One-sided Block Jacobi Methods for the Symmetric Eigenvalue Problem *

D. Giménez[1], J. Cuenca[1], R. M. Ralha[2] and A. J. Viamonte[3]

[1] Departamento de Informática y Sistemas. Univ de Murcia.
Aptdo 4021. 30001 Murcia, Spain. ({domingo,jcuenca}@dif.um.es)
[2] Departamento de Matemática. Univ do Minho. Campus Gualtar.
4710 Braga, Portugal. (r_ralha@math.uminho.pt)
[3] CEMPA. Univ Portucalense. R. Dr. António B. de Almeida, 541-619.
4200 Porto, Portugal. (ajs@uportu.pt)

Abstract. In this paper we study theoretically two different one-sided block Jacobi algorithms for solving the Symmetric Eigenvalue Problem. Sequential and parallel versions of the algorithms are analized and compared with a two-sided block Jacobi algorithm. The main advantage of the one-sided algorithms is that they are better suited to parallel computers, and when computing eigenvalues and eigenvectors on multicomputers a more reduced execution time is predicted for the one-sided algorithms than for the two-sided algorithm.

1 Introduction

In this work we studied the design of two one-sided block Jacobi algorithms for the Symmetric Eigenvalue Problem. The algorithms are designed using as a basic the two one-sided Jacobi algorithms proposed in [1].

We begin by explaining how a two-sided block Jacobi method works [2], and after that the two-sided method will be compared with two one-sided block Jacobi methods. The main goal of the comparison is to conclude if one-sided block Jacobi algorithms can be designed maintaining the high degree of parallelism of the algorithms not working by blocks, and the one-sided methods can be competitive with two-sided block algorithms.

2 A two-sided block Jacobi algorithm

The method works over two matrices: the matrix A and a matrix V where the rotations are accumulated. Matrix V is initially the identity matrix. Both matrices A and V are divided into columns and rows of square blocks of size $s \times s$, and these blocks are grouped to obtain bigger blocks of size $2s \times 2s$.

* Partially supported by Comisión Interministerial de Ciencia y Tecnología, project TIC96-1062-C03-02; Consejería de Cultura y Educación de Murcia, Dirección General de Universidades, project COM-18/96 MAT; and Acción Integrada Hispano-Lusa HP1996-0007.

Jacobi methods work by constructing a matrix sequence $\{A_l\}$ by means of $A_{l+1} = Q_l A_l Q_l^t$, $l = 1, 2, \ldots$, where $A_1 = A$. In a non block version of the method, Q_l represents a plane rotation and each product $Q_l A_l Q_l^t$ annihilates a pair of nondiagonal elements, a_{ij} and a_{ji}, of matrix A_l, but in a block version, each Q_l represents a set of rotations that nullify elements in a block of A_l. In each block the algorithm works by making a sweep over the elements in the block. The subdiagonal elements belonging to diagonal blocks will not be zeroed. To correct this, blocks corresponding to the first Jacobi set are considered to be of size $2s \times 2s$, adding to each block the two adjacent diagonal blocks and the symmetrical block. The work over each block can be performed using level-1 BLAS. The corresponding rotations are accumulated to form a matrix Q of size $2s \times 2s$. Finally, the corresponding columns and rows of blocks of size $2s \times 2s$ of matrix A and the rows of blocks of matrix V are updated using Q. These matrix-matrix multiplications can be effected using level-3 BLAS.

After completing a set of blocked rotations, a swap of column and row blocks is performed, according to the order we are using. The odd-even order will be used, [3], because it simplifies a block based implementation of the sequential algorithm, and allows parallelization. If $n = 8$, numbering indices from 1 to 8, and initially grouping the indices in pairs $\{(1,2),(3,4),(5,6),(7,8)\}$, the sets of pairs of indices are obtained as follows: $\{(1,2),(3,4),(5,6),(7,8)\}$, $\{2,(1,4),(3,6),(5,8),7\}$, $\{(2,4),(1,6),(3,8),(5,7)\},\ldots$.

This data movement brings the next blocks of size $s \times s$ to be zeroed to the subdiagonal, and the process continues similarly to operations performed in the first step. However, in this case the elements to be nullified are in square blocks of size $s \times s$ inside diagonal blocks of size $2s \times 2s$. This data movement will imply data transferences in the parallel version of the algorithm.

The cost per sweep is:

$$8k_3n^3 + (12k_1 - 16k_3)\, n^2 s + 8k_3 n s^2 \quad flops, \tag{1}$$

where k_1 and k_3 represent the cost of an arithmetic operation performed using BLAS 1 or BLAS 3, respectively.

2.1 A parallel algorithm

It is possible to obtain a balanced algorithm for a ring. Grouping blocks of size $2s \times 2s$ of matrix A and V in bigger blocks A_{ij} and V_{ij} of size $2sk \times 2sk$, we assign to each processor P_i, with $p = \frac{q}{2}$ and $\frac{n}{2sk} = q$, rows of blocks i and $q-1-i$ of matrices A and V. Therefore, each processor P_i contains blocks A_{ij} and $A_{q-1-i,j}$, with $0 \le j \le i$, and V_{ij} and $V_{q-1-i,j}$, with $0 \le j < q$.

