Proceedings of the 9th WSEAS International Conference on Applied Mathematics, Istanbul, Turkey, May 27-29, 2006 (pp580-586)

High Relative Precision of Eigenvalues Calculated with Jacobi Methods

ANA JULIA VIAMONTE*, RUI RALHA**

*Departamento de Inovação, Ciência e Tecnologia
Universidade Portucalense
Rua Dr. Ant. Bernardino de Almeida, 541-619, 4200-072 Porto
PORTUGAL

** Departamento de Matemática
Universidade do Minho
Campus de Gualtar, Braga
PORTUGAL

Abstract: – The revolution in numerical methods started by parallel computation brought new life to the 'old' Jacobi method. In recent works, one-sided Jacobi methods have been proposed for the calculation of the eigenvalues of symmetric matrices because they are highly suited to parallel computing and allow the calculation of eigenvalues and eigenvectors with high relative precision. We make a study of the parallelization of two different one-sided block Jacobi algorithms. The one-sided variants are interesting for parallel computers because they significantly reduce the communication between processors; furthermore, the reorganization of algorithms into blocks allows the use of BLAS3 modules which decreases the memory access time. In this paper we present some results obtained with our implementations of the proposed one-sided algorithms.

Key-Words: Eigenvalues, One-sided Jacobi Algorithms, Block Methods, Relative precision.

1 Introduction

The Jacobi's method for reducing a real symmetric matrix to diagonal form has a long history. First proposed in 1846, it was for many years the preferred method for calculating the eigenvalues and eigenvectors of symmetric matrices. With the development of more efficient algorithms in the sixties, the Jacobi methods were only used when the execution time was not a primary factor. In fact, for medium to large-sized matrices, Jacobi's methods require more computational time than the methods that first reduce the matrix to tridiagonal form (QR and "divide-and-conquer").

The revolution in numerical methods started by parallel computation brought new life to the 'old' Jacobi method, on account of its easy implementation on parallel machines.

The classical Jacobi method is a two-sided method but for parallel computation in distributed memory machines the one-sided variants are more interesting because they significantly reduce the communication between processors. Furthermore, the reorganization of algorithms into blocks considerably decreases the memory access time.

In this study, we perform a comparison between Jacobi methods and other methods in order to assess the relative numerical precision of results for some symmetric matrices.

In section 2 we present the sequential Jacobi algorithms, in section 3 we describe the corresponding block versions, in section 4 the parallel algorithms are proposed and in section 5 some numerical results are given; finally, some conclusions and drawn in section 6.

2 Sequential Algorithms

The classic two-sided Jacobi algorithm reduces a given symmetric matrix A to diagonal form by a sequence of plane rotations; the basic idea of one-

sided methods is to apply each rotation on the left side or on the right side of A only.

In the first one-sided algorithm studied, Method I, we compute the sequence of matrix pairs (B_k, V_k), k = 1, 2,…, defined by

$$\begin{cases} B_{k+1} = R_k B_k \\ V_{k+1} = R_k V_k \end{cases}$$

with $B_0 = A$ and $V_0 = I$. For every value of k, the product

$$B_k V_k^T = V_k A V_k^T = A_k$$

is a symmetric matrix similar to A. Consequently, if we choose the angles θ_k as in the double-sided process, we execute essentially the same transformation. However, we do not compute the matrix A_k explicitly but keep it in the form of the two factors B_k and V_k, instead. Now, the calculation of θ_k involves the (p, q), (p, p) and (q, q) elements of $V_k A V_k^T$. In the two-sided Jacobi method these are immediately available but in Method I they must be computed as scalar products of the pth and qth rows of B_k and V_k. As in the classical algorithm, the process terminates when $A_k = B_k V_k^T$ is almost diagonal. The eigenvalues of A are the diagonal elements of $B_k V_k^T$ and the eigenvectors are the rows of V_k.

