Routing in Modular Fault Tolerant Multiprocessor Systems

M. Sultan Alam and Rami G. Melhem

Department of Computer Science
The University of Pittsburgh

Abstract

In this paper, we consider a class of modular multiprocessor architectures in which spares are added to each module to cover for faulty nodes within that module, thus forming a Fault Tolerant Basic Block (FTBB). In contrast to reconfiguration techniques that preserve the physical adjacency between active nodes in the system, our goal is to preserve the logical adjacency between active nodes by means of a routing algorithm which delivers messages successfully to their destinations. We introduce two phase routing strategies that route messages first to their destination FTBB, and then to the destination nodes within the destination FTBB. This strategy may be applied to a variety of architectures including binary hypercubes and 3-D tori. In the presence of f faults in these systems, we show that the worst case length of the message route is $\max\{\sigma+f\ ,(K+1)\sigma\}+M$ *where* σ *is the shortest path in the absence of faults, and M and K are the numbers of primary nodes and spare nodes in an FTBB, respectively. The average routing overhead is much lower than the worst case overhead.*

1. INTRODUCTION

As the number of processors in a multiprocessor system increases, the complexity of the system increases, leading to a possible high rate of both transient and permanent failures. The reliability of such systems can be improved by incorporating some type of fault tolerance. For instance, fault tolerance can be achieved by distributing the load of a faulty processor to other non-faulty processors [4,11]. and then using fault tolerant routing to by-pass faults and deliver messages to their destinations. Several fault tolerant routing schemes have been proposed in the literature for general and specific architectures [6-8,10]. In these schemes, different adaptive routing algorithms have been used to by-pass faulty nodes. Olson *et al.* proposed a routing algorithm for HARTS (Hexagonal Architecture for Real-Time Systems) which ensures the delivery of a message as long as there exists a path between the message source and its destination [15]. Peleg *et al.* [16] proposed fault tolerant routing schemes for several families of graphs, including all graphs of maximal degree less than $cn^{1/3}$ for some $c>0$ (n is the number of nodes in the graph). The distribution of the load of the faulty node in such schemes is a non-trivial problem and the performance degradation can be as high as 50 percent [4].

An alternative approach to fault tolerance is to use spares. In this approach, the system performance degradation is minimized by allowing spare nodes to replace faulty ones. For applications where the topology of the underlying system is important, the adjacency relationship among the *active* nodes should be preserved after reconfiguration, where an *active* node is defined to be a non-faulty primary node or a spare node that had replaced a faulty node. That is, if a spare S replaces a faulty node P, then, after reconfiguration, S should become a neighbor of all the neighbors of P. Usually, hardware switches are used to preserve the adjacency relationship among the *active* nodes [1,5,9,13,14,18-20]. Note that in this approach the routing algorithm does not need to be modified. In some systems, however, preserving the adjacency relationship among the *active* nodes may not be crucial because the applications may not assume any specific topology. In such systems, an alternative to preserving the physical adjacency is to modify the routing algorithm so that messages can by-pass faulty nodes and be delivered to the destination node.

In this paper, we assume that when a spare node replaces a primary node, it inherits its address. Thus any message addressed to the failed node should be delivered to its replacement spare. The sender of a message addresses the message to a logical destination, and does not need to know whether the destination is a primary node or a spare that had replaced a failed primary node. In other words, the burden of maintaining the logical interconnection among the *active* nodes is assigned to the routing algorithm. There is no need to preserve the physical adjacency between active nodes by setting up reconfiguration switches. Hence, no hardware switches (usually assumed fail-safe) are needed and recovery from faults is faster.

This work was partially supported by NSF Grant MIP-8911303.

We introduce a two-phase routing approach which is general in the sense that it can be applied to different architectures and to different spare allocation strategies. Only local fault knowledge is assumed. That is, each node only has to know whether its neighbors are faulty or not. The distributed routing algorithm assumes total node failures in the sense that a failed node cannot be used to route messages and thus all links connected to a failed node are unusable. Conditions are given for the two phase routing algorithm to work correctly, and it is shown that these conditions can be satisfied in many well know architectures by properly connecting the spares to other nodes in the system.

The rest of this paper is organized as follows: Section 2 describes the two-phase routing approach and establishes its worst case performance. The approach is then applied in Section 3 and 4 to binary hypercubes and 3_dimensional toroidal systems. Although, analytical results show that the worst case routing overhead is more than 100 percent, simulation results show that, on the average, the routing overhead is low.

2. TWO PHASE ROUTING

Most common multiprocessor architectures are constructed from identical modules to achieve scalability as well as ease of maintenance and repair. Redundancy may be added to these systems by either adding spare modules or adding spare nodes in each module (or both [17]). We assume that the latter approach is taken. Specifically, we assume that each module consists of M primary nodes and that K spares are added to each module to replace faulty primary nodes according to some specific coverage policy that governs which spare may replace which primary node. In general, we assume that each spare may replace any of $flex$ primary nodes ($flex$ = flexibility of coverage) and each primary node may be covered by any of deg spare nodes (deg = degree of coverage). The M primary nodes and the K spare nodes, thus, form a fault tolerant basic block (FTBB) whose size is referred to as (M,K). Two FTBB's are called *neighbors* if there is a link between a node in the first and a node in the second. An FTBB is called *live* if it contains at most K faulty nodes (primary or spares) and the system is called *live* if all its FTBB's are live. Note that after a fault, a system remains *live* if there is an available spare to cover for the fault. Otherwise, system failure is declared.

The address of a node, P, may be divided into two parts, one identifying the FTBB that contains P (denoted by F_p) and the other identifying P within F_p. If no nodes in the system are faulty, then only the primary nodes are active and thus have addresses. When a primary node fails, a spare replaces that node by taking over its computational tasks and inheriting its address. From that point on, any message addressed to the failed node should be delivered to the spare that replaced it. This is the responsibility of the routing algorithm.