Due to the data movement between odd and even steps, it is necessary to reserve some additional memory, and $(2sk+s)(3n+2sk+2s)$ positions of memory are reserved on each processor.

The arithmetic cost per sweep when computing eigenvalues and eigenvectors is:

$$8k_3\frac{n^3}{p} + (12k_1 - 8k_3)\frac{n^2 s}{p} + 12k_1\frac{ns^2}{p} \quad flops. \tag{2}$$

And the cost per sweep of the communications is:

$$\beta(p+3)\frac{n}{s} + \tau\left(8n^2 + 2ns - \frac{2n^2}{p}\right) \quad , \tag{3}$$

where β and τ represent the start-up and the word-sending time, respectively.

3 A one-sided block Jacobi algorithm. First version

We analized the first one-sided Jacobi algorithm in the paper [1]. The algorithm works on matrices $B_0 = A$ and $W_0 = I$, obtaining $B_{r+1} = V_r B_r$, $W_{r+1} = V_r W_r$, with V_r the rotation matrix nullifying a non-diagonal element of matrix $A_r = B_r W_r^t = V_{r-1}V_{r-2}\dots V_0 B_0 W_0^t V_0^t \dots V_{r-1}^t$.

To nullify a_{ij} it is necessary to compute a_{ii}, a_{jj} and a_{ij}, because the algorithm works on matrices B_r and W_r, and not on matrix A_r. These elements are obtained with three dot products. After that, rows i and j of B_r and W_r are updated. If the diagonal elements are stored in an auxiliary vector, it is not necessary to compute a_{ii} and a_{jj} every time, and the cost per sweep is:

$$7n^3 - \frac{15}{2}n^2 + \frac{n}{2} \quad flops. \tag{4}$$

We propose a one-sided block Jacobi algorithm by combining the ideas of the two-sided block algorithm and the ideas of the one-sided algorithm.

Matrices B and W, of size $n \times n$, are divided in blocks of size $s \times n$, and blocks of $A = BW$ are treated using the odd-even ordering.

Initially the $\frac{n}{2s}$ blocks corresponding to the first Jacobi set are treated, making a two-sided sweep on blocks of size $2s \times 2s$ of matrix A and accumulating rotations. These operations are done using BLAS 1.

After that, matrices B and W are updated multiplying the rotation matrices, of size $2s \times 2s$, by the corresponding blocks of B and W, of size $2s \times n$. In this case matrices B and W are not symmetric.

In the two-sided algorithm a movement of rows and columns of blocks is performed in order to have the blocks grouped according to the next Jacobi set. This movement can be included in the updating of the matrix if it is done on the rotation matrix before updating A. In the one-sided algorithm the movement of rows of blocks of B and W can be done in the same way (figure 1).

In successive steps it is necessary to compute A_{ii}, A_{jj} and A_{ij}, because the work is not done directly with matrix A. If we call B_i and W_i the i-th row of blocks of B and W in figure 1.a), $A_{ii} = B_i W_i^t$, $A_{jj} = B_j W_j^t$ and $A_{ij} = B_i W_j^t$. If the diagonal blocks are stored it is not necessary to compute A_{ii} and A_{jj}.

After the blocks A_{ii}, A_{jj} and A_{ij} are computed, a matrix of size $2s \times 2s$ is formed, and a two-sided sweep is performed on this matrix, accumulating the rotations.

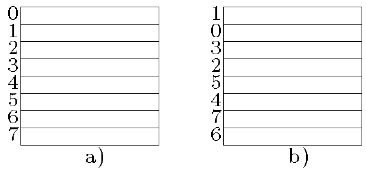

Fig. 1. Distribution of matrices B and W on the first one-sided block algorithm: a) initially, b) after application of the first set of rotations.

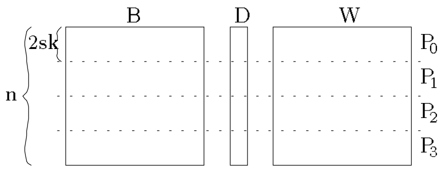

Fig. 2. Initial distribution of matrices B, D and W in the system of processors for the first one-sided parallel block algorithm.

The cost per sweep is:

$$9k_3n^3 + (12k_1 - 9k_3)\,n^2s \qquad flops. \tag{5}$$

3.1 A parallel algorithm

It is possible to assign to each processor k consecutive blocks of size $2s \times n$, with $n = 2skp$, of matrices B and W (figure 2). In the figure the distribution of the matrices is shown, but also in this case it is necessary to reserve some additional memory to store data in sucessive steps of the algorithm. The quantity of memory reserved in each processor is $(2k+1)sn$ to store elements of B, the same quantity to store elements of W, and $(2k+1)s^2$ to store elements of D.

