Fault Tolerance in Hyperbus and Hypercube Multiprocessors Using Partitioning Scheme

Shih-Chang Wang and Sy-Yen Kuo

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.
sykuo@cc.ee.ntu.edu.tw

Abstract

In this paper, the partitioning scheme is used to achieve fault tolerance in hyperbus and hypercube multiprocessors. Unlike other schemes, processor faults are assumed to be randomly distributed. We propose a novel and practical load redistribution method to tolerate processor faults in a hyperbus structure with insignificant overhead(a slowdown of 2 for computation and a slowdown of 3 for communication in the worst case). Standard routing and broadcasting algorithms were implemented on hypercube computers. To achieve fault tolerance, we present routing and broadcasting algorithms for a faulty hypercube with at most n-1 faults. Compared with other existing algorithms, our methods have better performance in most measures.

1 Introduction

The boolean n-dimensional hypercube (n-cube) computer is an interconnection with $N=2^n$ processors. The n-cube computer can be viewed as having processors(nodes) placed at the corners of an n-dimensional cube, and each edge connected to two processors. Processors in an n-cube communicate by passing messages. Extensive research has been done on hypercubes, and many commercial hypercube computers have successfully been implemented The topological properties of a hypercube were introduced in [3, 4] and various other topologies can be mapped into a hypercube. The hypercube structure can handle a reasonable message traffic, and has some degree of fault tolerance. Many issues of fault tolerance in a faulty n-cube have been proposed [6-7, 9].

The generalized hyperbus structure(GHB) were proposed in [5] and compared to other parallel computers. In addition, the properties of the GHB were investigated and presented in [5]. The hyperbus structure is obtained from GHB by assuming that there are only two buses in each dimension. One of the advantages of the hyperbus structure is that each processor contains only two I/O ports. This structure is extremely suitable for local area computer networks.

Several researchers have examined the partitioning scheme in hypercubes [1, 6, 8]. The *partitioning scheme* is attractive because it can make complex problems simple. An m-partition of an n-cube is to partition the n-cube into several m-cubes. Let F be the set of faulty processors in an n-cube. We say that an m-cube S tolerates F iff S contains a connected component of at least $2^{m-1}+1$ nodes and all of them are healthy. We will say that an m-partition P tolerates F iff for every m-cube S in P, S tolerates F [1]. A fault tolerant m-partition of an n-cube is important because it guarantees that for any pair of adjacent m-cubes there is at least one edge connecting the largest healthy connected components in the m-cubes. If we tolerate processor faults for one application in each m-cube of an m-partition, we can implement the application with an insignificant *efficiency slowdown* in a faulty n-cube. The *efficiency slowdown* of an implementation for an application on a faulty hypercube is defined to be the time of the application required on the faulty hypercube divided by the time required on the fault-free hypercube.

The partitioning scheme can also be applied to the hyperbus. If the number of processor faults in a hyperbus structure is in the *tolerant* range, then we use the partitioning scheme to propose a constructive approach which finds healthy processors to simulate faulty processors. We show that any job designed for the fault-free hyperbus structure can be implemented on a faulty hyperbus structure in which the number of "*special buses*" generated by the faulty processors is at most n with a slowdown of *2* for computation and a slowdown of *3* for communication in the worst case.

Several researchers have given methods for fault-tolerant routing[11-12, 14-15] and broadcasting[11-13, 15-16] algorithms in faulty hypercubes. Using the partitioning scheme, we develop simple routing and broadcasting algorithms in faulty hypercubes. Compared with existing algorithms, our methods have better

Acknowledgment: This research was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 83-0408-E002-022.

0-8186-6555-6/94 $04.00 © 1994 IEEE

performance in most measures. In section 2, we give the assumptions and definitions for our work. Fault tolerance in hyperbus structures is addressed in section 3. Sections 4 and 5 present routing and broadcasting in faulty hypercubes, respectively. Discussions and comparisons are given in section 6 followed by the conclusions in section 7.

2 Preliminaries

First, we make several assumptions which are commonly made in the literature: 1) Only processor faults are considered. In fact, link faults in a faulty hypercube can be tolerated by viewing the two end-nodes of the faulty link as "faulty" nodes. 2) All processor faults are static and detectable. 3) A faulty processor can not compute and can not send messages. 4) The messages sent to a faulty processor will be lost. Next, some definitions and notations are introduced[1]. A *multiset* is a collection of objects in which repetitions are allowed. Let T be a multiset and X be an element in T. The notation $mult(X,T)$ is the number of times X appears in T, and $mult(T)$ is the maximum number of $mult(X,T)$, for all $X \in T$.

Each node in an n-cube is labeled with an n-dimensional binary vector $(x_1\ x_2\ \ldots\ x_{n-1}\ x_n)$, where $x_i \in \{0,1\}$, for $1 \le i \le n$. The dimension associated with x_i is called the ith dimension. The neighboring node of a node X is different from X in a single dimension of their binary representations. The *Hamming distance* between two nodes u and v in an n-cube is denoted by $H(u,v)=\|u \oplus v\|$. Symbol $\oplus$ denotes the *bitwise exclusive-OR* (*XOR*) operation on binary numbers.

