A JACOBI METHOD BY BLOCKS TO SOLVE THE SYMMETRIC EIGENVALUE PROBLEM

Domingo Giménez* Vicente Hernández†‡ Robert van de Geijn§
Antonio M. Vidal†‡

September 29, 1997

Abstract

In this paper, we demonstrate how considerable improvement in the performance of the Jacobi method for solving the symmetric eigenvalue problem can be obtained by reformulating the algorithm to be rich in matrix-matrix multiplication. To achieve this, we borrow techniques developed for parallel Jacobi methods to enhance data locality. By basing the algorithm on the Basic Linear Algebra Subprograms (BLAS) we have achieved an implementation that will perform well independent of the machine where it will be executed, as long as optimized BLAS exist for that machine.

Keywords: Jacobi methods, symmetric eigenvalue problem, algorithms by blocks.

1 Introduction

Many different methods exist for computing the eigenvalues of a symmetric matrix, including bisection methods, the QR algorithm, Divide and Conquer (Cuppen's algorithm), and Jacobi's method. For a complete survey and references, we suggest the reader turn to [12].

Jacobi's method is the oldest, dating back to the mid-1800's. It has fallen out of favor and resurrected on many occasions. Its most recent resurgence is mainly due to better stability properties [6] and staight-forward parallelization.

The primary purpose of this paper is to direct the community's attention to the opportunities that exist to greatly improve the performance of this algorithm by reformulating the algorithm in terms of matrix-matrix operations. This methodology has been very effective for dense linear algebra libraries like LAPACK [2]. The underlying idea is that traditional implementations encounter a bottleneck in modern architectures due to limited bandwidth to main memory. Matrix-matrix operations can be implemented so that more computation is performed between memory accesses. As a result, these operations can take advantage of hierarchical memories that include data caches.

In Section 2, we review some of the basics of Jacobi's method. In Section 3, a Jacobi algorithm by blocks using the odd-even order to solve the symmetric eigenvalue problem will be developed. The theoretical study of the algorithm is done in Section 4. In Section 5 the computation of eigenvectors

*Departamento de Informática y Sistemas. Univ de Murcia. Aptdo 4021. 30001 Murcia. Spain. (domingo@dif.um.es)
†Departamento de Sistemas Informáticos y Computación. Univ Politécnica de Valencia. Aptdo 22012. 46071 Valencia. Spain. (cpvhg@dsic.upv.es, cpavm@dsic.upv.es)
‡Partially supported by ESPRIT III Basic Research Programm of the EC under contract No.9072 (Project GEPPCOM)
§Department of Computer Sciences. University of Texas at Austin. Austin, Texas, 78712. (rvdg@cs.utexas.edu)

is studied. In Section 6 the experimental results obtained on an i860 processor are presented and analyzed. In Section 7 the conclusions are summarized.

2 Jacobi's Method

The eigenvalues and eigenvectors of a real and symmetric matrix A can be obtained by means of Jacobi-like methods. The basic idea is to construct a matrix sequence $\{A_l\}$ by means of

$$A_{l+1} = Q_l A_l Q_l^t \ , \ l = 1, 2, \dots \tag{1}$$

where $A_1 = A$, and Q_l is a plane-rotation (Jacobi rotation) in the (p, q) plane with $1 \leq p, q \leq n$. Under certain conditions [10], sequence $\{A_l\}$ converges to a diagonal matrix D,

$$D = Q_k Q_{k-1} \dots Q_2 Q_1 A Q_1^t Q_2^t \dots Q_{k-1}^t Q_k^t \tag{2}$$

whose diagonal elements are then the eigenvalues of A.

