Fault-Tolerant Routing Strategy in Hypercube Systems

Ge-Ming Chiu and Shui-Pao Wu

Department of Electrical Engineering and Technology
National Taiwan Institute of Technology
Taipei, Taiwan

Abstract

We investigate fault-tolerant routing which aims at finding feasible minimal paths in a faulty hypercube. The concept of an unsafe node *is adopted to identify nodes that may cause routing difficulties. We propose a set of stringent criteria to reduce the number of fault-free nodes that are labeled unsafe. Routing efficiency can thus be improved. An algorithm is presented to facilitate the routing of messages. It only requires each node to maintain information about its local state, whereby each node is in one of four states. The issue of deadlock freeness is also addressed. Furthermore, an alternative deadlock-free routing algorithm, which requires a constant of five virtual networks in wormhole routing, is presented.*

I Introduction

Recently, hypercube systems have drawn considerable attention from many researchers [2, 14, 15]. Hypercube multiprocessors, owing to their regular structure and low diameter, are well suited for parallel processing [9]. Several commercially available hypercube multiprocessors, such as Intel's iPSC-860 and iPSC-2 machine, have been built in recent years [8].

Processors (or nodes) in a hypercube communicate with each other via passing messages. Efficient message routing is critical to the performance of a hypercube. A variety of routing algorithms have been proposed for hypercubes in the past [1, 5, 7, 10, 12]. However, most of these algorithms are not suitable for routing messages when nodes fail in a hypercube. Among the fault-tolerant ones, Gordon and Stout [11] proposed an approach called "sidetracking" to deroute messages in a hypercube with faulty nodes. A message is derouted to a randomly chosen fault-free neighbor when there exists no available link for advancing the message. Routing failures may occur although the probability is low, and excessive message delay may arise. Chen and Shin [4] developed and analyzed a routing scheme based on depth-first search in which backtracking is required when all forward links are blocked by faulty components. Every message must carry an information indicating the dimensions already traversed to avoid routing to the same node except when backtracking is enforced. Time overhead required is excessive when the number of faults is small. A simplified version of the method that tolerates less faults was presented in [3]. Most of foregoing strategies were designed for packet-switching. Further, the issue of deadlock freeness was not considered. Lee and Hayes [13] proposed a different fault-tolerant routing scheme which is based on the concept of *unsafe* node. Message routing is to avoid unsafe nodes which could lead to communication difficulties or excessive delay. It does not require each message to carry information about its routing history. Instead, each node only has to keep local information about the states of its neighbors. Computing requirement for routing is simplified, and hence time delay incurred at each node is small. They showed that the algorithm can route a message via a path of length no greater than two plus the Hamming distance between source and destination of the message as long as the hypercube is not fully unsafe, which is guaranteed as long as the number of faults is no more than $\lceil n/2 \rceil$ in an n-cube.

In this paper, we develop a routing strategy which adopts a similar philosophy, in its nature, to the one proposed in [13]. The concept of unsafe node and its extension is used in our scheme. A new set of stringent criteria is proposed to identify possibly bad candidates to forward a message. The number of such undesirable nodes is significantly reduced without sacrificing functionality of the mechanism. An efficient algorithm, in which the degree of unsafeness is used, is proposed to facilitate the routing function. We show that a feasible path of length no more than the Hamming distance between the source and destination plus four can always be established as long as the hypercube is not

0363-8928/94 $3.00

fully unsafe. According to our criteria, this condition is guaranteed if the number of faulty nodes is no more than $n-1$ in an n-cube. The issue of deadlock freeness is also addressed. Furthermore, we propose another algorithm which requires only a constant of five virtual networks for wormhole-routing hypercubes.

II Preliminaries

An n-dimensional hypercube, abbreviated n-cube, has $N = 2^n$ processors (or nodes). Each processor T, $0 \leq T \leq N-1$, can be represented by a sequence of n binary digits $(t_{n-1}, ..., t_0)$ where $t_i \in \{0, 1\}$ for $0 \leq i \leq n-1$. The bit t_i is called the i-dimensional bit of node T. Two processors are connected by a bidirectional link if and only if the binary representations of their labels differ in exactly one bit. A link is called an i-dimensional link, if it connects two nodes that differ in their i-dimensional bits. A path between two nodes can be represented as an ordered sequence of consecutive nodes, or equivalently as a sequence of communication links connecting these nodes. The number of links contained in a path is called the length of the path. The Hamming distance between two nodes X and Y, denoted as $H(X,Y)$, is the number of bits in which labels of X and Y differ. A path is said to be *feasible* if it traverses through no faulty components. A feasible path which has the shortest possible length between source and destination nodes is called a *minimal* path.

