On Fault-Tolerant Embedding of Meshes and Tori in a Flexible Hypercube with Unbounded Expansion

*JEN-CHIH LIN[1]
[1]Department of Digital Technology Design,
National Taipei University of Education,
No.134, Sec. 2, Heping E. Rd., Da-an District, Taipei City 106,
TAIWAN
E-mail:*yachih@tea.ntue.edu.tw http://tea.ntue.edu.tw/~yachih

Abstract: - The Flexible Hypercubes are superior to hypercube in terms of embedding a mesh and torus under faults. Therefore, this paper presented techniques to enhance the novel algorithm for fault-tolerant meshes and tori embedded in Flexible Hypercubes with node failures. The paper demonstrates that $O(n^2-\lfloor log_2 m \rfloor^2)$ faults can be tolerated and the algorithm is optimized mainly for balancing the processor and communication link loads. Also, the methodology is proven and an algorithm is presented to solve them. These existent parallel algorithms on mesh or torus architectures to be easily transformed to or implemented on the Flexible Hypercube architectures with load *1*, congestion *1*, dilation *3*, and unbounded expansion. The useful properties revealed and the algorithm proposed in this paper can find their way when the system designers evaluate a candidate network's competence and suitability, balancing regularity and other performance criteria, in choosing an interconnection network. Therefore, we can easily port the parallel or distributed algorithms developed for these structuring of mesh and torus to the Flexible Hypercube.

Key-Words: - Flexible Hypercube, hypercube, mesh, torus, fault-tolerance, embedding

1 Introduction

Parallel architectures and algorithms are essential in high-speed computing. As generally known massive parallelism can solve many related problems. To effectively resolve such problems, massively parallel computers require new architectures and algorithms to fully exploit this technology. For available algorithms on one architecture to be easily transformed in or implemented on another architecture, a new field deals with embedding among different networks.

Among the various interconnection networks that have been studied and built[12, 13, 15, 18, 19, 22, 26, 27], *hypercube*[3, 5, 10, 20, 28] networks have received much attention. Hypercube multiprocessors have recently offered a cost effective and feasible approach to supercomputing through parallelism at the processor level by directly connecting a large number of low-cost processors with local memories which communicate by message-passing instead of shared variables. This attention is mainly due to the hypercube advantages of rich interconnection, routing simplicity, and embedding capabilities. 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. In order to conquer the difficulties associated with hypercubes and these generalizations of the hypercubes, the *Flexible Hypercube*[8] has been proposed during past years. The Flexible Hypercube unlike both the *supercube*[23, 24] and the hypercube, may be expanded (or designed) in a number of possible configurations while guaranteeing the same basic fault-tolerant properties and without a change in the communication. The existence of hypercube subgraphs in the Flexible Hypercube ensures that hypercube embedding algorithms developed for the hypercube may also be utilized in the Flexible Hypercube. The flexibility in node placement may possibly be utilized to aid in supporting a specific embedding. The Flexible Hypercube, while maintaining the *fault-tolerance* of the other topologies and the ease of communication, allows the placement of new nodes at any currently unused addresses in the system. An effective means of achieving faulty-tolerance in hypercubes is to introduce spare nodes or links[7]. In doing so, the hypercube structure can still be maintained when nodes fail. In addition to that this approach can be expensive, hardware modifications on machines already in the market place are extremely difficult.

Using the unused nodes as spares (instead of adding extra nodes or links to alter the structure of a hypercube) is another approach to exploit the inherent redundant nodes or links in a hypercube[9]. In this study, we consider this second type of fault-tolerance design only in a faulty Flexible Hypercube.

In a multiprocessor system, two faulty models defined in [9] are adopted herein. 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 as well; this is the total faulty model. This study proposes the partial faulty model, in which the communication links are well when the computation nodes are faulty. In addition, only the faulty node is remapped.

The faulty model proposed herein is a partial model. That is, the communication links are well when the computation nodes are faulty,. Only the faulty node is remapped. This study largely focuses on a theoretical question associated with the simulation of mesh or torus in a faulty Flexible Hypercube.

The problem of allocating the subprocessors, structured by a mesh (or a torus)[6, 30], to processors in a given interconnection network will be reduced to the problem of embedding the torus structure. In order to execute a parallel program, tasks are to be mapped into processors of a parallel machine. It is possible to model this kind of problem in graph-theoretical terms of graph embedding. We model both the parallel algorithms and the parallel machines as graphs.

The power of a message-passing parallel computer depends on the topology chosen for underlying interconnection network, which can be modeled as undirected graph[4]. Different parallel architectures may require different algorithms to solve the same problem efficiently. In order for existent algorithms on one architecture to be easily transformed to or implemented on another architecture, a new field deals with embedding among different networks. Different graphs have been proposed as static interconnection topology for multiprocessors. They include linear arrays, rings, meshes, complete binary trees mesh of trees, de Bruijn networks, and so on. Therefore, we model both the parallel algorithm and the parallel machine as graphs. Given two graphs, $G(V,E)$ and $G'(V', E')$, embedding the guest graph G into the host graph G' maps each vertex in the set V into a vertex (or a set of vertices) in the set V' and each edge in the set E into an edge (or a set of edges) in the set E'. Let these nodes in a graph correspond to processors and edges to communication links in an interconnection network. 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. Four costs associated with graph embedding [1, 11, 14, 25, 29] are *dilation*, *expansion*, *load* and *congestion*. The maximum amount that we must stretch any edge to achieve an embedding is called the dilation of the embedding. By expansion, we mean the ratio of the number of nodes in the host graph to the number of nodes in the graph that is being embedded. The congestion of an embedding is the maximum number of edges of the guest graph that are embedded using any single edge of the host graph. The load of an embedding is the maximum number of nodes of the guest graph that are embedded in any single node of the host graph. 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.