An m-cube of an n-cube is denoted by $M=(m_1\ m_2 \ldots m_n)$, where exactly m of the m_i's are $*$'s(don't cares) and the remaining m_i's are either 0's or 1's. Let the set $P \subseteq \{1,2,\ldots,n\}$ and $|P|=m$. An m-partition P contains 2^{n-m} m-cubes where an m-cube M is in P iff for all i, $1 \le i \le n$, $m_i = *$ iff $i \in P$. Given the multiset of n-dimensional binary vectors T and an m-partition P of an n-cube, the notation $proj(P,T)$ is the multiset of $(n-m)$-dimensional binary vectors obtained by removing the m dimensions specified by P for each vector in T. The notation $@(P,T)$ denotes the value of $mult(proj(P,T))$.

For example: Let $T=\{(1100), (1011), (0001)\}$ and $P=\{2\}$, $proj(P,T)=\{(100), (111), (001)\}$ and $@(P,T)=1$.

3 Fault Tolerance in Hyperbus Structures

The generalized hyperbus structure (*GHB*)[5] consists of N buses with $N=m_n \times m_{n-1} \times \ldots \times m_1$. Let $N=2^n$, and $m_i=2$ for $1 \le i \le n$, we obtain the hyperbus structure. A bus in the hyperbus structure is denoted by an n-tuple $(x_1\ x_2 \ldots x_n)$ with $x_i \in \{0,1\}$ and $1 \le i \le n$. A processor in the hyperbus structure is denoted by $(x_1\ x_2 \ldots x_{i-1}\ [y_i z_i]\ x_{i+1} \ldots x_n)$, i.e. , with x_i replaced by a 2-tuple $[y_i z_i]$ and $y_i, z_i \in \{0,1\}$. This means that the processor is connected to the two buses $(x_1\ x_2 \ldots x_{i-1}\ y_i\ x_{i+1} \ldots x_n)$ and $(x_1\ x_2 \ldots z_i\ x_{i+1} \ldots x_n)$. A hyperbus structure with $2^3=8$ buses and $N=2\times2\times2$ is shown in Fig. 1.

We will use the following lemma in [6] to show our results in Theorem 1.

Lemma 1: For $n \ge 1$, given a set F of n or fewer faulty nodes in an n-cube , there exists a 1-partition P of the n-cube such that $@(P,F) \le 1$.

We can view a hyperbus structure with 2^n buses as an n-cube structure and assume that a bus can be in one of the following two states : "*special bus*" or "*normal bus*". If a processor in a hyperbus structure is faulty, the buses which are connected to the faulty processor are *special* buses, otherwise they are *normal* buses. In a hyperbus structure, each processor is connected to two buses and hence a faulty processor will generate two *special* buses. For example, in Fig. 1, if processor (00[01]) and processor (0[01]1) are faulty, the buses (000), (001), and (011) are *special* buses, and the rest are *normal* buses.

We assume that a bus can be held by only one pair of processors which can communicate through the bus in a single time step. Because a processor in a hyperbus structure has two I/O ports, we assume that a processor can process two I/O operations in one time step.

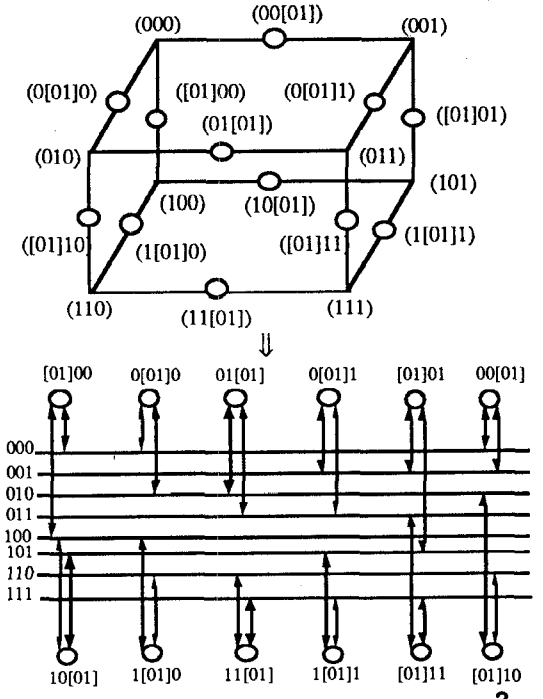

Fig. 1. A hyperbus structure with $2^3=8$ buses and $N=2\times2\times2$.

Theorem 1: For $n \geq 1$, given a faulty hyperbus structure with 2^n buses and a set F of faulty processors, if the number of *special* buses generated by the fault set F is less than or equal to n, then any job on the faulty hyperbus structure can be implemented with, in the worst case, a factor of 2 slowdown for computation, and a factor of 3 slowdown for communication.

