Manuscript received Jan. 4, 2008; revised June 20, 2008

Load-Balance and Fault-Tolerance for embedding a Complete Binary Tree in an IEH with *N*-expansion

Jen-Chih Lin
Department of Digital Content Design
National Taipei University of Education,
No.134, Sec. 2, Heping E. Rd., Da-an District,
Taipei City 106, Taiwan, R.O.C.
E-mail: *yachih@tea.ntue.edu.tw

Abstract: - Embedding is of great importance in the applications of parallel computing. Every parallel application has its intrinsic communication pattern. The communication pattern graph is embedded in the topology of multiprocessor structures so that the corresponding application can be executed. This paper presents strategies for reconfiguring a complete binary tree in a faulty Incrementally Extensible Hypercube (IEH) with *N*-expansion. This embedding algorithm show a complete binary tree can be embedded in a faulty IEH with dilation *4*, load *1*, and congestion *1* such that $O(n^2-h^2)$ faults can be tolerated, where *n* is the dimension of IEH and (*h-1*) is the height of a complete binary tree. Furthermore, the presented embedding methods are optimized mainly for balancing the processor loads, while minimizing dilation and congestion as far as possible. According to the result, we can embed the parallel algorithms developed by the structure of complete binary tree in an IEH. This methodology of embedding enables extremely high-speed parallel computation.

Key-Words: - Incrementally Extensible Hypercube, Binary tree, Load-Balance, Fault-Tolerance, Embedding

1 Introduction

Hypercube, as one of the most popular structures, has been used as the interconnection network in a wide variety of commercial and experimental distributed memory multiprocessors such as the Cosmic Cube[12], the Intel "hypercube" systems (iPSC, iPSC/2), the Ametek/Symult S-series, the NCUBE and the Connection Machines (CM-1, CM-2). The popularity of the hypercube multiprocessor or multicomputer systems is due to their tempting properties such as logarithmic diameter and node degree, high bisection width, ease to embed other common structures[11], and many known efficient data communication schemes. Although hypercubes possess many advantages for parallel and distributed computing, there are some limitations for constructing hypercubes. However, due to the power-of-2 size and logarithmic degree, hypercubes suffer two major disadvantages, namely, high cost extensibility and large internal fragmentation in partitioning. Limitations of the hypercube include its nonplanarity (which complicates the layout of hypercubes implemented within VLSI chips), and its inability to grow incrementally. The incremental extensibility is a very essential and desirable property in real world applications for designing interconnection networks.

Of late, a novel interconnection topology is proposed in [14, 15] called the Incrementally Extensible Hypercube or IEH graph, based on appropriate interconnection if different sized hypercubes of smaller dimensions. The IEH graph is showed to have the following desirable characteristics: (1) adding a new node to an existing network is easy and simple; in most cases no reorganization of existing edges is necessary, (2) the network is optimally fault-tolerant in the sense that the vertex connectivity is equal to the minimum degree of a node in the graph, (3) the number of edges is O(*NlogN*) where *N* is the number of nodes, and the diameter is logarithmic in the number of nodes, and (4) the network is near regular i.e., the difference between the maximum and the minimum degree of a node is at most *1*. At the same time, since the IEH graph is constituted by several hypercubes, which are turned to be as subcubes with smaller dimensions in this case, all parallel algorithms run in the hypercube can be easily ported in IEH graphs. Therefore, we consider how to embed the complete binary tree in the IEH with *N*-expansion.

The tree is a basic network topology. A

complete binary tree is special tree underlying divide-and-conquer algorithms. A complete binary tree arises in the solution of tridiagonal systems by even-odd cyclic reduction and solution of systems of equations that is noted in [6]. Suppose some process can be naturally decomposed into a collection of subprocesses that can be executed concurrently with certain communication between subprocesses by an edge between corresponding nodes. One obtains a complete binary tree by denoting each subprocess by a node and each communication between subprocesses by an edge between corresponding nodes. The problem of allocating those subprocesses, structured by a complete binary tree, to processors in a given interconnection networks will be reduced to the problem of embedding a complete binary tree.

Load Balancing, communication locality, communication congestion, and node utility in process graphs can be abstractly studies as the problem of embedding[7]. In a process graph, the nodes represent processes comprising a distributed program or a parallel program and the edges represent communications between processes. Embedding one graph into another is important because an algorithm may have been designed for a specific interconnection network, and it may be necessary to adapt it to another network. The quality of an embedding of a guest graph *G* in a host graph *H* is measured by the maximum number of processes of *G* placed on any processes of *H*, the maximum distance between any pair of processes of *H* corresponding to a pair of neighbor processes of *G*, the maximum number of edges of *G* placed on any edge of *H*, and the ratio of the order of *H* to the order of *G*. These factors are called load, dilation, congestion, and expansion, respectively. The embedding problem is to find embeddings with balanced loads, small dilations, and small congestions. An efficient simulation of one network on another network requires that these four costs be as small as possible. However, for most embedding problems, it is impossible to obtain an embedding that minimizes these costs simultaneously. Therefore, some tradeoffs among these costs must be made.

In a multiprocessor system, we follow two fault models defined in [5]. The first model assumes that, in a faulty node, the computational function of the node is lost while the communication function remains intact; this is the partial faulty model. The second model assumes that, in a faulty node, the communication function is lost too; this is the total faulty model. Conceptually, the network interface hardware operates independent of the computer's processor. In this paper, our model is the partial faulty model. That is, when the computation nodes are faulty, the communication links are well and only the faulty nodes are remapped.

