The Xor Embedding: An Embedding of Hypercubes onto Rings and Toruses

Antonio González and Miguel Valero-García

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}@ac.upc.es

Abstract

Many parallel algorithms use hypercubes as the communication topology among processes, which make them suitable to be executed on a hypercube multicomputer. In this way 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 multicomputer is constrained by the fact that the interconnection cost per node increases with the total number of nodes. From the point of view of scalability, meshes and toruses are a more interesting class of interconnection topologies. In this paper we propose an embedding of hypercubes onto toruses of any given dimension, including one-dimensional toruses which are also called rings. We also prove that this embedding is optimal in the sense that it minimizes the execution time on a ring of a class of parallel algorithms frequently found in real applications, such as FFT and some class of sorting algorithms.

1. Introduction

Many parallel algorithms use hypercubes as the communication topology among processes. Some examples include parallel algorithms for FFT, sorts, etc. [2]. We will call these algorithms *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 we consider the problem of executing d-cube algorithms on *multicomputers* [1]. A multicomputer is a distributed memory multiprocessor in which the *nodes* (processor + local memory) are interconnected through point to point links.

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). We say, in this case, 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 when it is executed on a hypercube multicomputer is kept minimum.

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. As an example, each node of a two-dimensional torus multicomputer is directly connected to 4 nodes, it does not matter the number of nodes of the multicomputer.

When a d-cube algorithm is executed on a multicomputer with a topology other than hypercube it is not possible, in general, to allocate processes to nodes in such a way that every process has its d neighbors at distance 1 in the multicomputer. As an example, in a two-dimensional torus multicomputer, every process has at most 4 of its d neighbors at distance 1. It must have at least $d-4$ neighbors at a distance greater than 1. A message to any of these "far" neighbors must be routed through the point to point links and nodes which are found along the path to the destination node. A good mapping of a d-cube algorithm onto a multicomputer will try to keep the neighbor processes as close as possible in the multicomputer, minimizing in this way the cost of the communication component of the d-cube algorithm when it is executed on the multicomputer.

In this paper we propose an embedding of d-cube algorithms onto torus multicomputers of any arbitrary dimension. This embedding has the following properties:

a) For each dimension of the d-cube, every process has its corresponding neighbor at the same distance (although these distances may be different for different dimensions).

b) The average distance of the d neighbors of any process is minimum, subject to (a).

As it will be shown later in this paper, property (a) is desirable for some kind of d-cube algorithms like FFT, since it guarantees that any process will never have to wait for a neighbor to finish its work before starting the interchange of data with it. Property (b) implies a minimum cost in the communication component of the d-cube algorithm, and therefore, a minimum execution time. We present a proof of this property for rings (one-dimensional toruses). The generalization of the proof for an arbitrary dimension is still under development.

This paper is organized as follows. In section 2 we introduce some notation and describe more precisely the contribution of this paper as well as some related work. Sections 3 presents the proposed embedding of d-cubes onto rings. In senction 4, the embedding is generalized to toruses of any arbitrary dimension. In section 5 we prove that the proposed embedding for rings is optimal according to the criterion introduced in section 2. Finally, we present some concluding remarks.

2. Preliminaries and related work

2.1. 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 i-th bit then m is the neighbor of n in dimension i, and n is the neighbor of m in the same dimension. Then, we write:

$$m = N_i(n)$$
$$n = N_i(m)$$

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 and a communication phase. In each stage, every process uses a different dimension to exchange information with one of its neighbors.

We assume that the duration of the compute phase and the amount of information to be exchanged is 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.

Any parallel algorithm can be modelled as a graph. 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 modelled 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.

Multicomputers can be classified according to their interconnection topology. In this paper, we are interested in 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***.

Figure 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 which 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 a bijective function f from the vertices of G to the vertices of H. We assume that G and H have the same number of vertices.

The problem of executing a CC d-cube algorithm on a multicomputer can be modelled 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

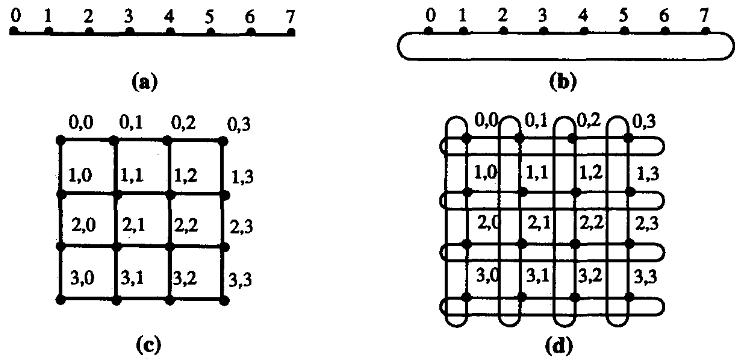


Figure 1: Different types of multicomputers: a) line, b) ring, c) (4,4) mesh and d) (4,4) torus.The picture also shows how the nodes are labeled.

