Routing in Modular Fault-Tolerant Multiprocessor Systems

M. Sultan Alam and Rami G. Melhem, *Member, IEEE Computer Society*

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. Such a strategy may be applied to a variety of architectures including binary hypercubes and three-dimensional tori. In the presence of f faults in hypercubes and tori, we show that the worst case length of the message route is min $\{\sigma + f, (K + 1)\sigma\} + c$ where σ is the shortest path in the absence of faults, K is the number of spare nodes in an FTBB, and c is a small constant. The average routing overhead is much lower than the worst case overhead.**

Index Terms—**Sparing, modular multiprocessors, fault-tolerant routing, hypercube multicomputers, mesh connected processors.**

I. 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 nonfaulty processors [3], [7], [13], 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 [5], [6], [9], [10], [12], [18]. In these schemes, different adaptive routing algorithms have been used to by-pass faulty nodes. Olson and Shin proposed a routing algorithm for hexagonal architecture for real-time systems (HARTS) which ensures the delivery of a message as long as there exists a path between the message source and its destination [21]. Peleg and Simons [22] 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 nontrivial problem and the performance degradation can be as high as 50% [3].

Manuscript received Nov. 30, 1992; revised Feb. 28, 1994.

M.S. Alam is with the AT&T Bell Laboratories, Red Hill, N.J.

R.G. Melhem is with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260; e-mail: melhem@cs.pitt.edu.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number D95054.

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 nonfaulty primary node or a spare node that has 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], [4], [8], [16], [20], [25], [30]. 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 bypass 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 has 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, recovery from faults is faster.

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. The routing algorithm is distributed and assumes total node failures in the sense that a failed node cannot be used to route messages. We give sufficient conditions for the two-phase routing algorithm to work correctly, and we show that these conditions can be satisfied in many well known architectures by properly connecting the spares to other nodes in the system. We also apply the routing algorithm to hypercube and toroidal systems and show that in these cases, only local fault knowledge is required.

The rest of this paper is organized as follows: Section II describes the two-phase routing approach and establishes its worst case performance. It also shows how to construct a modular fault-tolerant system in which two-phase routing may be used. The routing approach is then applied in Section III to spare-augmented binary hypercube systems and in Section IV to spare-augmented three-dimensional toroidal systems. Al-

though analytical results show that the worst case routing overhead is relatively high, simulation results show that, on the average, the routing overhead is low.

II. Two Phase Routing

Multiphase routing algorithms have been extensively studied in the literature, especially for binary hypercube architectures [23], [27], [28], [29]. These studies, however, dealt with message permutation models in which each node needs to send a message to a destination node and the destination nodes are all different. The basic idea in these algorithms is to avoid message congestion by introducing randomness in the routing algorithm. Specifically, during the first phase, messages are routed to random destinations and then during subsequent phases they are routed to their correct destinations. In [12], the authors apply the idea of randomization to a hypercube routing algorithm that uses only local information to route messages in the presence of faults. They show that the probability of successful routing is high even for an exceedingly large number of faults. In [17], fault information within a specified radius from a node is used to calculate the routes of messages at that node. This idea is applied to faulty hypercubes and is shown to lead to an efficient distributed routing algorithm.

The work presented in this paper is different from the work cited above because it assumes that the system contains spare nodes and that a spare node inherits the address of the faulty node it replaces. To the best of our knowledge, this work is the first in which the routing algorithm redirects the messages addressed to a faulty node to the spare that replaces it.

A. Routing in Modular Spare Augmented Systems

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 [24]). We assume that the spare node approach is taken. Specifically, we assume that modules are identical and that each module consists of M primary nodes to which K spares are added to replace faulty primary nodes. 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 FTBBs 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 FTBBs 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, F, that contains p, and the other identifying p within F. 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: 1) use the shortest path between two nodes when no faulty nodes are encountered, 2) exhibit graceful performance degradation with the number of faults, and 3) 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 in which the nodes in each FTBB do not need to have information about faults in other FTBBs. 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 the present routing node, p, is in FTBB F_p and that the destination node, d, is in FTBB F_d):

1) While $F_p \neq F_d$ send the message to some node, q, 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 FTBBs. Specifically, a message is always moved to an FTBB that is closer to its destination. This leads to efficient and simple implementations. However, without global fault knowledge, backtracking between FTBBs can be avoided only if the connectivity between any two neighboring FTBBs is rich enough to allow a message to be sent from one to the other in the presence of faults. The minimum connectivity that guarantees the success of two-phase routing is discussed in the next section.