The remainder of this paper is organized as follows. Section 2 defines the IEH structure. In section 3, we describe how to construct a complete binary tree in an IEH. In section 4, we describe how to embed a complete binary tree in a faulty IEH with *N*-expansion under partial faulty model. Finally, we conclude this paper.

2 Preliminaries

We briefly describe notations and definitions of the hypercube and the IEH graph.

The hypercube is based on the properties of binary *n*-cube in graph theory. An *n*-dimensional hypercube, simply called *n*-cube, can be modeled as a graph $H_n=(V,E)$ with node set $V(H_n)$ and edge set $E(H_n)$, where $|V| = 2^n$, $|E| = n * 2^{n-1}$. For a H_n ,each node in $V(H_n)$ can be distinctly labeled by a unique binary string of length *n*, then $V(H_n)=\{0,1,2,\cdots,2^n-1\}$ and $E(H_n)=\{(u,v) \mid u,v \in H_n\}$ and $HD(u,v)=1$. That is, each of the nodes corresponds to an *n*-bit binary string $v= X_{n-1}X_{n-2}\cdots X_1X_0$ as its label, and two nodes are connected by an edge if and only if their labels differ in precisely one bit.

The IEH graph is the composition of some *m* different hypercubes. Let $G_n(N)$ be an IEH graph with *N* nodes, and *N* can be expressed by the binary string $N= b_nb_{n-1}b_{n-2}\ldots b_1b_0$, and $b_i \in \{0,1\}$. An IEH graph $G_n(N)$ is composed of some different hypercubes which have lower dimension than $G_n(N)$ has. That is, $G_n(N)$ contains a hypercube, denoted by H_i, if and only if the i_{th} bit in the binary representation of *N* is *1*.

Accordingly, the IEH graph is composed of some hypercubes, so there is a new type of connections beside the usual connections in a hypercube. These edges (or links) are used for connecting two hypercubes are called *Inter-Cube* or IC edges. The basic philosophy in the design of the IEH graphs is to express *N* as a sum of several powers of *2*, i.e., to write *N* as a binary number,

build the smaller hypercubes, and then to add appropriate inter-cube edges to connect those smaller hypercubes. For any given N, $2^n \le N < 2^{n+1}$, the steps of finding IEH graphs are as follows.

Step 1 Build subcube graphs. Express N as $(n+1)$ bits a binary number as $N= b_n b_{n-1} b_{n-2} \ldots b_1 b_0$, where $b_i \in \{0,1\}$ and $b_n=1$ since $N \ge 2^n$. For each b_i, $b_i \neq 0$, construct a hypercube graph Q_i with 2^i nodes.

Step 2 Label the nodes. Note that each node has a $(n+1)$-bit binary label. Each hypercube Q_i is labeled as $11\ldots10 b_{i-1} b_{i-2} \ldots b_1 b_0$. Obviously each hypercube of dimension i (having 2^i nodes) has i number of dashed and the individual nodes of the hypercube can be obtained by filling the dashes with *0* or *1* in all possible ways. In other words, the binary representation of each node in Q_i has the same prefix of $(n-i)1$'s followed by a single zero.

Step 3 Construct the incremental hypercube in steps by providing the inter-cube edges. Find the minimum i such that $b_i \neq 0$. Set $j=i$ and $G_j=Q_i$.

Set $i=i+1$.

While $i \le n$ do

if $b_i \neq 0$ then

if $i-j=1$ then

each node x in G_j with label $11\ldots b_j b_{j-1} \ldots b_0$ is connected to the node $11\ldots10 b_j b_{j-1} \ldots b_0$ of Q_i.

else

each node x in G_j with label $11\ldots1 b_j b_{j-1} \ldots b_0$ is connected to $(i-j)$ different nodes of Q_i chosen in the following way:

$$\begin{cases} 11\ldots11011\ldots11 b_j b_{j-1} \ldots\ldots b_0 \\ 11\ldots11001\ldots11 b_j b_{j-1} \ldots\ldots b_0 \\ 11\ldots11010\ldots11 b_j b_{j-1} \ldots\ldots b_0 \\ \vdots \\ 11\ldots11011\ldots01 b_j b_{j-1} \ldots\ldots b_0 \\ 11\ldots11011\ldots10 b_j b_{j-1} \ldots\ldots b_0 \end{cases}$$

Set $j=i$ and set G_j to be the composite graph generated in the previous steps. Note that G_j has now $\sum_{k=0}^{j} b_k 2^k$ nodes and the binary label of each node in G_j has a prefix of $(n-j)$ *1*'s.

$i=i+1$

Return G_n as the desired incremental hypercube graph of N vertices. □

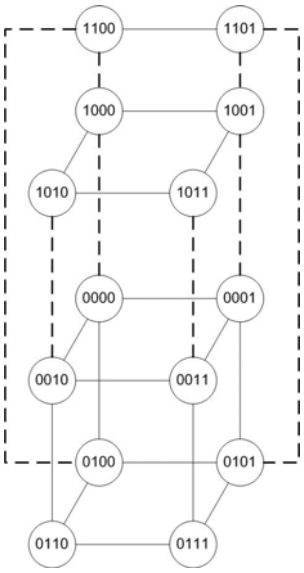

Fig. 1: The IEH graph contains 14 nodes

Figure 1 shows the example of $G_3(14)$. $G_3(14)$ consists of three subcubes. The three subcubes are *1*-subcube(H_1), *2*-subcube(H_2), and *3*-subcube(H_3). Nodes *12* and *13* are composed as a *1*-subcube(H_1), Nodes *8*, *9*, *10*, and *11* are composed as a *2*-subcube(H_2), and nodes *0*, *1*, *2*, *3*, *4*, *5*, *6* and *7* are the elements of a *3*-subcube(H_3). The edges (*8, 12*), (*9,13*) are IC edges connected between H_1 and H_2 such that H_1 and H_2 are connected to be an IEH graph containing *6* nodes($G_2(6)$). In addition, the H_3 connects to $G_2(6)$ with these IC edges (*0, 8*), (*1, 9*), (*2,10*), (*3,11*), (*4,12*), and (*5,13*).