Proof: Let the set of *special* buses generated by the fault set F be denoted by F'. Since $|F'| \leq n$, we know from Lemma 1 that there exists a 1-partition P of the faulty hyperbus structure such that each 1-cube in P contains at most one *special* bus. We only need to consider pairs of adjacent 1-cubes as in Fig. 2 where b_i and p_i denote a bus and a processor, respectively.

For any pair of adjacent 1-cubes $\{A,B\}$ in P(see Fig. 2), because each 1-cube contains at most one *special* bus, the faulty node must be located on the link which connects two 1-cubes in P. By symmetry, we can assume that the faulty processor is $p1$ without loss of generality. Then $b1$ and $b2$ are *special* buses; $b3$ and $b4$ are *normal* buses. The computation on the faulty processor is taken over by the processor which connects to the same pair of 1-cubes as the faulty processor. For example, in Fig. 2, the workload of processor $p1$ is taken by processor $p3$. Because each healthy processor is responsible for at most 1 faulty node, all computations can be performed with at most a factor of 2 slowdown.

Each communication operation is implemented in three steps as described below. Sending messages between healthy nodes is performed in the first time step. Sending messages from or to a faulty processor starts at the beginning of the second time step, and take the following new paths. The notation $(pi \xrightarrow{bj} pk)$ denotes that processor pi sends a message to the processor pk through bus bj in one time step.

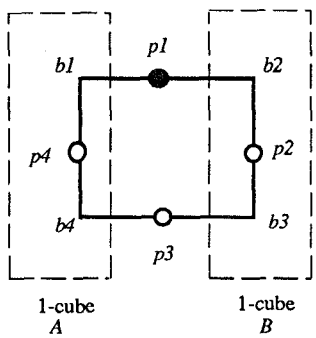

Fig. 2. A pair of adjacent 1-cubes with one faulty node(black).

The path $(p1 \xrightarrow{b2} *)$ (* denotes a healthy processor and $p1$ is faulty) is replaced by the path $(p3 \xrightarrow{b3} p2 \xrightarrow{b2} *)$. The path $(p1 \xrightarrow{b1} *)$ is replaced by the path $(p3 \xrightarrow{b4} p4 \xrightarrow{b1} *)$. The path $(* \xrightarrow{b1} p1)$ is replaced by the path $(* \xrightarrow{b1} p4 \xrightarrow{b4} p3)$. The path $(* \xrightarrow{b2} p1)$ is replaced by the path $(* \xrightarrow{b2} p2 \xrightarrow{b3} p3)$.

It is straightforward to verify that no processors execute more than two I/O operations and no bus conflicts exist in any single time step.

Theorem 1 provides a constructive approach in finding healthy processors to take over the work of faulty processors. In the worst case, the size of the set F is at most $\left\lfloor \frac{n}{2} \right\rfloor$. In the best case, the size of F is at most $n-1$. An example for Theorem 1 is shown in Fig. 3. The *special* buses are 000, 001, and 011. There is at most one special bus in any 1-cube. The work of the faulty node 00[01] is performed by node 10[01], and the work of the faulty node 0[01]1 by node 1[01]1. In addition, communication for each faulty node is completed as previously described. In the next two sections, we will demonstrate how the partitioning scheme can be used to implement communication algorithms in a faulty n-cube.

4 Routing in a Faulty Hypercube

The standard routing algorithm for a fault-free n-cube is described in [2]. This algorithm, which we call ROUTE, is shown in Fig. 4. Let *curr* denote the current node and *dest* denote the destination node. ROUTE sends a message through the link whose label corresponds to the first bit position in which the *curr* and *dest* node labels differ. This simple algorithm always finds a minimum length path. This length is at most n, and the computation overhead of the routing algorithm is very small. The algorithm ROUTE has been implemented on hypercube computers using various approaches. In the algorithm ROUTE, determining a next link for sending one message needs $O(n)$ bit-comparison time, where n is the number of dimensions in the n-cube.

However, algorithm ROUTE does not work if the hypercube is faulty. We will apply the partitioning scheme in a faulty hypercube with at most n-1 faulty nodes(processors) to implement routing. Our algorithm has computation time overhead with $O(n)$ bit-comparison time. This algorithm is easy to be implemented in a faulty hypercube. We also assume that there exists a mechanism as in [9] such that any healthy node in an n-cube can determine the faulty/nonfaulty status of its neighboring nodes, and each processor has a fault list for recording the status of neighboring nodes. Therefore, we need additional $O(n)$ bits space for each processor. To facilitate our discussion, we need some more notations and definitions. The dimensions which are local to the 2-cubes in a 2-partition are called the *internal* dimensions, otherwise they are called the *external* dimensions. The

neighboring nodes of a node W within the same 2-cube are called the *buddies* of the node W.