III Fault-Tolerant Routing Strategy

Faulty components may block and delay routing of messages, rendering a degradation of system performance. Hence, it is important to exploit fault-tolerant capability in routing algorithms by taking advantage of the large amount of nodes and communication links in a hypercube. For instance, there are $r!$ shortest paths between two nodes that are separated by a Hamming distance of r. To illustrate the routing problem, figure 1 shows a 4-dimensional hypercube with four faulty nodes, whose addresses are (0010), (0100),(1000) and (1111), respectively. Suppose that a message is to be sent from node (1101) to node (0000). A feasible minimal path of length 3, such as (1101,1001,0001,0000), exists in the system. However, if node (1101) has no knowledge of the failures of nodes (0100) and (1000), the message may first be sent to node (1100). It turns out that the message cannot possibly be forwarded to its destination through a feasible minimal path. Now consider another case where a message is to be sent from (0110) to (0000). It is obvious that no feasible shortest path of length 2

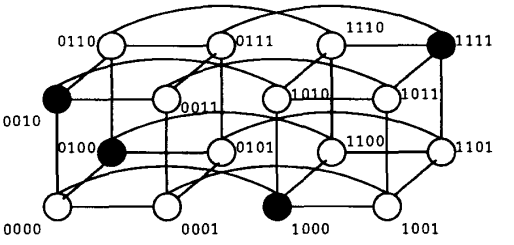

Figure 1: A 4-dimensional hypercube with four faulty (dark) nodes.

exists. If the message is first derouted to node (1110), it will next be forwarded to either of nodes (1010) and (1100), from which no feasible shortest paths are possible and another deroute is necessary. This increases routing delay and makes routing complex. On the other hand, if the message is first forwarded to node (0111) from node (0110), a feasible minimal path of length 4, as opposed to 6, can possibly be found.

It would be desirable to route a message through a feasible minimal path in a faulty hypercube. From previous examples, it is apparent that there exists a tradeoff between the amount of information known to each node and the likeliness for a distributed routing mechanism to route messages through feasible minimal paths. A natural way of defining the amount of information known to each node is in terms of the range of neighborhood of a node whereby the node has information about the states of all nodes within this neighborhood [13]. For example, if each node is aware of the state of every other node in an n-cube, i.e. the associated range of neighborhood is n, one can search through all possible routes to locate a feasible minimal path whenever it has a message to send. This, however, may incur a significant amount of computation time, and require a considerable size of storage to hold the information.

In the following, we investigate the design of a fault-tolerant routing mechanism for faulty hypercubes. Each node contains the states of its nearest neighbors only, a common and natural situation. To facilitate our discussion, consideration of deadlock freeness will not be addressed until section IV.

III.1 The Unsafe Nodes

A fault-tolerant routing strategy should avoid routing a message via a node which may lead to no minimal path. The notion of an *unsafe* node is introduced to indicate a potentially bad choice for a routing function [13]. Obviously, a large number of unsafe nodes will reduce the number of alternative paths

a message can traverse; flexibility of the underlying routing algorithm may be affected. Consider a node, say I, which currently has a message destined for a node D, and the Hamming distance $H(I, D)$ is k. If k equals 2, there are two node-disjoint shortest paths between these nodes, and they are the only such paths. One can determine the existence of a minimal path from node I to node D by examining whether the two nearest neighbors of I which are connected to D are both faulty or not. However, if $k \geq 3$, the mere faulty/nonfaulty state information of node I's nearest neighbors is not informative enough in preventing a message from reaching a "bad" next node as illustrated in previous example. This lays basis for the following definition.

Definition 1 *A fault-free node is defined as an* unsafe *node if it has either two or more faulty nearest neighbors, or three or more faulty or unsafe nearest neighbors.*

A nonfaulty node that is not unsafe is called a *safe* node. With the definition, unsafe nodes can be identified recursively in a hypercube. For example, figure 2 shows a 4-cube with a fault pattern which is identical to that shown in figure 1. Node (0110) is unsafe as it has two faulty neighbors, namely (0010) and (0100). Similarly, nodes (1010) and (1100) are also unsafe. Node (1110) will subsequently be marked unsafe due to the aforementioned three unsafe nodes. Unsafe nodes are shown with gray filling in figure 2. Message

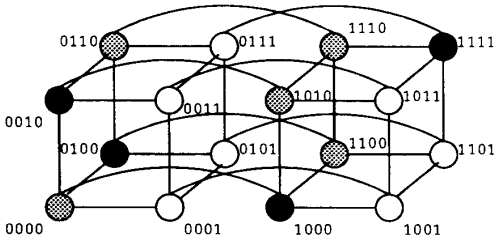

Figure 2: Unsafe nodes (with gray filling) for the 4-cube shown in figure 1.

routing should avoid unsafe nodes as much as possible. Consider the previous example where a message is sent from node (1101) to (0000). By not forwarding the message to node (1100) which is unsafe, a routing function can avoid the routing difficulty described earlier. Instead it may choose to route via a feasible minimal path such as (1101,1001,0001,0000). The notion of an unsafe node must be accompanied by the following property.