Each product $Q_l A_l Q_l^t$ represents a similarity transformation that annihilates a pair of nondiagonal elements, a_{ij} and a_{ji}, of matrix A_l. Matrix Q_l coincides with the identity matrix except in elements $q_{ii} = c$, $q_{ij} = s$, $q_{ji} = -s$, and $q_{jj} = c$, where $c = \cos\ \theta$, $s = \sin\ \theta$, and [12]

$$\tan\ 2\theta = \frac{2a_{ij}}{a_{ii} - a_{jj}} \ . \tag{3}$$

Notice that this transformation only affects rows i and j when applied from the left, and columns i and j when applied from the right. The different ways of choosing pairs (i, j) have given rise to different versions of the Jacobi method. In the cyclic-by-row method the pairs are taken in the order

$$(1,2), (1,3), \dots, (1,n), (2,3), (2,4), \dots, (2,n), \dots, (n-1,n) \ . \tag{4}$$

A cyclic Jacobi method works by making successive sweeps until some convergence criterion is fulfilled, normally until $off(A) < bound$. A sweep consists of successively nullifying the $n(n-1)/2$ nondiagonal elements in the lower triangular part of the matrix (and the corresponding symmetrical part). The convergence of the cyclic-by-row method is analyzed in [10]. More recently, [8, 9, 13, 14, 15] demonstrate that the Jacobi method converges with other ordering schemes when the cyclic-by-row method is convergent. These orderings are intended to simplify parallelization of the method.

In this paper the odd-even order will be used [16]. Again, this order was intended to allow parallelization of the method, but, as we shall see, also simplifies a block based implementation of the sequential algorithm. We will demonstrate this ordering by examining the case where $n = 8$. Numbering indices from 1 to 8, and grouping initially the indices in pairs $\{(1,2), (3,4), (5,6), (7,8)\}$, the sets of pairs of indices are obtained as follows:

$$\begin{array}{ll}
k = 1 & \{(1,2), (3,4), (5,6), (7,8)\} \\
k = 2 & \{2, (1,4), (3,6), (5,8), 7\} \\
k = 3 & \{(2,4), (1,6), (3,8), (5,7)\} \\
k = 4 & \{4, (2,6), (1,8), (3,7), 5\} \\
k = 5 & \{(4,6), (2,8), (1,7), (3,5)\} \\
k = 6 & \{6, (4,8), (2,7), (1,5), 3\} \\
k = 7 & \{(6,8), (4,7), (2,5), (1,3)\} \\
k = 8 & \{8, (6,7), (4,5), (2,3), 1\}
\end{array} \tag{5}$$

where odd sets have four pairs of indices and even sets have three pairs of indices, so n iterations are needed to obtain the $n(n-1)/2$ pairs of indices. It can be shown that this ordering is equivalent to the row-cyclic ordering, where the rows and columns of the matrix are initially permuted. Grouping pairs of indices using the odd-even order, n iterations are needed to complete a sweep.

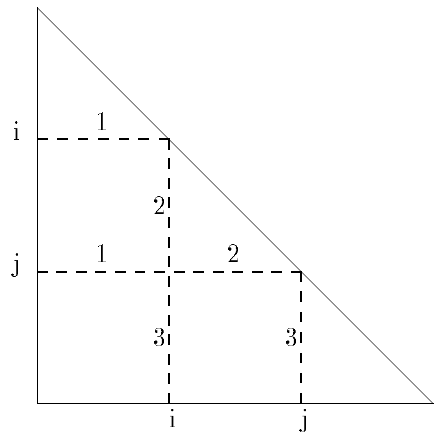

Figure 1: Updating matrix A using BLAS 1.

3 A Jacobi algorithm by blocks

A methodology which it is possible to obtain good results in matrix computation, consists of the design of block algorithms. These algorithms are rich in matrix operations of higher order, which can be implemented using optimized routines, for example, routines of level-3 BLAS [7]. In this way, numerical matrix problems can be solved very efficiently and machine-independent programs can be produced, supposing level-3 BLAS is available for the machine where the program will be run [1, 3, 4, 5]. In this section, we show how similar techniques can be applied to Jacobi's method.

3.1 The Level-1 BLAS implementation

We start by giving pseudo-code for a Jacobi's method implementation:

```
WHILE convergence not reached DO
    FOR every pair (i,j) of indices in a sweep DO
        compute parameters c and s
        update matrix
    ENDFOR
ENDWHILE
```

"Update matrix" consists of applying rotation to rows and columns i and j of A, and this application can be done using the routine `drot` of level-1 BLAS, with three calls to `drot` updating parts 1, 2 and 3 of matrix A in figure 1.