The details of each routing phase depend on the architecture. In fact, given a particular architecture, the choice of FTBB F_q in phase 1 is identical to the choice that a nonfault-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. Figs. 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 Fig. 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 Fig. 2). The second phase of the algorithm starts after the message reaches node d''. Note that the route indicated in Fig. 2b is not the shortest route between p and d.

B. Conditions for the Success of Two-phase Routing

Given a distribution of faults in the system, call a communication link *healthy* if it connects two nonfaulty nodes and call a path between two nonfaulty nodes *healthy* if it consists of *healthy* links. The two-phase routing strategy described in the previous section delivers messages correctly only if certain conditions are satisfied. For instance, phase 1 may fail if there are two adjacent FTBBs that are not connected by at least one

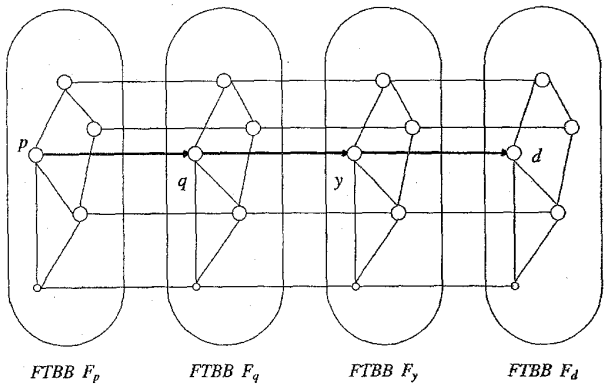


Fig. 1. Two-phase routing with no faults.

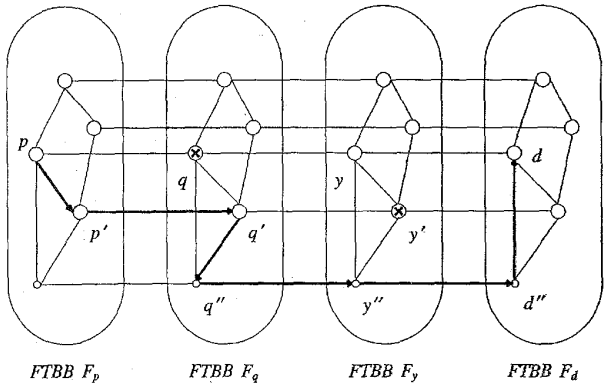


Fig. 2. Two-phase routing with faults at nodes q and y'.

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 FTBBs, 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 nonfaulty 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 between two neighboring FTBBs, F and F' there should be more than K nodes in F that are connected to more than K nodes in F'. In order to be more specific about the connectivity requirement between F and F', let q be a positive integer and let $\phi_{F,F'} = \{n_0, \cdots, n_{K+q-1}\}$ be the set of nodes in F that are connected to nodes in F'. Also, let q' be a positive integer and let $\phi_{F',F} = \{m_0, \cdots, m_{K+q'-1}\}$ be the set of nodes in F' that are connected to nodes in F. Clearly, the worst scenario that can lead to disconnecting F from F' is when the K faulty nodes in F are in $\phi_{F,F'}$ and the K faulty node in F' are in $\phi_{F',F}$. Assume that $q = q'$ and consider the following cases for q.

The Case q = 1. In this case, a healthy link is guaranteed to exist between F and F' for any K faults in F and K faults in F' if each node in $\phi_{F,F'}$ is connected to every node in $\phi_{F',F}$. This case is illustrated by an example in Fig. 3a, where $K = 3$. The figure shows the links between $\phi_{F,F'}$ and $\phi_{F',F}$.

