Executing Algorithms with Hypercube Topology on Torus Multicomputers

Antonio González, *Member, IEEE*, Miguel Valero-García, *Member, IEEE*, and Luis Díaz de Cerio

***Abstract*—Many parallel algorithms use hypercubes as the communication topology among their processes. When such algorithms are executed on hypercube multicomputers the communication cost is kept minimum since processes can be allocated to processors in such a way that only communication between neighbor processors is required. However, the scalability of hypercube multicomputers is constrained by the fact that the interconnection cost-per-node increases with the total number of nodes. From scalability point of view, meshes and toruses are more interesting classes of interconnection topologies. This paper focuses on the execution of algorithms with hypercube communication topology on multicomputers with mesh or torus interconnection topologies. The proposed approach is based on looking at different embeddings of hypercube graphs onto mesh or torus graphs. The paper concentrates on toruses since an already known embedding, which is called *standard embedding*, is optimal for meshes. In this paper, an embedding of hypercubes onto toruses of any given dimension is proposed. This novel embedding is called *xor embedding*. The paper presents a set of performance figures for both the standard and the xor embeddings and shows that the latter outperforms the former for any torus. In addition, it is proven that for a one-dimensional torus (a ring) the xor embedding is optimal in the sense that it minimizes the execution time of a class of parallel algorithms with hypercube topology. This class of algorithms is frequently found in real applications, such as FFT and some class of sorting algorithms.**

***Index Terms*—Graph embeddings, hypercubes, scalable distributed memory multiprocessors, torus multicomputers, mapping of parallel algorithms.**

I. INTRODUCTION

A HYPERCUBE communication topology is frequently found in real parallel applications. Some examples include parallel algorithms for FFT, sorts, etc. [3], [11]. These algorithms will be called *hypercube algorithms* or *d-cube algorithms,* where d is the number of dimensions of the hypercube. A hypercube algorithm of dimension d or d-cube algorithm, consists of 2^d processes labeled from 0 to $2^d - 1$ such that every process communicates only with its d neighbors, one in each dimension of the d-cube.

In this paper, the problem of executing d-cube algorithms on *multicomputers* [1] is considered. A multicomputer is a distributed memory multiprocessor in which the *nodes* (processor + local memory) are interconnected through point-to-point links.

Manuscript received May 9, 1994; revised Nov. 1, 1994.

A. González, M. Valero-García, and L. Díaz de Cerio are with the Universitat Politècnica de Catalunya, Departament d' Arquitectura de Computadors, c/ Gran Capitán s/n, Campus Nord–Edifici D6 E-08071 Barcelona, Spain. e-mail: {antonio, miguel, ldiaz}@ac.upc.es.

IEEECS Log Number D95003.

The nodes of a multicomputer are interconnected according to a given pattern or interconnection topology. If this topology is a hypercube of dimension d (d-cube multicomputer) then the d-cube algorithm can be executed on the multicomputer in such a way that neighbor processes are mapped onto adjacent nodes (nodes directly connected through a point-to-point link). In this case, it is said that each process of the d-cube algorithm has all its d neighbors at distance 1 in the multicomputer (i.e., all required communication is between neighbor nodes). In this way, the cost of the communication component of the d-cube algorithm is kept minimum when it is executed on a hypercube multicomputer.

An important drawback of hypercube as interconnection topology for multicomputers is that it is not scalable. In a d-cube multicomputer each of the 2^d nodes is directly connected to other d nodes through point-to-point links. Therefore, the cost (and the complexity) of the interconnection hardware per node increases with the number of nodes. Other interconnection topologies, such as meshes or toruses are considered more suitable for multicomputers with a large number of nodes, since the interconnection cost per node does not depend on the total number of nodes [13]. For instance, each node of a two-dimensional torus multicomputer is directly connected to other 4 nodes, it does not matter the number of nodes of the multicomputer.

To execute a d-cube algorithm on a multicomputer with topology other than hypercube, the first step is to find a mapping function that allocates each process of the parallel program onto a given processor of the multicomputer. The problem can be formulated as finding an *embedding* of the graph that represents the topology of the program (a hypercube) onto the graph that represents the topology of the multicomputer (a mesh or a torus).

The problem of embedding a given source graph into a destination graph has been extensively studied in the literature. In particular, embedding any type of graph into a hypercube is a widely studied topic (see, for instance, [2], [7], [11], [12], just to mention a few recent works). However, the problem of embedding hypercubes onto a mesh or a torus has not been so extensively studied. In Section II.C we review the most relevant works in this subject.

When the topology of the algorithm and the multicomputer are different, it may be impossible to allocate neighbor processes to neighbor processors. For instance, in a two-dimensional torus multicomputer, every process of a d-cube algorithm has at most four of its d neighbors at distance 1. It has at least $d - 4$ neighbors at a distance greater than 1. A message to any of these "far" neighbors is routed through the

point-to-point links and nodes which are found along the path to the destination node. A good mapping of a parallel algorithm onto a multicomputer will keep the neighbor processes as close as possible in the multicomputer, minimizing in this way the communication cost of the execution.

This paper begins by reviewing some related work on embeddings and then, it concentrates on a particular type of embeddings that is called *embeddings with constant distances*. It will be shown that these embeddings are more adequate for our purposes, that is, for executing d-cube algorithms onto meshes or toruses. A well known embedding of hypercubes onto meshes is the so called *standard embedding* [8]. It is an embedding with constant distances and it is optimal for meshes of any given dimension. In consequence, the contribution of this paper centers on embeddings of hypercubes onto toruses.

A new embedding, called *xor embedding*, is proposed. The paper presents a set of performance figures and shows that this embedding outperforms the standard embedding when it is used as the mapping function of a d-cube algorithm onto a torus multicomputer. In addition, it is proven that the xor embedding is optimal for one-dimensional toruses (also called rings).

This paper is organized as follows. In Section II, we introduce some notation and describe more precisely the contribution of this paper as well as some related work. Sections III presents the xor embedding. Section IV compares the performance of the xor embedding with that of the standard embedding using a set of different performance metrics. In Section V, it is proven that the proposed embedding is optimal for rings in the sense that it results in the shortest execution time of a class of d-cube algorithms. Finally, some concluding remarks are presented in Section VI.

II. Preliminaries and Related Work

A. Definitions

A *d-cube algorithm* is a parallel algorithm that consists of 2^d processes such that every process communicates with exactly other d processes. These d processes are called its neighbors. We also say that the communication topology of the algorithm is a hypercube. That means that the 2^d processes can be labeled from 0 to $2^d - 1$ in such a way that processes n and m are neighbor (i.e., they communicate) if the binary codes for n and m differ in a single bit. If this bit is the ith bit then m is the neighbor of n in dimension i, and n is the neighbor of m in the same dimension. Then, it is written:

$$m = N_i(n)$$

$$n = N_i(m)$$

In this paper, we focus on d-cube algorithms in which every process has the following structure:

```
do i=0,d-1
    compute
    communicate with neighbor in dimension i
enddo
```

In this algorithm every process consists of d stages, each of them composed of a computation phase followed by a communication phase. In each stage, every process uses a different dimension to exchange information with one of its neighbors.

The duration of the computation phase and the amount of information to be exchanged is assumed to be the same for all the stages and all the processes of the d-cube algorithm. A d-cube algorithm with the above features will be called a *compute-and-communicate d-cube algorithm*, or a *CC d-cube algorithm* for short. This kind of d-cube algorithms are common in real applications like FFT, some type of sorts, etc. [3], [11].

Parallel algorithms can be modeled by graphs. The vertices of the graph represent the processes of the algorithm and the edges of the graph represent the neighbor relationship among processes. A multicomputer can also be modeled by a graph. The vertices of the graph represent the nodes of the multicomputer and the edges of the graph represent the point-to-point links which interconnect these nodes. The terms edge and link will be used indistinctly in this paper.

Multicomputers can be classified according to their interconnection topology. The work presented in this paper focuses on mesh and torus multicomputers, since they have scalable interconnection topologies.

A $(k_1, k_2, ..., k_c)$ c-dimensional torus is an undirected graph in which the nodes can be labeled as c-tuples $(i_1, i_2, ..., i_c)$, $0 \le i_j < k_j$. Every node $(i_1, i_2, ..., i_c)$ of the graph has two neighbors in each dimension of the torus. Its left neighbor in dimension j is $(i_1, ..., (i_j - 1) \bmod k_j, ..., i_c)$ and its right neighbor in this dimension is $(i_1, ..., (i_j + 1) \bmod k_j, ..., i_c)$.

