THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-96-06

A parallel algorithm for the reduction to tridiagonal form for eigendecomposition

M. Hegland, M. H. Kahn and M. R. Osborne

June 1996

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

`http://cs.anu.edu.au/techreports/`

Recent reports in this series:

TR-CS-96-05 Andrew Tridgell and Paul Mackerras. *The rsync algorithm*. June 1996.

TR-CS-96-04 Peter Bailey and David Hawking. *A parallel architecture for query processing over a terabyte of text*. June 1996.

TR-CS-96-03 Brendan D. McKay. *`autoson` – a distributed batch system for unix workstation networks (version 1.3)*. March 1996.

TR-CS-96-02 Richard P. Brent. *Factorization of the tenth and eleventh Fermat numbers*. February 1996.

TR-CS-96-01 Weifa Liang and Richard P. Brent. *Constructing the spanners of graphs in parallel*. January 1996.

TR-CS-95-08 David Hawking. *The design and implementation of a parallel document retrieval engine*. December 1995.

A PARALLEL ALGORITHM FOR THE REDUCTION TO TRIDIAGONAL FORM FOR EIGENDECOMPOSITION

M. HEGLAND*, M. H. KAHN**, M. R. OSBORNE***

ABSTRACT. A new algorithm for the orthogonal reduction of a symmetric matrix to tridiagonal form is developed and analysed. It uses a Cholesky factorization of the original matrix and the rotations are applied to the factors. The idea is similar to the one used for the one-sided Jacobi algorithms [B. Zhou and R. Brent, *A Parallel Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations*, Proc. Sixth IASTED-ISMM International Conference on Parallel and Distributed Computing and Systems, pp. 369–372, 1994.]. The algorithm uses little communication, accesses data with stride one and is to a large extent independent of data distribution. It has been implemented on the Fujitsu VPP 500. The algorithm is designed to be the first step of an eigensolver so the procedure for accumulating transforms for eventual calculation of eigenvectors is given.

1. INTRODUCTION

Symmetric eigenvalue problems appear in many applications ranging from computational chemistry to structural engineering. Algorithms for symmetric eigenvalue problems have been extensively discussed in the literature [11, 9] and implemented in various software packages (e.g. LAPACK [1]). With the broader introduction of parallel computers in scientific computing new parallel algorithms have been suggested [7, 2]. In the following another new parallel algorithm is suggested which is particularly well adapted to vector parallel computers and has low operation counts.

Eigenvalue problems can only be solved by iterative algorithms in general as they are in an algebraic sense equivalent to finding the n zeros of a polynomial. There are, however, two main classes of methods to solve the symmetric eigenvalue problem. The first class only requires matrix vector products and does not inspect nor alter the matrix elements of the matrix. This class includes the Lanczos method [9] and

Date: November 1995.

1991 *Mathematics Subject Classification.* 65Y05,65F30.

Key words and phrases. Parallel Computing, Reduction Algorithms, One-sided Reductions.

*Computer Sciences Laboratory, Australian National University, Canberra ACT 0200, Australia.

**ANU Supercomputer Facility, Australian National University.

***Centre for Mathematics and its Applications, Australian National University.

has particular advantages for sparse matrices. However, in general, the Lanczos method has difficulties in finding all the eigenvalues and eigenvectors.

A second class of methods iteratively applies similarity transforms

$$A_i \mapsto A_{i+1} := Q_i A_i Q_i^T$$

with $A_0 = A$ to the matrix to get a sequence of orthogonally similar matrices which converge to a diagonal matrix. This second class of methods consists mainly of two subclasses. The first subclass uses Givens matrices for the similarity transforms and is Jacobi's method. It has been successfully implemented in parallel [2], [12]. A disadvantage of this method is its high operation count. A second subclass of methods first reduces the matrix with an orthogonal similarity transform to tridiagonal form and then uses special methods for symmetric tridiagonal matrices. Both parts of these algorithms pose major challenges to parallel implementation. For the second stage of tridiagonal eigenvalue problem solvers the most popular methods include divide and conquer [7] and multisectioning [9]. Here the reduction to tridiagonal form is discussed. Earlier algorithms use block matrix algorithms, see [6, 5, 3]. However, these methods have not achieved optimal performance. One problem is that similarity transforms require multiplications from both sides.

It was seen [12] that the Jacobi method based on one-sided transformations

$$B_i \mapsto B_i Q_i^T.$$

allows better vectorization and requires less communication than the original Jacobi algorithms. Assuming that A is positive semi-definite the intermediate matrices B_i can be defined as factors of the A_i, that is,

$$A_i = B_i^T B_i.$$

As will be seen in the following, the one-sided idea can also be used for the reduction algorithm.

