A Jacobi-based Algorithm for Computing Symmetric Eigenvalues and Eigenvectors in a Two-dimensional Mesh

Dolors Royo, Miguel Valero-García, and Antonio González
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
c/ Jordi Girona 1-3, Campus Nord - Edifici D6
E-08034 Barcelona (Spain)
Phone. +34-3-401 7189
E-mail: {dolors, miguel, antonio}@ac.upc.es

Abstract

This paper proposes an algorithm for computing symmetric eigenvalues and eigenvectors that uses a one-sided Jacobi approach and is targeted to a multicomputer in which nodes can be arranged as a two-dimensional mesh with an arbitrary number of rows and columns. The algorithm is analysed through simple analytical models of execution time, which show that an adequate choice of the mesh configuration (number of rows and columns) can improve performance significantly, with respect to a one-dimensional configuration, which is the most frequently considered scenario in current proposals. This improvement is especially noticeable in large systems.[1]

1 Introduction

Many authors have focused on the problem of computing the eigenvalues and eigenvectors of a real symmetric matrix on different types of computing machines, including systolic arrays (i.e. [2], [6]) shared memory multiprocessors (i.e. [1]) and distributed memory multiprocessors (multicomputers, for short) (i.e. [4]) Those proposals in which the target system is a mesh-connected multicomputer are particularly relevant, since this type of parallel machine organization is considered to be a promising platform for large scale efficient supercomputing.

In this paper, the target system is a 2^d-node wormhole multicomputer, arranged as a $2^r \times 2^c$ two-dimensional four-neighbour mesh, with any value for r and c, provided that $r+c = d$. Therefore, the results of this paper are useful in the context of application specific mesh design or in the context of multicomputers which allow such an arbitrary configuration of the nodes, as it is the case of transputer-based systems. Additionally, the results of the paper are useful as a first step to solve the problem of eigenvalue and eigenvector computation on a multicomputer with a fixed configuration of nodes (i.e. a squared two or three dimensional mesh), which is the subject of further work [8]

Proposals for parallel eigenvalue and eigenvector computation are frequently based on Jacobi methods because, although they require more arithmetic computations than others (e.g. QR iteration), they exhibit a large degree of potential parallelism. The one-sided Jacobi method [3] is particularly attractive since it has lower communication requirements than the alternative two-sided method [10]

Several approaches have been proposed for the parallel organization of the computations in a Jacobi method. Such approaches are referred to as Jacobi orderings A particular organization of a Jacobi ordering in the form of a set of cooperating processes (or nodes) is referred to as a Jacobi algorithm.

Parallel Jacobi algorithms for eigenvalue computation on mesh multicomputers known to us use a one-dimensional organization of the nodes, either with a wraparound link (i.e. rings) or without it (i.e. lines). For this reason, they will be called one-dimensional algorithms. See [4], [2], [9] for a few relevant proposals of one-dimensional algorithms.

This paper proposes a novel Jacobi algorithm that uses a two-dimensional organization of the nodes and will be called the two-dimensional algorithm. The algorithm uses the one-sided Jacobi method and a Jacobi ordering used in a one-dimensional algorithm proposed in [4]. We show that the two-dimensional algorithm is more efficient on the target system than the one-dimensional algorithm, since it has a lower communication cost. This will be shown through simple analytical models of performance, which enable us to derive the number of rows (2^r) and columns (2^c) that minimize the execution time on the configurable mesh.

The ideas brought together in this paper can be used to derive other two-dimensional algorithms which use alternative Jacobi orderings. Additionally, our proposal can also be used for singular value decomposition (SVD) since Jacobi methods can also be applied in this matrix

1. This work was supported by the Ministry of Education and Science of Spain (CICYT TIC-429/95)

0-8186-8332-5/98 $10.00 © 1998 IEEE

computation [11].

The rest of the paper is organized as follows: Section 2 describes the target system, the problem, and the one-dimensional algorithm that is used for comparison purposes. Section 3 describes the proposed two-dimensional algorithm. Section 4 develops analytical models of execution time which are used to evaluate the proposal. Finally, section 5 summarizes the main conclusions of this work.