Any efficient fault tolerant routing algorithm should use the shortest path between two nodes when no faulty nodes are encountered. Moreover, the routing performance degradation should be graceful with the number of faults and the routing algorithm should guarantee that a message does not cycle indefinitely in the system (a live-lock situation). The routing strategy suggested in this section leads to distributed algorithms that use only local fault knowledge. This avoids the need for a global controller and for storing global fault information in each node. In this strategy, a message is routed to its destination node, d, in two phases. First, the message is routed to the FTBB that contains d, and then to d or the spare node that replaces d. Once the message reaches the destination FTBB it does not leave that FTBB. The two phase routing strategy can be described as follows (Assume that p is the present routing node):

(1) While $F_p \neq F_d$ send the message to some node in a neighboring FTBB, F_q, which is closer to F_d.

(2) Route the message to node d without leaving F_d.

A key property of the two phase routing is that it does not require any backtracking between FTBB's. Specifically, a message is always moved to an FTBB that is closer to its destination. This leads to efficient and simple implementations. The restricted flexibility resulting from prohibiting backtracking will be compensated for by enhancing the interconnections in a way that guarantees the success of the routing in any live system.

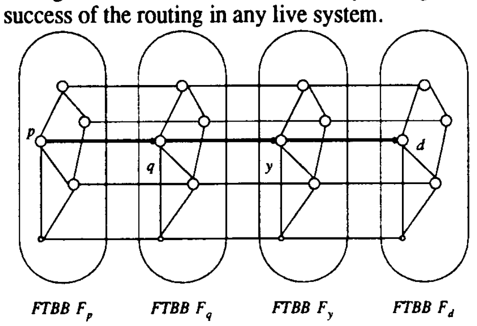


Figure 1. Two phase routing with no faults.

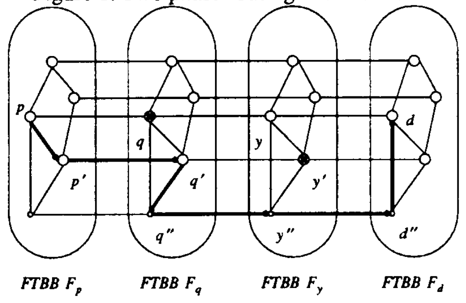


Figure 2. Two phase routing with a fault at node q.

The details of each routing stage depend on the architecture. In fact, given a particular architecture, the choice of FTBB F_q in stage 1 is identical to the choice that a non-fault tolerant routing algorithm would make for that architecture. With faulty nodes, however, it may not be possible to send a message from F_p to F_q using the same path that is used in the absence of faults. The message may have to take a longer path to get to F_q. Figures 1 and 2 are used to explain this. The large circles and the small circles in these figures denote primary nodes and spare nodes, respectively, and a crossed out node denotes a faulty node. In Figure 1, a message is routed from node p to node d through the nodes q and y. If node q, which is on the path of this message, fails, then in phase 1 of the algorithm, the message is routed to node q' in FTBB F_q (via p') and then to node y'' in FTBB F_y (via q'') and finally to node d'' in FTBB F_d (see Figure 2). The second phase of the algorithm starts after the message reaches node d''.

Given a distribution of faults in the system, a link which connects two non-faulty nodes is called a *healthy* link. A healthy path between two non-faulty nodes is either a *healthy* link between the two nodes or a sequence of *healthy* links that connect the two nodes through non-faulty nodes. The two-phase routing strategy described above may deliver messages correctly only if certain conditions are met. For instance, phase 1 may fail if there are two adjacent FTBB's that are not connected by at least one *healthy* link. Also, phase 2 may fail if there exists two nodes in an FTBB, F, that are not connected by a healthy path within F. In this context, a path within F is one that does not leave F. Therefore, the following two conditions are necessary for the two phase routing strategy to be successful in a live system:

1) For any two neighboring FTBB's, F and F', any set of K or fewer faults in F and any set of K or fewer faults in F', there exists a *healthy* link between F and F'.
2) For any set of K or fewer faults in an FTBB, any two non-faulty nodes in that FTBB are connected by a healthy path.

These two conditions ensure that there exists a healthy path between any two nodes in a live system and that a two phase routing strategy can route a message along that path. Thus, these conditions are sufficient for the two phase strategy to be successful provided that suitable algorithms are designed to implement each of the two phases. Before discussing specific algorithms, we investigate further the first condition. For that condition to be satisfied, there should be $K+q$ nodes in F, say $n_0, \cdots, n_{K+q-1}$, for some $q > 0$, such that each n_i is a neighbor to some nodes in F'. Similarly, there should be $K+q'$ nodes in F', say $m_0, \cdots, m_{K+q'-1}$, such that each m_i is a neighbor to some node in F. We assume that $q = q'$ and consider the following cases for q.

The case q = 1: In this case, each n_i, $0 \le i \le K$ should be connected to all m_j, $0 \le j \le K$ to guarantee a healthy link from F to F' if K nodes in F and K nodes in F' are faulty.

The case q = K+1: For any K faults in F and K faults in F', the existence of a healthy link from F to F' is guaranteed if each n_i, $0 \le i \le 2K$ is connected to a different m_j. That is, up to the relabeling of nodes, each n_i is connected to m_i.

The above two cases are special cases of the following Lemma:

Lemma 1: For $1 \le q \le K+1$ and any K faults in F and K faults in F', the existence of a healthy link from F to F' is guaranteed if each n_i, $0 \le i \le K+q-1$ is connected to $K+2-q$ nodes in F' such that, up to the relabeling of nodes, each n_i is connected to $m_{(i+u) \bmod (K+q)}$, $u=0,\ldots,K+1-q$.