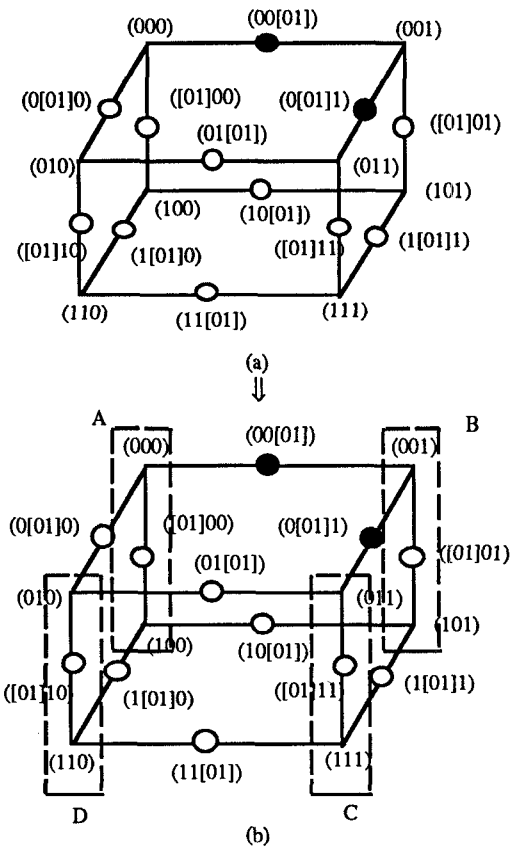

Fig. 3. (a) Two faulty nodes (00[01]) and (0[01]1). (b) Use a 1-partition P={ 1 } to partition the 3-cube into four 1-cubes, A, B, C, and D.

Algorithm ROUTE
{*curr* is the current node and *dest* is the destination node. $f_1(curr,dest)$ is the first dimension from left that the two labels *curr* and *dest* differ.}
begin
  for every message
    if *curr=dest*
    then Retain message in node *curr*;
    else Send message to neighboring processor via link $f_1(curr, dest)$.
end

Fig. 4. Algorithm ROUTE for node-to-node routing in a fault-free hypercube.

For example, in Fig. 5, a 4-cube is partitioned into four 2-cubes (**00), (**01), (**10), and (**11) using a 2-partition $P=\{1,2\}$. The *1st* and *2nd* dimensions from left are the *internal* dimensions, and the *3rd* and *4th* dimensions are the *external* dimensions. The buddies of node (1110) are nodes (0110) and (1010). In a 2-partition, each node has only two buddies. The function $f_i(x_k,x_j)$ is the position of the *i*th *external* bit from left, that is, the *i*th *external* dimension, at which the labels x_k and x_j differ. The function $g_i(x_k,x_j)$ is the position of the *i*th *internal* bit from left, that is, the *i*th *internal* dimension, at which the labels x_k and x_j differ. The function $h_i(curr)$ lets the current node *curr* check the status of its neighboring node in the *i*th dimension. If the neighboring node in the *i*th dimension is not faulty, then $h_i(curr)$ returns TRUE, else returns FALSE. If there is no value *i* for $f_i()$ or $g_i()$, then $f_i()$ or $g_i()$ is assumed to have the value 0. For example, in Fig. 5:

$f_1(0100,1100)=0$ $f_2(0100,0111)=4$
$g_1(0100,1100)=1$ $f_1(0100,0001)=4$
$f_2(0100,0001)=0$ $h_1(1000)$=FALSE
$h_2(1000)$=TRUE

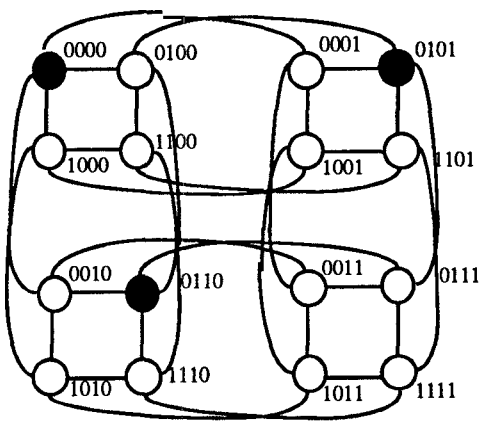

Fig. 5. A 4-cube which is partitioned into four 2-cubes using a 2-partition *P*={1, 2}.

We will need the following lemma in [1] for the development of our algorithm, and its complexity is $O(n)$.

Lemma 2: For $n \geq 2$, given a set F of n-1 or fewer faulty nodes in an n-cube, there exists a 2-partition P of the n-cube such that $@(P,F) \leq 1$.

If the number of faulty processors is equal to or greater than n in an n-cube, there may exist an isolated healthy processor. Therefore, we will assume that the number of faulty processors in a n-cube is at most n-1. From Lemma 2, for a set F of at most n-1 faulty nodes in an n-cube, there exists a 2-partition P such that $@(P,F) \leq 1$, that is, there exists none or one faulty node in any 2-cube in P. It denotes that there exists one link with both end nodes being healthy to connect a pair of 2-cubes for any pair of 2-cubes in P. So the routing algorithm can be implemented on a faulty n-cube. The main idea is to route one message from the 2-cube with the current node to the 2-cube with the destination node. This goal is easy to be accomplished by reducing the differences of the *external* dimensions between the current node and the destination node. The routing algorithm, F_ROUTING, for a faulty n-cube is shown in Fig. 6. The following

Lemma 3 and Theorem 2 formally present the above results.

