

Jacobi Orderings for Multi-Port Hypercubes

Dolors Royo, Antonio González and Miguel Valero-García

Universitat Politècnica de Catalunya

Department of Computer Architecture

Jordi Girona 1-3, Mòdul D6

08034 Barcelona (Spain)

Email: {dolors, antonio, miguel}@ac.upc.es

Abstract¹

The communication cost plays a key role in the performance of many parallel algorithms. In the particular case of the one-sided Jacobi method for symmetric eigenvalue and eigenvector computation the communication cost of previously proposed algorithms is mainly determined by the particular ordering being used. In this paper we proposed two novel Jacobi orderings: the permuted-BR ordering and the degree-4 ordering, aimed at efficiently exploiting the multi-port capability of a hypercube. It is shown that the former is nearly optimal for some scenarios and the latter outperforms previously known orderings by a factor of two.

1. Introduction

The one-sided Jacobi method for symmetric eigenvalue computation is very suited for its application on a multicomputer since it exhibits a high parallelism and potentially low communication requirements [5]. The one-sided method uses a series of similarity transformations to make the original symmetric matrix converge to a diagonal form. Every similarity transformation zeroes one off-diagonal element and its symmetric. The elements of the matrix can be zeroed in any order, giving place to different Jacobi orderings and, as a result, to different one-sided Jacobi algorithms. In addition to the convergence rate, different Jacobi orderings differ in the communication requirements of the resulting parallel algorithm.

In multicomputers, the communication overhead plays an important role on the performance of any particular algorithm [1]. In this paper, we focus on multicomputers with a hypercube interconnection topology and with multiple ports per node [14]. In such scenario one may design algorithms that communicate multiple messages simultaneously through different links of the same node (communication parallelism), which may result in a significant reduction in the communication overhead. One-sided Jacobi algorithms previously proposed for hypercubes

make a poor utilization of the multi-port capability because, at any given time, the information to be communicated by every node is sent through one (or at most two) link, constraining in this way the exploitation of communication parallelism.

In [9] a method was developed to design parallel algorithms that efficiently exploit the multi-port capability in hypercubes. The method requires the specification of the original problem in the form of a CC-cube algorithm, whose properties will be described later. The method reorganizes the computation in a systematic way to introduce the appropriate level of communication parallelism in order to efficiently exploit the multi-port capability.

It will be shown in this paper that one-sided Jacobi computation can take the form of a CC-cube algorithm. Thus, the method in [9] can be used to reduce the communication cost in multi-port hypercubes. We will show first that the impact of the method when applied to known Jacobi algorithms is limited by the structure of the Jacobi ordering used in these algorithms. The key contribution of this paper is the proposal of two novel Jacobi orderings, the *permuted-BR* and the *degree-4* orderings, which enable an efficient exploitation of the multi-port capability, through the use of the method described in [9]. The permuted-BR ordering has a performance that tends asymptotically (for large matrices) to 80% of a lower bound. The degree-4 ordering has a worse asymptotic performance but it behaves better for small matrices. In this case, it reduces the communication overhead of the algorithm to the half when compared with previous Jacobi orderings.

The rest of this paper is organized as follows. Section 2 provides a brief review of the one-sided Jacobi method and the orderings proposed for its implementation on a hypercube multicomputer. Section 3 presents the novel orderings. Section 4 evaluates the performance of the proposed orderings. Finally, the main conclusions are summarized in section 5.

2. Preliminaries

In this paper we focus on efficient algorithms for eigenvalues and eigenvectors computation of symmetric matrices through the one-sided Jacobi method on multicomputers with a hypercube interconnection topology and

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

multiple ports per node. Below we summarize what it is implied by each one of these terms. Then, we review the most relevant related work and point out the motivation for this work.

2.1. Target architecture

The target architecture on which the proposed algorithm is to be executed is a multi-port hypercube multicomputer. A hypercube multicomputer of dimension d , which is also called a d -cube multicomputer, consists of 2^d processors such that they can be labelled from 0 to 2^d-1 in such a way that all processors that differ in exactly one bit in the binary representation of their label are neighbors (they are connected by a link) [10]. The link that connects neighbor nodes whose labels differ in the i -th bit will be referred to as link i . The label of the link, which ranges from 0 to $d-1$, will be also called the dimension of the link. As an example, node 2 uses link 1 (or dimension 1) to send messages to node 0.

A multi-port multicomputer is distinguished by the fact that every node can simultaneously be transmitting/receiving messages from more than one link. In particular, in an *all-port* configuration every node can send and receive a message through each of its d links simultaneously. A configuration in which every node can send and receive a message from just one of its links is called *one-port* configuration [14].

2.2. The one-sided Jacobi method and parallel Jacobi orderings

The one-sided method for symmetric eigenvalue and eigenvector computation works with two matrices: \bar{A} (initially set to the original matrix A) and U (initially set to the identity matrix I). In every iteration of the method one similarity transformation is applied to zero one off-diagonal element of A and its symmetric one [15]. The key feature of the one-sided method is that the computation and application of the similarity transformation that zeroes elements (i,j) and (j,i) of A uses only columns i and j of \bar{A} and U . For this reason, such a transformation will be also referred to as the pairing of columns i and j . According to this, transformations that use disjoint pairs of columns can be applied in parallel, since they do not share any data. This is what makes the one-sided Jacobi method to be very suitable for parallel implementations [5].