Definition 1[7] Let $x=x_{n-1}\ldots x_0$, $y=y_{n-1}\ldots y_0$, then $Dim(x, y)=\{i \text{ in } (0\ldots n-1) \mid x_i \neq y_i\}$.

Definition 2[7] The Hamming distance between two nodes with labels $x=x_{n-1}x_{n-2}\ldots x_0$ and $y= y_{n-1}y_{n-2}\ldots y_0$ is defined as

$$HD(x, y)= \sum_{i=0}^{n-1} hd(x_i, y_i), \text{ where } hd(x_i , y_i)= \begin{cases} 0, \text{if } x_i = y_i, \\ 1, \text{if } x_i \neq y_i. \end{cases}$$

Definition 3[8] If a complete binary tree is a rooted binary tree and each internal nodes contains two offspring nodes, then a complete binary of height h denoted by T_h, contains 2^h-1 nodes.

Definition 4[8] A double-rooted binary tree DT_h, where h is the height of the tree, is a complete binary tree with the root replaced by a path of length two.

Lemma 1[8] A double-rooted complete binary tree can be embedded in a hypercube with dilation *2* and load *1*.

We illustrate an example as in figure *2* to figure *6*.

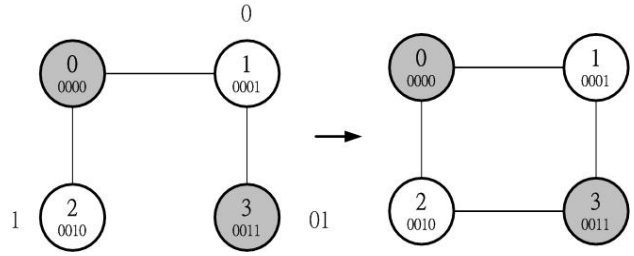

Figure 2: A double-rooted with *4* nodes can be embedded into a *2*-cube

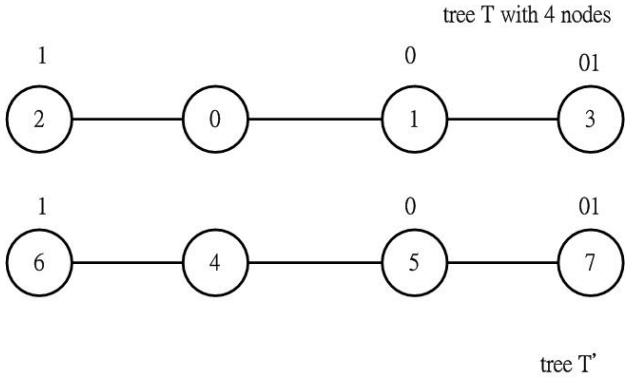

Figure 3: The transformation of mapping (1)

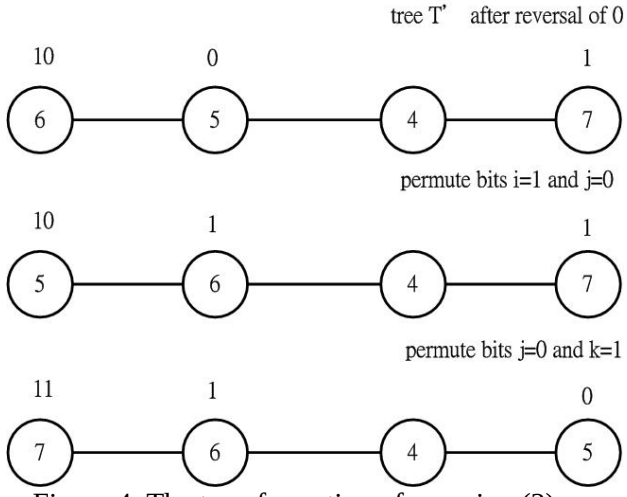

Figure 4: The transformation of mapping (2)

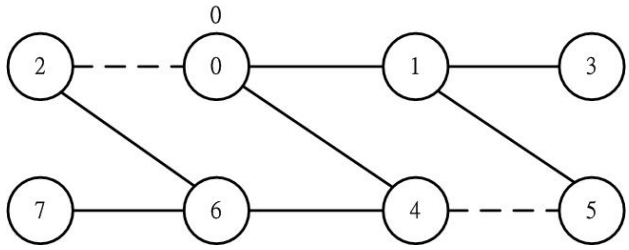

Figure 5: The transformation of mapping (3)

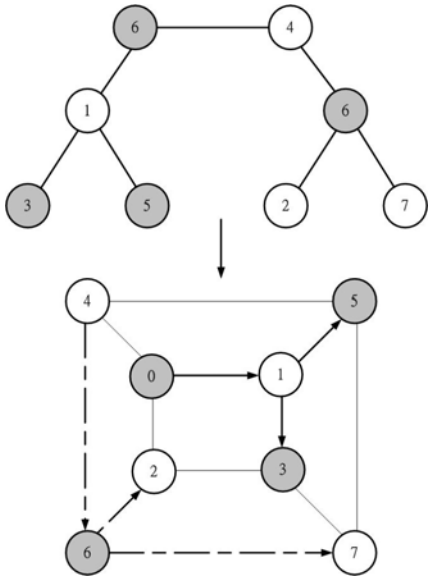

Figure 6: A double-rooted tree with 8 nodes can be embedded into a 3-cube

Definition 5[8] An IEH graph is called a full IEH graph, denoted by F_n, if and only if it has $2^{n+1}-1$ nodes. Intuitively, a full IEH graph must contain hypercubes $H_0, H_1, \ldots, H_n$ as its subcubes.

