A One-Sided Jacobi Algorithm for the Symmetric Eigenvalue Problem[*]

B. B. Zhou, R. P. Brent
E-mail: bing,rpb@cslab.anu.edu.au
Computer Sciences Laboratory
The Australian National University
Canberra, ACT 0200, Australia
Phone: +61-6-2490012
Fax: +61-6-2494812

M. Kahn
E-mail: Margaret.Kahn@anu.edu.au
Supercomputer Facility
The Australian National University
Canberra, ACT 0200, Australia
Phone: +61-6-2494541
Fax: +61-6-2473425

Abstract

A method which uses one-sided Jacobi to solve the symmetric eigenvalue problem in parallel is presented. We describe a parallel ring ordering for one-sided Jacobi computation. One distinctive feature of this ordering is that it can sort column norms in each sweep, which is very important to achieve fast convergence. Experimental results on both the Fujitsu AP1000 and the Fujitsu VPP500 are reported.

1 Introduction

Jacobi methods for the symmetric eigenvalue problem have recently attracted interest because they are readily parallelisable and are more accurate than QR-based methods for solving the same problem [6].

There are two basic types of Jacobi, that is, *one-sided* Jacobi and *two-sided* Jacobi. The traditional two-sided Jacobi method for the symmetric eigenvalue problem works by performing a sequence of orthogonal similarity updates $A \leftarrow Q^T AQ$ with the property that each new A, although full, is "more diagonal" than its predecessor. Eventually, the off-diagonal entries are small enough to be ignored. Because both column and row updatings are required, this method suffers from extensive communication of small amounts of data between processors in parallel computation, and nonunit strides in vector operations. One-sided Jacobi, though originally applied for singular value decomposition, can also be adapted for the symmetric eigenvalue problem. This method requires only column updating and so does not need as much communication and is more suitable for vector pipeline computing. Thus one-sided Jacobi is preferable to two-sided Jacobi in pipeline/parallel computation.

In parallel implementation of one-sided Jacobi SVD a key problem is how to choose a reasonable, systematic order of rotations in each sweep of the computation so that a fast convergence rate is achieved. In this paper we describe a parallel ring Jacobi ordering. One distinctive feature of this ordering is that it can sort column norms, which is very important for fast convergence. The experimental results show that the algorithm adopting this ordering can achieve the same efficiency (in terms of the total number of sweeps) as the cyclic Jacobi algorithm in sequential computation.

The paper is organised as follows. The sequential one-sided Jacobi algorithm for computing the SVD is outlined in §2. Our parallel ring Jacobi ordering is introduced in §3 and the experimental results are presented in §4. The method for adapting one-sided Jacobi in symmetric eigenvalue decomposition are described in §5. Some conclusions are given in §6.

2 Sequential One-sided Jacobi

For a matrix A of order $m \times n$ $(m \geq n)$ the one-sided Jacobi method produces an orthogonal matrix V such that $AV = S$, where the columns

[*]Copyright © 1994, the authors. Appeared in *Proc Third Parallel Computing Workshop*, Kawasaki, Japan, November 1994, P1-Q-1–P1-Q-7. rpb155 typeset using LaTeX

of S are orthogonal to within a given tolerance. The non-zero columns of S can be normalised to give

$$S = (U_r|0)\begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}$$

where $r \leq n$ is the rank of A, and $\Sigma_r = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$. Thus

$$A = U_r \Sigma_r V_r^T$$

where V_r is an $n \times r$ matrix consisting of the first r columns of V. This is the singular value decomposition of A.

The matrix V can be generated as a product of plane rotations. Consider the transformation by a plane rotation:

$$\begin{pmatrix} a_i & a_j \end{pmatrix}\begin{pmatrix} c & -s \\ s & c \end{pmatrix} = \begin{pmatrix} a_i{}' & a_j{}' \end{pmatrix}$$

where $c = \cos\theta$, $s = \sin\theta$, and a_i and a_j are the i-th and j-th columns of the matrix A. We choose θ to make $a_i{}'$ and $a_j{}'$ orthogonal. As in the traditional Jacobi algorithm, the rotations are performed in a fixed sequence called a *sweep*, each sweep consisting of $n(n-1)/2$ rotations, and every column in the matrix is orthogonalised with every other column exactly once per sweep. The iterative procedure terminates if one complete sweep occurs in which all columns are orthogonal to working accuracy and no columns are interchanged. If the rotations in a sweep are chosen in a reasonable, systematic order, the convergence rate is ultimately quadratic [9, 11]. Exceptional cases in which cycling occurs are easily avoided by the use of a threshold strategy [23].