In the literature, the process of zeroing every off-diagonal element exactly once is called a sweep. Since a similarity transformation may fill in elements that had been zeroed in previous transformations, several sweeps are required until the resulting matrix converges to a diagonal matrix. Since similarity transformations preserve the eigenvalues, the elements in the diagonal of the resulting matrix coincide with the eigenvalues of the original matrix.

If A is an $m \times m$ matrix, a total of $m(m-1)/2$ similarity transformations are required to complete a sweep. Assuming m even, these transformations can be organized into a maximum of $m-1$ groups of $m/2$ independent transformations each (transformations that involve disjoint pairs of columns). Such an organization of the sweep is called a parallel Jacobi ordering and every group of independent transformations is usually referred to as a step.

Parallel Jacobi orderings offer a good framework for the design of parallel implementations, since the computation involved in every step can be evenly distributed among the nodes of the multicomputer, and carried out without any communication. Communication is required however at the end of every step, when the columns of matrices \bar{A} and U must be exchanged to obtain the pairs required for the next step. Such a communication will be called a transition. A parallel Jacobi ordering is suitable for a multicomputer if the transitions between steps use a communication pattern that matches the topology of the interconnection network of the machine.

2.3. Related work

Jacobi orderings for different parallel architectures can be found in the literature [4,6,8,13]. Regarding hypercube multicomputers, which is the target architecture considered in this work, the most relevant proposals are [3,7,12]. All these proposals share the following features: (a) they use a minimum number of steps per sweep, achieving in this way a perfect load balance, and (b) the transitions between steps can be implemented through communication between neighbors in the hypercube. Particularly relevant to our work is the Block Recursive (BR) ordering proposed in [12]. The distinguishing feature of this ordering is that in every transition all nodes exchange information through the same dimension of the hypercube. This is one of the requirements for an algorithm to belong to the CC-cube class, and therefore, to enable the application of the communication pipelining technique. Below we describe the BR ordering in detail since some of the orderings proposed in this paper are obtained through the application of some transformations to it.

2.3.1. BR ordering

The BR ordering was originally proposed in [7] and reviewed later in [12], where the ordering was completely specified and its correctness was shown.

Given a d -cube and a $m \times m$ matrix, the m columns of matrices \bar{A} and U are grouped into 2^{d+1} blocks of $m/2^{d+1}$ columns each¹. Then, a pair of blocks are allocated to each node of the d -cube. The algorithm to perform the first sweep proceeds as follows (all the nodes are executing the following code in parallel):

1. If m is not a power of 2, then the number of columns per block will differ in one unit at most. This will result in a slight load imbalance.

- 1) Pair each column of a block with the remaining columns of the same block.
- 2) Pair each column of a block with all the columns of the other block allocated to the same processor.
- 3) Exchange one of the blocks with a neighbor along a given dimension of the hypercube (*transition*).
- 4) Go to (2) if not all the columns have been paired (i.e., (2) and (3) are repeated $2^{d+1}-1$ times).

In order to achieve that every pair of blocks is paired¹ exactly once, the transitions (i.e. the exchange operations required by (3)) must be performed in a particular order. To precisely define an exchange operation it suffices to identify the link (i.e., the dimension) that is used by it. Below we describe the sequence of transitions implied by the BR ordering as described in [12].

The $2^{d+1}-1$ steps of the ordering are grouped into three different types of *phases*: *exchange*, *division* and *last transition*. A phase is just one or more consecutive steps, each of them followed by a transition. A sweep consists of d exchange phases, each one of them followed by a division phase, and a final last transition phase. The exchange phases are numbered from d to 1. The exchange phase e ($e \in [d,1]$) consists of 2^e-1 steps and transitions. The sequence of links (i.e. dimensions) that define the transitions is denoted by D_e^{BR} and can be systematically generated as follows:

$$D_1^{BR} = \langle 0 \rangle$$

$$D_i^{BR} = \langle D_{i-1}^{BR}, i-1, D_{i-1}^{BR} \rangle \quad 1 < i \leq e$$

For instance, the sequence of links for $e=4$ is $D_4^{BR} = \langle 010201030102010 \rangle$.

The exchange phase e is followed by a division phase that consists of just one step and one transition through link e . Finally, the last transition phase consists of one step and one transition through link $d-1$.

The second and next sweeps use the same algorithm but after applying a permutation to the links. In particular, the permutation corresponding to sweep s (assuming $s=0$ for the first one) is defined as follows:

$$\sigma_0(i) = i$$

$$\sigma_s(i) = (\sigma_{s-1}(i) - 1) \bmod d \quad \text{for } i=0\dots d-1$$

After d sweeps, the links are used again in the order described for the first sweep.

2.4. Motivation

The previously summarized BR algorithm is efficient for a one-port hypercube since the system is using all the available ports for every transition. However, for a multi-port architecture it achieves the same performance as for a one-port because it uses just one out of the d links of each node at the same time.

In [9] we proposed a systematic transformation of a class of algorithms that is called CC-cube algorithms in order to optimize them for a multi-port environment. Such transformation is called *communication pipelining*. Every process of the CC-cube algorithm executes a loop that iterates K times. Every iteration consists of a certain computation and an exchange of information through one of the hypercube dimensions (all the processes use the same dimension in a given iteration of the loop). The key idea of communication pipelining is to decompose the computation in every iteration into Q packets and reorganize the computation as follows. Every process of the new algorithm (the pipelined CC-cube algorithm) compute the first packet of the computation in the first iteration of the original CC-cube and exchanges the result with the corresponding neighbor. Then, it computes the second packet of the first iteration and the first packet of the second one. The results of these packets can be sent in parallel to the corresponding neighbors, using simultaneously two hypercube dimensions. Proceeding in this way, the pipelined CC-cube can send an increasing number of messages in parallel, being able to exploit the multi-port feature.