3 Mapping of Complete Binary Tree

IEH graphs are provided with all properties of hypercubes because an IEH graph may contain some different-sized hypercubes as its subgraphs. Thus the IEH graphs are selected to be the host graph of our embedding. Complete binary tree are usually used in a lot of algorithms and communications, so we proposed the method of embedding complete binary trees in IEH graphs.

We describe our approach that maps a complete binary tree in the full IEH or the IEH.

Lemma 2 A complete binary tree with height n contains the same number of nodes as an (n-*1*)-dimensional full IEH graph.

Proof. Suppose a tree T_n is said to be a complete binary tree with height n, then each node of T_n contains the same number of nodes in its right and left subtrees. Consequently, the number of nodes in a complete binary tree T_n has totally 2^n-1 nodes. A (n-*1*)-dimensional full IEH graph must, by the definition, contain n hypercubes with different dimensions. Therefore, an (n-*1*)-dimension IEH graph, simply denoted by F_{n-1}, will consist of hypercube with dimension with *0*, *1*, *2*, …, and (n-*1*). The k_{th}-dimensional hypercube in a IEH is denoted by H_k. Obviously, an (n-*1*)-dimensional full IEH

contains $2^0+2^1+2^2+\ldots+2^{n-1} = 2^n-1$ nodes because each k_{th}-dimensional hypercube in the IEH graph contains 2^k nodes. Hence, a complete binary tree with height n contains the same number of nodes as an (n-1)-dimensional full IEH graph. □

Lemma 3 An (n-1)-dimensional full IEH graph is a subgraph of an n-dimensional full IEH graph.

Proof. Let F_n be a full IEH graph, then two divided components H_n and F'_n exist in F_n where H_n is an n-dimensional hypercube and F'_n is an (n-1)-dimensional full IEH graph. According to the method of constructing an IEH graph, the vertex set of F_{n-1} is a subset of the vertex set of F_n and the edge set of F_{n-1} is a subset of the edge set F_n. Therefore, F_{n-1} is said to be a subgraph of F_n.□

Lemma 4 Assume that H_n is an n-dimensional hypercube and F_{n-1} is and (n-1)-dimensional full IEH graph, F_{n-1} is a subgraph of H_n.

Proof. According to the method of constructing an IEH graph, H_0, H_1, …, H_{n-1} construct F_{n-1}. These *IC* edges connecting between the n hypercubes are all of Hamming distance *1*. Hence, these *IC* edges in F_{n-1} are normal edges in H_n. Consequently, the vertex set of F_{n-1} is a subset of the vertex set of H_n and the edge set of F_{n-1} is a subset of the edge set H_n. Therefore, F_{n-1} is said to be a subgraph of F_n.□

Lemma 5 A full IEH graph contains a complete binary tree.

Proof. In these trivial cases, complete binary trees with *1* and *3* nodes can be directly embedded into IEH graphs F_0 and F_1, respectively. The case of embedding a tree with a single node into F_0 is straightforward. Embedding a complete binary tree T_2 into an IEH F_1 must select node labeled by binary string *00* to be the root of T_2. The other two nodes of F_1 are the right and left leave nodes of the tree respectively. The embedding of T_3 into F_2 is the base case (shown in figure 7) of our embedding method. According to the method of generating an IEH graph, F_2 is constructed by connecting three hypercubes such as H_0, H_1 and H_2. In F_2, H_0, H_1 and H_2 are connected by *IC* edges. Not all of the nodes in an IEH graph are connected by *IC* edge. Node *011* is the only one that does not link to any node by an *IC* edge in F_2. As a consequence, the node *011* can not be the interior node of a binary tree because the node *011* belongs to H_2 and the degree of *011* is two. Additionally, the number of nodes of H_2 except for the node *011* is equal to the sum of number of nodes of H_0 and H_1 in the full IEH graph F_2. Therefore, *011* must be the root of T_3. The adjacent nodes of node *011* are the nodes *001* and *010*, that can be the root of the left subtree and the root of the right subtree respectively in the complete binary tree T_3. For the leave nodes of T_3, two nodes, *000* and *101*, are connected by *001*, then *000* and *101* can be the left and right leaves of *001* respectively. There is a problem to map the leave nodes of *010* in F_2 because the node *010* connects only one unused node *110*. The solution of connecting the last free node *100* to the root node *010* of the right subtree must pass a used node, which is called midway node. As a result, the edge between *010* and *100* is mapped into the path from *010* to *100* through a midway node *000*. Hence, the embedding of T_3 into F_2 is constructed. Assume a complete binary tree T_n can be embedded into a F_{n-1}. Let a complete binary tree T_{n+1} Can be embedded into a F_n. There exist two divided components H_n and F'_n in F_n where H_n is an n-dimensional hypercube and F'_n is an (n-1)-dimensional full IEH graph. We select the node $0_n\underline{1_{n-1}1_{n-2}\cdots 1_0}$ to be a root of F_n, because node $0_n\underline{1_{n-1}1_{n-2}\cdots 1_0}$ belongs to H_n and node $\underline{1_{n-1}1_{n-2}\cdots 1_0}$ does not exist in F_{n-1}. According to lemma 10, F_{n-1} is a subgraph of H_n. In addition, the number of nodes of H_n except for the node $0_n\underline{1_{n-1}1_{n-2}\cdots 1_0}$ is equal to the sum of number of nodes of H_0, H_1, …, and H_{n-1} in the full IEH graph F_{n-1}. So, we can extend a bit with zero in the left of the most significant bit of these nodes in F_{n-1} such that these nodes can map in H_n. Hence, we can construct the left subtree of T_{n+1}. According to Lemma 9, F_{n-1} is a subgraph of F_n. In addition, the number of nodes of F'_n is equal to the sum of number of nodes in the full IEH graph F_{n-1}. In other words, $\delta : V(F'_n) \to V(F_{n-1})$ is the one-to-one mapping function. So, we can extend a bit with one in the left of the most significant bit of these nodes in F_{n-1}, such these nodes can map in F'_n. Therefore, we can construct the right subtree of T_{n+1}. Without loss of generality, we know the root of the left subtree and the root of the right subtree in F_n is $0_{n+1}0_n\underline{1_{n-1}1_{n-2}\cdots 1_0}$ and $1_{n+1}0_n\underline{1_{n-1}1_{n-2}\cdots 1_0}$, respectively. Therefore, there is a problem of connecting the root node and the root of the right subtree because *HD*(root, the root of right subtree) is not equal to *1*. The solution of connecting the root node *011…11* and the root node *101…1* of the right subtree to must pass a used node *001…1*. As a result,