A $(k_1, k_2, ..., k_c)$ c-dimensional mesh is an undirected graph in which the nodes can be labeled as c-tuples $(i_1, i_2, ..., i_c)$, $0 \le i_j < k_j$. Every node of the graph has two neighbors in each dimension j of the mesh if $0 < i_j < k_j - 1$. Its left neighbor is $(i_1, ..., i_j - 1, ..., i_c)$ and its right neighbor is $(i_1, ..., i_j + 1, ..., i_c)$. If $i_j = 0$, the node has only a right neighbor and if $i_j = k_j - 1$ then it only has a left neighbor.

A *line* is a one-dimensional mesh while a one-dimensional torus is called a *ring*.

Fig. 1 shows some examples and illustrates how their nodes are labeled.

The *distance* in a graph between two vertices is the minimum number of edges that join those vertices. In the particular case of the graph that models a d-cube, the distance between two vertices is known as the Hamming distance (number of different bits in their binary representations).

An *embedding* of graph G into graph H is an injection from the vertices of G to the vertices of H. In this paper, our attention is restricted to embeddings in which G and H have the same number of vertices, and therefore the mapping is given by a bijective function.

The problem of executing a CC d-cube algorithm on a multicomputer can be restated as the embedding of graph G, which represents the CC d-cube algorithm, onto graph H, which represents the multicomputer.

The *dilation* of an edge (n, m) of G (edge joining vertices n and m) is the distance in H between $f(n)$ and $f(m)$.

If G models a CC d-cube algorithm, an edge exists between vertices n and m if $m = N_i(n)$, for some $i \in [0, d - 1]$. The dilation of this edge will be denoted by $D_i(n)$. Obviously, since

$n = N_i(m)$, $D_i(n) = D_i(m)$. When a CC d-cube algorithm is executed on a multicomputer, as defined by a given embedding f, a communication between processes n and $N_i(n)$ (required in iteration i of the CC d-cube algorithm) is implemented by a message which is routed through $D_i(n)$ point-to-point links and $D_i(n) - 1$ nodes of the multicomputer represented by H, which are found in the shortest path between nodes $f(n)$ and $f(N_i(n))$. In the following, a store and forward routing strategy is assumed. Therefore, the cost of sending a message from $f(n)$ to $f(N_i(n))$ is proportional to $D_i(n)$.

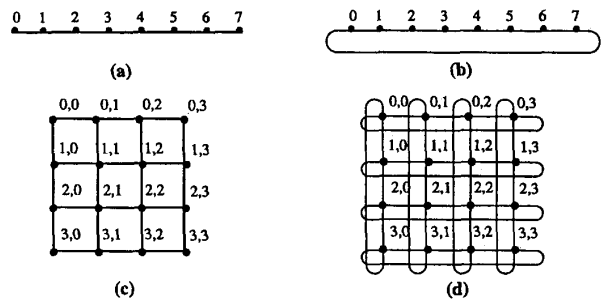


Fig. 1. Different types of multicomputers: (a) line, (b) ring, (c) (4, 4) mesh, and (d) (4, 4) torus. The picture also shows how their nodes are labeled.

B. Contributions

As it was mentioned in the introduction, this paper focuses on executing CC d-cube algorithms on scalable multicomputers. The function that maps processes onto processors is an embedding of the graph defined by the communication topology of the algorithm (hypercube) onto the graph defined by the interconnection topology of the multicomputer. In particular, we are interested in torus multicomputers since for meshes, an already known embedding, called *standard embedding* and described in the next section, is optimal for CC d-cube algorithms.

The work presented in this paper centers on those embeddings in which $D_i(n) = D_i$ ($i \in [0, d-1]$ and $n \in [0, 2^d - 1]$). This means that every process has its neighbor in dimension i at the same distance in the target multicomputer. In the following, an embedding with this feature is called *embedding with constant distances* and the values of $D_i(i \in [0, d-1])$ are called the *distances* of the embedding.

Embeddings with constant distances have the property that every process takes the same time to communicate in any given stage of the CC d-cube algorithm. Because the duration of the compute phase is also the same for every process, waiting intervals are avoided since neighbor processes arrive at the same time at the point where they have to communicate. This fact will be illustrated later through an example.

In this paper, an embedding with constant distances of hypercubes onto toruses of any arbitrary dimension is proposed. The embedding is called *xor embedding*. It will be shown that this embedding outperforms the standard embedding using a set of different performance metrics. Moreover, we prove that the proposed embedding is optimal for rings (one-dimensional toruses) in the sense that it minimizes the execution time of CC d-cube algorithms when they are executed on a ring multicomputer. Another additional property of the proposed embeddings is their simplicity, which means a negligible cost to compute the location of any process in the multicomputer. Some preliminary results about the xor embedding were presented in [4].

C. Related Work

The problem of embedding d-cubes onto meshes and toruses has been previously considered by other authors. Here, a review of the most related work is presented.

Matic presents in [10] a study of the *standard embedding* (defined below) of d-cubes onto two-dimensional meshes and toruses. To define the standard embedding (which will be denoted by f_{std}) of a d-cube onto a line or a ring, the nodes of the target multicomputer are numbered from 0 to $2^d - 1$ (see Fig. 1a and 1b). Then, the standard embedding is defined by (see Fig. 2a):

$$f_{std}(n) = n$$

In general, the standard embedding of a d-cube onto a (k_1, k_2, ..., k_c) c-dimensional mesh or torus is defined as follows:

$$f_{std}(n) = (p_1, p_2, \ldots, p_c)$$

where

$$p_i = \left(n \bmod \prod_{j=1}^{i} k_j \right) div \prod_{j=1}^{i-1} k_j$$

Fig. 2b shows an example in which $c = 2$ and $k_1 = k_2 = 4$. Obviously, the standard embedding is an embedding with constant distances. For the particular case in which $k_i = 2^{d/c}$, $i \in [1, c]$, the distances of the standard embedding are:

$$D_i = 2^{i \bmod (d/c)} \qquad i \in [0, d-1]$$

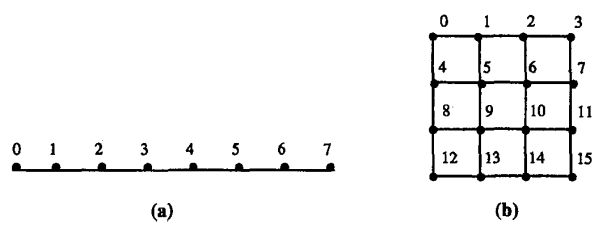


Fig. 2. Standard embeddings of: (a) a 3-cube onto a line or a ring and (b) a 4-cube onto a (4, 4) mesh or torus. Each label indicates which vertex of the d-cube is mapped onto each node of the multicomputer. Wraparound links are not shown for clarity.

It can be shown that the standard embedding is optimal for meshes, in the sense that it minimizes the average distance [5], which in turn results in the shortest execution time. However, it is not optimal for toruses, as it will be shown later in this paper.

Harper in [6] and Lai and Spague in [8] solve the problem of embedding d-cubes onto meshes to minimize the dilation of the embedding (the maximum dilation of any edge). Both proposals use the *byweight* embedding, denoted by f_{bw}, which is not an embedding with constant distances. Next, this embedding is briefly described.

In the case of a line, the labels of the vertices which represent the processes of the d-cube algorithm are ordered by their weights. The weight of a label is the number of 1s in its binary representation. Labels with the same weight are ordered in descending order. Then, the processes of the d-cube ordered in that way are allocated to the nodes of the line, from left to right. Fig. 3a shows an example. The byweight embedding can be extended to meshes of any dimension. In particular, Lai and Spague extend this embedding to two-dimensional meshes in [8]. Fig. 3b shows an example.

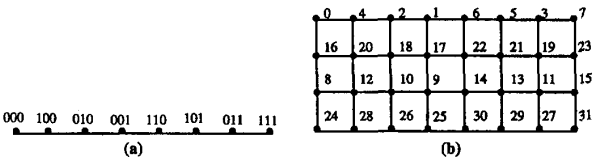


Fig. 3. Byweight embeddings of: (a) a 3-cube onto a line and (b) a 5-cube onto a (8, 4) mesh.