The value of Q is called *pipelining degree* and in [9] it is shown how to determine the pipelining degree that minimizes the execution time of any particular CC-cube algorithm and any particular hypercube architecture. In this section we only summarize the result of the application of the communication pipelining technique and omit any detail that is not strictly necessary to understand this paper. The interested reader is referred to the original paper for such details.

If Q is not higher than K , the pipelined CC-cube algorithm is said to work in shallow pipelining mode. Every process can send up to Q messages simultaneously through different links if the architecture supports it. These Q links correspond to all the subsequences of Q elements of the sequence that describes the link usage in the CC-cube algorithm.

For instance, if in the original CC-cube $K=7$ and communications are carried out through links 0, 1, 0, 2, 0, 1 and 0, the pipelined CC-cube with $Q=3$ will perform a series of computations each one followed by communication through links 0-1-0², 1-0-2, 0-2-0 and 2-0-1 respectively. This part of the pipelined CC-cube is called the kernel and has $K-Q$ stages (computation followed by communication).

Like in the software pipelining technique [11], the kernel is preceded by a prologue and followed by an epilogue. The prologue consists of $Q-1$ stages and each of these stages consists of a computation followed by a communication though an increasing number of the first links of the original sequence. In the previous example, the prologue

1. The pairing of two blocks is defined as all the pairings that consists of two columns from different blocks.

2. $a-b-c$ means that three messages are sent in parallel through links a , b , and c . If several messages use the same link, they are packed into a single message and sent together.

consists of two stages that use the following links: 0 and 0-1 respectively. Similarly, the epilogue consists of $Q-1$ stages and each of these stages consists of a computation followed by a communication through a decreasing number of the last links of the original sequence. In the previous example, the epilogue consists of two stages that use the following links: 1-0 and 0.

If Q is higher than K , the pipelined CC-cube is said to work in deep pipelining mode. In this case, the prologue and epilogue have $K-1$ stages each and the kernel has $Q-K+1$ stages. In each kernel stage, K messages can be sent in parallel (one packet from every iteration of the original CC-cube). For instance, if $K=3$ and the links are used in the order 0, 1, and 0, the pipelined CC-cube with $Q=100$ has a prologue with two stages that use links 0 and 0-1. Every one of the 98 stages of the kernel use links 0-1-0. Finally, the two stages of the epilogue use the links 1-0 and 0.

Communication pipelining can be applied to any CC-cube algorithm that meets some requirements as specified in [9]. In particular, it cannot be applied to the whole BR algorithm but it can be applied to every exchange phase, which are the most time-consuming part of the BR algorithm. Communication pipelining can reduce the communication overhead of the algorithm by a factor of d , which produces very important improvements in some problems where the communication operations have an important weight. Unfortunately, the application of communication pipelining to the BR algorithm can reduce the communication overhead by a factor not higher than 2 regardless of the value of d . This is basically due to the properties of the BR ordering: in the sequence D_e^{BR} , which defines the order in which the links of each node are used in exchange phase e , any subsequence of Q consecutive elements has at least $\lfloor Q/2 \rfloor$ elements equal to 0. In consequence, the capability to send simultaneously messages through multiple links can provide a reduction in the communication cost by a factor not higher than 2, since about half of the messages must be sent through the same link (link 0).

In this paper we propose novel Jacobi orderings that exhibit a more balanced use of the links of each node and thus, communication pipelining can provide a much higher improvement than for the BR ordering.

3. New Jacobi orderings

All the new orderings proposed in this section are obtained from the BR ordering by replacing the sequence D_e^{BR} ($e \in [d, 1]$) by alternative sequences that, beside generating all the required pairings of columns, exhibit a more balanced use of the links of every node, enabling a more efficient exploitation of the multi-port capability.

The first proposal is optimal but it is only defined for small hypercubes. The second one has a good asymptotic performance for large problem sizes. Finally, third one

behaves better for small problem sizes. This section also includes a discussion on the convergence of the new orderings.

3.1. Minimum- α ordering

Let us denote by D_e the sequence of links used in exchange phase e for the first sweep (e.g. D_e^{BR} for the BR ordering). Communication pipelining allows each node to send messages through all or several consecutive links in D_e , depending on the degree of pipelining (value of Q). In the case of deep pipelining, all the links of the sequence D_e are used in each communication operation (except for the prologue and epilogue). A message of a fixed size S must be sent per each element of D_e through the corresponding link. If there are repeated elements in D_e , then the corresponding messages are combined into a single message. Since all the elements in $[0, e-1]$ must appear at least once in D_e (otherwise, it would be impossible to generate all the pairings required by exchange phase e) the time to perform the communication operation in every kernel stage, in an all-port hypercube is $eT_s + \alpha ST_w$, where T_s is the start-up time to initiate the communication through one link, S is the fixed message size, T_w is the transmission time per data element, and α is the maximum number of messages that must be combined into a single message and sent through the same link, that is, α is the maximum number of repetitions of the same link in the sequence D_e .

Since T_s and T_w are fixed for a given architecture and S and e are fixed for a particular problem, the cost of each communication operation just depends on α , which is a function of the sequence D_e .

