduction algorithm that eliminates d subdiagonals of a symmetric banded matrix with semibandwidth b ($d < b$), in a fashion akin to the MHL tridiagonalization algorithm. Then, like the Rutishauser algorithm, the band reduction algorithm is repeatedly used until the reduced matrix is tridiagonal. If $d = b-1$, it is the MHL-algorithm; and if $d = 1$ is used for each reduction step, it results in the Rutishauser algorithm. However, d need not be chosen this way; indeed, exploiting the freedom we have in choosing d leads to a class of algorithms for banded reduction and tridiagonalization with favorable computational properties. In particular, we can derive algorithms with

(1) minimum algorithmic complexity,
(2) minimum algorithmic complexity subject to limited storage, and
(3) enhanced scope for employing Level 3 BLAS kernels through blocked orthogonal reductions.

Setting $b = n-1$ and $d = b-1$ results in the (nonblocked) Householder tridiagonalization for full matrices. Alternatively, we can first reduce the matrix to banded form and then tridiagonalize the resulting band matrix. This latter approach significantly improves the data locality because almost all the computations can be done with the Level 3 BLAS, in contrast to the blocked tridiagonalization [Dongarra et al. 1989], where half of the computation is spent in matrix-vector products. This paper extends the work of [Bischof and Sun 1992].

The paper is organized as follows. In the next section, we introduce our framework for band reduction of symmetric matrices. In Section 3 we show that several known tridiagonalization algorithms can be interpreted as instances of this framework. Then we derive new algorithms that are optimal with respect to either computational cost or space complexity. In Section 4 we discuss several techniques for blocking the update of an orthogonal matrix U; this is required for eigenvector computations. Then, in Section 5 we present some experimental results. Section 6 sums up our findings.

2. A FRAMEWORK FOR BAND REDUCTION

In this section we describe a framework for band reduction of symmetric matrices. The basic idea is to repeatedly remove sets of off-diagonals. Therefore, we first present an algorithm for peeling off some diagonals from a banded matrix.

Suppose an n-by-n symmetric band matrix A with semibandwidth $b < n$ is to be reduced to a band matrix with semibandwidth $\tilde{b} = b - d$, with $1 \leq d < b$. That is, we want to eliminate the outermost d of the b nonzero sub(super)diagonals.

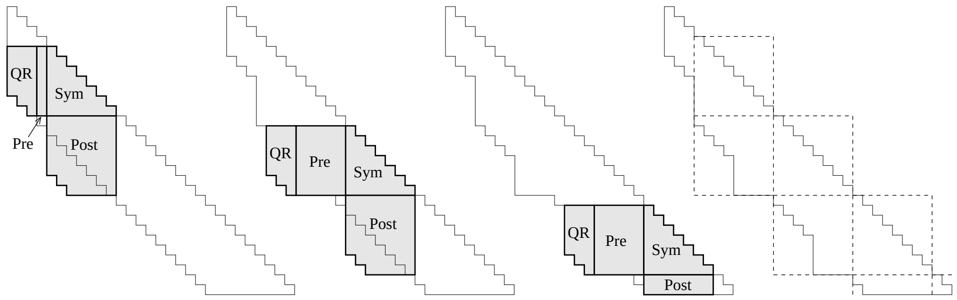

Figure 2. Annihilation and chasing in the first sweep of the band reduction algorithm. "QR" stands for performing a QR decomposition, "Pre" and "Post" denote pre- and post-multiplication with Q^T and Q, resp., and "Sym" indicates a symmetric update (Pre and Post). The last picture shows the block partitioning before the second sweep.

Because of symmetry, it suffices to access either the upper or the lower triangle of the matrix A; we will focus on the latter case.

Our algorithm is based on an "annihilate and chase" strategy, similar to the RS- and MHL-algorithms for tridiagonalization. As in the MHL-algorithm, Householder transformations are used to annihilate unwanted elements, but in the case $\tilde{b} > 1$ we are able to aggregate $n_b \leq \tilde{b}$ of the transformations into the WY or compact WY representation [Bischof and Van Loan 1987; Schreiber and Van Loan 1989]. Thus, data locality is further improved.

First, the d outmost subdiagonals are annihilated from the first n_b columns of A. This step can be done with a QR decomposition of an $h \times n_b$, $h = d + n_b$, upper trapezoidal block of A, as shown in the first picture of Figure 2. Then the WY representation $Q = I + WY^T$ of the transformation matrix is generated. To complete the similarity transformation, we must apply this block transform from the left and from the right to A. This requires applying Q from the left to an $h \times (d - n_b)$ block of A ("Pre"), from both sides to an $h \times h$ lower triangular block ("Sym"), and from the right to a $b \times h$ block ("Post").

The "Post" transformation generates fill-ins in d diagonals below the band. The first n_b columns of the fill-in are removed by another QR decomposition (second picture in Figure 2), with "Pre", "Sym", and "Post" to complete the similarity transformation. The process is then repeated on the newly generated fill-in, and so on. Each step amounts to chasing n_b columns of the fill-in down along the diagonal by b diagonal elements until they are pushed off the matrix. Then we can start the second "annihilate and chase" sweep.

Each sweep starts with a matrix that has the following properties:

—The remainder of the matrix, (i.e., the current trailing matrix) is block tridiagonal, with all diagonal blocks but the last one being of order b and the last one being of order $\leq b$; see the last picture in Figure 2.

—Every subdiagonal block is upper triangular in its first $\tilde{b} = b - d$ columns.

—The matrix is banded with semibandwidth $b + d$.

Every similarity transformation within the sweep

—maintains the working semibandwidth $b + d$,

—involves the same number h of rows and columns, and
—restores the form described above in one subdiagonal block, while destroying it in the next subdiagonal block.

We arrive at the following algorithm.

ALGORITHM 1. *One-step band reduction bandr1*(n, A, b, d, n_b)

Input. An $n \times n$ symmetric matrix with semibandwidth b, $1 < b < n$. The number d, $1 \leq d < b$, of subdiagonals to be eliminated, and a block size n_b, $1 \leq n_b \leq b - d$.

