

Fault Tolerance in Hypercube Systems and Implementation of Fault Tolerant Algorithms

Kayhan ERCİYEŞ¹

Novruz ALLAHVERDİ²

A.Belma ŞAHİN¹

¹Ege University, International Computing Institute, İzmir, Turkey

e-mail : erciyes@ube.ege.edu.tr, sahin@alpha.ube.ege.edu.tr

²Selçuk University, Faculty of Technical Education, Dept. Of Electrical and Computer Education, Konya, Turkey

e-mail : noval@alaeddin.cc.selcuk.edu.tr

Abstract : In this study, we've analyzed and implemented three different algorithms developed for fault-tolerant routing in hypercube systems. First algorithm[1] proposes an approach to determine the shortest path between the source and the destination nodes in a faulty (or non-faulty) hypercube. Second algorithm[2] has a different approach to hypercube node structure and proposes that direct use of sharp product is not sufficient to discard only computational part (processor and memory) when only this part of a node is faulty. The procedures proposed within this algorithm, allow to obtain a set of extended fault-free subcubes which is a beginning set for further manipulation in hypercube multiprocessors and to increase the reliability of such systems. Third algorithm[3] proposes a procedure for definition of neighborhood and fault-free subcubes in a fault-free subcubes in a faulty-hypercube. We've implemented these algorithms in UNIX and in Intel iPSC/860 Simulation platforms. Simulations are done for static and dynamic routing. Performances are compared for path availability percentages and time used to find path.

Keywords : Hypercube, fault-tolerance, shortest-path, routing, cube algebra, computational part, communicational part, iPSC

1 INTRODUCTION

A hypercube is a distributed parallel system consisting of 2^n identical processors, each provided with its own sizable memory and interconnected with n neighbors. It has a homogeneous symmetric structure and has necessarily rich connectivity. It also has useful topology in which many other topologies, such as meshes, rings, trees, etc. can be embedded. Hypercube systems have been designed and used since the 1980's. The Ncube's 3200 and 6400, Intel's iPSC series (iPSC/1, iPSC/2, iPSC/860), the Amatek's series (S-14), etc. are among them that can have from 128 up to 4096 processors [6, 11, 17].

As the size of a hypercube grows, the probability of some processors or links in the system that may have fault increases. One of the important issues in these systems is how to communicate messages in the presence of component failures. Reliable data communication is essential, especially when hypercubes are used for safety-critical and time-critical applications requiring high reliability [5]. It is very important to detect and isolate faulty processors to ensure correct completion of computation in such applications. In order to determine and avoid the faulty components and to find the shortest path between the source and the target nodes, there are many different kinds of methods [5,9,10,11,15,21-24]. But most of these methods depend on the number and the scattering of the faulty components in hypercube.

In this study, we've implemented three different algorithms developed to find the shortest-path in faulty hypercube systems. First algorithm allows to define any path between two given nonisolated nodes in presence of a lot of faulty nodes and links, i.e. $f \gg n$, where f is the number of faulty elements and n is the dimension of hypercube.

Second algorithm provides procedures to overcome the effects of faulty components and to exploit a fault-free architecture in the presence of node and

link faults. We consider node design, where each node contains two units, namely, a computational part (CPP) and a communication part (CMP) that can operate independently [17, 8, 16, 20]. The CMP is often called a *router*. The routers have the capability of forwarding the messages from other routers toward the destination node.

Third algorithm allows us to calculate firstly a configuration that has only non-faulty components. Secondly, we search the subcubes covered the source and target vertices (nodes). Thirdly we look for intersection or bridge between the fault-free subcubes, with the aim of definition of their common parts. Thus, we have a possibility to define a path between the source and destination nodes, using only the non-faulty cubes (subcubes) if source and target nodes aren't placed in isolated subcubes. This algorithm, doesn't depend on the number of faulty components and is not affected by the scattering of these components in the hypercube.

These algorithms are based on cube algebra operations. The next section provides the background on the elements and operations of cube algebra. The rest of the paper is organized as follows. A method for determination of a set of nonfaulty and nonmaximal subcubes, two methods for local determination of nonfaulty and maximal subcubes including current source, routing algorithm in faulty hypercube and implementation results for UNIX platform is defined in Section 3. The structure of a hypercube node(CPP, CMP), the procedures developed to find extended set of non-faulty subcubes set in a faulty hypercube system and implementation results for UNIX platform is defined in Section 4. Procedure for definition of neighborhood in faulty hypercube multiprocessor and implementation results of this procedure for UNIX platform is defined in Section 5. Dynamic and static routing implementation details on iPSC simulator are considered in Section 6 for all three algorithms. Conclusion notes are described in Section 7.

2 OPERATIONS OF CUBE ALGEBRA

The cube algebra was developed to execute some operations on the binary cubes (hypercubes). It includes almost all operations of the set theory and three own operations, that are called *coordinate product* (*star product*, *consensus* or *★-operation*), *coordinate subtraction* (*sharp product* or *#-operation*) and *coordinate intersection* (\cap -operation).

Earlier, the cube algebra was used to present the Boolean functions in the form of cubes [29, 31,]. Later, the operations of cube algebra were used for the minimization of switching functions [9, 13, 18]. Now these operations are also used for subcube allocation problem [12] and for routing of the information over fault-free nodes and links in hypercube [1]. In these studies, it is shown that these operations are a good tool for the definition of complete nonfaulty subcubes of a hypercube with faulty, prohibited or busy components, also for definition of the paths, including the shortest path from any vertex to other vertex in hypercube.

The operations of cube algebra were explained in detail in [7, 4,] and [10,14]. We also modified the interpretation of these operations, dividing its execution to two parts, where first we define a vector of the respective operation, second we define the ending result based on the vector. Thus, we can apply these intermediate results (vectors) to determine some other parameters in hypercube, for example; Hamming distance [14, 19]. Since, we also use these operations, let us consider them briefly.

Following [14, 10, 9, 19, 1], the *coordinate subtraction* of two cubes C_1 and C_2 , denoted by $C_1 \# C_2$, is the set of subcubes including the nodes from C_1 , but not from C_2 . For example, $**0\#*11=**0$ or $\{**0,1**\}\#110=\{0*0,*00,10*,1*1\}$. The *coordinate product* of two cubes C_1 and C_2 , denoted by $C_1 \star C_2$, is the subcube founded in C_1 and C_2 simultaneously or the subcube one part of which is in C_1

and the other is in C_2 . For example, $*00 \star 1*1 = 10*$. The *coordinate intersection* of two cubes C_1 and C_2 , denoted by $C_1 \cap C_2$, is the subcube presented in C_1 and in C_2 at the same time. For example, $0*0 \cap 10* = \emptyset$ or $10* \cap 1*1 = 101$.

After executing these operations, one can obtain repeated cubes and cubes with smaller dimensions which can be included in the cubes with bigger dimensions. Everywhere in this study, we will assume that the repeated cubes and the cubes with smaller dimensions included in the cubes with bigger dimensions are avoided.

The coordinate subtraction operation was applied to find the local prime implicants of the switching functions. It was later known that this operation is a good tool for the definition of complete nonfaulty subcubes of hypercube with faulty or prohibition vertices, also for the definition of the paths, including the shortest path from any vertex to other vertex in the hypercube [1]. To discard the faulty components from the hypercube, the subtraction operation may be used. By means of the coordinate product, the following operations may be realized: 1) Definition of the Hamming distance between the cubes, which is the number of links between them along the shortest path; 2) Intersection of two or more cubes with the aim of definition of their common parts (subcubes); 3) Unification of two m -cubes A and B in the $(m+1)$ -cube C . The intersection operation is very useful for the determination of the common parts of cubes. At first, it was applied to definition of the parts of some prime implicant for switching functions which are common with other prime implicants. This allowed to answer the question: is the given prime implicant extreme? From the point of routing in hypercube, the coordinate intersection operation can be used for the revelation of the common parts of cubes with the aim of defining the paths to pass from one subcube to another. But cube algebra's operations, especially, subtraction operation can not directly take into consideration the fact that the hypercube node consists from two parts: the computational part (a processor and local memory) and the communication part (router). This issue will be discussed in Section 4 in detail.

3 ROUTING ALGORITHM IN FAULTY HYPERCUBE

Before analyzing the algorithm, we have to determine the methods of finding the set of the non-faulty and maximal subcubes (SNMS) in faulty hypercube and methods for local finding of all SNMS including any node.

The theorems given below allow to determine the method of finding the set of the nonfaulty and maximal subcubes (SNMS) in faulty hypercube.

Theorem 1 : If F is a set of faulty cubes of complete n -cube and if $z_i \in \{0,1\}$ is the value of the i th coordinate of any nonfaulty and maximal subcube Z of this n -cube, then at least one faulty subcube $f \in F$ exists in which the value of i th coordinate is \bar{z}_i .

Proof : If the value of the i th coordinate of nonfaulty cube Z is z_i , and for example $z_i = 1$, then there is at least one faulty cube in which the value of i th coordinate is 0. If there is not such a faulty cube, the value $z_i = 1$ should not exist. In this case the value should be $z_i = \bar{1}$.