The byweight embedding minimizes the dilation of the embedding for lines, and it has a lower dilation than the standard embedding for two-dimensional meshes. This is an interesting property in some particular applications of embeddings. For instance, Lai and Spague propose this embedding to solve the problem of placing the processors of a hypercube on a printed circuit board or a chip (which can be modeled as a two-dimensional mesh). However, the byweight embedding is not an embedding with constant distances, which is an important property in the context of executing CC d-cube algorithms onto multicomputers. Variable distances result in waiting intervals during the execution of the CC d-cube algorithm. These are due to the fact that two neighbors that are going to communicate finish their respective previous computation at different times. The one that finishes earliest must wait for the other to finish. These waiting intervals contribute to increase the execution time. To illustrate this fact, Fig. 4 shows an example in which the execution time of a CC 3-cube algorithm on a line for both the standard embedding and the byweight embedding are compared. The waiting intervals which contribute to make the byweight embedding run slower than the standard embedding are also shown.

In [9], Ma and Tao proposed several embeddings among toruses and meshes of different dimensions. Their proposals are based on generalizing the concept of gray code from radix-2 numbering system to mix-radix numbering systems. Since a d-cube can also be seen as a d-dimensional mesh or torus with two elements in each dimension, their embedding can also be applied to solve the problem addressed in this paper. However, they focus on minimizing the dilation (the longest dilation of any link of the d-cube) and, therefore, the resulting embeddings in general do not have constant distances, which is a desirable property for our objective. However, if one starts with a d-cube represented by means of a (2, 2, ..., 2) d-dimensional mesh or torus, then the resulting embedding onto a ring or a two-dimensional torus has constant distances. Nevertheless, its average distance and therefore its performance

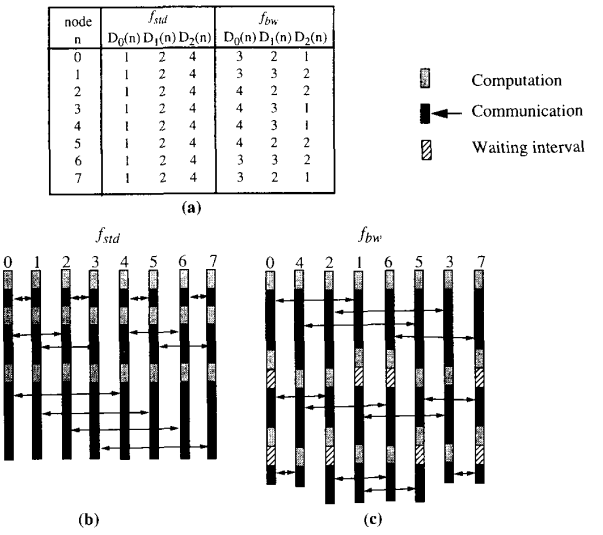


Fig. 4. (a) Dilations for the standard and byweight embeddings (d = 3). Executing a CC 3-cube algorithm on a line using: (b) the standard embedding and (c) the byweight embedding.

for executing our target algorithm is worse than the embedding proposed in this paper.

III. The XOR Embedding

Since the standard embedding is optimal for meshes, we focus just on toruses. The proposed embedding is called *xor embedding* and it is denoted by f_{xor}. It belongs to the class of embeddings with constant distances. In this section, the xor embedding for the case of a one-dimensional torus (a ring) is first described, and then it is generalized for any dimension.

A. One-Dimensional Torus (Ring)

Given a positive integer x, let $x(i)$ denote the ith bit of the binary representation of x. The least significant bit is considered to be the 0th bit. Let G be the graph which represents the CC d-cube algorithm and R be the graph which represents the ring multicomputer. Assume that the vertices of R are labeled from 0 to $2^d - 1$ clockwise (see Fig. 1b). Let $(n(d-1), n(d-2), ..., n(1), n(0))$ be the label (in binary code) of vertex n in G. This vertex is mapped onto vertex $m = f_{xor}(n)$ in R, whose label in binary code $(m(d-1), ..., m(0))$ is:

$$m(i) = n(i) \qquad i \in [0, d-1],\ i \neq d-2$$
$$m(d-2) = XOR\,(n(d-1), n(d-2))$$

where $XOR\ (a, b)$ is the exclusive-or of bits a and b. Fig. 5 shows an example for $d = 4$.

B. General Case

The xor embedding of a d-dimensional hypercube onto a $(2^{d1}, 2^{d2}, ..., 2^{dc})$ c-dimensional torus such that $d_1 + d_2 + ... + d_c = d$ is now presented.

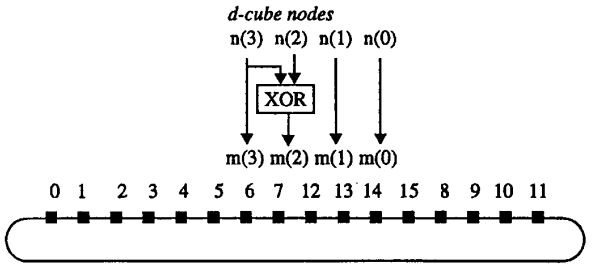


Fig. 5. An xor embedding of a 4-cube onto a ring. The labels indicate which node of the d-cube is mapped onto the corresponding node of the ring.

Let us first define K_j in the following way: $K_1 = 0$, and for every $1 < j \leq c + 1$ we have that:

$$k_j = \sum_{j=1}^{j-1} d_i$$

Let G be the graph which represents the d-cube and T be the graph which represents the torus. Then, vertex n of G is mapped onto vertex $(m_1, m_2, ..., m_c) = f_{xor}(n)$ in T as follows:

$$m_j(i) = n(i + K_j) \qquad i \in [0, d_j - 1], i \neq d_j - 2$$
$$m_j(d_j - 2) = XOR\,(n(K_{j+1} - 1), n(K_{j+1} - 2))$$

Fig. 6 shows an example for $d = 6$. It can be noted that both the standard and the xor embedding of a d-cube onto a c-dimensional torus can be viewed as multiple embeddings of smaller hypercubes onto rings. For instance, in Fig. 6, nodes from 8 to 13 of the 6-cube constitute a 3-cube that is mapped onto the 8 nodes of the second row of the torus which constitute a ring. This embedding is again an xor embedding.

Note the simplicity of function $f_{xor}(n)$. This function, which is used very frequently for routing messages during the execution of the CC d-cube algorithm, consists of simple bit operations, and its computational cost is negligible.

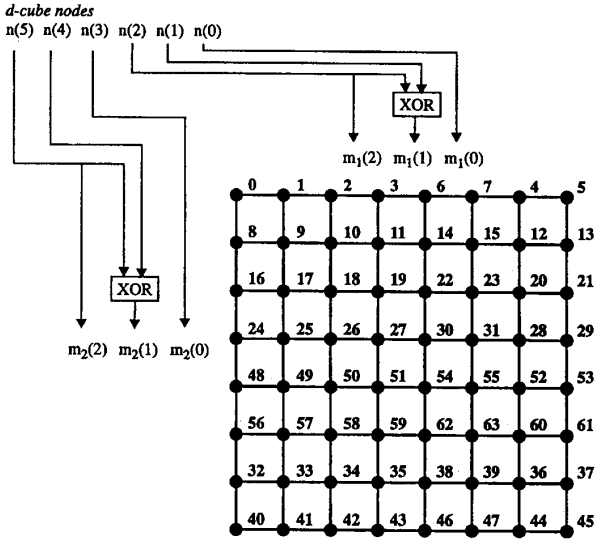


Fig. 6. An xor embedding of a 6-cube onto a (8, 8) torus. The wraparound links are not shown, for clarity.