Output. An $n \times n$ symmetric matrix with semibandwidth $b - d$.

```
b̃ = b − d
for j = 1 to n − b̃ − 1 step n_b
    j1 = j; j2 = j1 + n_b − 1; i1 = j + b̃; i2 = min(j + b + n_b − 1, n)
    while i1 < n
        QR:   Perform a QR decomposition of the block B ≡ A(i1 : i2, j1 : j2)
              and replace B by (R over 0).
        Pre:  Replace the block B ≡ A(i1 : i2, j2 + 1 : i1 − 1) by Q^T B.
        Sym:  Replace the block B ≡ A(i1 : i2, i1 : i2) by Q^T BQ.
        Post: Replace the block B ≡ A(i2 + 1 : min(i2 + b, n), i1 : i2) by BQ.
        j1 = i1; j2 = j1 + n_b − 1; i1 = i1 + b; i2 = min(i2 + b, n)
    end while
end for
```

The working semibandwidth of the algorithm is $b+d$. Given the one-step band reduction algorithm, we can now derive a framework for band reduction in a straightforward fashion by "peeling off" subdiagonals in chunks.

ALGORITHM 2. *Multistep band reduction bandr*$(n, A, b, \{d^{(i)}\}, \{n_b^{(i)}\})$

Input. An $n \times n$ symmetric matrix with semibandwidth b, $1 < b < n$. A sequence of positive integers $\{d^{(i)}\}_{i=1}^{k}$ with $d = \sum d^{(i)} < b$, and a sequence $\{n_b^{(i)}\}_{i=1}^{k}$ of block sizes, where $n_b^{(i)} \leq \tilde{b}^{(i)} = b - \sum_{j=1}^{i} d^{(i)}$.

Output. An $n \times n$ symmetric matrix with semibandwidth $b - d$.

```
b^(1) = b
for i = 1 to k
    call bandr1( n, A, b^(i), d^(i) )
    b^(i+1) = b^(i) − d^(i)
end for
```

The working semibandwidth for the multistep algorithm is

$$\max_{1 \leq i \leq k} (b^{(i)} + d^{(i)}) = b + \max_{1 \leq i \leq k} (d^{(i)} - \sum_{j=1}^{i-1} d^{(j)}),$$

which is $b + d^{(1)}$ if the $d^{(i)}$ satisfy

$$d^{(i)} \leq d^{(1)} + \sum_{j=1}^{i-1} d^{(j)}, \quad 1 < i \leq k. \tag{1}$$

There is also a class of systolic array algorithms that use Givens rotations to remove several outer diagonals at a time [Bojanczyk and Brent 1987; Ipsen 1984; Schreiber 1990]. Here, $d^{(i)}$ is chosen as large as $b^{(i)}/2$ [Bojanczyk and Brent 1987] in order to increase the parallel scope per systolic operation and hence minimize the total number of systolic operations.

3. INSTANCES OF THE FRAMEWORK

In this section we discuss several instances of Algorithm 2, including the (nonblocked) Householder tridiagonalization, Rutishauser's algorithm, and the MHL-algorithm for tridiagonalizing symmetric band matrices. In addition to these, the multistep framework allows for new tridiagonalization methods featuring lower flops count and/or better data locality.

3.1 One-step Tridiagonalization of Full Symmetric Matrices

A full symmetric matrix can be tridiagonalized with $bandr(n, A, b = n-1, \{d^{(1)} = b-1\}, \{n_b^{(1)} = 1\})$. Note that $\tilde{b} = 1$ implies $n_b = 1$. Then, the QR decomposition and the Sym step of Algorithm 1 reduce to determining and applying a suitable Householder transformation, while the Pre and Post steps vanish. Thus, we arrive at the *nonblocked* standard Householder tridiagonalization.

3.2 Two-step Tridiagonalization of Full Symmetric Matrices

Another way to tridiagonalize a full symmetric matrix is the two-step sequence $bandr(n, A, b = n-1, \{d^{(1)} = b - \tilde{b}; d^{(2)} = \tilde{b} - 1\}, \{n_b^{(1)}; n_b^{(2)} = 1\})$, where $1 < \tilde{b} < b$ is some intermediate semibandwidth and $n_b^{(1)} \leq \tilde{b}$. That is, we first reduce A to banded form and then tridiagonalize the resulting banded matrix.

In contrast to the blocked Householder tridiagonalization [Dongarra et al. 1989], where one half of the approximately $4/3n^3$ flops for the reduction of A is confined to matrix-vector products, almost all the operations in the reduction to banded form can be done within the Level 3 BLAS. For $b \ll n$ this first reduction constitutes the vast majority of flops. Tridiagonalizing the banded matrix requires roughly $6bn^2$ flops, which can be done with Level 2 BLAS.

Therefore, the two-step tridiagonalization may be superior on machines with a distinct memory hierarchy (see Table I in Section 5.1).

3.3 One-step Tridiagonalization of Symmetric Band Matrices

As for full matrices, the simplest way to tridiagonalize a matrix with semibandwidth b is the one-step sequence $bandr(n, A, b, \{d^{(1)} = b-1\}, \{n_b^{(1)} = 1\})$. Again, the QR decomposition and the Sym steps reduce to determining and applying single Householder transformations, and the Pre and Post steps vanish. This one-step sequence is equivalent to the MHL-algorithm.

3.4 Tridiagonalization by Peeling off Single Diagonals

The sequence $bandr(n, A, b, \{d^{(1)} = \ldots = d^{(b-1)} = 1\}, \{n_b^{(1)} = \ldots = n_b^{(b-1)} = 1\})$ tridiagonalizes a banded matrix by repeatedly peeling off single diagonals. This corresponds to Rutishauser's algorithm, except that the rotations are replaced with length-2 Householder transformations.