Theorem 2 : If F is a set of complete faulty cubes, then the SNMS of n -cube including F is determined by the # -subtracting the set F from the complete n -cube and avoiding non-maximal cubes from the result.

Proof : The theorem can be proven according to the Theorem 1. Assume that one of the subcubes from the SNMS has a form as: $z_1 z_2 \dots z_j \dots z_g \dots z_m$. If we subtract a cube $f_1 \in F$ with the coordinate j , the value of which is z_j from the complete n -cube, then we obtain a cube $K_{1,1} = z_1 z_2 \dots \bar{z}_j \dots z_g \dots z_m$ and a set of cubes $K_{1,2}$. We continue subtracting a cube $f_2 \in F$ with the coordinate g , the value of which is \bar{z}_g from $K_{2,1}$, then we obtain a cube $K_{2,1} = z_1 z_2 \dots \bar{z}_j \dots \bar{z}_g \dots z_m$ and a set of cubes $K_{2,2}$. We also subtract a cube $f_3 \in F$ with the coordinate m , the value of which is \bar{z}_m from $K_{2,1}$ and $K_{2,2}$, then we obtain a cube $K_{3,1} = z_1 z_2 \dots \bar{z}_j \dots \bar{z}_g \dots \bar{z}_m$ and a set of cube $K_{3,2}$. Thus, if $F = \{f_i\}_{i=1}$, then we obtain the sets of cubes $K = K_{i,1} \cap K_{i,2}$,

by means of successive subtraction of all the cubes that are in the set F from complete n -cube. As we see, the set K includes the cube $K_{l,1}$ that we seek for and the other cubes of the subset $K_{l,2}$. As we look for any nonfaulty and maximal subcubes, we can assure that all these cubes must be in the set K .

Two methods are known for local finding of all SNMS including any node (Nadjafov and Kahramanov, 1973; Kahramanli and Allahverdi, 1993). These two methods are equivalent from point of view of the first data and ending results. But for a given programming system, one of these methods may be more favorable than another because of using different operations.

Assume $A = a_1 a_2 \dots a_i \dots a_n$ and $f_j = \phi_{j1} \phi_{j2} \dots \phi_{ji} \dots \phi_{jn}$ are the codes of the source node and any faulty node, respectively. Then the cube f_j may be transformed to cube $Q_j = q_{j1} q_{j2} \dots q_{ji} \dots q_{jn}$. According to Nadjafov and Kahramanov, 1973:

$$1. \quad q_{ji} = \phi_{ji} \text{ if } \phi_{ji} = \bar{a}_i$$

$$q_{ji} = * \text{ if } \phi_{ji} = a_i \text{ or}$$

$$\phi_{ji} = *, j=1, \dots, l; i=1, \dots, n$$

2. The set of Q_m is formed by taking out the nonmaximal cubes from the set of $Q = \{ Q_j \}_{j=1}^l$

3. The set of Q_m is $\#$ -subtracted from complete n -cube. Nonmaximal and repeated cubes are taken out from this result and thus, the set of D_A including the source node A is formed.

According to Kahramanli and Allahverdi, 1993:

$$1. q_{ji} = a_i \text{ if } \phi_{ji} = \bar{a}_i$$

$$q_{ji} = * \text{ if } \phi_{ji} = a_i \text{ or}$$

$$\phi_{ji} = *, j=1, \dots, l; i=1, \dots, n$$

2. It is the same with the step 2 of preceding method.
3. The set of Q_m is separated into two cube sets Q_{m1} and Q_{m2} which are formed with $r = n - 1$ and $r < n - 1$ dimensional cubes, respectively.
4. The cubes in the set Q_{m1} are logically intersected, as a result of which the cube D_1 is formed.
5. All the cubes with dimension $r = \{1, 2, \dots, n - 2\}$ in the set Q_{m2} are transformed to $n - r$ pieces $(n - 1)$ cubes, i.e. $r \times 1 \rightarrow (n - 1) \times (n - r)$. The repeated cubes in the result of transformations are taken out and the set D_2 is formed.
6. The sets D_1 and D_2 are logically intersected in the result of which the set of subcubes D_A including the source node A is formed.

3.1 Routing Algorithm In Faulty Hypercube

The primary data for the algorithm are as follows: A - source node, B - target node, F (or F_{min}) - the set of faulty nodes, L - the set of faulty links (1-cubes).

As shown above, the routing from the source node to target node is realized in a few stages. The set of current sources $G_i = \{ c_{i1} c_{i2} \dots c_{ij} \dots c_{imi} \}$ is defined in stage i . These nodes might not be processed together, they could be separately processed. If we indicate the processing source as P , then the source with the number j , processing in stage i will be $P = c_{ij}$. Thus:

1. Begin. $i = 1, j = 1, c_{11} = A, G_1 = c_{11}, m_i = 1$
2. $P = c_{ij}, G_i = G_i \setminus c_{ij}, m_i = m_i - 1$
3. $D_P = \gamma(F, P)$. Here instead of F, F_{min} may be used.

If the cube P is #-subtracted from the cube $d_g \in D_P$, then we obtain the cubes which can be reached from the node P through the cube d_g . However, the d_g cube may have the arbitrary dimension. Therefore, resulting cubes are not always cubes. For the definition on exact path it is necessary to present the result mentioned above by the 1-cubes. For this purpose, when processing cube d_g that has a dimension $r \geq 2$, it is necessary to divide it into r piece 1-cubes. This process is important for omitting the faulty links outside the algorithm. We indicate the dividing process as $1 \times r \rightarrow r \times 1$.

4. If $D_P = P$, then "P is deadlock node" and goto 18, else 5.
5. For all $d_g \in D_P$ the transformation $1 \times r \rightarrow r \times 1$ is realized, the repeated cubes are canceled in obtaining the result and the set D_{P1} is formed.
6. The operation $D_{P1N} = D_{P1} \setminus L$ is realized and the set of nonfaulty links (1-cubes) is found.
7. If $D_{P1N} = \phi$ then "P is deadlock node according to link" and goto 18, else 8.
8. Then for node P the current targets may be found. These targets will be current sources for the next stage. Consequently:
8. $i = i + 1, j = 1, g = 1, G_i = \phi, m_i = 0$;
9. $c_{ij} = d_g \# P$, here $d_g \in D_{P1N}, m_i = m_i + 1$
10. $G_i = G_i \cup c_{ij}, D_{P1N} = D_{P1N} \setminus d_g$
11. If $D_{P1N} \neq 0$ then $j = j + 1, g = g + 1$ and goto 9, else 12. Thus we determine the set $G_i = \{ c_{ij} \} \{ j=1, \dots, m \}$, where each element is current target in current stage and will be current source in the next stage. Now it

is necessary to define whether there are real targets among elements of the set G_i . For this:

12. $j = 1$;
13. Hamming distance $H(A,B)$ is defined between $A = c_{ij}$ and B ;
14. If $H(A,B) = 0$ use “ c_{ij} is target” and goto 15, else 16. As other paths may exist, it is necessary to continue searching.
15. $G_i = G_i \setminus c_{ij}$;
16. $j = j + 1$;
17. If $j \leq m_i$, then goto 13, else 18;
18. If $G_i \neq \emptyset$ then goto 19, else 21. In order to obtain the shortest path to target node it is necessary to arrange G_i according to the value of H_{ij} and look for a short path first.
19. $F = F \cup P$ (the used source is included in the set of faulty nodes);
20. $j = 1$ and goto 2;

If the set of current targets G_i is not empty, then to use all its elements as source separately, it is necessary to return to step 2. If G_i is empty, then we pass to step 21 and the elements of the preceding set of target, which was not used as source, will now be used as source.

21. $i = i - 1$;
22. If $i > 1$, then goto 18, else 23;
23. End.

3.2 Implementation And Implementation Results

This algorithm is implemented in UNIX platform using C programming language. The results are taken in various dimensions while various faulty nodes exist in the system.

For every different dimension and number of faulty nodes, 100 different samples are taken and graphs are constructed with their average values. We've taken source and target nodes in two different methods;

1. Source and target nodes are independent from each other and chosen randomly.
2. Source is chosen randomly, target is taken as n-hamming distance from source (i.e source=0100, target =1011)

Consider that n = dimension of hypercube, l = number of faulty nodes, H.D= Hamming Distance.