Among the static interconnection networks used for SIMD[21] computers with an array of processors[2], one of the oldest and very popular architectures is a two-dimensional-mesh. Many important algorithms for solving various problems, e.g., matrix operations, simultaneous linear equations, graph-theoretic and image processing problems, etc., have been efficiently embedded in this mesh architecture. Furthermore, we study reconfiguration of task graphs that are embedded in Flexible Hypercubes and study run-time fault-tolerance. Upon failure, our goal is to maintain the structure of a task graph via reconfiguration by using unused nodes in the Flexible Hypercube. In other words, we remap the faulty nodes' tasks to some other fault-free nodes. In order to keep the recovery time small, the reconfiguration must be efficient; that is, we would like to keep data/task movements between nodes that are close by.

The efficiency of a reconfiguration scheme is strongly affected by how tasks are initially mapped to a parallel computer. If a task graph (representing the task) is embedded in a proper way, the reconfiguration scheme can be simple and involve only local movements. Such initial embeddings, called fault-tolerant embedding, however, require more nodes than embeddings with no fault tolerance. Thus, the idea of fault-tolerant embedding is to

leave some spare nodes intentionally in the initial embedding such that, when faults occur, the faulty nodes can be quickly replaced by nearby spare nodes. The main design issue of fault-tolerant embedding is how to distribute the spare nodes and minimize their number such that more faults can be tolerated. In this investigation, we discuss our embedding function with unbounded expansion, congestion *1*, dilation *3*, load *1*.

Also, we developed the methods for finding meshes or tori in a Flexible Hypercube. As the result, we can transit the parallel algorithms developed under the structure of meshes or tori to the Flexible Hypercube. This embedding approach enables extremely high-speed parallel computation in Flexible Hypercubes. Although Flexible Hypercubes are not absolutely asymmetric, it has the same power as the hypercube in terms of meshes and tori.

The embedding of one interconnection network in another is a very important issue in the design and analysis of parallel algorithms. The rest of this paper is organized as follows. Section 2 introduces the necessary notations and definitions. Section 3 presents how to map a mesh in a Flexible Hypercube. Section 4 presents the embedding of a mesh in a faulty Flexible Hypercube with unbounded expansion. Conclusions are finally made in section 5.

2 Preliminaries

We briefly describe these definitions of these topologies of the hypercube, the mesh network, and the flexible hypercube.

A hypercube H_n of order n, is defined to be a symmetric graph $G= (V, E)$ where V is the set of 2^n vertices, each representing a distinct n-bit binary number and E is the set of symmetric edges such that two nodes are connected by an edge iff the number of positions where the bits differ in the binary labels of the two nodes is *1*.

There are many topologies can be mapped in hypercubes or hypercube-like computers. One of these is mesh network. It is very popular network interconnection. One of the most attractive properties of the binary n-cube topology is that meshes of arbitrary dimensions can be mapped in it. This is one of the main reasons for the success of hypercube architectures. Because of these, we consider the mesh size in each direction is a power of *2*. The figure 1 and the figure 2 show us two examples. First example, a 2×2 *2*-dimensional mesh has *4* nodes which are bi-directional connection between two nodes. Second example, A $2^2 \times 2^1$ *2*-dimensional mesh has *8* nodes which are bi-directional connection between two nodes.

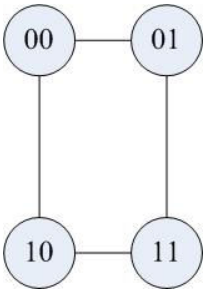

Fig. 1 A 2×2 *2*-dimensional mesh

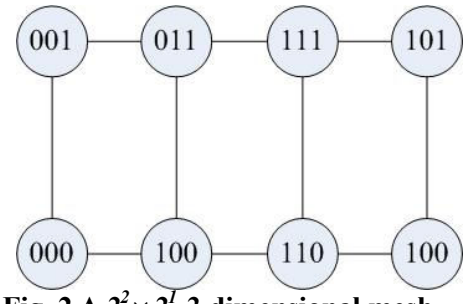

Fig. 2 A $2^2\times 2^1$ *2*-dimensional mesh

The Flexible Hypercube is constructed by any number of nodes and based on a hypercube. A Flexible Hypercube, denoted by FH_N, is defined as an undirected graph $FH_N=(V,E)$,where V is the set of processors (called nodes) and E is the set of bidirectional communication links between the processors (called edges). In an n-dimensional Flexible Hypercube with N nodes where $2^n \le N < 2^{n+1}$ (n is a positive integer), each node can be expressed by an $(n+1)$-bit binary string $i_n.....i_0$ where

$i_p \in \{0, 1\}$ and $0 \le p \le n$

Definition 1[17] A $(2^{n+1}-t)$-node Flexible Hypercube is a lack of t nodes, which are referred to herein as virtual nodes. For any virtual node y, denoted as $I(x)$ where x is any node of Flexible Hypercube, if the function $I(x)$ exists, then $x_n = \overline{y_n}$ and $x_i = y_i$ for $0 \le i \le n-1$

Definition 2[16] The Hamming distance between two nodes with labels $x=x_{n-1}x_{n-2}...x_0$ and $y=y_{n-1}y_{n-2}...y_0$ 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[17] Let $x=x_{n-1}...x_0$, $y=y_{n-1}...y_0$, then $Dim(x, y)=\{i \text{ in } (0...n-1) \mid x_i \neq y_i\}$

Definition 4[17] For any two nodes x and y in a Flexible Hypercube, let $x=x_n \ldots x_0$, $y=y_n \ldots y_0$, then $Dim(x, y)=\{i \text{ in } (0 \ldots n) \mid x_i \neq y_i\}$.