Proof: For any K faults in F, there are q non-faulty nodes in F connected to F'. Let these nodes be n_{i_l}, $l=1,\ldots,q$ and assume that $i_l < i_{l+1}$. Node n_{i_1} is connected to $K+2-q$ nodes in F', and each n_{i_l} is connected to at least one node in F' that is not connected to any previous node n_{i_z} for $z<l$. Hence, there are at least $K+1$ nodes in F' that are connected to the q non-faulty nodes in F. Thus, for any K faults in F' there will be a healthy link between a node in F and a node in F'. □

Corollary: For $q > K+1$, if each n_i, $0 \le i \le K+q-1$, is connected to m_i, then there is a healthy link from F to F' for any K faults in F and K faults in F'.

For any given modular architecture in which K spare nodes are added to each module, the above lemma may be used to establish the connections between the spare nodes and the other nodes in the system in a way that allows for the design of two phase routing algorithms. In Figure 3(a), an example is given for a mesh architecture with $K=1$ and $q=K+1$. Dashed lines are used to enclose FTBB's and the mesh interconnections are highlighted in bold lines. The mesh in Figure 3(b) has $K=q=1$.

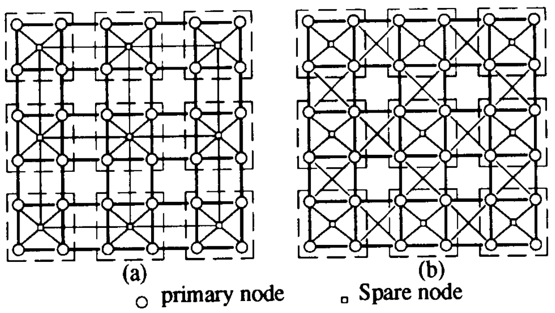


Figure 3 - Meshes amenable to two phase routing.

Lemma 1 may be applied to other architectures such as trees, Cube Connected Cycles, hypercubes and tori. We will focus our attention in this paper on the later two architectures since they may be constructed such that modules exhibit a strong neighbor property in the sense of the following definition:

Definition: Two neighboring FTBB's are *strong neighbors* if each node in one FTBB is a neighbor of exactly one node in the other. □

Consider two live FTBB's, F and F' of size (M,K). From Lemma 1 and its corollary, if F and F' are strong neighbors, then there will be a healthy link between them if $M > K$. A routing algorithm can send a message from any node p in F to some node in F' if it sends the message along a path, π, in F, which starts at p and passes through at least K other non-faulty nodes (a healthy path of length $K+1$). If each node on that path tries to send the message to F', then one node will be successful because there are at most K faults in F'. The following theorem specifies conditions that guarantee the existence of such a path starting at any non-faulty node, p, in F. Moreover, it will provide for a uniform implementation of a two phase routing algorithm at any node in the system.

Theorem 1: For a modular system constructed from FTBB's of size (M,K) such that any two neighboring FTBB's are strong neighbors, a two phase routing algorithm may be designed for that system if:

1) $M > K$, and
2) For any subset, S, of M nodes in an FTBB, there is a cycle, Λ, connecting at least $K+1$ nodes from S such that any node in S is either on Λ or is connected to a node on Λ.

Moreover, in the presence of f faults, the routing algorithm will route a message in at most $\sigma + M + \max\{K\sigma, f\}$ steps, where σ is the minimum number of routing steps in the absence of faults.

Proof: Given two FTBB's F and F', let S be the set of active nodes in F. For any p in S, if p is on Λ, then the path π consists of $K+1$ consecutive nodes from Λ. If p is not on Λ, then it is directly connected to a node, u, on Λ and thus the path π consists of p, u and $K-1$ other consecutive nodes in Λ. Thus, by sending the message along π, and observing that there are at most K faults in F', we can guarantee that one of the $K+1$ nodes on π will be able to send the message to F'. This shows that the first phase of a two phase routing strategy may be implemented.

For the implementation of the second phase, let S be the set of active nodes in the destination FTBB, F. Starting at $p \in S$, a message that is addressed to another node in S may be sent along Λ (if p is not on Λ, then it is connected to a node on Λ). Each node that receives the message checks if the message is for it. If not it checks if it is for a node connected to it. If not, it sends the message to the next node on Λ. The message will reach its destination before visiting any node twice.

To compute the routing overhead, consider two nodes s and d and assume that, in the absence of faults (and spares), σ is the length of the shortest path from s to d. Given that neighbor FTBB's are strong neighbors, all of these σ steps can be across FTBB's. In the presense of faults, the FTBB's will be crossed in the same order but to move between two FTBB's it may take up to $K+1$ steps. Moreover, in the destination FTBB, the message may need up to $L+1$ steps to reach the destination where L is the number of nodes on Λ. Hence, in the worst case, the message may need $(K+1)\sigma+M$ routing steps from s to d. Note that to cross from FTBB F to FTBB F', if the message is at a node p in F, then it is sent to the next node on Λ only if the neighbor of p in F' is faulty. Thus the number of extra steps in the message routes while crossing FTBB's cannot exceed the number of faults in the system. Thus the number of routing steps from s to d is at most $\sigma+f+M$. □

The proof of Theorem 1 may be used to construct routing algorithms that are uniform across the nodes in the system. Define one of the directions on the cycle Λ to be positive and the opposite direction to be negative. If P is on Λ, then let $next^{+}(P)$ and $next^{-}(P)$ be the nodes that follow P on Λ in the positive and negative directions, respectively. We will use $next(P)$ to indicate $next^{+}(P)$. if P is not on Λ, then let $next(P)$ be a node on Λ that is connected to P. With this, the general form of the routing algorithm at node P is:

```
Algorithm Two-phase Route
Phase 1: If the destination, $d$, is in an FTBB different
from the current FTBB, $F_P$, then
    1) compute the next FTBB, $F'$ to which the message is
    to be sent.
    2) If the neighbor, $P'$, of $P$ in $F'$ is active, then send
    the message to $P'$, else send the message to $next(P)$.
Phase 2: If $d$ is in $F_P$, then
    1) if $d = P$ keep the message,
    2) elseif $P$ is directly connected to $d$, then send the
    message to $d$,
    3) elseif the message was received from a node not on
    $\Lambda$, then chose a direction $dir$ and send the message to
    $next^{dir}(P)$,
    4) else send the message to $next^{dir}(P)$, where $dir$ is
    direction from which the message was received.
```