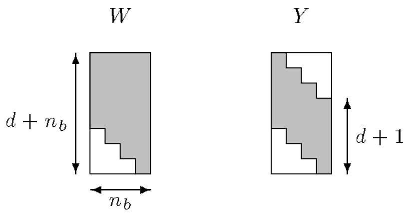

Figure 3. Nonzero structure of the matrices W and Y for $d = n_b = 4$.

3.5 Optimal Tridiagonalization Algorithms for Band Matrices

In the following, we derive tridiagonalization algorithms that have a minimum flops count for a given working semibandwidth, that is,

$$\begin{array}{rl} \textbf{minimize} & \text{number of flops to tridiagonalize an } n \times n \text{ banded matrix} \\ & \text{with semibandwidth } b \\ \textbf{subject to} & \text{working semibandwidth} \leq s, \text{ where } s \geq b+1. \end{array} \tag{2}$$

For the reduction of banded matrices, blocking the transformations always significantly increases the flops count. Figure 3 shows the nonzero pattern of the $(d+n_b)\times n_b$ matrices W and Y that result from aggregating n_b length-$(d+1)$ Householder transformations into a blocked Householder transformation. The nonzeros in Y form a parallelogram, while W is upper trapezoidal. Multiplying W to the columns of some $m \times (d+n_b)$ block B of A takes approximately $n_b(2d+n_b+1)m$ flops and multiplication with Y^T costs another $2n_b(d+1)m$ operations, *even if the multiplication routine _GEMM is able to take full advantage of zeros.* Thus, in the banded context, using the WY representation increases the overall flops count in two ways. First, the W and Y factors must be generated. Second, applying the blocked Householder transform requires more than the $4n_b(d+1)m$ operations that would be needed for the n_b single length-$(d+1)$ Householder transformations in W and Y.

The same argument applies also to the compact WY representation $Q = I + YTY^T$ [Schreiber and Van Loan 1989]. In addition, this blocking technique requires one more matrix multiplication, $W = YT$. The cost for this additional multiplication is not negligible in the banded case, in contrast to the reduction of full matrices, where the average length of the Householder vectors significantly exceeds n_b. Therefore, we prefer the "standard" WY representation for blocking the Householder transformations.

Because this section focuses on minimizing the flops count, we consider only nonblocked reduction algorithms (i.e., $n_b^{(i)} = 1$ for all i). On machines with a distinct memory hierarchy, however, the higher performance of the Level 3 BLAS may more than compensate for the overhead introduced by blocking the transformations. Thus, both flops count and BLAS performance should be taken into account in order to minimize the *execution time* on such machines. For clarity in our analysis, we omitted the resulting weighting of the various algorithmic components, but the ideas presented here easily can be extended to this more general analysis.

In the nonblocked case, the arithmetic cost of Algorithm 1 in flops is

$$\mathrm{cost}(n,b,d) \approx (4(d+1) + (2d^2+7d+5.5)/b) \times (n(n-1) - (b-d)(b-d+1)) \ . \tag{3}$$

For any band difference sequence $\{d^{(i)}\}$, Algorithm 2 therefore requires

$$\sum_{i=1}^{k} \mathrm{cost}(n, b^{(i)}, d^{(i)})$$

flops.

Given a limit on s, we can use dynamic programming [Aho et al. 1983] to determine an optimal sequence $\{d^{(i)}\}$ from the cost function given in (3). By taking storage requirements into account, we allow for space tradeoffs to best use the available memory.

3.5.1 *No Storage Constraints.* We first consider Problem (2) with $s \geq 2b - 1$. Here are no storage constraints, since the maximum working semibandwidth of Algorithm 2 is $2b - 1$. However, the optimal sequence $\{d^{(i)}\}$ is quite different from the one-step sequence of the MHL-algorithm.

For example, for a $50,000$-by-$50,000$ symmetric matrix with semibandwidth 300, the optimal sequence is

$$\{10, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26\},$$

the reduction requires $3.48 \cdot 10^{12}$ floatingpoint operations, and the working semibandwidth is 310. In contrast, the MHL-algorithm requires $4.47 \cdot 10^{12}$ flops, and its working semibandwidth is 599. We also note that the constant sequence

$$\{16, 16, \ldots, 16, 11\}$$

requires $3.49 \cdot 10^{12}$ flops, with a working semibandwidth of 316. Another constant sequence

$$\{32, 32, \ldots, 32, 11\}$$

requires $3.56 \cdot 10^{12}$ flops, with a working semibandwidth of 332. Hence a *constant-stride* sequence seems to be just as good a choice as the optimal one from a practical point of view, and saves the dynamic programming overhead.

3.5.2 *Minimum Storage.* Now consider another extreme case of the constraint in Problem (2): $s = b + 1$. That is, we have space for at most one other subdiagonal. Even for small b, the MHL-algorithm is not among the candidates, since its working semibandwidth is $2b - 1 > b + 1$. Indeed, a candidate d-sequence for this case should satisfy the condition (1) with $d^{(1)} = 1$. The sequence $d^{(i)} = 1$ from Section 3.4 satisfies this condition. Instead, we suggest

$$\begin{aligned} d^{(1)} &= 1, \\ d^{(i+1)} &= 2d^{(i)}, \qquad\qquad 1 \leq i < k = \lfloor \log_2(b-1) \rfloor \\ d^{(k+1)} &= (b-1) - \sum_{i=1}^{k} d^{(i)}. \end{aligned} \tag{4}$$

We call it the *doubling-stride sequence*, since the bandwidth reduction size doubles in each round. For the example in the preceding subsection, the RS-algorithms require $4.49 \cdot 10^{12}$ flops, whereas the doubling-stride algorithm requires $3.90 \cdot 10^{12}$ flops.

