Simulation of Meshes in a Faulty Supercube with Unbounded Expansion

[1] Jen-Chih Lin, *[2] Shih-Jung Wu
[1,] *Department of Digital Technology Design,*
National Taipei University of Education,
E-mail: yachih@tea.ntue.edu.tw
[*2,] *Department of Innovative Information and Technology, Tamkang Universit,*
E-mail: wushihjung@mail.tku.edu.tw

Abstract

Reconfiguring meshes in a faulty Supercube is investigated in the paper. The result can readily be used in the optimal embedding of a mesh (or a torus) of processors in a faulty Supercube with unbounded expansion. There are embedding algorithms proposed in this paper. These embedding algorithms show a mesh with any number of nodes can be embedded into a faulty Supercube with load 1, congestion 1, and dilation 3 such that $O(n^2\text{-}w^2)$ faults can be tolerated, where n is the dimension of the Supercube and 2^w is the number of nodes of the mesh. The meshes and hypercubes are widely used interconnection architectures in parallel computing, grid computing, sensor network, and cloud computing. In addition, the Supercubes are superior to hypercube in terms of embedding a mesh and torus under faults. Therefore, we can easily port the parallel or distributed algorithms developed for these structuring of mesh and torus to the Supercube.

Keywords: *Supercube, Hypercube, Mesh, Torus, Grid Computing*

1. Introduction

From the computational perspective, hypercube [3] multiprocessors have recently offered a cost effective and feasible approach to supercomputing through parallelism at the processor level by directly connection a large number of low-cost processors with local memories which communicate by message-passing instead of shared variables. Therefore, hypercubes are widely used interconnection architectures in parallel computing, grid computing, and cloud computing [5, 18, 20].

The hypercube topology has been used as the basis of several parallel computers since it offers a rich interconnection structure, high data bandwidth, low message latency, and small diameter. Some examples include the Connection Machine from Thinking Machines, Intel iPSC, NCUBE/10, Caltech/JPL, and the Cosmic Cube developed at California Institute of Technology [13]. A hypercube or a binary *n*-cube computer is a multiprocessor characterized by the presence of $N=2^n$ processors interconnected as an *n*-dimensional binary cube. Each processor P_i forms a node (vertex) of the cube and is a self-contained computer with its own CPU and local main memory. P_i has edges (communication links) to *n* other processors (its neighbors), which correspond to the edges of the cube that are connected directly to P_i. 2^n distinct *n*-bit binary addresses or labels may be assigned to the processors so that each processor's address differs from that of each of its neighbors in exactly one bit position. Figure 1 illustrates the hypercube topology for $n<3$; note that a zero-dimensional hypercube is a conventional SISD computer [8].

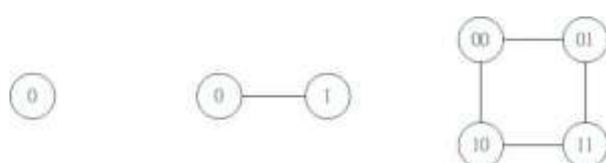

Figure 1. The hypercube topology for $n<3$

A number of two-dimensional mesh-based networks have been proposed, owing to their advantages of scalability, modularity, expandability, and degree boundedness. Commercial multiprocessor products based on the mesh and torus has been announced from Ametek and Intel Scientific Computers. Mesh-based designs have been used in the ILLIAC IV computer, Intel Paragon, Cray T3D, and the