Lemma 3: For $n \geq 2$, given a set F of n-1 or fewer faulty nodes in an n-cube, there exists a 2-partition P such that any pair of adjacent 2-cubes X and Y in P is different in some dimension i. If the neighboring node $y1$ (in Y) of the healthy node $x1$ (in X) in the ith dimension is faulty(see Fig. 7), there exist some links with both end nodes being healthy, and one of the end nodes is the healthy buddy of node $x1$, and the other is in the 2-cube Y.

Proof: From Lemma 2, there exists a 2-partition $P1$ such that $@(P1,F) \leq 1$. Let $P=P1$. Any pair of adjacent 2-cubes can be represented as in Fig. 7. The following cases are possible scenarios for a pair of adjacent 2-cubes in P.

1. Neither X nor Y has a faulty node.
2. Only X or Y contains one faulty node.
3. X and Y each contains a faulty node.

By using the symmetry property, it is straightforward to check the three cases to verify Lemma 3.

```
Algorithm  F_ROUTING
{curr denotes the current node, and dest denotes the
  destination node.}
1  begin
2  for every message
3    if curr=dest then
4                retain message in node curr;
5    else  if there exists i such that f_i(curr,dest) ≠ 0
6          and h_{f_i(curr,dest)}(curr)=TRUE then
7                node A= the neighboring node of
                 f_i(curr,dest) dimension
8                of node curr with the smallest i;
9          else if there exists  j such that
10         g_j(curr,dest) ≠ 0 and h_{g_j(curr,dest)}(curr)=TRUE then
11               node A=the neighboring node of
                 g_j(curr,dest) dimension
12               of node curr with the smallest j;
13           else node A= the healthy buddy of the node curr
                          with the
14                        smallest Hamming distance
15                        between the healthy
16                        buddies of curr
17                        and dest.
19           endif
20         endif
21         send message from curr to the neighboring node A;
22   endif
23end
```

Fig. 6. Routing algorithm F_ROUTING for an *n*-cube with at most *n*-1 faulty nodes.

Theorem 2: For $n \geq 2$, given a set F of n-1 or fewer faulty nodes in an n-cube, there exists a 2-partition P such that the algorithm F_ROUTING will find a correct path of finite length to transfer a message.

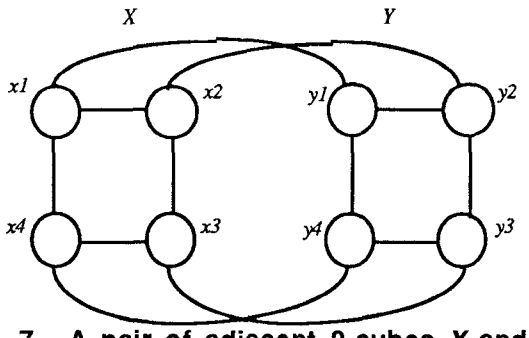

Fig. 7. A pair of adjacent 2-cubes *X* and *Y*.

Proof: From Lemma 2, there exists a 2-partition P such that $@(P,F) \leq 1$. We use the 2-partition P to partition the faulty n-cube. Recall that the time to compute P is $O(n)$. We can install the information of P in each healthy processor. This action just takes a very small time overhead. Note that there are three possible relationships between the nodes *curr* and *dest* in P.

The first one is if node labels *curr* and *dest* are the same, then node *curr* retains the message. This is done in line 3 of Fig. 6.

The second one is that the nodes *curr* and *dest* have differences in the *external* dimensions. If the neighboring nodes of node *curr* in the different *external* dimensions are not all faulty, then F_ROUTING sends the message to the healthy neighbor which has the smallest different *external* dimension i from the left. This is done in line 5. If the neighbors of node *curr* in the different *external* dimensions are all faulty and because any healthy node has at least one healthy buddy, from Lemma 3, there exists at least one healthy buddy of node *curr* to reduce the differences in the *external* dimensions. We select the healthy buddy which can also reduce the difference in *internal* dimensions if possible. This is done in line 9. If no healthy buddy can reduce the difference in *internal* dimensions, then we select the healthy buddy x which has the minimum Hamming distance between the nodes x and *dest*. This is done in line 13. From the above discussion, we know that F_ROUTING can send a message from the 2-cube with node *curr* to the 2-cube with node *dest*.

The last one is that the nodes *curr* and *dest* are local to the same 2-cube. It is straightforward to show that any pair of healthy nodes in a 2-cube of P can exchange a message successfully in line 9. Obviously, a message can be sent correctly by using algorithm F_ROUTING, and it finds a path of finite length.