the edge between *011…1* and *101…1* is mapped into the path from *011…1* to *101…11* through a midway node *001…1*. So the embedding of T_{n+1} into F_n is constructed.□

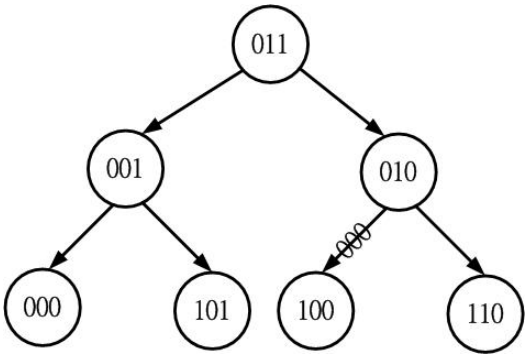

Figure 7: A complete binary tree T_3 can be embedded into F_2.

We extend the process of the base case to embed T_n to F_{n-1}, where $n \geq 4$. The process can be divided into three steps:

1. Identify the root of T_n
2. The left subtree and the right subtree is easily constructed by embedding T_{n-1} into F_{n-2}.
3. Combine the root, the left subtree and the right subtree.

The construction of right subtree is simply to extend one significant bit in the binary strings of nodes that are the embedding nodes of T_{n-1} into F_{n-2} and label the bit with *1*. Then, the construction of left subtree is simply to change the most significant bit of the right subtree from *1* to *0*. Therefore, the algorithm needs a subprocess concatenate, to accomplish the embedding. The algorithm is described as follows:

Algorithm Embedding (T_n)
Input: T_3, n /* T_3 is the base case, n is the size of a complete binary tree */
Output: T_n /* The complete binary tree with height n */
Begin
1. If n = 3 then Return T_3
2. Else
3. *Begin*
4. Root = 01^{n-1}
5. /* 01^{n-1} is the binary string with MSB *0* following (*n-1*) *1*'s. */
6. *Right = concatenate* (*1, Embedding* (T_{n-1}))
7. /* All nodes of T_{n-1} extend their MSB and set them to *1*'s. */
8. *Left = concatenate* (*0, Embedding* (T_{n-1}))
9. /* All nodes of T_{n-1} extend their MSB and set them to *0*'s. */
10. $V(T_n) = \{root\} \cup V(Right) \cup V(Left)$
11. $E(T_n) = E(Right) \cup E(Left) \cup \{(root, root_R), (root, root_L)\}$
12. /* $root_R$, and $root_L$ are the roots of right and left subtrees */
13. End;

End.

There are two examples in this section. The first example is to embed the complete binary tree T_4 into a full IEH graph F_3, the root of T_4 is the node *0111*, which is selected by the above discussion. Certainly, the node *0111* is the only one node without linked *IC* edge. The construction of the right subtree of T_4 is based on the result of embedding T_3 into F_2. By extending one significant bit of the binary strings of all nodes in T_3, the right subtree of T_4 is constructed with setting the extended bit to *1*'s and the left subtree of T_4 is constructed with setting the extended bit to *0*'s. After the right and left subtree are constructed, the embedding complete binary tree is accomplished by connecting the root node *0111* to the roots of left and right subtree. The midway nodes in the embedding complete binary tree are transformed as the same process as the nodes in trees. To embed the complete binary tree T_4 into a full IEH graph F_3 are shown in figure 8.