On each sweep "update matrix" is run $n(n-1)/2$ times, and each "update matrix" needs $6n$ floating point operations (flops), thus taking a total of $3n^3$ flops per sweep.

The object of the game now becomes to perform those flops efficiently. In general, vector-vector operations like `drot` are memory intensive, and hence inefficient. Moreover, while applying a rotation from the left, the vectors require a stride through memory (when column-major storage is used), leading to further inefficiencies.

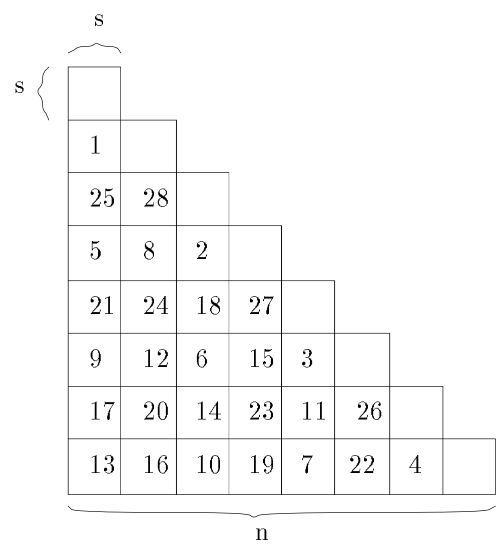

Figure 2: Odd-even order by blocks.

3.2 The use of level-3 BLAS

If matrix A is divided into columns and rows of square blocks of size s, it is possible to make a sweep over these blocks using some Jacobi order, *treating blocks as units*. We will use the odd-even order. So, with $n = 8s$, blocks are treated in the order indicated by the numbers in figure 2, where the work is done on the lower triangular part of the matrix. In each block the algorithm works by making a sweep over the elements in the block, that is, nullifying each element in the block once by means of some order (to program the method an order by rows has been used). But there are some nondiagonal elements that are not in some of the square blocks numbered in figure 2; these are the subdiagonal elements belonging to diagonal blocks. To nullify these elements, blocks 1, 2, 3 and 4 in figure 2 are considered to have size $2s \times 2s$, adding to each block the two adjacent diagonal blocks and the symmetrical block (figure 3).

Then the algorithm works on the blocks in the order 1, 2, 3, 4,..., until a sweep is completed, and making successive sweeps until convergence is reached.

It is possible to design an algorithm rich in level-3 operations (in this case matrix-matrix multiplications) as follows: Sweep over block 1 exploiting the symmetry of the matrix and using level-1 BLAS. In the process, the corresponding rotations are accumulated in form of a matrix Q of size $2s \times 2s$. Finally, the first column of blocks of size $2s \times 2s$ is postmultiplied by this Q. This matrix-matrix multiplication can be done using level-3 BLAS. The work on blocks 2, 3 and 4 is done similarly.

After completing the first sets of blocked rotations, a swap of columns and rows is performed, using the odd-even order. This is an additional task but it brings the next blocks of size $2s \times 2s$ to be nullified to the diagonal, and so it is possible to work with these blocks the same as with blocks 1, 2, 3 and 4. In this case the elements to be nullified are in square blocks of size $s \times s$ inside diagonal blocks of size $2s \times 2s$.

So, if initially the pairs of indices are $\{(1,2),(3,4),(5,6),(7,8)\}$, after swapping indices a new set of pairs is obtained $\{(1,4),(3,6),(5,8)\}$ (5), and similarly swapping rows and columns of blocks a new

