Implementation of parallel one-sided block Jacobi methods for the symmetric eigenvalue problem [1]

Javier Cuenca
Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia. 30071 Murcia. Spain.
javiercm@ditec.um.es

Domingo Giménez
Departamento de Informática, Lenguajes y Sistemas Informáticos
Universidad de Murcia. 30071 Murcia. Spain.
domingo@dif.um.es

1 Introduction

The Symmetric Eigenvalue Problem (SEP) can be solved in at least three different ways [10]: 1) methods working by reduction of matrices into certain condensed form, like the QR-algorithm, 2) Jacobi-like methods, and 3) spectral division methods. Jacobi method is the oldest but the interest in Jacobi's approach is renewed due to its inherent parallelism and good stability [3]. Traditional implementations of dense linear algebra algorithms encounter a bottleneck in modern architectures due to limited bandwidth between the CPU and main memory. Using algorithms by blocks, matrix-matrix operations can be arranged so that more computation is performed between memory accesses. As a result, these operations can take advantage of hierarchical memories. A suite of such matrix-matrix operations are part of the basic linear algebra subprograms (BLAS) [4]. An algorithm coded in terms of calls to BLAS becomes a portable high performance implementation. The use of blocks also allows us to design parallel algorithms with low communication necessities.

In this work we study the implentation of two one-sided block Jacobi algorithms for the Symmetric Eigenvalues Problem. These algorithms were designed using as a basic the one-sided Jacobi algorithms proposed in [2] and the two-sided block Jacobi algorithm proposed in [7]. We prove experimentally that the one-sided algorithms are better suited to parallel computers, obtaining a bigger reduction of the execution time than the reduction obtained by the two-sided algorithms. This reduction of the execution time is obtained because in one-sided Jacobi algorithms there is less communication among the processors than in two-sided algorithms and also because in one-sided algorithms the data are always accessed in the same order they were stored. In contrast to other block methods [1], these algorithms work with several sizes of blocks independently of the size of the matrix and the number of processors used.

[1]Partially supported by Comisión Interministerial de Ciencia y Tecnología, project TIC96-1062-C03-02.

In Section 2, the two-sided Jacobi method, a blocks version and a parallel algorithm are described. In Sections 3 and 4 the same with the first and the second version of one-sided Jacobi algorithm is made, respectively. Finally, in Section 5 a theoretical comparison of the three methods is made and the experimental results obtained in different multiprocessors systems are shown.

2 A two-sided block Jacobi algorithm

A Jacobi method works with two matrices: matrix A and matrix V where the rotations are accumulated. Matrix V is initially the identity matrix. This method constructes a matrix sequence $\{A_l\}$ by means of $A_{l+1} = Q_l A_l Q_l^t \;\; l = 1, 2, \ldots$, where $A_1 = A$ and Q_l is a plane-rotation that anhilates a nondiagonal element of matrix A_l.

In a block version, 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 $2s \times 2s$. 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 it, 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 performed using level-3 BLAS. After completing a set of block rotations, a swap of column and row blocks is performed, according to the order we are using. The odd-even order has been used because it simplifies a block based implementation of the sequential algorithm, and allows parallelization. The data movement can be included in the updating of the matrices if it is done on the rotation matrix before updating them. 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 of this block algorithm when computing eigenvalues and eigenvectors 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 and BLAS 3, respectively.

2.1 A parallel algorithm

Parallel versions could be developed for a square mesh or a triangular mesh [6], but, in order to compare the algorithm with the block one-sided algorithms that we have developed, an implementation for a ring topology will be analised. A ring of processors is the more logic topology for one-sided Jacobi algorithms, although more scalable implementations can be dessigned for a mesh[9].

Considering $q = \frac{n}{2sk}$, and a ring with $p = \frac{q}{2}$ processors: $P_0, P_1, ..., P_{p-1}$, a balanced algorithm can be obtained assigning to each processor P_i, with $0 \leq i < p$, the rows i and $q-1-i$ of matrices A and V. So, each processor P_i contains blocks A_{ij} and $A_{q-1-i,j}$, with $0 \leq j \leq i$, and V_{ij} and $V_{q-1-i,j}$, with $0 \leq j \leq q$.