doi: 10.4156/ijact.vol3.issue8.33

Goodyear MPP massively parallel computer. A *mesh connected computer* [8] is easy to construct because it is regular, it has short connections, it requires only four connections per node, and it is possible to build in two dimensions without having any connections cross. Each node that is not on an edge of the array has a direct connection with its four nearest neighbors. At the same time, the top row is connected to the bottom row and leftmost column is connected to the rightmost column, so the interconnections logically form a *torus*. The construction of such a machine in two dimensions requires that some connections cross. The mesh and torus are two of the most important networks for parallel computers. A great deal of research has focused on the mesh and torus networks and several parallel computers have been built with 2- or 3-dimensional mesh or torus topologies. Examples include the CLIP4, the GAPP (NCR Microelectronic Products Division), the MPP (of Goodyear Aerospace), the MP-1 (sold by MASPAR Corporation), and the J-machine is a project at MIT in a *3*-dimensional mesh topology. Mesh connected computers were shown to be efficient in performing many image and matrix operations. If a mesh connected computer can be simulated with a hypercube or a hypercube-derived computer, those same algorithms can be used on these other topologies. In the paper, one of the most important issues in the design of a system which contains many components is the system's performance in the presence faults. Among the static interconnection networks used for SIMD computers with an array of processors, 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. The mapping of a task graph or an algorithm to a parallel architecture is a fundamental problem in parallel computation. It arises in the context of efficiently implementing an algorithm developed for a particular architecture onto another architecture of different topology and size, as well as in the context of allocating processes with dependencies to processors. The objective of a mapping is to minimize execution time. A general approach is to distribute work evenly among the processors and to minimize interprocessor communication. Graph embeddings have been used successfully as models for developing efficient mappings and for understanding the computational equivalence between parallel architectures.

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 *Supercube* [14] has been proposed during past years. The Supercube 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 Supercube ensures that hypercube embedding algorithms developed for the hypercube may also be utilized in the Supercube. The flexibility in node placement may possibly be utilized to aid in supporting a specific embedding. The Supercube, 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 [6]. 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. In this study, we consider this second type of fault-tolerance design only in a faulty Supercube. Load Balancing, communication locality, communication congestion, and node utility in process graphs can be abstractly studies as the problem of embedding. In a process graph, the nodes represent processes comprising a parallel program and the edges represent communications between processes. 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 [1, 6, 19].

The embedding problem is to find embeddings with balanced loads, small dilations, and small congestions. 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 a multiprocessor system, two faulty models defined in 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. Hypercube multiprocessor systems usually have a large number of processors, so the probability that some processor fails cam be high. Fault tolerance in hypercubes has been studied by several researchers, and several interesting techniques have been proposed [2, 3]. The technique discussed in [6] employs hardware redundancy and uses reconfiguration to tolerate faults. Using this approach, the researchers obtain either an *n*-dimensional hypercube or a smaller subcube through reconfiguration. Adding redundant hardware components requites hardware modifications which can be difficult and expensive.

Alternatively, there are some techniques that exploit the inherent redundant nodes and links in hypercube to achieve fault tolerance. The emulation approach is used to simulate the entire hypercube by the residual hypercube. The emulation approach can tolerate multiple faults, however, with constant slowdown (*2* or more) for both computation and communication performance. Another approach to achieve fault tolerance with no extra nodes and/or links is to embed a smaller cube in the faulty hypercube, as in [17]; techniques for embedding task graphs to embed a larger size task graph if the embedding us attempted in the entire faulty hypercube. The motivation for this is to continue execution of tasks on faulty hypercubes, possibly with some performance degradation.

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 Supercube. Efficiently simulating one network on another one requires that these four costs be as minimum as possible. However, for most embedding problems, an embedding can not be obtained that minimizes these costs simultaneously. Therefore, some tradeoffs among these costs must be made. In this investigation, we discuss our embedding function with expansion *2*, congestion *1*, dilation *3*, load *1*. Also, we developed the methods for finding meshes or tori in a Supercube. As the result, we can transit the parallel algorithms developed under the structure of meshes or tori to the Supercube. This embedding approach enables extremely high-speed parallel computation in Supercubes. Although Supercubes 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. In Section 3, the paper presents how to map a mesh in a Supercube. Section 4 presents the embedding of a mesh in a faulty Supercube 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 supercube. For the formal description of an n-dimensional hypercube, it is necessary to define the Cartesian product of graphs as follows.