Definition 5[8] Suppose $FH_N=(V, E)$ is an n-dimensional Flexible Hypercube ,then the node sets H_1, H_2, V_1, V_2, V_3 are defined as follows

1. $H_1 = \{x \mid x \in V \text{ and } x_n = 0\}$,
2. $H_2 = \{x \mid x \in V \text{ and } (x_n = 1 \text{ or } I(x) \notin V)\}$,
3. $V_1 = H_1 - H_2$
4. $V_2 = H_1 \cap H_2$
5. $V_3 = H_2 - H_1$

Definition 6[8] Suppose $FH_N=(V, E)$ is an n-dimensional Flexible Hypercube ,then the edge set E is the union of E_1, E_2, E_3, and E_4 ,where

1. $E_1 = \{(x, y) \mid x, y \in H_1 \text{ and } HD(x, y) = 1\}$,
2. $E_2 = \{(x, y) \mid x, y \in V_3 \text{ and } HD(x, y) = 1\}$,
3. $E_3 = \{(x, y) \mid x \in V_3, y \in V_1 \text{ and } HD(x, y) = 1\}$,
4. $E_4 = \{(x, y) \mid x \in V_3, y \in V_2 \text{ and } HD(x, y) = 2\}$.

Addressing of nodes in a Flexible Hypercube is constructed as follows. As discussed above, addresses consist of binary strings of $(n+1)$-bits. The first 2^n-1 addresses correspond to nodes in H_1 and must be the binary representations of 0 through 2^n-1. Each of the remaining nodes (up to 2^n-1 nodes) in the set $V_3=H_2-H_1$ may be placed adjacent to any node x in H_1 and is given the addressing $I(x)$. Any node in H_1 is a hamming distance of 1 from at most one node in V_3. This method of node addressing effectively relaxes the constraint that all nodes in the network must be numbered consecutively. This is unique among the hypercube topologies mentions above. Notably, hypercubes are special cases of a Flexible Hypercube; it can also be expanded flexibly with respect to the placement of new nodes in the system while maintaining fault-tolerant. When a new node is added to a Flexible Hypercube system, $(n+1)$ new connections should be added and at most n existing edges must be removed.

An inevitable consequence of the flexible of construction and the fault-tolerant of a Flexible Hypercube is an uneven distribution of the utilized communication ports over system nodes. Although the Flexible Hypercube loses its property of regularity, more links help obtain the replacement nodes of the faulty nodes of the Flexible Hypercube. The Flexible Hypercube with 12-node is shown in the figure 3. In the figure 3, $H_1 = \{0000, 0001, 00010, 0011, 0100, 0101, 0110, 0111\}$, $H_2 = \{0000, 0001, 0011, 0110, 1010, 1100, 1101, 1111\}$, $V_1 = \{0010, 0100, 0101, 0111\}$, $V_2 = \{0000, 0001, 0011, 0110\}$, and $V_3 = \{1010, 1100, 1101, 1111\}$

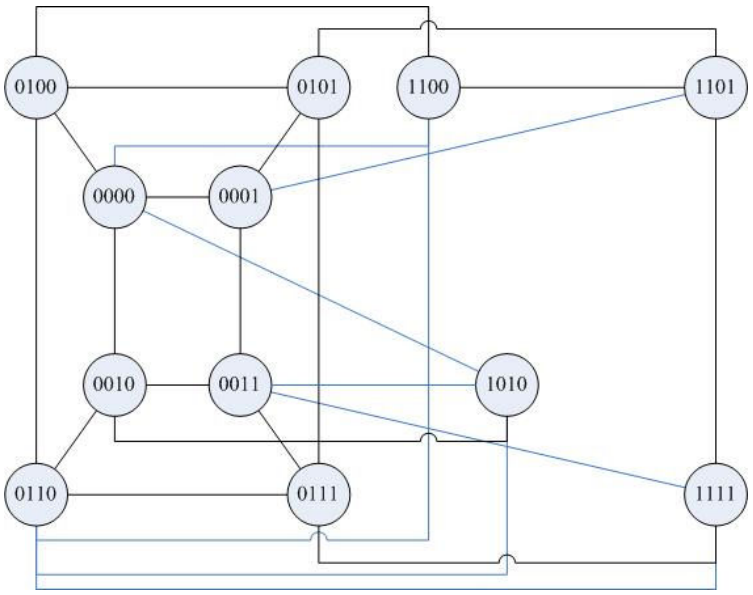

Fig. 3 The Flexible Hypercube contains 12 nodes

Definition 7[1] If *G* is a graph, the vertex set of *G* is denoted by *V* and the edge set of *G* is denoted by *E*. A graph *G'* is said to be a subgraph of *G* if $V' \subseteq V$ and $E' \subseteq E$.

Definition 8[17] Any $m_1 \times m_2$ mesh or torus, denoted by $M_{m_1 \times m_2}$, is a *2*-dimensional mesh or torus, where $m_1 = 2^r, m_2 = 2^s$. □

Definition 9[17] Any $m_1 \times m_2 \times \cdots \times m_d$ mesh or torus, denoted by $M_{m_1 \times m_2 \times \cdots \times m_d}$, in the *d*-dimensional space R_d, where $m_i = 2^{p_i}$. □

Definition 10[11] The *Binary-Reflected Gray Code* (BRGC) is defined recursively as follows.

$C_{n+1}=\{0C_n, 1(C_n)^R\}$, where $C_1=\{0, 1\}$ and $C_2=\{0C_1, 1(C_1)^R\}$

For example, a *2*-bit Gray Code can be constructed by the sequence, defined in definition 10, and insert a cipher in front of each codeword in C_1, then insert an one in front of each codeword in $(C_1)^R$. We get the code $C_2=\{00, 01, 11, 10\}$. Now, we can then repeat the procedure to built a *3*-bit Gray Code, and also get the code $C_3=0C_2 \cup 1(C_2)^R=\{000, 001, 011, 010, 110, 111, 101, 100\}$.