Due to the data movement between odd and even steps, it is necessary to reserve some additional memory, so $(2sk+s)(3n+2sk+2s)$ 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:

$$(p+3)\frac{n}{s}\beta + \left(10n^2 + 4ns - \frac{n^2}{p}\right)\tau, \tag{3}$$

where β and τ represent the start-up and the word-sending time, respectively. If we use, instead of a multiprocessor, several computers communicated by a local network, like Ethernet, with a shared physical medium the cost per sweep of the communications is:

$$(p+3)\frac{n}{s}\beta + (p-1)\left(5n^2 + 2ns + \frac{n^2}{2p}\right)\tau, \tag{4}$$

because when the physical medium is shared the communications between two pairs of computers cannot be overlapped.

3 A one-sided block Jacobi algorithm. First version

The first one-sided algorithm we analyse works on matrices $B_0 = A$ and $W_0 = I$, obtaining $B_{r+1} = Q_r B_r$ and $W_{r+1} = Q_r W_r$, with Q_r the rotation matrix nullifying a nondiagonal element of matrix $A_r = B_r W_r^t = Q_{r-1}Q_{r-2}....Q_0 B_0 W_0^t Q_0^t....Q_{r-1}^t$.

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} each time, and the cost per sweep is:

$$7n^3 - \frac{15}{2}n^2 + \frac{n}{2} \quad flops. \tag{5}$$

A one-sided block algorithm is obtained by combining the previous ideas with the ideas of the two-sided block algorithm [5]. The matrices A and W, of size $n \times n$, are divided in blocks of size $s \times n$, and the 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 the rotations. These operations

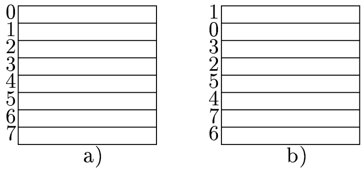

Figure 1: Distribution of matrices B and W on the first one-sided block algorithm: a) initially, b) after application of the first set of rotations.

are done using BLAS 1. After that, matrices A and W are updated multiplying the rotation matrices, of size $2s \times 2s$, by the corresponding blocks of B and W, of size $2s \times n$, using BLAS 3. After completing a set of block rotations, a movement of rows is done like in the two-sided algorithm (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, using them a block of size $2s \times 2s$ is formed and the process continues as above. If the diagonal blocks are stored it is not necessary to compute A_{ii} and A_{jj}, and then the cost per sweep is:

$$9k_3 n^3 + (12k_1 - 9k_3)\, n^2 s \qquad flops. \tag{6}$$

3.1 A parallel algorithm

A parallel version of this algorithm in a ring of p processors consists of assigning to each processor k consecutive blocks of size $2s \times n$, with $n = 2skp$, of matrices B and W. The distribution of matrices B, W and the matrix D, where the diagonal blocks are stored, is shown in figure 2. In this algorithm it is also necessary to reserve additional memory to store data in sucessive steps of the algorithm, so the quantity of memory reserved in each processors is $(2k+1)(2sn+s^2)$.

The arithmetic cost per sweep is:

$$9k_3\frac{n^3}{p} + 12k_1\frac{n^2 s}{p} + 12k_1\frac{ns^2}{p} \qquad flops. \tag{7}$$

In this method 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 $s \times s$ of matrix D are sent from P_i to P_{i-1}, with $1 \leq i < p$ and in even steps the same communications are done from P_i to P_{i+1}, with $0 \leq i < (p-1)$. Therefore, the cost of communications per sweep is:

$$3\frac{n}{s}\beta + \left(4n^2 + 2ns\right)\tau, \tag{8}$$

If we use, instead of a multiprocessor, several computers communicated by a local network, like Ethernet, with a shared physical medium, the cost of communications per sweep is:

$$3\frac{n}{s}\beta + (p-1)\left(2n^2 + ns\right)\tau. \tag{9}$$

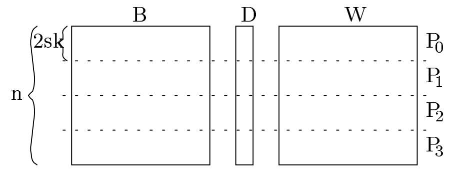

Figure 2: Initial distribution of matrices B, D and W in a system of processors for the first one-sided parallel block algorithm.

4 A one-sided block Jacobi algorithm. Second version

The second one-sided Jacobi algorithm proposed in [2] has the advantage of a lower cost, but also has the disadvantage of a worse precision [11]. This algorithm works by diagonalizing matrix $B = A^2$ without constructing B. Rotations nullifying elements b_{ij} of B are applied to A. If iniatilly $A_1 = A$ and $B_1 = A_1 A_1^t$, we will have $A_{r+1} = Q_r A_r$, where Q_r is a plane-rotation that anhilates a nondiagonal element of matrix B_r, so 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_{ij}, b_{ii} and b_{jj}, which are needed to obtain the next Jacobi rotation.