Definition 1 [1] A graph $G_p=(V_p, E_p)$ is called the Cartesian product of two graphs $G_1=(V_1, E_1)$ and $G_2=(V_2, E_2)$ if two nodes $u=(u_1, u_2)$ and $v=(v_1, v_2)$ are adjacent in G_p if and only if one of the following conditions are true.

(1) $u_1=v_1$ and u_2 adjacent to v_2,

(2) $u_2=v_2$ and u_1 adjacent to v_1.

The Cartesian product of G1 and G2 is denoted by G1×G2.

Definition 2 [11**]** An *n*-dimensional hypercube H_n for $n \geq 2$ can be defined recursively in terms of the graph product operations $\times$ as follows, where H_2 is the complete *2*-nodes graph:
$H_n=H_2 \times H_{n-1}$.

Gray code with the prefix *0*, followed by the *d*-bit Gray code in reverse order with the prefix *1*. Using this technique, we build the following *2*-bit code $C_2=\{00, 01, 11, 10\}$. From this *2*-bit Gray code we generate the following *3*-bit Gray code $C_3=0C_2 \cup 1(C_2)^R=\{000, 001, 011, 010, 110, 111, 101, 100\}$. 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. The paper considers mapping a 2^3*2^2 mesh in a *32*-node hypercube. Two bit positions are reserved for the row and three bit positions are set aside for the column. Let us assume that the first two bit positions are used for the row. The *2*-bit Gray code *{00, 01, 11, 10}* corresponds to a traversal through columns *0*, *1*, *2*, *3*, and *4*. The *3*-bit Gray code *{000, 001, 011, 010, 110, 111, 101, 100}* corresponds to a traversal through rows *0*, *1*, *2*, *3*, *4*, *5*, *6*, and *7*. Hence we have the following mapping of a 2^3*2^2 mesh.

Definition 4[9] 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 5[9] Let $x=x_{n-1}...x_0$, $y=y_{n-1}...y_0$, then $Dim(x, y)=\{i \text{ in } (0...n-1) | \ x_i \neq y_i\}$

The following formal definition of the supercube graph is from [14]. A supercube is constructed by any number of nodes and based on hypercube. A supercube, denoted by S_N, is defined as an undirected graph $S_N=(V, E)$, where *V* is the set of processors (called nodes in our discussion) and *E* is the set of bidirectional communication links between the processors (called edges). Assume that *V* contains *N* nodes and each node can be numbered by an identical number in the range over *(0, N-1)*, in an *(n-1)*-dimensional supercube, each node can be expressed by an *n*-bit binary string because $2^{n-1} \leq N < 2^n$, where *n* is a positive integer.

Definition 6[14] Suppose $S_N = (V, E)$ is an *n*-dimensional supercube, then the node set *V* can be divided into three subsets V_1, V_2, V_3, where

1. $V_3 = \{x \mid x \in V, x = 1u$, *where u is n-bit sequences}.*
2. $V_2 = \{x \mid x \in V, x = 0u$, *1u does not exist in V, where u is n-bit sequences},and*

$V_1 = \{x \mid x \in V, x =0u, 1u \in V$, *where u is n-bit sequences}.*

Definition 7[14] Suppose $S_N = (V, E)$ is an *n*-dimensional supercube, then the edge set *E* is the union of E_1, E_2, E_3 and E_4, where

1. $E_1 =\{(x, y)| x, y \in V, x = 0u, y = 0v$, *where u, v are n-bit sequences and HD(x, y)= 1},*
2. $E_2 =\{(x, y)| x, y \text{ in } V_3, x = 1u, y = 1v$, *where u, v are n-bit sequences and HD(x, y)= 1},*
3. $E_3 =\{(x, y)| x \text{ in } V_3, y \text{ in } V_2, x = 1u, y = 0v$, *where u, v are n-bit sequences and HD(x, y)= 2}, and*

$E_4 =\{(x, y)| x \text{ in } V_3, y \text{ in } V_1, x = 1u, y = 0u$, *where u is (n-1)-bit sequences }.*