There are two important implementation details which determine the speed of convergence of the one-sided Jacobi method for computing the SVD. The first is the method of ordering, i.e., how to order the $n(n-1)/2$ rotations in one sweep of computation. Various orderings have been introduced in the literature. In sequential computation, the most commonly used is the cyclic Jacobi ordering (cyclic ordering by rows or by columns) [9, 12]. When discussing sequential Jacobi algorithms in this paper, we assume that the cyclic ordering by rows is applied.

The second important detail is the method for generating the plane rotation parameters c and s in each iteration. For the one-sided Jacobi method there are three main rotation algorithms, which we now describe.

Rotation Algorithm 1 This algorithm is derived from the standard two-sided Jacobi method for the eigenvalue decomposition of the matrix $B = A^T A$.

Suppose that after k sweeps we have the updated matrix $A^{(k)} = \begin{bmatrix} a_1^{(k)} & a_2^{(k)} & \cdots & a_n^{(k)} \end{bmatrix}$. To annihilate the off-diagonal element $b_{ij}^{(k)}$ of $B^{(k)} = (A^{(k)})^T A^{(k)}$ in the $(k+1)^{th}$ sweep, we first need to compute $b_{ii}^{(k)}$, $b_{ij}^{(k)}$ and $b_{jj}^{(k)}$, that is,

$$b_{ii}^{(k)} = (a_i^{(k)})^T a_i^{(k)} = \|a_i^{(k)}\|^2,$$

$$b_{ij}^{(k)} = (a_i^{(k)})^T a_j^{(k)}$$

and

$$b_{jj}^{(k)} = (a_j^{(k)})^T a_j^{(k)} = \|a_j^{(k)}\|^2.$$

where $\|x\|$ is the 2-norm of the vector x. The plane rotation factors c and s, which are used to orthogonalise the corresponding two columns, are then generated based on the two-sided Jacobi method. It can be proved that the value of $b_{ii}^{(k)}$ is increased and the value of $b_{jj}^{(k)}$ is decreased after a plane rotation operation if $b_{ii}^{(k)} > b_{jj}^{(k)}$. Otherwise, $b_{ii}^{(k)}$ is decreased and $b_{jj}^{(k)}$ is increased.

Rotation Algorithm 2 The second algorithm, introduced by Hestenes [13], is the same as the Algorithm 1 except that the columns $a_i^{(k)}$ and $a_j^{(k)}$ are to be swapped if $\|a_i^{(k)}\| < \|a_j^{(k)}\|$ for $i < j$ before the orthogonalisation of the two columns. Therefore, we always have $b_{ii}^{(k+1)} \geq b_{jj}^{(k+1)}$. When the cyclic ordering by rows is applied, the computed singular values will be sorted in a nonincreasing order.

Rotation Algorithm 3 The third algorithm was derived by Nash [19] and implemented on the ILLIAC IV by Luk [16]. To determine the rotation parameters c and s for orthogonalising two columns i and j, one extra condition has to be satisfied in this algorithm, that is,

$$\|a_i^{(k+1)}\|^2 - \|a_i^{(k)}\|^2 = \|a_j^{(k)}\|^2 - \|a_j^{(k+1)}\|^2 \geq 0.$$

With this extra condition the rotation parameters are chosen so that $\|a_i^{(k+1)}\|$ is greater than $\|a_j^{(k+1)}\|$ after the orthogonalisation, without explicitly exchanging the two columns. As in Algorithm 2, the computed singular values will appear in a nonincreasing order if the cyclic ordering by rows is applied.

It is known from numerical experiments that an implementation which uses Rotation Algorithm 2 or 3 is more efficient than the one using Rotation Algorithm 1 when the cyclic ordering is applied.