Theorem 1 *In a faulty hypercube, if node A is a safe node, for any nonfaulty node B, there always exists a feasible shortest path of length equal to $H(A, B)$ between them.*

Proof: Let $h = H(A, B)$. If h is 1, nodes A and B are neighbors and a link exists. If h equals 2, there are two node-disjoint shortest paths from A to B. By the definition, node A, which is safe, has at most one faulty neighbor. Consequently, at least one feasible shortest path of length 2 exists. Consider the case where $h \geq 3$. Node A has h nearest neighbors that are on some shortest paths of length h to node B. As node A is safe, the number of unsafe or faulty neighbors of A is less than three, which means that there exists at least one nearest neighbor of A that is safe. Take anyone of these safe nodes, say node I. Apparently, $H(I, B)$ is equal to $h - 1$. Iteratively apply this procedure to node I and subsequent ones as long as the associated distance is greater than or equal to 3. Eventually, one will reach a safe node which is 2 hops from node B. Borrowing the argument for the case of $h = 2$, one can conclude that there exists a shortest partial path from this safe node to node B, and hence a full path of length h is feasible. □

Identification of unsafe nodes in a hypercube is a recursive operation. Each node exchanges its current state with all of its fault-free neighbors, and then check if itself should be marked as unsafe with the latest state information. Initially, each node contains a list of its faulty neighbors. At the end of the first iteration, all the unsafe nodes that are due to two or more faulty neighbors are identified. It takes three or more unsafe or faulty neighbors to make a nonfaulty node unsafe afterwards. A simple algorithm for identifying the unsafe nodes is given in [6].

III.2 Properties of the Unsafe Area

Before proceeding, let us study some properties of the structure formed of safe nodes, or alternatively of faulty and unsafe nodes. The set of safe nodes in a hypercube is called the *safe set*. It would be interesting to see if the safe set can be partitioned into disconnected components. A component in this case is a subset of safe nodes that are connected among themselves.

Lemma 1 *Consider an n-cube with a given set of faulty nodes. If the safe set contains two disconnected subsets, denoted as SA and SB, of safe nodes, there exist two nodes $A \in SA$ and $B \in SB$ such that $H(A, B) = 2$.*

Proof: Consider a pair of nodes $A \in SA$ and $B \in SB$ where $H(A,B)$ is the smallest one among all such pairs of nodes. Since SA and SB are disconnected, $H(A,B) > 1$, i.e. A and B cannot be nearest neighbors. Assume that $l = H(A,B) \geq 3$. There are l disjoint shortest, but not necessarily feasible, paths from node A to node B. As node A is a safe node and $l \geq 3$, at least one of these l nearest neighbors is safe., i.e. there exists a safe neighbor, say A', of A such that $H(A',B) = l-1$. Since nodes A and A' are connected, SA should contain node A' as well. However, this contradicts previous assumption that $l = H(A,B)$ is the shortest distance between SA and SB. Hence we conclude that the minimum distance between two nodes, one each in SA and SB, equals 2. □

Theorem 2 *For an n-cube with a given set of faulty nodes, if the safe set is not connected, it must be composed of exactly two subsets of safe nodes, where each of them exactly forms a full $(n-2)$-dimensional subcube, and they are located in the opposite quarters.*

Proof: Consider any two disconnected subsets SA and SB of safe nodes. From lemma 1 there exist two nodes $A \in SA$ and $B \in SB$ such that $H(A,B) = 2$. Assume nodes A and B differ in dimensions i and j. Divide the hypercube into four $(n-2)$-subcubes along dimensions i and j. Let us denote these four subcubes SQ_{00}, SQ_{01}, SQ_{10}, and SQ_{11} where the subscripts indicate binary values of the i-th and j-th bits of the corresponding subcube. Nodes A and B must be in two opposite subcubes; without loss of generality, assume A is in SQ_{00} and B in SQ_{11}. Since SA and SB are disconnected, neither of the two common neighbors of A and B, denoted as A' and B', is safe. In addition, A' and B' must be contained, separately, in the other two $(n-2)$-subcubes. Assume A' is in SQ_{10} and B' is in SQ_{01}. Now consider a neighboring node A'' of A where A'' differ from A in dimension k, $k \neq i$ and $k \neq j$. Then A'' must be in SQ_{00}. As node A is safe and it already has two non-safe neighbors A' and B', A'' must be a safe node. Similarly the neighbor B'' of B along dimension k is in SQ_{11} and is safe. Note that nodes A'' and B'' differ exactly in dimensions i and j. By the same token, neither of the two common neighbors of A'' and B'', along dimensions i and j, is safe. Apparently these two non-safe nodes are contained in SQ_{01} and SQ_{10} respectively. By repeatedly applying this argument, one can conclude that all nodes in SQ_{00} and SQ_{11} are safe, whereas none of the nodes in SQ_{01} and SQ_{10} is safe. It also rules out the possibility that a safe set can be composed of more than two disconnected subsets. □

The structure of the safe set is very much dependent on the fault pattern existing in a hypercube. A connected safe set in a hypercube does not necessarily appear in form of a complete subcube. A faulty hypercube is said to be *fully unsafe* if the corresponding safe set is empty, i.e. every node in the hypercube is either faulty or marked unsafe. Consider an n-cube in which there are n faulty nodes such that these faults happen to occur in the n neighbors around a nonfaulty node. Without loss of generality, let us assume node A, which has binary representation of its address being all zeroes, i.e. $a_i = 0$ for $0 \leq i \leq n-1$, is the aforementioned nonfaulty node. Obviously node A is unsafe. If node B is of distance 2 from A, it has two common neighbors with node A, and these neighbors are both faulty. Hence node B is unsafe. In other words, all the nodes that are of distance 2 from A are unsafe. As to those nodes that are of distance 3 from A every one of them has three unsafe neighbors that are of distance 2 from A, hence all of such nodes are unsafe. Applying the above argument to other nodes that are further from A one can easily conclude that the n-cube is fully unsafe.