The supercube with *12*-node is shown in figure 2. Notably, hypercubes are special cases of a Supercube; 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 Supercube 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 Supercube is an uneven distribution of the utilized communication ports over system nodes. Although the Supercube loses its property of regularity, more links help obtain the replacement nodes of the faulty nodes of the Supercube. The Supercube with *12*-node is shown in the figure 2. In the figure 2, V_1 *= {0000, 0001, 0010, 0011},* V_2 *= {0100, 0101, 0110, 0111}, and* V_3 *= {1000, 1001, 1010, 1011},* E_1 *= {(0000, 0001), (0000, 0010), (0000, 0100), (0001, 0011), (0001, 0101), (0010, 0011), (0010, 0110), (0011, 0111), (0100, 0101), (0100, 0110), (0101, 0111), (0110, 0111},* E_2 *= {(1000, 1001), (1000, 1010), (1001, 1011), (1010, 1011)},* E_3*={(0100, 1000), (0101, 1001), (0110, 1010), (0111, 1011)},* E_4*={(0000, 10000), (0001, 1001), (0010, 1010), (0011, 1011)}.*

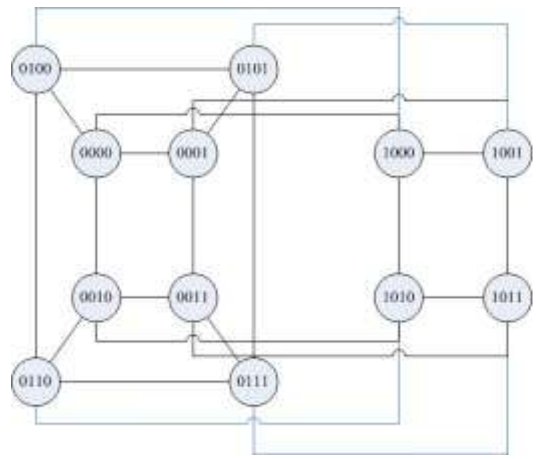

Figure 2. The Supercube contains *12* nodes

Definition 8[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 9[9] 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 10[9] 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 $m_i = 2^{p_i}$

3. Mesh and Torus Embedding

In this section, the paper describes how to embed a mesh and torus in a S_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 $n = r + s$.
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 $n = p_1 + p_2 + \ldots + p_d$. The numbering of the mesh or torus nodes is any numbering such that its restriction to each i_{th} variable is a Gray code. Note that the assumption that all m_i*'s* be power of *2*.

Consider a *2*-dimensional $2^1 \times 2^2$ mesh i.e., $d = 2$, $p_1 = 1$, $p_2 = 2$, $n = p_1 + p_2 = 3$. A binary number M of any node of the *3*-dimensional hypercube can be regarded as consisting of two parts: its first *1* bit and its last *2* bits, which we write in the form $M = \alpha_1\beta_1\beta_2$, 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 p_1-dimensional hypercube (with $p_1 = 1$). Whenever we fix the first *1* bit we obtain a p_2-dimensional hypercube. The embedding then becomes clear. Choosing a *1*-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 $\alpha_1\beta_1\beta_2$ where α_1 is the *1*-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.

The figure 3 shows a *2*-dimensional $2^1 \times 2^2$ torus ($d = 2$, $p_1 = 1$, $p_2 = 2$, $n = p_1 + p_2 = 3$) which are bi-directional connection between nodes.

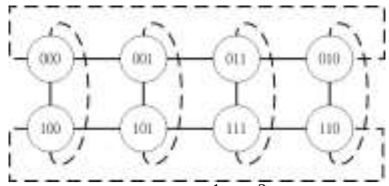

Figure 3. A $2^1 \times 2^2$ torus

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

Proof. A S_N must contain a hypercube H_n. That is trivially by the generation schema of a S_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 S_N is as follows.