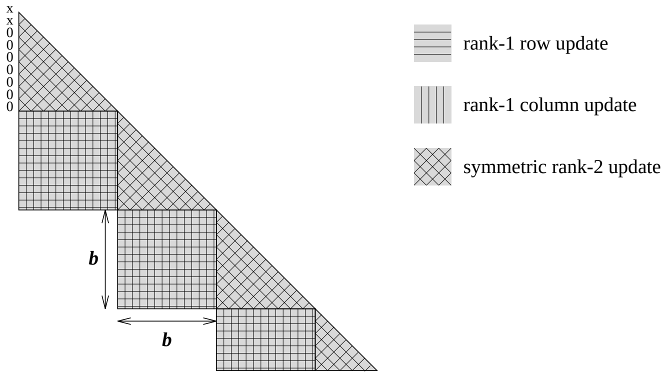

Figure 4. Data area visited by the MHL-algorithm.

3.5.3 *Understanding Optimality.* Rutishauser's algorithm peels off subdiagonal one by one and requires minimum storage, but it is not optimal among the algorithms with minimum storage. On the other hand, although the MHL-algorithm eliminates all subdiagonals in one step, it is also not optimal when b is large. In the following, we give an intuitive explanation why the complexity of our scheme is superior to both these approaches.

First, let us compare the MHL-algorithm and Algorithm 2 with a sequence $\{d, b-d-1\}$. During the bulge chasing following the reduction in column 1, the data area accessed by the MHL-algorithm with rank-1 row and/or column updating and symmetric rank-2 updating is shown in Figure 4. The data area accessed by Algorithm 2 is shown in Figure 5. In comparing different bandreduction sequences, we take the rank-1 updating as basic unit of computation and examine the total data area visited by each algorithm for the purpose of rank-1 updating. Since a rank-1 update applied to an $m \times n$ matrix requires approximately $4mn$ flops, the area involved in a rank-1 update is a good measure of complexity. The symmetric rank-2 update of a triangular $n \times n$ matrix is as expensive as a rank-1 update of a full matrix of that size; both require about $4n^2$ flops.

For the MHL-algorithm, the total area visited with rank-1 updates is

$$a_{\mathrm{MHL}} = 3nb.$$

For the two successive band reductions, it is

$$a_{\mathrm{two-step}} = \{2n(d+1) + \frac{n}{b}(d+1)^2\} + 3n(b-d).$$

Denoting $\delta = (d+1)/b$, we obtain as the ratio

$$\rho = \frac{a_{\mathrm{two-step}}}{a_{\mathrm{MHL}}} = \frac{\delta^2 - \delta + 3(1+\frac{1}{b})}{3},$$

which takes its minimum $\rho^* \approx 11/12$ at $\delta^* = 1/2$. Therefore, the two-step reduction can save some 8% of the flops, as compared with the MHL-algorithm. Of course, the same idea can then be applied recursively on the tridiagonal reduction for the matrix with reduced semibandwidth $b-d$.

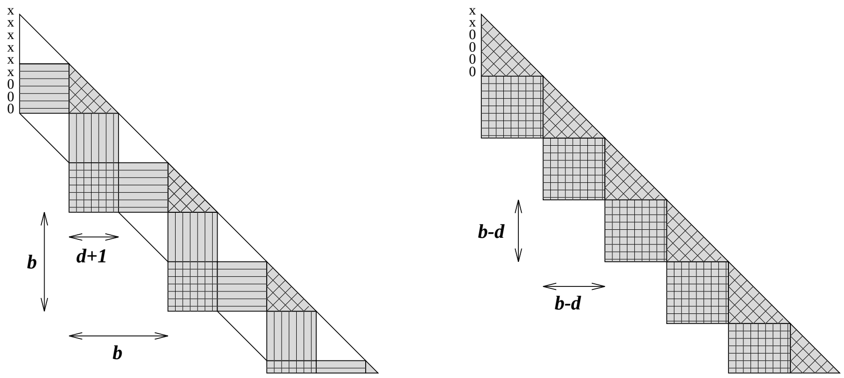

Figure 5. Data area visited in the two steps bandr1(n, A, b, d) and bandr1(n, A, $b-d$, $b-d-1$) of Algorithm 2.

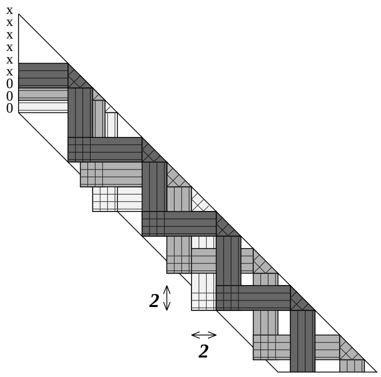

Figure 6. Data area visited by Rutishauser's algorithm in the "elimination and bulge chasing" rounds for the three outmost elements a_{91}, a_{81}, and a_{71} of the first column (light, medium, and dark grey shading, resp.).

Let us consider now how often a row or column is repeatedly involved in a reduction or chasing step. The data area visited by Rutishauser's algorithm for annihilating 3 elements in the first column in 3 rounds is shown in Figure 6. We see in particular that the total area visited in the first $b \times b$ block is almost twice as big as the one by Algorithm 1 with $d = 3$, as a result of the revisiting of the first row and column in the last visited area.

Thus, in comparison with Rutishauser's and the MHL-algorithm, our algorithm can be interpreted as balancing the counteracting goals of

—decreasing the number of times a column is revisited (by the use of Householder transformations), and

—decreasing the area involved in updates (by peeling off subdiagonals in several chunks).

3.6 Bandwidth Reduction

In some contexts it is not necessary to fully reduce the banded matrix to tridiagonal form. For example, in the invariant subspace decomposition approach (ISDA) for eigensystem computations [Lederman et al. 1991], the spectrum of a matrix A is condensed into two narrow clusters by repeatedly applying a function f to the matrix. If f is a polynomial of degree 3, each application of f roughly triples A's bandwidth. Therefore, A will be full after a moderate number of iterations if no countermeasures are taken. To prevent this situation, the bandwidth of A can be periodically reduced to a "reasonable" value after a few applications of f.

This bandwidth reduction can also be done by using Algorithm 2, either in one or multiple reduction steps.

