Optimal Triple Modular Redundancy Embeddings in the Hypercube

Larry Brown and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431

Abstract

To achieve reliability without sacrificing performance, the tasks of a computation are redundantly assigned to the processors of a hypercube multiprocessor. The computation is represented by a task interaction graph in which nodes represent tasks, and edge weights represent the amount of communication between tasks. To provide fault-tolerance, each node in the graph is replaced by three nodes that act together as a Triple Modular Redundancy (TMR) unit. We develop a formula to calculate the number of TMR units that can be supported in an n-dimensional hypercube, and a formula to calculate the distance between two TMR units. Then we give algorithms for TMR embeddings of weighted 1-level k-ary trees and unweighted rings in a hypercube. These algorithms minimize expansion, and are optimal in that they minimize dilation for a given expansion.

1 Introduction

The execution time of a computation can be greatly reduced if it can be divided into parallel units of execution or tasks, and each task can be scheduled to run on a separate processor. Scheduling is the process of assigning tasks to processors. In a static schedule, the number of tasks and their characteristics, such as execution time and the amount of inter-task communication, is known *a priori* [2]. If each task is assigned to a separate processor, there may be more tasks than available processors, and the amount of processor interaction is large. The goal is to construct an optimal schedule of tasks to processors, such that some function of the number of processors used, the amount of inter-processor communication, and the overall length of the computation is minimized [16]. This scheduling problem is NP-complete, so heuristic methods [5], [12] are often employed to obtain near-optimal schedules.

As the number of processors used by a computation grows, the probability of one of them failing grows. Hardware redundancy can be used in multiprocessor architectures to improve the reliability of a system. This is especially important for real-time systems where temporal based fault-tolerance techniques such as checkpointing with rollback are not applicable. A fault-tolerant schedule assigns each task to multiple processors in order to take advantage of hardware redundancy.

A computation is often represented as a *task precedence graph*. A task precedence graph is a directed acyclic graph in which each node represents a task, and the edges represent precedence constraints among the tasks. A task precedence graph is typically a single-entry-node single-exit-node connected (SEC) graph [14]. Another way to represent a computation is with a *task interaction graph*. A task interaction graph is a weighted graph in which nodes represent tasks, and edge weights represent the amount of communication between tasks. Gangadhar [7] has studied the relationship and possible transformations between these two representations. The computation must be executed on some real machine. The architecture of the machine can also be represented by a graph in which each node represents a processor, and edges represent processor interconnections. This graph is known as the *host graph*.

In order to reduce the number of processors needed to execute a computation, multiple tasks can be executed on the same processor. In order to minimize the total execution time of the computation, tasks that can execute in parallel should be assigned to different processors. To keep inter-task communication to a minimum, tasks that communicate should execute on the same processor. These conflicting goals must be balanced. A special scheduling process which maps each node of the task interaction graph to a unique node of the host graph is called *embedding*. The con-

0-8186-6322-7/94 $04.00 © 1994 IEEE

cepts of *expansion* and *dilation* describe the quality of an embedding. Expansion is a measure of processor utilization, and dilation is a measure of communication cost. An optimal embedding minimizes expansion or dilation when the other is fixed.

In this paper we consider a system of homogeneous processors with a hypercube interconnection topology. The hypercube is a well-studied structure with many desirable properties that is capable of embedding many other topologies [15]. It is known that optimally embedding an arbitrary graph into a hypercube is NP-complete problem [1]. Most of the literature therefore attempts to find optimal embeddings of special graphs such as trees [17], quadtrees [10], $2-d$ meshes [3], $3-d$ meshes [11], or pyramids [8], or heuristic algorithms for embedding when optimal algorithms cannot be found [6]. The graph being embedded is usually unweighted.

In [9], Kiskis and Shin propose using Triple Modular Redundancy (TMR) in a hypercube to provide fault-tolerance. A TMR unit consists of three processors acting together to perform one task. Each processor executes the same computation, exchanges results with the other two processors, and uses majority voting to determine the correct result (see Figure 1). Using this procedure, a TMR unit can detect and recover from a single non-Byzantine fault. Using TMR, a general purpose hypercube computer can be used to run applications requiring fault-tolerance without special hardware support. The fact that TMR is being used can be transparent to the application program, and TMR processing can be enabled or disabled for any particular execution.

Kiskis and Shin [9] prove the correctness of the TMR structure for the hypercube, show how to group processors in the hypercube into a set of independent TMR units, and show how to convert a task interaction graph into a TMR task interaction graph to tolerate single-point faults. In this paper, we extend their work to show TMR embeddings in a hypercube for two common task interaction graph topologies. The approach used in this paper differs from the traditional embedding problem in that we embed a weighted, rather than an unweighted, graph. This leads to a definition of dilation based on an average, rather than a maximum, measure. Our embeddings use the minimal dimension hypercube that can embed the task interaction graph, this minimizes expansion. Given the minimal expansion, we then optimize the embedding by minimizing dilation. Optimal TMR embeddings are given for weighted 1-level k-ary trees and unweighted rings. The problem of embedding a weighted TMR ring is also discussed.