2 Background

2.1 Target architecture

Consider a multicomputer system with 2^d nodes that can be arranged as a $2^r \times 2^c$ two-dimensional (2D) four-neighbour mesh (neighbours in the north, east, west and south directions), with any value for r and c, provided that $r+c = d$. We assume full-duplex links between nodes and a communication system that uses the wormhole switching model [7]. In this model, the header of the message is sent through the network to establish a path between the source and the destination. Then, the rest of the message is sent in a pipeline fashion. The conclusions of this work can be readily extended to systems with circuit switching or virtual cut-through routing. This is because the differences among these models, which are found in the strategies to establish the path or to solve a conflict in the links or buffers, are relevant neither to the proposed algorithms nor the evaluation assumptions.

A one-port model is assumed with regard to the node organization [7]. In this model, every node can send/receive only one message to/from the network, at the same time. Finally, we assume a synchronous communication model. In other words, a sending node is blocked until the message has been completely received by the destination node.

For algorithm modelling purposes, we assume that nodes spend a time t_c to perform a floating-point computation, and a time $t_s + N \times t_e$ to send a message of N floating point numbers (as is common, in a wormhole system, the time required to establish the path between the sending and receiving nodes is neglected in comparison to the startup time t_s and the transmission time $N \times t_e$).

2.2 Jacobi methods for symmetric eigenvalue and eigenvector computation

Let A be a $m \times m$ real symmetric matrix. λ is an eigenvalue of A if there exists a vector x such that.

$$Ax = \lambda x$$

Vector x is the eigenvector associated with eigenvalue λ. A Jacobi method for eigenvalue and eigenvector computation is an iterative procedure aimed at reducing A to a diagonal form by applying a series of similarity transformations. Since similarity transformations preserve the eigenvalues, the elements in the diagonal of the resulting matrix are the eigenvalues of A.

Plane rotations are used to build every similarity transformation [5]. Specifically, in a given iteration k of the algorithm, a plane rotation R_k is used to zero one pair of off-diagonal symmetric element, by applying a similarity transformation as follows:

$$A_{k+1} = R_k^T \times A_k \times R_k \text{ for } k=0,1,2,\ldots \text{ and } A_0 = A$$

The plane rotation R_k required to zero elements $A_k(p,q)$ and $A_k(q,p)$ is built upon the values $s = \sin(\alpha_{p,q})$ and $c=\cos(\alpha_{p,q})$, being $\alpha_{p,q}$ the rotation angle computed from $A_k(p,q)$, $A_k(p,p)$ and $A_k(q,q)$ as described in [5][1].

A total of $m(m-1)/2$ distinct transformations are required to zero all the off-diagonal elements exactly once. This collection of transformations is called a sweep. The Jacobi method is iterative because a transformation may fill elements that had been zeroed in previous transformations and therefore several sweeps are required so that A approaches a diagonal form.

The convergence rate of the algorithm depends on the ordering in which the transformations are applied in one sweep. The classic Jacobi ordering zeroes the off-diagonal elements in the order determined by their absolute value, from the highest to the lowest. When using this ordering, the algorithm converges very quickly, but spends most of the time searching for the next element to be zeroed. In the cyclic Jacobi ordering, the off-diagonal elements are zeroed according to a predetermined ordering (i.e., by rows or by columns). Even though the cyclic Jacobi ordering needs more sweeps to converge, it is in general faster than the classic ordering since every sweep has a lower cost. Finally, when required, the eigenvectors are obtained from the columns of matrix $R_0 \times R_1 \times R_2 \times \ldots$

The method described so far is referred to as the two-sided Jacobi method, since in iteration k matrix A_k is multiplied from the left (by R_k^T) and from the right (by R_k), involving row and column update. Due to this feature, the two-sided method incurs a high communication cost when implemented on a multicomputer, where matrices are distributed among the nodes.

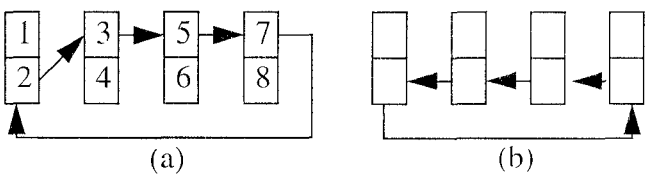
As an alternative to the two-sided method, the one-sided method organizes the computations in such a way that only column updates are required. Specifically, the one-sided method organizes the computations as follows [3]:

$$\overline{A}_0 = A \text{ and } U_0 = I$$
$$\overline{A}_{k+1} = \overline{A}_k \times R_k \text{ and } U_{k+1} = U_k \times R_k \text{ for } k=0,1,2,\ldots$$