3 Mesh and Torus Embedding

The section describes the representation used to solve that embeds a mesh and torus in a FH_N.

Lemma 1 Any $m_1 \times m_2$ mesh or torus, denoted by $M_{m_1 \times m_2}$, is a *2*-dimensional mesh or torus, where $m_1 = 2^r, m_2 = 2^s$ can be embedded in an *n*-dimensional hypercube where $n = r+ s$.

Proof: A binary number *M* of any node of the *n*-dimensional hypercube can be regarded as consisting of two parts: its first *r* bits and its last *s* bits, which we write in the form $M = \alpha_1\alpha_2...\alpha_r\beta_1\beta_2...\beta_s$, where α_i and β_i are bits *0* or *1*. It is clear from the definition of *n*-dimensional hypercube that when the last *s* bits are fixed, then the resulting 2^{p_1} nodes form a *p1*-dimensional hypercube (with $p1 = r$). Whenever we fix the first *r* bits we obtain a *p2*-dimensional hypercube (with $p2 = s$). The embedding then becomes clear. Choosing a *r*-bit *BRGC* for the *x* direction and *s*-bit *BRGC* for the *y* direction, the point (x_i, y_i) of the mesh is assigned to the node $\alpha_1\alpha_2...\alpha_r\beta_1\beta_2...\beta_s$ where $\alpha_1\alpha_2...\alpha_r$ is the *r*-bit *BRGC* for dimension of p_1 while $\beta_1\beta_2...\beta_s$ is the *s*-bit *BRGC* for dimension of p_2. Therefore, any $m_1 \times m_2$ mesh or torus can be embedded in an *n*-dimensional hypercube where $n = r+ s$.

Lemma 2 Any $m_1 \times m_2 \times \cdots \times m_d$ mesh or torus, denoted by $M_{m_1 \times m_2 \times \cdots \times m_d}$, in the *d*-dimensional space R_d, where $m_i = 2^{p_i}$ can be embedded in an *n*-dimensional hypercube where $n = p_1 + p_2 + ... + p_d$. The numbering of the mesh or torus nodes is any numbering such that its restriction to each i_{th} variable is a Gray code which is described in definition 6. Note that the assumption that all m_i's be power of *2*.

Proof: It is clear from the definition of *n*-dimensional hypercube. A binary number *M* of any node of the *n*-dimensional hypercube can be regarded as consisting of *d* parts: its first part is log_2m_1 bits, its second part is log_2m_2 bits and so on. The numbering of the mesh or torus nodes is any numbering such that its restriction to each i_{th} variable is a Gray code which is described in definition 6. Note that the assumption that all m_i's be power of *2*. Therefore, generalizations to higher dimensions are straightforward and one can state the above lemma 1. Any $m_1 \times m_2 \times \cdots \times m_d$ mesh or torus (with $m_i = 2^{p_i}$) can be embedded in an *n*-dimensional hypercube where $n = p_1 + p_2 + ... + p_d$.

Our proposition is best illustrated by an example. Consider a *2*-dimensional 4×4 mesh i.e., $d = 2$, $p_1 = 2$, $p_2 = 2$, $n = p_1 + p_2 = 4$. A binary number *M* of any node of the *4*-dimensional hypercube can be regarded as consisting of two parts: its first *2* bits and its last *2* bits, which we write in the form

$M = \alpha_1\alpha_2\beta_1\beta_2$, where α_i and β_i are bits *0* or *1*. It is clear from the definition of *n*-dimensional hypercube that when the last *2* bits are fixed, then the resulting 2^{p_1} nodes form a *p1*-dimensional hypercube (with $p1 = 2$). Whenever we fix the first *2* bits we obtain a *p2*-dimensional hypercube. The embedding then becomes clear. Choosing a *2*-bit *BRGC* for the *x* direction and *2*-bit *BRGC* for the *y* direction, the point (x_i, y_i) of the mesh is assigned to the node $\alpha_1\alpha_2\beta_1\beta_2$ where $\alpha_1\alpha_2$ is the *2*-bit *BRGC* for dimension of p_1 while $\beta_1\beta_2$ is the *2*-bit *BRGC* for dimension of p_2.

The binary node number of any mesh node is obtained by concatenation its binary *x* coordinate and its binary *y* coordinate. Therefore, if we call the Gray code any subcode of a *BRGC*, we observe that

any column of mesh nodes forms a Gray code and any row of mesh nodes forms a Gray code. Thus, we will refer to the codes defined above as *2-D* Gray codes. Generalizations to higher dimensions are straightforward and one can state the above lemma 2.

Lemma 3 For any given *N*, a Hypercube H_n must be a subgraph of a Flexible Hypercube FH_N, where $2^n \le N < 2^{n+1}$.

Proof: A FH_N must contain a hypercube H_n. That is trivially by the generation schema of a FH_N graph. It must contain the maximum hypercube H_n.

The embedding approach that a $M_{m_1 \times m_2 \times \cdots \times m_d}$ mesh or torus can be embedded in a FH_N is as follows.