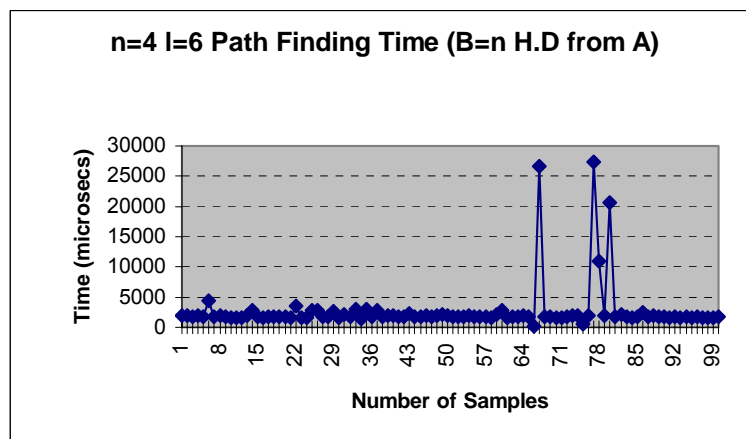


Figure 3.1 Path Finding Time when $n=4$, $l=6$

Figure 3.1 shows the time (in microseconds) , each sample used to find a path from source to destination when $n=4$ and $l=6$.

Figure 3.2 shows the average percentages of path availability (from 100 samples), when $n = 4$ and $l = \{1,2,\dots,12\}$.

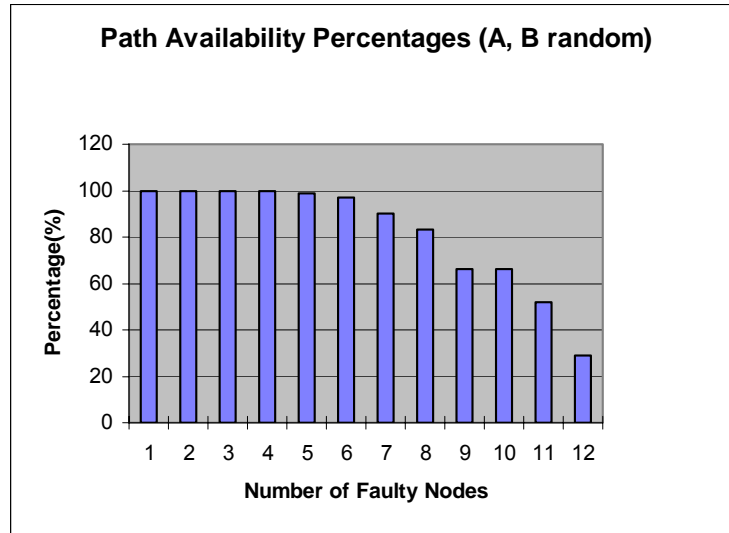


Figure 3.2 Path Availability Percentages when $n=4$, $l = \{1,2,\dots,12\}$

Figure 3.3 shows the average time (from 100 samples) used to find a path between source and destination, when $n = 4$ and $l = \{1,2,\dots,12\}$.

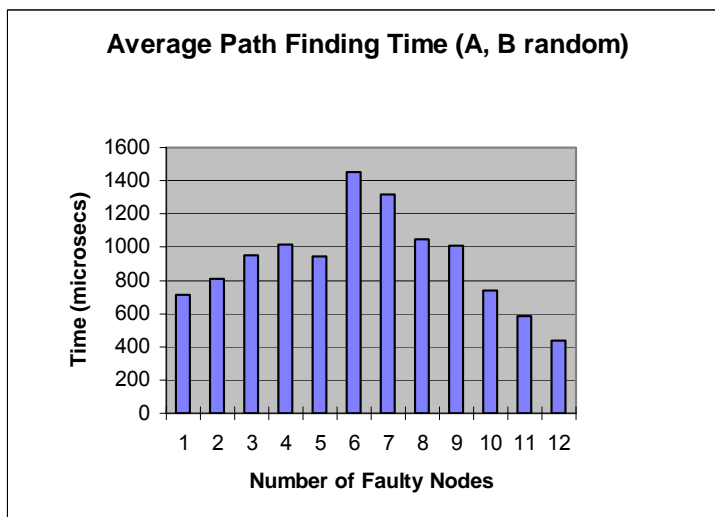


Figure 3.3 Average Path Finding Time when $n=4$, $l = \{1,2,\dots,12\}$

By using this algorithm, we can find all the alternative routes from source to destination nodes. Figure 3.4 shows the number of alternative routes when $n=4$ and $l=5$.

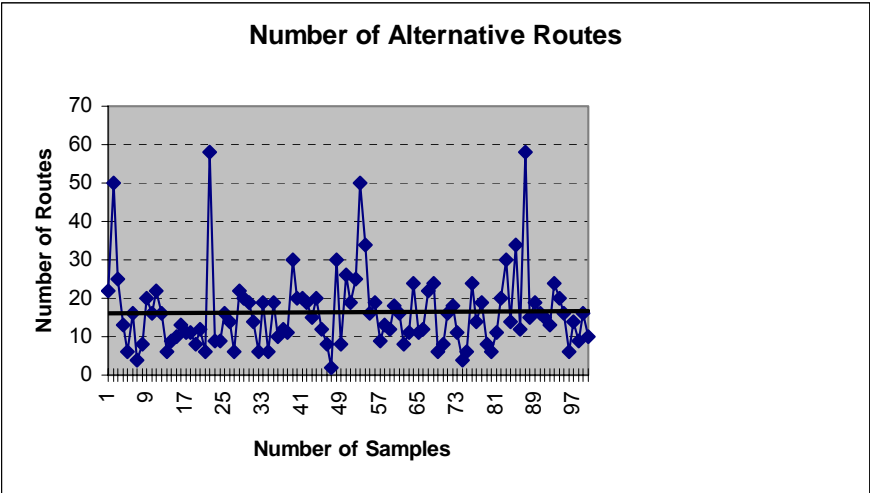


Figure 3.4 Number of Alternative Routes when $n=4$, $l=5$

4 EXTENSION OF HEALTHY SUBCUBES SET IN A FAULTY HYPERCUBE MULTIPROCESSOR

In this algorithm, procedures are provided to overcome the effects of faulty components and to exploit a fault-free architecture in the presence of node and link faults. We consider node design, where each node contains two units, namely, a computational part (CPP) and a communication part (CMP) that can operate independently. The CMP is often called a *router*. The routers have the capability of forwarding the messages from other routers toward the destination node.

The CPP consists of a node processor and some local memory. The CMP or router has a crossbar switch with $(n+1)$ -inputs and $(n+1)$ -outputs. The neighboring nodes are connected through n -input and n -output links. The processor, attached to the router, uses the other two lines. The router can connect multiple inputs to multiple outputs simultaneously as long as there is no destination conflict. If multiple messages are to be delivered to the processor, it is assumed that the router can accept all of the messages. The router is responsible for all communications. It sends messages generated by the local processor over the system to the destination node. Each router compares the destination address of a message with its own address. If they match, the message is delivered to the local processor. Otherwise, the router chooses one of its neighboring nodes to transfer the message.

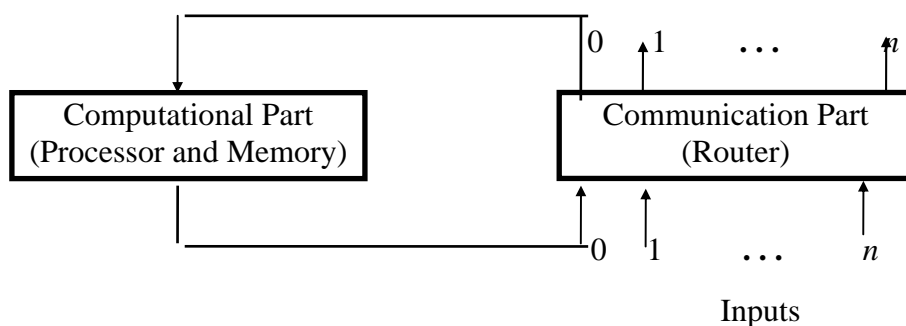


Figure 4.1 The structure of hypercube node

In a faulty hypercube where it is necessary to perform reliable data communication, subcube allocation and similar processes, a faulty node is generally not considered as a node consisting of two parts. But in a faulty node, different types of faults may take place, which must be taken into consideration. Possible cases of faults in a faulty node are shown in Table 4.1. In this table, we show fault and faultless cases by 0 and 1, respectively.

Case No	Computation. Part	Communication Part	Situation of Nodes and Links
1	0	0	Node is faulty
2	0	1	CPP is faulty, CMP is not faulty
3	1	0	CPP is not faulty, CMP is faulty
4	1	1	Node is not faulty
5	1	F-H*	CPP is not faulty, some links in CMP are faulty
6	0	F-H*	CPP is faulty, some links in CMP are faulty

Table 4.1 Fault and faultless of CPP and CMP

If a link or more links (but not all links) in a CMP are faulty (Case No 5), then the respective node can communicate only with definite neighbors. In this case, in order to reach a set of healthy subcubes, we must subtract only a link (or links) without the nodes incident to this link (links) from the set of the subcubes. The coordinate subtraction operation does not directly allow to make this procedure. One way to obtain this procedure is the splitting of m -cube ($m \geq 2$) to m pieces of 1-cubes and execute the subtract operation of only the faulty links from the set of subcubes. But in many cases, the splitting process in hypercube is not desirable (for example, in subcube and task allocation problems). On the contrary, in these cases it is required to have the cubes (subcubes) with dimensions as greater as possible.