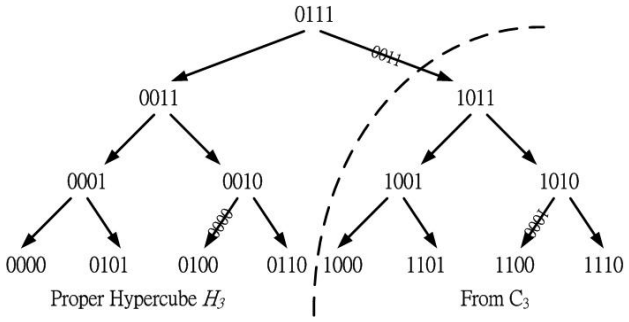

Figure 8: T_4 is embedded into F_3

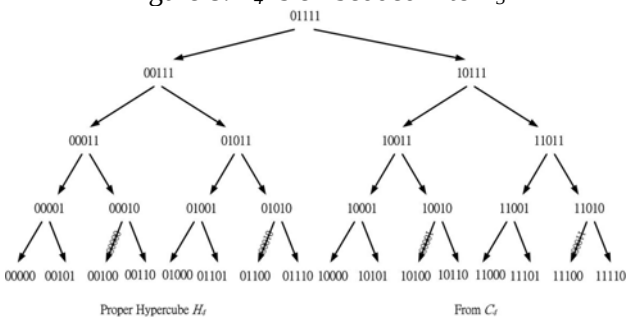

Figure 9: T_5 is embedded into F_4

For the detail description, the second example of embedding T_5 into F_4 is shown in figure 9 and listed

as follows:

1. Identify node *01111* is the root of T_5.
2. T_4 is constructed in the previous example. We use T_4 as the original tree of the right subtree in T_5.
3. We extend the significant bit of all nodes in the original right subtree and set the bit to *1*. Then the formal right subtree of T_5 is constructed.
4. Construction of left subtree is simply to change the most significant bit of the right subtree from *1* to *0*.
5. We connect the roots of the left and right binary tree to the selected root *01111*, then the embedded T_5 in F_4 is constructed.

In consequence, our embedding algorithm can also be applied into hypercubes with one faulty node. Therefore, we develop an algorithm to embed a complete binary tree with height *n* into an *n*-dimensional hypercube.

Lemma 6 A complete binary tree can be embedded in a hypercube with one faulty node.

Proof. Because F_{n-1} is a subgraph of H_n, a complete binary tree can also be embedded into a hypercube. H_n has more than one node than F_{n-1} and F_{n-1} which does not contain the node $(1)^n$. As a reason, a complete binary tree can be embedded into a hypercube with one faulty node. As a result, these edges in a tree transformed by the right subtree that is produced by F_{n-2} exist in the F_{n-1}. So, the embedding algorithm is correct for constructing the complete binary tree. The embedding algorithm is with dilation *2*, expansion *1*, congestion *1*, and load *1*. The following theorems show the measurement of our embedding algorithm. □

Lemma 7 Embedding the complete binary tree in a full IEH graph or a hypercube graph is dilation *2*, expansion *1*, load *1* and congestion *1*.

Proof. In the embedding of T_n into F_{n-1}, there are some midway node must be passed to connect two nodes That is, some edges in the tree T_n are mapped in a path of the length *2* in the IEH graph F_n. As the consequence, the dilation of the embedding is *2* and congestion is *1*. The embedding of T_n into F_{n-1} is one-to-one mapping for these nodes, so the expansion and load of our embedding is exactly equal to *1*. Therefore, embedding the complete binary tree into a full IEH graph or a hypercube graph is dilation *2*, expansion *1*, load *1* and congestion *1*.□

Theorem 1 A complete binary tree T_n can be embedded in $G_n(N)$ with expansion 2, dilation *2*, congestion *1*, and load *1*.

Proof. It is trivial by lemma 7.

Theorem 2 A complete binary tree T_h can be embedded in $G_n(N)$ with *N*-expansion, dilation *2*, congestion *1*, and load *1*.

Proof. It is trivial by theorem 1.

4 Load-Balancing and Fault-Tolerance Embedding with *N*-Expansion

In section *3*, T_h can be embedded in IEH $G_n(N)$ with *N*-expansion by theorem *2*. In this section, we present how to embed a complete binary tree to a faulty IEH graph. Hence, we consider a complete binary tree can be embedded in an IEH with *N*-expansion graph which contains faulty node.

By theorem *2*, T_n can be embedded in $G_n(N)$ with *N*-expansion. Furthermore, we propose an algorithm LB_Tree_Embedding( ) for embedding a complete tree in a faulty IEH as follows.

Algorithm LB_Tree_UB_Embedding(*x*)

Input: *x* /*the faulty node*/, $G_n(N)$, T_h
Output: *y* /*the replace node*/
1. if the root *r* is faulty then
2. search the other root node *r'*
3. if the node *r'* is faulty then backtrack the root *r*
4. else
5. return(*r'*) /* node *r* is replaced by node *r'*/
6. *exit()*
7. if any node *x* is faulty then
8. *j*=0
9. construct a queue Q; Q= Φ
10. while $j \le (n-h+1)$ do
11. we can search the node ϖ
12. /* $HD(x,\varpi)=1$, $Dim(x,\varpi)=h+j$ */
13. if node ϖ is exist and it is free then
14. node *x* is replaced by node ϖ
15. remove all of the nodes in a queue
16. *exit()*
17. insert($\varpi, h+j-1$) in a queue
18. *j=j+1;*
19. while Q is not empty do
20. remove the first pair (α,β) from Q
21. *k=0*
22. while $k \le \beta$ do
23. we can search the node λ

24. /* $HD(\alpha,\lambda)=1,\ Dim(\alpha,\lambda)=k$ */
25. if node λ is exist and it is free then
26. node *x* is replaced by node λ
27. *exit()*
28. *k=k+1*
29. Declare the replaceable node is faulty□