The algorithm will form part of the subroutine library for a distributed memory computer, the Fujitsu VPP 500. Often the application of subroutines from libraries allow the user little freedom in the choice of the distribution of the data to the local memories of the processors. The one-sided algorithms allow a large range of distributions and perform equally well on all of them.

In the next section the one and two-sided reduction to tridiagonal form is described. Section 3 reinterprets the reduction as an orthogonalisation procedure similar to the Gram-Schmidt procedure. This reinterpretation is used to introduce the new one-sided reduction algorithm. In Section 4 the computation of eigenvectors is discussed and Section 5 contains timings and comparisons with other algorithms.

2. Reduction to tridiagonal form

A class of methods to solve the eigenvalue problem for symmetric matrices $A \in \mathbb{R}^{n\times n}$ first reduces them to tridiagonal form and in a second step solves the eigenvalue problem for this tridiagonal matrix. The problem of finding the eigendecomposition of the tridiagonal matrix will not be discussed here but that of accumulating transformations to be used for finding the eigenvectors of the symmetric matrix is investigated.

The reduction to tridiagonal form produces a factorization

$$A = Q^T T Q \tag{2.1}$$

where Q is orthogonal and T is symmetric and tridiagonal. If the offdiagonal elements of T are (nonzero) positive the factorization is uniquely determined by the first column of Q [9]. The proof of this fact leads directly to the Lanczos algorithm. While the Lanczos method has advantages especially for sparse matrices, methods based on sequences of simple orthogonal similarity transforms [9, p.118] are preferable for dense matrices.

2.1. Householder's reduction.

A method attributed to Householder uses Householder transformations or reflections

$$H(w) = I - \gamma w w^T \tag{2.2}$$

with $\gamma = 2/w^T w$. In the following let α_{ij} denote the elements of A. That is,

$$A = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{bmatrix}.$$

With $w = [0, \alpha_{21} - \beta_1, \alpha_{31}, \cdots, \alpha_{n1}]$ and $\beta_1 = \operatorname{sign}(\alpha_{21})\left(\alpha_{21}^2 + \cdots + \alpha_{n1}^2\right)^{1/2}$ the matrix $H(w)AH(w)$ has zeros in rows 3 to n in the first column and in columns 3 to n in the first row. The computation of $H(w)$, or equivalently of w, γ and β_1 requires n multiplications and n additions (up to $O(1)$).

Using the matrix vector product v

$$p = \gamma A w,$$

the application of $H(w)$ is a rank two update as

$$\begin{aligned} H(w)AH(w) &= A - wp^T - pw^T + (\gamma w^T p) w w^T \\ &= A - wq^T - qw^T \end{aligned}$$

where

$$q = p - w(\gamma w^T p)/2.$$

This takes n^2+n multiplications and n^2-n additions for the computation of p, $2n+2$ multiplications and $2n-1$ additions for the computation of q and n^2 multiplications and $2n^2$ additions for the rank two update. This gives a total of $2n^2 + 3n + 2$ multiplications and $3n^2 + n - 1$ additions.

In a second step, a v is found such that $H(v)H(w)AH(w)H(v)$ has additional zeros in columns 4 to n in the second row and in rows 4 to n in the second column and the procedure is repeated until the remaining matrix is tridiagonal. The sizes of the remaining submatrices decrease and at step $n-k$ a matrix of size k has to be processed requiring $2k^2 + 4k + 1$ multiplications and $3k^2 + 2k - 2$ additions. This gives a total of

$$\sum_{k=3}^{n} 2k^2 + 4k + 1 = 2n^3/3 + O(n^2)$$

multiplications and

$$\sum_{k=3}^{n} 3k^2 + 2k - 2 = n^3 + O(n^2)$$

additions. The tridiagonal matrix is not uniquely determined by the problem. For example, different starting vectors for the Lanczos procedure lead to different tridiagonal matrices. Also, different matrices can be obtained if different arithmetic precision is used [9, p.123/124]. However, despite this apparent lack of definition, the eigenvalues and eigenvectors of the original problem can still be determined with an error proportional to machine precision.