Embedding approach

$M_{m_1 \times m_2 \times \cdots \times m_d}$ ($m_i = 2^{p_i}$),

FH_N ($2^n \le N < 2^{n+1}$),

$\forall p_1 + p_2 + \ldots + p_d = w, w \le n,$ $FH_N = G(V, E)$

$p_1, p_2, \ldots, p_d \ge 1$

$M_{m_1 \times m_2 \times \cdots \times m_d} = G(V', E')$,

$v \in V$ $v' \in V'$ (Denoted by unique binary string)

$v = X_n \ldots X_{w-1} X_{w-2} \ldots X_1 X_0$

$v' = X_{w-1} X_{w-2} \ldots X_1 X_0$

$v' \in V'$ can be embedded in V denote as $v = 0 \ldots 0 X_{w-1} X_{w-2} \ldots X_1 X_0$ □

Theorem 1 Any $M_{2^r \times 2^s}$ *2*-dimensional mesh or torus can be embedded in a FH_N where $r + s = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1*, and expansion *2*.

Proof: By lemma 3, a Hypercube H_n must be a subgraph of a Flexible Hypercube FH_N . A FH_N must contain a maximum hypercube H_n, where $n=log_2N$. By lemma 1, any $m_1 \times m_2$ mesh or torus can be embedded in an *n*-dimensional hypercube where $n = r+ s$.

Therefore, any $M_{2^r \times 2^s}$ *2*-dimensional mesh or torus can be embedded in a FH_N where $r + s = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1*, and expansion *2* by the above embedding approach.

Theorem 2 Any $M_{m_1 \times m_2 \times \cdots \times m_d}$ *d*-dimensional mesh or torus, where $m_i = 2^{p_i}$ can be embedded in a FH_N, where $p_1 + p_2 + \ldots + p_d = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1* and expansion *2*.

Proof: By lemma 3, a Hypercube H_n must be a subgraph of a Flexible Hypercube FH_N . A FH_N must contain a maximum hypercube H_n, where $n=log_2N$. By lemma 2, any $m_1 \times m_2 \times \cdots \times m_d$ mesh or torus can be embedded in an *n*-dimensional hypercube where $n = p_1 + p_2 + \ldots + p_d$. Therefore, any $M_{m_1 \times m_2 \times \cdots \times m_d}$ *d*-dimensional mesh or torus, where $m_i = 2^{p_i}$ can be embedded in a FH_N, where $p_1 + p_2 + \ldots + p_d = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1* and expansion *2* by the above embedding approach.

This is the best illustrated by an example in figure 4. That is a $2^1 \times 2^1$ mesh (with *4* nodes) can be embedded in a FH_{12}.

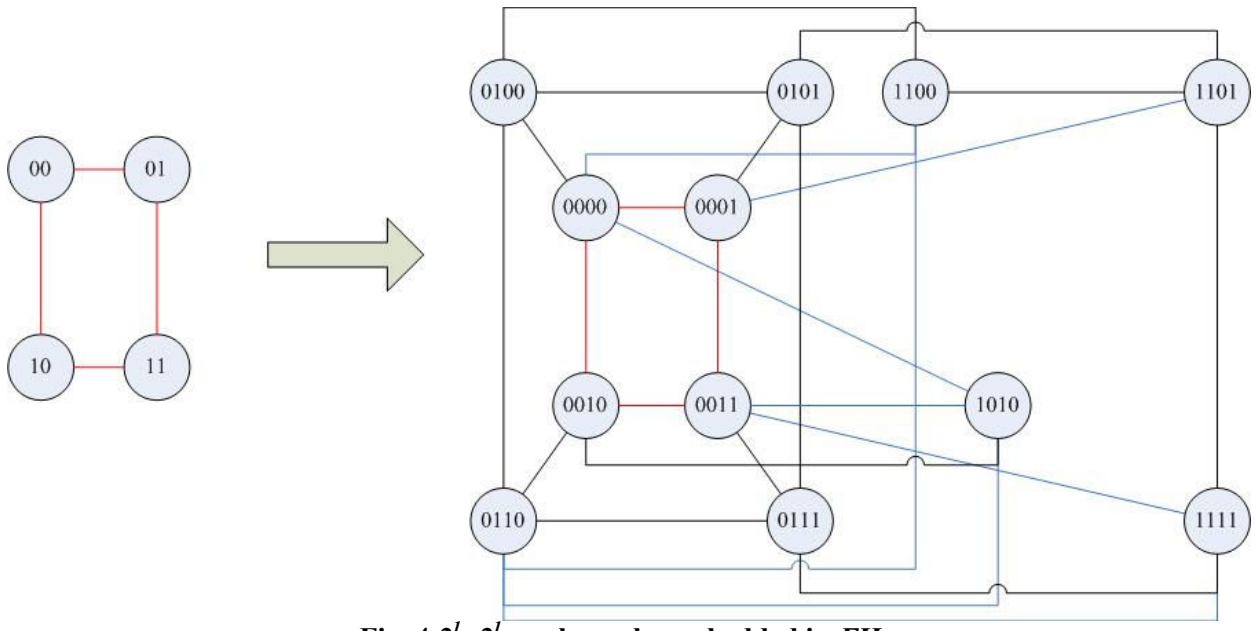

Fig. 4 $2^1 \times 2^1$ mesh can be embedded in FH_{12}

4 Fault-Tolerant Embedding with Unbounded Expansion

In the previous section, we have constructed a mesh and a torus in a FH_N graph. In the section, we consider a faulty FH_N with unbounded expansion embedding.

Theorem 3 Any mesh or tori can be embedded in a FH_N graph with unbounded expansion.

Proof: By theorem 1 and theorem 2, any mesh or tori can be embedded in a FH_N graph with unbounded expansion.