Theorem 3 *In an n-dimensional hypercube, if a nonfaulty node A has k, $2 \leq k \leq n$, faulty neighbors, then every node in the associated k-subcube, which includes node A and its k faulty neighbors, is unsafe.*

Lemma 2 *If an k-dimensional hypercube is fully unsafe, the following statement is always true:*
Divide the k-cube into two $(k-1)$-subcubes, say SQ_0 and SQ_1, along any dimension i, $0 \leq i \leq k-1$. Now construct another independent $(k-1)$-dimensional hypercube such that if either of the two neighboring nodes across dimension i, one in each of SQ_0 and SQ_1, is faulty, the corresponding node in the new $(k-1)$-cube is also faulty. Then the new $(k-1)$-cube is itself fully unsafe.

Proof: Denote the newly constructed $(k-1)$-cube Q. Unsafe nodes in a hypercube can be classified into two categories. Those unsafe nodes having two or more faulty neighbors are called type 1. The rest of unsafe nodes belong to type 2. Q can be visualized as overlapping SQ_0 and SQ_1 and marking a node faulty if either of corresponding nodes in SQ_0 and SQ_1 is faulty. Now compare Q with SQ_0. If a node is faulty in SQ_0, its corresponding node in Q is faulty too. Consider an unsafe node of type 1 in SQ_0. If the opposite node in SQ_1 is faulty, the corresponding node in Q is faulty. Otherwise it must have two or more faulty neighbors in SQ_0. Hence the corresponding node in Q is unsafe.

The same argument applies to a comparison between Q and SQ_1. Consequently, the set of faulty or type-1 unsafe nodes in Q contains the union of corresponding sets of nodes in SQ_0 and SQ_1. Note that all unsafe nodes of type 2 are generated iteratively out of this set of nodes. Consider an unsafe node X of type 2 in SQ_0, which is generated immediately as a result of the set of faulty or type-1 unsafe nodes and whose corresponding node in Q is neither faulty nor type-1 unsafe. The opposite node of X in SQ_1 can be neither faulty nor type-1 unsafe. X has at least three faulty or type-1 unsafe neighbors in SQ_0, hence the corresponding node in Q must also be unsafe. This argument leads to the conclusion that the set of faulty or unsafe nodes, regardless of the type, in Q after each iteration will include the union of corresponding sets of nodes in SQ_0 and SQ_1. Consequently, the newly constructed $(k-1)$-cube Q is fully unsafe. □

Lemma 3 *If an n-cube with n faulty nodes is fully unsafe, then any two of the n faulty nodes cannot be nearest neighbors.*

Proof: We prove by induction. For $n = 3$ it is easy to see that the lemma is true. Assume the lemma is true for $n = k$. We want to prove that it is also true for $n = k+1$. By contradiction, let us assume the lemma is untrue for $n = k+1$, i.e. there exists certain case where two of the $k+1$ faults, say nodes A and B, in a fully unsafe $(k+1)$-cube are nearest neighbors. Partition the $(k+1)$-cube, along the dimension in which A and B differ, into two k-subcubes, called SQ_0 and SQ_1, where A is in SQ_0 and B is in SQ_1. Let F_0 and F_1 represent the sets of faulty nodes in SQ_0 and SQ_1 respectively. Construct a new k-cube Q with respect to SQ_0 and SQ_1 along with their fault pattern. According to lemma 2, Q is fully unsafe. Furthermore, the number of faulty nodes in Q is less than or equal to k as A and B would merge into one fault only. Observe that the distance between any two nodes in F_0 cannot be 1, otherwise there would be a corresponding pair of faulty nodes in Q that are nearest neighbors, and this contradicts previous assumption for the case of k. The same is true for nodes in F_1. For any pair of nodes $X \in F_0$ and $Y \in F_1$, it must be true that $H(X,Y) \neq 2$. This is due to the same reason that if $H(X,Y) = 2$, the two faulty nodes in Q, which correspond to X and Y, would be nearest neighbors, and contradiction occurs. Next, the cardinalities of F_0 and F_1 can be characterized as follows:

$$\min(|F_0|, |F_1|) \leq \left\lfloor \frac{k-1}{2} \right\rfloor + 1 = \left\lfloor \frac{k+1}{2} \right\rfloor = \left\lceil \frac{k}{2} \right\rceil$$

Now assume $|F_0| = \min(|F_0|, |F_1|)$. Construct another k-cube Q' in which the fault pattern is exactly identical to that in SQ_0. Note that the number of faults in Q' is no more than $\lceil \frac{k}{2} \rceil$. Hence, if, instead, the criterion used in [13] for labeling a nonfaulty node unsafe is used in Q', Q' is not fully unsafe as per our proof in [6]. Because any pair of nodes $X \in F_0$ and $Y \in F_1$ has distance other than 2, there is no type-1 unsafe node in SQ_0, which is due to two faulty neighbors, one in F_0 and another in F_1. As a result, the unsafe nodes of type 1 in SQ_0 must be identical to those in Q'. Consider type-2 unsafe nodes in SQ_0. According to our criteria, it takes one more unsafe or faulty neighbor to mark a node type-2 unsafe in SQ_0 than it does in Q'. Therefore, if a node is labeled type-2 unsafe in SQ_0, the corresponding node in Q' must also be marked unsafe after each iteration of labeling. In other words, if a node is safe in Q' the corresponding node must be safe in SQ_0. Since Q' is not fully unsafe, the nodes in SQ_0 cannot possibly be fully unsafe. However, this contradicts the assumption that the $(k+1)$-cube is fully unsafe. Hence we prove the lemma. □