Since $A_k = R_k^T \times \overline{A}_k$, the values of $A_k(p,q)$, $A_k(p,p)$ and $A_k(q,q)$, which are required in iteration k to compute the rotation angle $\alpha_{p,q}$ used to build R_k, are recovered from $\overline{A}_k$ as follows:

---

1 From now on, R will be used to denote either the rotation matrix or the similarity transformation which uses the rotation matrix R. In addition, when required, $R(p,q)$ denotes the similarity transformation which zeroes elements $A(p,q)$ and its symmetric



(a) (b)

step	(c)	step	(d)
1	(1,2)(3,4)(5,6)(7,8)	1	(1,2)(8,7)(6,5)(4,3)
2	(1,7)(2,4)(3,6)(5,8)	2	(1,4)(2,7)(8,5)(6,3)
3	(1,4)(2,6)(3,8)(5,7)	3	(1,7)(2,5)(8,3)(6,4)
4	(1,5)(4,6)(2,8)(3,7)	4	(1,6)(7,5)(2,3)(8,4)
5	(1,6)(4,8)(2,7)(3,5)	5	(1,5)(7,3)(2,4)(8,6)
6	(1,3)(6,8)(4,7)(2,5)	6	(1,8)(5,3)(7,4)(2,6)
7	(1,8)(6,7)(4,5)(2,3)	7	(1,3)(5,4)(7,6)(2,8)

Figure 1: Column exchange in the 1D algorithm: (a) Pattern exchange for odd steps (b) Pattern exchange for even steps. (c) Column pairing for odd sweeps (d) Column pairing for even sweeps.

$$A_k(p,q) = < U_k(*,p), \overline{A}_k(*,q), >$$
$$A_k(p,p) = < U_k(*,p), \overline{A}_k(*,p), >$$
$$A_k(q,q) = < U_k(*,q), \overline{A}_k(*,q), >$$

where $<x,y>$ denotes the inner product of vectors x and y and $\overline{A}(*,p)$ denotes column p of $\overline{A}$. If the eigenvector are to be computed[1], as assumed in our paper, both one-sided and two-sided methods require the same amount of computation. On the other hand, both recovering the elements of A and applying the transformation involve only column operations, which is a very interesting property for parallel distributed memory implementation Because of this potential benefit in a parallel environment, we focus on the one-sided Jacobi method

2.3 Parallelism in Jacobi methods

An interesting property of Jacobi methods is that several transformations can be applied in parallel to zero several elements, reducing in this way the time required to complete a sweep. For example, elements $A(1,2)$ and $A(3,4)$ (and their symmetrics) can be zeroed in parallel because transformation $R(1,2)$ updates only columns 1 and 2 of A and transformation $R(3,4)$ updates only columns 3 and 4. For this reason, these transformations are said to be independent transformations. In general, two transformations $R(i,j)$ and $R(r,s)$ are independent if $i \neq r$, $i \neq s$, $j \neq r$, and $j \neq s$. This feature has motivated the proposal of parallel Jacobi orderings in which the similarity transformations required to complete a sweep are organized into groups of independent transformations. Each of these groups will be called a step Parallel algorithms using such Jacobi orderings can exploit the parallelism provided by a multicomputer since the work associated with one step can be distributed among the nodes in the system.

Parallel Jacobi algorithms have been proposed for different parallel computing scenarios (systolic arrays, shared memory multiprocessors, hypercube multicomputers, mesh-connected multicomputers). In the particular case of mesh-connected multicomputers, the one-dimensional organization of the nodes (either as a line or as a ring) has been the most frequent assumption.

1. Note that in the one-sided method, the eigenvectors are the columns of U_∞

2.4 A one-dimensional algorithm

In this section we describe in detail a parallel algorithm which uses one of the Jacobi orderings proposed in [4]. This parallel algorithm will be referred to as one-dimensional (1D) since it uses a one-dimensional organization of the nodes. The 1D algorithm will be used here to motivate our proposal and for comparison purposes.