In summary, the sequential Householder tridiagonalization algorithm is as follows:

For $k = 1 : n - 2$

Calculate (γ, w) from $A(k + 1 : n, k)$

$p = \gamma A(k + 1 : n, k + 1 : n)w$

$q = p - w(\gamma w^T p)/2$

$A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - wq^T - qw^T$

End

For vector and parallel processors the Householder algorithm has some disadvantages. Firstly, as the iterations progress, the length of the vectors used in the calculations decreases but for efficient use of a vector processor we prefer long vector lengths. Secondly, in a parallel environment, if the input matrix A is partitioned in a banded manner across a one-dimensional array of processors, the algorithm will

be severely load imbalanced. To avoid this various authors have suggested using cyclic [5] or torus-wrap mappings of the data [10, 3]. Also, for a parallel implementation, the rank two update of A requires copies of both vectors w and q on all processors which leads to a heavy communication load.

2.2. **One-sided reduction.** A one-sided algorithm is developed to overcome the difficulties inherent in a parallel version of the sequential Householder algorithm. In addition, it is expected that the one-sided algorithm generates less fill-in for sparse matrices than the two-sided algorithms if the matrix is given in finite element form. Real symmetric matrices A are either given by or can be factored as

$$A = B^T D B \tag{2.3}$$

so A can be represented in factored form by D and B. If Cholesky factorization is to be used then the spectrum of A might have to be shifted so that $A - \mu I$ is positive definite. The parameter μ can be chosen using the Gershgorin shift. If exact arithmetic is used for the reduction, the eigenvectors of $A - \mu I$ are equal to the eigenvectors of A and the eigenvalues are shifted by μ. However, the precision of the computations will be affected by the introduction of the shift.

A Householder similarity transform of A is done by applying $H(w)$ to B as

$$H(w)AH(w) = (BH(w))^T D(BH(w)). \tag{2.4}$$

For this, a rank one modification must be computed

$$BH(w) = B - pw^T \tag{2.5}$$

with $p = \gamma Bw$. The computation of p requires $n^2 + n$ multiplications and $n^2 - n$ additions, the rank one modification n^2 multiplications and n^2 additions giving together $2n^2 + n$ multiplications and $2n^2 - n$ additions. In contrast to the two-sided algorithm, the number of additions is approximately the same as the number of multiplications in this algorithm. This is advantageous for architectures which can do addition and multiplication in parallel as it means better load balancing.

The computation of w is more costly for this method than for the original Householder method. As the Householder vector w is computed from the first column a_1 of A, this column has to be reconstructed first from the factored representation by

$$a_1 = B^T D b_1. \tag{2.6}$$

This requires $n^2 + n$ multiplications and $n^2 - n$ additions. Thus the computation of the Householder matrix $H(v)$ requires (up to O(1) terms) $n^2 + 2n$ multiplications and n^2 additions.

Adding these terms up, one reduction step needs $3n^2 + 2n$ multiplications and $3n^2 - n$ additions and so the overall costs of this algorithm are

$$\sum_{k=3}^{n} 3k^2 + 2k = n^3 + O(n^2)$$

multiplications and

$$\sum_{k=3}^{n} 3k^2 - k = n^3 + O(n^2)$$

additions. The total number of operations has increased compared with the original algorithm. But the time used on a computer which does additions and multiplications in parallel and at the same speed is the same. If the matrix A is not already factorized, however, the time to do this would have to be taken into account as well.

In summary, the sequential version of the one-sided algorithm is as follows.

For $k = 1 : n - 2$

Form $a_k = B(:, k+1:n)^T DB(:,k) \in \mathbb{R}^{n-k+1}$

Calculate (γ, w) from a_k

$p = \gamma Bw$

$B(:, k+1:n) = B(:, k+1:n) - pw^T$

End

3. The one-sided algorithm as an orthogonalization procedure

In order to develop the parallel version of the algorithm the one-sided reduction is interpreted as an orthogonalization procedure like the Gram-Schmidt process. Let b_i denote the ith column of B, that is,

$$B = [b_1, \ldots, b_n]. \tag{3.1}$$

Then the matrix A can be interpreted as the Gramian of the b_i as follows,

$$\alpha_{ij} = b_i^T D b_j, \quad i, j = 1, \ldots, n. \tag{3.2}$$

The one-sided tridiagonalization procedure constructs

$$C = [c_1, \ldots, c_n] \tag{3.3}$$

such that $T = C^T DC$ is tridiagonal and $C = BQ$ where Q is orthogonal. As T is the Gramian of the c_i the tridiagonality is a condition on the orthogonality of certain c_i as

$$c_i^T D c_j = 0, \quad \text{if } |i - j| \geq 2. \tag{3.4}$$

In a first step the orthogonality of $c_3, \ldots, c_n$ with c_1 is established by setting

$$c_1 = b_1 \tag{3.5}$$

and

$$c_j = \sum_{i=2}^{n} \gamma_{ji} b_i, \quad j = 2, \ldots, n \tag{3.6}$$

such that

$$c_j^T D c_1 = 0, \quad j = 3, \ldots, n \tag{3.7}$$

and

$$\sum_{i=2}^{n} \gamma_{ki} \gamma_{ji} = \delta_{kj}, \quad k, j = 2, \ldots, n. \tag{3.8}$$