Theorem 4 *If the number of faulty nodes in an n-cube is no greater than $n-1$, then the n-cube cannot become fully unsafe.*

Proof: We prove by induction. For $n = 3$ it is easy to see that the theorem is true. Assume the theorem is true for $n = k$. We are to prove it is also true for $n = k+1$. By contradiction, let us assume that the theorem is untrue for $n = k+1$, i.e. some fault pattern with k faults can make the $(k+1)$-cube fully unsafe. Partition the $(k+1)$-cube along an arbitrary dimension into two k-subcubes, SQ_0 and SQ_1. The notations F_0 and F_1 are defined as in the previous lemma. Note that neither F_0 nor F_1 can be empty. For any pair of nodes $X \in F_0$ and $Y \in F_1$, X and Y cannot be nearest neighbors, i.e. $H(X,Y) \neq 1$, otherwise one can construct a fully unsafe k-cube with $k-1$ faulty nodes, as X and Y would merge into one single fault, and contradicts the case of $n = k$. In addition, if $H(X,Y) = 2$, the two corresponding faulty nodes in the aforementioned fully unsafe k-cube would be nearest neighbors, and this violates lemma 3. In fact, the foregoing arguments can be applied to any pair of the k faulty nodes. Hence, we have $H(A,B) \geq 3$ for any pair of faulty nodes A and B. Consequently, there is no type-1 unsafe node generated, and hence the $(k+1)$-cube cannot be fully unsafe. This contradicts previous assumption, and the theorem is proved. □

Theorem 4 provides a lower bound on the number of faulty nodes for which a hypercube can become fully unsafe as per our criteria. Although n faults can make an n-cube fully unsafe, it only occurs in very limited cases even when the size of the hypercube is small. In fact, the chance that a hypercube would become fully unsafe is rare even for the case of having more than n faults.

III.3 Fault-Tolerant Routing Algorithm

We now use the state information of the nodes in a hypercube to assist routing of messages. The fundamental idea is to avoid forwarding messages to an unsafe intermediate node whenever possible to prevent potential pitfalls. The routing strategy is based on knowledge of the states of nearest neighbors only. The assurance of deadlock freeness is not considered until later in section IV.

Before proceeding, we further classify the unsafe nodes to facilitate the design of fault-tolerant routing algorithms. An unsafe node is said to be *strongly unsafe* if none of its nearest neighbors is safe. Apparently, it is more difficult to find a "good" next node for a message to traverse at a strongly unsafe node. We may refer to an unsafe but not strongly unsafe node as an *ordinarily unsafe* node hereafter. Unless otherwise specified, an unsafe node would refer to either a strongly unsafe or an ordinarily unsafe node. Consequently, a node can be in one of four states: *safe*, *ordinarily unsafe*, *strongly unsafe*, and *faulty*.

Let *curr* denote the address of current node and *dest* denote the address of destination node. To facilitate the description of algorithm we introduces the following notations. Each node contains three lists, namely *FAULT*, *UNSAFE*, and *S_UNSAFE*, which, respectively, maintain its faulty, ordinarily unsafe, and strongly unsafe nearest neighbors. Let $d_i(curr, dest)$ denote the dimension of the i-th smallest bit in which *curr* and *dest* differ. Also, $s_i(curr, dest)$ denote the dimension of the i-th smallest bit in which *curr* and *dest* are identical. A routing algorithm *ROUTE* is presented in figure 3. From theorem 1, there exists at least a shortest feasible path of length equal to the Hamming distance between a safe node and any nonfaulty node. Consider a safe source node *sour*. Initially we have $curr = sour$. If $H(sour, dest) \geq 3$, there exists at least one safe neighbor, which is on a shortest path, to which ROUTE forwards the message. Eventually the message will reach a safe node *curr* for which $H(curr, dest) = 2$. As *curr* is safe, the two nearest neighbors common with *dest* cannot be both faulty. ROUTE selects one such node to forward the

```
Algorithm ROUTE
begin
  l := H(curr,dest);
  if (l = 0)
    Deliver the message to the local node;
  for i = 1 to l
      if (neighbor of curr connected by link of
          dimension d_i(curr, dest) is not in any of
          FAULT, S_UNSAFE, or UNSAFE lists)
        then begin
             Route message via link d_i(curr, dest);
             exit;
        end
  for i = 1 to l
      if (neighbor of curr connected by link of
          dimension d_i(curr, dest) is in neither
          FAULT nor S_UNSAFE lists)
        then begin
             Route message via link d_i(curr, dest);
             exit;
        end
  for i = 1 to l
      if (neighbor of curr connected by link of
          dimension d_i(curr, dest) is not in
          FAULT list)
        then begin
           if (curr is strongly unsafe) or (l ≤ 2)
             then begin
                  Route message via link
                   d_i(curr, dest);
                  exit;
             end
        end
  for i = 1 to n − l
      if (neighbor of curr connected by link of
          dimension s_i(curr, dest) is not in any of
          FAULT, S_UNSAFE, or UNSAFE lists)
        then begin
             Route message via link s_i(curr, dest);
             exit;
        end
  for i = 1 to n − l
      if (neighbor of curr connected by link of
          dimension s_i(curr, dest) is in neither
          FAULT nor S_UNSAFE lists)
        then begin
             Route message via link s_i(curr, dest);
             exit;
        end
```