It is easy to verify that implicit in the cyclic ordering is a sorting procedure which can sort the values of n elements into nonincreasing (or nondecreasing) order in $n(n-1)/2$ steps. Since rotation algorithms 2 and 3 always increase $b_{ii}^{(k)}$ and decrease $b_{jj}^{(k)}$ for $i < j$ when orthogonalising the two columns, the column norms tend to be sorted after each sweep of computations. Therefore, the columns and their norms tend to be approximately determined after a few sweeps and only change by a small amount during each sweep. Since the column norms are not sorted during each sweep when using rotation algorithm 1, it is possible that the norm of column i may be increased when two columns i and j are orthogonalised in a sweep, but norm of column j will be increased when the two columns meet again in the next sweep. Thus there are oscillations in column norms and (empirically) it takes more sweeps for the same problem to converge. This effect was also noted in [6, 20]. It is probably the main reason why applying Rotation Algorithm 2 or 3 is more efficient than applying Rotation Algorithm 1.

In order to compare the performance in terms of the total number of sweeps with parallel implementations which are described in the following sections, we give in Table 1 some experimental results obtained on a (sequential) Sun Sparc workstation.

Size	Alg. 1	Alg. 2	Alg. 3
80	11	9	9
100	12	8	8
120	11	9	9
140	12	9	9
160	12	9	9
180	12	9	9
200	12	9	10

Table 1: Results for the cyclic Jacobi ordering on a Sun workstation.

3 The Ring Jacobi Ordering

We have seen in the previous section that sorting the column norms in each sweep is a very important issue. Our experimental results confirm that if an ordering does not include a proper sorting procedure in each sweep, it may converge relatively slowly [24]. In this section we describe a parallel ring ordering. This ordering can not only generate the required index pairs in a minimum number of steps, but also sort column norms at the same time.

Our Jacobi ordering consists of two procedures, *forward sweep* and *backward sweep*, as illustrated in Fig. 1. They are applied alternately during the computation.

In either forward or backward sweep the n indices are organised into two rows. Any two indices in the same column at a step form one index pair. One index in each column is then shifted to another column as shown by the arrows so that different index pairs can be generated at the next step. The up-and-down arrow in Fig. 1 indicates the exchange of two indices in the column before one is shifted. Each sweep (forward or backward), taking $n-1$ steps, can generate $n(n-1)/2$ different Jacobi pairs, as well as sort the values of n elements into nonincreasing (or nondecreasing) order.

We outline a proof that $n(n-1)/2$ different Jacobi pairs can be generated in $n-1$ steps by either a forward or backward sweep. To do this, we first permute the initial positions of n indices for the *round robin* ordering [5] and then show that the orderings can generate the same index pairs at any step. Since it is well known that the round robin ordering generates $n(n-1)/2$ different index pairs in $n-1$ steps, this shows the correctness of our claim. The detailed proof of this claim is tedious and is omitted.

It can easily be verified that the forward sweep and the backward sweep are essentially the same,

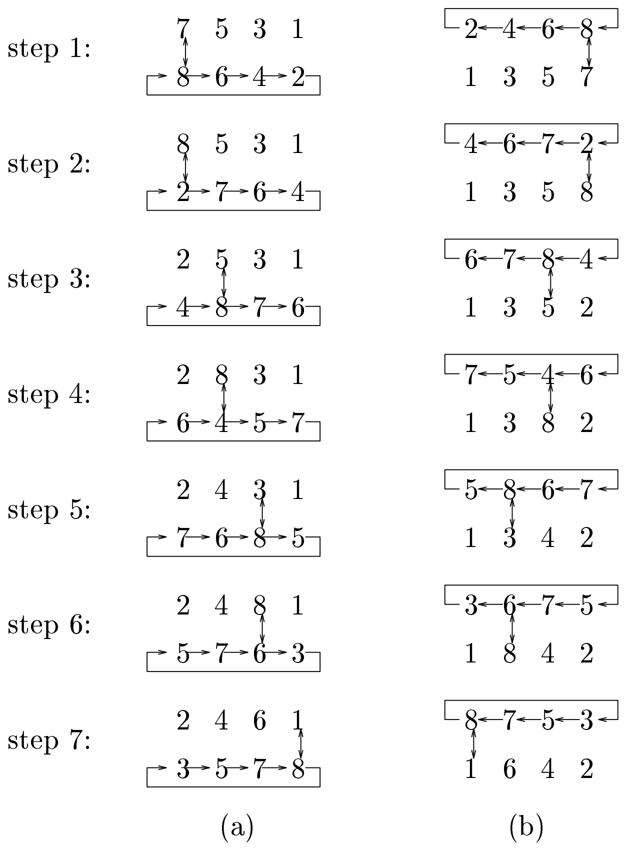

Figure 1: The ring Jacobi ordering. (a) forward sweep and (b) backward sweep.

except that one sorts the elements into nondecreasing order and the other sorts the elements into nonincreasing order. Thus we only use the forward sweep as an example to show the procedure on how to sort n elements into nondecreasing order. (For details see [24].)