Algorithm Mesh_Embedding(*x*)

Input: x /*the faulty node*/,
$M_{m_1 \times m_2 \times \cdots \times m_d}$ ($m_i = 2^{p_i}$), $G_n(N)$ ($2^n \le N < 2^{n+1}$),
$\forall p_1 + p_2 + \ldots + p_d = w, w \le n,$
$p_1, p_2, \ldots, p_d \ge 1$
$FH_N = G(V, E)$,
$M_{m_1 \times m_2 \times \cdots \times m_d} = G(V', E')$,

Output: y /*the replaceable node*/

1. $i=0; j=0; k=0$
2. Create a Queue Q; $Q=\Phi$
3. if a node x is faulty
4. then
5. {
6. while $i < (n+1-\lfloor log_2 m \rfloor)$ do
7. {
8. search the node y
   /* $HD(x, y)=1$, $Dim(x, y)=\lfloor log_2 m \rfloor+i$*/
9. if y is not a virtual node and it is free
10. then
11. return(y) /*replace x with y*/
12. remove all nodes in Q
13. *exit()*
14. else
15. *enqueue(y, $\lfloor log_2 m \rfloor+i$)*
16. *i=i+1*
17. }
18. }
19. while Q is not empty do
20. {
21. *dequeue(a,b)*
22. while $j < b$ do
23. {
24. search the node z
    /* $HD(a, z)=1$, $Dim(a, z)=j$*/
25. if z is not a virtual node and it is free
26. then
27. return(z)
    /*replace x with y*/
28. remove all nodes in Q
29. *exit()*
30. *j=j+1*
31. }
32. }
33. return("Failure")
34. end

Finding the replaceable node as follows:

node 0 = $0X_{n-1}X_{n-2}\ldots X_{\lfloor log_2 m \rfloor}\ldots X_1X_0$

node 1 = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor}\ldots X_1X_0$

node 2= $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor+1}\, X_{\lfloor log_2 m \rfloor}\ldots X_1X_0$

$\vdots$

node $(n-\lfloor log_2 m \rfloor)$ = $0X'_{n-1}X_{n-2}\ldots X_{\lfloor log_2 m \rfloor}\ldots X_1\, X_0$

node $(n-\lfloor log_2 m \rfloor+1)$ = $1X_{n-1}X_{n-2}\ldots X_{\lfloor log_2 m \rfloor}\ldots X_1\ X_0$

node $(n-\lfloor log_2 m \rfloor+2)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor}\ldots X_1X'_0$

node $(n-\lfloor log_2 m \rfloor+3)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor}\ldots X'_1X_0$

$\vdots$

node $(n-\lfloor log_2 m \rfloor+1+\lfloor log_2 m \rfloor)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor}$ $X'_{\lfloor log_2 m \rfloor-1}\ldots X_1X_0$

node $(n-\lfloor log_2 m \rfloor+1+\lfloor log_2 m \rfloor+1)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor+1}\ldots X_1X'_0$

node $(n-\lfloor log_2 m \rfloor+1+\lfloor log_2 m \rfloor+2)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor+1}\ldots X'_1X_0$

$\vdots$

node $(n-\lfloor log_2 m \rfloor+1+2*\lfloor log_2 m \rfloor)$ = $0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor+1}X_{\lfloor log_2 m \rfloor}X'_{\lfloor log_2 m \rfloor-1}\ldots X_1X_0$

node$(n-\lfloor log_2 m \rfloor+1+2*\lfloor log_2 m \rfloor+1)$=$0X_{n-1}X_{n-2}\ldots X'_{\lfloor log_2 m \rfloor+1}X'_{\lfloor log_2 m \rfloor+1}X_{\lfloor log_2 m \rfloor-1}\ldots X_1X_0$

$\vdots$

node $((n-\lfloor log_2 m \rfloor+1)*(\lfloor log_2 m \rfloor+1))+(1+2+\ldots+n-\lfloor log_2 m \rfloor)$ = $1X'_{n-1}X_{n-2}\ldots X_{\lfloor log_2 m \rfloor-1}\ldots X_1X_0$

We give a simple example in this section to explain the operations of the Mesh_Embedding algorithm when the faulty nodes exist. For the FH_{12} as Figure 5, where the $M_{2\times2}$ has been embedded in it.

1. If the node *0* is faulty, it visits or signals the node *4*, to check whether it is free or not. If it is, it terminates.
2. If not, insert the node *4* to the queue, and search the node *8*, to check whether it is free or not. If it is, it terminates.
3. If not, insert the node *8* to the queue, and delete the node *4* from the queue, search the node *5*, to check whether it is free or not. If it is, it terminates.
4. If not, search the node *6*, to check whether it is

free or not. If it is, it terminates.

5. If not, delete the node *4* from the queue, search the node *9*, to check whether it is free or not. If it is, it terminates.
6. If not, search the node *10*, to check whether it is free or not. If it is, it terminates.
7. If not, search the node *14*, to check whether it is free or not. If it is, it terminates.
8. If not, return("Failure").

Therefore, the whole searching path is listed as *{4(0100), ~~8(1000)~~, 5(0101), 6(0110), ~~9(1001)~~, 10(1010), 12(1100)}*.

We illustrate the searching path of finding a replaceable node in a FH_{12} as shown figure 6.