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 $f(N_i(n))$.

We denote by T_a the duration of the arithmetic computation phase in every stage of the CC d-cube algorithm, when it is executed on the target multicomputer. We denote by T_c 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, as defined by 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:

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

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

$$T_{-1}(n) = 0$$

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 $D_i(n)T_c$. 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). In the following, we call those idle periods as *waiting periods*.

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

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

2.2. Contributions

In this paper, we are interested in 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 periods are avoided since neighbor processes arrive at the same time to the point at which they have to communicate.

The time to execute a CC d-cube algorithm onto a multicomputer, as defined by an embedding with constant distances 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:

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

Therefore, the embedding with constant distances which minimizes T_f is that whose average distance D_a is minimum. An embedding with such property is said to be optimal.

As it was mentioned in the introduction, we are interested in executing CC d-cube algorithms on scalable multicomputers. In particular, we are interested in torus multicomputers since for meshes a well-known embedding, described in the next section, is optimal for CC d-cube algorithms. In this paper we propose an embedding with constant distances of CC d-cube algorithms onto torus multicomputers of any arbitrary dimension. Moreover, we prove that the proposed embedding with constant distances is optimal for rings (one-dimensional toruses). Another additional property of the proposed embedding is its simplicity, which means a negligible cost to compute the location of any process in the multicomputer.

2.3. Related work

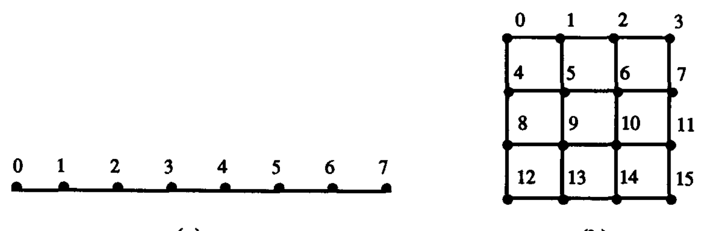
The problem of embedding d-cubes onto meshes and toruses has been previously considered by other authors. We now review some related work.

Matic presents in [7] 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 figures 1.a and 1.b). Then, the standard embedding is defined by (see figure 2.a):

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

The standard embedding of a d-cube onto a $(k_1,k_2,\dots,k_c)$ c-dimensional mesh or torus is defined as follows:

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



(a) (b)

Figure 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. Wrap-around links are not shown for clarity.

where:

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

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

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

It can be shown that the standard embedding is optimal for meshes, in the sense that it minimizes the average distance [3],[8]. However, it is not optimal for toruses, as it will be shown later in this paper.

Harper in [4] and Lai and Spague in [5] 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, we describe briefly this embedding.

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 *1*'s 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. Figure 3.a 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 [5]. Figure 3.b shows an example.

The byweight embedding minimizes the dilation of the embedding. 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 modelled 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. Therefore, during the execution of the CC d-cube algorithm, waiting periods will appear which contribute to increase the execution time. To illustrate this fact, figure 4 shows an example in which the execution times of a CC 3-cube algorithm

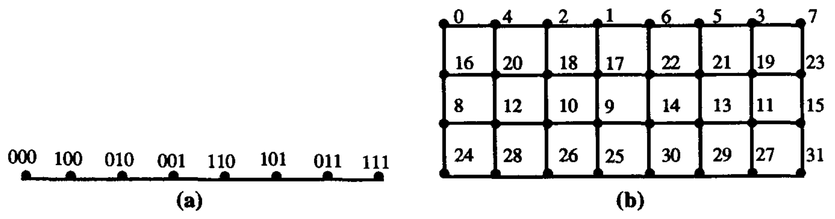


Figure 3: Byweight embeddings of: a) a 3-cube onto a line and b) a 5-cube onto a (8,4) mesh.

on a line for both the standard embedding and the byweight embedding are compared. The waiting periods which contribute to make the byweight embedding run slower than the standard embedding are also shown.