IV. Performance Analysis

In this section, the performance of the xor embedding and the standard embedding are compared using a set of different performance metrics. Most of these metrics were used by Matic in [10] to evaluate the standard embedding for two-dimensional meshes and toruses. Here, the corresponding expressions for both the standard and the xor embeddings of a d-cube onto a $(2^{d1}, 2^{d2}, ..., 2^{dc})$ c-dimensional torus are derived. In some cases where the general expressions are not easy to compare, we derive the expression corresponding to the particular case of a *squared torus*. A squared torus is a $(2^{d/c}, 2^{d/c}, ..., 2^{d/c})$ c-dimensional torus, that is, a torus whose all dimensions have the same size. These list of metrics is the following:

- The *execution time* $(T_f(d_1, d_2, ..., d_c))$. This represents the execution time of a CC $(d_1 + d_2 + ... + d_c)$-cube algorithm onto a $(2^{d1}, 2^{d2}, ..., 2^{dc})$ c-dimensional torus when the embedding f is used as the mapping function.
- The *links dilation spectrum* $(A^f_{d_1,d_2,...,d_c}(D))$. This gives the number of links with dilation D when a $(d_1 + d_2 + ... + d_c)$-cube is embedded onto a $(2^{d1}, 2^{d2}, ..., 2^{dc})$ c-dimensional torus as defined by the mapping function f.
- The *longest dilation* $D^f_{max}(d_1, d_2, ..., d_c))$. This is the maximum dilation of any link of the hypercube when it is embedded onto a $(2^{d1}, 2^{d2}, ..., 2^{dc})$ c-dimensional torus as defined by the embedding f.
- The *total dilation* $(D^f_t(d_1, d_2 ..., d_c))$. This represents the sum of dilations of all links of the hypercube.
- The *maximum load and minimum load* $(L^f_{max}(d_1, d_2, ..., d_c), L^f_{min}(d_1, d_2, ..., d_c))$. The load of a node due to communication tasks is measured as the number of links of the hypercube that traverse that particular node (those links that begin or finish at that node are not considered). These parameters give the maximum and minimum value of the load of any node as a result of using the embedding f.
- The *average load* $(L^f_{ave}(d_1, d_2, ..., d_c))$. This is the average load of a node due to communication tasks.

A. Execution Time

When using an embedding as a mapping function of a parallel algorithm onto a multicomputer, the most important performance measure of the embedding is the time that the execution of the algorithm takes as a result of using such a mapping.

Let T_a be the duration of the arithmetic computation phase in every stage of the CC d-cube algorithm, when it is executed on the target multicomputer. Let T_c be the cost of sending a message through a point-to-point link on the multicomputer.

The time to execute a CC d-cube algorithm on a multicomputer with 2^d nodes, using the embedding f can be expressed as:

$$T_f = dT_a + T_{cf}$$

where T_{cf} is the cost of the communication component of the CC d-cube algorithm. T_{cf} can be expressed as follows:

$$T_{cf} = max\{T_{d-1}(n): n = 0..2^d - 1\} \quad \text{(a)}$$

$$T_i(n) = D_i(n)T_c + max\{T_{i-1}(n), T_{i-1}(N_i(n))\}$$
$$T_{-1}(n) = 0 \quad \text{(b)}$$

In the above expressions, $T_i(n)$ is the cost of the communication component for process n from the beginning of the execution to the end of stage i. Expression (a) indicates that T_{cf} is equal to the highest communication component cost of any process at the end of the d stages of the CC d-cube algorithm. Expression (b) gives the communication component cost for process n at the end of stage i. In this stage process n must exchange information with its neighbor $N_i(n)$. The cost of exchanging this information is $D_i(n)T_c$ (since a store and forward routing is assumed). However, this exchange cannot start until both processes n and $N_i(n)$ are ready to do it. In general, either process n or process $N_i(n)$ will have to wait for its neighbor to arrive to the point in which communication can be started. This is why the term "max" appears in expression (b). These idle intervals were called *waiting intervals* in Fig. 4.

Obviously, if the multicomputer has a d-cube interconnection topology then the best embedding is $f(n) = n$ (identity embedding). In this case $D_i(n) = 1$ (for every i and n) and the execution time is

$$T_f = d(T_a + T_c)$$

If the embedding has constant distances then $D_i(n) = D_i$ for every n. In this case, the time to execute a CC d-cube algorithm onto a multicomputer, as defined by an embedding f is:

$$T_f = \sum_{i=0}^{d-1} (T_a + D_i T_c) = dT_a + T_c \sum_{i=0}^{d-1} D_i = d(T_a + T_c D_a)$$

where D_a is the average distance of the embedding:

$$D_a = \frac{\sum_{i=0}^{d-1} D_i}{d} = average\ distance\ (f)$$

For a $(2^{d1}, ..., 2^{dc})$ c-dimensional torus, the average distance of the standard embedding is:

$$D_a^{std} = \frac{\left(\sum_{i=1}^{c} 2^{d_i}\right) - c}{d}$$

and the average distance corresponding to the xor embedding is:

$$D_a^{xor} = \frac{\left(\sum_{i=1}^{c} 3 \cdot 2^{d_i - 2}\right) - c}{d}$$

Since the execution time of the CC d-cube algorithm is proportional to the average distance of the embedding we can conclude that the standard embedding results in about a 33% increase in the execution time when compared with the xor embedding.

Obviously, the embedding with constant distances which minimizes the execution time of the CC d-cube algorithm is that whose average distance D_a is minimum. An embedding with such property is said to be optimal. The standard embedding is optimal for meshes of any dimension but not for toruses since we have just seen that the xor embedding outperforms it. In addition, it will be proven in Section V that the xor embedding is optimal for one-dimensional toruses.

B. Links Dilation Spectrum

B.1 Standard Embedding on Rings

Here, the links dilation spectrum for the standard embedding in the case of a one-dimensional torus is derived.

Notice that any node of the hypercube has its neighbor in dimension i at distance 2^i in the torus. Since there are 2^d nodes, we have 2^{d-1} links with dilation 2^i for each $i \in [0, d-1]$. We can then conclude that the links dilation spectrum is:

$$A_d^{std}(2^i) = 2^{d-1} \quad ; \quad (0 \le i < d-1)$$

B.2 Xor Embedding on Rings

In the xor embedding for rings, every node has a neighbor at distance 2^i for each $i \in [0, d-2]$ and two neighbors at distance 2^{d-2}. In consequence, the links dilation spectrum is as follows:

$$A_d^{xor}(2^i) = \begin{cases} 2^{d-1}; & 0 \le i \le d-3 \\ 2^d; & i = d-2 \end{cases}$$

B.3 General Case

The links dilation spectrum for both the standard and xor embeddings can be computed from the spectrum of the one-dimensional case using the following expression:

$$A_{d_1,d_2,...,d_c}(2^i) = \sum_{j=1}^{c} \left(A_{d_j}(2^i) \prod_{\substack{k=1, \\ k \ne j}}^{c} 2^{d_k} \right)$$

In the particular case of a squared c-dimensional torus, for the standard embedding we have that

$$A_{d_1,d_2,...,d_c}^{std}(2^i) = c \cdot 2^{d-1}; (0 \le i \le d/c - 1)$$

and, assuming $d/c \ge 2$, for the xor embedding the corresponding expression is

$$A_{d_1,d_2,...,d_c}^{xor}(2^i) = \begin{cases} c \cdot 2^{d-1}; & 0 \le i \le d/c - 3 \\ c \cdot 2^d; & i = d/c - 2 \end{cases}$$

C. Longest Dilation

The longest dilation can be obtained from the links dilation spectrum functions previously developed. For the standard embedding we have that

$$D_{max}^{std}(d_1, d_2, ..., d_c) = max(2^{d_1 - 1}, 2^{d_2 - 1}, ..., 2^{d_c - 1})$$

and for the xor embedding the corresponding expression is (assuming $d_i \ge 2$)

$$D_{max}^{xor}(d_1, d_2, ..., d_c) = max(2^{d_1 - 2}, 2^{d_2 - 2}, ..., 2^{d_c - 2})$$

It can be seen that the longest dilation of the xor embedding is 50% shorter than that of the standard embedding.

D. Total Dilation

This parameter can also be computed using the links dilation spectrum. It is given by the following expression:

$$D_t = \sum_{i=0}^{x} 2^i A(2^i)$$

where $2^x = D_{max}$. Next, this expression is further developed for both the standard and the xor embeddings and for some particular toruses.

In the case of the standard embedding on rings we have that

$$D_t^{std}(d) = \sum_{i=0}^{d-1} 2^i \cdot 2^{d-1} = 2^{d-1}(2^d - 1)$$

whereas for the xor embedding on rings the total dilation is

$$D_t^{xor}(d) = \sum_{i=0}^{d-3} 2^i \cdot 2^{d-1} + 2^{d-2} \cdot 2^d = 2^{d-1}(3 \cdot 2^{d-2} - 1)$$

In the case of a squared c-dimensional torus we have that

$$D_t^{std}(d/c, \ldots, d/c) = \sum_{i=0}^{d/c-1} 2^i \cdot c \cdot 2^{d-1}$$
$$= c \cdot 2^{d-1}(2^{d/c} - 1)$$
$$D_t^{xor}(d/c, \ldots, d/c) = \sum_{i=0}^{d/c-3} 2^i \cdot c \cdot 2^{d-1} + 2^{d/c-2} \cdot c \cdot 2^d$$
$$= c \cdot 2^{d-1}(3 \cdot 2^{d/c-2} - 1)$$

Notice that the total dilation of the standard embedding is about 33% higher than that of the xor embedding in both cases.