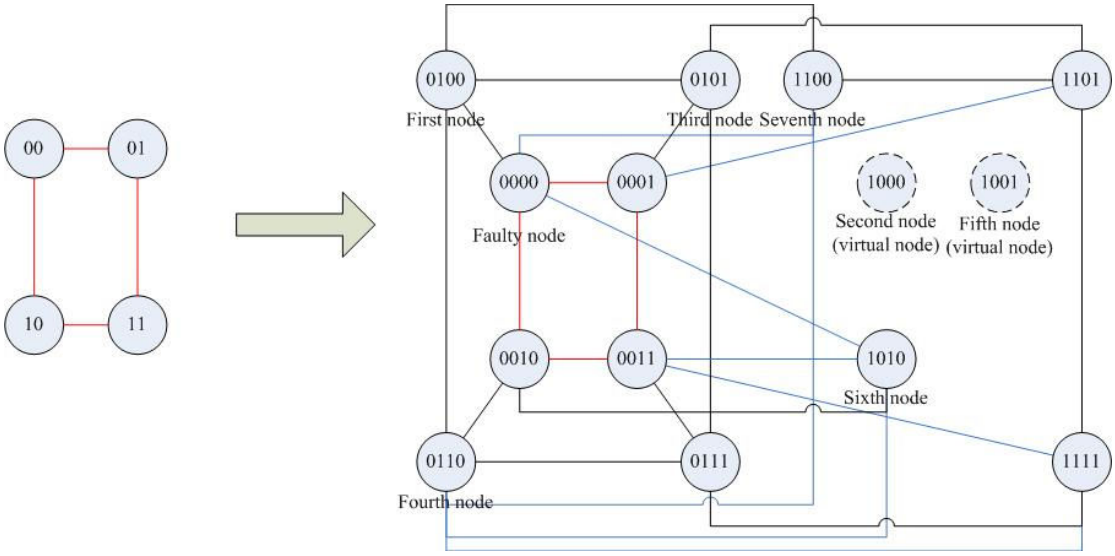

Fig.5 Embedding of a $M_{2\times 2}$ mesh and torus in a FH_{12}

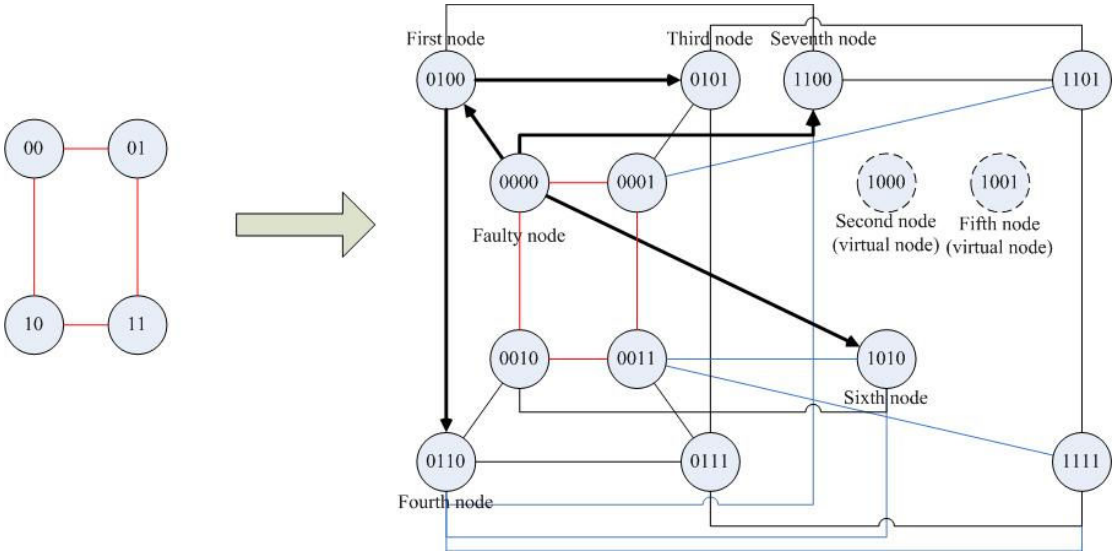

Fig.6 The searching path of finding a replaceable node in a faulty FH_{12}

Theorem 4 Any mesh or torus $M_{m_1\times m_2\times\cdots\times m_d}$ can be embedded in a faulty FH_N with dilation *3*, congestion *1*, load *1*, and unbounded expansion.

Proof: Every searching path is only one path according to the algorithm Mesh_Embedding, allowing us to obtain congestion *1* and load *1*. Herein, we allow unbounded expansion to obtain the replaceable node of the faulty node. When a node is faulty, it is a worse case in which the dilation=*1+2=3* at most by algorithm Mesh_Embedding. Because these nodes and links of searching paths are not replicated from algorithm Mesh_Embedding, These costs associated with graph embedding are dilation *3*, congestion *1*, load *1*, and unbounded expansion.

Theorem 5 A searching path of algorithm Mesh_Embedding is including $1/2*n^2 + 3/2*n-1/2*\lfloor log_2 m \rfloor) - 1/2*\lfloor log_2 m \rfloor^2+1$ nodes.

Proof: We can embed $M_{m_1\times m_2\times\cdots\times m_d}$ in a FH_N by

theorem 4. If a node is faulty, we can change a bit in the binary string sequence from bit $\lfloor log_2 m \rfloor$ to bit n and insert its corresponding node in the queue. In the worst case, we can get $(n-\lfloor log_2 m \rfloor+1)$ different nodes. Then we delete the node from the queue. From the first node we can change a bit in the sequence from bit 0 to bit $(\lfloor log_2 m \rfloor-1)$, and we can get $\lfloor log_2 m \rfloor$ different nodes. We can also change a bit in the sequence from bit 0 to bit $\lfloor log_2 m \rfloor$ from the second node of the queue, and we can also get $(\lfloor log_2 m \rfloor+1)$ different nodes. Until the queue is empty, the sum of all searched nodes is $(n-\lfloor log_2 m \rfloor+1)*(\lfloor log_2 m \rfloor+1))+(1+2+...+n-\lfloor log_2 m \rfloor)$. The search path includes $(n-\lfloor log_2 m \rfloor+1)*(\lfloor log_2 m \rfloor+1))+(1+2+...+n-\lfloor log_2 m \rfloor)$ nodes. That is, the whole searching path includes $(n-\lfloor log_2 m \rfloor+1)*(\lfloor log_2 m \rfloor+1))+(1+2+...+n-\lfloor log_2 m \rfloor)= 1/2*n^2 + 3/2*n -1/2* \lfloor log_2 m \rfloor) – 1/2*\lfloor log_2 m \rfloor^2+1$ nodes.