The functions $f_i()$ and $g_i()$ take $O(n)$ bit-comparison time. The function $h_i()$ and line 13 in Fig. 6 take O(1) time. Therefore, the time complexity of algorithm F_ROUTING is $O(n)$. The algorithm F_ROUTING takes $O(n)$ bit-comparison time to determine a next link for sending one message in each processor of an n-cube. $O(n)$ additional bits is required in each processor for maintaining a fault list. It is easy to see that the

maximum length for sending a message from x to y is $2H(x,y)+2$

5 Broadcasting in a Faulty Hypercube

The standard algorithm(see Fig. 8.) for broadcasting a message to all nodes from a single source is given in [2]. The number of time steps in the algorithm is n, where n is the number of dimensions of the hypercube. Recall that there may exist an isolated healthy processor if the number of faulty processors is greater than n-1 in an n-cube, so we assume the number of faulty processors is less than n. An example for the standard broadcasting algorithm is shown in Fig. 9(The integer i next to a link denotes that the message passing of the link is executed at the ith time step, for $1 \le i \le 4$).

A straightforward broadcasting algorithm on a faulty n-cube with at most n-1 faults is described as follows. It uses algorithm BROADCAST in Fig. 8 and the function $h_i(X)$. Only when CONTROL[i]=1 and the neighbor of node *curr* in the ith dimension is healthy, can node *curr* send both the message and the modified CONTROL via link i . Unfortunately, This method will make some healthy nodes unable to receive the message. For example, let the faulty nodes in Fig. 9 be (1010), (1001) and (1111). Let node (1000) be the source, it is evident reason that node 1011 will not receive the broadcasted message. Such an incomplete broadcasting process is called a *loss broadcasting* .

We will present a new fault-tolerant broadcasting algorithm that can be implemented on a faulty n-cube with at most n-1 faults. Let the functions $f_i(X)$ and $g_i(X)$ denote the positions of the ith *external* and *internal* dimensions from the left for X in P, respectively. The function $l(X)$ denotes that the assignment operations executed on X are only for *internal* dimensions. For example, if $P=\{3,4\}$, $f_1(****)=1$, $f_2(****)=2$, $g_1(****)=3$, and $g_2(****)= 4$. Let X be (1000). The operation $l(X)$= (11) will result in X=(1011).

```
Algorithm BROADCAST
{src denotes the source node, and curr denotes the current
  node.}
begin
        if curr=src then
            for i=1 to n do
            CONTROL[i]=1;
        endif
        for i=1 to n do
         if CONTROL[i]=1 then
           CONTROL[i]=0;
            Send a message and CONTROL to neighbor via
            link i;
         endif
end
```

Fig. 8. Algorithm BROADCAST in a fault-free hypercube.

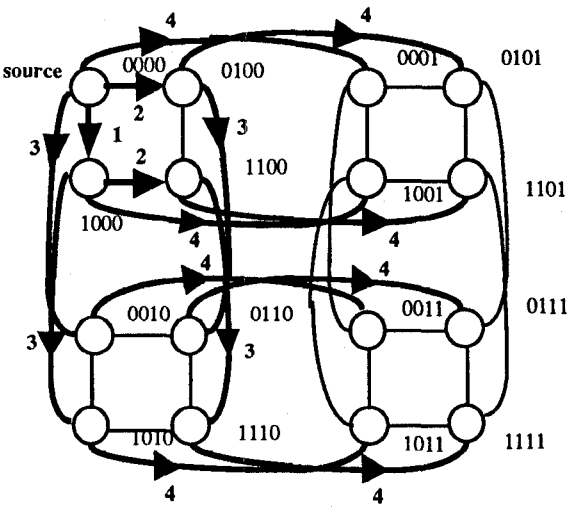

Fig. 9. Broadcasting a message in a fault-free 4-cube using the algorithm BROADCAST.

Theorem 3: For $n \ge 2$, given a set F of n-1 or fewer faulty nodes in an n-cube, there exists a 2-partition P such that the algorithm F_BROADCAST(see Fig. 10) can complete the broadcasting on the healthy processors.

Proof: From Lemma 2, there exists a 2-partition P such that $@(P,F) \le 1$. We use the 2-partition P to partition the n-cube. The n-cube can be viewed as an (n-2)-cube structure with 2^{n-2} 2-cubes. Recall that for a pair of adjacent 2-cubes, there is at least one edge connecting the 2-cubes. The links connecting a pair of adjacent 2-cubes are in the *external* dimensions. The 2-cubes of the n-cube in P are called *SB*s(subcube blocks), and we have 2^{n-2} *SB*s in P. In the algorithm BROADCAST, each node only receives one message, and the process of the broadcasting is complete(that is, all processors can receive the broadcasting message). In the algorithm BROADCAST, if the ith bit of CONTROL is equal to 1, then the node *curr* has to broadcast the message corresponding to the CONTROL to the ith neighboring node. Therefore, the node *curr* makes the ith dimension of the CONTROL be 0, and sends the message and the modified vector CONTROL to the ith neighboring node. We can use the same approach as algorithm BROADCAST to implement our CONTROL vector.