In the next section we discuss basic notation. We then review how processors in the hypercube are grouped into TMR units, develop a formula to calculate the number of TMR units that can be supported in an n-dimensional hypercube, and develop a formula to calculate the distance between two TMR units. Then we give embeddings with examples for 1-level k-ary trees and rings, and present some conclusions.

2 Preliminaries

If a problem is presented in the form of a task precedence graph, it must first be scheduled, or transformed into a task interaction graph [7]. The task interaction graph, whether derived or given, can now be embedded into the hypercube. An n-dimensional hypercube (Q_n) [15] is a network structure consisting of 2^n processors. Each processor is distinctly addressed by an n-bit binary number from 0 to 2^n-1. Two processors are directly connected by a link if and only if their binary addresses differ by exactly one bit.

In terms of the embedding problem, the graph representing the computation that is to be embedded is called the *guest graph* $G_g =< V_g, E_g >$. The graph into which the guest graph is being embedded called the host graph, $G_h =< V_h, E_h >$. For a TMR embedding, we will group disjoint sets of three processors in the hypercube into TMR units before embedding G_g. Therefore, each node in our G_h represents three processors in the hypercube.

The embedding problem considered here is to find a function that maps nodes of G_g onto unique nodes of G_h. Two measures of the quality of an embedding are expansion and dilation. Expansion is the ratio of the number of nodes in G_h to the number of nodes in G_g:

$$e = \frac{|V_h|}{|V_g|}$$

The larger the expansion, the more unused nodes (processors) in G_h (the machine). Dilation is a measure of how much G_g must be stretched to fit G_h. Nodes that were adjacent in G_g may no longer be adjacent in G_h. Dilation is given by:

$$d = \frac{\sum_{i,j} w_{ij} d_{ij}}{\sum_{i,j} w_{ij}}$$

where w_{ij} is the weight of the edge between nodes i and j in G_g, and d_{ij} is the distance between the nodes

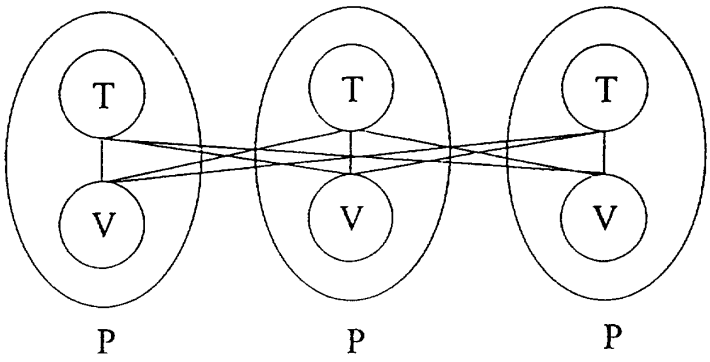


Figure 1: A TMR unit. P–processor, T–task, V–voter.

of G_h onto which nodes i and j from G_g were mapped[1]. Since the nodes of G_h represent three processors in the hypercube, the calculation of distances between nodes is not obvious, see Section 3 for a discussion. Our objective is to find an embedding with minimum d in the smallest possible hypercube.

Because processors in a TMR unit exchange results and vote, the communication costs within a TMR unit are higher than between TMR units. Therefore, the three processors making up a TMR unit should be located as close together as possible. In a hypercube, the closest three processors can get is as three nodes of a 2-dimensional cube (Q_2). Our optimal embeddings are based on these assumptions.

Table 1: TMR unit address in Q_5.

TMR unit	Address	Cube
0	$000**$	Q_3
1	$001**$	Q_3
2	$010**$	Q_3
3	$011**$	Q_3
4	$100**$	Q_3
5	$101**$	Q_3
6	$110**$	Q_3
7	$111**$	Q_3
8	$0**11$	Q_1
9	$1**11$	Q_1

3 TMR Units in the Hypercube

To make use of the available resources, as many processors as possible must be grouped into TMR units. Consider an n-dimensional hypercube Q_n. Any node can be identified by an n-bit binary address: $a_n a_{n-1} \ldots a_2 a_1$. A Q_n can be viewed as a Q_{n-2} with each node in the Q_{n-2} being a 2-cube. The addresses of these 2-cubes have the form: $a_n a_{n-1} \ldots a_4 a_3 * *$, where $*$ is the *don't care* symbol. Any two dimensions could have been replaced by $*$. Without loss of generality, we will always replace the two least significant bits of the address. The nodes of these 2-cubes with 00, 01, and 10 as their two least significant bits can be grouped into TMR units. Any three nodes of the 2-cube could have been selected without loss of generality. Nodes having addresses of the form: $a_n a_{n-1} \ldots a_4 a_3 11$ are unused. To group these unused nodes into TMR units, consider a Q_{n-4} with addresses of the form: $a_n a_{n-1} \ldots a_6 a_5 **11$. This procedure can be applied until all dimensions are exhausted. In general, for Q_{n-i} there are $n-i$ bits to the left of $**$ in the address, and any bits to the right of $**$ are 1s. Figure 2 shows the TMR units in Q_5 and the addresses of those TMR units are listed in Table 1.