Embedding approach:

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

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

$\forall p_1 + p_2 + \ldots + p_d = w, w \le n,$

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

$S_N = G(V, E)$

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

$v \in V \quad v' \in V'$ (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 S_N where $r + s = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1*, and expansion *2*.

Proof. This is trivial by lemma 1 and 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 S_N, where $p_1 + p_2 + \ldots + p_d = w = \lfloor \log_2 N \rfloor$ with load *1*, dilation *1*, congestion *1* and expansion *2*.

Proof. It is trivial by lemma 2 and the above embedding approach.

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

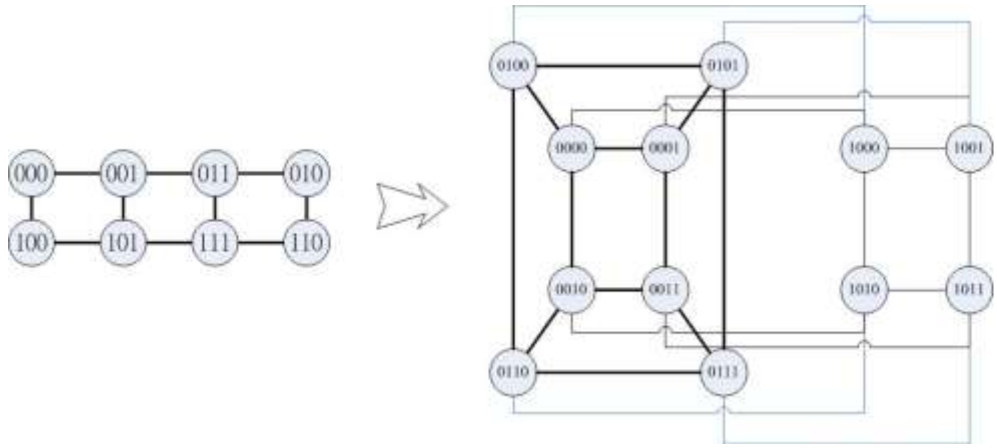

Figure 4. A $2^1 \times 2^2$ mesh can be embedded in S_{12}

Lemma 4 Any mesh or tori contains any number of nodes can be embedded in a S_N graph with load *1*, congestion *1*, and dilation *1*.

4. Fault-Tolerant Embedding with Unbounded Expansion

In the previous section, we have constructed a mesh and a torus in a S_N graph. In the section, we propose a new scheme for a faulty S_N with unbounded expansion embedding.

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

Proof. It is trivial by the embedding approach.

Algorithm Fault-Tolerance_Embedding(*x*):

Input: *x* /*the faulty node*/,

$M_{m_1 \times m_2 \times \cdots \times m_d}$ ($m_i = 2^{p_i}$), $G_n(N)$ ($2^n \le N < 2^{n+1}$),

$\forall p_1 + p_2 + \ldots + p_d = w, w \le n,$

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

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

Output: *y* /*the replaceable node*/

i=0; *j=0*; *k=0*
Create a Queue *Q*; *Q=Φ*
if a node *x* is faulty
then
{
while *i* < *(n+1-w)* do
{
search the node *y*
/* *HD(x, y)=1*, *Dim(x, y)=w+i**/
if *y* is not a virtual node and it is free
/* If the node is an inexistent node, we called a virtual node. */
then
return(*y*) /*replace *x* with *y**/
remove all nodes in *Q*
exit()
else
enqueue(y,w+i)
i=i+1

```
                        }
                }
        while Q is not empty do
        {
                dequeue(a,b)
                while j < b do
                {
                search the node z
            /* HD(a, z )=1, Dim(a, z)=j*/
                if z is not a virtual node and it is free
                then
                return(z)
                /*replace x with y*/
                remove all nodes in Q
                exit()
                j=j+1
                }
        }
        return("Failure")
        end
```