E. Maximum and Minimum Loads

In this section, the load due to communication tasks of every node is analyzed. The objective is to determine the value for the most loaded node and the least loaded one.

E.1 Standard Embedding on Rings

Assume that a d-dimensional hypercube is to be embedded onto a one-dimensional torus with 2^d nodes. Let $L_i^{std}(n)$ be the load of node n due to the links whose dilation is 2^i. We have that

$$L_i^{std}(n) = \begin{cases} n \bmod 2^{i+1}; & 0 \le n \bmod 2^{i+1} < 2^i \\ 2^{i+1} - 1 - n \bmod 2^{i+1}; & 2^i \le n \bmod 2^{i+1} < 2^{i+1} \end{cases}$$

Notice that $L_i^{std}(n)$ is a periodic function with period 2^{i+1}, and it is defined in the interval $n \in [0, 2^d - 1]$. Fig. 7 illustrates an example when $d = 4$. The figure shows the load of every node due to links whose dilation is 2^2.

Let $L_{i,i-1}^{std}(n)$ be the load of node n due to links whose dilation is either 2^i or 2^{i-1}, that is,

$$L_{i,i-1}^{std}(n) = L_i^{std}(n) + L_{i-1}^{std}(n)$$

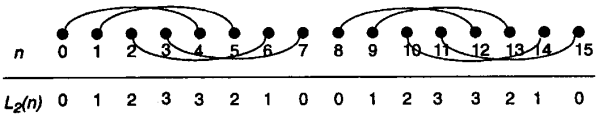


Fig. 7. Load of each node due to links with dilation equal to four for the standard embedding of a 4-cube onto a ring.

It can be shown that

$$L_{i,i-1}^{std}(n) = \begin{cases} 2(n \bmod 2^i); & n \bmod 2^{i+1} < 2^{i-1} \\ (2^i - 1); & 2^{i-1} \le n \bmod 2^{i+1} < 2^i + 2^{i-1} \\ 2(2^i - 1 - n \bmod 2^i); & n \bmod 2^{i+1} \ge 2^i + 2^{i-1} \end{cases}$$

Again $L_{i,i-1}^{std}(n)$ is a periodic function with period 2^{i+1}, and it is defined in the interval $n \in [0, 2^d - 1]$. Fig. 8 shows graphically how these expressions were obtained.

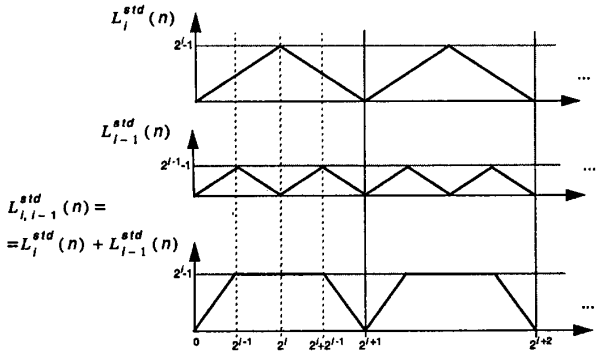


Fig. 8. Computing $L_{i,i-1}^{std}(n)$ from $L_i^{std}(n)$ and $L_i^{std}(n)$.

Now, the total load of a given node, which is denoted by $L_t^{std}(d, n)$, can be computed. If d is even then

$$L_t^{std}(d, n) = L_{d-1,d-2}^{std}(n) + L_{d-3,d-4}^{std}(n) + \ldots + L_{2,1}^{std}(n) + L_0^{std}(n)$$

and if d is odd we have that

$$L_t^{std}(d, n) = L_{d-1,d-2}^{std}(n) + L_{d-3,d-4}^{std}(n) + \ldots + L_{1,0}^{std}(n)$$

and obviously,

$$L_{max}^{std}(d) = max\left[L_t^{std}(d, n)\right]$$

Due to the fact that the period of $L_{i,i-1}^{std}(n)$ is four times the period of $L_{i-2,i-3}^{std}(n)$, there are always two periods of $L_{i-2,i-3}^{std}(n)$ where $L_{i,i-1}^{std}(n)$ is maximum for every n inside these two periods. In consequence, there is always at least one node n such that both $L_{i,i-1}^{std}(n)$ and $L_{i-2,i-3}^{std}(n)$ get its maximum value for this node (see Fig. 9). Therefore, if d is even then

$$L_{max}^{std}(d) = max\left(L_{d-1,d-2}^{std}(n)\right) + max\left(L_{d-3,d-4}^{std}(n)\right) + \ldots + max\left(L_{2,1}^{std}(n)\right)$$

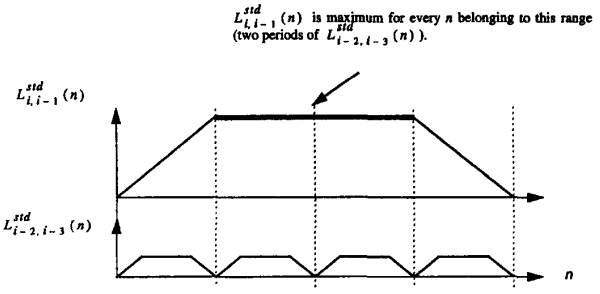


Fig. 9. This figure illustrates that:
$$max\left(L^{std}_{i,i-1}(n)+L^{std}_{i-2,i-3}(n)\right)=max\left(L^{std}_{i,i-1}(n)\right)+max\left(L^{std}_{i-2,i-3}(n)\right).$$

and if d is odd

$$L^{std}_{max}(d)=max\left(L^{std}_{d-1,d-2}(n)\right)+max\left(L^{std}_{d-3,d-4}(n)\right)+\ldots+max\left(L^{std}_{1,0}(n)\right)$$

Since

$$max\left(L^{std}_{i,i-1}(n)\right)=2^i-1$$

the maximum load of the standard embedding on rings is given by the following expression:

$$L^{std}_{max}(d)=\sum_{i=1}^{\lfloor d/2\rfloor}\left(2^{d-2i+1}-1\right)=\frac{2^{d+1}-2^{d-2\cdot\lfloor d/2\rfloor+1}}{3}-\lfloor d/2\rfloor$$

Regarding the minimum load, it can be seen that nodes 0 and 2^d-1 have a null load; then, $L^{std}_{min}(d)=0$.

E.2 Xor Embeddings on Rings

Notice that the load of a node due to links whose dilation is less than 2^{d-2} is the same for both the standard and the xor embedding, that is

$$L^{xor}_i(n)=L^{std}_i(n)\quad;\quad 0\le i<d-2$$

In consequence

$$L^{xor}_t(d,n)=L^{xor}_{d-2}(n)+L^{std}_t(d-2,n)$$

The load due to links whose dilation is $2^{d-2}(L^{xor}_{d-2}(n))$ is equal to $2^{d-2}-1$. This is illustrated in Fig. 10 by means of a particular example. Since this is a constant function, we can conclude that

$$L^{xor}_{max}(d)=\left(2^{d-2}-1\right)+L^{std}_{max}(d-2)$$

which results in

$$L^{xor}_{max}=\left(2^{d-2}-1\right)+\sum_{i=1}^{\lfloor(d-2)/2\rfloor}\left(2^{d-2i-1}-1\right)$$

$$=\left(2^{d-2}-1\right)+\frac{2^{d-1}-2^{d-2\cdot\lfloor(d-2)/2\rfloor-1}}{3}-\lfloor(d-2)/2\rfloor$$

Regarding the minimum load, we have that

$$L^{xor}_{min}(d)=\left(2^{d-2}-1\right)+L^{std}_{min}(d-2)=\left(2^{d-2}-1\right)$$

since $L^{std}_{min}(d-2)=0$.

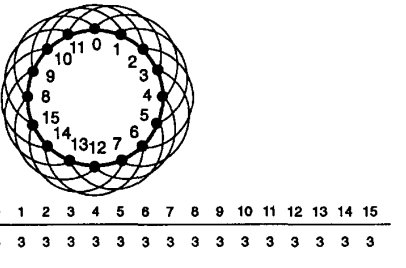


Fig. 10. Load of each node due to links with dilation equal to four for the xor embedding of a 4-cube onto a ring.