Here δ_{kj} denotes the Kronecker delta. In a subsequent step, linear combinations of $c_3, \ldots, c_n$ are formed such that $c_4, \ldots c_n$ are orthogonal to c_2. As the $c_3, \ldots, c_n$ are orthogonal to c_1 the linear combinations are as well and so the subsequent steps do not destroy the earlier orthogonality relations. This is the basic observation used in the proof of this method.

The algorithm for reduction to tridiagonal form is then as follows:

$c_i^{(1)} := b_i, \quad i = 1, \ldots, n$

for $k := 2, \ldots, n-1$

$c_j^{(k)} := \sum_{i=k}^{n} \gamma_{ji}^{(k)} c_i^{(k-1)}, \quad j = k, \ldots, n$

end for

where the $\gamma_{ij}^{(k)}$ are such that the new $c_j^{(k)}$ are orthogonal to $c_{k-1}^{(k-1)}$. That is,

$$c_j^{(k)T} D c_{k-1}^{(k-1)} = 0, \quad j = k+1, \ldots, n$$

and the matrix $\left[\gamma_{ij}^{(k)}\right]_{i,j=k,\ldots,n}$ is orthogonal with

$$\sum_{h=k}^{n} \gamma_{ih}^{(k)} \gamma_{jh}^{(k)} = \delta_{ij}, \quad i, j = k, \ldots, n.$$

That is, at each iteration k, find an orthogonal transformation of the $c_i^{(k-1)}$ such that $c_j^{(k)}$ is orthogonal to $c_{(k-1)}^{(k-1)}$ for $j = k+1, \ldots, n$. This is equivalent to making the offdiagonal elements $\alpha_{j,k-1}$ and $\alpha_{k-1,j}$ of A zero for $j = k+1, \ldots, n$.

Proposition 3.1. *Let $c_j^{(k)}$ be computed by the previous algorithm and*

$$C = \left[c_1^{(1)}, \ldots, c_{n-2}^{(n-2)}, c_{n-1}^{(n-1)}, c_n^{(n)}\right]$$

where $c_n^{(n)} = c_n^{(n-1)}$. *Then* $C^T DC = T$ *is tridiagonal and there is an orthogonal matrix* Q *such that* $B = CQ$.

Proof. The proof is based on the fact mentioned earlier that some orthogonality relations are invariant. It uses induction. The proof is very similar to the one given in the next section for the corresponding parallel algorithm. □

Remark. The original Householder algorithm can be formulated in a similar way treating B as an inner product. Coordinate transformations change the matrix until it is tridiagonal.

3.1. **Parallel Algorithm.** In the following the single program multiple data model (SPMD) will be used. The basic assumption is that all the processors are programmed in the same way although their actions might be slightly different. Thus an SPMD algorithm is described by the pseudocode denoting what one processor has to do. The data in the matrix B is distributed to the processors by columns in a cyclic fashion. This means that processor 1 contains columns $1, 1+p, \ldots$, processor 2 contains $2, 2+p, \ldots$ and so on where p is the total number of processors. More formally, processor q contains b_i for $i \in \mathbb{N}_q = \{q, q+p, \ldots, q + \left\lfloor \frac{n-q}{p} \right\rfloor p\}$. In order to simplify notation let $\mathbb{N}_{q,k} = \mathbb{N}_q \cap \{k, \ldots, n\}$ where $q = 1, \ldots, p$ is the processor number. Furthermore mod denotes the mod function mapping positive and negative integers to $0, \ldots, p-1$.