Finding the replaceable node as follows:

node 0 = $0X_{n-1}X_{n-2}\ldots X_w\ldots X_1X_0$

node 1 = $0X_{n-1}X_{n-2}\ldots X'_w\ldots X_1X_0$

node 2= $0X_{n-1}X_{n-2}\ldots X'_{w+1}\ X_w\ \ldots X_1X_0$

$\vdots$

node (n-w) = $0X'_{n-1}X_{n-2}\ldots X_w\ \ldots X_1\ X_0$

node (n-w +1) = $1X_{n-1}X_{n-2}\ldots X_w\ \ldots X_1\ X_0$

node (n-w +2) = $0X_{n-1}X_{n-2}\ldots X'_w\ \ldots X_1X'_0$

node (n-w +3) = $0X_{n-1}X_{n-2}\ldots X'_w\ \ldots X'_1X_0$

$\vdots$

node (n-w +1+w) = $0X_{n-1}X_{n-2}\ldots X'_w X'_{w-1}\ldots X_1X_0$

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

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

$\vdots$

*node (n-w+1+2*w)* = $0X_{n-1}X_{n-2}\ldots X'_{w+1}\ X_wX'_{w-1}\ldots X_1X_0$

*node(n-w+1+2*w+1)*=$0X_{n-1}X_{n-2}\ldots X'_{w+1}\ X'_wX_{w-1}\ldots X_1X_0$

$\vdots$

node ((n-w+1)(w +1))+(1+2+...+n-w)* = $1X'_{n-1}X_{n-2}\ldots X_{w-1}\ldots X_1X_0$

We illustrate an example to explain the operations of the **Fault-Tolerance_Embedding** algorithm when the faulty nodes exist. For the S_{12} as figure 5, the $M_{2^1\times 2^1}$ has been embedded in it.

1. If the node *2* is faulty, it visits or signals the node *6*, to check whether it is free or not. If it is, it terminates.
2. If not, insert the node *6* to the queue, and search the node *10*, to check whether it is free or not. If it is, it terminates.
3. If not, insert the node *10* to the queue, and delete the node *6* from the queue, search the node *7*, to check whether it is free or not. If it is, it terminates.
4. If not, search the node *4*, to check whether it is free or not. If it is, it terminates.
5. If not, delete the node *10* from the queue; search the node *11*, to check whether it is free or not. If it is, it terminates.
6. If not, search the node *8*, 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 *{6(0110), 10(1010), 7(0111), 4(0100), 11(1011), 8(1000), ~~14(1110)~~}*.

The node *14(1110)* is a virtual node, we show the node with deleted line.

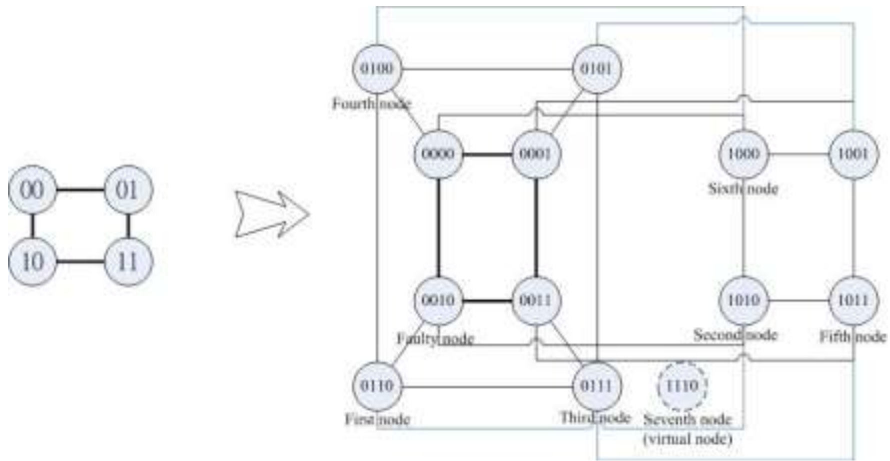

Figure 5. Embedding of a $M_{2\times 2}$ mesh in a S_{12}

We illustrate the searching path of finding a replaceable node in a S_{12} as shown figure 6 by figure 5.