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 the connection between the nodes in $\phi_{F,F'}$ and those in $\phi_{F',F}$ is one-to-one. That is, up to the relabeling of nodes, each node n_i, $0 \le i \le 2K$, is connected to node m_i. This case is illustrated by an example in Fig. 3d.

The above two cases are special cases of the following Lemma. Fig. 3 clarifies the conditions of the lemma when $K = 3$ and $q = 1, 2, 3$, and 4.

LEMMA 1. *For a given q, $1 \le q \le K + 1$, 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 node n_i, in $\phi_{F,F'}$ is connected to $K + 2 - q$ nodes in $\phi_{F',F}$ such that, up to the relabeling of*

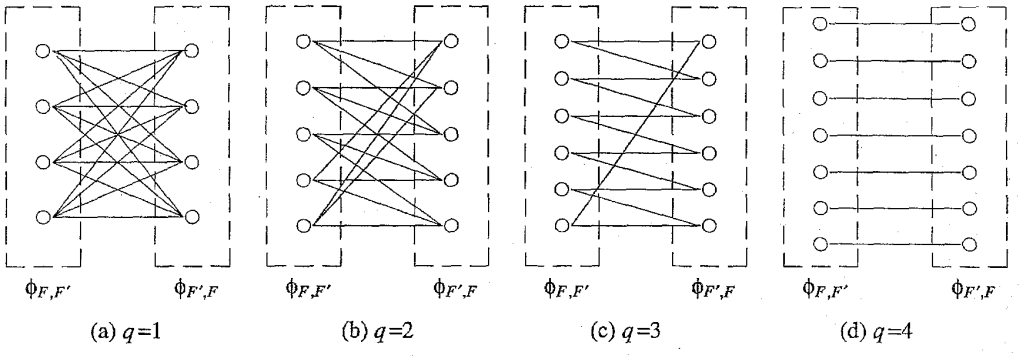


Fig. 3. Required connectivity between neighboring FTBBs for the case $K = 3$.

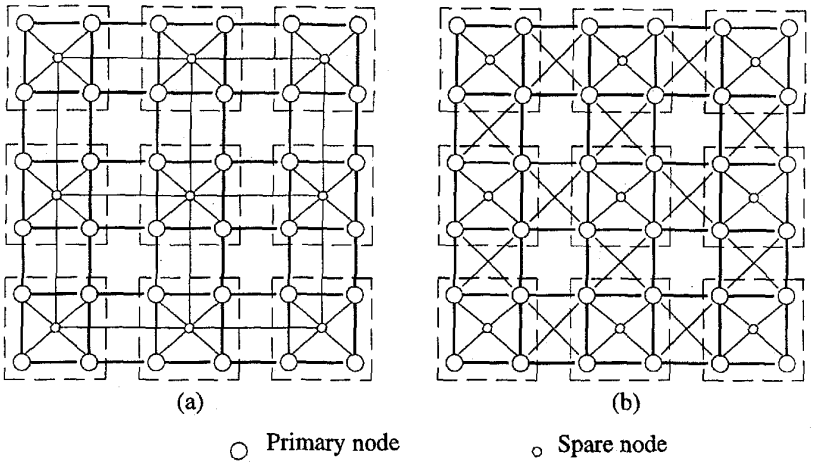


Fig. 4. Fault-tolerant meshes amenable to two-phase routing.

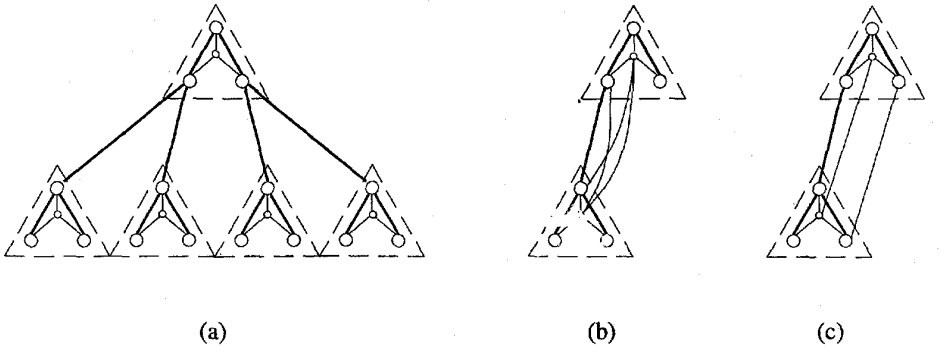


Fig. 5. A fault-tolerant trees architecture (a), the enhanced FTBB interconnections with $q = 1$ (b), and $q = 2$ (c).

nodes, n_i is connected to $m_{(i+u) mod (K+q)}$, $u = 0, \cdots, K+1-q$.