In our implementation of Method I, we do not apply a rotation when the entry to be annihilated is smaller than a quantity which depends upon a threshold which gets smaller in every sweep as the method progresses. We do not go into details here. A very simply description of the method is as follows:

Algorithm 1

```
while (not converged)
  for p =1 to (n-1)
   for q=(p+1) to n
     compute entries  in positions (p, p),
     (p, q) and (q, q) of matrix A_k
        if abs(Apq/sqrt(App×Aqq)) > threshold
            compute the rotation R_k in the plane (p, q)
            update rows p and q of B_k and V_k
        end if
     end for q
   end for p
   update threshold
end while
```

In the second one-sided Jacobi method studied, Method II, we do not diagonalize A but we convert $M = A^2$ into diagonal form, instead. In fact, we compute the sequence of matrices C_k, k = 1, 2, ... based upon the formulae

$$\begin{cases} C_k = R_{k-1}\, C_{k-1} \\ M_k = C_k\, C_k^{\mathrm{T}} \end{cases}$$

with $C_0 = A$. Hence

$$M_k = R_{k-1}\, M_{k-1}\, R_{k-1}^{\mathrm{T}}$$

and we choose the angles θ_k in such a way that M_k converges to a diagonal matrix. Again, the calculation of θ_k requires the entries (p, p), (p, q), and (q, q) of M_k to be known and, since we do not form this matrix explicitly, those entries must be computed as scalar products with the pth and qth rows of C_k. This is the price to be paid in any one-sided method.

Therefore, we see that Method II converts $M = A^2$ to diagonal form in an implicit manner by making rows of C_k mutually orthogonal. This can be understood as a method that computes the eigenvalues of A from the square roots of the singular values of A^2. Because of this, the accuracy of the smaller eigenvalues may be unsatisfactory when A is not well-conditioned. The situation may be improved by using Method II for the computation of the eigenvectors only and by using the Rayleigh Quotient for the calculation of the corresponding eigenvalues, i. e.,

$$\lambda = x^{\mathrm{T}} A x / x^{\mathrm{T}} x$$

where x is one row of matrix C_k and λ the corresponding eigenvalue. A simple description of the method follows:

Algorithm 2

```
while (not converged)
  for p =1 to (n-1)
    for q=(p+1) to n
      compute entries in positions (p, p),
      (p, q) and (q, q) of matrix M_k
      if abs(Mpq/sqrt(Mpp×Mqq)) > threshold
         compute the rotation R_k in the plane (p, q)
         update rows of matrix C_k (C_k = R_k C_k)
      end if
    end for q
  end for p
  update threshold
end while
compute eigenvalues (Rayleigh quotient)
```

3 Block Algorithms

3.1. Method I

As we have seen, this method works with two matrices, B and V. We now decompose B and V to get matrices of square blocks of size b × b; the block algorithm works with these blocks in a way that mimics the way used by the non-blocked algorithm for simple scalar entries; inside each block, the algorithm works by making a sweep over the elements in the block.

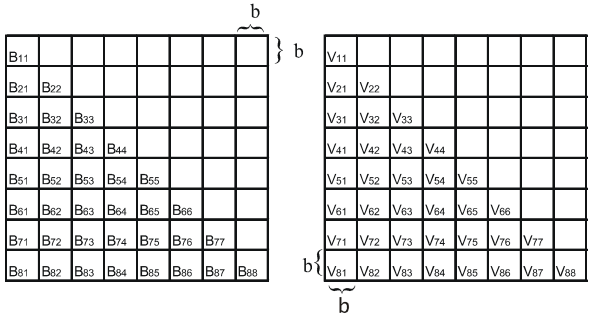

Fig.1: block decomposition of matrices B and V.

Jacobi methods produce a sequence of matrices $A_{k+1} = Q_k A_k Q_k^t$, k = 0, 1,....; in the classical method Q_k represents a simple plane rotation and each update $Q_k A_k Q_k^t$ annihilates a pair of off-diagonal elements of A_k, in the block version each Q_k represents a set of rotations that brings to zero all the entries in a block.

With each pair of consecutive diagonal blocks of A_k in positions (2i-1,2i-1) and (2i,2i), for i=1,…,n/2b, we form a symmetric block of order 2b by joining to these diagonal blocks the off-diagonal blocks in positions (2i,2i-1) and (2i,2i-1). Exploiting symmetry, we only need to annihilate the entries below the main diagonal, i.e., the entries below the main diagonal in each one of the two diagonal blocks (we will refer to these as triangular sweeps) and all the entries in the off diagonal block (we will refer to this as a square sweep). The work over each one of these (2b × 2b) blocks is performed using level-1 BLAS. The corresponding rotations are accumulated to form a matrix Q of order 2b. Finally, the corresponding rows of blocks of order 2b of matrices B and V are updated with Q. These matrix-matrix multiplications are carried out using level-3 BLAS.