The following theorem shows how to calculate the number of TMR units that can be formed in an n-dimensional hypercube.

Theorem 1 *The number of TMR units in an n-dimensional hypercube Q_n, $n > 1$, is*

$$2^{n-2\lfloor n/2 \rfloor} \sum_{k=0}^{\lfloor (n-2)/2 \rfloor} 2^{2k} = \begin{cases} \frac{2^n-1}{3} & \text{for even } n \\ \frac{2^n-2}{3} & \text{for odd } n \end{cases}$$

[1] When embedding unweighted graphs, dilation is defined to be the maximum distance between nodes of the host graph onto which adjacent nodes of the guest graph were mapped.

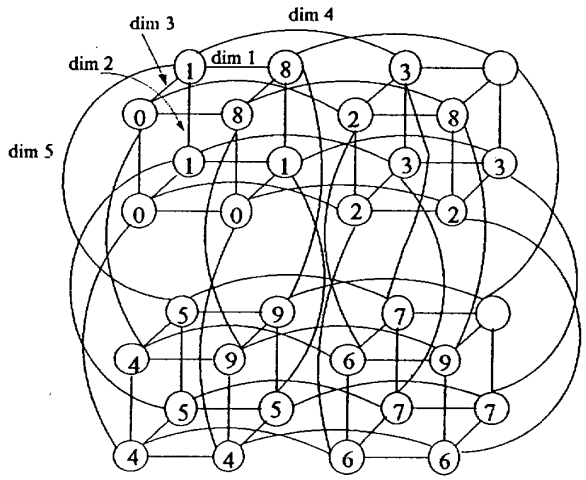


Figure 2: TMR units in Q_5.

Proof. First consider the case for even n. When n is even, $2^{n-2\lfloor n/2 \rfloor} = 1$ and can be ignored. As described above, Q_n consists of a series of subcubes, $Q_{n-2}, Q_{n-4}, \ldots, Q_0$, made up of TMR units. Since there are 2^m nodes in an m-dimensional hypercube Q_m, the total number of TMR units in Q_n is

$$2^0 + 2^2 + \ldots + 2^{n-4} + 2^{n-2} = \sum_{k=0}^{(n-2)/2} 2^{2k}$$

For even n, $\lfloor (n-2)/2 \rfloor = (n-2)/2$ and we have proven the left hand side of the theorem for even n.

To prove the right hand side for even n, we use the equality

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

$$\sum_{k=0}^{(n-2)/2} 2^{2k} = \sum_{k=0}^{(n-2)/2} 4^k = \frac{4^{(n-2)/2+1} - 1}{4 - 1} = \frac{4^{n/2} - 1}{3} = \frac{2^n - 1}{3}$$

For odd n, we can view Q_n as consisting of two Q_{n-1} hypercubes of even dimension. The limit of the summation, $\lfloor (n-2)/2 \rfloor$, gives the number of TMR units in Q_{n-1} when n is odd. When n is odd, $2^{n-2\lfloor n/2 \rfloor} = 2$, so the left hand side gives twice the number of TMR units in Q_{n-1}. Since only one node in Q_{n-1}, the one with address $1 \ldots 1$, is not a part of any TMR unit, viewing Q_n as two Q_{n-1}'s does not leave enough unused nodes to make another TMR unit that is not accounted for in the summation.

The right hand side for odd n can be derived by multiplying the right hand side for $n-1$ by 2

$$\frac{2(2^{n-1} - 1)}{3} = \frac{2^n - 2}{3}$$

□

If two TMR units need to communicate, they must be connected by finding a path between them (see Figure 3). Two TMR units, TMR_a and TMR_b, are connected by a 1-to-1 mapping from processors in TMR_a to processors in TMR_b. That is, there must be a path from each processor in TMR_a to a unique processor in TMR_b. Kiskis and Shin [9] show that these paths, as well as the paths within each TMR unit are node disjoint, thus allowing all single-point faults to be masked. For TMR units TMR_a and TMR_b in the same cube, Q_m (** in the same location), the optimal mapping is straightforward:

$$a_n a_{n-1} \ldots 001 \ldots 1 - b_n b_{n-1} \ldots 001 \ldots 1,$$
$$a_n a_{n-1} \ldots 011 \ldots 1 - b_n b_{n-1} \ldots 011 \ldots 1,$$
$$a_n a_{n-1} \ldots 101 \ldots 1 - b_n b_{n-1} \ldots 101 \ldots 1.$$

The distance between the TMR units, D_{TMR}, is the

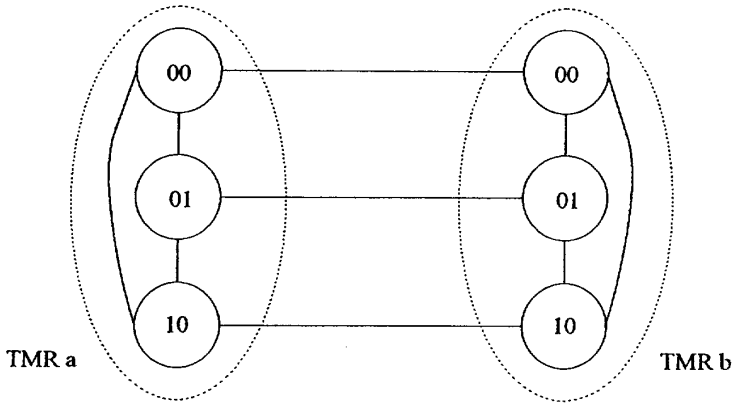


Figure 3: Communication within and between TMR units.

average Hamming distance, D_H, between the corresponding nodes. For TMR units in the same cube, this is the Hamming distance between any pair of nodes in the above mapping.