To design an algorithm by blocks the 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). The reduction is terminated when no rotation is applied in a sweep, in this moment all pairs of rows of A are mutually orthogonal, i.e., A is of the form DW with D a diagonal matrix and W a matrix whose rows are the eigenvectors of A, and the eigenvalues of A are obtained as the square roots of the norms of these rows. So, this method has worse precision due to we do not calculate directly the eigenvalues of A.

The cost per sweep is:

$$5k_3 n^3 + (12k_1 - 5k_3)\, n^2 s \quad flops. \tag{10}$$

Even if the diagonal blocks are stored, in the first step all the blocks must be computed with an additional cost of order $n^2 s$, because the algorithm works with A and not with B.

4.1 A parallel algorithm

A parallel version of this algorithm in a ring of p processors consists of assigning to each processor k consecutive blocks of size $2s \times n$, with $n = 2skp$, of matrix A. The distribution of the matrix A and matrix D, where the diagonal blocks are stored, is shown in figure 3. It is also necessary to reserve additional memory to store data in sucessive steps of the algorithm, so the quantity of memory reserved in each processor is $(2k+1)(sn+s^2)$.

The arithmetic cost per sweep is:

$$5k_3 \frac{n^3}{p} + 12k_1 \frac{n^2 s}{p} + 12k_1 \frac{ns^2}{p} \quad flops. \tag{11}$$

The only communications are those between steps to group data according to the next Jacobi set. In odd steps $s(n+s)$ elements are sent from P_i to P_{i-1}, with $1 \le i < p$, and in

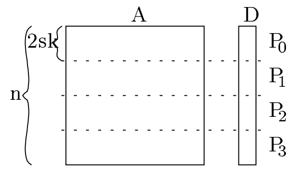

Figure 3: Initial distribution of matrices A and D in the system of processor for the second one-sided parallel block algorithm.

$Proces.$	1	1	1	2	2	2	4	4	4
$size/alg$	$t-s$	$o-s,1$	$o-s,2$	$t-s$	$o-s,1$	$o-s,2$	$t-s$	$o-s,1$	$o-s,2$
$Execution time$									
256	7.72	10.12	5.94	4.46/0.9	4.86/1.0	2.43/1.2	2.81/0.7	2.74/0.9	1.52/1.0
384	25.82	36.83	19.27	16.41	18.73	7.55	12.40	11.77	4.18
640	120.82	171.80	92.95	82.31	88.72	44.45	55.52	51.42	27.54
$Efficiency$									
256	1.0	1.0	1.0	0.9	1.0	1.2	0.7	0.9	1.0
384	1.0	1.0	1.0	0.8	1.0	1.3	0.5	0.8	1.2
640	1.0	1.0	1.0	0.7	1.0	1.0	0.5	0.8	0.8

Table 1: Execution time, in seconds, per sweep and efficiency of different parallel block Jacobi algorithms in PenFEt.

even steps the same quantity is sent from P_i to P_{i+1}, with $0 \leq i < (p-1)$. Therefore, the cost of communications per sweep is:

$$2\frac{n}{s}\beta + \left(2n^2 + 2ns\right)\tau. \tag{12}$$

If we use, instead of a multiprocessor, several computers communicated by a local network, like Ethernet, with a shared physical medium, the cost of communications per sweep is:

$$2\frac{n}{s}\beta + \left(n^2 + ns\right)(p-1)\tau. \tag{13}$$

5 Comparison and Experimental Results

The first version of the one-sided algorithm has higher arithmetic cost than the two-sided method, but the level-3 BLAS operations in the two-sided algorithm are made along rows and columns; however, in the one-sided algorithm these operations are made only along rows, therefore are less costly. The second one-sided algorithm has the lowest cost, but, as we have previously mentioned, also 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 than the first one-sided algorithm because it works with one matrix and only half of the data must be transferred.

$Processors$	2	2	2	4	4	4
$size/alg$	$two-sided$	$one-sided,1$	$one-sided,2$	$two-sided$	$one-sided,1$	$one-sided,2$
			$SunEt$			
256	3.40	2.68	1.30	4.21	3.50	1.75
512	31.89	18.36	7.01	32.53	15.78	7.77
1024	211.83	89.00	66.14	165.62	91.60	56.14
			$SP2Sw$			
256	0.60	0.59	0.33	0.50	0.35	0.18
512	4.26	4.05	2.26	3.05	2.43	1.39
768	12.85	13.46	7.47	8.29	7.44	4.29

Table 2: Execution time, in seconds, per sweep of different parallel block Jacobi algorithms in SunEt and in SP2Sw.