$$c_i^{(1)} := b_i, \quad i \in \mathbb{N}_q$$

if $q = 1$ **broadcast** $c_1^{(1)}$ **else receive** $c_1^{(1)}$

for $k := 2, \ldots, n-1$

$$c_j^{(k)} := \sum_{i \in \mathbb{N}_{q,k}} \gamma_{ji}^{(k)} c_i^{(k-1)}, \quad j \in \mathbb{N}_{q,k}$$

gather $c_j^{(k)}$ **where** $j = k + \mathrm{mod}(q-k, p)$

$$\tilde{c}_j^{(k)} := \sum_{i=k}^{k+p-1} \tilde{\gamma}_{ji}^{(k)} c_i^{(k)}, \quad j = k, \ldots, k+p-1$$

end for

As before, the coefficients $\gamma_{ij}^{(h)}$ and $\tilde{\gamma}_{ij}^{(h)}$ are such that the modified columns are orthogonal to the last unmodified one. Thus

$$c_j^{(k)^T} D c_{k-1}^{(k-1)} = 0, \quad j \in \{k+p, \ldots, n\} \cap \mathbb{N}_q,$$

and

$$\tilde{c}_j^{(k)^T} D c_{k-1}^{(k-1)} = 0, \quad j = k+1, \ldots, k+p-1.$$

The coefficients also form orthogonal matrices so it follows that,

$$\sum_{h\in\mathbb{N}_{q,k}}^{n} \gamma_{ih}^{(k)}\gamma_{jh}^{(k)} = \delta_{ij}, \quad i,j \in \mathbb{N}_{q,k}$$

and

$$\sum_{h=k}^{k+p-1} \tilde{\gamma}_{ih}^{(k)}\tilde{\gamma}_{hj}^{(k)} = \delta_{ij}, i,j = k,\dots,k+p-1.$$

The $c_j^{(k)}$ are overwritten with the $\tilde{c}_j^{(k)}$. Note that the last calculations of the $c_j^{(k)}$ are duplicated on all p processors which leaves $c_k^{(k)}$ on all processors ready for the next step. The only communication required for each iteration is in the gathering of at most p vectors $c_j^{(k)}$.

Proposition 3.2. *Let $c_j^{(k)}$ be computed by the previous algorithm and*

$$C = \left[c_1^{(1)},\dots,c_{n-2}^{(n-2)},c_{n-1}^{(n-1)},c_n^{(n)}\right]$$

where $c_n^{(n)} = c_n^{(n-1)}$. Then $C^TDC = T$ is tridiagonal and there is an orthogonal matrix Q such that $B = CQ$.

Proof. Let

$$C^{(k)} = \left[c_1^{(1)},\dots,c_k^{(k)},c_{k+1}^{(k)},\dots,c_n^{(k)}\right].$$

First show that, for $k = 1,\dots,n-1$,

1. there is an orthogonal matrix $Q^{(k)}$ such that $B = C^{(k)}Q^{(k)}$ and
2. the linear hull of $c_{k+1}^{(k)},\dots,c_n^{(k)}$ is orthogonal to the linear hull of the $c_1^{(1)},\dots,c_{k-1}^{(k-1)}$ and
3. $T^{(k)}(1:k+1,1:k+1)$ is tridiagonal where $T^{(k)} = {C^{(k)}}^TDC^{(k)}$.

Proposition 3.2 is a consequence of this, obtained by setting $k = n-1$. The proof uses induction over k. The statement is easily seen to be true in the case of $k = 1$ with $Q^{(1)} = I$, the n dimensional identity matrix. In this case the orthogonality conditions are empty and the matrix $T^{(1)}$ is a two by two matrix and thus tridiagonal.

The remainder of the proof consists of proving the induction step. Assume that the three properties are valid for $k = 1,\dots,m$. Then it has to be shown that it is also valid for $k = m+1$.

¿From the construction it follows that $C^{(m+1)} = C^{(m)}G^{(m)}$ for an orthogonal $G^{(m)}$. Thus, with $Q^{(m+1)} = Q^{(m)}G^{(m)}$ one retrieves the first property. In particular, the existence of $G^{(m)}$ follows from the existence of the constructed Householder matrices.

Then the linear hull of $c_{m+2}^{(m+1)},\dots,c_n^{(m+1)}$ is a subspace of the linear hull of $c_{m+1}^{(m)},\dots,c_n^{(m)}$ and thus orthogonal to $c_1^{(1)},\dots,c_{m-1}^{(m-1)}$. Furthermore they have been constructed to be orthogonal to $c_m^{(m)}$. From this the second property follows.

Finally, $T^{(m+1)}(1:m+1,1:m+1) = T^{(m)}(1:m+1,1:m+1)$ which is tridiagonal and it remains to show that the the $m+2$nd column of $T^{(m+1)}(1:m+2,1:m+2)$ has zeros in the first m rows. But the first m elements of this column are just ${c_{m+2}^{(m+1)}}^T D c_j^{(j)}$ for $j = 1, \ldots, m$. They are zero because of the second property. □

This proof can be used for both the sequential and the parallel algorithm.