After completing a set of block rotations, a permutation of rows of B_k and V_k is performed; this corresponds to a permutation of rows and also columns of A_k. We have used the odd-even order [3] because it simplifies the block version of the sequential algorithm and allows parallelization. If n = 8, numbering indices from 1 to 8, and initially grouping the indices in the 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)}, {(1,7), (2,4), (3,6), (5,8)}, {(1,4), (2,6), (3,8), (5,7)}, … .

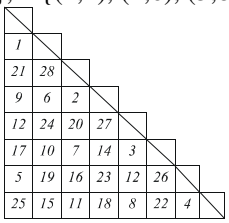

Fig.2: the odd-even order for n=8.

Each permutation brings a new set of n/2b blocks of size b × b into the annihilating positions (2i,2i-1). In the parallel version of the algorithm, the accomplishment of these permutations requires some data transference between processors.

The cost per sweep of the algorithm when computing eigenvalues and eigenvectors is:

$$11k_3n^3 + (12k_1 - 19k_3)n^2b + 8k_3nb^2$$

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

A very simplified description of the block version of Method I is as follows:

Algorithm 3

while (not converged)
  for i = 1 to n/2b
    Compute blocks A(2i-1,2i-1), A(2i,2i) and A(2i,2i-1)
    Execute triangular and square sweeps and accumulate rotations in Q (2b×2b).
    Update matrices B and V (B =QB, V = QV)
  end i
  apply permutation according to odd-even order
end while

3.2 Method II

The second one-sided Jacobi algorithm works with only one matrix C_k such that $A_k^2 = M_k = C_k C_k^T$. This method has the advantage of being fast but, as

said before, it is less accurate. The strategy for the design of the block version of this algorithm is similar to that used in Method I: M_k is implicitly decomposed into blocks of size b×b and with each pair of consecutive diagonal blocks of M_k in positions (2i-1,2i-1) and (2i,2i), for i=1,…,n/2b, we form a symmetric block of size 2b × 2b by joining to these diagonal blocks the off-diagonal blocks in positions (2i,2i-1) and (2i,2i-1). Before each sweep, the blocks of M_k need to be computed from the corresponding blocks of our working matrix C_k.

The cost per sweep is:

$$7k_3n^3 + (12k_1 - 15k_3)n^2b + 8k_3nb^2$$

Let A a real symmetric matrix, and $C_0 = A$. If *n* is the dimension of the matrix and *b* the dimension of the block, the block version of Method II can be described as follows:

Algorithm 4
while (not converged)
 for i = 1 to n/2b
  Compute blocks M(2i-1,2i-1), M(2i,2i) and M(2i,2i-1)
  Execute triangular and square sweeps and accumulate rotations in Q (2b×2b).
  Update matrix C (C =QC)
 end i
 apply permutation according to odd-even order
end while

4 Parallel Algorithms

One-sided algorithms are well suited to parallel implementation. If we assign a number of complete rows of B and V to each processor, then, unlike in the two-sided algorithm where each rotation has to be broadcast, communication is required only between adjacent processors, to exchange rows. The key of a parallel algorithm is a mobile scheme that reduces communication between processors and brings each off-diagonal entry to an annihilating position in the course of each sweep.