PROOF. Consider the worst case scenario in which K nodes in $\phi_{F,F'}$ are faulty and K nodes in $\phi_{F',F}$ are faulty. Because $\phi_{F,F'}$ contains $K + q$ nodes, there are q nonfaulty nodes in $\phi_{F,F'}$ connected to nodes in $\phi_{F',F}$. Let these nodes be n_{i_a}, $a = 0, \cdots, q-1$ and assume that $i_a < i_{a+1}$. Node n_{i_0} is connected to $K + 2 - q$ nodes in $\phi_{F',F}$ Given the connectivity specified in the hypothesis, no two nodes in $\phi_{F,F'}$ are connected to the same set of nodes in $\phi_{F',F}$ *except when* $q = 1$. Hence, each n_{i_a}, $a > 0$, is connected to at least one node in $\phi_{F',F}$ that is not connected to any previous node n_{i_b} for $b < a$. This means that there are at least $K + 1$ nodes in $\phi_{F',F}$, that are connected to the q nonfaulty nodes in $\phi_{F,F'}$. Thus, for any K faults in $\phi_{F',F}$, 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 each module contains K spares, 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. For example, consider the interstitial mesh architecture proposed in [25]. In Fig. 4, we show two different ways for enhancing the interconnections in that architecture to allow for successful two-phase routing. In this figure, every four nodes and a spare form an FTBB with $K = 1$. FTBBs are enclosed in dashed boxes and the mesh interconnections are highlighted in bold lines. The enhanced connectivity among FTBBs in Fig. 4a and 4b satisfy Lemma 1 with $q = 2$ and $q = 1$, respectively.

Another example is the fault-tolerant tree architecture [26] shown in Fig. 5a. Given that $K = 1$ for this architecture, we show, in Fig. 5b and 5c two ways for enhancing the connectivity between neighboring FTBBs to satisfy Lemma 1.

Lemma 1 may be applied to other architectures such as k-ary trees, Cube Connected Cycles, hypercubes, and tori. In the rest of this paper, however, we will restrict our attention to a class of modular architectures that exhibit a strong inter-module connection as described in the following section.

C. Routing in Architectures with Strong FTBB Interconnections

Given two neighboring FTBBs, these FTBBs are called *strong neighbors* if each node in one FTBB is a neighbor of

exactly one node in the other. Using the notation of the last section, if F and F' are strong neighbors, then, $\phi_{F,F'}$ contains all the nodes of F and $\phi_{F',F}$. contains all the nodes of F'.

Consider two live FTBBs, F and F' of size (M, K). From Lemma 1 and its corollary, if F and F' are strong neighbors, then there is always 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 nonfaulty nodes. If each node on π, 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 states the conditions for the existence of such a path and provides for a uniform implementation of a two-phase routing algorithm at any node in the system.

THEOREM 1. *For a modular system constructed from FTBBs of size (M, K) such that any two neighboring FTBBs are strong neighbors, a two-phase routing algorithm may be designed for that system if:*

1) $M > K$, *and*
2) *The minimum bisection width of the nodes in an FTBB is $K + 1$. That is, after the removal of any K nodes from an FTBB, the remaining M nodes are still connected.*

Moreover, in the presence of f faults, the routing algorithm will route a message in at most $\min\{\sigma + 2f, (2K + 1)\sigma\} + 2M - 1$ steps, where σ is the minimum number of routing steps in the absence of faults.

PROOF. Let F and F' be two neighboring FTBBs and let S be the set of M active nodes in F. From Lemma 1, the first condition of the theorem guarantees that there is a healthy link between any two neighboring FTBBs. The second condition of the theorem guarantees that all the nodes in S are connected. Hence, the conditions for successful two-phase routing are satisfied.