In the BR ordering, the value of α corresponding to D_e^{BR} is 2^{e-1} . The communication overhead can be minimized by finding an alternative sequence D_e whose α is minimum.

The sequence D_e can be also regarded as the definition of a hamiltonian path in an e -cube. This is so because the half of the columns of \bar{A} and U initially allocated to each node must visit a different node after each exchange operation and thus, they must have visited all the nodes after the 2^e-1 exchange operations that make up the exchange phase e . For instance, the exchange phase $e=3$ of the BR ordering is defined by $D_3^e = \langle 0102010 \rangle$. Starting at any node of the hypercube and moving through the links that form this sequence in the order they appear, all the nodes of a 3-cube are visited exactly once.

We can then conclude that the problem of defining alternative sequences D_e can also be stated as the problem of finding alternative hamiltonian paths in an e -cube. The resulting sequence consists of the link identifiers that have been used for traversing the hypercube. Since there are many different hamiltonian paths in a hypercube, there are many alternative Jacobi orderings that can be generated in this way.

Minimizing the communication cost of exchange phase e when communication pipelining is used can be achieved by computing all the hamiltonian paths of an e -cube and choosing that with the minimum α . Since any sequence D_e has $2^e - 1$ elements and all values in $[0, e-1]$ must appear in D_e at least once, the minimum value for α is given by:

$$\alpha \geq \left\lceil \frac{2^e - 1}{e} \right\rceil$$

Unfortunately, finding a hamiltonian path with minimum α is an NP-hard problem and thus, it can be solved only for small values of e . In particular, following this approach we could compute the sequences with minimum α for values of e lower than 7. These sequences are called $D_e^{\min-\alpha}$ and are the following:

$$D_2^{\min-\alpha} = \langle 010 \rangle, \alpha = 2$$

$$D_3^{\min-\alpha} = \langle 0102101 \rangle, \alpha = 3$$

$$D_4^{\min-\alpha} = \langle 010203212303121 \rangle, \alpha = 4$$

$$D_5^{\min-\alpha} = \langle 0102010301021412321230323414323 \rangle, \alpha = 7$$

$$D_6^{\min-\alpha} = \langle 010201030102010401021312521312432313234350542453542414345254345 \rangle, \alpha = 11$$

The Jacobi ordering that uses the sequences $D_e^{\min-\alpha}$ is called the minimum- α ordering. However, it is only defined for hypercubes with $d < 7$.

3.2. Permuted-BR ordering

The Jacobi ordering proposed in this section, which is called the permuted-BR ordering, uses the sequence denoted by D_e^{p-BR} . This sequence is defined for any value of e and it will be shown that the value of α for the sequence is close to the minimum. As reflected in the name of the ordering, the sequence D_e^{p-BR} is obtained by applying some permutations to the link identifiers in D_e^{BR} . This transformation is based on the following property.

Property 1. Let D_e be a sequence of links that define a hamiltonian path in an e -cube. Let σ be any permutation of the link identifiers. If we apply σ to any subsequence of D_e that corresponds to a hamiltonian path of a subcube, then the resulting sequence defines also a hamiltonian path of the e -cube.

Proof. Given a n -cube, if we apply any permutation to all its links and re-label the nodes accordingly, the resulting graph is isomorphically equivalent to the initial one. Thus, if a link permutation is applied to a sequence of link identifiers that define a hamiltonian path in a n -cube, the resulting sequence is also a hamiltonian path in the same n -cube. \square

Examples

- $\langle 010 \rangle$ is a hamiltonian path in a 2-cube. If links 0 and 1 are exchanged in the whole sequence, the resulting sequence $\langle 101 \rangle$ is also a hamiltonian path.

- $\langle 0102010 \rangle$ is a hamiltonian path in a 3-cube. If we apply the previous permutation to the subsequence made up of the last 3 elements, which is a hamiltonian path in a 2-cube, we obtain the sequence $\langle 0102101 \rangle$, which is also a hamiltonian path in a 3-cube.

Definition 1. A sequence that corresponds to a hamiltonian path of a d -cube will be called a d -sequence, or a d -subsequence if it appears within a larger sequence.

Obviously, D_e^{BR} is an e -sequence. Otherwise, it could not be used to implement exchange phase e (see [12] for a proof of the correctness of the BR ordering). Due to the way D_e^{BR} is built, (see section 2.3.1), it is an e -sequence that consists of two $(e-1)$ -subsequences separated by the link $e-1$. Each one of these subsequences is in turn composed of two $(e-2)$ -subsequences separated by the link $e-2$. In general, each $(e-i)$ -subsequence is composed of two $(e-i-1)$ -subsequences separated by the link $e-i-2$. This is a useful property of the D_e^{BR} sequence that will be used in the following.

3.2.1. Methodology to generate the D_e^{p-BR} sequence

The D_e^{p-BR} sequence is obtained by applying some transformations to the D_e^{BR} sequence. Such transformations meet property 1 stated above and in consequence, the resulting sequences are guaranteed to be valid in the sense that they also correspond to a hamiltonian path of an e -cube. The objective of this transformations is to obtain a sequence in which the number of repetitions of each link identifier is about the same in order to minimize the value of α .