The algorithm is organized in terms of n/2 computational nodes, each node dealing with a pair of rows; we will assume that these computational nodes are connected in a ring. In each step, node j, j = 1, …, n/2, performs a rotation in the (2j-1,2j) plane that is intended to introduce, in an implicit manner, a zero in positions (2j-1,2j) and (2j,2j-1) of a symmetric matrix. One sweep of Method I can be described as follows:

Algorithm 5 (node j, j=1,…, n/2)
for i = 1 to n-1
 Compute A(2j-1,2j-1), A(2j-1,2j) and A(2j, 2j)
 Compute the rotation R in the plane (2j-1, 2j).
 Update rows 2j-1 and 2j of B and V
 Move Rows of B and V
end i

With the same data distribution, one sweep of Method II can be described as follows:

Algorithm 6 (node j, j=1,…, n/2)
for i = 1 to n-1
 Compute M(2j-1,2j-1), M(2j-1,2j) and M(2j, 2j)
 Compute the rotation R in the plane (2j-1, 2j).
 Update rows 2j-1 and 2j of C (C=RC)
 Move Rows of C
end i

For the Rayleigh quotient improvement of the eigenvalues, each processor will use each one of the local rows of C to compute approximations for the corresponding eigenvalues. Assuming that each processor has kept in its local memory the assigned rows of the matrix A, the parallel implementation of the Rayleigh quotient requires more communication since each processor will need the complete A to compute products Ax.

Let A be a real symmetric matrix of order n, $B_0 = A$ and $V_0 = I$. If *b* is the dimension of the block and *nbp* the number of blocks by processor, for each node j, the parallel implementation of the block version of Method I is as follows:

Algorithm 7:
while (not converged)
 For i = 1 to $nbp/2$
  Compute the local blocks of matrix A.
  Execute triangular sweep in the block 2b×2b
  Update rows and columns in matrix Q
  Update rows of local matrices locB and locV
{locB = Q locB and locV = Q locV}
 Communication (even step)
 For i = 1 to $(n/b - 2)$
  If step is odd
   for j =1 to $nbp/2$
    Compute the corresponding local blocks of matrix A
    Execute total Sweep in the block b×b

Update rows and columns in matrix Q
Update rows of local matrices locB and locV {locB = Q locB and locV = Q locV}
Communication (odd step)
If step is even
for j =1 to $nbp/2$
Compute the corresponding local blocks of matrix A
Execute total Sweep in the block b×b
Update rows and columns in matrix Q
Update rows of local matrices locB and locV {locB = Q locB and locV = Q locV}
Communication (even step)
Compute Eigenvalues

Let A a real symmetric matrix and C_0 = A. If *n* is the dimension of the matrix, *b* the dimension of the block and *nbp* the number of blocks by processor. For which node j, the Method II by blocks is defined by the passes,

Algorithm 8:
While (convergence is not obtained)
For i = 1 to $nbp/2$
Compute the local blocks of matrix $M=A^2$.
Execute triangular sweep in the block 2b×2b
Update rows and columns in matrix Q
Update local rows locC {locC = Q locC}
Communication (even step)
For i = 1 to $(n/b - 2)$
If step is odd
for j =1 to $nbp/2$
Compute the corresponding local blocks of matrix $M = A^2$
Execute total Sweep in the block b×b
Update rows and columns in matrix Q
Update locC {locC = Q locC}
Communication (odd step)
If step is even
for j =1 to $nbp/2$
Compute the corresponding local blocks of matrix $M = A^2$
Execute total Sweep in the block b×b
Update rows and columns in matrix Q
Update rows of local matrix locC {locC = Q locC}
Communication (even step)
Compute Eigenvalues and eigenvectors

5 Numerical Results

5.1 Accuracy of results

In [4], Demmel and Veselic proved that for scaled matrices H = DAD, with $D = diag(H^{1/2})$ and A a matrix whose diagonal elements are equal to the unity, the accuracy of the eigenvalues computed with the Jacobi method depend upon $k(A)$ instead of $k(H)$. More precisely, it has been shown that we have for the relative errors of the eigenvalues of H computed with the Jacobi method

$$\frac{\left|\lambda_i - \hat{\lambda}_i\right|}{\lambda_i} \leq \eta k(A)$$

where η denotes a quantity not much larger than the arithmetic precision. Therefore, when $k(H) >> k(A)$ the Jacobi method is more accurate than QR.

We now present some results obtained for some matrices of this type. These results have been produced with the following codes: two-sided Jacobi method (*JBS*), one-sided Jacobi method I (*J1S*), one-sided Jacobi method II (*J2S*), block one-sided method I (*J1SB*), block one-sided method II (*J2SB*), functions **eig** (*MATLAB*), **dsyevx** (*LAPACK*) and **lin_eig_self** (*IMSL*).