We can identify two methods for routing messages along the nodes on Λ with only local information about active nodes. One method is to compute Λ distributively for every message through a backtracking algorithm similar to the one described in [6]. In such a method, a stack of nodes visited so far should be kept in the message to prevent looping. This implies that the message needs to be updated at each node that it visits resulting in slower message delivery. In some cases it may be possible to carefully design the distributive algorithm in a way that prevents looping without updating the message [2]. An alternative method is to compute Λ after each fault and store it by keeping in each node, p, the values of $next^{+}(p)$ and $next^{-}(p)$. In general, the computation of Λ may require that each node knows about the active nodes in its own FTBB. In some cases, however, as will be shown in the next sections, p may determine the next node on Λ by having only local information about its own neighbors.

The bound given by Theorem 1 on the number of routing steps for two phase routing does not reduce to σ in the absence of faults. This is because in the second phase of routing, a conservative approach is taken to guarantee that messages will not cycle indefinitely. The number of routing steps may be reduced to $\sigma + f$ if the routing in stage 2 of the algorithm does not follow Λ, but rather route the message directly to the destination using some information about the active nodes in the destination FTBB. For small size FTBB's like the ones described in Sections

3 and 4, only local information is enough to design stage 2 efficiently such that the routing algorithm takes σ steps in the absence of faults.

3. APPLICATION TO BINARY HYPERCUBES

Hypercubes are very modular systems. An n-dimensional binary hypercube can be viewed as 2^{n-m} modules each containing 2^m nodes, for any $0 \le m \le n$. Each node is given an m bit address such that the addresses of neighboring nodes differ in exactly one bit. Communication between nodes is done via message passing; If the destination address is $d_n d_{n-1} \cdots d_1$ then the present routing node, $p_n p_{n-1} \cdots p_1$, executes the following routing algorithm (x_i is the exclusive or of d_i and p_i) [13, 14]:

```
IF ( $x_j$=0, for all $1 \le j \le n$ ) then send the message to the
local processor
ELSE route the message along dimension $j$, where $j$ is
the largest integer such that $x_j$=1
```

We will discuss two schemes for adding spares to modules in hypercubes. In the first scheme, denoted BH-I, a spare is added to each module to replace any of the $M = 2^m$ primary nodes, thus forming an FTBB. Although this scheme is simple and results in an efficient routing algorithm, it has very little flexibility for fault coverage. Specifically, more than one failure in an FTBB results in system failure. In order to improve the reliability, a second scheme, BH-II, is introduced. In this scheme, more than one spare is added to each module but the interconnection complexity is kept low by limiting the number of nodes that may be replaced by each spare (limiting *flex*). Both schemes are amenable to efficient fault tolerant routing algorithms that adapt to faults gracefully and reduce to the usual hypercube routing in the absence of faults.

3.1. BH-I: A single fault tolerant architecture

In this scheme, there is one spare node in each FTBB and this spare is connected to the 2^m primary nodes in the FTBB. Moreover, the 2^{n-m} spare nodes are interconnected as a binary hypercube structure of dimension $n-m$, which is called the *spare cube*. For clarity, we use the term *primary cube* to refer to the binary hypercube formed by the 2^n primary nodes. Each node in the *primary cube* has a n bit address and all the primary nodes in an FTBB have $n-m$ identical most significant address bits. These $n-m$ bits are used to identify the FTBB. For example in Figure 4, n=4, m=2 and an FTBB consists of a column of four primary nodes and a spare node.

Clearly, BH-I leads to single fault tolerant systems since the failure of any two nodes in the same FTBB leads to system failure. The 2^m active nodes in an FTBB may be connected by a cycle, Λ, which, in the absence of faults in the FTBB, results from the Gray code embedding of a ring in the 2^m-node subcube [12]. If the FTBB contains a fault, then this fault may be by-passed in Λ using the spare node. Thus the conditions of Theorem 1 are satisfied. The hypercube bit-wise routing algorithm is converted into a two-phase algorithm in order to route messages around faulty nodes. Specifically, a message addressed to some node $d_n \cdots d_1$ is routed to FTBB $d_n \cdots d_{n-m}$ first, and then is routed within that FTBB to the destination node. If the primary node $d_n \cdots d_1$ is not faulty, then it is the destination node. Otherwise, the spare node in FTBB $d_n \cdots d_{n-m}$ is the destination node because it is the only spare that can cover for the fault. The proposed routing algorithm that achieves this goal is completely distributed and only requires that the neighbors of a failing node know about the failure.

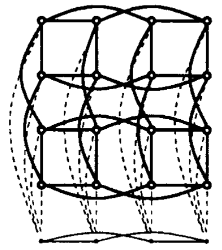


Figure 4. The augmented hypercube, BH-I.

The routing algorithm is described at a node $p=p_n \cdots p_1$ which computes the next dimension j to be crossed and tries to send the message to $C_j(p)$, the neighbor of p across dimension j. If that node is faulty, then the message is sent to *next*(p), the next node on Λ.

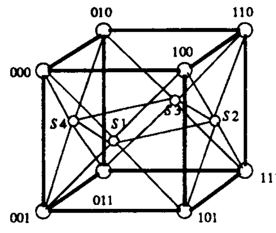
Algorithm ROUTE_BH-I

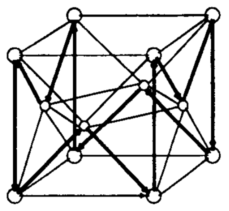
```
1) If $d = p$, keep the message ;
2) Find the largest $j$ such that $d_j$ xor $p_j$ = 1 ;
3) If $j > m$,   /* Phase 1 */
     If $C_j(p)$ is active, then send the message to $C_j(p)$
     else send the message to next$(p)$
4) else          /* Phase 2 */
     If $p$ is a spare node, then send the message to $d$.
     elseif $C_j(p)$ is active, then send the message to $C_j(p)$
     else send the message to the spare node connected to $p$
```

Given that K=1, only two nodes need to be tried in Phase 1 before a message is sent to the next FTBB. Phase 2 of Route_BH_I is different from the general algorithm, Two-phase Route, because it does not send messages along Λ. Rather, it uses the usual hypercube routing within each FTBB and only deviates from that routing if the next node on the route is faulty. In this case the destination is the spare node or a node connected to the spare node. With this, the routing overhead is eliminated when the system does not contain any faulty nodes.