4. AGGREGATING THE TRANSFORMATIONS

If the eigenvectors of the matrix A are required, too, then all the transformations from Algorithm 1 must also be applied to another matrix, say, an $n \times n$ matrix U. For banded matrices A, the update of U dominates the floating-point complexity ($2n^3$ for updating U versus $6bn^2$ for the reduction of A). Therefore, we should strive to maximize the use of BLAS 3 kernels in the update of U to decrease the total time.

In this section we discuss several techniques for updating U with blocked Householder transformations. Some of these methods are tailored to the cases $\tilde{b} = 1$, where the transformations of A cannot be blocked.

4.1 On-the-Fly Update

The update can easily be incorporated in the reduction by inserting the lines

> if the aggregate transformation matrix U is required
>   Replace $U(:, i_1 : i_2)$ by $U(:, i_1 : i_2)Q$.

between the **QR** and **Pre** steps of Algorithm 1. That is, each transformation is applied to U as soon as it is generated and applied to A (*on-the-fly update*).

If the matrix is not reduced to tridiagonal form, i.e., $\tilde{b} = b - d > 1$ then the work on A and U can be done with blocked Householder transformations, thus enabling the use of the Level 3 BLAS.

4.2 Backward Accumulation

For tridiagonalization, however, $\tilde{b} = 1$ implies $n_b = 1$, which means that *the work on A cannot be done in a blocked fashion.*

At first glance, this fact seems also to preclude the use of Level 3 BLAS in the update of U. Fortunately, however, the obstacle can be circumvented by decoupling the work on U from the reduction of A.

Let us first consider the tridiagonalization of a full symmetric matrix A. As in the LAPACK routine `_SYTRD`, the update of U can be delayed until the reduction of A is completed. Then, the Householder transformations are aggregated into blocked Householder transforms with arbitrary n_b^U, and these are applied to U. In addition to enabling the use of the Level 3 BLAS, the decoupling of the update from the reduction allows reducing the flop count by reverting the order of the transformations (*backward accumulation*).

For complexity reasons, the backward accumulation technique should also be used in the reduction from full to banded form, even if the on-the-fly update can also be done with blocked Householder transforms.

4.3 Update in the Tridiagonalization of Banded Matrices

When a full matrix is reduced to either banded or tridiagonal form, the Householder vectors can be stored conveniently in the zeroed-out portions of A and an additional vector τ. In the tridiagonalization of banded matrices, this strategy is no longer possible: eliminating one length-$(b-1)$ column of the band requires n/b length-b Householder vectors, because of the bulge chasing. Therefore, it is impractical to delay the update of U until A is completely tridiagonalized. We use another technique [Bischof et al. 1994] in this case.

Let H_k^j denote the kth Householder transformation that is generated in the jth sweep of Algorithm 1. That is, H_1^j eliminates the first column of the remaining band, and H_2^j, H_3^j, … are generated during the bulge chasing.

During the reduction, the transformations of each sweep must be determined and *applied to* A in the canonical order H_1^j, H_2^j, H_3^j, …, because each H_k^j depends on data modified in the Post step of H_{k-1}^j. Once the transformations are known, this dependence no longer exists. Since the transformations from one sweep involve disjoint sets of U's columns, they may be applied to U in any order (see Figure 7). We are, however, not entirely free to mix transformations from different sweeps. H_k^j must be preceded by H_{k-1}^{j-1} and H_k^{j-1}, since it affects columns that are modified by these two transformations in sweep $j-1$ (*intersweep dependence*).

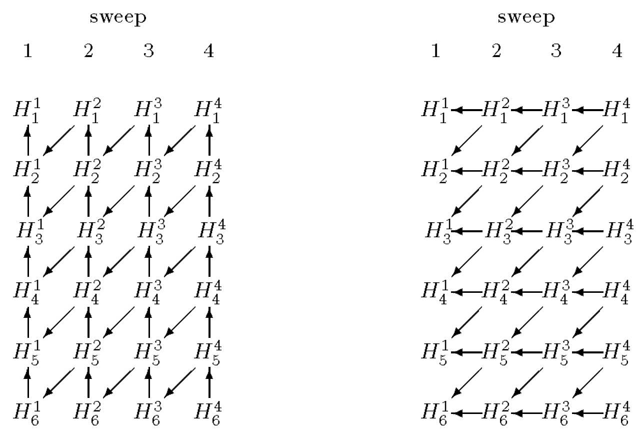

Figure 7. Interdependence of the Householder transformations H_k^j for the work on A (left picture) and on U (right picture). "$H \leftarrow \tilde{H}$" indicates that $\tilde{H}$ cannot be determined and applied to A (cannot be applied to U) until H has been applied to A (to U).

To make use of the additional freedom, we delay the work on U until a certain number n_b^U of reduction sweeps j, $j+1$, …, $J = j + n_b^U - 1$ are completed. Then the update of U is done *bottom up* by applying the transformations in the order

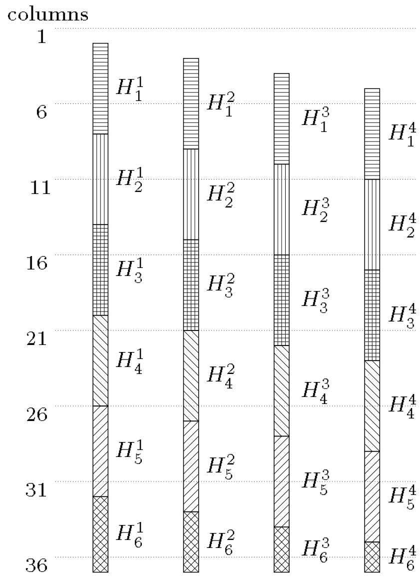

Figure 8. Columns of U affected by each transformation H_k^j of the first four reduction sweeps. In this example, $n = 36$ and $b = 6$. The transformations with the same hatching pattern can be aggregated into a blocked Householder transformation.