In a first test we considered the following matrix (symmetric positive definite)

$$H = \begin{bmatrix} 10^{60} & 10^{49} & 10^{39} & 10^{29} \\ 10^{49} & 10^{40} & 10^{29} & 10^{19} \\ 10^{39} & 10^{29} & 10^{20} & 10^{9} \\ 10^{29} & 10^{19} & 10^{9} & 1 \end{bmatrix}.$$

We have H=DAD with

$$D = \begin{bmatrix} 10^{30} & 0 & 0 & 0 \\ 0 & 10^{20} & 0 & 0 \\ 0 & 0 & 10^{10} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and}$$

$$A = \begin{bmatrix} 1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 1 & 0.1 \\ 0.1 & 1 & 0.1 & 1 \end{bmatrix}$$

In Table 2 we present the maximum relative error of the computed eigenvalues of H and the ortogonality of the eigenvectors produced by each one of the 8 codes

	$\left\|V^tV - I\right\|$	$\max\left\|\left(\lambda_i - \hat{\lambda}_i\right)/\lambda_i\right\|$
JBS	*1,18e-32*	*1,78e-16*
J1S	*1,18e-32*	*1,78e-16*
J2S	*1,35e-21*	*1,78e-16*
J1SB	*1,18e-32*	*1,78e-16*
J2SB	*2,25e-16*	*1,67e-16*
MATLAB	*1,90e-17*	*1,89e+33*
LAPACK	*1,83e-32*	*7,31e-04*
IMSL	*2,96e-16*	*1,85e-06*

Table 2 – Relative error of the eigenvalues and ortogonality of the eigenvectors of H

As expected, the implementations of the Jacobi methods produce eigenvalues with better relative precision that the other implementations which are based upon the QR method. We carried out other tests that essentially produce similar results. From a random symmetric matrix A with $k(A) \approx 1{,}4e+3$, we generated a very ill-conditioned H=DAD using a diagonal D with elements of very different magnitudes. The results are given in Fig.3 and Fig.4; for the block methods, square blocks of dimension 2, 4, 8 and 16 have been used.

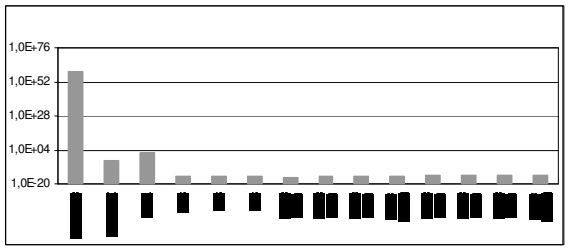

Fig.3: relative error of the eigenvalues of a matrix DAD of order 512

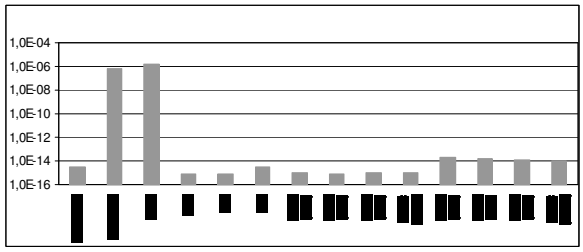

Fig.4: orthogonality of the eigenvectors of a matrix DAD of order 512

5.2 Efficiency

In Fig.5 we display the time (in seconds) taken by each one of the tested codes for the case of our matrix of order 512. It shows that the Jacobi methods are in fact much slower than the competitors.

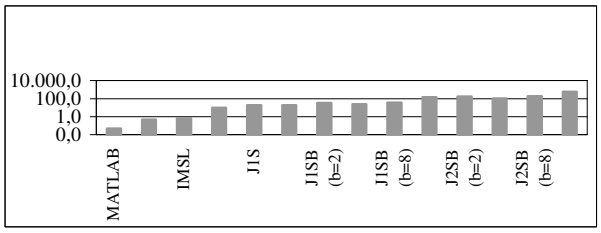

Fig.5: Execution time (in seconds) for a matrix of order 512

In Fig.6 we compare the execution time of the parallel algorithms.