The 1D algorithm uses 2^d nodes arranged as a ring (a line with a wraparound link) The algorithm is first described assuming that $m=2^{d+1}$ Initially, every node stores two columns of $\overline{A}_0=A$ and the corresponding columns of $U_0=I$ Figure 1a shows this initial column distribution, for the particular case of $2^d = 4$ (columns of $\overline{A}$ and U are numbered from 1 to m)

Every sweep has $2^{d+1}-1$ steps, each of them consisting of $m/2$ independent transformations. In every step, each node performs one of the independent transformations and exchanges one column of $\overline{A}$ and the corresponding column of U, with one of the nodes of the ring. This column exchange is carried out according to the patterns shown in figure 1a (for odd steps) and figure 1b (for even steps). Figure 1c shows the pairs of columns that constitute every step in the first sweep. In this figure, a pair (i,j) indicates that the corresponding node, say q, computes in that step the rotation $R(i,j)$ and updates $\overline{A}$ and U. The one-sided approach guarantees that the only data required to perform this computation are columns i and j of matrices $\overline{A}$ and U, which reside in node q during the step. Figure 1d shows the pairs which constitute every step in the second sweep. In general, odd sweeps are as in figure 1c and even sweeps are as in figure 1d.

The extension of the 1D algorithm to an arbitrary matrix of size m is straightforward. The columns of matrices $\overline{A}_0$ and U_0 are organized into 2^{d+1} blocks of $m/2^{d+1}$ consecutive columns each Now, an index i identifies a block of columns and the pair (i,j) identifies the transformations which involve the columns of blocks i

and j. In each one of the $2^{d+1}-1$ steps, every node must perform $m/2^{d+1} \times m/2^{d+1}$ transformations corresponding to the pairing of every column of block i with all the columns of block j. In the particular case of the first step, the nodes perform $m/2^{d+1} \times (m/2^{d+1}-1)$ additional transformations corresponding to the pairing of every column with all the columns in its own block. Finally, the exchanges at the end of every step now involve blocks of columns instead of single columns.

The 1D algorithm has two interesting properties related to the efficiency of a parallel implementation. First, the ordering uses a minimum number of steps. Second, it requires that at every step, each node only exchanges one block of matrices $\overline{A}$ and U. This is in contrast to other orderings such as round-robin [2], which require that almost all nodes exchange both blocks of $\overline{A}$ and U at every step.

Consider now the problem of executing the 1D algorithm on the target scenario. The straightforward approach is to make $r=0$ and $c=d$. This configures the mesh as a line, matching the one-dimensional topology assumed by the algorithm. In what follows, we develop a simple analytical model to assess the performance of this approach.

First note that computing a rotation matrix requires $6m$ operations (we count only the operations required for inner products), and applying it requires $12m$ operations. Second, note that all the column exchanges required in the odd steps (figure 1a) and even steps (figure 1b) can be carried out in the wormhole line simultaneously without conflicts in the use of the links. Therefore, the time required for the column exchange at every step is:

$$t_s + \frac{m^2}{2^d} \times t_e$$

As a result, the time required to complete one sweep can be written as:

$$t_{1D} = \frac{9m^3 - 9m^2}{2^d} \times t_c + \left(2^{d+1} - 1\right)\left(t_s + \frac{m^2}{2^d} \times t_e\right) \qquad (1)$$

Expression (1) shows that the communication cost grows as $2^{d+1}-1$. This fact may degrade the performance of the 1D algorithm for large values of d.

Note that alternative configurations of the target scenario (i.e. other values for r and c) would not reduce the communication cost of the one-dimensional algorithm since the number of steps would still be $2^{d+1}-1$.

3 Two-dimensional algorithm

3.1 Motivation

The two-dimensional (2D) algorithm proposed here is aimed at reducing the communication cost incurred by the 1D algorithm on a line. The 2D algorithm uses a $2^r \times 2^c$ mesh of nodes. The computation is organized in such a way that the transformations that were applied independently by a group of consecutive nodes in the line