In order to design the routing algorithm, construct a cycle Λ that passes through the M nodes in S. Such a cycle always exists because we can, in the most general case, construct a spanning tree of the nodes in S and then obtain Λ by a preorder traversal of the spanning tree. Clearly Λ does not visit each node more than twice. Moreover, for any node p on Λ, the node following p on Λ is uniquely determined by the node preceding p. Hence, Λ is specified unambiguously.

For the first phase of the routing algorithm, a message can be routed from a node p in S to some node in F' by sending the message along Λ. Given that Λ passes through at least $K + 1$ distinct nodes and that there can be at most K faults in F', then one of the nodes on Λ will be able to send the message to F'. Moreover, given that Λ visits each node at most twice, the message will reach F', in at most $2(K + 1)$ steps. For the second phase of the routing algorithm, the message is sent along Λ, and in less than $2M - 1$ steps, the message will reach its destination.

To compute the routing overhead, consider two nodes s and d and assume that, in the absence of spares (and faults), σ is the length of the shortest path from s to d. In the presense of faults, the FTBBs will be crossed in the same order but to move between two FTBBs it may take up to $2(K + 1)$ steps. Moreover, in the destination FTBB, the message may need up to $2M - 1$ steps. Hence, in the worst case, the message may need $2(K + 1)\ \sigma + 2M - 1$ 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 nodes visited in F while crossing to F' cannot exceed the number of faults in the system. Thus, the number of routing steps from s to d is at most $\sigma + 2f + 2M - 1$. □

The worst case routing performance given in Theorem 1 may be reduced by a factor of almost two if the cycle Λ used in the proof of Theorem 1 is Hamiltonian, that is, if Λ visits each node in S exactly once. Constructing Hamiltonian cycles or checking if one exists in a graph is an NP complete problem. However, for small M and K, finding such a cycle may be possible. In fact, as shown in the following theorem, a condition which is slightly less restrictive than having a Hamiltonian cycle in S is enough to obtain the factor of 2 improvement in the worst case routing. This condition will be used in the routing algorithm given in Section IV.

THEOREM 2. *If, in addition to the first condition in Theorem 1, the following is true:*

For any subset, S, of M nodes in an FTBB, there is a simple 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,

then, a two-phase routing algorithm will route a message in at most $\min\{\sigma + f, (K+1)\sigma\} + M$ steps, where σ is the minimum number of routing steps in the absence of faults and f is the number of faults.

PROOF. For routing a message from a node p in an FTBB, F to a neighboring FTBB, we find a simple path, π, starting at p and passing through K other nodes in S. If p is on Λ, then, π consists of $K + 1$ consecutive nodes on 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. Given that π is a simple path (visits a node at most once), it is clear that the first phase of the routing algorithm will take at most $\min\{\sigma + f, (K+1)\sigma\}$ steps.

For the second phase, a message at a node p is routed 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 or 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 after at most M steps. □

In the following, we give pseudocode for a two-phase routing algorithm that can be used whether the conditions of Theorem 1 or Theorem 2 are satisfied. The algorithm is executed at a node p in FTBB F, and assumes that Λ_F is a cycle (not necessarily simple) that passes through at least $K + 1$ of the M active nodes in F. The active nodes of F that are not on Λ_F are assumed to be directly connected to a node on Λ_F. One of the directions on the cycle Λ_F is defined as positive and the opposite direction is defined as negative. If p is on Λ_F, then $next^+(p)$ and $next^-(p)$ are defined to be the nodes that follow p on Λ_F in the positive and negative directions, respectively.

We will use $next(p)$ to indicate $next^+(p)$. If p is not on Λ_F, then let $next(p)$ be a node on Λ_F 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, 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 the current FTBB, 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 Λ_F, 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 the direction opposite to the one from which the message was received.