In [6], Y.W. Ma and L. Tao proposed several embeddings among toruses and meshes of different dimensions. Their proposals are based on generalizing the concepts of gray code for 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 largest distance between any two neighbors 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 for executing our target algorithm is worse than the embedding proposed in this paper.

3. An embedding with constant distances for rings

We describe now the proposed embedding with constant distances of a d-cube onto a ring with 2^d vertices (we assume $d > 1$). We call this embedding *xor embedding* and it is denoted by f_{xor}. The xor embedding is optimal, in the sense that it minimizes the average distance. The xor embedding is described next. Section 5 presents a proof of its optimality.

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-1clockwise (see figure 1.b). 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. Figure 5 shows an example for $d=4$.

The distances of the xor embedding are:

$$D_i = 2^i \qquad i \in [0, d-2]$$

$$D_{d-1} = 2^{d-2}$$

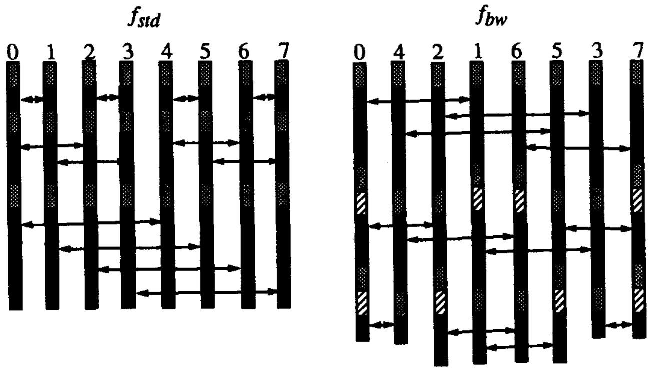
node	f_{std}			f_{bw}		
n	$D_0(n)$	$D_1(n)$	$D_2(n)$	$D_0(n)$	$D_1(n)$	$D_2(n)$
0	1	2	4	3	2	1
1	1	2	4	3	3	2
2	1	2	4	4	2	2
3	1	2	4	4	3	1
4	1	2	4	4	3	1
5	1	2	4	4	2	2
6	1	2	4	3	3	2
7	1	2	4	3	2	1

(a)

▮ Computation

▮← Communication

▨ Waiting period



(b) (c)

Figure 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.

Therefore, the average distance is:

$$D_a = \frac{2^{d-2} + \sum_{i=0}^{d-2} 2^i}{d} = \frac{2^{d-1} + 2^{d-2} - 1}{d}$$

4. An embedding with constant distances for c-dimensional toruses

Now we describe the xor embedding of a d-dimensional hypercube onto a $(2^{d1}, 2^{d2}, \ldots, 2^{dc})$ c-dimensional torus such that $d_1 + d_2 + \ldots + d_c = d$.

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

$$K_j = \sum_{i=1}^{j-1} d_i$$

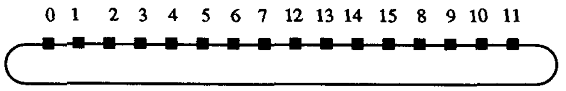


Figure 5: A xor embedding of a 4-cube onto a ring. The labels indicate which node of the d-cube is mapped onto the corresponding node of the ring.

Let 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:

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

$$m_j(d_j - 2) = XOR\,(n(K_{j+1} - 1),\ n(K_{j+1} - 2))$$

For the particular case of a $(2^{d/c}, ...,2^{d/c})$ c-dimensional torus, the distances corresponding to this embedding are:

$$D_i = 2^{d/c-2} \qquad if\ i = l\,(d/c) - 1,\ 0 < l \leq c$$

$$D_i = 2^{i \bmod d/c} \qquad otherwise$$

and the average distance is:

$$D_a = \frac{c\left(2^{d/c-2} + \sum_{i=0}^{d/c-2} 2^i\right)}{d} = \frac{c\,(2^{d/c-2} + 2^{d/c-1} - 1)}{d}$$

Figure 6 shows an example for $d=6$. 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.

5. Proof of optimality of f_{xor} for rings

Our criterion to measure the goodness of any embedding with constant distances is its average distance as defined in section 2, since minimizing the average distance implies minimizing the execution time of CC d-cube algorithms. In this section we prove 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 we present several lemmas and corollaries that are needed to prove that result. First we find a lower bound for the sum of any set of $d-1$ distances corresponding to any embedding with constant distances. Then, we find a lower bound for the highest distance of the embedding. 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 we reach if we start at node n and we move 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 $N_D(n) = N_3(N_1(n)) = N_1(N_3(n))$.