E.3 General Case

Since both the standard and xor embeddings of a hypercube onto a c-dimensional torus can be regarded as several embeddings of smaller hypercubes onto rings, there is always at least one node for which the load is maximum in all the dimensions of the torus, and at least one other node for which the load is minimum in all the dimensions. Then, it follows that

$$L_{max}(d_1,d_2,\ldots,d_c)=L_{max}(d_1)+L_{max}(d_2)+\ldots+L_{max}(d_c)$$

$$L_{max}(d_1,d_2,\ldots,d_c)=L_{min}(d_1)+L_{min}(d_2)+\ldots+L_{min}(d_c)$$

Table I compares the maximum and minimum load of both embedding onto different toruses. We can conclude that the xor embeddings has a higher minimum load but a lower maximum load. That is, the load of the nodes with communication tasks is more evenly distributed, which is a desirable property.

TABLE I
MAXIMUM AND MINIMUM LOAD FOR BOTH THE STANDARD AND THE XOR EMBEDDINGS

Size of the torus $2^m \times 2^k$	$L^{std}_{min}(m,k)$	$L^{std}_{max}(m,k)$	$L^{xor}_{min}(m,k)$	$L^{xor}_{max}(m,k)$
1 x 8	0	3	1	1
2 x 4	0	1	0	0
1 x 16	0	8	3	4
2 x 8	0	3	1	1
4 x 4	0	2	0	0
8 x 8	0	6	2	2
16 x 16	0	16	6	8
16 x 32	0	26	10	14
32 x 32	0	36	14	20
32 x 64	0	57	22	33

F. Average Load

Taking into account that a link with dilation D results in a unitary additional load to $D-1$ nodes, the average load of nodes due to communication tasks can be computed from the links dilation spectrum using the following expression for both embeddings:

$$L_{ave}(d_1,d_2,\ldots,d_c)=\sum_{i=0}^{x}(2^i-1)\cdot\left(A_{d1,d2,\ldots,dc}(2^i)\right)/2^d$$

where $2^x=D_{\max}$.

For the particular case of a ring, the average load is

$$L^{std}_{ave}(d)=\sum_{i=1}^{d-1}(2^i-1)/2=(2^d-d-1)/2$$

$$L^{xor}_{ave}(d)=\sum_{i=1}^{d-3}(2^i-1)/2+(2^{d-2}-1)=\left(3\cdot 2^{d-2}-d-1\right)/2$$

For a c-dimensional squared torus the average load is

$$L_{ave}^{std}(d_1, d_2, \ldots, d_c) = c \sum_{i=1}^{d/c-1} (2^i - 1)/2 = c\left(2^{d/c} - d/c - 1\right)/2$$

$$L_{ave}^{xor}(d_1, d_2, \ldots, d_c) = c \sum_{i=1}^{d/c-3} (2^i - 1)/2 + c\left(2^{d/c-2} - 1\right)$$

$$= c\left(3 \cdot 2^{d/c-2} - d/c - 1\right)/2$$

In both cases, the average load of the standard embedding is about 33% higher than that of the xor embedding. The difference is even higher for small hypercubes. We can then conclude that for the execution of any parallel algorithm with a hypercube communication topology the xor embedding will result in a quite less number of communication conflicts. Notice that in the case of the CC d-cube algorithms analyzed in this paper, due to their particular structure, conflicts never occur for the two embeddings.

V. Proof of Optimality of f_{xor} for Rings

The average distance as defined in Section IV.A, will be used as the main criterion to measure the goodness of any embedding with constant distances, since minimizing the average distance implies minimizing the execution time of CC d-cube algorithms. In this section, it is proven that the xor embedding has the minimum average distance for embeddings with constant distances of hypercubes onto rings.

To show that the xor embedding is optimal for rings, we will prove that the average distance of any embedding with constant distances is higher than or equal to the average distance of the f_{xor} embedding. This is stated by Theorem 10. Before this theorem, several lemmas and corollaries that are needed to prove that result are presented. First, a lower bound for the sum of any set of $d - 1$ distances corresponding to any embedding with constant distances is found. Then, a lower bound for the highest distance of the embedding is computed. Both together give a lower bound for the average distance of any embedding with constant distances. This lower bound is the average distance of the f_{xor} embedding, which proves its optimality.

Definition. *Given any node of a hypercube, we define $N_D(n)$, where D is any subset of dimensions of the hypercube, as the node that is reached by starting at node n and moving through every dimension in D, one after another, using each dimension exactly once (as we know, the order in which the dimensions are used does not matter, the result will be the same). For instance, if $D = \{1, 3\}$, then $N_D(n) = N_3(N_1(n)) = N_1(N_3(n))$.*

In the following, $N_i(N_j(n))$ will be written as $N_iN_j(n)$. The parenthesis are removed for the sake of clarity, but the meaning referring the order in which dimensions are used is preserved. That is, $N_iN_j(n)$ means that we move from node n first using dimension j and then dimension i.

The first lemma of this section proves that the sum of any subset of $d - 1$ distances corresponding to $d - 1$ dimensions must be at least $2^{d-1} - 1$.

Lemma 1. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring. Let V be any subset with $d - 1$ of the dimensions of the d-cube, that is, V contains all the dimensions of the d-cube except one. Then,*

$$\sum_{\forall i \in V} D_i \geq 2^{d-1} - 1$$

Proof. Let $H(d, n, V)$ be the subset of nodes of a d-cube that consists of nodes n and $N_W(n)$ for every $W \subseteq V$. Obviously, the number of elements in $H(d, n, V)$ is two to the power of the number of elements in V. In particular, if V has $d - 1$ elements, then $H(d, n, V)$ consists of 2^{d-1} elements. Given any set of 2^{d-1} nodes of a ring, there will always be two nodes in this set whose distance is at least $2^{d-1} - 1$. Since it is possible to go from any node in $H(d, n, V)$ to any other node in the same set, using each dimension in V at most once, the distances corresponding to the dimensions in V must add up to at least $2^{d-1} - 1$. □

The next lemma states that if two distances are equal when embedding a d-cube onto a ring then these distances must be equal to $2^{d-2} + k \cdot 2^{d-1}$ for some integer $k \geq 0$.

Lemma 2. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. If $D_i = D_j = K (i \neq j)$ then $K \equiv_{d-1} 2^{d-2}$ (in the following, $x \equiv_n y$ means that $x \bmod 2^n = y \bmod 2^n$; if $n = d$ we will just write $x \equiv y$).*

Proof. Suppose the nodes of the ring are labeled clockwise from 0 to $2^d - 1$. Let us take any node n of the hypercube and let $x = f_d(n)$. Suppose that $D_i = D_j = K (i \neq j)$. Then, $y = f_d(N_i(n))$ is equal to either $(x + K) \bmod 2^d$ or $(x - K) \bmod 2^d$. For short we will write $f_d(N_i(n)) = (x \pm K) \bmod 2^d$. We have also that $z = f_d(N_j(n)) = (x \pm K) \bmod 2^d$. Since $N_j(n) \neq N_i(n)$, the only possible solution is either $y = (x + K) \bmod 2^d$ and $z = (x - K) \bmod 2^d$ or $y = (x - K) \bmod 2^d$ and $z = (x + K) \bmod 2^d$. Since both situations are symmetrical, let us suppose the first one holds without loss of generality.

We have also that $N_jN_i(n)$ is the same as $N_iN_j(n)$. Let $w = f_d(N_jN_i(n))$. Then $w = (y + K) \bmod 2^d$ (it cannot be equal to $(y - K) \bmod 2^d$ since $(y - K) \bmod 2^d = x$ but x and w must be different). Since w is also equal to $f_d(N_iN_j(n))$, $w = (z - K) \bmod 2^d$. Therefore, $y + K \equiv z - K$, that is, $x + 2K \equiv x - 2K$. This means that $4K \equiv 0$ which implies that $K \equiv_{d-2} 0$. Then $K = k \cdot 2^{d-2}$ for some integer $k > 0$ (distances must be positive integers). However, K cannot be a multiple of 2^{d-1} because this would imply that $y = z$. In consequence, $K = 2^{d-2} + k \cdot 2^{d-1}$, that is, $K \equiv_{d-1} 2^{d-2}$. □

Definition. *Given two nodes x and y of a ring we say that y is clockwise in relation to x if the shortest path from x to y is clockwise. Otherwise we say that y is counterclockwise in relation to x. Obviously, if y is clockwise in relation to x, then x is counterclockwise in relation to y.*

Definition. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring. Let us define $S = \{i \mid 0 \leq i < d \text{ and } D_i < 2^{d-2}\}$, that is, S (it stands for Short) is the set of dimensions whose corresponding dis-*

tances are less than 2^{d-2} when a hypercube is embedded onto a ring. Given any node n of the hypercube, we define $C(n) = \{i \mid i \in S$ *and* $f_d(N_i(n))$ *is clockwise in relation to* $f_d(n)\}$ *and* $\overline{C}(n) = \{i \mid i \in S$ *and* $f_d(N_i(n))$ *is counterclockwise in relation to* $f_d(n)\}$. *Obviously,* $C(n) \cup \overline{C}(n) = S$.