4.1 THE ONLY NODES AND ONLY LINKS SUBTRACTION PROCEDURES

A. Subtract Only Nodes

When the CPP of a node V is faulty and CMP is not faulty, that means this node may participate in the communication processes only. That is, the CPP can not receive and send any information to other nodes. We may say that the CPP is isolated. Therefore the communication topology will remain as before, when CPP of this node was not faulty.

When CPP of a node is faulty, a cube that includes this node will have the possibility to communicate messages between adjacent nodes excluding itself. It can be seen from here that the fault of a CPP does not influence the communication processes between the nodes of hypercube. But we must show this node in the description of the cube, in order to exclude it from further computational processes. Therefore, we will show it as $\{A/V\}$, where V is a node from the subcubes in a set A and CPP of this V is faulty.

B. Subtract Only Links

When a CPP is not faulty and one or more links (but not all links) of this node's CMP are faulty (Case No 5 in Table), in order to reach a set of healthy subcubes, we must subtract only a link (or links) from the other set of subcubes. It is clear that a $\#$ -operation will also discard the nodes incident to these faulty links. But as a matter of fact, these nodes are not faulty and they must remain in the set of healthy subcubes, say A . Consequently, the healthy links incident to these nodes will also be discarded from the set A . If we add these healthy links to the set A we will obtain the set called *partly extended set*, say P_E . But in the set P_E there may be the subcubes, which can unite and organize the subcubes with more dimensions. In the case where we obtain the subcubes with more dimensions in the set P_E , we call this set the *extended set*, say E .

When we will execute the subtract operation $S=A \# L^{U,V}$ in order to obtain the fault-free set, in this set the nodes U and V will be absent, though they are not faulty. In this manner, the links incident to the nodes U and V will be absent in the set S , though they are also not faulty, except the link $L^{U,V}$. On the other hand, if we add these fault-free links incident to the nodes U and V (except the link $L^{U,V}$), we will obtain all the fault-free links and nodes in the set S . Thus we can execute

$$P_E = S \cup (L^U, L^V / L^{U,V}),$$

where L^U, L^V are the links incident to the nodes U and V , respectively and $L^{U,V}$ is the link between the nodes U and V . The symbol $/$ shows exception (subtraction) the link $L^{U,V}$ from the set of links incident to the nodes U and V .

In order to achieve the removal of the set of nonfaulty links, we must take the links incident to faulty links. For this, in binary representation of a link (1-cube), we change the first bit (from right to left) by the symbol $*$, if it is not the symbol $*$. Since any link has one symbol $*$, instead of this symbol we first put 0 and then 1. The rest of bits does not change. Thus we obtain the first incident links. We continue this procedure (Fig. 3) with the second bit in 1-cube. Repeating this process with all the bits of 1-cube we obtain the set of links, incident to the faulty link. But among these links may be a link (links), which is (are) faulty. Therefore, we must check this set with the set of faulty links. If there are coincidences, these links will be removed out from the obtained set and thus we will achieve the set of healthy (fault-free) links, say L_{ff} . The procedure “adding of the nonfaulty links” is applied to the set S and thus we obtain the intermediate set E .

Procedure Finding_nonfaulty_links ($L_f^{u,v}, L_f^u, L_f^v$)

begin

for $k = 1$ to n do

input L_f^u, L_f^v

$L_{nf1}^k := L_f^{u,v}, L_{nf0}^k := L_f^{u,v}$

endfor

for $k = 1$ to n do

for $i = 1$ to n do

if $L_f^{u,v}[i] = *$ then

$L_{nf1}^k := 1, L_{nf0}^k := 0$

$m := i$

endfor

endfor

for $i = 1$ to n do

if $i \neq m$ then

$L_{nf1}^i := *, L_{nf0}^i := *$

else

$L_{nf1}^i := \phi, L_{nf0}^i := \phi$

endfor

for $i = 1$ to n do

if $L_{nf1}^i = L_f^u$ or $L_{nf1}^i = L_f^v$

then $L_{nf1}^i := \phi$

if $L_{nf0}^i = L_f^u$ or $L_{nf0}^i = L_f^v$

then $L_{nf0}^i := \phi$

endfor

end

The full subcube recognition ability is one of the main issues of the hypercube processes. The procedure to increase the sizes of the available subcubes in the set E is presented in the below procedure.

Procedure Increase_size_subcubes (S, L_{nf})

```

begin
  for  $i=1$  to  $n-1$  do
     $E^i = S \star L_{nf}$ 
    Define in  $E^i$ 
    (1) 2- and more subcubes (let  $E^M$ )
    (2) These of 1- and 0-cubes, which are not included in the
        subcubes in  $E^M$  (let  $E^L$ )
     $E^N := E^M \cup E^L$ 
     $L_{nf} := E^N$ 
  endfor
end

```

4.2 Implementation And Implementation Results

This algorithm is implemented in UNIX platform using C programming language. The results are taken in various dimensions while various faulty nodes exist in the system.

For every different dimension and number of faulty nodes, 100 different samples are taken and graphs are constructed with their average values. We've taken source and target nodes in two different methods;

1. Source and target nodes are independent from each other and chosen randomly.

2. Source is chosen randomly, target is taken as n-hamming distance from source (i.e source=0100, target =1011)

Consider that n = dimension of hypercube, l = number of faulty nodes, H.D= Hamming Distance.

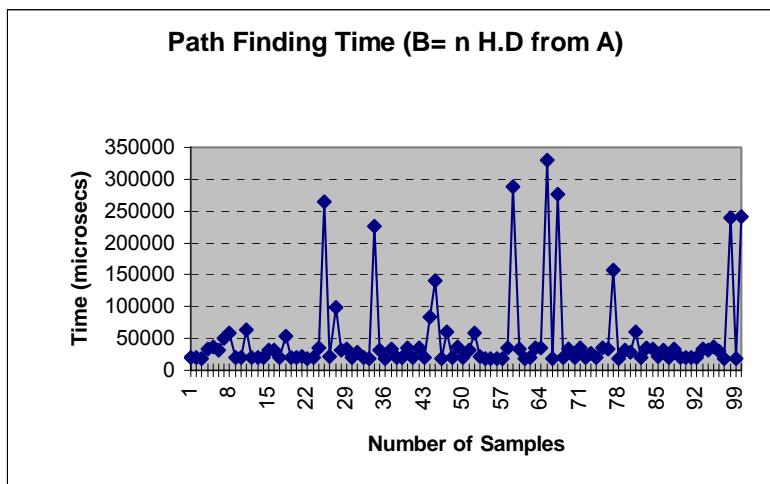


Figure 4.2 Path Finding Time when $n=4$, $l=12$

This graph show time values when B is n hamming distance from A. Figure 4.2 shows time values when both A and B are randomly chosen.

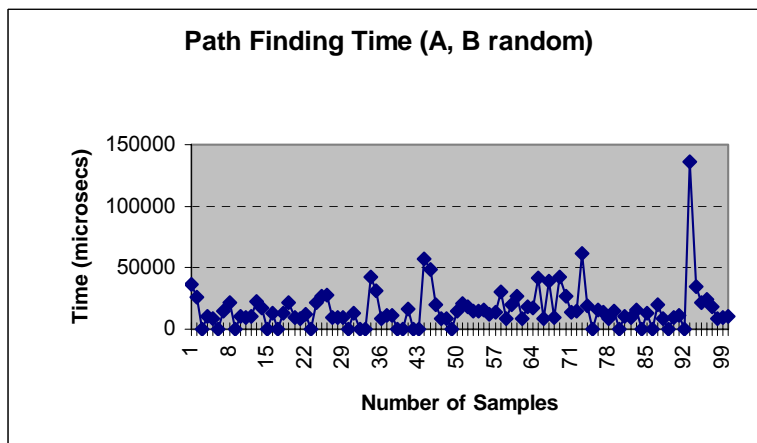


Figure 4.2 Path Finding Time when $n=4$, $l=12$

Figure 4.3 shows the average percentages of path availability (from 100 samples), when $n = 4$ and $l = \{1,2,\dots,12\}$.