We define the distance between two cubes, D_C, to be the Hamming distance between two cubes where neither cube address is $*$.

Definition 1 *The distance between two cubes, D_C, is:*

$$D_C = \sum_{\forall i: a_i \neq *, b_i \neq *} (a_i \oplus b_i)$$

where a_i, b_i for $1 < i < n$ are the cube addresses, and $\oplus$ is the exclusive or operation.

The following theorems show how to calculate the distance between TMR units in the same and different cubes.

Theorem 2 *The distance between TMR units in the same cube ($**$ in same locations), is $D_{TMR} = D_C$*

Proof. Follows from the above discussion and the definition of D_C.
□

Theorem 3 *The distance between TMR units in different cubes ($**$ in different locations), is*

$$D_{TMR} = 2 + D_C + \frac{D_H(\alpha, 00)}{3}$$

where α is the 2 bits of the higher dimensional cube address where the lower dimensional cube address is $**$.

Proof. At the bit positions where neither address is $*$, each bit where the addresses are different contributes one to the distance between the TMR units, this is D_C.

Consider the two bits where the higher dimensional cube address is $**$. The lower dimensional cube address at these bit positions is 11. The node mappings at these two bit positions are 00–11, 01–11, 10–11, with corresponding $D_H = 2 + 1 + 1 = 4$.

Consider the two bits where the lower dimensional cube address is $**$. Let the higher dimensional cube address at these bit positions be $\alpha = 00$. The node mappings at these two bit positions are 00–00, 01–00, 10–00, with corresponding $D_H = 0 + 1 + 1 = 2$. If $\alpha = 01$ or $\alpha = 10$ then $D_H = 3$ and if $\alpha = 11$ then $D_H = 4$. At these bit positions, the TMR units are at least distance 2 apart, plus additional distance if the higher dimensional cube address is not 00. So at these two bit positions $D_H = 2 + D_H(\alpha, 00)$.

At the bit positions where either address is *, we are adding the distances between three nodes. To obtain the average distance between the individual nodes of each TMR unit, we must divide by 3. Adding the above distances together gives

$$D_C + \frac{4 + 2 + D_H(\alpha, 00)}{3} = 2 + D_C + \frac{D_H(\alpha, 00)}{3}$$

□

Corollary 1 *The minimum distance between TMR units in different cubes is 2.*

For consistency in this paper, we always select nodes with addresses 00, 01, and 10 in the 2-cube to make up the TMR units. However, any three nodes

could be chosen. If another choice is made, the distance formula given above changes slightly. The 00 in the term $D_H(\alpha, 00)$ is the complement of the unused address. For example, if addresses 00, 01, and 11 where chosen to make up the TMR units, address 10 would be unused, its complement is 01. Therefore, the term in the above formula would be $D_H(\alpha, 01)$.

For example, consider TMR_a with address $0^{(n-2)}**$ in Q_{n-2}, and TMR_b with address $0^{(n-4)}**11$ in Q_{n-4} ($0^{(n)}$ is a string of n 0's). While some nodes in TMR_a are only distance 1 from nodes in TMR_b, other nodes are distance 3 from each other. There is no way to connect the two TMR units so the average distance between corresponding nodes is less than 2. One possible connection yields a distance 2 between all corresponding nodes ($0^{(n-2)}00$ - $0^{(n-4)}0011$, $0^{(n-2)}01$ - $0^{(n-4)}0111$, $0^{(n-2)}10$ - $0^{(n-4)}1011$). This connection pattern may be desirable because of its symmetry, but the paths between nodes are not node disjoint. The connection ($0^{(n-2)}00$ - $0^{(n-4)}0111$, $0^{(n-2)}01$ - $0^{(n-4)}1011$, $0^{(n-2)}10$ - $0^{(n-4)}0011$) yields node disjoint paths, but the distances between nodes are 3, 2, 1 respectively. In this example, $D_C = 0$, $\alpha = 00$, and $D_{TMR} = 2$. As another example, using Theorem 3, the distance bewteen TMR units $0110100**$ and $101**1111$ is $2 + 4 + \frac{1}{3} = 6\frac{1}{3}$.

4 Optimal Embedding of 1-level k-ary Trees

In a compute-aggregate-broadcast style of programming [13], G_g can be a 1-level k-ary tree. During each round of computation, the leaves compute a value and send it to the root node. The root node might then distribute values to the leaves for further rounds of computation. The amount of communication with each leaf might be different, so the edges might not have equal weights.