Figure 3: Algorithm ROUTE for message routing in a hypercube with unsafe labels.

message; the message is then passed to *dest* in the last hop. Obviously this path is of shortest length equaling the Hamming distance between *sour* and *dest*, and it exists as long as *dest* is nonfaulty. Now assume *sour* is nonfaulty and *dest* is a safe node. Since *dest* is safe, observe that $H(sour, dest) \geq 2$ neighbors of *sour* along dimensions in which *sour* and *dest* differ cannot be all either faulty or strongly unsafe. This can be argued as follows. From previous discussion, we know that there exists a safe node *curr* on a minimum path from *dest* to *sour* for which $H(sour, curr) = 2$. This implies that at least one of the common neighbors of *curr* and *sour*, which are also on some minimum paths from *sour* to *dest*, must be neither faulty nor strongly unsafe as strongly unsafe node cannot have a safe neighbor. As a result, ROUTE, at *sour*, can find a nonfaulty next node which is not strongly unsafe to forward a message. By repeating this argument, a shortest path of length $H(curr, dest)$ is eventually found as long as *dest* is a safe node. Hence we conclude that communication between a safe node and any nonfaulty node can be achieved via a feasible minimal path of length equaling the Hamming distance between source and destination nodes.

Consider the case in which *sour* is an ordinarily unsafe node and *dest* is non-safe. ROUTE forwards the message toward *dest* via a nearest neighbor, which is safe or ordinarily unsafe if there are no safe ones, on a shortest path. ROUTE eventually leads to a feasible minimal path if it continues to find such neighbors. If neighbors on all shortest paths to *dest* are either faulty or strongly unsafe at some intermediate node *curr*, which apparently is not strongly unsafe itself, ROUTE deroutes to a safe neighbor with smallest dimension if $H(curr, dest) \geq 3$. As described in the previous case, the message is routed via a shortest path hereafter. This increases the length by 2. The derouting is taken to avoid possible routing difficulty, which may subsequently require backtracking and longer path after entering a strongly unsafe node. In case of $H(curr, dest) = 2$, if there exists a nonfaulty neighbor, whether strongly unsafe or not, on a shortest path, ROUTE may use such node to forward a message. If the two associated nearest neighbors are both faulty, a deroute similar to the one described earlier can be applied. Hence, for a message from an ordinarily unsafe source node to any non-safe destination, ROUTE establishes a path of length no greater than two plus the Hamming distance between them.

Finally, consider the case where the source node is strongly unsafe and the destination is non-safe. We need the following theorem for further discussion.

Theorem 5 *If an n-cube is not fully unsafe, then every strongly unsafe node has at least one ordinarily unsafe neighbor.*

Proof: Assume A is a strongly unsafe node in such hypercube. Suppose k, $k \leq n-1$, neighbors of A are faulty, and the other $n-k$, $n-k \geq 1$, ones are non-safe. Assume all of the $n-k$ non-safe neighbors are strongly unsafe. Let N_f and N_s represent the sets of k faulty neighbors and $n-k$ strongly unsafe neighbors of A, respectively. Now examine any nonfaulty node B which is of distance 2 from A. If the two common neighbors between A and B are both in N_f, B must be unsafe. If any one of these neighbors is in N_s, B is unsafe. Hence we show that all nonfaulty nodes of distance 2 from A are either faulty or unsafe. This is the same situation as the case in theorem 3 for $k = n$ after the first iteration, which then leads to a fully unsafe hypercube. This contradicts our assumption in the statement. □

ROUTE forwards a message via a shortest path whenever there is an ordinarily unsafe node, or strongly unsafe one in case no ordinarily unsafe ones exist, on such path. If it truns out, in the worst case, that all neighbors on shortest paths are faulty, then choose the node across the smallest dimension that is connected to an ordinarily unsafe neighbor. As shown in theorem 5, such ordinarily unsafe node exists as long as the hypercube is not fully unsafe. Referring back to previous discussion of the case where the source is an ordinarily unsafe node, one can easily obtain that the worst-case length of a path is four plus the Hamming distance between a source and a destination. The following theorem draws the summary.

Theorem 6 *If an n-cube is not fully unsafe, then ROUTE routes a message via a path of length no more than the Hamming distance between the source and destination plus four.*

IV Deadlock-Freeness

Deadlock freeness is an essential and desirable property in an interconnection network. Since deadlock freeness is not specifically taken into consideration in previous routing algorithm, additional mechanisms are required to maintain such property. To facilitate following discussion, we assume that the underlying switching technique is wormhole routing with virtual channels [7].