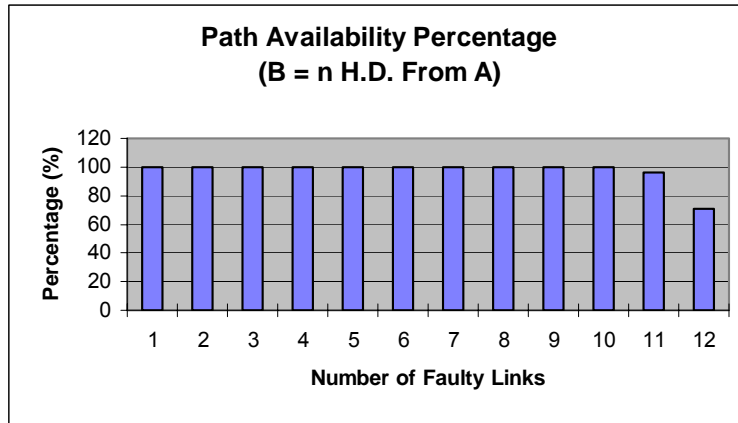


Figure 4.3 Path Availability Percentages when $n=4$, $l = \{1,2,\dots,12\}$

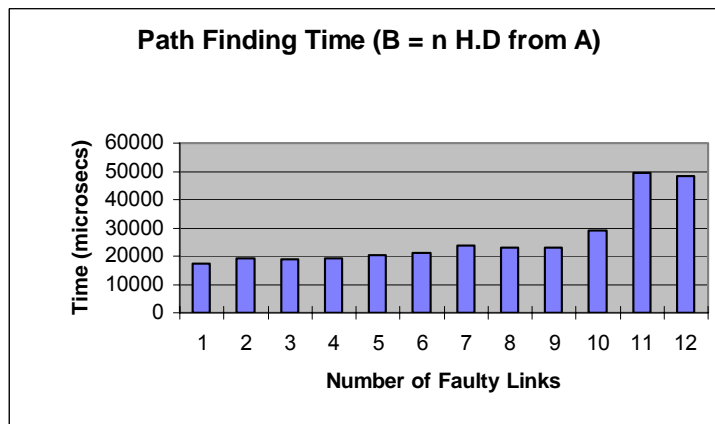


Figure 4.4 Average Path Finding Time when $n=4$, $l = \{1,2,\dots,12\}$

By using this algorithm, we can find all the alternative routes from source to destination nodes. Figure 4.5 shows the number of alternative routes when $n=4$ and $l=8$.

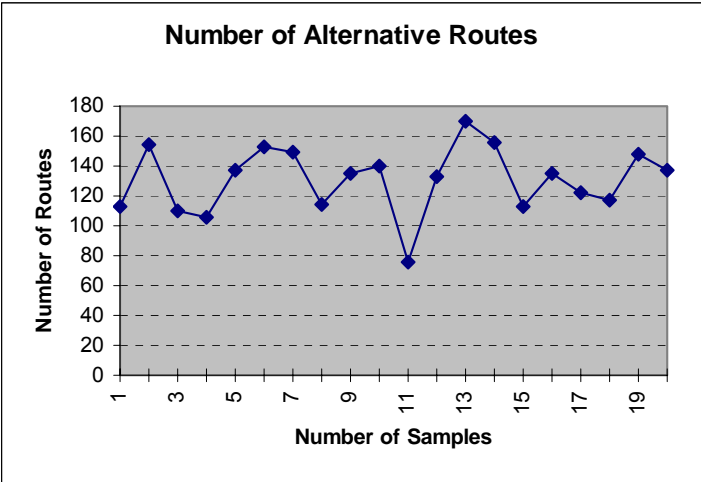


Figure 4.5 Alternate routes when $n=4$, $l=8$

5 PROCEDURE FOR DEFINITION OF NEIGHBORHOOD IN FAULTY HYPERCUBE MULTIPROCESSOR

The procedure developed in this algorithm allows us to calculate firstly a configuration that has only non-faulty components. Secondly we search the subcubes, covered the source and target vertices (nodes). Thirdly we look for intersection between fault free subcubes, with the aim of definition of their common parts. Thus, we have a possibility to define a path between the source and destination nodes, using only the non-faulty cubes (subcubes). It is clear that the source node and the target node must not be placed in isolated subcubes. The procedure does not depend on the number of faulty components and are not affected by the scattering of these components in the hypercube.

In order to determine the set of non-faulty and maximal subcubes (NMS) in faulty hypercube we use the following Theorem [1, 2].

Theorem 1. If F is a set of complete faulty cubes, then the set NMS of n -cube including F is determined by the # -subtracting the set F from the complete n -cube and avoiding non-maximal cubes from the result.

After determination of a set NMS we can realize different operations over the cubes, which are contained in the set of NMS. It is important to search the method of determination of NMS, each of which includes a current source in every stage of routing process. This method is based on the Theorem 2.

Theorem 2. If $A = a_1 a_2 \dots a_j \dots a_n$ is any using vertex and $B_i = b^1_1 b^1_2 \dots b^1_j \dots b^1_n$, $B_i \hat{=} F$ is any forbidden cube, then for calculation of the prime implicant covered vertex A by procedure $K_i = K_{i-1} \# B_i$, $i = 1, 2, \dots, m$, $K_0 = ** \dots *$, it is necessary and sufficiently preserved for only all such b^i_j , which have opposite value with respect to corresponding a_i .

Using the Theorem 2 in the search of subcube (prime implicant), covered the vertex A , the faulty cube $B_i = b^1_1 b^1_2 \dots b^1_j \dots b^1_n$ can be transformed to cube $Q_i = q^1_1 q^1_2 \dots q^1_j \dots q^1_n$ by the means of rule:

$$Q_j^i = * \text{ if } b_j^i = * \text{ or } a_j = b_j^i$$

$$Q_j^i = b_j^i \text{ if } b_j^i = \bar{a}_j$$

Thus, one can define the fault free subcubes including the source node, also the target node. Now we will look for the existing connection from the subcube (s) including the source node (let source subcube or subcubes) to the subcubes, including the target node (let target subcube or subcubes).

In order to describe the developed procedure by a formal way; Let $S=s_1s_2\dots s_n$ is a source node and $T=t_1t_2\dots t_n$ is a target node in an n -dimensional hypercube. Let F is a set of complete faulty cubes. Then:

1. Definition of the set NMS.
2. Definition $S \otimes T$. If $S \cap T \neq \emptyset$ then goto 10, else goto 3.
3. Definition the source subcube(s) and target subcube(s) by (1).
4. Definition $S \otimes T$. If $S \cap T \neq \emptyset$ then goto 10, else goto 5.
5. From the set NMS is subtracted source subcube(s) and target subcube(s). The obtained set be NMSS
6. Put in order the set NMSS on dimension. The ordered set is NMSO.
7. Execution of the star product operation the source subcube(s) with NMSO. Obtained nonempty set is NMSN.
8. Control if some elements of NMSN include the other elements. If such elements exist remove them from the set NMSN. Put in order the set NMSN.
9. Goto 4.
10. End.

5.1 Implementation And Implementation Results

This algorithm is implemented in UNIX platform using C programming language. We have implemented this algorithm with two different methods. First method uses only “Sharp-Product” method to obtain SNMS, second method uses “Extended Set of Healthy Subcubes” method to obtain SNMS. The results are taken in various dimensions while various faulty nodes exist in the system.

For every different dimension and number of faulty nodes, 100 different samples are taken and graphs are constructed with their average values. We’ve taken source and target nodes in two different methods;

3. Source and target nodes are independent from each other and chosen randomly.
4. Source is chosen randomly, target is taken as n -hamming distance from source (i.e source=0100, target =1011)

Consider that n = dimension of hypercube, l = number of faulty nodes, H.D= Hamming Distance.