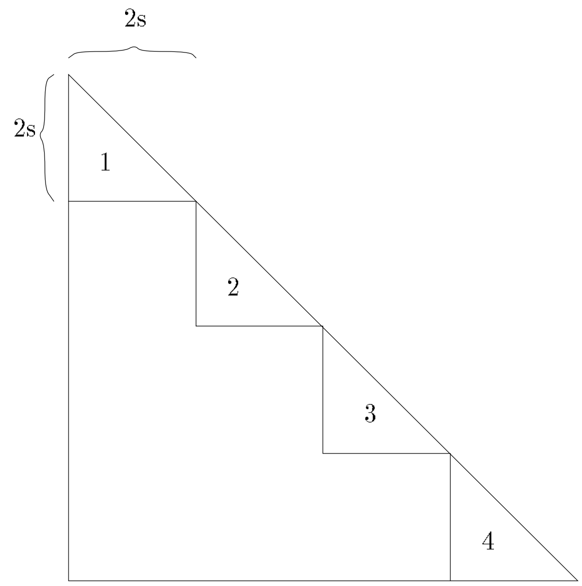

Figure 3: Work on triangular blocks.

distribution is obtained (figure 4). Now the work is done on the diagonal blocks of size $2s \times 2s$ marked in figure 4, but nullifying only elements in blocks 5, 6 and 7. After this the algorithm works similarly with the rest of the blocks until a sweep is completed.

The Jacobi algorithm using this ordering by blocks converges, as can be easily proved using results in [15].

The swapping of columns and rows can be done efficiently using level-1 BLAS to swap and copy part of rows (if matrix A is stored by rows) as is shown in figures 5 and 6, where swaps and copies, done after an odd and even step, are shown. In these figures a bidirectional arrow $\longleftrightarrow$ represents a swap and an unidirectional arrow $\rightarrow$ a copy. Part of the data movement can be avoided because it consists in premultypling and postmultypling the updated matrix $A = QAQ^t$ by a permutation matrix P. Thus, these multiplications can be reorganized in the way $(PQ)A(Q^tP^t)$, and can be done by just swapping rows of Q.

4 Theoretical study

To study the cost of the algorithm only the number of flops will be considered, but there is also an additional time due to the data movement.

It is possible to see in figure 2 that there are

$$\frac{\frac{n}{s}\left(\frac{n}{s}-1\right)}{2} = \frac{q\,(q-1)}{2} \tag{6}$$

blocks of size $s \times s$ ($q = n/s$).

The first $q/2$ blocks are considered triangular blocks of size $2s \times 2s$, thus needing $3(2s)^3$ flops to make a sweep over the block and $3(2s)^3$ flops to accumulate the rotations, thus obtaining $6(2s)^3$ flops.

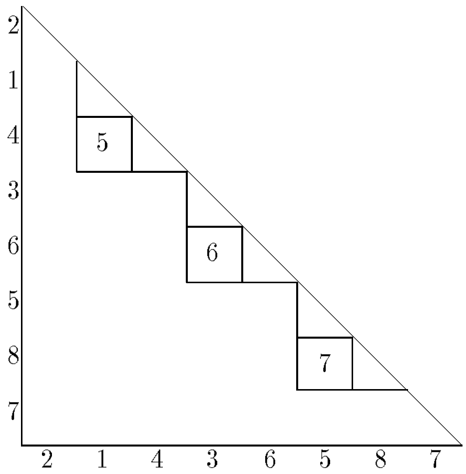

Figure 4: Work on square blocks.

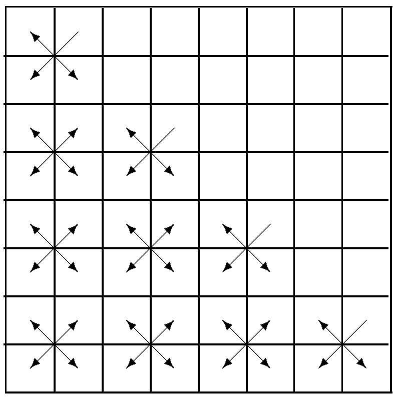

Figure 5: Movement of blocks. Odd step.

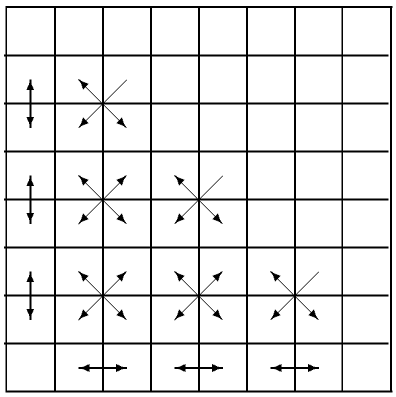

Figure 6: Movement of blocks. Even step.