3.2. BH-II: A double fault tolerant architecture

In this scheme, modules are 3-dimensional subcubes and four spare nodes are added to each module to form an FTBB. The spare allocation strategy is such that each spare may replace any of four primary nodes and each primary node is covered by two spares ($deg = 2$ and $flex = 4$). In Figure 5(a), we show an FTBB in this architecture where the four spares are connected as a 2-dimensional cube and communication links are added between each spare and the four primary nodes that it can replace. Specifically, spare S1 can replace any of the nodes 000, 001, 101 or 100. Spare S2 can replace any of 100, 101, 111 or 110. Spare S3 can replace any of 110, 111, 011 or 010, and spare S4 can replace any of 010, 011, 001 or 000. Note that the number of spares is equal to that of the BH-I scheme with $m=1$. However, allowing each spare to be shared among four nodes rather than two nodes improves the reliability of the system [2]. This is primarily due to the capability of tolerating any two faults, while some two faults configurations in BH-I with $m=1$ cannot be tolerated.





(a) An (8,4) FTBB. (b) The cycle Γ

Figure 5.

Dimensions 1, 2 and 3 are chosen to span each module, and thus the 8 primary nodes in each FTBB have $n-3$ identical high order bits. These are used to identify the FTBB. Similar to scheme BH-I, the spare nodes are interconnected as a hypercube (of dimension $n-1$). The architecture of a system composed of two such FTBB's is shown in Figure 6. To show that the FTBB of Figure 5 satisfies the conditions of Theorem 1, we consider the cycle Γ = 000, 001, S1, 101, 100, S2, 110, 111, S3, 011, 010, S4, 000 (see Figure 5(b)). This cycle spans the 12 nodes in the FTBB and may be used as the basis for defining a cycle Λ that spans any 8 active nodes in the FTBB.

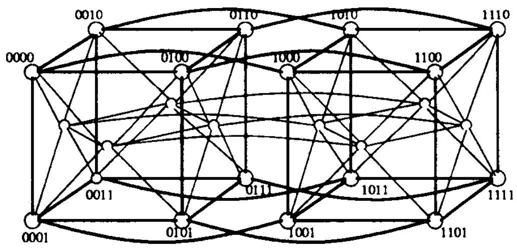
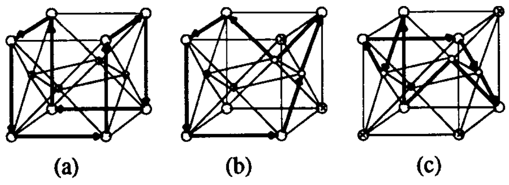


Figure 6. A 4-dimensional augmented hypercube

Let $\Gamma.n(P)$ and $\Gamma.n^{-1}(P)$ be the nodes following and preceding P in Γ and let $\Gamma.n^j(P)$ be the node following $\Gamma.n^{j-1}(P)$ in Γ for $j > 1$. Recall that a non-active node is a faulty node or a spare node that is not used to replace any primary node. Recall also that the spare allocation policy allows a faulty primary node to be replaced only by one of the two spares connected to it. This restriction prevents any four consecutive nodes on Γ to be non-active. In fact, it allows three consecutive nodes, p_1, s, p_2, on Γ to be non-active only if p_1 and p_2 are primary nodes and s is a spare node. With this observation, given any 4 non-active nodes in an FTBB, Λ may be defined as follows:



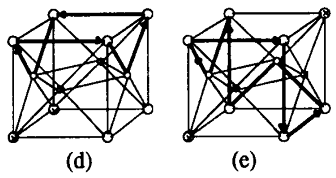


Figure 7 - Λ for different configurations of active nodes.

case 1: If no two non-active nodes are consecutive in Γ, then Λ is the 8 node cycle defined by specifying for each active node, p, the node $next(p)$ as follows:

$$next(p) = \begin{cases} \Gamma.n(p) & \text{if } \Gamma.n(p) \textit{ is active} \\ \Gamma.n^2(p) & \textit{otherwise} \end{cases}$$

An example of this case is given in Figure 7(b) where the nodes S1, 100, 111 and S4 are non-active (designated by x in the figure). If the four spares are not active, then the cycle Λ is the gray code embedding of a ring in a 3-dimensional cube as shown in Figure 7(a). This cycle is denoted by Λ_8.

case 2: If three non-active nodes, p_1, s, p_2, are consecutive on Γ, then p_1 and p_2 are primary nodes and s is a spare. Λ is, thus, the 8 node cycle specified at each active node, p, as follows: $next(p)$ =

$$\begin{array}{ll} \Gamma.n(p) & \text{if } \Gamma.n(p) \textit{ is active} \\ \Gamma.n^2(p) & \text{if } \Gamma.n(p) \textit{ is not active} \text{ and } \Gamma.n^2(p) \textit{ is active} \\ \Gamma.n^4(p) & \textit{otherwise} \end{array}$$

An example of this case is given in Figure 7(c) where the nodes 001, S1, 101 and 110 are not active.