$H_{k_{\max}}^j$, $H_{k_{\max}}^{j+1}$, $\ldots$, $H_{k_{\max}}^J$, $H_{k_{\max}-1}^j$, $H_{k_{\max}-1}^{j+1}$, $\ldots$, $H_{k_{\max}-1}^J$, $\ldots$, H_1^j, H_1^{j+1}, $\ldots$, H_1^J. That is, the transformations in Figure 7 would be applied in the order H_6^1, H_6^2, H_6^3, H_6^4, H_5^1, H_5^2, $\ldots$, H_1^1, H_1^2, H_1^3, H_1^4. This order preserves the inter-sweep dependence mentioned above. In addition, the n_b transformations H_k^* with the same index k can be aggregated into a blocked Householder transformation. In Figure 8, the transformations contributing to the same block transformation are hatched identically.

A similar technique can also be used, for example, in the QR algorithm for computing the eigensystem of a symmetric tridiagonal matrix [Lang 1995]. It allows updating the eigenvector matrix with matrix-matrix products instead of single rotations.

5. EXPERIMENTAL RESULTS

The numerical experiments were performed on single nodes of the IBM SP parallel computer located at the High-Performance Computing Research Facility, Mathematics and Computer Science Division, Argonne National Laboratory, and on single nodes of the Intel Paragon located at the Zentralinstitut für Angewandte Mathematik, Forschungszentrum Jülich GmbH.

All timings are for computations in double precision. The matrices had random entries chosen from $[0, 1]$; since none of the algorithms is sensitive to the actual matrix entries, the following timings can be considered as representative.

Five codes were used in the experiments:

—`DSYTRD`: LAPACK routine for blocked tridiagonalization of full symmetric matrices [Dongarra et al. 1989],

Table I. Timings (in seconds) on one node of the IBM SP for the one-step (LAPACK routine DSYTRD) and two-step reduction (routines DSYRDB and DSBRDT) of full symmetric matrices of order $n = 3000$ to tridiagonal form. The timings do not include the update of U. The intermediate semibandwidth in the two-step reduction was always $b^{(2)} = 24$.

	One-Step Reduction	Two-Step Reduction		
n_b	DSYTRD	Total Time	full $\longrightarrow$ banded	banded $\longrightarrow$ tridiagonal
1	838	1831	1787	44
4	761	797	753	44
8	635	593	549	44
16	591	496	452	44
24	627	477	433	44

—`DSBTRD`: LAPACK routine for tridiagonalizing banded matrices using Kaufman's modification of Schwarz's algorithm [Schwarz 1968; Kaufman 1984] (called SK-algorithm in the following) to improve vectorization,

—`DSYRDB`: blocked reduction of full symmetric matrices to banded form (Algorithm 1),

—`DSBRDB`: blocked reduction of banded matrices to narrower banded form (Algorithm 1), and

—`DSBRDT`: tridiagonalization of banded matrices (MHL-algorithm with the technique from Section 4.3 for updating U).

The latter three codes are described in more detail in [Bischof et al. 1996].

All programs are written in Fortran 77. For the IBM SP node, which for our purposes can be viewed as a 66 MHz IBM RS/6000 workstation, the codes were compiled with `xlf -O3 -qstrict` and linked with `-lessl` for the vendor-supplied BLAS. For the Intel Paragon node, the compilation was done with `if77 -Mvect -O4 -nx`, and the BLAS were linked in with `-lkmath`.

The data presented in this section demonstrate that it may be advantageous to consider multistep reductions instead of conventional direct methods. In particular, on machines with memory hierarchies, it is not clear a priori which approach is superior for a given problem.

5.1 Tridiagonalization of Full Matrices

We first compared the one-step tridiagonalization routine `DSYTRD` from LAPACK with the two-step reduction (routines `DSYRDB` and `DSBRDT`) from Section 3.2. Table I shows that for large matrices, the two-step reduction can make better use of the Level 3 BLAS than can the direct tridiagonalization, where onehalf of the operations is confined to matrix-vector products. The reduction to banded form runs at up to 83 MFlops, which is close to the peak performance of the IBM SP node. Note that the tridiagonalization of the banded matrix cannot be blocked; therefore, the block size n_b affects only the reduction to banded form. The bad performance of this reduction with $n_b = 1$ is due to the fact that the routine `DSYRDB` does not provide optimized code for the nonblocked case.

If the transformation matrix U is required, too, then direct tridiagonalization is always superior because it has a significantly lower flops count.

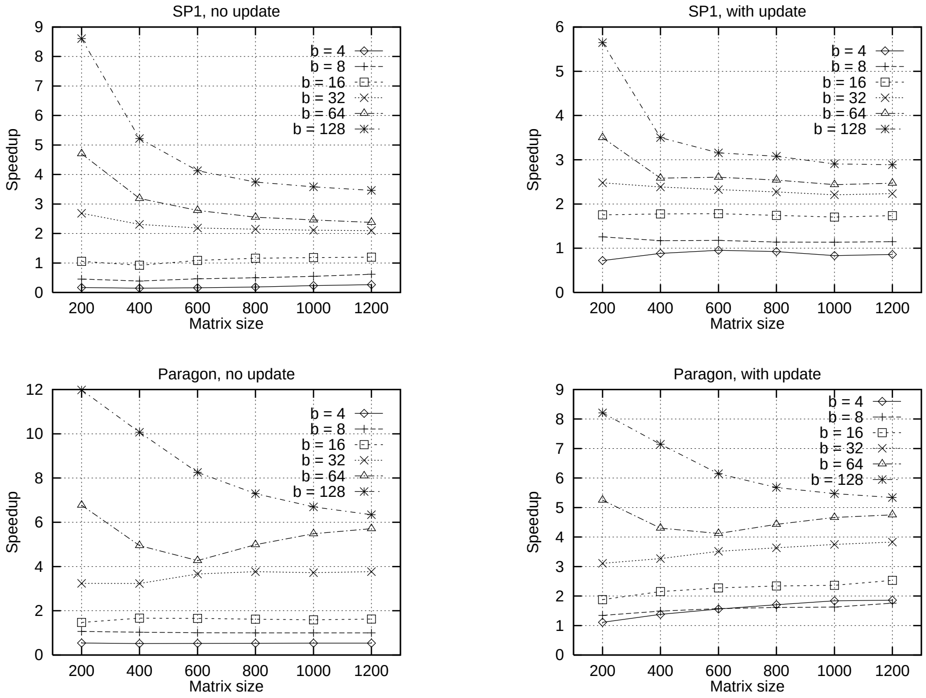

Figure 9. Speedup of the MHL-algorithm (routine `DSBRDT`) for tridiagonalizing banded matrices over the SK-algorithm (LAPACK routine `DSBTRD`) with and without updating the matrix U (left and right pictures, resp.). In the update, $n_b^U = 6$ Householder transformations were aggregated into a block transform.