In practical implementations the orthogonalisation of the $c_j^{(k)}$ is achieved by forming the products ${c_j^{(k-1)}}^T D c_{k-1}^{(k-1)}$ which correspond to individual elements in the updated version of the symmetric matrix A. Householder transformations are then used to zero the relevant off-diagonal elements. At the first stage in each iteration these transformations are carried out locally and after the gathering step the transformation on the (at most) p formed elements of A is replicated on all processors. Although Householder transformations are used here, it would also be possible to use Givens transformations.

3.2. **Error analysis.** When using the one-sided reduction to tridiagonal as one step in a complete eigensolver there are several possible components to the error. First there is the Cholesky factorization. However a result in Wilkinson [11] (equation 4.44.3) shows that the quantity $\|A - B^T B\|$ is extremely small. Thus the error incurred in working with the Cholesky factor B in the subsequent calculations is minimal.

In the second step, the tridiagonal matrix $C^{(n-2)}$ is produced by successive reduction of $B^T B$ by orthogonal similarity transforms where $C^{(k)}$ is the transformed matrix calculated after the k'th stage. The error in the eigenvalues of $C^{(n-2)}$ is bounded by the numerical error in the transform [11]. After the k'th step this is given by

$$\|C^{(k)} - B \prod_{i=1}^{k} P_i\|$$

where P_i is the exact orthogonal matrix corresponding to the actual computed data at stage i. A bound on this difference follows from Wilkinson [11] (equation (3.45.3)). In fact,

$$\|C^{(k)} - B \prod_{i=1}^{k} P_i\| \leq (12.36) n 2^{-t} \|B\|$$

where $\|B\| = \sqrt{\sum_{i,j} B_{ij}^2}$ and t is the word length. So the error introduced by the reduction to tridiagonal is small.

The final stage is the calculation of the eigenvalues of the tridiagonal matrix. These are determined to an accuracy which is high relative to the largest element of the tridiagonal matrix. This applies for example to eigenvalues computed using

the Sturm procedure. But this result does not guarantee high relative accuracy in the determination of small eigenvalues so, if this is important, the Jacobi method becomes the method of choice [4].

4. Calculating Eigenvectors

In order to calculate the eigenvectors of the symmetric matrix, the orthogonal matrix Q defining the reduction is accumulated. This is achieved by starting with the identity matrix (distributed cyclically over the processors), then updating it by multiplying it by the same Householder transformations used to update C. Forming Q explicitly is preferable to storing the details of the transformations and then applying them to the eigenvectors of the tridiagonal matrix which is the usual procedure for sequential implementations. The reason for this is that the matrix of Q is distributed in a way which renders multiplication from the left inefficient. This is discussed more fully in the following.

4.1. **Calculation of Eigenvectors for One-Processor Version.**

The $n \times n$ symmetric matrix A is reduced to tridiagonal form by a sequence of Householder transformations represented by the orthogonal $n \times n$ matrix Q and

$$Q^T A Q = T \tag{4.1}$$

where $T \in \mathbb{R}^{n*n}$ is tridiagonal. For the one-sided algorithm, we have actually calculated $S = BQ$ where, $A = B^T B$ and $T = S^T D S$.

The eigendecomposition of T is given by

$$TV = V\Lambda \tag{4.2}$$

where Λ is a diagonal matrix containing the eigenvalues and $V \in \mathbb{R}^{n*n}$ is the matrix of eigenvectors. Combining the above two equations gives

$$V^T Q^T A Q V = \Lambda$$

and so the eigenvectors of A are the columns of $U = QV$.

The matrix Q obtained during the rediagonalization procedure is represented by the Householder matrices $H_1, \ldots, H_{n-2}$ and

$$Q = H_1 \ldots H_{n-2}.$$

The eigenvector matrix V of the tridiagonal matrix T is represented by its matrix elements. So in order to get the matrix element representation of U form the product

$$U = H_1 \ldots H_{n-2} V.$$

This product can be done in two different ways. The first method multiplies V with the H_k. Formally, define a sequence $U_1, \ldots, U_{n-1}$ such that

$$\begin{aligned} U_{n-1} &:= V \\ U_k &:= H_k U_{k+1} \end{aligned}$$

for $k = n-2, \ldots, 1$. Then $U = U_1$. This method is often used, for example [8], where it is called backward transformation.

A second method computes the element-wise representation of Q first by the recursion

$$\begin{aligned} Q_0 &:= I \\ Q_k &:= Q_{k-1} H_k \end{aligned}$$

for $k = 1, \ldots, n-2$ and then specifically computes the matrix-vector product $U = QV$ where $Q = Q_{n-2}$. This is called forward accumulation in [8].