5.1.1 Sharp-Product Method Results

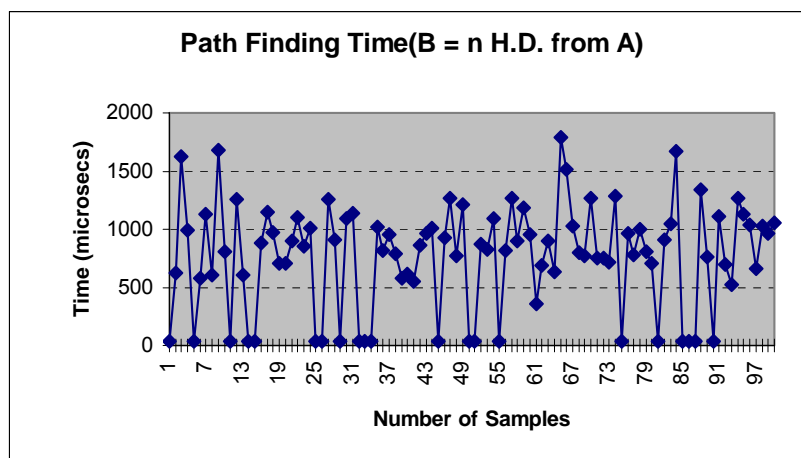


Figure 5.1 Path Finding Time when $n=4$, $l=7$ for 100 samples

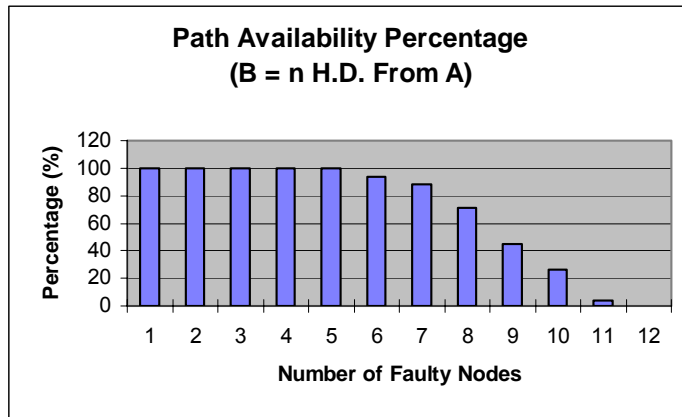


Figure 5.2 Path Availability Percentages when $n=4$, $I = \{1,2,\dots,12\}$

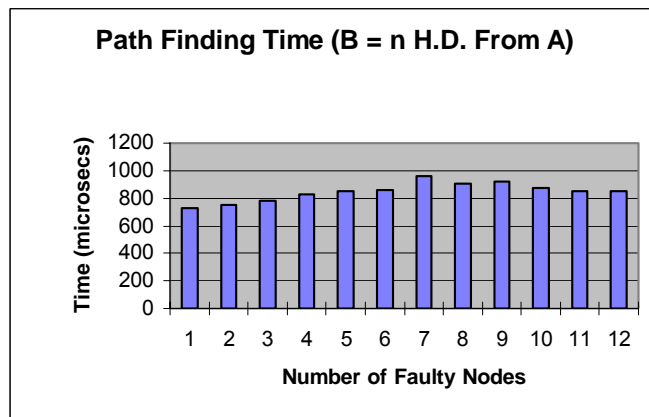


Figure 5.3 Path Finding Time when $n=4$, $I = \{1,2,\dots,12\}$

5.1.2 Extended Set of Healthy Subcubes Method

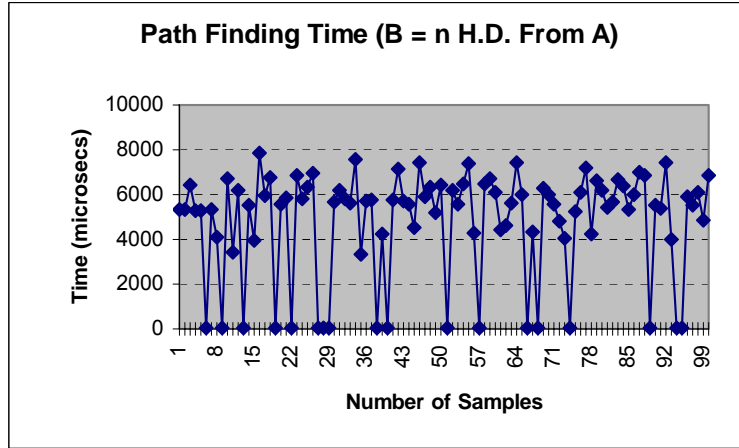


Figure 5.4 Path Finding Time when $n=4$, $l=7$ for 100 samples

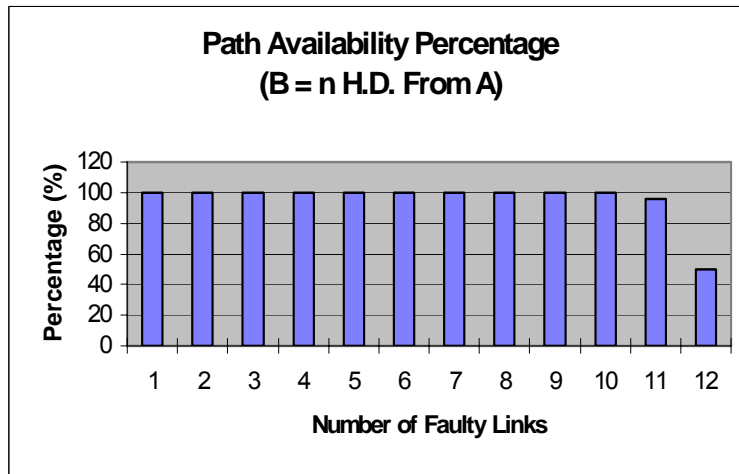


Figure 5.5 Path Availability Percentages when $n=4$, $l = \{1,2,\dots,12\}$

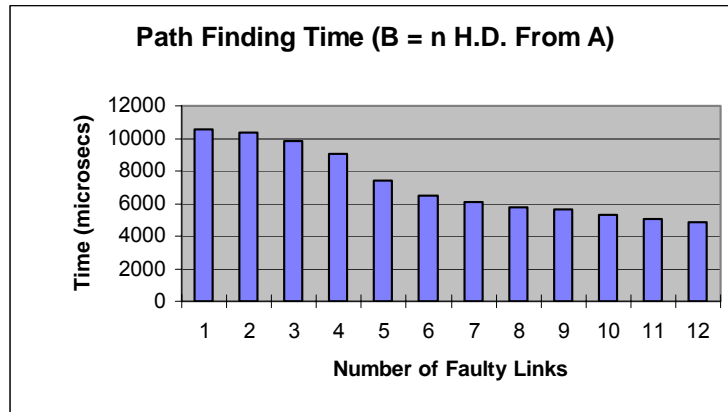


Figure 5.6 Path Finding Time when $n=4$, $l = \{1,2,\dots,12\}$

6 INTEL iPSC/860 SIMULATOR IMPLEMENTATIONS AND RESULTS

We've implemented the algorithms described in section 3,4 and 5 in Intel iPSC/860 Simulation environment. Implementations are done in two different methods, "Static Routing" and "Dynamic Routing".

Faulty nodes/links are defined before simulation starts in static routing method. In dynamic routing method, simulation starts with no faulty link/node information. System gets information with help of consensus algorithms and intra-cube messages.

In both methods, there are two programs, HOST program and NODE program.

HOST : HOST program is the interface program between user and hypercube node programs. HOST will be executed by user and it'll initiate nodes of hypercube and load NODE programs. According to our implementation, when HOST is executed, it loads NODE programs and starts to wait for a message from a node. This message can be either "**DESTINATION_FOUND**" message or "**NO_PATH_AVAILABLE**" message.

"**DESTINATION_FOUND**" message can be send from destination node, indicating that message created by source node has reached its destination. After receiving this message, HOST kills the allocated hypercube and itself.

"**NO_PATH_AVAILABLE**" message can be send from any node (for static routing method, only source node can send) indicating that target node cannot be reached. After receiving this message, HOST kills the allocated hypercube and itself.

NODE : Node program is responsible from routing functions, messaging and making consensus (for dynamic routing) and informing HOST about the status.

In this section, details about implementation and results will be given.

6.1 Static Routing

In static routing method, HOST program is the same as explained above. NODE program is implemented as;

When HOST loads node programs, every node gets faulty nodes/links, source and destination information. Every node checks if it is the source node, or not. If node A is the source node, it has to execute routing algorithm and find a path to target node.

```
if mynode() == src
    find_route(source, destination, F);
    if path_available
        then
            /*include path information to message, which will be sent to next
            node. So that, every node on the path will use this information
            instead of executing find_route() */

            construct_msg_to_send()
            send_msg_to_next_node()
        else
            construct_no_path_available_msg()
            send_msg_to_HOST(NO_PATH_AVAILABLE)
        endif
    endif
endif
```

Figure 6.1 Algorithm executed by source node

If node is the destination node;

```
if mynode() == dest
    then
        wait_msg()
        send_msg_to_HOST(DESTINATION_FOUND)
    endif
```

Figure 6.2 Algorithm executed by destination node

If node is not source, destination or faulty, then it is a tandem node. It can be on the path from source to destination.

```
if mynode() == tandem_node
    then
        wait_msg()
        /*If no msg. received, then this node is not on the chosen path .*/
        find_next_node_from_message_received()
        send_msg_to_next_node()
    endif
```

Figure 6.3 Algorithm executed by tandem node

According to these algorithms, we have implemented HOST and NODE programs to get static routing performances of algorithms. We get results for $n=4$, $k = \{4,5,6,7,8\}$. Figure 6.4 shows static routing performances of algorithms.

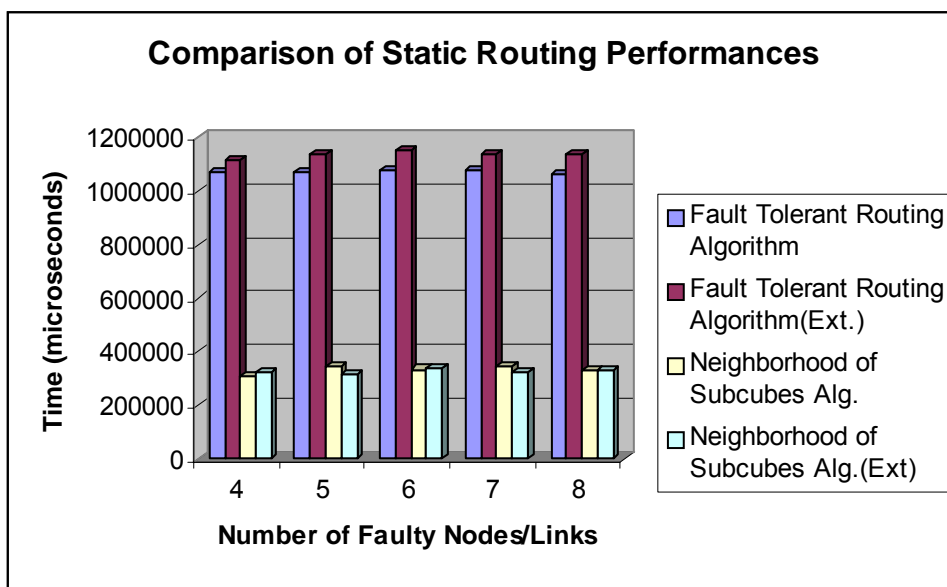


Figure 6.4 Static Routing Performances of Algorithms

6.2 Dynamic Routing

In this section, we developed and implemented two different consensus methods to achieve dynamic routing in hypercube simulator. First method relies on “all-to-all” consensus algorithm, second method relies on “only-neighbors” consensus algorithm.