If the numbers in Fig. 1(a) are not considered as indices, but as the values of n elements, the Figure gives an example of sorting n elements from nonincreasing order to nondecreasing order. In each step the smaller element in each column is placed on the top except in even steps the larger element is placed on the top if the column has a up-and-down arrow in it. Since the up-and-down arrow indicates the exchange of the two elements in the column, these arrows can be removed in even steps by letting the smaller elements be placed at the top of the corresponding columns. Thus, we may describe the sorting procedure as follows:

One forward sweep can be applied to sort n elements in a nondecreasing order. Each step in the sweep consists of two substeps. The first substep compares the two elements in each column and places the smaller one on the top and the larger one at the bottom. The second substep then shifts the elements located at the bottom to the next column according to the arrows which form a ring, as depicted in Fig. 1(a). At each odd step the two elements in the column with a up-and-down arrow have to exchange their positions before the shift takes place. The n elements are sorted into nondecreasing order after $n-1$ such steps (see top row of Fig. 1(b)).

Since both index ordering and sorting can be done simultaneously in either a forward or a backward sweep, it may seem that applying these two sweeps alternately in the SVD computation is not necessary. The reason why we perform the two sweeps alternately is as follows. Suppose that the n indices are initially placed in a nonincreasing order. They will be sorted into nondecreasing order during a forward sweep. However, the natural order of indices for index ordering at each step is maintained during sorting. Thus the $n(n-1)/2$ different index pairs are also generated during the computation. Although the original (nonincreasing) order is restored when a backward sweep is performed, the exchange of positions of some indices is probable. As a consequence some index pairs may not be produced during the computation. This can easily be verified by an example of sorting a small number of indices (which are initially placed in a nonincreasing order) using the backward sweep.

4 Experimental Results

In order to see the importance of sorting the column norms in a parallel implementation of the one-sided Jacobi SVD, we implemented our ring ordering algorithm on the Fujitsu AP1000 at the Australian National University. In the experiment both singular values and singular vectors are computed on the AP1000, which is configured as a one-dimensional array.

An algorithm without partitioning not very useful in practice for general-purpose parallel computation because the system configuration is fixed, but the size of user's problem may vary.

size	Algorithm 1		Algorithm 2		Algorithm 3	
	T	S	T	S	T	S
200	11.58	12	10.01	10	10.02	10
400	63.35	13	57.39	11	57.42	11
600	210.8	14	187.1	12	185.6	12
800	499.5	15	416.7	12	416.3	12
1000	945.2	15	799.1	12	799.8	12
1200	1702	16	1407	12	1445	13
1400	2787	17	2280	13	2272	13

Table 2: Results for the ring Jacobi ordering on an AP1000 with 100 cells configured as a linear array (T = time (sec.), S = sweeps).

Our partitioning strategy is based on the method described in [21]. However, a major difference is that we take sorting into consideration. Assume that the given system has p processors. We first divide n columns of the matrix into $2p$ blocks. (The block sizes are not necessarily the same.) At the beginning of a sweep, the columns in each block are orthogonalised with each other exactly once using the cyclic-by-row ordering. If Rotation Algorithm 2 or 3 is applied, the norms of columns in each block should be sorted in order. We then consider each block as a super index and follow the designed ordering so that $p(p-1)$ super index pairs can be generated in $p-1$ super steps. In the computation of each super index pair each column in one block must be orthogonalised with each column in the other block once only using the cyclic-by-row ordering, but no columns in the same block are orthogonalised. If a block in a super index pair is considered as the column associated with index i (or index j), the norms of all columns in that block should be increased (or decreased) during the orthogonalisation with the columns in the other block when Rotation Algorithm 2 or 3 is applied. It is easy to show that the sorting procedure is also implemented on the completion of the sweep.