Finding the replace node as follow:

node 0 = $0_n\ 0_{n-1}\ldots 0_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X_0$

node 1 = $0_n\ 0_{n-1}\ldots 0_{h+1}1_h X_{h-1}X_{h-2}\ldots X_1X_0$

node 2 = $0_n\ 0_{n-1}\ldots 1_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X_0$

…

node n-h+1 = $1_n\ 0_{n-1}\ldots 0_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X_0$

node n-h+2 = $0_n\ 0_{n-1}\ldots 0_{h+1}1_h X_{h-1}X_{h-2}\ldots X_1X'_0$

…

node n+1 = $0_n\ 0_{n-1}\ldots 0_{h+1}1_h X'_{h-1}X_{h-2}\ldots X_1X_0$

node n+2 = $0_n\ 0_{n-1}\ldots 1_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X'_0$

…

node n+h+2 = $0_n\ 0_{n-1}\ldots 1_{h+1}1_h X_{h-1}X_{h-2}\ldots X_1X_0$

node n+h+3 = $0_n\ 0_{n-1}\ldots 1_{h+2}0_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X_0$

…

node n+2h+4 = $0_n\ 0_{n-1}\ldots 1_{h+2}1_{h+1}0_{h)}X_{h-1}X_{h-2}\ldots X_1X_0$

…

node (n-h+1)+(n+h+1)(n+1-h+1)/2*
= $1_{n-1}1_{n-1}\ 0_{n-2}\ldots 0_{h+1}0_h X_{h-1}X_{h-2}\ldots X_1X_0$

We also give a simple example in this section to explain the operations of the algorithm LB_Tree_UB_Embedding( ) when the faulty nodes exist. For the IEH $G_3(15)$ as figure 10.

1. If the root *0000* is faulty, it visits the other root *0001*, to check whether it is free or not. If it is, it terminates.
2. If not, backtrack to the root *0000* .
3. If the node *0000* is faulty, it visits or signals the node *0100*, to check whether it is free or not. If it is, it terminates.
4. If not, insert the node *0100* to the queue, and search the node *1000*, to check whether it is free or not. If it is, it terminates.
5. If not, insert the node *1000* to the queue, and remove the node *0100* from the queue, search the node *0101*, to check whether it is free or not. If it is, it terminates.
6. If not, search the node *0110*, to check whether it is free or not. If it is, it terminates.
7. If not, remove the node *1000* from the queue, search the node *1001*, to check whether it is free or not. If it is, it terminates.
8. If not, search the node *1010*, to check whether it is free or not. If it is, it terminates.
9. If not, search the node *1100*, to check whether it is free or not. If it is, it terminates.
10. If not, return("Failure").

Therefore, the whole searching path is listed as {*(0001), (0100), (1000), (0101), (0110), (1001), (1010), (1100)*}.

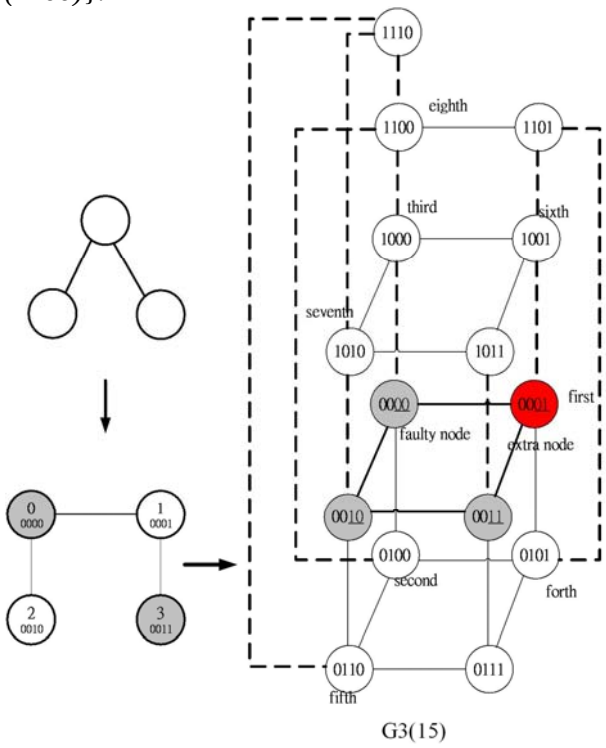

Figure 10: T_2 can be embedded in faulty $G_3(15)$

Theorem 3 A complete binary tree (T_h) can be embedded in faulty IEH $G_n(N)$ graph with dilation *4*, expansion *n*, congestion *1*, and load *1*.

Proof. Every searching path is only one path according to the algorithm LB_Tree_UB_ Embedding( ), allowing us to obtain congestion *1* and load *1*. Herein, we allow *N*-expansion to obtain the replace node of the faulty node. Finding the replace node in that dilation become *2+2=4* in worst case by algorithm LB_Tree_UB_Embedding( ). And these nodes and links of searching paths are not replicated from algorithm LB_Tree_UB_ Embedding( ). These costs associated with graph embedding are dilation *4*, congestion *1*, and load *1*.

Theorem 4 A searching path of algorithm LB_Tree_UB_Embedding( ) is including *[(n-h+1)+(n+h+1)*(n+1-h+1)/2]* nodes.