case 3: Only two of the four non-active nodes are consecutive on Γ, while each of the other two is preceded and followed by active nodes. The later two nodes can be bypassed in Γ as in case 1. If the two consecutive nodes are primary nodes, say p_1 and p_2, then they are preceded and followed in Γ by spare nodes. Namely $\Gamma.n^{-1}(p_1)$ and $\Gamma.n(p_2)$. These two spares may be directly connected, thus bypassing p_1 and p_2. Finally, if the two non-active consecutive nodes include a spare, s, then these two nodes should be among three consecutive nodes p_1, s and p_2 on Γ. In this case, it is possible to bypass all of the three nodes by connecting directly $\Gamma.n^{-1}(p_1)$ and $\Gamma.n(p_2)$. For example if 001 and S1 or S1 and 101 are not active, then the three nodes 001, S1 and 101 can be bypassed by making $next(000)$ = 100 in Λ (see Figure 7(d)). Note that this is a case where an active node is excluded from Λ, but is connected to a node on Λ.

case 4: If two of the non-active nodes are consecutive and the other two are also consecutive. Each two consecutive nodes may be dealt with by a bypass as in case 3 (see Figure 7(d)). This is always possible except for one case described next.

case 5: If in a sequence $p_1, s_1, p_2, p_3, s_2, p_4$, of consecutive nodes on Γ, the primary nodes p_1 and p_4 and the spare nodes s_1 and s_2 are not active. In this case it is not possible to bypass all of the six nodes. It is possible, however, to bypass the four non active nodes using links not originally on Γ. Specifically, it is possible to go from $\Gamma.n^{-1}(p_1)$ to $\Gamma.n(p_4)$ via p_3 then p_2. An example is given in Figure 7(e) where the nodes 001, S1, S2 and 110 are not active. Note that because of the spare allocation policy, only two of S1, 101,100 and S2 may be non-active.

The computation of Λ may be entirely distributed. Specifically, each node, p needs only to compute and store $next(p)$. Given the sequence Γ, which is independent of the active node configuration, node p may compute $next(p)$ by only knowing the status of its own neighbors. For cases 1-2 above, p needs to know only which of its neighbors is not active. However, to handle all five cases, p needs also to know if an active neighbor is Γ-isolated, which is defined as follows:

Definition: for any active node q, a node e is called *inaccessible* from q if e is not a neighbor of q or if e is not active. The node q is called Γ-isolated if the nodes $\Gamma.n^j(q)$, j=1,2,3,4 are inaccessible from q. □

With this definition and from the discussion of the five possible cases for the distribution of the four non-active nodes in the FTBB, the computation of $next(p)$ at any active node p is given by:

1. If $\Gamma.n(p)$ is active and is not Γ-isolated, then $next(p) = \Gamma.n(p)$,
2. elseif $\Gamma.n^2(p)$ is active then, $next(p) = \Gamma.n^2(p)$,
3. elseif $\Gamma.n^3(p)$ is active then, $next(p) = \Gamma.n^3(p)$,
4. elseif $\Gamma.n^4(p)$ is active then, $next(p) = \Gamma.n^4(p)$,
5. else $next(p) = \Gamma.n^{-1}(p)$.

The requirement, in step 1, that $\Gamma.n(p)$ is not Γ-isolated is needed to implement **case 3** and ensure that if $\Gamma.n^2(p)$ and $\Gamma.n^3(p)$ are not active, then $\Gamma.n(p)$ is also excluded from Λ. Step 5, then ensures that the function *next* is defined for any active node that is excluded from Λ. This same step also ensures that **case 5** is handled properly.

With the above specification of Λ, two phase routing may be applied with the first phase being identical to that of Route_BH-I. The second phase of Route_BH-1, however, was specific to scheme BH-I. For scheme BH-II, the second phase of the general algorithm Two-Phase Route may be applied. This phase starts at the first node visited by the message in the destination FTBB or at the source node, if the source and destination are in the same FTBB. This node is called the *entry node*. If the entry node is on Λ, then the direction of routing on Λ may be chosen to guarantees that, in the absence of faults, the message will follow the shortest path to the destination, d. Specifically, the second phase of the routing algorithm at a node, P, may be described as follows:

Phase 2: If d is in F_P, then

1) if $d = P$ keep the message,
2) elseif P is directly connected to d, then route to d,
3) elseif P is the *entry node* and P is on Λ, then route to $next^{+}(P)$ or $next^{-}(P)$ depending on which one is closer to d,
4) elseif P is the *entry node*, then route to $next(P)$,
5) else route to $next^{dir}(P)$, where dir is direction from which the message was received.

Note that in the absence of faults, $\Lambda = \Lambda_g$ (see Figure 7(a)). Sending the message to the node on Λ_g closer to d ensures that the message will follow the shortest route to d. For example, assume that a message is to be sent from 001 to 111 (see Figure 5(a)). If 001 sends the message to 101, then 101 will send the message to 111 because it is directly connected to it (step 2 of Phase 2). In other words, in the absence of faults the two phase routing reduces to the usual cube routing.

3.3. Experimental Analysis

The upper bound on the number of routing steps given in Theorem 1 assumes a very pessimistic distribution of faults in the system. For example, in the two-FTBB system of Figure 6, the maximum number of steps, $4(\sigma+1)+8$ is obtained if the source and the destination nodes are not in the same FTBB, and the eight faults in the system are such that no two nodes across dimension j are faulty. The probability that the system reaches the above configuration is very small and thus the average number of routing steps is much less than this worst case bound.

A simulation software tool was designed in order to determine the routing performance. For a given dimension, n, and node reliability $r=e^{-\lambda t}$, where λ is the fault rate, the simulation software generates a set of faults FS such that the system is *alive*, i.e. all the nodes in a FS can be replaced by available spare nodes. The software generates a total of 1000 different FS's. For each FS, 1000 messages with random sources and destinations are generated. These messages are then routed from their sources to their respective destinations using the two-phase routing algorithm, and the total number of steps, ts, is determined. Also, the minimum number of steps, ts', that is required to route these messages in a fault-free system using the usual bitwise cube routing is calculated. The overall routing overhead is determined as the average of $(ts-ts')/ts'$. The routing performance of the algorithms in BH-I and BH-II are shown in Figure 8 for n=7. Obviously, the overheads are very small compared to the theoretical overheads of approximately 100 and 400 percents given by Theorem 1 for BH-I and BH-II, respectively. The routing overhead increases with the number of faults in the system, which is expected.