The following algorithm assigns each node in G_g to a TMR unit in a hypercube. The algorithm is optimal in that it minimizes expansion and dilation. First, a total ordering of the nodes of G_g based on their distance from the root of G_g is found. Then a total ordering of the TMR units in Q_n based on their distance from TMR unit $0^{(n-2)}**$ is found by using binomial spanning trees for each cube. These two total orderings are then used to assign nodes in G_g to a TMR units in the hypercube.

Algorithm 1 *Tree Embedding*

Inputs: G_g, a weighted 1-level k-ary tree.

Q_n, an n-dimensional hypercube where n is the minimum dimension that will yield enough TMR units to embed G_g.

Outputs: An optimal TMR embedding of G_g in Q_n.

1. Order the nodes of G_g by decreasing distance (weight) from the root of G_g. In the case of nodes with equal distances, choose an arbitrary ordering.
2. Organize Q_n into TMR units as discussed in Section 3.
3. For $m = n-2, n-4, \ldots, 0$, construct a binomial tree rooted at $0^{(m)}**1^{(n-m-2)}$ that spans Q_m and order the TMR units in Q_m by increasing distance from the root node of the binomial spanning tree. For TMR units with equal distances, choose an arbitrary ordering.
4. Construct an ordering of the root TMR units of the binomial spanning trees constructed above based on increasing distance from TMR unit $0^{(n-2)}**$. Ties may be broken arbitrarily.
5. Using the ordering of TMR units within each Q_m, and the ordering of root TMR units, construct an ordering of all TMR units based on increasing distance from TMR unit $0^{(n-2)}**$. Ties may be broken arbitrarily.
6. Remove the first node from the total ordering of nodes of G_g, and remove the first TMR unit from the total ordering of TMR units in Q_n. Assign the node chosen from G_g to the TMR unit chosen from Q_n. Continue until all nodes of G_g are assigned to a TMR unit.

The ordering of TMR units based on increasing distance from TMR unit $0^{(n-2)}**$ could be found by direct application of Theorem 3 to each TMR unit. Using binomial spanning trees requires that Theorem 3 be used only for the root of the binomial spanning trees. It can be seen that the root of the binomial spanning tree of Q_{n-4} is distance 2 from the root of the binomial spanning tree of Q_{n-2}, the root of the binomial spanning tree of Q_{n-6} is distance 4 from the root of the binomial spanning tree of Q_{n-2}, etc. The root of the binomial spanning tree for Q_m, $0^{(m)}**1^{(n-m-2)}$, is chosen to be closer to $0^{(n-2)}**$ than any other TMR unit in Q_m.

The two total orderings are used to map the nodes of G_g to the TMR units of Q_n as follows. The first node in the G_g total ordering, the root node, is

mapped to TMR unit $0^{(n-2)}**$. The next node in the G_g total ordering gives the leaf that is the greatest distance (greatest weight) from the root node. This node should be mapped to the TMR unit closest to TMR unit $0^{(n-2)}**$. The next TMR unit in the TMR unit total ordering gives such a TMR unit. The node from G_g is assigned to this TMR unit. As nodes and TMR units are selected, they are removed from their respective total orderings and the process is repeated until all nodes are assigned to TMR units. Algorithm 1 is optimal in that it minimizes expansion by using the smallest Q_n that can embed G_g, and given that expansion, it minimizes dilation by associating large w_{ij} with small d_{ij}. If expansion is increased, dilation can be reduced.

Figure 4 shows a 1-level 9-ary tree and the binomial spanning trees of the Q_5 of Figure 2 used by Algorithm 1. Figure 4 shows a 1-level 9-ary tree and its TMR embedding into the 5-cube of Figure 2. The total ordering of the nodes in the 9-ary tree of Figure 4 is: $a, b, h, e, j, g, f, c, i, d$. The total ordering of the TMR units in the 5-cube, based on the binomial spanning trees of Figure 4, is: $0, 1, 2, 4, 3, 5, 6, 8, 7, 9$. The embedding of nodes to TMR units based on these total orderings is $(a, 0)$, $(b, 1)$, $(h, 2)$, $(e, 4)$, $(j, 3)$, $(g, 5)$, $(f, 6)$, $(c, 8)$, $(i, 7)$, $(d, 9)$.

5 Optimal Embedding of Unweighted Rings

Another common process interaction graph is the ring and its variation, the linear order. Processors arranged in a linear order can be used to process data in a pipelined fashion. In a ring, nodes only exchange information with a left and right neighbor, and the graph forms a closed loop. Since pairs of nodes may exchange different amounts of information, the edges may have unequal weights, however we will first consider unweighted rings.

A ring can easily be embedded in a conventional hypercube using the Gray code [4]. Since a Q_n can be viewed as a concatenation of Q_{n-2}, Q_{n-4}, ..., it follows that a ring of 2^{n-2} TMR units can be embedded in the Q_{n-2}, a ring of 2^{n-4} TMR units can be embedded in the Q_{n-4}, etc. We describe two methods for joining these rings together into one ring. In the first method, each of the rings formed by the $Q_m (m < n-2)$ cubes are inserted between two nodes of the ring formed by the Q_{n-2} cube. In the second method, the rings are nested one inside the other by inserting the Q_{n-4} ring between two TMR units of the Q_{n-2} ring, inserting the Q_{n-6} ring between two TMR units of the Q_{n-4} ring, etc.