6.2.1 All-to-all Consensus Algorithm

Every node (current source), who has to obtain information about the current status of hypercube sends a STATUS message to all nodes of the hypercube. When a node receives STATUS message, sends a STATUS_ACK message, including its faulty/non-faulty status data. When STATUS requesting node receives all STATUS_ACK messages, it executes routing algorithm with the information obtained from other nodes.

6.2.2 Only – Neighbors Consensus Algorithm

Every node (current source), who has to obtain information about the current status of hypercube sends a STATUS message to all its **neighbor** nodes. When a node receives STATUS message, first checks if this message has already been received, if not transmits this message to its own neighbors (except the node which sent this message) and sends a STATUS_ACK message to all of its neighbors, including its faulty/non-faulty status data. When STATUS requesting node receives all STATUS_ACK messages, it executes routing algorithm with the information obtained from other nodes.

In order to prevent a message passing from one node to another continuously, we used two different data structures; STATUS_LIST[] and STATUS_ACK_LIST[][]. When a STATUS message received from a neighbor node, we check the STATUS_LIST with message number. Since every STATUS message has a unique number, if the STATUS_LIST[message_number] = 0 then we can accept this message and process it, otherwise message is a duplicate one, we won't accept and process it, it'll simply be ignored. Same control mechanism is valid for STATUS_ACK messages. When a STATUS_ACK is received, we check STATUS_ACK_LIST[message_number][node_number], here message_number is STATUS message number, node_number is the number of the node which has sent this STATUS_ACK message. If for example, STATUS_ACK_LIST[1][6]=1, this means that, 6th node has sent STATUS_ACK message to 1st STATUS message.

6.2.3 Communication Protocol Between Hypercube Nodes

Nodes are communicating in this hypercube system using a communication protocol developed for this purpose. This protocol has four main messages; STATUS, STATUS_ACK, DATA and MSG.

STATUS : This message is sent from current source node to request/give information about nodes in order to reach a consensus. We use STATUS with four different parameters, INFO_REQUEST, READY, NO_PATH_FOUND, DEST_FOUND.

STATUS(INFO_REQUEST) is used to request information from nodes. STATUS_ACK message is expected after sending this message.

STATUS(READY) is used to inform next node, that a message is going to be sent to it, to route to destination.

STATUS(NO_PATH_FOUND) message is sent to all nodes, to inform that an available path couldn't be found from current node to destination.

STATUS(DEST_FOUND) message is sent to all nodes, to inform that routing is successfully ended.

STATUS_ACK : This message is sent after receiving STATUS message from current source node. This message contains information(faulty/non-faulty) about the node itself and its links.

DATA : This message is sent from current source node to next source node, which contains information to be delivered to destination node.

MSG : This message can be sent from any node to HOST node. It includes information about the routing status. NO_PATH_AVAILABLE or DEST_FOUND information can be the types of this message.

We've implemented this algorithms in Intel iPSC/860 Simulator environment. Figure 6.5 shows the results we've obtained using all-to-all consensus algorithm.

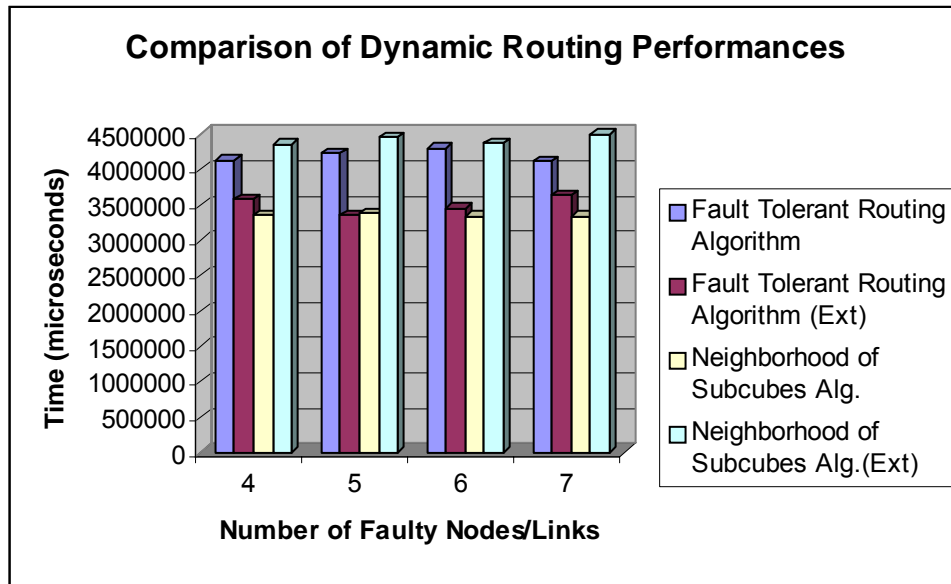


Figure 6.5 Comparison of Routing Performances using All-to-all consensus algorithm

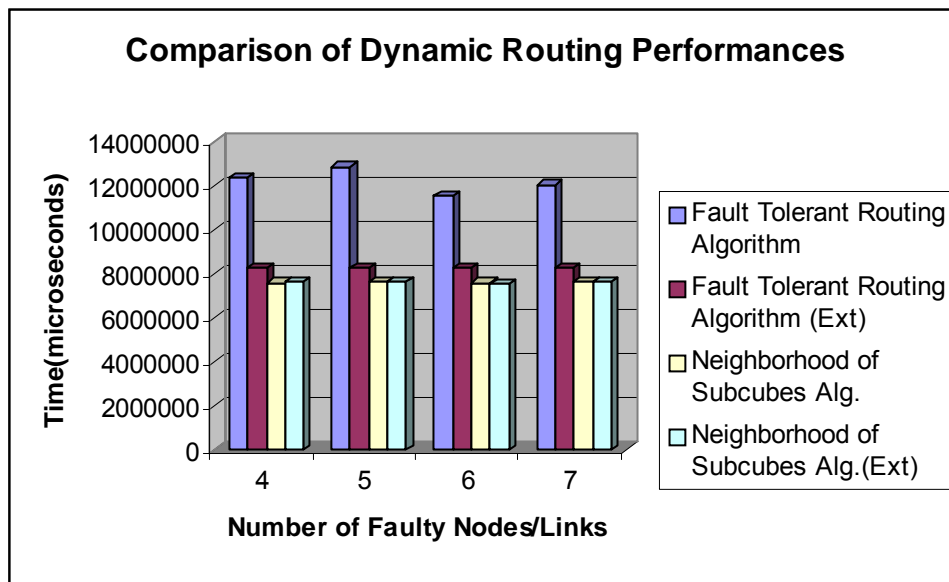


Figure 6.6 Comparison of Routing Performances using Only-Neighbors consensus algorithm

7 CONCLUSION AND DISCUSSION

We've implemented the algorithms and discussed the results at each section. It can be seen in graphs that algorithms that use "Extended set of healthy subcubes" procedures has a higher path availability percentage than the algorithms that use only "Sharp-product" to find SNMS. Figure 7.1 shows this conclusion is true for our implementations.

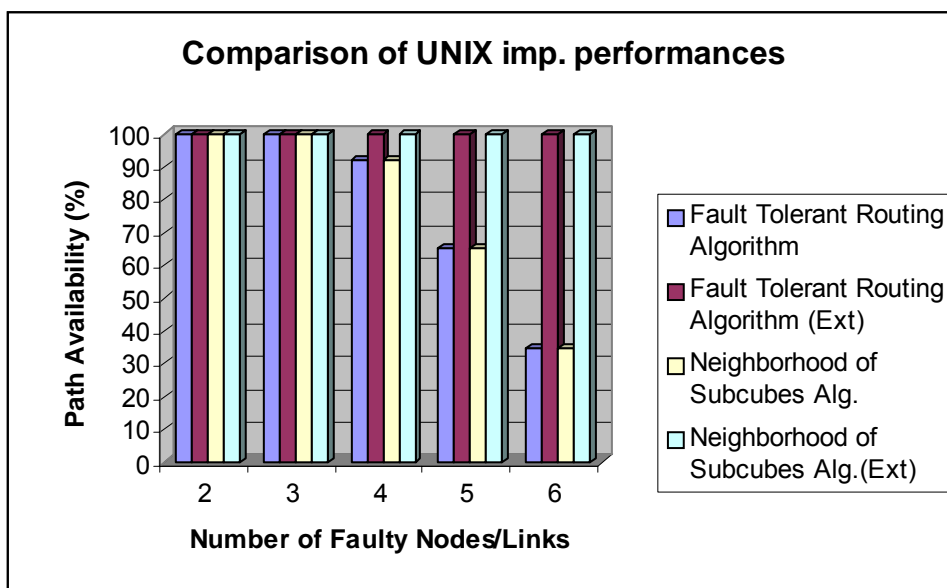


Figure 7.1 Comparison of Path Availability Percentages

Figures in previous sections also show us routing time performances of algorithms. According to these graphs, Section 5 algorithms show best performance on UNIX platform and also iPSC platform. For iPSC platforms comparison results, please refer Section 6. Figure 7.2 shows us comparison results on UNIX platform.