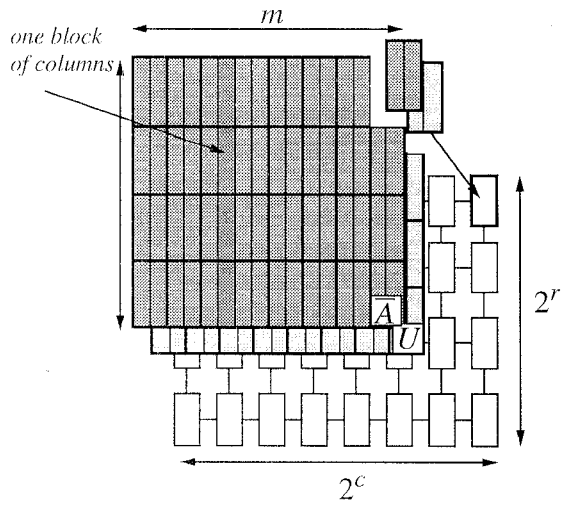


Figure 2: Data distribution in the 2D algorithm

are now applied by a column of nodes in the 2D mesh. Since the number of steps per sweep depends on the number of columns in the mesh (2^c), a 2D algorithm with $c<d$ would have less steps than the 1D algorithm. This could result in potentially lower cost due to column exchanges. From now on, the column exchanges at the end of every step will be referred to as horizontal communication. However, since the nodes in a column must cooperate to compute and apply the transformations, a new type of communication appears. This will be referred to as vertical communication and its cost depends on the number of rows in the mesh (2^r). It will be shown that a proper choice for r and c can reduce significantly the communication cost (horizontal communication plus vertical communication) with respect to the 1D algorithm. In the following, we describe in detail the proposed 2D algorithm.

3.2 Data and computation distribution

The data distribution required by the 2D algorithm is shown in figure 2. The initial matrices $\overline{A}_0$ and U_0 are decomposed into 2^{c+1} blocks of $m/2^{c+1}$ consecutive columns. Every pair of blocks of $\overline{A}_0$ (and the corresponding blocks of U_0) are distributed among the 2^r nodes of a mesh column. As a result, every node stores $m/2^c$ pieces of columns of $\overline{A}_0$ and $m/2^c$ pieces of columns of U_0. Each of these pieces of columns has $m/2^r$ elements.

In the 2D algorithm, each sweep consists of $2^{c+1}-1$ steps. At each step, the 2^r nodes of a column cooperate to compute and apply a total of $m/2^{c+1} \times m/2^{c+1}$ transformations. This corresponds to pairing every column of one block with all the columns in the other block. In the first step, $m/2^{c+1} \times (m/2^{c+1}-1)$ additional transformations must be computed and applied, corresponding to pairing every column with all the remaining columns in its own block.

The computation and the application of every

transformation is now distributed among the nodes of the column, since every node stores only a piece of $m/2^r$ elements of the two columns from $\bar{A}$ and U involved in the transformation. Let us focus on transformation $R(i,j)$. The activity of the nodes in the column of the mesh responsible for computing and applying $R(i,j)$ can be organized into three phases as follows:

Phase 1. Every node in the column performs three partial inner products corresponding to the recovering of elements $A(i,j)$, $A(i,i)$ and $A(j,j)$. To that end, every node uses the pieces of columns i and j of matrices $\bar{A}$ and U. The result is a set of three values in every node of the column.

Phase 2. The nodes in the column cooperate to accumulate the partial inner products and obtain the final values of $A(i,j)$, $A(i,i)$ and $A(j,j)$. At the end of this phase, a copy of these three final values must be stored in every node in the column. It is in phase 2 where the vertical communication appears.

Phase 3. Every node computes $s=\sin(\alpha_{i,j})$ and $c=\cos(\alpha_{i,j})$ and applies the transformation to the pieces of columns i and j.

At each step, every column of nodes must repeat phases 1, 2 and 3 for every one of the assigned transformations. However, proceeding in this way would make the cost of the vertical communication very high since the nodes would exchange a large amount of very short messages (only three elements per message). In order to amortize the communication startup cost, the transformations assigned to each mesh column at every step are organized into groups of independent transformations. Now, for each group of independent transformations, phase 1 is performed for every transformation within the group. Then, vertical communication is carried out to exchange the partial inner products corresponding to all the transformations in the group. Finally, phase 3 can be carried out for all the transformations in the group. In this way, the number of messages involved in the vertical communication is reduced since every message now contains a larger number of partial inner products.

For a given step, the problem of organizing the computation assigned to a column of the mesh into groups of independent transformations can be solved in many different ways. In fact, the approaches to obtain parallel Jacobi orderings can be readily applied to solve this problem since the objective is exactly the same, that is, to find groups of independent transformations. In general, in the first step of every sweep, every column of the mesh organizes the computation into $m/2^c-1$ groups of $m/2^{c+1}$ independent transformations each. In every one of the remaining steps of a sweep, each column of the mesh works with $m/2^{c+1}$groups, with $m/2^{c+1}$ independent transformations in each group.