Theorem 6 There are $O(n^2-\lfloor log_2 m \rfloor^2)$ faults, which can be tolerated.

Proof: By theorem 5, the whole searching path includes $1/2*n^2 + 3/2*n -1/2* \lfloor log_2 m \rfloor) – 1/2*\lfloor log_2 m \rfloor^2+1$ nodes. That is, $O(n^2-\lfloor log_2 m \rfloor^2)$ faults can be tolerated.

5 Conclusion

Hypercubes, meshes, and tori are well known interconnection networks for parallel computing. In this paper, we try to find the replaceable node of the faulty node. This paper proposes novel algorithms of fault-tolerant meshes and tori embedded in the Flexible Hypercube with node failures. The main results obtained (1) these existent parallel algorithms in mesh architectures can be easily transformed to or implemented in FH_N architectures with load *1*, congestion *1*, dilation *3*, and unbounded expansion. (2) A searching path of a Flexible Hypercube is including approximate to $(1/2*n^2 + 3/2*n -1/2* \lfloor log_2 m \rfloor) – 1/2*\lfloor log_2 m \rfloor^2+1)$ nodes. Therefore, there are $O(n^2-\lfloor log_2 m \rfloor^2)$ faults, which can be tolerated. (3) The result implies that simulation of mesh and torus in a faulty Flexible Hypercube for balancing the processor and communication link loads at present. According to the result, we can easily port the parallel or distributed algorithms developed for these structures to the Flexible Hypercube. Therefore, these methods of reconfiguring enable extremely high-speed parallel computation.

References:

[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] J. R. Armstromg and F. G. Gray, Fault-diagnosis in n-Cube array of microprocessor, *IEEE Trans. on Computers*, Vol. C-30, No. 4, 1992, pp. 587-590.

[3] L. Bhuyan and D.P. Agrawal, Generalized Hypercubes and Hyperbus structure for a computer network, *IEEE Trans. on Computers*, Vol. 33, 1984, pp. 323-333.

[4] C. Chartand and O. R. Oellermann, *Applied and Algorithmic Graph Theory,* McGRAW-HILL Inc., 1993.

[5] K. Day and A. E. Al-Ayyoub, Fault Diameter of k-ary n-cube Networks, *IEEE Trans. on parallel and distributed systems*, Vol. 8, No. 9, 1997, pp. 903-907.

[6] Q. Dong, X. Yang, J. Zhao, and Y. Y. Tang, Embedding a family of disjoint 3D meshes into a crossed cube, *Information Sciences*, Vol. 178, No. 11, 2008, pp. 2396-2405.

[7] S. Dutt and J. P. Hayes, An automorphic approach to the design of fault-tolerance Multiprocessor, *Proc. 19th Inter. Symp. on Fault-Tolerant Computing*, 1989.

[8] T. Hameenanttila, X.-L. Guan, J. D. Carothers, and J.-X. Chen, The Flexible Hypercube: A New Fault-Tolerant Architecture for Parallel Computing, *Journal of Parallel and Distributed Computing*, Vol. 37, 1996, pp. 213-220.

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

[10] J. P. Hayes, and T.N. Mudge, Hypercube supercomputing, *Proc. IEEE*, Vol. 77, 1989, pp. 1829-1842.

[11] S. L. Johnson, and C.-T. Ho, "On the conversion between binary code and binary-reflected gray code on binary cubes," *IEEE Trans. on Computers*, Vol. 44, 1995, pp. 47-53.

[12] H. P. Katseff, "Incomplete Hypercubes," *IEEE Trans. on Computers*, Vol. 37, 1988, pp. 604-608.

[13] J. Kuskin, et al., The Stanford FLASH Multiprocessor, *Proceedings of the 21st Annual International Symposium on Computer Architecture*, 1994, pp. 302-313.

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

[15] D. Lenoski, et al., The StanfordDASH Multiprocessor, *Computer*, Vol. 224, 1971, pp. 63-79.

[16] J.-C. Lin, T.-H. Chi, H.-C. Keh and A.-H. A. Liou, Embedding of Complete Binary Tree with 2-expansion in a Faulty Flexible Hypercube," *Journal of Systems Architecture*, Vol. 47, No. 6, 2001, pp. 543-548.

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

[18] C.D. Park, and K.-Y. Chwa, Hamiltonian properties on the class of hypercube-like networks, *Information Processing Letters*, Vol. 91, 2004, pp. 11-17.

[19] F. P. Preparata, and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, Vol. 24, 1981, pp. 300-309.

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

[21] J. L. C. Sanz, *The SIMD Model of Parallel Computation*, Springer-Verlag New-York, Inc., 1994.

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

[23] A. Sen, Supercube: An Optimally Fault Tolerant Network Architecture, *Acta Informatica*, Vol. 26, 1989, pp. 741-748.

[24] A. Sen, A. Sengupta and S. Bandyopadhyay, Generalized Supercube: An incrementally expandable interconnection network, *Proceedings of the Third Symposium on Frontiers of Massively Parallel Computation-Frontiers'90*, 1990, pp. 384-387.

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

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

[27] S. Sur, and P. K. Srimani, IEH graphs: A novel generalization of hypercube graphs, *Acta Informatica*, Volume 32, 1995, pp 597-609.

[28] 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, 2004, pp. 707-732.

[29] C. Xu and F. C. M. Lau, *Load Balancing in Parallel Computers-Theory and Practice*, Kluwer Academic Publishers, Inc., 1997.

[30] 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, 1994, pp. 608-618.