5.2 One-step Tridiagonalization of Banded Matrices

Next, we compared the SK-algorithm (LAPACK routine `DSBTRD`) with the MHL-algorithm `DSBRDT`. Both are one-step tridiagonalization algorithms for band matrices. In the MHL-algorithm, the update of U was done by using blocked Householder transformations with the default block size $n_b^U = 6$. Figure 9 shows the results for various matrix dimensions and semibandwidths on both machines.

Lang [1993] already noted that the LAPACK implementation is not optimal except for very small semibandwidths. The reason is that the bulk of computations must be done in explicit Fortran loops, since no appropriate BLAS routines cover them. Therefore, the routine runs at Fortran speed, whereas the MHL-algorithm can rely on the Level 2 BLAS for the reduction and Level 3 BLAS for the update of U.

On the SP node, the MHL-algorithm runs at up to 45 MFlops in the reduction and 53 MFlops in the update, while `DSBTRD` only reaches 20 and 30 Mflops, respectively. (In addition, the SK-algorithm requires 50% more flops when U is updated.) This situation may be different on some vector machines because `DSBTRD` features vector operations with a higher average vector length, albeit with nonunit stride.

Table II. Timings (in seconds) on one node of the IBM SP for tridiagonalizing banded symmetric matrices of order $n = 1200$. The intermediate semibandwidth in the two-step reduction was $b^{(2)} = b/2$.

	$b = 32$	$b = 64$	$b = 128$
One-step reduction with `DSBRDT`	8.3	15.0	25.0
Two-step reduction with `DSBRDB` and `DSBRDT`	11.3	13.6	22.9

5.3 Two-step Tridiagonalization of Banded Matrices

In Section 3.5.3 we showed that peeling off the diagonals in two equal chunks requires fewer flops than direct tridiagonalization with the MHL-algorithm. Table II shows that the *time* for the two-step approach can be lower, too, if the semibandwidth is large enough.

In contrast to the theoretical results, however, the intermediate semibandwidth $b^{(2)} = b/2$ is not always optimal. For example, it took only 20.03 seconds to first reduce the semibandwidth from $b = 128$ to $b^{(2)} = 32$ and then tridiagonalize that matrix.

The explanation is that in the case $b^{(2)} = 32$, more of the work is done in the bandwidth reduction, which can rely on blocked Householder transformations, whereas the final tridiagonalization cannot. Therefore, the higher performance of the Level 3 BLAS more than compensates for the slighly higher flops count as compared with $b^{(2)} = b/2 = 64$.

For small semibandwidths b, the lower flops count of the two-step scheme was outweighed by the lower overhead (e.g., fewer calls to the BLAS with larger submatrices) of the one-step reduction. As in the reduction of full matrices, the one-step tridiagonalization is always superior when U is required, too.

5.4 Doubling-stride Tridiagonalization of Banded Matrices

While the MHL-algorithm is clearly superior on machines where the BLAS performance significantly exceeds that of pure Fortran code, it may not be applicable if storage is tight. Therefore, we also compared two algorithms that need only one additional subdiagonal as working space:

—the SK-algorithm (routine `DSBTRD` from LAPACK), and

—the doubling-stride sequence from Section 3.5.2: multiple calls to `DSBRDB` (with $n_b = 6$ for the reduction of A and the update of U) and one call to `DSBRDT` (with $n_b^U = 6$).

The results given in Figure 10 show that the doubling-sequence tridiagonalization, too, can well outperform the SK-algorithm on both machines.

6. CONCLUSIONS

We introduced a framework for band reduction that generalizes the ideas underlying the Householder tridiagonalization for full matrices and Rutishauser's algorithm and the MHL-algorithm for banded matrices. By "peeling off" subdiagonals in chunks, we arrived at algorithms that require fewer floating-point operations and less storage. We also provided an intuitive explanation of why our approach, which eliminates subdiagonals in groups, has a lower computational complexity than that