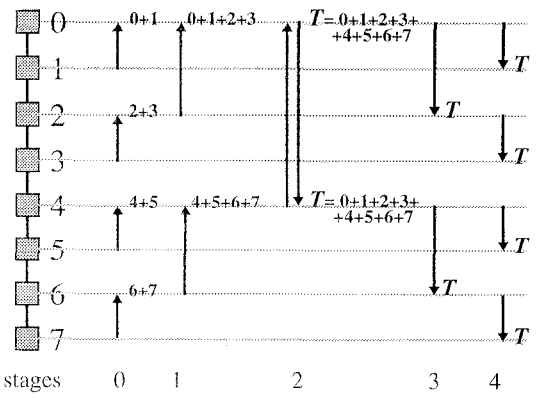


Figure 3: An example (with r=3) of the algorithm to implement the vertical communication arising when applying the transformations in a group.

3.3 Vertical and horizontal communication

We describe first the proposed scheme to perform the vertical communication which arises when performing a group of independent transformations. The communication problem consists of a multiple vector addition. Every one of the 2^r nodes in the column stores a vector of $3m/2^{c+1}$ values, corresponding to the partial inner products of the $m/2^{c+1}$ independent transformations of the group. All these vectors must be accumulated and, at the end, every node must have a copy of the resulting vector. Figure 3 shows an example of the scheme to solve this problem assuming a wormhole line for the case of r=3. In this figure, the numbers identify the nodes in the column and also the vectors to be accumulated. Arrows represent communication between nodes. T denotes the final vector. In the following, we describe the communication algorithm in detail. We denote by $<n_{r-1},...,n_1,n_0>$ the binary form of integer n ($n \in [0,2^r-1]$).

The algorithm proceeds in $2r-1$ stages, numbered from 0 to $2r-2$. In stages i, with $i \in [0,r-2]$, only nodes with label n such that $n_{i-1}=...=n_0 = 0$ are active. In particular, a node with label $<n_{r-1},...,n_{i+1},1,0,...,0>$ sends its partial vector to the node with label $<n_{r-1},...,n_{i+1},0,0,...,0>$, which, after receiving the messages, accumulates the received vector within its own partial vector. In stage $r-1$, nodes 2^{r-1} and 0 exchange a copy of their partial vectors and accumulate the received vector. At this time, both nodes, 2^{r-1} and 0, have a copy of the final vector T. Finally, in stages $2r-2-i$, with $i \in [0,r-2]$, the node with label $<n_{r-1},...,n_{i+1},0,0,...,0>$ sends a copy of the final vector to the node with label $<n_{r-1},...,n_{i+1},1,0,...,0>$

We now focus on the horizontal communication that is required at the end of every one of the $2^{c+1}-1$ steps of a sweep. In this communication operation, the columns of matrices $\bar{A}$ and U are exchanged so that every node receives the pieces of the columns required to perform the transformations in the next step. Every process