In the rest of the blocks only s^2 elements are nullified, thus needing $12s^3$ flops per block.
So, the total number of flops to make the sweeps over all blocks is:

$$\frac{q}{2}\ 6\ (2s)^3\ +\ \left(\frac{q^2}{2}\ -\ q\right)\ 12\ s^3\ =\ 6\ n^2\ s\ +\ 12\ n\ s^2 \qquad flops. \tag{7}$$

The two extreme values of s are 1 and $n/2$, so, the extreme values of (7) are $6n^2 + 12n$ and $6n^3$, but optimal execution times will be obtained with moderate values of s.

The rest of the work in the algorithm is the update of rows and columns of blocks. After a sweep over one of the blocks of size $2s \times 2s$, matrices of sizes $2s \times (n - 2s - i)$ and $2s \times i$ are updated (where i takes values $0,\ s,\ 2s, \ldots,\ n - 2s$), and these updates are done by premultiplying or postmultiplying by matrices of size $2s \times 2s$, thus needing $2\,(2s)^2\,(n - 2s) =\ 8s^2\,(n - 2s)$ flops. Because the amount of $s \times s$ blocks is $q(q-1)/2$, the cost of "update matrix" is:

$$\frac{q(q-1)}{2}\ 8s^2\,(n-2s) = n\left(\frac{n}{s} - 1\right) 4s\,(n-2s) = 4n^3 - 12n^2 s + 8ns^2 \qquad flops. \tag{8}$$

So, adding (7) and (8) the cost of the algorithm is obtained (if s is small), but it is important to observe that formula (7) is affected by a coefficient that incorporates the lower execution rate of Level-1 BLAS, and formula (8) by a coefficient corresponding to the higher execution rate of Level-3 BLAS. Anyway, a cost of $4n^3$ flops is obtained. This cost is obtained with Level-3 operations, and on many architectures, a reduction in the execution time would be obtained with respect to a Level-1 version of the Jacobi method, which requires $3n^3$ flops. The improvement depends on the implementations of BLAS 1 and 3 used on the machine where the program is run, but, if s is small in comparison with n (as it seems to be by the experimental results), the quotient between the execution times using BLAS 1 and BLAS 3 would be

$$\frac{3\ k_1}{4\ k_3} \tag{9}$$

where k_1 and k_3 are constants affecting the cost in flops using BLAS 1 and BLAS 3, respectively.

5 Computing eigenvectors

To compute eigenvectors it is necessary to accumulate the orthogonal transformations, and it can also be done using matrix-matrix multiplications. After a sweep over a block of size $2s \times 2s$, it is necessary to multiply the $2s \times 2s$ matrix where the transformations have been accumulated and the corresponding rows of matrix Q where the eigenvectors are computed. To do this efficiently a movement of rows on Q is necessary after each step of the order, and this movement is done permutating rows in the $2s \times 2s$ matrix where rotations were accumulated before updating Q.

The cost of updating Q in each step is $2(2s)^2 n = 8s^2 n$, and $q(q-1)/2$ steps are needed per sweep. So, the cost per sweep of updating Q is:

$$4n^3 - 4n^2 s \qquad flops. \tag{10}$$

The cost of computing eigenvalues and eigenvectors is thus (formulas (7), (8) and (10)):

$$8k_3 n^3 + (6k_1 - 16k_3)n^2 s + (12k_1 + 8k_3)ns^2 \qquad flops \tag{11}$$

and in this case the same quotient (9) is obtained.

6 Experimental results

The results presented here have been obtained on an Intel i860 processor. We choose this microprocessor for two reasons: It is representative of modern microprocessors that utilize hierarchical memory, and it is the processor used for several different parallel architectures, including the Intel and Meiko family of parallel supercomputers. Since we are ultimately interested in parallel implementations, this makes the study of this microprocessor particularly appropriate.

The matrices have been randomly generated with values between -10 and 10, and the programs have been done in C, using double precision. The number of sweeps to reach the convergence is similar to those obtained by cyclic methods and is approximately $log_2\ n$, so, only times per sweep will be studied.