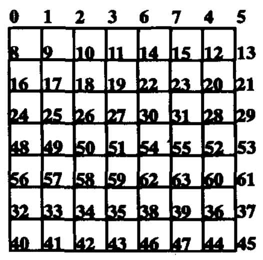


Figure 6: A xor embedding of a 6-cube onto a (8,8) torus. The wrap-around links are not shown, for clarity.

In the following, $N_i(N_j(n))$ will be written as $N_iN_j(n)$. We remove the parenthesis 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 excepting one. Then,

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

Proof: Let us define $H(d,n,V)$ as the subset of nodes of a d-cube that consists of node 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, we can always find two nodes in this set whose distance is at least $2^{d-1}-1$. Since we can 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. We 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 distances are less than 2^{d-2} when a hypercube is embedded onto a ring. Given any node n of the hypercube, we define $C(n) = \{i \mid i \in S \text{ and } f_d(N_i(n)) \text{ is clockwise in relation to } f_d(n)\}$ and $\overline{C}(n) = \{i \mid i \in S \text{ and } f_d(N_i(n)) \text{ is counterclockwise in relation to } f_d(n)\}$. 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}, we can find a node which has all its neighbors in those dimensions clockwise in the ring (we can find another node with all those neighbors 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 $j \in \overline{C}(n)$, $C(N_j(n)) \subseteq C(n) \cup \{j\}$.

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 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, our 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 we take any $i \in \overline{C}(m)$ and we 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. We can label the nodes of the ring in the most convenient way for us. From now on we will label the node $f_d(c)$ 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 $C(N_i(c)) = S - \{i\}$. ❑

We will prove now 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. ❑

We will prove now 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 we can 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 we proposed in section 3. ❑

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

$$\left(\sum_{i=0}^{d-1} D_i \geq 2^{d-1} - 1 + 2^{d-2} = 3 \cdot 2^{d-2} - 1\right) \Leftrightarrow \text{(average distance } (f_d) \geq \frac{3 \cdot 2^{d-2} - 1}{d}\text{)}$$

❑

Corollary 11: The xor embedding of a d-cube onto a ring proposed in section 3 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. ❑

6. Conclusions

We have presented an embedding of hypercubes onto toruses of any arbitrary dimension. This embedding, called *xor embedding*, belongs to a class of embeddings whose distinguishing property is that every node of the hypercube has its neighbor in a given dimension at the same distance, for each dimension of the hypercube. This class of embeddings are called *embeddings with constant distances.*

Embeddings can be applied to mapping a parallel algorithm with a hypercube topology onto a multicomputer with a more scalable topology, like a c-dimensional torus. 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 sort among others. For this type of algorithms, constant distances may be desirable because they imply that the communication phase has the same duration for every process, avoiding waiting periods which can degrade performance.

For those parallel algorithms, the embedding with constant distances that results in the lowest execution time is that whose average distance is minimum. We have proved 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. A detailed study of other properties of the xor embedding, comparing it with other embeddings can be found in [8]. An important result of this comparison refers to the average load of nodes due to communication tasks. The xor embedding has the lowest average load of any embedding with constant distances.

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 CEPBA (European Center for Paralellism in Barcelona).

Rerefences

[1] W.C. Athas and C.L. Seitz,
Multicomputers: Message-Passing Concurrent Computers,
IEEE Computer, August 1988, pp. 9-24.

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

[3] L.H. Harper,
Optimal Assignments of Numbers to Vertices,
J. Soc. Industrial Appl. Math., Vol. 12, 1964, pp. 131-135

[4] L.H. Harper,
Optimal Numbering and Isoperimetric Problems on Graphs,
J. Combinatorial Theory, Vol. 1, 1966, pp. 385-396.

[5] T.H. Lai and A.P. Sprague,
Placement of the Processors of a Hypercube,
IEEE Trans. on Computers, Vol. 40, No. 6, pp. 714-722, 1991.

[6] Y.-W. Ma and L. Tao
Embeddings among Toruses and Meshes
Int. Conf. on Parallel Processing, 1987, pp. 178-187.

[7] S. Matic,
Emulation of Hypercube Architecture on Nearest-Neighbor Mesh-Connected Processing Elements,
IEEE Trans. on Computers, Vol. 39, No. 5, May 1990, pp. 698-700.

[8] M. Valero-García, A. González and L. Díaz de Cerio,
Embedding Hypercubes onto Rings and Toruses
Research report DAC/UPC RR-93/01, Jan. 1993.