Algorithm F_BROADCAST broadcasts the message among those *SB*s. The same scheme of CONTROL used in BROADCAST is used to implement the *external* dimensions of CONTROL using in F_BROADCAST. It is clear that each *SB* of the n-cube will correspond to an unique sending *SB* except the *SB* with source. Let one healthy node in *S1*(is a subcube block) send the message to the other *SB*s which *S1* corresponds to. So, each node only corresponds to one sending node except the source node. F_BROADCAST first broadcasts the message to every *SB*s(lines 9..19). If there are 1's in the *external*

dimensions of CONTROL for node X, but the neighbors of X in such *external* dimensions are all faulty, then from Lemma 3, the healthy buddy which is not broadcasted can broadcast the message to such *SB*s. This goal can be achieved as follows. First, node X sends message and the modified CONTROL to one healthy buddy. Next, X sets its *external* dimensions of CONTROL to be all 0's and broadcasts continually the message to another buddy if possible(lines 22..30). From above discussions, we know that F_BROADCAST first sends the message to each *SB* if possible. Notice that the *internal* dimensions of CONTROL which is sent from one *SB* to another *SB* must be all 1's(This denotes that the message must be broadcasted inside a *SB*). If the broadcasting among *SB*s is finished, then F_BROADCAST begins broadcasting within a *SB*. It is easy to check that lines 22..30 achieve the goal. The proof is completed.

In the worst case, $2n$-2 time steps are needed to broadcast a message. The time steps for broadcasting among those *SB*s are at most $2(n-2)$(if *curr* can not broadcast the message to any neighboring *SB*s directly), and the time steps for broadcasting within a *SB* are at most 2. The functions $f_i()$ and $h_i()$ take O(1) time. We need $O(n)$ time to determine a next link for broadcasting a message. $O(n)$ bits are required in each processor for maintaining a fault list. If the n-cube is nonfaulty, then just n time steps are required by F_BROADCAST.

6 Discussions and Comparisons

In this section, we compare the algorithms F_ROUTING and F_BROADCAST with existing fault-tolerant algorithms in the literature and give discussions.

6.1 Performance Measures

The performance measures examined are:

» *Time Complexity for Preprocessing*, T_p, which is the time overhead needed before fault-tolerant communications. For example, the preprocessing work in our approach is to determine a fault-tolerant 2-partition.

» *Time Complexity for Each Node*, T_n, which is the time that each node takes in determining the next intermediate node for a received message.

» *Maximum Number of Lengths/Steps*, L_m, which is the lengths/steps in the worst case to complete a routing/broadcasting.

» *Message Size*, M_s, which is the size of message for sending data. A message may contain data field and other control fields.

» *Fault Type*, F_k, which is the faults that can be tolerated in a fault-tolerant scheme.

» *Range of Faults*, Rf, which is the upper bound on the number of faults that the algorithm is guaranteed to work correctly.

6.2 Discussions

The algorithms F_ROUTING and F_BROADCASTING are also suitable for link faults with some modifications. First, we can view the two end-nodes of a faulty link as "faulty" nodes and then determine the fault-tolerant 2-partition for faulty nodes. Second, we let the end-nodes of a faulty link return to their real states (that is, if the end-node is healthy, it is the nonfaulty node.) when our algorithms are used. For a nonfaulty node G, if the link which connects G with its healthy neighbor U(in the ith dimension) is faulty, the function $h_i()$ for G and U will be viewed as FALSE in our algorithms. With the above modifications, it is clear that our algorithms are also suitable for link faults. These modifications need at most n bits to record the states of connecting links for a nonfaulty node.

```
Algorithm F_BROADCAST
{src is the source node label, and curr is the current node
  label.}
01 begin
02 if curr=src then
03     for i=1 to n do
04         CONTROL[i]=1;
06 endif
09 for i=1 to n-2 do
10     if CONTROL[f_i(CONTROL)] ≠0 and
           h_{f_i(CONTROL)}(curr)=TRUE then
12         CONTROL[f_i(CONTROL)]=0;
13         let NEW_CONTROL=CONTROL;
14         let l(NEW_CONTROL)=(11);
15         send a message and NEW_CONTROL to the
           neighbor of the
16     f_i(CONTROL) dimension;
19     endif
22 for i=1 to 2 do
23     if g_i(CONTROL) ≠0 and
           h_{g_i(CONTROL)}(curr)=TRUE then
25         CONTROL[g_i(CONTROL)]=0;
26         send a message and CONTROL to the neighbor of
27         the g_i(CONTROL) dimension;
28         Set the exteranl dimensions of CONTROL to be all
           0's;
30         endif
34 end
```

Fig. 10. A broadcasting algorithm for a faulty n-cube with at most n-1 faults.

6.3 Comparisons

We will examine three alternative schemes in this section. Y. Lan [12] proposed a multicasting scheme *MUTICAST(source address U_s, destination list D)* for a faulty n-cube. If we let the size of parameter D in MUTICAST be 1, we can view it as a fault-tolerant routing scheme. This scheme is denoted by F_M(1). Moreover, a fault-tolerant broadcasting scheme is obtained

if the size of parameter D in MULTICAST is n-1. Note that these n-1 destinations are all the nodes except the source node in an n-cube. This scheme is denoted by F_M(n). The preprocessing time T_p for Lan's approach is spent at step 2 and step 3 in MULTICAST[12]. The time T_n for each node is the execution time from step 4 to step 10 in MULTICAST. J. Wu [23] also proposed a broadcasting scheme F_B for a faulty hypercube. The preprocessing work for this scheme is a *splitting process*. The performance analysis for these schemes and ours is shown in Table 1.