$\lfloor \log_2(e-1) \rfloor$ transformations are applied to D_e^{BR} in order to obtain D_e^{p-BR} . Each transformation consists of a link permutation that is applied to some subsequences of the whole sequence. In particular, transformation k , k being an integer from 0 to $\lfloor \log_2(e-1) \rfloor - 1$, consists of a link permutation applied to every other $(e-k-1)$ -subsequence, starting at the second one. That is, the first transformation is applied to the second $(e-1)$ -subsequence. The second transformation is applied to the second and forth $(e-2)$ -subsequences and so on. Since property 1 holds, the resulting sequence is also a valid sequence to implement the exchange phase e .

For each transformation k , the link permutation applied to the second $(e-k-1)$ -subsequence is defined as the transposition of the following pairs of link identifiers:

$$i \leftrightarrow \lfloor (e-1)/2^k \rfloor - 1 - i$$

$$i \in [0, \lfloor (e-1)/2^k \rfloor - 1] \quad k \in [0, \lfloor \log_2(e-1) \rfloor - 1]$$

The link permutation applied to any of the remaining subsequences (just to every other) is derived by compound-ing the permutation applied to the second $(e-k-1)$ -subsequence with all the permutations applied in previous transformations to any subsequence that includes the current one.

e	α	lower bound	$\alpha/\text{lower-bound}$
7	23	19	1.21
8	43	32	1.34
9	67	58	1.16
10	131	103	1.28
11	289	187	1.55
12	577	342	1.69
13	776	631	1.23
14	1543	1171	1.32

Table 1. Value of α corresponding to the permuted BR ordering and compared to a lower bound.

Example

Let us show how D_5^{p-BR} is obtained (we take $e=5$ to shorten the example; note however that an optimal sequence $D_5^{\min-\alpha}$ is known). We start from the sequence D_5^{BR} , which is equal to:

$$D_5^{BR} = \langle 0102010301020104010201030102010 \rangle$$

Two transformations ($\lfloor \log_2 4 \rfloor$) are required. The first one is applied to the second 3-subsequence and it consists of exchanging each element i ($i \in [0,3]$) with element $3-i$ respectively. The resulting sequence is (the elements affected by the permutation are shown in boldface):

$$\langle 0102010301020104\mathbf{323132303231323} \rangle$$

The second transformation permutes the links in the second and forth 2-subsequences. The permutation for the second 2-subsequence is defined as the transposition of elements 0 and 1. For the forth 2-subsequence, the corresponding permutation is obtained by compounding the one applied to the second 2-subsequence with the permutation defined in the first transformation. That is, since in the first transformation 0 was exchanged with 3 and 1 was exchanged with 2, the resulting compounded permutation consists of transposing element 3 and 2. The resulting sequence is:

$$D_5^{p-BR} = \langle 01020103\mathbf{1012101}432313230\mathbf{2321232} \rangle$$

3.2.2. Optimality of D_e^{p-BR}

The appendix of this paper contains the proof that the value of α corresponding to the D_e^{p-BR} sequence tends to be 1.25 times the lower bound given by $\lceil (2^e - 1)/e \rceil$ (see section 3.1), for large values of e . Therefore, D_e^{p-BR} is close to the optimal for deep pipelining when the degree of pipelining is large enough to neglect the prologue and epilogue phases. Table 1 shows the value of α for sequences D_e^{p-BR} with $e \in [7,14]$ and compares these values with the lower bound.

3.3. Degree-4 ordering

The permuted-BR ordering proposed in the previous section is expected to work well when the pipelined CC-cube works in deep pipelining mode and the cost is dominated by the kernel phase. In other words, when Q is very large. In the case that the matrix size is not large enough to enable large values of Q the pipelined CC-cube will work in shallow pipelining mode. In this mode of operation the quality of sequence D_e is no longer related to the value of α since only a subsequence of the links in D_e are used in every communication operation. In this context, the sequence D_e^{p-BR} proposed before is not adequate since when considering small subsequences of links, nearly half of the elements are equal (see for instance the D_5^{p-BR} sequence at the end of section 3.2.1).

What would be desirable for shallow pipelining is a sequence such that any subsequence of length Q consist of different elements if Q is not higher than e , or it consists of all the elements from 0 to $e-1$ repeated the same number of times, with a maximum difference of one if Q is not multiple of e . Finding such optimal sequences is still an open problem with unknown solution. The Jacobi ordering proposed in this section, called *degree-4* ordering, uses the sequence D_e^{D4} whose main feature is that most subsequences of length 4 consist of different elements and thus, it outperforms previous orderings for shallow pipelining.

Definition 2. We say that a sequence D_e has degree n if the majority of subsequences of size n consists of different elements but the majority of subsequences of size $n+1$ have less than $n+1$ different elements. For instance, D_e^{BR} has degree 2 for any e .

Shallow pipelining can reduce the communication cost of the Jacobi method by a factor near to n if the ordering being used is defined by sequences D_e of degree n . The new ordering proposed in this section is called degree-4 ordering since it uses sequences of degree 4, enabling a reduction of the communication cost by a factor of about 4.

Definition 3. The sequence D_e^{D4} can be systematically generated for any value of e as follows:

$$\begin{aligned} E_3 &= \langle 0123012 \rangle \\ E_i &= \langle E_{i-1}, i, E_{i-1} \rangle & 4 \leq i < e \\ D_e^{D4} &= \langle E_{e-1}, 1, E_{e-1} \rangle & e \geq 4 \end{aligned}$$

For instance, $D_5^{D4} = \langle 0123012401230121012301240123012 \rangle$. Note that the sequence D_e^{D4} has degree four because only four central subsequences of length 4 have not different elements ($\langle 0121 \rangle$, $\langle 1210 \rangle$, $\langle 2101 \rangle$ and $\langle 1012 \rangle$ in the previous example). This is true for any $e > 3$. When e is large, these subsequences have a negligible effect on performance.