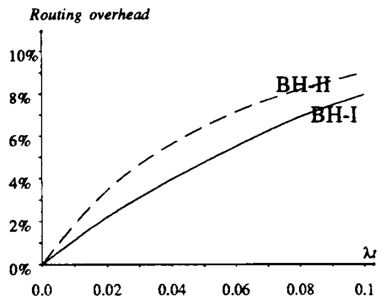


Figure 8. Average routing overhead

4. APPLICATION TO 3-D TOROIDAL SYSTEMS

Consider a $c_1^{max} \times c_2^{max} \times c_3^{max}$ 3-dimensional torus. The address of a node in the torus is denoted by $[c_1, c_2, c_3]$, where c_j, $1 \le c_j \le c_j^{max}$, is the coordinate of a node along dimension j. The positive direction along dimension j from a node with j^{th} coordinate c_j is towards the node with j^{th} coordinate $(c_j+1)\ mod\ c_j^{max}$. A node $P = [c_1, c_2, c_3]$ has two neighbors across dimension j along the positive and negative directions. They are denoted by $N_j^+(P)$ and $N_j^-(P)$, respectively. A line of nodes $L_q^{c_j,c_k}$ is defined to be the set of c_q^{max} nodes along dimension q, such that the address of each node in this set has coordinates c_j and c_k along dimensions j and k respectively, where $j \ne k \ne q$. The superscripts c_j and c_k are ordered such that $j < k$. This is done in order to avoid the specification of j and k in $L_q^{c_j,c_k}$

Routing in a 3-D torus is simple. When a message destined to d arrives at an intermediate routing node P, that node calculates the shortest distance, δ_j, that the message needs to travel along each dimension j. If $\delta_j > 0$, then the message should travel through dimension j along the positive direction and if $\delta_j < 0$, then the message should travel through the negative direction.

If the present routing node is the destination of a message, then it keeps the message. Otherwise, based on the values of δ_1, δ_2 and δ_3, it routes the message to another node according to the following algorithm in which dir_j denotes the sign of δ_j [19]:

```
1) if $d = P$ then keep the message,
2) elseif ($\delta_1 \ne 0$) then route the message to $N_1^{dir_1}(P)$
3) elseif ($\delta_2 \ne 0$ ) then route the message to $N_2^{dir_2}(P)$
4) else route the message to $N_3^{dir_3}(P)$;
```

This algorithm routes a message through lower dimensions first and follows the shortest path to a destination.

One way to add redundant nodes to the torus architecture is to start with a $c_1^{max} \times c_2^{max} \times (c_3^{max} + K)$ torus, where $K \ge 1$, and use only $c_1^{max} \times c_2^{max} \times c_3^{max}$ nodes as active primary nodes and allocate the remaining nodes as spares. In this particular case, an FTBB is considered to be the set of nodes (including spares) in $L_3^{c_1,c_2}$, where $1 \le c_1 \le c_1^{max}$ and $1 \le c_2 \le c_2^{max}$. The K spare nodes in the FTBB can replace any primary node within that FTBB. For example, in the torus of Figure 9 (where the wrap-around connections along dimension 1 and 3 are not shown), the nodes along a horizontal line belong to an FTBB with $M = c_3^{max} = 5$ primary nodes and $K = 1$ spare node.

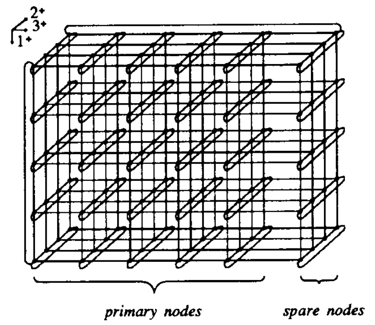


Figure 9. A 5x4x5 Torus augmented with 20 spare nodes.

If $p_1 \in F_{p_1}$ and $p_2 \in F_{p_2}$, then the distance, $dist(F_{p_1}, F_{p_2})$, between the two FTBB's F_{p_1} and F_{p_2} is $|\delta_1| + |\delta_2|$. If $dist(F_{p_1}, F_{p_2}) < dist(F_{p_1}, F_{p_3})$, then F_{p_1} is considered to be closer to F_{p_2} than to F_{p_3}.

In order to satisfy the conditions of Theorem 1 in the 3-D toroidal system, we should have $K < c_3^{max}$. Moreover, we assume that non-active nodes are *shorted* across dimension 3. That is, for any faulty node or unused spare node, the input and output links across dimension 3 are directly connected. Therefore, the active nodes in an FTBB are always connected as a ring across dimension 3, which is the cycle Λ used in the implementation of the two-phase routing algorithm. If a node P fails and a spare node S replaces P, then P is *shorted* and S is brought into the system and inherits P 's address.

A two-phase routing algorithm can be designed to route a message around the faulty nodes and deliver it to the destination. In the first phase, a node first tries to route the message to a neighboring FTBB F along dimension 1 or 2 provided that F is closer to the destination FTBB. If the message cannot be sent to such an FTBB, then the message is sent to a node in the FTBB of the current node. The message does not leave that FTBB until it can be sent to an FTBB that is closer to the destination FTBB. As in the routing for BH-II, the *entry node* in the destination FTBB determines the direction of routing in the second phase. The message is routed along that direction until it reaches the destination node. The *entry node* choses the routing direction as follows: If the *entry node* is a spare node, then the message is routed along the positive direction of dimension 3 otherwise the message is routed along dir_3. The choice in the case of a primary entry node ensures that the shortest path is followed in the absence of faults. In the case of a spare entry node, however, a fixed direction is always chosen (the positive) since the address of an active spare node is not ordered with respect to the other nodes in Λ.