The difference between these two methods is that the first one applies the Householder transforms H_k from the left and the second applies them from the right. In addition the second method requires the computation of a product of two matrices in element form. For the sequential case when using the two-sided Householder reduction, the first method is preferred as it avoids matrix multiplication. However, for the multi-processor version, multiplication from the left by the Householder transformation requires extra communication when the columns of the matrix of eigenvectors V are distributed over the processors. In fact, one purpose of the one-sided reduction is to avoid this communication in the reduction to tridiagonal form. There is certainly communication required for the matrix multiplication QV but this is of fewer large blocks of the matrix so will be less demanding.

4.2. **Multi-processor Version.** In the multi-processor case there is an added complication in that B is assumed to be cyclically distributed (although it is not explicitly laid out as such). So use

$$\tilde{B} = BP$$

where P is the permutation that transforms B to cyclic layout.

The matrix Q is a product of $2 \times (n-2)$ matrices formed from the Householder transformations.

$$Q = H_1^{(1)} H_1^{(2)} \ldots H_{n-2}^{(1)} H_{n-2}^{(2)}$$

Here, $H_j^{(1)}$ refers to the transformations carried out locally at each step of the reduction and $H_j^{(2)}$ refers to the transformation using one column of B from each processor

which is carried out redundantly on all processors. To find Q start with an identity matrix distributed cyclically across the processors to match the implicit layout of B. This matrix is then updated by transforming it with the same Householder transformations that are used to update B to obtain the tridiagonal matrix.

The matrix of eigenvectors V of the tridiagonal matrix is obtained in block layout whilst the matrix Q is in cyclic layout. To find the eigenvectors of the original symmetric matrix this must be taken into account so instead of $U = QV$, as above, we need QPV. The cyclic ordering must then be reversed and finally the eigenvectors are given by

$$U = P^{(-1)}QPV.$$

Pre-multiplication by the permutation P involves a re-ordering of the coefficients of the eigenvectors which is carried out locally on the processor and does not involve any communication.

5. Timings

The parallel one-sided reduction to tridiagonal has been implemented and tested on the Fujitsu VPP500. The VPP500 is a parallel supercomputer consisting of vector processors connected by a full crossbar network. The theoretical peak performance of each PE is 1.6 Gflops and the maximum size of memory for each processor is 1 Gbyte. The VPP500 is scalable from 4 to 222 processing elements but access was available only to a 16 processor machine. Each processor can perform send and receive operations simultaneously through the crossbar network at a peak data transfer rate of 400 Mbytes/s each.

The one-sided algorithm is particularly suited to the architecture of the VPP500 because the calculation of the elements of the updated matrix A from the current version of C are vectorisable with loops of length n, the size of the input matrix. In the conventional two-sided Householder reduction the vector length decreases at each iteration.

Table 1 shows some timings and speeds obtained on up to a 16 processor VPP500. The two times and speed given are, first, for the reduction without accumulating the transformations for later eigenvectors calculations and, second, for the reduction with accumulation. The speed is given in Gflops. The code was written in the parallel language VPP-Fortran which is basically FORTRAN77 with added compiler directives to achieve parallel constructs such as data layout, interprocessor communication and so on.

¿From these performance figures it appears that the algorithm is scalable in so far as its performance is maintained as the size of the problem is increased along with the

p	n	Secs	Gflops	Secs	Gflops
1	512	.64	.626	.87	.775
	1024	3.8	.846	5.3	1.018
	2048	26	1.000	37	1.164
2	1024	2	1.534	3	1.806
	2048	13	1.920	19	2.211
	3072	51	1.718	77	1.897
4	1024	1.5	2.265	2.2	2.481
	2048	8	3.278	12	3.611
	3072	28	3.135	43	3.371
	4096	60	3.455	92	3.742
8	4096	41	5.180	67	5.250
	8192	272	6.157	441	6.337
16	4096	46	5.018	84	4.602
	8192	236	7.447	408	7.010

TABLE 1. Timings and Speed for Matrices of Increasing Size

number of processors. The one-sided reduction to tridiagonal of a matrix of size 2048 using 2 processors achieves nearly 70% of peak performance and approximately 60% for a matrix of size 4096 using 4 processors. A formal analysis of the communication required by the one-sided algorithm gives the communication volume proportional to $p(p-1)n^2$ where p is the number of processors. As the computational load is proportional to n^3, isoefficiency is obtained when $p(p-1)n^2/n^3$ is constant, that is, $n = \mathcal{O}(p^2)$. The scalability of the algorithm is evident if speeds for matrices of size n on p processors are compared with those of matrices of size $4n$ using $2p$ processors. For example, compare the speeds for $n = 1024$, $p = 4$ and $n = 4096$, $p = 8$ to see the doubling of Gflop rate.