The distance between TMR units in each ring is one. If the rings are joined at an arbitrary point, the distance at the jump between rings could be large. To allow an optimal embedding, the rings should be joined so the distance between them is minimized. Using Theorem 3, we can minimize the distance between TMR units where the rings join by setting the bits in Q_m to 0 where the bits in Q_{m-i} are $**$, and by keeping the other bits equal as much as possible. Even so, by Corollary 1 the distance between nodes where rings join is at least two.

To illustrate our first method, consider that within a Q_m, TMR units can be labeled by their binary digits to the left of the $**$ bits. For example, TMR unit $0010**11$ can be labeled as TMR unit 2 in Q_4. A ring can always be embedded in a Q_m such that TMR units 0 and 1 are adjacent. The link between TMR units 0 and 1 in Q_m can be deleted, and Q_m can be inserted between two TMR units of Q_{n-2}. The two TMR units of Q_{n-2} must be chosen to minimize the distance between the rings as discussed above. Table 2 and Table 3 show which TMR units are selected to join rings for Q_8 and Q_7. Figure 5 shows how a ring is embedded in Q_8

Again, we always select nodes with addresses 00, 01, and 10 in the 2-cube to make up the TMR units. If another choice is made, the above scheme should be adjusted to minimize the distance as discussed for Theorem 3.

When joining the Q_m ring to the Q_{n-2} ring, TMR unit 0 of Q_m is joined to TMR unit j of Q_{n-2} where

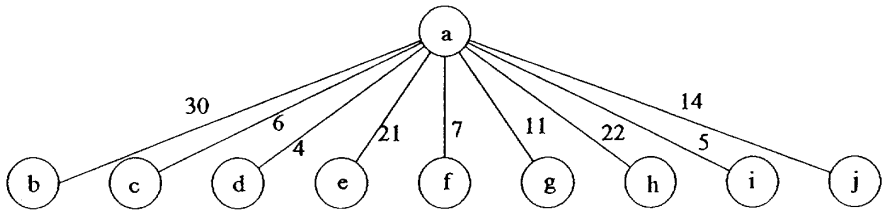
$$j = \begin{cases} 0 & \text{if } m = n-4 \\ \sum_{i=0}^{n-m-5} 2^i & \text{otherwise} \end{cases}$$

and TMR unit 1 of Q_m is joined to TMR unit k of Q_{n-2} where

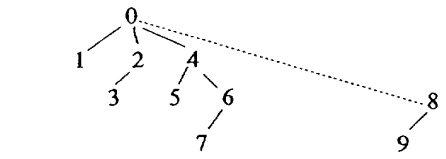
$$k = j + 2^l, \text{ where } l = \begin{cases} n-4 & \text{if } m = 0 \\ n-m-2 & \text{otherwise} \end{cases}$$

The formula for j accounts for the 1's in the Q_{n-2} TMR unit address that match up with the 1's to the right of the $**$ bits in the address of TMR unit 0 of Q_m. For the special case when $m = n-4$, there is no room to match up 1's, so TMR unit 0 in Q_n is used.

The formula for k accounts for the 1 in the Q_{n-2} TMR unit address that matches up with the 1 to the left of the $**$ bits in the address of TMR unit 1 of Q_m. When n is even, the lowest dimensional ring is Q_0 which consists of only a single node. This leads to the need to set $l = n-4$ when $m = 0$.



a)



Binomial tree spanning Q3 Binomial tree spanning Q1

b)

Figure 4: a) A 1-level 9-ary tree. b) Binomial spanning trees of the 5-cube of Figure 2.

Table 2: Joining rings in Q_8.

TMR unit	Cube	Address	Joins to	TMR unit	Cube	Address	Distance
0	Q_6	000000**	$\longleftrightarrow$	0	Q_4	0000**11	2
4	Q_6	000100**	$\longleftrightarrow$	1	Q_4	0001**11	2
3	Q_6	000011**	$\longleftrightarrow$	0	Q_2	00**1111	2
19	Q_6	010011**	$\longleftrightarrow$	1	Q_2	01**1111	2
15	Q_6	001111**	$\longleftrightarrow$	0	Q_0	**111111	2
31	Q_6	011111**	$\longleftrightarrow$	0	Q_0	**111111	$2\frac{1}{3}$

Table 3: Joining rings in Q_7.