Some of the experimental results from applying different rotation algorithms are given in Table 2. It is easy to see from the table that the program adopting Rotation Algorithm 1 is not as efficient as those adopting rotation algorithms 2 or 3, especially when the problem size is large. If the total number of sweeps is counted, these results are consistent with those in Table 1 (obtained in sequential computation using the cyclic ordering by rows). In our experiment we also measured the sensitivity of the performance to the number of processors used in the computation. The results show that the total number of sweeps required for the computation of the same SVD will not vary as the processor number is changed. Our experimental results are thus clear evidence which shows how important it is to adopt a proper sorting procedure in each sweep.

Size	Sweeps	Time (sec.)	Mflop rate
1000	12	30.7335	1365
1200	12	48.0632	1513
1400	13	73.0251	1626
1600	13	99.4422	1786
1800	13	134.6897	1879
2000	13	175.2245	1983
2400	13	299.7816	2006
2800	13	444.6302	2151
3200	13	625.6428	2285

Table 3: Results on a Fujitsu VPP500 using 4 processors.

PEs	Size	Time (sec.)
4	2000	195
8	2000	102
16	2016	53
32	2048	27

Table 4: Results on a Fujitsu VPP500 using different numbers of processors.

We recently implemented our one-sided Jacobi SVD algorithm on a Fujitsu VPP500. Some experimental results are given in Tables 3 and 4. It can be seen from Table 3 that our algorithm achieves over one third of the peak performance for solving large size problems. (The peak performance of a four-processor VPP500 is 6.4 Gflops.) We can also see from Table 4 that a linear speedup is achieved by using different number of processors (ranging from 4 to 32) for solving a given problem. These results confirm that for massively parallel computation of singular value decomposition the best approach may be to adopt one-sided Jacobi as advocated in [4, 5].

5 The Eigenvalue Problem

The SVD algorithm can be used to find the eigenvalues and eigenvectors of a symmetric matrix. For a symmetric matrix A of size $n \times n$ the one-sided Jacobi method produces an orthogonal matrix V such that $AV = S$, where S has orthogonal columns. We have $S^T S = V^T A^T AV = \Sigma^2$. Thus the eigenvalues λ_i and singular values σ_i of a symmetric matrix are equal, except possibly for signs, i.e. $\lambda_i = \pm\sigma_i$. The signs of the eigenvalues can be obtained using the Rayleigh quotient:

$$\lambda_i = \frac{v_i^T A v_i}{v_i^T v_i}.$$

If we calculate eigenvalues one by one, it is impossible to achieve peak performance. This is because matrix-vector products suffer from the need for one memory reference per multiply-add. The performance may be limited by memory accesses rather than by floating-point arithmetic. In order to achieve high efficiency we should compute all eigenvalues simultaneously using the equation

$$V^T AV = \Lambda$$

(where V is assumed to be orthonormal).

To minimise the communication cost the computation is divided into two steps, i.e., $Y = AV$ and $V^T Y = \Lambda$. There are various parallel algorithms for computing matrix multiplications. We choose an efficient one which places the resulting matrix Y (in the first step) in a natural order. Since V and Y are stored in the same manner and Λ is diagonal, the multiplication of the two matrices in the second step only involves local operations and has operation count $O(n^2)$.

If A is positive definite, an alternative way of finding the eigenvalues and eigenvectors of A is by first computing the Cholesky factorization of A and then performing and SVD [6, 22].

6 Conclusions

We have shown that the one-sided Jacobi method can achieve high efficiency with parallel orderings provided consideration is given to sorting the column norms. Our parallel ring Jacobi ordering can do both index ordering and sorting simultaneously during a sweep. The experimental results show that this ring ordering algorithm can achieve the same convergence rate as the sequential cyclic Jacobi ordering. Some experiments have been conducted on Fujitsu AP1000 and VPP500 computers. We found that for certain problems Jacobi produces results with high accuracy, but QR-based methods do not.

Finally, we point out that the parallel odd-even index ordering [17] and the parallel odd-even transposition sort [1, 2] both have the same communication structure. The two procedures can be combined into a new algorithm which can efficiently implement one-sided Jacobi on general-purpose distributed memory machines [25].

Acknowledgements

The work was partly supported by the Fujitsu-ANU Research Agreement. Thanks are due to Hiroshi Ina and his colleagues at Fujitsu Limited for providing access to a Fujitsu VPP500.