A well-known method to enforce deadlock freeness in a resource sharing environment is to assign a set of

priorities to shared resources and to restrict the usage of the resources in certain order. Note that the length of a possible longest path determines the number of virtual channels a hypercube requires in order to achieve a deadlock-free implementation of ROUTE algorithm. The longest distance traversed by a message, using ROUTE algorithm, is bounded by $n+4$ in an n-cube which is not fully unsafe. However, the worst-case of $n+4$ cannot occur as described in [6]. Each physical channel can be divided into $n+3$ virtual channels numbered from 0 to $n+2$. The $n+3$ classes of virtual channels are used in an increasing order in an n-cube. A class-0 virtual channel is allocated for the first hop in routing a message. The class of virtual channel used for subsequent forwarding is incremented by one for each hop. This implementation is obviously deadlock-free for ROUTE algorithm.

Constant-Number Resources

Previous deadlock-free implementation has the disadvantage that the number of required virtual channels is varying and is dependent on the size of hypercube. This makes a system difficult to scale up. In the following, we present another routing algorithm in which a physical channel consists only of a constant number of virtual channels while the property of deadlock freeness is maintained.

Consider that each physical channel is divided into five virtual channels. Virtual channels in a hypercube are classified in five virtual networks, denoted as VN_i, $0 \leq i \leq 4$. Each virtual network contains a virtual channel corresponding to each physical channel. A virtual channel of up-transition ($0 \rightarrow 1$) [5] is called an ***up virtual channel***. Similarly we can define ***down virtual channels***. Each VN_i, $0 \leq i \leq 4$, is further divided into two virtual subnetworks, denoted as VN_{i0} and VN_{i1}, which consist of up virtual channels and down virtual channels within VN_i respectively. In the following, we route a message using up virtual subnetwork first and then switch to a down virtual subnetwork when no up virtual channels are available in a virtual network. Within any subnetwork, messages can be routed in an arbitrary order of dimensions.

In figure 4, we present another fault-tolerant routing algorithm called *ROUTE_C*. Let $d^u(curr, dest)$ and $d^d(curr, dest)$ denote the sets of up-transition and down-transition dimensions in which *curr* and *dest* differ. The notations $s^u(curr, dest)$ and $s^d(curr, dest)$ denote the sets of up-transition and down-transition dimensions in which *curr* and *dest* are identical. Also, (vn, i) represents the virtual channel in virtual network VN_{vn} along dimension i. Consider that a mes-

```
Algorithm ROUTE_C
begin
 l := H(curr,dest);
 Set vn← the incoming virtual network number;
 if (currently using up-transition channel)
    call FORWARD(d^u(curr, dest), d^d(curr, dest));
 else
    call FORWARD(d^d(curr, dest), d^u(curr, dest));
 if (the message has not been forwarded)
   if (currently using up-transition channel)
     call FORWARD(s^u(curr, dest), s^d(curr, dest));
    else
     call FORWARD(s^d(curr, dest), s^u(curr, dest));
end

Function FORWARD(dim_set1, dim_set2)
begin
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set1 which is not in any of FAULT,
      S_UNSAFE, and UNSAFE lists)
      Route the message via (vn, i);
      exit;
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set2 which is not in any of FAULT,
      S_UNSAFE, and UNSAFE lists)
      if (currently using down-transition channel)
          vn← vn + 1;
      Route the message via (vn, i);
      exit;
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set1 which is in neither of
      FAULT and S_UNSAFE lists)
      Route the message via (vn, i);
      exit;
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set2 which is in neither of
      FAULT and S_UNSAFE lists)
      if (currently using down-transition channel)
          if (vn = 0 or l ≤ 2)
              vn← vn + 1;
          else  exit;
      Route the message via (vn, i);
      exit;
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set1 which is not in FAULT list) and
      (curr is strongly unsafe or l ≤ 2)
      Route the message via (vn, i);
      exit;
  if (∃ a neighbor of curr along dimension
      i ∈ dim_set2 which is not in FAULT list) and
      (curr is strongly unsafe or l ≤ 2)
      if (currently using down-transition channel)
          if (l ≤ 2)
              vn← vn + 1;
          else  exit;
      Route the message via (vn, i);
      exit;
end
```

Figure 4: Algorithm ROUTE_C which requires a constant of five virtual networks.

sage is sent from a safe source node. Among the dimensions on shortest paths, there may be some up transitions. The algorithm continues selecting a safe node along such dimension, if it exists, to forward the message using VN_{00} subnetwork. It then switches to route on VN_{01} via safe nodes along down-transition dimensions on shortest paths. There are at the most four hops, two up-transition and two down-transition ones, left before it has to switch to VN_{10}. Repeating a similar argument, one can see that virtual network VN_1 is available for at least two more hops. Finally VN_2 is enough for forwarding the message to its destination. Hence a total of three virtual networks suffice to route a message originating from a safe source via a shortest path of length equal to the Hamming distance between the source and the destination nodes.