TMR unit	Cube	Address	Joins to	TMR unit	Cube	Address	Distance
0	Q_5	00000**	$\longleftrightarrow$	0	Q_3	000**11	2
4	Q_5	00100**	$\longleftrightarrow$	1	Q_3	001**11	2
3	Q_5	00011**	$\longleftrightarrow$	0	Q_1	0**1111	2
19	Q_5	10011**	$\longleftrightarrow$	1	Q_1	1**1111	2

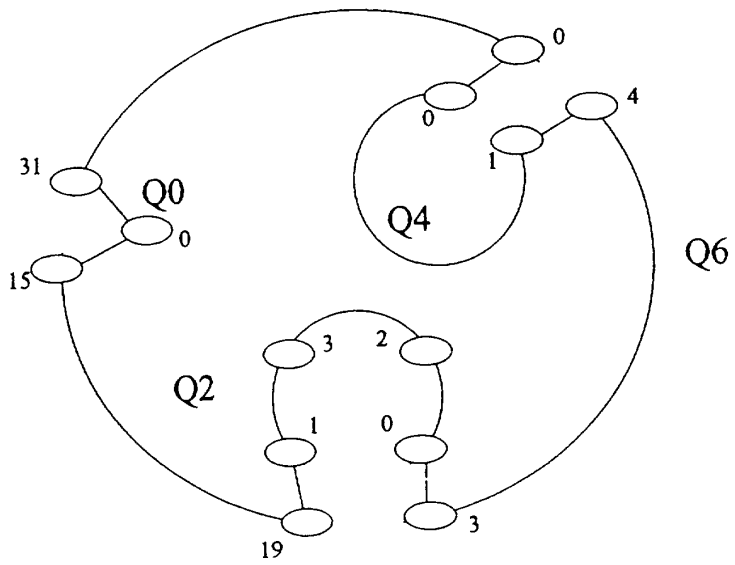


Figure 5: Embedding a ring in Q_8.

The method of joining rings just described leads to a two level structure where all the Q_m $(m < n-2)$ rings are inserted between TMR units of the Q_{n-2} ring. Finding a ring embedding for Q_{n-2} with all the pairs of j, k TMR units selected by the above formulas adjacent in the ring can be difficult. A straight forward application of a Gray code will not work.

Another way to join the rings into one ring is to insert the Q_{n-4} ring between two TMR units of the Q_{n-2} ring, insert the Q_{n-6} ring between two TMR units of the Q_{n-4} ring, etc. This forms a hierarchical structure of rings. Jumps to lower dimensional rings can always be made from TMR unit 0 in Q_m to TMR unit 0 in Q_{m-2}. Jumps to higher dimensional rings can be made from TMR unit 1 in Q_{m-2} to TMR unit 4 in Q_m when $m \geq 2$. When $m = 2$, Q_0 has only one TMR unit, and the shortest jump is from TMR unit 0 in Q_0 to TMR unit 2 in Q_2. The individual ring embeddings must have TMR units 1, 0, and 4 adjacent. Such embeddings are easier to find than the ring embedding for Q_{n-2} in the one level scheme described above. Figure 6 shows the hierarchical ring embedding for Q_8.

Both ring embedding schemes are optimal in that the distance between TMR units is minimized. Within each Q_m ring, the distance between TMR units is one. Where rings from different cubes are joined, the distance is two (which is minimal by Corollary 1), except when joining Q_0 which, because it has only one node, has distance $2\frac{1}{3}$. The two methods can be combined with some rings being nested more than two levels deep as in the second method, and multiple rings being inserted at each level as in the first method.

The following algorithm gives an optimal TMR embedding of an unweighted TMR ring, G_g into a hypercube.

Algorithm 2 *Ring Embedding*

Inputs: G_g, an unweighted ring.
Q_n, an n-dimensional hypercube where n is the minimum dimension that will yield enough TMR units to embed G_g.

Outputs: An optimal TMR embedding of G_g in Q_n.

1. Arrange the TMR units in each subcube into one logical ring using one of the methods described above. If there are more TMR units than nodes in G_g, the ring in the cube(s) with the lowest dimension can be shortened or eliminated.

2. Select any node, n, from G_g and assign it to any TMR unit, u, in the ring formed from Q_n.

3. Set $n = successor(n)$ and $u = successor(u)$ and assign n to u. Repeat until all nodes from G_g are assigned to TMR units.

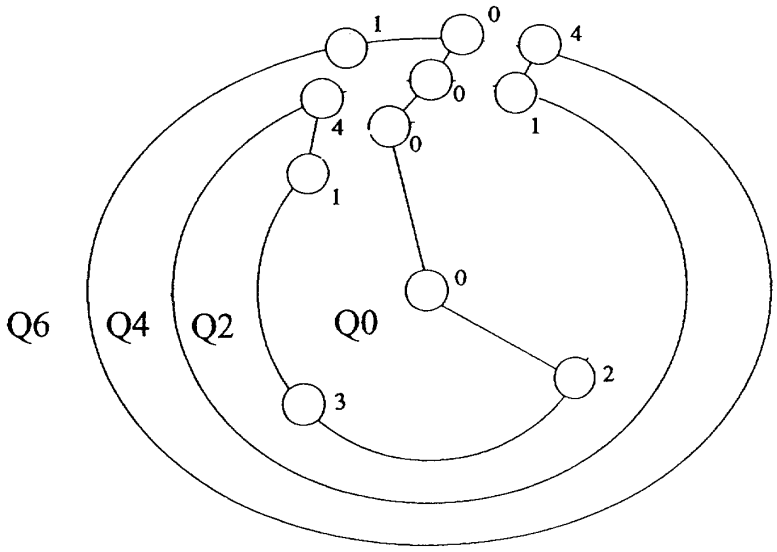


Figure 6: Hierarchical ring embedding for Q_8.