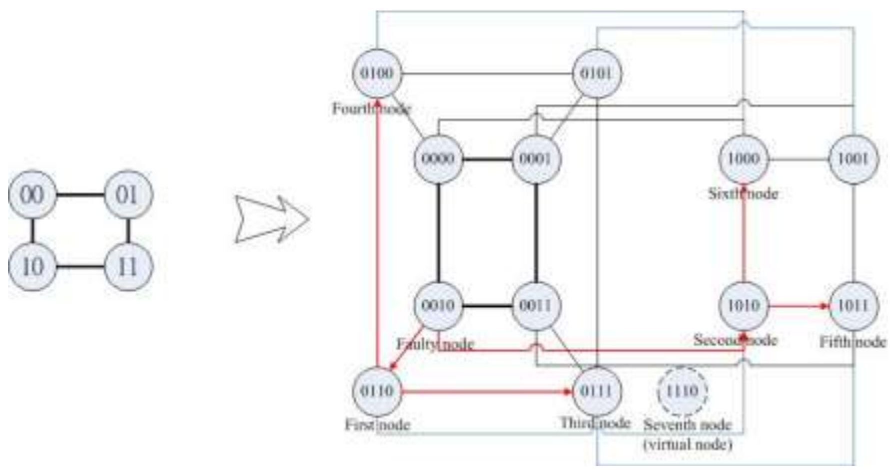

Figure 6. The searching path of finding a replaceable node of the $M_{2\times 2}$ in a faulty S_{12}

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

Proof. Every searching path is only one path according to the algorithm Fault-Tolerance_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 Fault-Tolerance_Embedding. Because these nodes and links of searching paths are not replicated from algorithm Fault-Tolerance_Embedding, These costs associated with graph embedding are dilation *3*, congestion *1*, load *1*, and unbounded expansion.

Theorem 5 A searching path of algorithm Fault-Tolerance_Embedding is including $1/2*n^2 + 3/2*n-1/2* w) – 1/2*w^2+1$ nodes.

Proof. We can embed $M_{m_1\times m_2\times\cdots\times m_d}$ in a S_N by theorem 6. If a node is faulty, we can change a bit in the binary string sequence from bit *w* to bit *n* and insert its corresponding node in the queue. In the worst case, we can get *(n-w+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 *(w-1)*, and we can get *w* different nodes. We can also change a bit in the sequence from bit *0* to bit *w* from the second node of the queue, and we can also get *(w+1)* different nodes. Until the queue is empty, the sum of all searched nodes is $(n-w+1)*(w+1))+(1+2+...+n-w)$. The search path includes $(n-w+1)*(w+1))+(1+2+...+n-w)$ nodes. That is, the whole searching path includes $(n-w+1)*(w+1))+(1+2+...+n-w)= 1/2*n^2 + 3/2*n -1/2* w) – 1/2*w^2+1$ nodes.

Theorem 6 There are $O(n^2-w^2)$ faults, which can be tolerated.

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

5. Conclusions

Hypercubes, meshes, and tori are well known interconnection networks for parallel computing, grid computing, and cloud computing. The Supercubes are superior to hypercube in terms of embedding a mesh and torus under faults. 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 Supercube with node failures. The main results obtained (1) these existent parallel algorithms on mesh or torus architectures to be easily transformed to or implemented on the Supercube architectures with load *1*, congestion *1*, dilation *3*, and unbounded expansion. (2) 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. (3) There are $O(n^2-w^2)$ faults, which can be tolerated. Therefore, we can easily port the parallel or distributed algorithms developed for these structuring of mesh and torus to the Supercube. According to the result, we can easily port the parallel or distributed algorithms developed for these structures to the Supercube. Therefore, these methods of reconfiguring enable extremely high-speed parallel computing, grid computing, and cloud computing.