Proof. By theorem *2* we can embed complete binary tree(T_h) in $G_n(N)$ with *N*-expansion. If node is faulty, we can change a bit in the binary string sequence form bit *h* to bit *n* and insert its corresponding node

into the queue. In the worst case, we can get $n-h+1$ different node. Then we remove the node from the queue. From the first node we can change a bit in the sequence from bit *0* to bit h-1, and we get h different nodes. Until the queue is empty, the sum of all searched node is $(n-h+1)+[h+(h+1)+ \ldots +(n+1)] = [(n-h+1)+(n+h+1)*(n+1-h+1)/2]$ nodes.

Theorem 5 There are $O(n^2-h^2)$ faults, which can be tolerated.

Proof. It is trivial by theorem *4*.

Theorem 6 Our results for the embedding methods are optimized mainly for balancing the processor and communication link loads.

Proof. Because these nodes and edges of searching paths are not replicated from the algorithm LB_Tree_UB_Embedding( ) and load 1, this observation implies that the primary optimization objective of embedding is minimize the inter processor communication cost and to balance the workload of processors having reached.

5 Conclusions

In this paper, we consider the algorithm LB_Tree_Embedding for a complete binary tree can be embedded in an IEH. Considering embedding of complete binary tree in a faulty IEH, allowing N-expansion shows that up to $O(n^2-h^2)$ faults can be tolerated. The main result of this paper is that it is always possible to give solutions to the embedding of complete binary trees in a faulty IEH graph with N-expansion. The costs associated with graph embedding in our strategies of reconfiguration are dilation *4*, congestion *1* and load *1*. By the result, we can embed the parallel algorithms developed by the structure of complete binary tree in an IEH. These methods of reconfiguring enable extremely high-speed parallel computation. Therefore, we can easily port the parallel or distributed algorithms developed for these structures to the IEH graphs and hypercubes.

Reference

[1] S. B. Akers, and B. Krishnamurthy, A Group-Theoretic Model for Symmetric Interconnection Networks, *IEEE Trans. on Computers*, Vol. 38, 1989, pp. 555-565.

[2] D. P. Bertsekas and J. N. Tsitsiklis, "Parallel and Distributed Computation: numerical methods," Prentice Hall, Englewood Ciffs, New Jersey, 1989.

[3] C.-C. Chen, Dynamic Reconfiguration of Complete Binary Trees in Faulty Hypercubes, *Journal of Information Science and Engineering*, Vol. 21, No. 1, 2005, pp. 195-207.

[4] K. Efe, "Embedding large Complete Binary Trees in Hypercubes with load balancing," *Journal of Parallel and Distributed Computing*, Vol. 35, 1996, pp. 104-109.

[5] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," *ACM Theory of Computing*, 1987, pp. 274-284.

[6] F. T. Leighton, *Introduction to parallel algorithms and architectures: Arrays, Trees, Hypercubes*, MORGAN KAUFMANN PUBLISHERS, Inc., 1992.

[7] J.-C. Lin, Load Balancing and Embedding Rings in Faulty Incrementally Extensible Hypercubes, *WSEAS Transactions on Computers*, Vol. 5, 2006, pp. 1867-1872.

[8] J.-C. Lin and H.-C. Keh, Reconfiguration of Complete Binary Trees in Full IEH Graphs and Faulty Hypercubes," *International Journal of High Performance Computing Applications*, Vol. 15, No.1, 2001, pp. 47-55.

[9] J.-C. Lin**,** S. K.C. Lo, S.-J. Wu, and H.-C. Keh, Distributed Fault-Tolerant embeddings of rings in Incrementally Extensible Hypercubes with Unbounded Expansion, *Tamkang Journal of Science and Engineering*, Vol. 9, No. 2, 2006, pp. 121-128.

[10] J.-C. Lin, Faulty-Avoiding Methods for Mapping Meshes in an IEH, *WSEAS Transactions on Computers*, Vol. 6, No.6, 2007, pp. 888-893.

[11] Y. Saad, and M. Schultz, Topological properties of Hypercube, *IEEE Trans. on Computers*, Vol. 37, 1988, pp. 867-871.

[12] C. Seitz, "The Cosmic Cube," *Commun. ACM*, Vol. 28, pp. 22-33, 1985.

[13] H. Sullivan, T. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," *Proc. 4th Symp. Computer Architecture, ACM*, pp. 105-177, March 1977.

[14] S. Sur, and P. K. Srimani, Incrementally Extensible Hypercube Networks and Their Fault Tolerance, *Mathematical and Computer Modeling*, Vol. 23, 1996, pp. 1-15.

[15] S. Sur, and P. K. Srimani, IEH graphs: A

novel generalization of hypercube graphs, *Acta Informatica*, Vol. 32, 1995, pp. 597-609.

[16] S.-H. Wang, Y.-R. Leu and S.-Y. Kuo, "Distributed Fault-Tolerant Embedding of Several Topologies in Hypercubes," *Journal of Information Science and Engineering*, Vol. 20, No. 4, pp. 707-732, 2004.

[17] A.Y. Wu, "Embedding of tree networks into Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 2, pp. 238-249, 1985.

[18] P.-J. Yang, S.-B. Tien, and C.S. Raghavendra, "Embedding of Rings and Meshes onto Faulty Hypercube Using Free Dimensions," *IEEE Trans. on Computers*, Vol. 43, No. 5, pp. 608-618, 1994.

[19] I. Zelina, P. Pop, C. P. Sitar, and I. Tascu, A parallel algorithm for interpolation in Pancake graph, *Proceedings of the 6th WSEAS International Conference on SOFTWARE ENGINEERING, PARALLEL and DISTRIBUTED SYSTEMS (SEPADS '07)*, 2007, pp.98-101.