Table 1 shows the execution time per sweep in seconds when only eigenvalues are computed. These times have been obtained with the algorithm using BLAS 1 and the algorithm using BLAS 3 with different block sizes. The optimal value obtained is marked. The optimal value of block size increases with the matrix size, and with the matrices used the optimal value is between 16 and 32.

Table 2 shows the times per sweep when eigenvalues and eigenvectors are computed. These are times obtained with the algorithm using BLAS 3, and the optimal values are marked.

In this case the optimal value of s increases more quickly than when only eigenvalues are computed. This is so because the amount of computation is bigger when eigenvalues and eigenvectors are computed, and it is possible to obtain better performance using level-3 BLAS.

Table 3 shows the quotient between the execution times obtained using BLAS 3 with optimal block size and using BLAS 1, when only eigenvalues are computed. This quotient tends to 1/4 when the matrix size increases.

And table 4 shows the Mflops obtained with the algorithm using BLAS 3 with optimal block size, when only eigenvalues are computed and when eigenvalues and eigenvectors are computed. As can be seen in this table, when eigenvalues and eigenvectors are computed a better performance is obtained.

Finally, we would like to mention that other experiments have been done to improve this algorithm, but without success:

1. Always working with all the blocks of size $2s \times 2s$ (making each subsweep over these blocks as a complete sweep and not only over the block of size $s \times s$). In some cases, with small matrix sizes, a

Table 1: Execution times of algorithms using BLAS 1 and BLAS 3 with different block sizes. Computing eigenvalues.

size	BLAS1	BLAS3. 4	BLAS3. 8	BLAS3. 16	BLAS3. 32	BLAS3. 64
128	1,14	1,27	0,92	**0,89**	1,10	
256	9,25	8,86	5,73	**4,92**	5,48	7,64
384	35,36	28,77	17,44	**14,18**	15,10	21,12
512	88,54	68,09	40,11	**30,83**	31,81	43,62
640	179,20	128,87	74,65	**57,21**	57,72	76,37
768	306,29	220,16	126,73	94,15	**92,30**	117,47
896	505,91	344,50	196,94	146,41	**141,28**	175,10
1024	738,13	510,51	291,65	211,12	**207,18**	244,39
1152	1128,88	718,58	407,78	298,87	**279,90**	332,86

Table 2: Execution times of algorithms using BLAS 3 with different block sizes. Computing eigenvalues and eigenvectors.

size	BLAS3. 4	BLAS3. 8	BLAS3. 16	BLAS3. 32	BLAS3. 64
128	2,08	1,33	**1,15**	1,27	
256	17,47	9,87	7,56	**7,38**	9,06
384	59,40	32,05	23,59	**21,96**	26,53
512	143,53	75,88	53,57	**48,23**	57,15
640		144,92	102,84	**91,48**	104,01

reduction of 1 or 2 sweeps is obtained, but the total execution time is worse than that obtained with the presented algorithm.

2. Increasing the number of subsweeps made over each block in the first sweeps. In some cases, with small matrix sizes, a small reduction in the number of sweeps is obtained, but the total execution time is also worse in this case.

7 Conclusions

The use of BLAS 3 to solve the symmetric eigenvalue problem by Jacobi methods can produce efficient algorithms. The performance of the method depends on the size of the blocks into which the matrix is divided, and a small block size (between 16 and 32) gives good results. The optimal block size increases with the matrix size. When eigenvalues and eigenvectors are computed a bigger optimal block size is obtained. This is due to a higher amount of computations, which produces a better utilization of BLAS 3.

The use of BLAS 3 permits the design of efficient algorithms independently of the machine where the program is run (supposing BLAS 3 is available for this machine).

It is possible to apply these ideas to designing an efficient parallel Jacobi algorithm for the symmetric eigenvalue problem. It is necessary to combine this blocked algorithm with an appropriate storage scheme [11, 17]. We are currently working in this direction.

Acknowledgment: This work was performed in part on the 32 node Intel iPSC/860 operated by the University of Texas Center of High Performance Computing.