In a fault free system, the distance between the source node of a message and its destination node is given by $\Delta_1 + \Delta_2 + \Delta_3$, where $\Delta_j = \delta_j$ calculated at the source node. In the presence of faults, it is straight forward to show that algorithm ROUTE_T routes a message to its destination through a cycle free path in at most $(K+1)(\Delta_1 + \Delta_2) + (c_3^{max} - 1)$. Although this worst case message path seems to be almost $K+1$ times that of the shortest path, it can be shown that the probability that a message is routed through the worst case path is low and that the average routing overhead is much less than the worst case overhead [3].

5. CONCLUSION

A fault tolerant routing approach is proposed for modular multiprocessor systems that utilize spare nodes to achieve fault tolerance. Routing is performed in two phases; In the first phase, the message is routed to the destination FTBB and in the second phase the message is routed to the destination node within the destination FTBB. This approach uses only local fault information and ensures that in a *live* system messages are delivered to their destinations and never circulate in loops indefinitely. The simplicity and efficiency of the two phase routing are mainly due to the restrictions implicitly imposed on the fault distribution in the system. Specifically, the modular architecture allows spares to only replace faults within their own modules.

Two fault tolerant schemes that use the two-phase routing strategy in binary hypercube architectures are proposed. The first scheme applies a straight forward reconfiguration technique and a fairly simple routing algorithm. It suffers, however, from rapid reliability degradation when the number of faults increases. This rapid degradation is avoided in the second scheme by allowing any of two spare nodes to replace a primary node. The routing algorithms are particularly attractive because, in the absence of faults, they degenerate to the ordinary bit-wise algorithm used in non fault-tolerant hypercubes. The systematic routing strategy presented in this paper is simpler and more general than the routing strategy suggested in [2] for modular hypercubes.

Two phase routing is also applied to three dimensional toroidal systems that are augmented with spares. The hypercube and the torus architectures are only two example that demonstrate the applicability of the technique to modular fault tolerant architectures. In addition to its adaptability to different architectures and its use of only local fault knowledge, the proposed routing approach is relatively easy to develop and results in a low average routing overhead.

References

1. M. Alam and R. Melhem, "Channel Multiplexing in Modular Fault Tolerant Multiprocessors," *Proc. of the International Conference on Parallel Processing*, 1991.
2. M. S. Alam and R. G. Melhem, "An Efficient Modular Spare Allocation Scheme and its Application to Fault Tolerant Binary Hypercubes," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, no. 1, pp. 117-125, Jan 1991.
3. M. S. Alam, "Fault Tolerance in Modular Multiprocessor Systems," PH.D Thesis, department of Computer Science, The University of Pittsburgh., 1991.
4. C. Aykanat and F. Ozguner, "A Concurrent Error Detecting Conjugate Gradient Algorithm on a Hypercube Multiprocessor," *Proc. 17th Int. Symp. on Fault Tolerant Computing*, pp. 204-209, Pittsburgh, July 1987.
5. P. Banarjee, S. Y. Kuo, and W. K. Fuchs, "Reconfigurable Cube-Connected Cycles Architectures," *Proc. 16th Int. Symp. Fault Tolerant Computing*, pp. 286-291, June 1986.
6. D. Blough and N. Bagherzadeh, "Near-optimal Message Routing and Broadcasting in Faulty Hypercubes," *Int. J. of Parallel Programming*, vol. 19, pp. 405-423, 1991..
7. M. Chen and K. Shin, "Depth-First Search Approach for Fault Tolerant Routing in Hypercube Multicomputers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152-159, Apr 1990.
8. E. Chow, H. Madan, and J. Peterson, "An Adaptive Message Routing Network for the Hypercube Computer," *Proc. 15th Symp. on Computer Arch.*, pp. 90-99, 1988.
9. S. Dutt and J. Hayes, "An Automorphic Approach to the Design of Fault Tolerant Multiprocessors," *Proc. 19th Int. Symp. Fault Tolerant Computing*, pp. 496-503, June 1989.
10. J. M. Gordon and Q. F. Stout, "Hypercube Message Routing in the Presence of Faults," *3rd Conf. on Hypercube Concurrent Computers and Applications*, pp. 318-327, 1988.
11. J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a Hypercube in the Presence of Faults," *Proc. of the Symp. on Theory of Computation*, pp. 274-284, May 1987.
12. S. L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures," *J. of Parallel and Dist. Comp.*, vol. 4, pp. 133-172, 1987.
13. S. Y. Kung, S. N. Jean, and C. W. Chang, "Fault-Tolerant Array Processors Using Single-Track Switches," *IEEE Trans. of Computers*, pp. 501-514, April 1989.
14. R. Negrini, R. Stefanelli, and M. G. Sami, "Time Redundancy in WSI Arrays of Processing Elements," *Proc. of the Int. Conf. of Supercomputing Sys.*, pp. 429-438, 1985.
15. A. Olson and K. G. Shin, "Message Routing in HARTS with Faulty Components," *The 19th Int. Symp. on Fault Tolerant Computing Systems*, pp. 331-338, 1989.
16. D. Peleg and B. Simons, "On Fault Tolerant Routing in General Networks," *Proc. Principles of Database Conf.*, pp. 98-107, 1986.
17. D. A. Rennels, "On Implementing Fault-Tolerance in Binary Hypercubes," *Proc. IEEE Fault Tolerant Computing*, pp. 344-349, 1985.
18. A. Singh, "Interstitial Redundancy: An Area Efficient Fault Tolerant Scheme for Large Area VLSI Processor Arrays," *IEEE Trans. on Computers*, vol. 37, no. 11, pp. 1398-1410.
19. M. Wang, M. Cutler, and S. Su, "Reconfiguration of VLSI/WSI Mesh Arrays with Two-Level Redundancy," *IEEE Trans. on Computers*, pp. 547-554, April 1989.
20. R. Yanney and J. Hayes, "Distributed Recovery in Fault Tolerant Multiprocessor Networks," *IEEE Trans. on Computers*, vol. C-35, no. 10, pp. 871-879.