$Processors$	2	3	4	5	6	8	10	12	14
				$PenFEt$					
$two-sided$	0.63	1.45	2.81	4.59	6.97				
$one-sided, version1$	0.64	1.49	2.75	3.92	5.80				
$one-sided, version2$	0.36	0.93	1.52	1.20	1.79				
				$SunEt$					
$two-sided$	0.67	1.72	4.21	12.86					
$one-sided, version1$	0.48	1.27	3.50	6.86					
$one-sided, version2$	0.21	0.65	1.75	3.66					
				$SP2Sw$					
$two-sided$	0.12	0.20	0.39	0.24	1.03	4.35	12.70	11.04	27.38
$one-sided, version1$	0.08	0.16	0.35	0.12	0.48	1.81	3.69	5.55	7.13
$one-sided, version2$	0.05	0.10	0.18	0.30	0.22	0.94	1.74	2.60	4.82

Table 3: Comparison of times per sweep of different parallel block Jacobi algorithms with a scaled problem: $n = 64p$.

Consistently, we can predict that when the number of processors increase or/and when the values of the communication constants τ and β are high the execution time of the first one-sided algorithm will be less than that of the two-sided algorithm.

The algorithms have been compared in different parallel systems using MPI[8] and matrices generated randomly with values between -10 and 10. The parallel systems we have used are:

1.- PenFEt : 6 PC Pentiums 200Mhz connected by a Fast Ethernet 100 Mbps.

2.- SunEt : 5 workstations Sun Ultra1 143Mhz connected by a Ethernet 10Mbps.

3.- SP2Sw : a multiprocessor IBM-SP2 with 32 thin160 640Mflops/s connected by a switch network 120 MB/s.

In table 1 we can see the good efficiency obtained when we paralyze one-sided Jacobi methods, because the cost of the communications does not increase so much as in the two-sided one.

In table 2 we can see the second one-sided algorithm is faster than the two others. The first one-sided algorithms is better than the two-sided algorithm mainly in systems where the communications are more costly (SunEt), this difference decreases in systems where the cost of the communications is less in relation to the arithmetic cost (SP2Sw).

In table 3 we can see the time execution of the one-sided algorithms increases more slowly than the time execution of the two-sided one when the size of the problem is increased proportionally.

References

[1] P. Arbenz and M. Oettli. *Block Implementations of the Symmetric QR and Jacobi Algorithms.* Technical Report 178. Institute for Scientific Computing, ETH Zürich, 1992.

[2] B. A. Chartres. *Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store.* The Computer Journal, 5:51-60, 1963.

[3] J. Demmel and K. Veselić. *Jacobi's method is more accurate than QR.* SIAM J. Matrix Anal. Appl., 13, 1204-1245 (1992).

[4] J. J. Dongarra, J. Du Croz, S. Hammarling and I. Duff. *A set of level 3 basic linear algebra subprograms.* ACM Trans. Math. Softw., 16, (1), 1-17 (1990).

[5] D. Giménez, J. Cuenca, R. M. Ralha and A. J. Viamonte. *One Sided Block Jacobi Methods for the Symmetric Eigenvalue Problem.* Proceedings of the 3^{rd} International Meeting on Vector and Parallel Processing. 1998, Porto (Portugal). 687-692.

[6] D. Giménez, V. Hernández and A.M. Vidal. *A Unified Approach to Parallel Block-Jacobi Methods for the Symmetric Eigenvalue Problem.* Proceedings of the 3^{rd} International Meeting on Vector and Parallel Processing. 1998, Porto (Portugal). 139-151.

[7] 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.

[8] *MPI2: a message-passing interface standard.* The International Journal of High Performance Computing Applications, volume 12, number 1/2, 1998. Sage Science Press

[9] D. Royo, M. Valero and A. González *A Jacobi-bases Algorithm for Computing Symmetric Eigenvalues and Eigenvectors in a Two-dimensional Mesh.* Proceedings of the Euromicro Workshop on Parallel and Distributed Processing 1998. Madrid. 463-469

[10] Xiaobai Sun. *Parallel Algorithms for Dense Eigenvalue Problems.* High Performance Computing and Gigabit Local Area Networks. Essen, Germany. 1996. 202-212.

[11] A. J. Viamonte. *Metodos de Jacobi para o Calculo de Valores e Vectores proprios de Matrizes Simetricas.* Tesis de Mestrado. Universidade do Minho. 1996.