Table 3: Quotient between execution times using BLAS 3 and BLAS1. Computing eigenvalues

size	128	256	384	512	640	768	896	1024	1152
quotient	0,78	0,53	0,40	0,34	0,31	0,30	0,27	0,28	0,24

Table 4: Mflops obtained using BLAS3 with optimal block sizes.

size	128	256	384	512	640	768	896	1024	1152
values	9,42	13,64	15,97	17,41	18,32	19,63	20,36	20,73	21,84
values, vectors	14,58	18,18	20,62	22,26	22,92				

References

[1] Amestoy, P. R; Dayde, M. J; Duff, I. S: Use of Level 3 BLAS kernels in the solution of full and sparse linear equations. Proceedings of the European Symposium on High Performance Computing. 1989.

[2] Anderson, E; Bai, Z; Bischof, C; Demmel, J; Dongarra, J; Du Croz, J; Greenbaum, A; Hammarling, S; McKenney, A; Ostrouchov S; Sorensen, D: LAPACK Users' Guide. SIAM. 1992.

[3] Bischof, H: Computing the singular value decomposition on a distributed system of vector processors. Parallel Computing 11. pp 171-186. 1989.

[4] Dackland, K; Elmroth, E; Kågström, B; Van Loan; C. F: Parallel Block Matrix Factorizations on the Shared Memory Multiprocessor IBM 3090 VF/600J. The Inter. Jour. of Supercomputer Applications. V 6. N 1. pp 69-97. 1992.

[5] Dayde, M. J; Duff, I. S: Use of Level 3 BLAS in LU factorization in a multiprocessing environment on three vector multiprocessors: the Alliant FX/80, the Cray-2, and the IBM 3090 VF. The Int. Jour. of Supercomputer Applications. V 5. N 3. pp 92-110. 1991.

[6] Demmel, J; Veselić, K: Jacobi's method is more accurate than QR. SIAM J. Matrix Anal. Appl. V 15. pp 1204-1245. 1992.

[7] Dongarra, J. J; Du Croz, J; Hammarling, S; Duff, I: Algorithm 679: A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs. ACM Trans. on Math. Soft. V 16. N 1. pp 18-28. 1990.

[8] Eberlein, P. J; Mantharam, M: Jacobi Sets for the Eigenproblem and Their Effect On Convergence studied by Graphic Representation. Proc. Fourth SIAM Conf. on Parallel Processing for Scientific Computing. Ed SIAM. pp 15-22. 1989.

[9] Eberlein, P. J; Park, H: Efficient Implementation of Jacobi Algorithms and Jacobi Sets on Distributed Memory Architectures. Jour. Parallel and Distrib. Computing. 8. pp 358-367. 1990.

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

[11] Giménez, D; Hernández, V; Vidal, A. M: Computing Eigenvalues of Symmetric Matrices on a Mesh of Processors. Dept de Informática y Sistemas. Univ de Murcia. TR 1-94. 1994.

[12] Golub, G. H; Van Loan, C. F: Matrix Computations. Second Edition. The Johns Hopkins University Press. 1989.

[13] Luk, F. T; Park, H: A Proof of Convergence for Two Parallel Jacobi SVD Algorithms. IEEE Trans. on Computers, Vol 28, N 6. pp 806-811. 1989.

[14] Mantharam, M; Eberlein, P. J: New Jacobi-sets for parallel computations. Parallel Computing 19. pp 437-454. 1993.

[15] Schroff, G; Schreiber, R: On the convergence of the cyclic Jacobi method for parallel block orderings. SIAM J. Matrix Anal. Appl, Vol 10, N 3. pp 326-346. 1989.

[16] Stewart, G. H: A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. SIAM J. Sci. Stat. Comput. V 6. N 4. pp 853-864. 1985.

[17] Van de Geijn, R. A: A novel storage scheme for parallel Jacobi methods. Dept of Computer Sciences. Univ of Texas at Austin. TR-88-26. 1988.

A JACOBI METHOD BY BLOCKS TO SOLVE
THE SYMMETRIC EIGENVALUE PROBLEM

Domingo Giménez
Vicente Hernández
Robert van de Geijn
Antonio M. Vidal

TR1-95