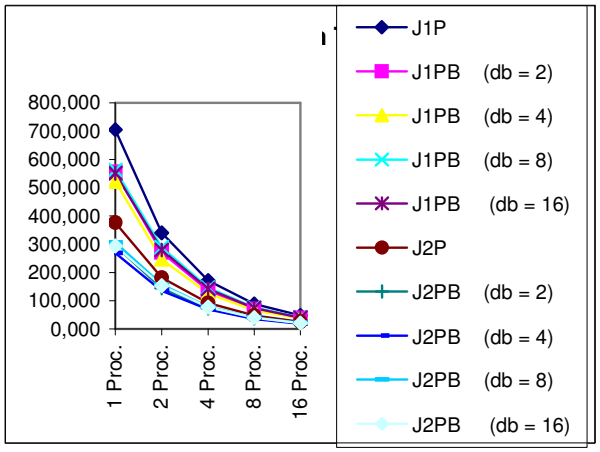

Fig.6: Execution time (in seconds) for the parallel algorithms

In Fig.7 we display the efficiency of the parallel algorithms.

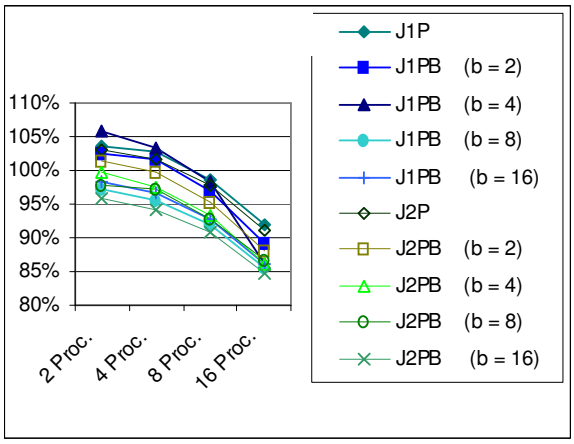

Fig.7: Efficiency of parallel algorithms

6 Conclusions

The choice of the method for the determination of the eigenvalues and eigenvectors of a given matrix is not a simple problem. Although Jacobi methods are slower than methods that first reduce the matrix to tridiagonal form, those are able to find all eigenvalues with high relative accuracy when the matrix is well scaled. Therefore, Jacobi methods are of interest in applications that require results with high precision.

Moreover, Jacobi methods are better suited for parallel implementation on systems with hundreds or thousand of processors. One-sided variants reduce significantly the communication and may become viable in the context of grid computing applications that require the computation of eigenvalues and eigenvectors of very large dense matrices.

We have considered the parallel implementation of two one-sided Jacobi algorithms. One of these one-sided methods (method I) is equivalent to the standard two-sided method. Method II is faster than method I but not as accurate.

The block methods are as accurate as the non-block versions and considerably reduce the memory access time.

References:

[1] A. Chartres, Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store, *The Computer Journal*, No.5, 1963, pp.51-60.

[2] Ana Julia Viamonte, *Métodos de Jacobi para o Cálculo de Valores e Vectores Próprios de Matrizes Simétricas*, Master Thesis, University of Minho, Braga, Portugal, July 1996.

[3] Ana Julia Viamonte and R. M. Ralha, Parallel Jacobi Algorithms for the Computation of the Eigensystem of Real Symmetric Matrices, *VECPAR96*, June 1996.

[4] J. Demmel e K. Veselic, Jacobi's Method Is More Accurate Than QR, *SIAM J. Matrix Anal. Appl.*, Vol.4, No.13, 1992, pp.1204-1245.

[5] D. Gimenez, V. Hernandez, R. van de Geijn and A. M. Vidal, A block Jacobi method on a mesh of processors, *Concurrency: Practice and Experience*, Vol.5, No.9, 1997, pp. 391-411.

[6] D. Gimenez, M. J. Majado, R. M. Ralha and A. J. Viamonte, One-sided block Jacobi Methods for the Symmetric Eigenvalue Problem, *VECPAR98*, June 1998.

[7] Ana Julia Viamonte, *Métodos Unilaterais de Jacobi para a Computação paralela de Valores e Vectores Próprios de Matrizes Simétricas*, PhD Thesis, University of Minho, Braga, Portugal. September 2003.