It remains to be shown that D_e^{D4} is an e -sequence and, therefore, it can be used to implement the exchange phase e . This is shown in the following theorem, that requires a previous lemma.

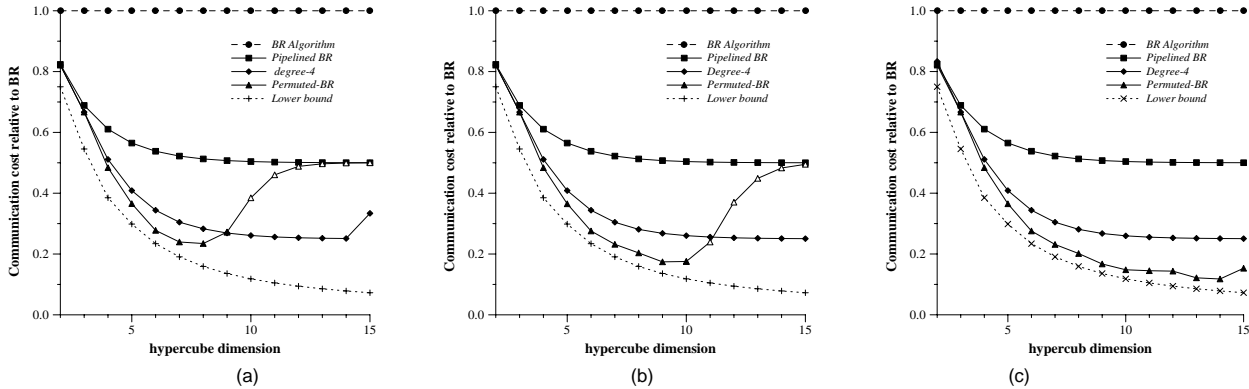


Figure 2. Performance of the BR, pipelined BR and permuted-BR and degree-4 algorithms. The matrix size m is: a) 2^{18} , b) 2^{23} and c) 2^{32} . T_w and T_s are equal to 100 and 1000 respectively.

The performance of the permuted-BR ordering approaches the lower bound when deep pipelining is used. However, when the hypercube size forces the use of shallow pipelining, it tends to be similar to the BR ordering. On the other hand, the degree-4 ordering exhibits a more stable performance behavior, since its communication cost is about one fourth of the cost of the CC-cube BR algorithm in all the considered scenarios.

5. Conclusions

We have proposed two novel Jacobi orderings: the permuted-BR ordering and the degree-4 ordering. These new orderings significantly outperform previous orderings in a hypercube with multiple ports when using the communication pipelining technique. The permuted-BR ordering is obtained by applying a series of permutations to the BR ordering. The result is an ordering that makes a nearly balanced use of all the hypercube links, which results in a performance close to the optimum when all the links of the hypercube are used simultaneously. Depending on the start-up cost and the transmission cost there are cases in which the most efficient solution is to use just a few number of links simultaneously. In this scenario, the permuted-BR ordering is not nearly optimal anymore. For such cases, we have proposed the degree-4 ordering, which outperforms by a factor of 2 the BR ordering.

References

[1] W.C. Athas and C.L. Seitz, "Multicomputers: message-passing concurrent computers," *IEEE Computer*, August 1988, pp. 9-24.

[2] M. Berry and A. Sameh, "Multiprocessor Jacobi Schemes for Dense Symmetric Eigenvalue and Singular Value Decompositions," *Proceedings of ICCP 86*, pp. 433-440, 1986.

[3] C. Bischof, "The two-sided Jacobi method on a hypercube," *SIAM Proceedings of the Second Conference on Hypercube Multiprocessors*, 1987.

[4] 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.

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

[6] 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.

[7] G.R. Gao and S.J. Thomas, "An optimal parallel Jacobi-like solution method for singular value decomposition," *Proceedings of Int'l Conf. on Parallel Processing*, 1988, pp. 47-53.

[8] J. Gotze, S. Paul, and M. Sauer, "An efficient Jacobi-like algorithm for parallel eigenvalue computation," *IEEE Trans. Comput.* C-42, 9 (1993), pp. 1058-1063.

[9] L. Díaz de Cerio, A. González and M. Valero-García, "Communication pipelining in hypercubes," *Parallel Processing Letters*, Vol.6, No.4, December 1996, pp. 507-523.

[10] G. Fox et al., *Solving Problems on Concurrent Processors*, Englewood Cliffs, N.J. Prentice-Hall, 1988.

[11] M. Lam, "Software pipelining: An effective scheduling technique for VLIW machines," *Conf. on Programming Language, Design and Implementation*, 1988, pp. 318-328.

[12] M. Mantharam and P.J. Eberlein, "Block-recursive algorithm to generate Jacobi-sets," *Parallel Computing* 19 (1993) pp. 481-496.

[13] M. Mantharam and P.J. Eberlein, "New Jacobi-sets for parallel computation," *Parallel Computing* 19 (1993) p. 437

[14] 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.

[15] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

1. In the case of the permuted-BR ordering we have used the sequences D_e^{p-BR} for all values of e . For $e < 7$ we could have used the optimal sequences $D_e^{\min-\alpha}$. However this would have had a negligible impact on the performance since it would affect only the less time-consuming exchange phases.