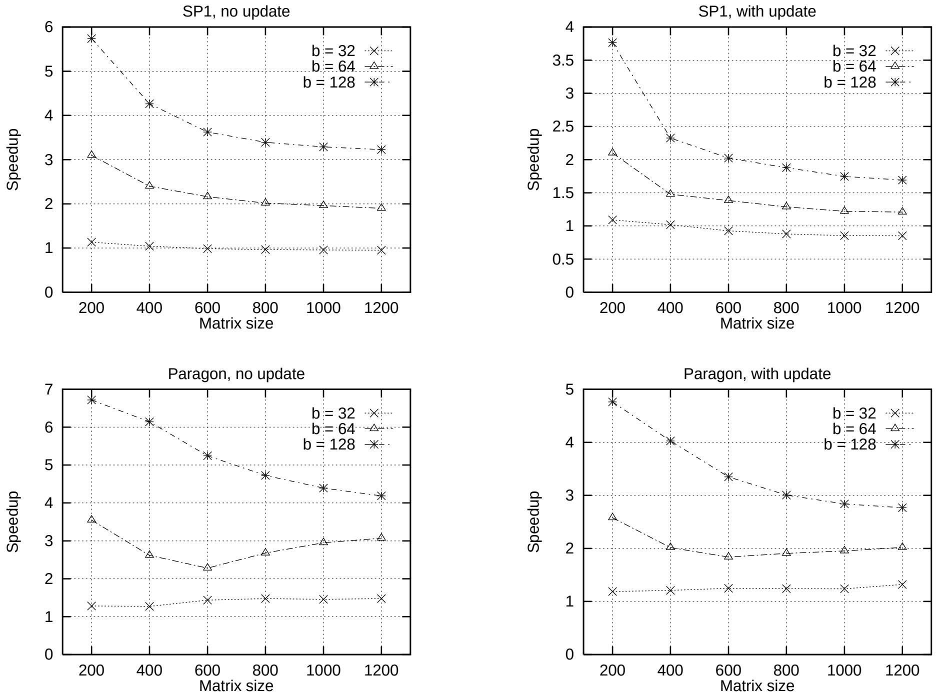

Figure 10. Speedup of the doubling-sequence tridiagonalization (routines **DSBRDB** and **DSBRDT** with $n_b = n_b^U = 6$) over the SK-algorithm (LAPACK routine **DSBTRD**) with and without updating the matrix U (left and right pictures, resp.).

of the previous algorithms for banded matrices, which eliminated subdiagonals either one by one or all at once. The successive bandreduction (SBR) approach improves the scope for block operations. In particular, the update of the transformation matrix U can always be done with blocked Householder transformations.

We also presented results that show that SBR approaches can provide better performance, by either using less memory to achieve almost the same speed, or by achieving higher performance. Our experience suggests that it is hard to provide a "rule of thumb" for selecting the parameters of an *optimal* bandreduction algorithm. While the flops count can be minimized by using the cost function (3), the actual performance of an implementation depends on the machine-dependent issues of floating-point versus memory access cost. In our experience, developing a more realistic performance model for advanced computer architectures is difficult (see, for example [Bischof and Lacroute 1990]), even for simpler problems. Another paper [Bischof et al. 1996] describes the implementation issues of a public-domain SBR toolbox enabling computational practioners to experiment with the SBR approach on problems of interest.

References

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. 1983. *Data Structures and Algorithms.* Addison-Wesley, Reading, Mass.

ANDERSON, E., BAI, Z., BISCHOF, C., DEMMEL, J., DONGARRA, J., DU CROZ, J., GREEN-

BAUM, A., HAMMARLING, S., MCKENNEY, A., OSTROUCHOV, S., AND SORENSEN, D. 1995. *LAPACK User's Guide* (2nd ed.). SIAM, Philadelphia.

BISCHOF, C., LANG, B., AND SUN, X. 1994. Parallel tridiagonalization through two-step band reduction. In *Proceedings of the Scalable High-Performance Computing Conference.* (Washington, D.C., 1994), pp. 23–27. IEEE.

BISCHOF, C. AND VAN LOAN, C. 1987. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput. 8*, 1 (January), s2–s13.

BISCHOF, C. H. AND LACROUTE, P. G. 1990. An adaptive blocking strategy for matrix factorizations. In H. BURKHART Ed., *Lecture Notes in Computer Science 457* (New York, 1990), pp. 210–221. Springer.

BISCHOF, C. H., LANG, B., AND SUN, X. 1996. The SBR toolbox – software for successive band reduction. Preprint ANL/MCS-P587-0496, Mathematics and Computer Science Division, Argonne National Laboratory.

BISCHOF, C. H. AND SUN, X. 1992. A framework for band reduction and tridiagonalization of symmetric matrices. Preprint MCS-P298-0392, Mathematics and Computer Science Division, Argonne National Laboratory.

BOJANCZYK, A. AND BRENT, R. P. 1987. Tridiagonalization of a symmetric matrix on a square array of mesh-connected processors. *Journal of Parallel and Distributed Computing 8*, s2–s13.

DONGARRA, J. J., HAMMARLING, S. J., AND SORENSEN, D. C. 1989. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math. 27*, 215–227.

GARBOW, B. S., BOYLE, J. M., DONGARRA, J. J., AND MOLER, C. B. 1977. *Matrix Eigensystem Routines – EISPACK Guide Extension.* Springer Verlag, Berlin.

GOLUB, G. H. AND VAN LOAN, C. F. 1989. *Matrix Computations* (2nd ed.). The Johns Hopkins University Press, Baltimore.

IPSEN, I. 1984. Singular value decompositions with systolic arrays. In *Proceedings of the SPIE*, Volume 549 (Bellingham, 1984), pp. 13–21. SPIE.

KAUFMAN, L. 1984. Banded eigenvalue solvers on vector machines. *ACM Trans. Math. Soft. 10*, 1 (March), 73–86.

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

LANG, B. 1995. Using level 3 BLAS in rotation based algorithms. Preprint.

LEDERMAN, S., TSAO, A., AND TURNBULL, T. 1991. A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues. Technical Report TR-91-042, Supercomputing Research Center, Institure for Defense Analysis, Bowie, Md.

MURATA, K. AND HORIKOSHI, K. 1975. A new method for the tridiagonalization of the symmetric band matrix. *Information Processing in Japan 15*, 108–112.

RUTISHAUSER, H. 1963. On Jacobi rotation patterns. In *Proceedings of Symposia in Applied Mathematics, Vol. 15: Experimental Arithmetic, High Speed Computing and Mathematics* (1963), pp. 219–239.

SCHREIBER, R. 1990. Bidiagonalization and symmetric tridiagonalization by systolic arrays. *Journal of VLSI Signal Processing 1*, 279–285.

SCHREIBER, R. AND VAN LOAN, C. 1989. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput. 10*, 1 (January), 53–57.

SCHWARZ, H. R. 1968. Tridiagonalization of a symmetric band matrix. *Numer. Math. 12*, 231–241.

SMITH, B. T., BOYLE, J. M., DONGARRA, J. J., GARBOV, B. S., IKEBE, Y., KLEMA, V. C., AND MOLER, C. B. 1976. *Matrix Eigensystem Routines – EISPACK Guide* (2nd ed.). Springer Verlag, Berlin.