Given any node, its neighbor in a given dimension of the hypercube is at a fixed distance in the ring, but it can be clockwise or counterclockwise. The next two lemmas prove that, if we take into account only those dimensions of the hypercube such that their corresponding D_i are less than 2^{d-2}, there is always a node which has all its neighbors in those dimensions clockwise in the ring (there is another node with all the neighbors in those dimensions being counterclockwise).

LEMMA 3. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. Let n be any node of the hypercube. Then, for any $j \in \overline{C}(n)$, $C(n) \cup \{j\} \subseteq C(N_j(n))$.*

PROOF. It is obvious that $j \in C(N_j(n))$ because $N_jN_j(n) = n$; so, if $N_j(n)$ is counterclockwise in relation to n then, $N_jN_j(n) = n$ is clockwise in relation to $N_j(n)$.

It only remains to be proved that for every $k \in C(n)$, $k \in C(N_j(n))$. Let us suppose that there is a k such that $k \in C(n)$, $k \notin C(N_j(n))$. Assume that the nodes of the ring are labeled clockwise from 0 to $2^d - 1$ and let $x = f_d(n)$. Since $N_j(n)$ is counterclockwise in relation to n, then $f_d(N_j(n)) = (x - D_j) \bmod 2^d$. By hypothesis $N_kN_j(n)$ is counterclockwise in relation to $N_j(n)$, so $f_d(N_kN_j(n)) = (x - D_j - D_k) \bmod 2^d$. Since $N_k(n)$ is clockwise in relation to n, then $f_d(N_k(n)) = (x + D_k) \bmod 2^d$ and $f_d(N_jN_k(n)) = (x + D_k \pm D_j) \bmod 2^d$. Because $N_kN_j(n)$ and $N_jN_k(n)$ are the same node, $x - D_j - D_k \equiv x + D_k \pm D_j$. This implies that either $D_j + D_k \equiv_{d-1} 0$ or $D_k \equiv_{d-1} 0$; but none of these can hold since $0 < D_j, D_k < 2^{d-2}$ ($j, k \in S$). So, the hypothesis was wrong and then $k \in C(N_j(n))$. □

LEMMA 4. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. Then, there is a node n of the hypercube such that $C(n) = S$, that is, $\overline{C}(n) = \emptyset$.*

PROOF. Lemma 3 gives us an algorithm to find this node n. We can start from any node m of the hypercube. If $\overline{C}(m) = \emptyset$, then $n = m$; if not, take any $i \in \overline{C}(m)$ and move to $N_i(m)$. Lemma 3 states that the number of elements in $\overline{C}(N_i(m))$ is strictly less than the number of elements in $\overline{C}(m)$. Repeating this step we will finally find a node n such that $\overline{C}(n) = \emptyset$, that is, $C(n) = S$. □

From now on we will refer to the node designated by Lemma 4 as node c of the hypercube. The nodes of the ring can be labeled in the most convenient way for us. From now on, the node $f_d(c)$ will be labeled as node 0, and the rest of the nodes of the ring will be labeled clockwise from 0 to $2^d - 1$. By the above lemma, $f_d(N_i(c)) = D_i$ for any $i \in S$. The next lemma states that for any $i \in S$, the neighbors of $N_i(c)$ in every dimension $j \in S - \{i\}$ are clockwise. Obviously, the neighbor of $N_i(c)$ in dimension i is counterclockwise, since it is c.

LEMMA 5. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. Then, for any $i \in S$, $C(N_i(c)) = S - \{i\}$. This is equivalent to say that $f_d(N_jN_i(c)) = D_i + D_j$ for any $i, j \in S$, $i \neq j$.*

PROOF. Suppose there is some $j \in S - \{i\}$ such that $j \in \overline{C}(N_i(c))$. That means that $x = f_d(N_jN_i(c)) = (D_i - D_j) \bmod 2^d$. Since x is also equal to $f_d(N_iN_j(c)) = (D_j \pm D_i) \bmod 2^d$, this implies that either

- $D_i \equiv_{d-1} D_j$, which is not possible because $D_i \neq D_j$ (by Lemma 2) and $D_i, D_j < 2^{d-2}$, or
- $D_j \equiv_{d-1} 0$, which cannot hold since $0 < D_j < 2^{d-2}$.

In consequence, for every $j \in S - \{i\}$, $j \in C(N_i(c))$ and then $C(N_i(c)) = S - \{i\}$. □

Next, it is proven that given any subset of dimensions $W \subseteq S$, the neighbors of $N_W(c)$ in every dimension $i \in S - W$ are clockwise.

LEMMA 6. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. Then, for any $W \subseteq S$, $C(N_W(c)) \subseteq S - W$.*

PROOF. The lemma will be proved by induction over the number of elements in W.

If W has just one element the lemma holds (it has been proved in Lemma 5, which can be seen as a particular case of Lemma 6).

Assume that the lemma holds for any set with less than N elements and let us suppose that it does not hold for a set W with N elements. This means that there is a dimension $k \in S - W$ such that $f_d(N_k(N_W(c)))$ is counterclockwise in relation to $f_d(N_W(c))$.

The fact that the lemma holds for any subset with less than N elements implies that

$$f_d(N_D(c)) = \left(\sum_{\forall i \in D} D_i\right) \bmod 2^d$$

for any subset $D \subseteq S$ with N elements. Therefore, since W has N elements

$$f_d(N_W(c)) = \left(\sum_{\forall i \in W} D_i\right) \bmod 2^d = r$$

Let V be equal to the set W after taking any element of it and being replaced by k, that is, let $i \in W$ be any element of W, then $V = (W - \{i\}) \cup \{k\}$. Since V has also N elements

$$f_d(N_V(c)) = \left(\sum_{\forall i \in V} D_i\right) \bmod 2^d = s$$

We know that $N_k(N_W(c)) = N_i(N_V(c))$. Then, $f_d(N_k(N_W(c))) = f_d(N_i(N_V(c)))$. By hypothesis, since N_k is counterclockwise in relation to $N_W(c)$, then the left part of the equality must be equal to $(r - D_k) \bmod 2^d$. The right part is equal to $s \pm D_i \bmod 2^d$. In consequence, $r - D_k \equiv s \pm D_i$. Substituting r and s by their corresponding expressions and simplifying we obtain that $D_i - D_k \equiv D_k \pm D_i$. This equation can be satisfied just in two ways: either $D_k \equiv_{d-1} 0$ or $D_i \equiv_{d-1} D_k$. None of them can hold since $0 < D_i, D_k < 2^{d-2}$, and as Lemma 2 states, D_i and D_j cannot be equal.

We then conclude that for every $k \in S - W$, $f_d(N_k(N_W(c)))$ is clockwise in relation to $f_d(N_W(c))$ and therefore, $C(N_W(c)) \subseteq S - W$. □

COROLLARY 7. *If we start from node c and we want to move to node $N_W(c)$ for any $W \subseteq S$, using each dimension in W exactly once, any time we move through a dimension in W we will be moving clockwise in the ring, no matter in which order we use the dimensions in W, that is,*

$$f_d(N_W(c)) = \left(\sum_{\forall i \in W} D_i \right) \bmod 2^d$$

PROOF. It is a direct implication of Lemmas 4 and 6. The former states that node c has all its neighbors in S clockwise, so the first hop must be necessarily clockwise. Then Lemma 6 says that if we have moved from node c to a node r using a subset W of dimensions of S, making use of each dimension just once, all the neighbors of node r in any dimension not used yet (i.e., belonging to $S - W$) are clockwise, so the next hop must also be clockwise. □

Next, it is proven that, when embedding a hypercube onto a ring with constant distances, if all distances are lower than 2^{d-2}, the sum of all distances must be at least $2^d - 1$.