Algorithm 2 is optimal in that it minimizes dilation given the minimal expansion. Expansion is minimized by using the smallest Q_n that can embed G_g. Dilation is minimized because G_g is unweighted and one of the two optimal methods described above is used to join the Q_{n-2}, $Q_{n-4}\ldots$ rings together.

When G_g is a weighted ring, the jumps between TMR units in cubes with different dimensions must correspond to edges in G_g with small weights. Once one node from G_g is mapped to a TMR unit in Q_n, all other node to TMR unit mappings are determined. Thus making a good choice when jumping between two rings may lead to a bad choice when jumping between two other rings. All combinations must be considered, making this an NP-complete optimization problem.

6 Conclusions

In this paper we have discussed optimal TMR embeddings of special classes of process interaction graphs in a hypercube to provide fault-tolerance. The goal of our TMR embeddings is to minimize dilation given the minimum expansion. We developed a formula to determine how many TMR units a hypercube of a given dimension can support. For an odd dimensional hypercube, the formula doubles the result of the next smallest even dimensional hypercube. We also developed a formula for the distance between TMR units in different subcubes. The distance is the average Hamming distance between corresponding nodes in the two TMR units.

While finding optimal embeddings for general graphs is an NP-complete problem, we have considered two special types of graphs. An algorithm that produces an optimal TMR embedding for weighted 1-level k-ary trees was given. This embedding was based on finding a total ordering of nodes of the guest graph based on edge weight, and a corresponding total ordering of TMR units in the hypercube based on distance from TMR unit $0^{(n-2)}**$.

An algorithm that produces an optimal TMR embedding for an unweighted ring was also given. This algorithm was based on joining rings embedded in the individual subcubes, Q_{n-2}, $Q_{n-4}\ldots$, into one ring so the distance between TMR units where the rings are joined is minimized. It was noted that the problem of constructing a TMR embedding of a weighted ring is NP-complete.

Optimal TMR embeddings were found only for simple graphs. Additional work is needed to derive optimal embeddings for other graphs, and to develop efficient heuristics when the embedding problem is NP-complete.

References

[1] S. N. Bhatt, F. Chung, T. Leighton, and A. Rosenberg. Optimal simulation of tree machines. In *Proceedings of the 27th Annual IEEE Symposium on the Foundations of Computer Science*, pages 272–282, 1986.

[2] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, Feb. 1988.

[3] M. Y. Chan and F. Y. L. Chin. On embedding rectangular grids in hypercubes. *IEEE Trans. Comput.*, 37(10):1285–1288, Oct. 1988.

[4] M. S. Chen and K. G. Shin. Processor allocation in an n-cube multiprocessor using Gray codes. *IEEE Trans. Comput.*, C-36(12):1396–1407, Dec. 1987.

[5] W. W. Chu, L. J. Holloway, M.-T. Lan, and K. Efe. Task allocation in distributed data processing. *IEEE Computer*, 13(11):57–69, Nov. 1980.

[6] Y.-C. Chung and S. Ranka. Mapping finite element graphs on hypercubes. *Third Symposium on the Frontiers of Massively Parallel Computation*, pages 135–144, 1990.

[7] M. Gangadhar. Fault tolerant scheduling schemes. Master's thesis, Florida Atlantic University, Department of Computer Science and Computer Engineering, Aug. 1992.

[8] C. T. Ho and S. L. Johnsson. Dilation d embedding of a hyper-pyramid into a hypercube. In *Proceedings of the Supercomputing '89*, pages 294–303, Nov. 1999.

[9] D. L. Kiskis and K. G. Shin. Embedding triple-modular redundancy into a hypercube architecture. In G. Fox, editor, *The Third Conference on Hypercube and Concurrent Computers and Applications*, pages 337–345, Pasadena, CA, 1988. Jet Propulsion Laboratory, Association for Computing Machinery.

[10] N. Krishnakumar, V. Hegde, and S. S. Iyengar. Fault tolerant based embeddings of quadtrees into hypercubes. In *1991 International Conference on Parallel Processing*, pages 244–249, 1991.

[11] H. Liu and S. H. Huang. Dilation-6 embeddings of 3-dimensional grids onto optimal hypercubes. In *1991 International Conference on Parallel Processing*, pages 250–254, 1991.

[12] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Comput.*, 37(11):1384–1397, Nov. 1988.

[13] P. A. Nelson and L. Snyder. Programming paradigms for nonshared memory parallel computers. In *The Characteristics of Parallel Algorithms*, pages 3–20. The MIT Press, Cambridge, MA, 1987.

[14] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez, Jr. Optimal scheduling stategies in a multiprocessor system. *IEEE Trans. Comput.*, C-21(2):137–146, Feb. 1972.

[15] Y. Saad and M. H. Schultz. Topological properties of hypercubes. *IEEE Trans. Comput.*, 37(7):867–872, July 1988.

[16] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Softw. Eng.*, 3(1):85–93, Jan. 1977.

[17] A. Y. Wu. Embedding of tree networks into hypercubes. *J. Parallel Dist. Comput.*, 2:238–249, 1985.