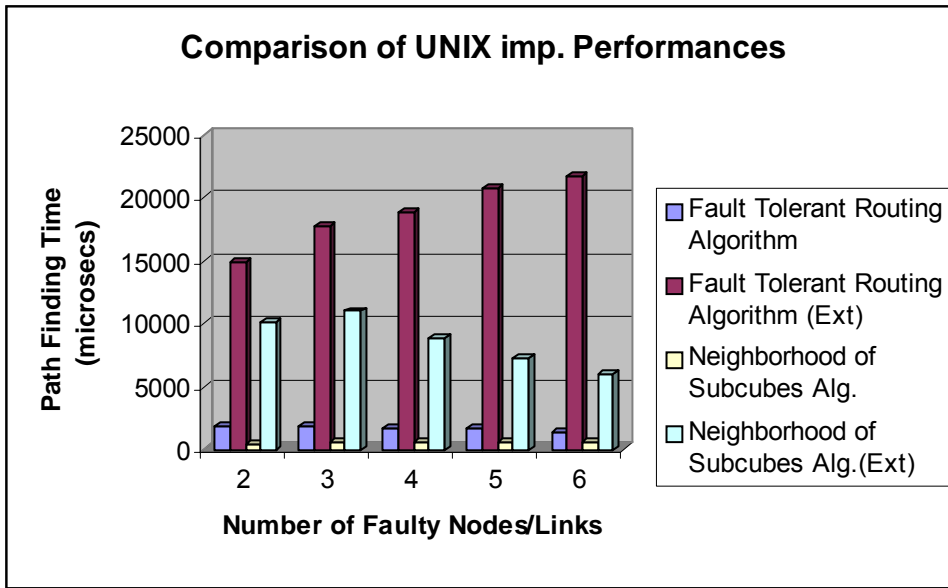


Figure 7.2 Comparison of Path Finding Time Performances

We've stated that, we use two different methods to take destination (B). We either take B at Hamming Distance from A (source), or independent from A. These two methods make difference in path availability percentages. Figure 7.3 shows that, if B is independent from A, path availability percentage will be higher.

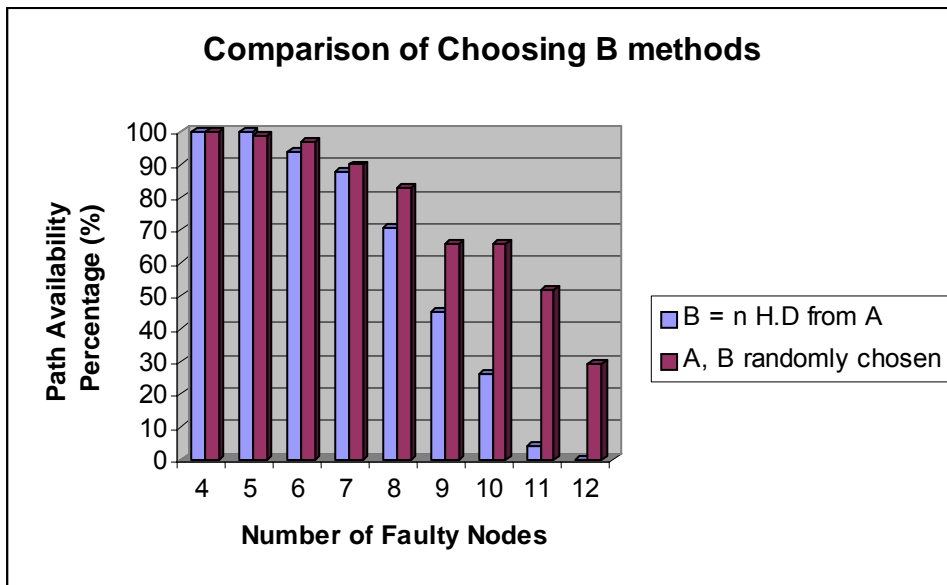


Figure 7.3 Comparison of "Choosing B" methods

References

[1] N. M. Allahverdi, S. S. Kahramanli, K. Erciyeş, "A Fault Tolerant Routing Algorithm in Hypercube with Application Cube Algebra", To appear in Journal of Systems Architecture, 1999.

[2] N. M. Allahverdi, "Procedure For Definition Neighborhood In Faulty Hypercube Multiprocessor", 1999

[3] N. M. Allahverdi, K. Erciyeş, "Extension of Non-faulty Subcubes Set In a Faulty Hypercube Multiprocessor", in Proceedings of the International Fourth Symposium on High Performance Computing and Interconnection Networks, Durham, USA, Vol.IV, pp.115-118, 1998.

[4] R. H. Urbano, R. K. A. Mueller, "Topological Method for the Determination of the Minimal Forms of a Boolean Function", IRE Trans.on Electronic Comput. EC-6, 4 (Sept. 1956):126-132, 1956.

[5] Y. Chang, L. Bhuyan, "Subcube Fault Tolerance in Hypercube Multiprocessors", IEEE Trans. Comput., 44:1108-1120, 1995.

[6] S. Rai, J. L. Trahan, T. Smailis, "Processor Allocation in Hypercube Multiprocessors", IEEE Trans. on Paral. Distrib. Syst., 6:606-616, 1995.

[7] J. P. Roth, "Algebraic Topological Methods for the Synthesis of Switching Systems in n-Variables", The Institute for Advanced Study, Princeton, New Jersey, ESP56-02, April, 1956.

[8] Ncube, Ncube 6400 Processor Manual, Ncube V1.0, Beavertor, OR, 1990.

[9] M. R. Dagenais, V. K. Agarwal , N. C. Rumin, "McBOOLE: A New Procedure for Exact Logic Minimization", IEEE Trans. Comput.-Aided Design:Vol.CAD-5:229-237, 1986.

[10] D. L. Dietmeyer, "Logic Design of Digital Systems", Boston, MA: Allyn and Bacon, 1979.

[11] T. H. Dunigan, "Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes", Parallel Computing, 17:1285-1302, 1991.

[12] S. Dutt, J. P. Hayes, "Subcube Allocation in Hypercube Computers", IEEE Trans. Comput., 40:341-352, 1991.

[13] E. M. Nadjafov, S. S. Kahramanov, "On the Synthesis of Multiple Output Switching Scheme", Scientific Notes of Azerbaijan Institute of Petroleum and Chemistry, Vol. IX: 65-69, 1973.

[14] R. E. Miller, Switching Theory, Vol.1, Combination Circuits, New York; John Wiley and Sons, 1965.

[15] G. M. Chiu, S. P. Wu, "A Fault-Tolerant Routing Strategy in Hypercube Multicomputers", IEEE Trans. Comput., 45:143-155, 1996.

[16] F. T. Hady, B. L. Menezes, "The Performance of Crossbar-Based Binary Hypercubes", IEEE Trans. on Comput., 44:1208-1215, 1995.

[17] Intel, iPSC User's Guide, Intel 17455-03, Portland, OR, Oct., 1985.

[18] S.S. Kahramanli, N. M. Allahverdi, "Compact Method of Minimization of Boolean Functions with Multiple Variables", Proc. Intern. Symp. of Application of Computers:433-440, June 1993, Konya, Turkey.

[19] S. S. Kahramanli, N. M. Allahverdi, "Algebraic Approach to Transformations on Hypercube Systems", Mathematical and Computational Applications, 1:50-59, Published by Association for Scientific Research (Turkey), 1996.

[20] J. Kim, R. D. Chita, "Hypercube Communication Delay with Wormhole Routing", IEEE Trans. on Comput., 43:806-814, 1994.

[21] J. Bruck, R. Cypher, D. Soroker, "Tolerating Faults in Hypercubes Using Subcube Partitioning", IEEE Trans. Comput., 41:599-605, 1992.

[22] M. S. Chen, K.G. Shin, "Subcube Allocation and Task Migration", IEEE Trans. Comput., 39:1146-1155, 1990.

[23] M. S. Chen, K. G. Shin, "Depth-first Search Approach for Fault-tolerant Routing in Hypercube Multicomputers", IEEE Trans. on Paral. Distrib. Syst. 1, 2:152-159, 1990.

[24] B. Becker, H. U. Simon, "How Robust Is the n-Cube?", Information and Computation, 77:162-178, 1988.