LEMMA 8. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes such that every $D_i < 2^{d-2}$. If such embedding exists, then*

$$\sum_{i=0}^{d-1} D_i \geq 2^d - 1$$

PROOF. Since all distances are less than 2^{d-2}, S (set of dimensions whose distance is less than 2^{d-2}) consists of all dimensions of the d-cube. Then, Lemma 4 states that there must be a node c such that all its neighbors are clockwise. In addition, Corollary 7 says that it is possible to go from node c to any node of the hypercube given at most d hopes (each one corresponding to a different dimension) and going always clockwise. In particular, we can go from node c to the node just next to it counterclockwise. Moving always clockwise, the distance between these two nodes is $2^d - 1$, so the sum of all distances must be at least equal to this amount. □

COROLLARY 9. *An embedding with constant distances such that all distances are less than 2^{d-2} is not optimal, if it exists. In this context, to be optimal means that it has the lowest average distance for embeddings with constant distances. In other words, the optimal embedding must have at least one distance greater than or equal to 2^{d-2}.*

PROOF. Lemma 8 states that, if such embedding exists, then the sum of its distances is at least $2^d - 1$. To prove this corollary it suffices to find an embedding whose distances add up to less than this amount. Such embedding can be the xor embedding, the one proposed in Section III. □

We are now ready to prove that the xor embedding is optimal. This is proved in the next theorem and its corollary.

THEOREM 10. *Let f_d be an embedding with constant distances $(D_i, i \in [0, d-1])$ of a d-cube onto a ring with 2^d nodes. Then the average distance of f_d is at least $(3 \cdot 2^{d-2} - 1)/d$.*

PROOF. The proof is based on Lemma 1 and Corollary 9. The lemma says that the sum of any subset of $d - 1$ distances is at least $2^{d-1} - 1$, so in particular, the sum of the $d - 1$ lowest distances of the embedding must be at least equal to this amount. Corollary 9 states that the optimal embedding has at least one distance that is higher than or equal to 2^{d-2}, so in particular, the highest distance of the embedding must be higher than or equal to 2^{d-2}. Both together imply that

$$\left(\sum_{i=0}^{d-1} D_i \geq 2^{d-1} - 1 + 2^{d-2} = 3 \cdot 2^{d-2} - 1 \right)$$

$$\Leftrightarrow \left(\text{average distance } (f_d) \geq \frac{3 \cdot 2^{d-2} - 1}{d} \right) \quad \square$$

COROLLARY 11. *The* xor *embedding of a* d*-cube onto a ring proposed in Section III is optimal in the sense that it has the lowest average distance for embeddings with constant distances.*

PROOF. The average distance of the xor embedding is equal to the lower bound introduced in Theorem 10. □

VI. CONCLUSIONS

This paper focuses on the execution of algorithms with a hypercube communication topology onto multicomputers with a torus interconnection topology. The problem is tackled by means of graph embeddings. An embedding of hypercubes onto toruses of any arbitrary dimension has been presented. This embedding, called *xor embedding*, belongs to a class of embeddings whose distinguishing property is that all the links of the same dimension of the hypercube have the same dilation on the torus. This class of embeddings are called *embeddings with constant distances*.

Many parallel algorithms with hypercube topology have the property that all the processes perform the same activity with different data. This activity consists of a number of stages (usually as many as number of dimensions of the hypercube) and each stage is composed of a computing phase followed by a communication phase in which data is interchanged with one of its neighbors. This structure is found in parallel algorithms for FFT and sorting among others. For this type of algorithms, called CC d-cube algorithms, constant distances may be desirable because they imply that the communication phase has the same duration for every process, avoiding waiting intervals which can degrade performance.

The xor embedding has been compared to the standard embedding using a set of different performance metrics (execution time of a CC d-cube algorithm, links dilation spectrum, longest dilation, total dilation, maximum and minimum load, and average load). For all of them the performance of the xor embedding is significantly better than that of the standard embedding.

For CC d-cube algorithms, the embedding with constant distances that results in the shortest execution time is that

whose average distance is minimum. It has been proven that the average distance of the xor embedding is minimum for rings (one-dimensional torus), and therefore, it maximizes the performance of the multicomputer for those algorithms.

Another important property of the xor embedding is the simplicity of the function which determines the location where a node of the d-cube is found in the target multicomputer.

We are currently working on the generalization of this work in two different directions. First, we are looking at more general hypercube algorithms. Second, we are considering more general torus multicomputers in which the number of processing elements is not necessarily a power of two.

ACKNOWLEDGMENTS

This work has been supported by the Ministry of Education and Science of Spain (CICYT TIC-92/880 and TIC-91/1036) and the European Center for Parallelism in Barcelona (CEPBA).

REFERENCES

[1] W.C. Athas and C.L. Seitz, "Multicomputers: Message-passing concurrent computers," *Computer*, vol. 21, no. 8, pp. 9-24, Aug. 1988.

[2] C.Y.R. Chen and Y.-C. Chung, "Embedding networks with ring connections in hypercube machines," *Proc. Int'l Conf. Parallel Processing*, vol. 3, pp. 327-334, 1990.

[3] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*. Englewood Cliffs, N.J.: Prentice Hall, 1988.

[4] A. González and M. Valero-García, "The xor embedding: Embedding hypercubes onto rings and toruses," L. Dadda and B. Wah, eds., *Proc. Int'l Conf. Application-Specific Array Processors*, IEEE CS Press, 1993, pp. 15-28.

[5] L.H. Harper, "Optimal assignments of numbers to vertices," *J. Soc. Industrial Applied Math.*, vol. 12, pp. 131-135, 1966.

[6] L.H. Harper, "Optimal numbering and isoperimetric problems on graphs," *J. Combinatorial Theory*, vol. 1, pp. 385-396, 1966.

[7] C.-T. Ho and S.L. Johnsson, "Embedding three-dimensional meshes in boolean cubes by graph decomposition," *Proc. Int'l Conf. Parallel Processing*, vol. 3, pp. 319-326, 1990.

[8] T.H. Lai and A.P. Sprague, "Placement of the processors of a hypercube," *IEEE Trans. Computers*, vol. 40, no. 6, pp. 714-722, 1991.

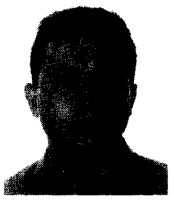
[9] E. Ma and L. Tao, "Embeddings among meshes and tori," *J. Parallel and Distributed Computing*, vol. 18, pp. 44-55, 1993.

[10] S. Matic, "Emulation of hypercube architecture on nearest-neighbor mesh-connected processing elements," *IEEE Trans. Computers*, vol. 39, no. 5, pp. 698-700, May 1990.

[11] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, Calif.: Morgan Kaufmann, 1992.

[12] S.W. Turner, L.M. Ni, and B.H.C. Cheng, "Contention-free 2D-mesh cluster allocation in hypercubes," *Proc. Int'l Conf. Parallel Processing*, vol. 2, pp. 125-129, 1993.

[13] A. Varma and C.S. Raghavendra, ed., "Interconnection networks for multiprocessors and multicomputers: Theory and practice," IEEE CS Press, 1993.



Antonio González received the computer science degree in 1986 and the PhD degree in computer science in 1989, both from the Polytechnic University of Catalonia (UPC), Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at the Polytechnic University of Catalonia. His teaching and research interests center on computer architecture (in particular, memory organization), parallel architectures and algorithms, and logic programming. Dr. Gonzalez is a member of the Board of Directors of Euromicro and a member of the IEEE and ACM.

Miguel Valero-García received the computer science degree in 1986 and the PhD degree in computer science in 1989, both from the Polytechnic University of Catlonia (UPC), Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at the Polytechnic University of Catalonia. His primary research interest is in parallel architectures and algorithms. Dr. Valero-García is a member of the IEEE and ACM and a researcher of the European Center for Parallelism of Barcelona (CEPA).

Luis Díaz de Cerio received the higher diploma degree from the Polytechnic University of Catalonia (UPC), Barcelona, Spain, in electrical engineering in 1993. He is currently an assistant professor in the Computer Architecture Department at the Polytechnic University of Catalonia, where he is pursuing his PhD. His research interests focus on parallel architectures and algorithms.