1 st transformation	2 nd transformation	3 rd transformation	4 th transformation
2 nd 16-subsequence: (0,15)	2 nd 15-subsequence: (0,7)	2 nd 14-subsequence: (0,3)	2 nd 13-subsequence: (0,1)
(1,14)	(1,6)	(1,2)	4 th 13-subsequence: (2,3)
(2,13)	(2,5)		6 th 13-subsequence: (6,7)
(3,12)	(3,4)	4 th 14-subsequence: (4,7)	8 th 13-subsequence: (4,5)
(4,11)		(5,6)	10 th 13-subsequence: (14,15)
(5,10)	4 th 15-subsequence: (8,15)	6 th 14-subsequence: (12,15)	12 th 13-subsequence: (12,13)
(6,9)	(9,14)	(13,14)	14 th 13-subsequence: (8,9)
(7,8)	(10,13)	8 th 14-subsequence: (8,11)	18 th 13-subsequence: (10,11)
	(11,12)	(9,10)	

Figure 3. Transformations to generate D_{17}^{p-BR} . Each tuple denotes a transposition of the two link identifiers that constitute the tuple.

APPENDIX

In the following it is shown that the value of α corresponding to the D_e^{p-BR} sequence tends to be 1.25 times the lower bound given by $\lceil (2^e - 1)/e \rceil$ for large values of e .

To simplify the expressions we assume that $e-1$ is a power of two. For any other value of e , the corresponding α will take an intermediate value between the α of the two surrounding powers of two and thus, the asymptotic behavior observed for powers of two will also apply to any other value.

Let $e = 2^S + 1$. In this case, the sequence D_e^{p-BR} will be obtained after applying S transformations to the sequence D_e^{BR} . To illustrate the following development we will use the particular case of $e=17$. This case requires four transformations, which are shown in figure 3.

We denote by $D_e^{p-BR}(k)$ the sequence that is obtained after transformation k , with $k \in [0, S-1]$. In general, some of the elements in $D_e^{p-BR}(k)$ will be affected by further transformations while some others will not. This second group of elements will be referred to as the elements that have been fixed after transformation k . We denote by $r_k(i)$ the number of repetitions of link i in $D_e^{p-BR}(k)$ (i.e. the amount of elements in $D_e^{p-BR}(k)$ that are equal to i) that have been fixed after transformation k . We denote by $p_k(i)$ the number of repetitions of link i in $D_e^{BR}(k)$ that will be affected by further transformations. Obviously, the number of repetitions of link i in the final sequence D_e^{p-BR} is $r_0(i) + \dots + r_{S-1}(i) + p_{S-1}(i)$. In the following, we derive analytical expressions for $r_k(i)$ and $p_k(i)$ that will enable the analysis of the asymptotic behavior of α .

Notice that the first transformation is a permutation that is applied to the second $(e-1)$ -subsequence and it consists of a series of transpositions, as defined in section 3.2.1. Such transpositions are defined by pairing all elements from 0 to $e-2$ in such a way that the most frequent element is exchanged with the least frequent one. The same with the second-most and second-least frequent elements, and so on. Since the first and second $(e-1)$ -subsequences are exactly equal but the permutation is only applied to the second one, after the permutation, the number of repetitions of the most frequent element will decrease to the half

it was before plus the half of the number of repetitions of the least frequent element before the permutation. The number of repetitions of the least frequent element will increase in the same amount as the number of repetitions of the most frequent element is decreased.

Notice that any element $i \in [0, (e-1)/2-1]$ appearing in the second $(e-1)$ -subsequence is fixed after the first transformation. The same is true for any element $i \in [(e-1)/2, e-2]$ in the first $(e-1)$ -subsequence. In addition the number of repetitions of elements $i \in [0, (e-1)/2-1]$ in the second $(e-1)$ -subsequence is $2^0, 2^1, \dots, 2^{(e-1)/2-1}$ respectively. For elements $i \in [(e-1)/2, e-2]$ in the first $(e-1)$ -subsequence we have the same number of repetitions but in reverse order.

Something similar happens after the second permutation. In this case, any element $i \in [0, (e-1)/2^2-1]$ in the second $(e-2)$ -subsequence is fixed. The same is true for all the following cases:

- Any element $i \in [(e-1)/2^2, (e-1)/2-1]$ in the first $(e-2)$ -subsequence.
- Any element $i \in [(e-1)/2, 3(e-1)/2^2-1]$ in the third $(e-2)$ -subsequence.
- Any element $i \in [3(e-1)/2^2, e-2]$ in the fourth $(e-2)$ -subsequence.

The number of repetitions of elements $i \in [0, (e-1)/2^2-1]$ of the second $(e-2)$ -subsequence that have been fixed is:

$$2^{(e-1)/2-1}, \dots, 2^{3(e-1)/2^2-2}$$

For the other intervals of elements and their corresponding $(e-2)$ -subsequences, the number of repetitions is the same as those of the first interval but in reverse order for some of them.

In general, due to the recursive nature of D_e^{p-BR} , we have that after each transformation $k \in [0, S-1]$, the number of repetitions of any element $i \notin [0, (e-1)/2^{k+1}-1]$ that have been fixed is the same as that of some other element $j \in [0, (e-1)/2^{k+1}-1]$.