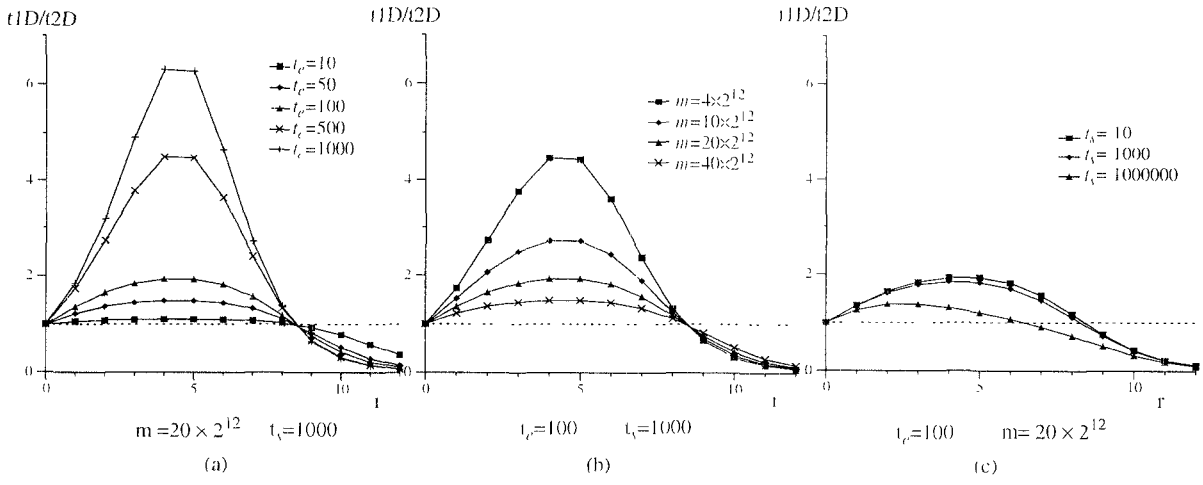


Figure 4: Impact of parameters t_e, m and t_s in the performance improvement of the proposed 2D algorithm

participates in a communication in the horizontal dimension of the mesh, according to the same exchange patterns that are used by the 1D algorithm (and shown in figures 1a and 1b). In particular, nodes exchange messages of $m^2/2^d$ elements, corresponding to the pieces of columns in one block of $\overline{A}$ and the corresponding block of U.

4 Analytical modelling and evaluation of the 2D algorithm

4.1 An analytical model

It is obvious that increasing the value of r (and as a result decreasing c) reduces the number of steps of the 2D algorithm and therefore, the cost of the horizontal communication, at the expense of an increase in the cost of vertical communication. The analytical model of execution time that will be developed in this section, will enable us to derive the optimal configuration of the mesh. This would correspond to the values of r and c which minimizes the execution time.

We consider first the cost of the partial inner computation and the application of the transformation (phases 1 and 3, described in section 3.2).This cost is:

$$t_R = (m-1) \times \frac{m}{2^{c+1}} \times \frac{18m}{2^r} \times t_c = \frac{9m^3 - 9m^2}{2^d} \times t_c$$

The term m–1 corresponds to the number of groups of independent transformations in a sweep, the term $m/2^{c+1}$ is the number of independent transformations per group, and the term $18m/2^r$ is the number of operations to compute the partial inner product (phase 1) and apply (phase 3) the transformation.

The cost of the vertical communication (including the cost of partial vector accumulation) is given by the following expression:

$$t_V = (m-1)(2^r - 1)\left(t_s + \frac{3m}{2^{c+1}} \times t_e\right) + (m-1)(r-1)\left(\frac{3m}{2^{c+1}} \times t_e\right)$$

This expression is obtained by considering again the number of groups of independent transformation in a sweep (m–1) and the cost of the vertical communication per group, according to the algorithm proposed in section 3.3.

The cost of the horizontal communication is written as follows:

$$t_H = \left(2^{c+1} - 1\right)\left(t_s + \frac{m^2}{2^d} \times t_e\right)$$

Finally, the total cost of the algorithm is $t_{2D} = t_R + t_V + t_H$. Although it is not possible to write a simple analytical expression for the optimal values of c and r, these values can be computed by inspection, when fixing the values for the rest of the parameters of the analytical model.

4.2 Performance evaluation

In this section, we plot some figures to show the improvement in performance that can be obtained with the 2D algorithm, with respect to the 1D algorithm. These figures have been obtained from the analytical models of t_{1D} and t_{2D} derived in the previous sections.

The figures are presented in two types of plots. Plots in figure 4 assume a fixed value of d (specifically d=12) and show the ratio t_{1D}/t_{2D} for all possible values of r ($r \in [0,d]$) and varying values of parameters t_e(figure 4a), m (figure 4b), and t_s (figure 4c)(t_c is fixed at 1). In every plot, one of these three parameter takes different values while the remaining two are fixed, showing in this way the impact of the varying parameter on the performance

improvement and on the optimal configuration of the mesh.

The following conclusions can be drawn from the plots in figure 4:

- Parameter t_e (transmission time per floating point number) has a strong impact on performance improvement (see figure 4a). This improvement can be very high for large values of t_e. On the other hand, the value of t_e does not have an impact in the optimal configuration.
- The performance improvement is also very sensitive to the values of m, but in this case, large values of m reduce this improvement (see figure 4b). Varying m does not changes the optimal configuration.
- Large values of t_s reduce the performance improvement and favour configurations with smaller number of rows (see figure 4c). However, these effects are only noticeable for extremely large values of t_s. Therefore, the performance improvement is not very sensitive to the value of t_s.
- The optimal configuration usually has less rows than columns, since the vertical communication cost grows very fast with the number of rows (note that in the expression of t_V, the term t_c is affected by m).