The L_m values for schemes F_M(1) and F_M(n) are unknown because it is difficult to analyze algorithm MULTICAST. Table 1 shows that our approaches are better than others in all measures except L_m. Algorithm F_B uses more time in preprocessing, so it can obtain better result for L_m. However, our approach also has *linear order* for L_m. In addition, F_B only considers link faults. If faults occur with a high probability, F_B will have a greater overhead in preprocessing than our schemes. It is clear that the worst case of L_m occurs in our schemes with a low probability since such a faulty distribution is a rare.

	F_M(1)	F_ROUTING	F_BROADCAST	F_M(n)	F_B
function	routing	routing	broadcasting	broadcasting	broadcasting
Tp	$O(n^2)$	O(n)	O(n)	$O(n^2)$	$O(n^2)$
Tn	$O(n^2)$	O(n)	O(n)	$O(n^2)$	O(n)
Lm	—	2H(u,v)+2	2n-2	—	n+2
Fk	link/node	link/node	link/node	link/node	link
Ms	n-bit source ,n-bit destination, and data filed	n-bit destination, and data filed	n-bit control and data filed	n-bit source, n-bit destination and data filed	n-bit control field and data filed
Rf	n-1	n-1	n-1	n-1	n-1

Table 1. Performance analysis for several schemes.

7 Conclusions

In this paper, we have presented several approaches to implement fault tolerant hyperbus and hypercube multiprocessors. These approaches are based on the *partitioning* scheme. The partitioning scheme makes the analysis and implementation of distributed algorithms under multiple faults much easier. We have proposed a way to design a hyperbus structure in which the number of faulty processors is in a *tolerant* range with insignificant slowdowns. We have also presented ways for message routing and broadcasting in a faulty hypercube with at most n-1 faults. Compared with other existing algorithms, our methods have better performance in most measures.

References

[1] J. Bruck, R. Cypher, and D. Soroker, "Tolerating faults in hypercubes using subcube paritioning," *IEEE Tran. Comput.*, vol. 41, no. 5, pp. 599-605, May 1992.

[2] H. Sullivan, T. Bashkow, and D. Klappholz, "A large scale, homogeneous, fully distributed parallel machine," in *Proc. 4th Symp. Comput. Architecture*, pp. 105-124, Mar. 1977.

[3] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Tran. Comput.* vol. 37, no. 7, July 1988.

[4] A. Y. Wu, "Embedding of tree networks in to hypercubes," *J. Parallel Distributed Comput.*, vol. 2, pp. 238-249, 1985.

[5] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Tran. Comput.* vol. c-33, no. 4, pp. 323-333, April 1984.

[6] M, Y. Chan and S. J. Lee, "Fault-tolerant permutation routing in hypercubes," *Univ. Texas at Dallas Tech. Rep.*, UTDCS-5-90.

[7] J. Hastad, T. Leighton, and M. Newman, "Fast computation using faulty hypercubes," in *Proc. 21st Annu. ACM Symp. Theory Comput.*, pp. 251-263, 1989.

[8] J. P. Sheu, Y. S. Chen, and C. Y. Chang, "Fault-tolerant sorting algorithm on hypercube multicomputers," *J. Parallel and Distributed Comput.*, pp. 185-197, 1992.

[9] T. C. Lee and J. P. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Tran. Comput.*, vol 41, no. 10, pp. 1242-1256, Oct. 1992.

[10] S. C. Chau and A. L. Liestman, "A proposal for a fault-tolerance binary hypercube architecture," in *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 323-330, 1989.

[11] C.S. Raghavender, P. J. Yang, and S.B. Tien, "Free dimension- An effective approach to achieve fault tolerance in hypercubes," in *Proc. Int'l. Symp. on Fault-Tolerant Computing*, pp. 170-177, July 1992.

[12] Y. Lan, "Multicast in faulty hypercubes," in *Proc. Int'l Conf. on parallel Processing*, pp. I58-61, Aug. 1992.

[13] J. Wu, "Fault-tolerant nonredundant broadcasting in hypercubes," in *Proc. Int'l Conf. on Parallel Processing*, pp. III23-26, Aug. 1992.

[14] M. S. Chen and K. G. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multiprocessors," *IEEE Tran. on Parallel and Distributed Systems*, pp. 152-159, April 1990.

[15] M. Peercy and P. Banerjee, "Optimal distributed deadlock-free algorithms for routing and broadcasting in arbitrarily faulty hypercubes," *in Proc. Int'l Symp. on Fault-Tolerant Computing*, 1990.

[16] P. Ramanathan and K. G. Shin, "Reliable broadcast in Hypercube Multicomputers," *IEEE Tran. Comput.*, pp. 1654-1657, Dec. 1988.