The arithmetic cost per sweep is:

$$9k_3\frac{n^3}{p} + 12k_1\frac{n^2s}{p} + 12k_1\frac{ns^2}{p} \qquad flops. \tag{6}$$

It is not necessary to broadcast the rotation matrices because each processor updates the rows of blocks it contains. The only communications are those between steps to group data according to the next Jacobi set. In odd steps blocks of size $s \times n$ of B, and W, and a diagonal block of size $s \times s$ are sent from P_i to P_{i-1}, with $i = 1, 2, \ldots, p-1$, and in even steps the same communications are done from P_{i-1} to P_i. Therefore, the cost per sweep of communications is:

$$2\frac{n}{s}\beta + \left(4n^2 + 2ns\right)\tau \quad . \tag{7}$$

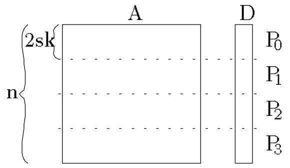

Fig. 3. Initial distribution of matrices A and D in the system of processor for the second one-sided parallel block algorithm.

4 A one-sided Jacobi algorithm. Second version

The second one-sided Jacobi algorithm proposed in [1] has the advantage of a lower execution time, but also has the disadvantage of a worse precision [4].

The method works by diagonalizing matrix $B = A^2$ but without explicitly form B. Rotations V nullifying elements b_{ij} of B are applied to A. If initially $A_1 = A$ and $B_1 = A_1 A_1^t$, we will have $A_{r+1} = V_r A_r$, and A must be updated only by one side. Because $B_r = A_r A_r^t$, it is necessary to perform dot products to obtain b_{ii}, b_{jj} and b_{ij}, which are needed to obtain the next Jacobi rotation.

The cost per sweep is approximately $4n^3$ flops if the elements of the diagonal are stored.

The method has some problems derived from the fact that the eigenvalues computed are those of A^2, but not the eigenvalues of A [1, 4].

To design an algorithm by blocks matrix A is divided into consecutive blocks of size $s \times n$.

Before each subsweep on a block B_{ii}, B_{jj} and B_{ij}, are computed (or only B_{ij} if the diagonal blocks are stored). Even if the diagonal blocks are stored, in the first step all the blocks must be computed, because the algorithm works with A and not with B.

The cost per sweep is:

$$5k_3 n^3 + (12k_1 - 5k_3)\, n^2 s \quad flops. \tag{8}$$

4.1 A parallel algorithm

The distribution of matrix A and matrix D, where the diagonal blocks are stored, can be that shown in figure 3. Also in this case it is necessary to reserve some additional memory. The size of memory reserved on each processor to store data from matrix A is $(2k+1)sn$ and to store data from matrix D is $(2k+1)s^2$.

The arithmetic cost per sweep is:

$$5k_3 \frac{n^3}{p} + 12k_1 \frac{n^2 s}{p} + 12k_1 \frac{ns^2}{p} \quad flops. \tag{9}$$

Table 1. Predicted execution time of different parallel block Jacobi algorithms.

$n = 1024$	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$
$two - sided$	412.3	217.5	108.9	54.6	27.5	15.9	9.4	
$one - sided, version1$	463.8	246.5	123.6	62.1	31.3	16.0	9.3	5.2
$one - sided, version2$	257.6	143.1	71.7	36.0	18.1	9.2	5.2	2.9

The only communications are those produced by the data movements between steps. In odd steps $s(n+s)$ elements are sent from P_i to P_{i-1}, and in even steps the same quantity is sent from P_i to P_{i+1}. The cost per sweep of communications is:

$$\frac{2n}{s}\beta + \left(2n^2 + 2ns\right)\tau \quad . \tag{10}$$

5 Comparison and Conclusions

The first version of the one-sided algorithm has a higher cost than the two-sided method, but the difference is smaller in the algorithms working by blocks than in the algorithms not working by blocks. The second one-sided algorithm has the lowest cost, but has worse precision. Communications are less costly in the one-sided algorithms because it is not necessary to broadcast the rotation matrices. Also in the communications the second one-sided algorithm is better because it works with one matrix and only half of the data must be transferred. Table 1 shows the execution time predicted on the Touchstone Delta for matrix size 1024 and a variable number of processors. The estimated values of the constants are ([2]) $k_1 = 0.137\mu s$, $k_3 = 0.048\mu s$, $\beta = 61\mu s$ and $\tau = 0.149\mu s$. We can see the behaviour of the one-sided algorithms is better when the number of processors increases. This is why it could be interesting to implement the algorithms here analized and to compare them experimentally. This is what we are doing now.

References

1. B. A. Chartres. Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store. *The Computer Journal*, 5:51–60, 1963.
2. D. Giménez, V. Hernández, R. van de Geijn and A. M. Vidal. A block Jacobi method on a mesh of processors. *Concurrency: Practice and Experience*, 9(5):391–411, May 1997.
3. G. H. Stewart. A Jacobi-like algorithm for computing the Schur decomposition of a nonhermitian matrix. *SIAM J. Sci. Stat. Comput.*, 4:853–864, 1985.
4. Ana J. Viamonte. Métodos de Jacobi para o Cálculo de Valores e Vectores próprios de Matrizes Simétricas. Tesis de Mestrado. Universidade do Minho. 1996.

This article was processed using the LaTeX macro package with LLNCS style