The plots in figure 5 show the scalability properties of the proposed algorithm. These plots show the ratio t_{1D}/t_{2D} for fixed values of t_s and t_c, but varying values of d and m (note that m is now scaled with the number of nodes). In this case, for a given value of d, the optimal configuration in considered for t_{2D}. The plots show that the performance improvement obtained with the 2D algorithm grows when increasing the number of nodes in the system. This growth is particularly high for small problem sizes.

5 Summary

The figures in section 4 show that the proposed 2D algorithm for symmetric eigenvalue and eigenvector computation can provide a significant performance improvement with respect to the 1D algorithm, for a wide range of system and problem parameters. This improvement increases with the system size, showing that the algorithm is scalable, which is an important property when looking at large scale parallel systems.

Further work is currently focusing on assuming a mesh with a fixed configuration, and in particular, a squared mesh (a $2^{d/2}\times2^{d/2}$ mesh, with d even)[8]. In this case, an obvious solution is to use the algorithm proposed in this paper, with $r=c=d/2$. However, this may not be the optimal choice. A more general approach would be to embed a $2^r\times2^c$ mesh onto the squared mesh and analyse the impact of such an embedding on the cost of vertical and horizontal communication, in order to select the optimal values for r and c.

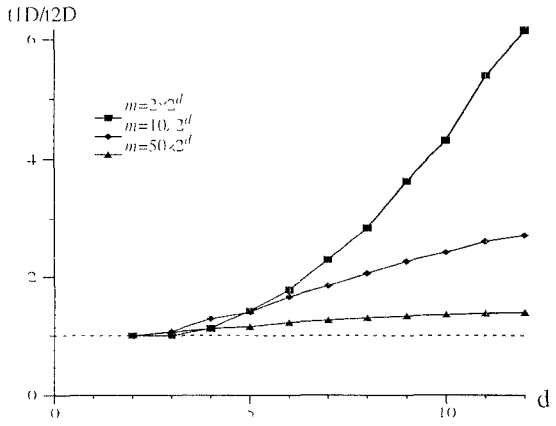


Figure 5: Performance improvement of the proposed 2D algorithm for an increasing number of nodes.

6 References

[1] M. Berry and A. Sameh, "Multiprocessor Jacobi Schemes for Dense Symmetric Eigenvalue and Singular Value Decompositions," *Proceedings of ICPP 98* (1986), pp. 433-440.

[2] R.P Brent and F.T. Luk, "The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays," *SIAM J. Sci. Statist. Comput.* 6 (1985), pp. 69-84.

[3] P.J. Eberlein, "On one-sided Jacobi methods for parallel computation," *SIAM J. Algebraic Discrete Methods* 8 (1987), pp. 790-796

[4] P.J. Eberlein and H Park, "Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures," *Journal of Parallel Distributed Computing* 8 (1990), pp. 358-366.

[5] G H. Golub and C.F. Van Loan, *Matrix computations*, Baltimore, MD: The John Hopkins Univ. Press, 1989.

[6] F.T. Luk, "Architectures for computing eigenvalues and SVDs," *SPIE Vol. 614 Highly Parallel Signal Processing Architectures* (1986), pp. 24-33.

[7] L.M. Ni and P.K. McKinley, "A survey of wormhole routing techniques in directed networks," *IEEE Computer*, Vol. 26, No 2, February 1993, pp. 62-76.

[8] D. Royo, M Valero-García and A. González, "A Jacobi-like Algorithm for Eigenvalue Computation on a 2D/3D mesh multicomputer". *Technical Report UPC-DAC-1997-73, Universitat Politècnica de Catalunya*

[9] R.A Whiteside, N S Ostlund, and P.G. Hibbard, "A parallel Jacobi diagonalization algorithm for a loop multiple processor system," *IEEE Trans. Comput.* C-33 (1984), pp 409-413

[10] J H. Wilkinson, *The Algebraic Eigenvalue Problem*, Claredon Press, Oxford, 1965.

[11] B.B. Zhou and R.P. Brent, "A Parallel Ring Ordering Algorithm for Efficient One-Sided Jacobi SVD Computations," *Journal of Parallel and Distributed Computing* (1997), 42, 1-10.