It is interesting to compare these performance figures with other published results for alternative parallel two-sided Householder reductions to tridiagonal. The comparisons can only be general as the algorithms and machine architectures are very different. The most straightforward comparison is time taken to reduce a matrix of fixed size measured on machines of similar peak Gflop rate. In practice, the first step of the Cholesky factorization adds an overhead of about one tenth of the time taken for the one-sided reduction to tridiagonal. All accessible published results refer to algorithms implemented on Intel machines.

Dongarra and van de Geijn [5] give times for a parallel reduction to tridiagonal using panel wrapped storage on 128 nodes of a 520 node Intel Touchstone Delta. Equating peak Mflop rates suggests that this is comparable to a 6 processor VPP500. For a matrix of size 4000 their implementation on the Intel took twice as long as the one-sided reduction of the same size matrix on a 4 processor VPP500.

The ScaLAPACK implementation of a parallel reduction to tridiagonal is given by Choi, Dongarra and Walker [3]. Extrapolating from their graphs of Gflop rates it can be seen that their times taken for various sized problems are two or three times that taken by the one-sided algorithm on the same size problems on a VPP500 of comparable peak performance.

Smith, Hendrickson and Jessup [10] use a square torus-wrap mapping of matrix elements to processors and tested their code on an Intel machine corresponding to a 12 processor VPP500. Their two-sided algorithm can be inferred to have taken about the same time as a slightly larger problem on a 16 processor VPP500 using the one-sided algorithm. The Smith et al algorithm is more sophisticated than the other two-sided algorithms as it uses the torus-wrap mapping and Level 3 BLAS.

6. Conclusion

A new algorithm for reduction of a symmetric matrix to tridiagonal as the first step in finding the eigendecomposition has been developed. Starting with the Cholesky factorization of the symmetric matrix, orthogonal transformations based on Householder reductions are applied to the factor matrix until the tridiagonal form is reached. This is referred to as a one-sided reduction and leads to the updating of the factor matrix at each iteration being rank one rather than rank two as in the conventional Householder reduction to tridiagonal. Transformations can also be accumulated to allow for calculation of eigenvectors. This algorithm is suited to parallel/vector architectures such as the Fujitsu VPP500 where it has been shown to perform well. In a complete calculation of the eigenvalues and eigenvectors of a symmetric matrix the extra time for the Cholesky factorization is observed to be about one tenth of that required for the reduction to tridiagonal so it is not a significant overhead.

7. Acknowledgement

The authors would like to thank Prof. R. Brent for his helpful comments and Mr. Makato Nakanishi of Fujitsu Japan for help with access to the VPP500.

References

[1] E. Anderson, Z.Bai, C.Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson, *Lapack users' guide*, SIAM, Philadelphia, 1992.

[2] R.P. Brent and F.T. Luk, *The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays*, SIAM J. Sci. Stat. Comput. **6** (1985), 69-84.

[3] J. Choi, J. Dongarra, and D. W. Walker, *The design of a parallel, dense linear algebra software library: Reduction to Hessenberg, tridiagonal and bidiagonal form*, submitted to SIAM J. Sci. Comp., 1994.

[4] J. Demmel and K. Veselic, *Jacobi's method is more accurate than QR*, SIAM J. Matrix Analysis and Applications, **13**, 1992, 1204–1245.

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

[6] J. Dongarra, S. J. Hammarling, and D. C. Sorensen, *Block reduction of matrices to condensed forms for eigenvalue computations*, J. Comp. and Applied Mathematics **27** (1989), 215-227.

[7] J. Dongarra and D.C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput. **8** (1987), 139-154.

[8] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, 2nd ed, 1989.

[9] B. N. Parlett, *The symmetric eigenvalue problem*, Prentice Hall, 1980.

[10] C. Smith, B. Hendrickson, and E. Jessup, *A parallel algorithm for Householder tridiagonalization*, Proc. Fifth SIAM Conf. Appl. Linear Alg., 1994, pp. 361–365.

[11] J.H. Wilkinson, *The algebraic eigenvalue problem*, Oxford University Press, 1965.

[12] B.B. Zhou and R.P. Brent, *A Parallel Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations*, Proc. Sixth IASTED-ISMM International Conference on Parallel and Distributed Computing and Systems, 1994.

E-mail address: Markus.Hegland@anu.edu.au