On the other hand, after the first transformation, any element $i \in [0, (e-1)/2-1]$ in the first $(e-1)$ -subsequence will be changed by any further permutation. The same is true for any element $i \in [(e-1)/2, e-2]$ in the second $(e-1)$ -subsequence. In addition the number of repetitions of

the elements of the first interval that appear in the first $(e-1)$ -subsequence is 2^{e-2} , 2^{e-3} , ..., $2^{(e-1)/2}$ respectively. For the second interval and the second $(e-1)$ -subsequence we have the same number of repetitions but in reverse order. In addition, any further permutation that reduces/increases the number of repetitions of an element of the first interval will result in exactly the same reduction/increase in that dimension of the second interval that has the same number of repetitions.

In general, due again to the recursive nature of D_e^{p-BR} , we have that after each transformation $k \in [0, S-2]$, the number of repetitions of any element $i \notin [0, (e-1)/2^{k+1}-1]$ that will be affected by further permutations is the same as that of some other dimension $j \in [0, (e-1)/2^{k+1}-1]$. Besides, the number of repetitions of these two elements i and j will be increased/decreased by the same amount by any further permutation, thus they will be the same.

As a conclusion of the previous analysis, in order to obtain the range of values of functions $r_k(i)$ and $p_k(i)$, it is enough to compute them for elements $i \in [0, (e-1)/2^{k+1}-1]$ since its value for any other element will be the same as that for an element of this interval.

Lemma 2.

$$\left(p_k(i) = 2^{e-2-k-i} \right) \quad \forall k \in [-1, S-1] \quad \forall i \in [0, \frac{e-1}{2^{k+1}} - 1]$$

Proof. For $k=-1$, that is, before applying any permutation, the equation holds since it corresponds to the number of repetitions of each dimension in D_e^{BR} . Then, any transformation k exchanges every dimension $i \in [0, (e-1)/2^{k+1}-1]$ with dimension $(e-1)/2^k - i$, but just in the second $(e-k-1)$ -subsequence. Therefore, the number of remaining repetitions of dimension i in the first $(e-k-1)$ -subsequence is halved by each transformation k since dimension i appears the same number of times in the first and second $(e-k-1)$ -subsequences. \square

Lemma 3.

$$r_k(i) = 2^{\frac{e-1}{2^k} + i - k - 1} \quad \forall k \in [0, S-1] \quad \forall i \in [0, \frac{e-1}{2^{k+1}} - 1]$$

Proof. In transformation k , a given element $i \in [0, (e-1)/2^{k+1}-1]$ is exchanged with element $(e-1)/2^k - i$ in the second $(e-k-1)$ -subsequence. Therefore, the number of repetitions of i in the second $(e-k-1)$ -subsequence that after this permutation will appear in the second $(e-k-1)$ -subsequence corresponds to the number of repetitions of element $(e-1)/2^k - i$ in that subsequence before this permutation. That is, $r_k(i) = p_{k-1}((e-1)/2^k - i)/2$, which results in the expression shown in the lemma.

Lemma 4. After transformation k , the maximum number of repetitions of any element that will not be affected by any other permutation because they are in the second $(e-k-1)$ -subsequence is equal to

$$N_k = 2^{\frac{e-1}{2^{k+1}} - k - 2} \quad \forall k \in [0, S-1]$$

Proof. It is obvious that $N_k = \max r_k(i)$. Using the expression of $r_k(i)$ given in lemma 3, the above result is obtained. \square

Lemma 5.

$$N_k \leq 2^{e-2S+k} \quad \forall k \in [0, S-2]$$

Proof. Given the definition of N_k provided by lemma 4, lemma 5 can be proved by showing that:

$$\frac{e-1}{2^{k+1}} \geq 2S-2k-2$$

Since $e-1 = 2^S$, the previous inequality holds if $2^{S-k-1} \geq 2(S-k-1)$. Since the left-hand side expression grows faster than the right-hand side expression when k decreases, it is enough to prove that the inequality holds for the highest value of k . By substituting k by $S-2$, we obtain the same value in both sides. \square

Lemma 6.

$$\sum_{k=0}^{S-2} N_k \leq 2^{e-S-1} - 2^{e-2S}$$

Proof. This is a straightforward result that comes from computing the sum from 0 to $S-2$ of the bound computed for N_k in the lemma 5. \square

Theorem 2. The value of α corresponding to the sequences D_e^{p-BR} is bounded by the following expression:

$$\alpha \leq \frac{2^e}{e-1} + \frac{2^{e-2}}{e-1} - \frac{2^e}{(e-1)^2}$$

Proof. The value of α is bounded by the sum of the following terms: the maximum value of $p_{S-1}(i)$; N_0 ; N_1 ; ... and N_{S-1} . That is

$$\alpha \leq p_{S-1}(0) + N_{S-1} + \sum_{k=0}^{S-2} N_k$$

By adding the value of $p_{S-1}(0)$ given by lemma 2; the value of N_{S-1} given by lemma 4; and the bound given by lemma 6, the bound established by this theorem follows. \square

Theorem 3. The upper-bound of α established by theorem 2 tends to 1.25 times the lower bound given by

$$\left[\left(2^e - 1 \right) / e \right]$$

Proof. This is proved by observing that

$$\lim_{e \rightarrow \infty} \frac{\frac{2^e}{e-1} + \frac{2^{e-2}}{e-1} - \frac{2^e}{(e-1)^2}}{\left(2^e - 1 \right) / e} = 1.25 \quad \square$$