6. References

[1] Sheldon B. Akers, Balakrishnan Krishnamurthy, "A Group-Theoretic Model for Symmetric Interconnection Networks", IEEE Trans. on Computers, Vol. 38, pp. 555-565, 1989.
[2] James R. Armstromg, Festus G. Gray, "Fault- diagnosis in a Boolean n-Cube array of microprocessor", IEEE Trans. on Computers, Vol. C-30, No. 4, pp. 587-590, 1981.
[3] Laxmi N. Bhuyan, Dharma P. Agrawal, "Generalized Hypercubes and Hyperbus structure for a computer network", IEEE Trans. on Computers, Vol. 33, pp. 323-333, 1984,.
[4] Khaled Day, Abdel Elah Al-Ayyoub, "Fault Diameter of k-ary n-cube Networks", IEEE Trans. on parallel and distributed systems, Vol. 8, No. 9, pp. 903-907, 1997.
[5] Jaliya Ekanayake, Geoffrey Fox, "High Performance Parallel Computing with Clouds and Cloud Technologies", First International Conference CloudComp on Cloud Computing, pp.20-38, 2009
[6] Johan Torkel Hastad, Tom Leighton, Mark j Newman, "Reconfiguring a Hypercube in the Presence of Faults", ACM Theory of Computing, pp. 274-284, 1987.
[7] S. Lennart Johnson, Ching-Tien Ho, "On the conversion between binary code and binary-reflected gray code on binary cubes," IEEE Trans. on Computers, Vol. 44, pp. 47-53, 1995.
[8] Frank Thomson Leighton, Introduction to parallel algorithms and architectures: Arrays, Trees, Hypercubes, MORGAN KAUFMANN PUBLISHERS, Inc., 1992.
[9] Jen-Chih Lin, "Fault-Tolerant Mapping of a Mesh in a Flexible Hypercube", WSEAS Transactions on Computers, Vol. 8, No. 9, pp. 1587-1596, 2009.
[10] Jen-Chih Lin, "Faulty-Avoiding Methods for Mapping Meshes in an IEH", WSEAS Transactions on Computers, Vol. 6, No. 6, pp. 888-893, 2007.
[11] Chong-Dae Park, Kyung-Yong Chwa, "Hamiltonian properties on the class of hypercube-like networks", Information Processing Letters, Vol. 91, pp. 11-17, 2004.
[12] Franco P. Preparata, Jean Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," Commun. ACM, Vol. 24, pp. 300-309, 1981.
[13] Charles L Seitz, "The Cosmic Cube", Commun. ACM, Vol. 28, pp. 22-33, 1985.

[14] Arunabha Sen, Abhijit Sengupta, Subir Bandyopadhyay, “Generalized Supercube: An incrementally expandable interconnection network”, Proceedings of the Third Symposium on Frontiers of Massively Parallel Computation-Frontiers'90, pp. 384-387, 1990.
[15] Herbert Sullivan, Theodore R. Bashkow, David Klappholz, “A large scale, homogeneous, fully distributed parallel machine”, I, Proc. 4th Symp. Computer Architecture, ACM, pp. 105-177, 1977.
[16] Shih-Chang Wang, Yuh-Rong Leu, Sy-Yen 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] Pei-Ji Yang, Sing-Ban Tien, 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.
[18] Siyuan Jing, Kun She, "A Novel Model for Load Balancing in Cloud Data Center", JCIT: Journal of Convergence Information Technology, Vol. 6, No. 4, pp. 171 - 179, 2011.
[19] PENG Limin, XIAO Wenjun, "A Binary-Tree based Hierarchical Load Balancing Algorithm in Structured Peer-to-Peer Systems", JCIT: Journal of Convergence Information Technology, Vol. 6, No. 4, pp. 42 - 49, 2011.
[20] Li Dequan, Wang Ying, Xiong Anyuan, Tinghuai Ma, "High Performance Computing Model for Processing Meteorological Data in Cluster System", JCIT: Journal of Convergence Information Technology, Vol. 6, No. 4, pp. 92- 98, 2011.