References

[1] S. G. Akl, *Parallel Sorting Algorithms*, Academic Press, Orlando, Florida, 1985.

[2] G. Baudet and D. Stevenson, "Optimal sorting algorithms for parallel computers", *IEEE Trans. on Computers*, C–27, 1978, 84–87.

[3] C. H. Bischof, "The two-sided block Jacobi method on a hypercube", in *Hypercube Multiprocessors*, M. T. Heath, ed., SIAM, 1988, pp. 612-618.

[4] R. P. Brent, "Parallel algorithms for digital signal processing", *Proceedings of the NATO Advanced Study Institute on Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms*, Leuven, Belgium, August, 1988, pp. 93-110.

[5] R. P. Brent and F. T. Luk, "The solution of singular-value and symmetric eigenvalue

problems on multiprocessor arrays", *SIAM J. Sci. and Stat. Comput.*, 6, 1985, pp. 69-84.

[6] J. Demmel and K. Veselić, "Jacobi's method is more accurate than QR", *SIAM J. Sci. Stat. Comput.*, 11, 1992, pp. 1204-1246.

[7] P. J. Eberlein and H. Park, "Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures", *J. Par. Distrib. Comput.*, 8, 1990, pp. 358-366.

[8] L. M. Ewerbring and F. T. Luk, "Computing the singular value decomposition on the Connection Machine", *IEEE Trans. Computers*, 39, 1990, pp. 152-155.

[9] G. E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix", *Trans. Amer. Math. Soc.*, 94, 1960, pp. 1-23.

[10] G. R. Gao and S. J. Thomas, "An optimal parallel Jacobi-like solution method for the singular value decomposition", in *Proc. Internat. Conf. Parallel Proc.*, 1988, pp. 47-53.

[11] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, second ed., 1989.

[12] P. Henrici, "On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing eigenvalues of Hermitian matrices", *J. Soc. Indust. Appl. Math.*, 6, 1958, pp. 144-162.

[13] M. R. Hestenes, "Inversion of matrices by biorthogonalization and related results", *J. Soc. Indust. Appl. Math.*, 6, 1958, pp. 51-90.

[14] T. J. Lee, F. T. Luk and D. L. Boley, *Computing the SVD on a fat-tree architecture*, Report 92-33, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York, November 1992.

[15] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing", *IEEE Trans. Computers*, C-34, 1985, pp. 892-901.

[16] F. T. Luk, "Computing the singular-value decomposition on the *ILLIAC IV*", *ACM Trans. Math. Softw.*, 6, 1980, pp. 524-539.

[17] F. T. Luk, "A triangular processor array for computing singular values", *Lin. Alg. Applics.*, 77, 1986, pp. 259-273.

[18] F. T. Luk and H. Park, "On parallel Jacobi orderings", *SIAM J. Sci. and Stat. Comput.*, 10, 1989, pp. 18-26.

[19] J. C. Nash, "A one-sided transformation method for the singular value decomposition and algebraic eigenproblem", *Comput. J*, 18, 1975, pp. 74-76.

[20] P. P. M. De Rijk, "A one-sided Jacobi Algorithm for computing the singular value decomposition on a vector computer", *SIAM J. Sci. and Stat. Comput.*, 10, 1989, pp. 359-371.

[21] R. Schreiber, "Solving eigenvalue and singular value problems on an undersized systolic array", *SIAM. J. Sci. Stat. Comput.*, 7, 1986, pp. 441-451.

[22] K. Veselić and V. Hari, "A note on a one-sided Jacobi algorithm", *Numerische Mathematik*, 56, 1990, pp. 627-633.

[23] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965, pp. 277-278.

[24] B.B. Zhou and R. P. Brent, "A parallel ordering algorithm for efficient one-sided Jacobi SVD computations", to appear in *Proc. of Sixth IASTED-ISMM International Conference on Parallel and Distributed Computing and Systems*, Washington, DC, October 1994.

[25] B. B. Zhou and R. P. Brent, "On the parallel implementation of the one-sided Jacobi algorithm for singular value decompositions", to appear in *Proc. of 3rd Euromicro Workshop on Parallel and Distributed Processing*, San Remo, Italy, January 1995.