Now consider the situation where the source node of a message is strongly unsafe. Virtual network VN_0 is used first. If there are ordinarily unsafe nodes along some dimensions on shortest paths, the algorithm selects one of them as next intermediate node. Otherwise it forwards the message to an strongly unsafe node along a shortest path, with up-transition traversal having higher priority over down-transition one. The above procedure is repeated until no virtual channel in VN_{01} is eligible for routing (VN_{00} is obviously not available by now). When either all shortest paths are blocked due to node failures or VN_0 cannot be used for further routing, the message is derouted to an ordinarily unsafe node, which is guaranteed to exist according to theorem 5. In other words, we are restricted to consume at most one virtual network before the message reaches an ordinarily unsafe intermediate node. Once in an ordinarily unsafe node, a message is routed to a safe node on a shortest path if it exists regardless of the direction of transition. Otherwise it traverses through ordinarily unsafe nodes via a shortest path in a similar fashion as described earlier for the case where the source node is strongly unsafe. Basically, the algorithm may consume, at the most, one more virtual network before it leads to a safe node. As in previous case, a deroute may be necessary to take the message to a safe node. At most three virtual networks are required to reach the destination afterwards. Therefore, a total of five virtual networks suffice for ROUTE_C. Note that the dependence relationship of virtual channels within a virtual subnetwork is loop-free as they are all of the same direction (either all up-transition or all down-transition). Since virtual channels can only be reserved in an increasing order of network numbers and up-transition channels have higher priority over down-transition ones within a virtual network, dependence relationship of virtual channels between different virtual subnetworks is acyclic as well. From above arguments, we come to the following conclusion.

Theorem 7 *If an hypercube of any size is not fully unsafe, five virtual networks suffice to guarantee deadlock freeness for ROUTE_C.*

The requirement of constant virtual networks in ROUTE_C is achieved at the expense that some messages, with small probability, may be routed via a path longer than they should have been with ROUTE, when source nodes of them are not safe nodes. In the case of a safe source node, a shortest path of length equal to the Hamming distance between source and destination is guaranteed.

V Conclusions

In this paper, we proposed a fault-tolerant routing strategy which is based on the idea of unsafe nodes for a faulty hypercube. A new set of criteria is presented to identify unsafe nodes that may cause routing difficulty. Each node in a hypercube may then be in one of four states. Algorithm ROUTE guarantees to route a message via a feasible shortest path when either the source or the destination of the message is a safe node. In the case where the source of a message is ordinarily unsafe, the message is guaranteed to reach its destination via a path whose length is no longer than two plus the Hamming distance between the source and destination. In the worst case, ROUTE can always route a message via a feasible path of length no more than four plus the associated Hamming distance when the source node is strongly unsafe. The last two results are made under the assumption that the hypercube under consideration is not fully unsafe. A sufficient condition for this assumption is that the number of faulty nodes is no more than $n-1$ for an n-cube. In fact, it takes more faults to make a hypercube fully unsafe in most of the cases. The issue of deadlock freeness is also considered in this paper. Most importantly, we proposed another routing algorithm ROUTE_C which requires a constant of five virtual networks for a hypercube of any size. This scheme makes a hypercube system scalable.

Although broadcasting is not specifically addressed in this paper, our strategy may be extended to handle such communication. More general communication such as multicast in a faulty hypercube will be investigated in the future.

References

[1] A. Borodin and J. E. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.

[2] Ming-Syan Chen and Kang G. Shin. Message routing in an injured hypercube. In *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, volume I, pages 312–317, January 1988.

[3] Ming-Syan Chen and Kang G. Shin. Adaptive fault-tolerant routing in hypercube multicomputers. *IEEE Trans. Computers*, 39(12):1406–1416, December 1990.

[4] Ming-Syan Chen and Kang G. Shin. Depth-first search approach for fault-tolerant routing in hypercube multicomputers. *IEEE Trans. Parallel and Distributed Systems*, 1(2):152–159, April 1990.

[5] G. M. Chiu, S. Chalassani, and C. S. Raghavendra. Flexible routing criteria for circuit-switched hypercubes. *To appear in the Journal of Parallel and Distributed Computing.*

[6] G.-M. Chiu and S. P. Wu. A study on fault-tolerant routing in hypercube systems. Technical Report TR-1993-EE-001, NTIT, November 1993.

[7] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers*, C-36(5):547–553, May 1987.

[8] A. L. DeCegama. *The Technology of Parallel Processing – Parallel Processing Architectures and VLSI Hardware Volume 1.* Prentice Hall, Inc., 1989.

[9] P. J. Denning. Parallel computing and its evolution. *Commun. ACM*, (29):1163–1167, December 1986.

[10] J. Duato. On the design of deadlock-free adaptive routing algorithms for hypercubes: Theoretical aspects. In *Proc. 2nd European Distributed Memory Conference*, pages 234–245, 1991.

[11] J. M. Gordon and Q. F. Stout. Hypercube message routing in the presence of faults. In *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, volume I, pages 318–327, January 1988.

[12] S. Konstantinidou. Adaptive, minimal routing in hypercubes. In *Proc. 6th MIT Conf. on Advanced Research in VLSI*, pages 139–153. MIT Press, 1990.

[13] T. C. Lee and J. P. Hayes. A fault-tolerant communication scheme for hypercube computers. *IEEE Trans. Computers*, 41(10):1242–1256, October 1992.

[14] C. L. Seitz. The cosmic cube. *Commun. ACM*, (28):22–33, July 1985.

[15] Sing-Ban Tien and C. S. Raghavendra. Algorithms and bounds for shortest paths and diameter for faulty hypercubes. *IEEE Trans. Parallel and Distributed Systems*, 4(6):713–718, June 1993.