We can identify two methods for routing messages along the nodes on Λ_F without keeping global information about active nodes. One method is to compute Λ_F 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. Moreover, the size of the message grows at every visited node. In some cases, however, 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 for each node p in F to compute Λ_F after each fault in F and keep the values of $next^+(p)$ and $next^-(p)$. In general, the computation of Λ_F may require that each node knows about the active nodes in its own FTBB. With this knowledge, the complexity of the algorithm to compute Λ_F is either exponential in M or linear in M, depending on whether a simple Λ_F is to be found (if one exists), or a non-simple Λ_F is to be found, respectively. As discussed in Theorems 1 and 2, a simple Λ_F reduces the routing overhead by almost one half. Hence, especially for small values of M and K, the complexity of finding a simple Λ_F is justified. Moreover, in regular architectures, such as the hypercubes discussed in the next section, it may be possible to compute Λ_F off-line and devise simple rules by which any node p determines the next node on Λ_F by having only local information about its own neighbors.

The bounds given in Theorems 1 and 2 for the number of routing steps do 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 σ in the absence of faults if the routing in phase 2 of the algorithm does not follow Λ, but rather routes the message directly to the destination using some information about the active nodes in the destination FTBB. For small size FTBBs like the ones described in Sections III and IV, only local information is enough to design phase 2 efficiently such that the routing algorithm takes σ steps in the absence of faults.

D. Constructing Architectures Amenable to Two–Phase Routing

A fault-tolerant architecture that is amenable to two-phase routing may be constructed as the cartesian product [11], [29] of an FTBB which satisfies the conditions of Theorem 1 with any connected graph, G. Specifically, each node, v in G is replaced by a copy of the FTBB and each edge, $<v_1, v_2>$, in G is replaced by $K + M$ communication links connecting the $K + M$ nodes in the FTBB that replaces v_1 with the $K + M$ nodes in the FTBB that replaces v_2. Clearly, the FTBBs in the resulting system will satisfy the strong neighbors property. Hypercubes, meshes, and tori are examples of cartesian product networks. For instance, an n-dimensional binary hypercube is the cartesian product of an m-dimensional hypercube, $0 < m < n$, with an $(n - m)$-dimensional hypercube. A mesh is the cartesian product of two linear arrays, and a torus is the cartesian product of two rings.

Two important issues that arise during the above construction are the choice of the graph, G, and the choice of the FTBB. The graph, G, should be chosen such that an efficient routing algorithm exists for routing in G in the absence of faults. This is important since the interconnection between FTBBs follows G, and in the first phase of the routing algorithm of Section II.A, a message should be routed to an FTBB which is closer to the destination node.

The choice of the number of primary nodes, M, and the number of spare nodes, K in each FTBB depends on the reliability requirements of the system, but should satisfy $M > K$ (the first condition of Theorem 1). Given, M and K, the choice of the interconnections within the $M + K$ nodes in the FTBB should satisfy the second condition of Theorem 1. Efficient (polynomial time) algorithms for checking this condition exist based on network flow techniques [14], [19]. Moreover, for small $M + K$, the verification of this condition may be done exhaustively, especially when the interconnections within the FTBB is symmetric. For example, it is easy to verify that each of the FTBBs in Fig. 6 remains connected when any $K = 2$ nodes are removed. The value of M is 6, 8, and 10 in the hypercube of Fig. 6a, the Petersen graph of Fig. 6b [11], [31], and the chordal ring of Fig. 6c, respectively. To clarify the verification procedure, we consider the Petersen graph of Fig. 6b. Because of symmetry, connectivity should be verified for only seven cases. Namely, the removal of the following pairs of nodes: (1, 2), (1, 3), (6, 7), (6, 8), (1, 6), (1, 7), or (1, 8). Checking connectivity for each of these cases is straightforward. Note that none of the FTBBs in Fig. 6 satisfies the conditions of Theorem 1 when $K = 3$. For example, for the FTBB of Fig. 6b, the removal of nodes 3, 5, and 9 will leave node 4 isolated from the remaining nodes.

The more restrictive conditions of Theorem 2 are also satisfied in the FTBBs shown in Fig. 6. Specifically, after the removal of any $K = 2$ nodes, we can always find a cycle such

that each of the remaining M nodes either lie on the cycle or is directly connected to the cycle. Again, the verification procedure can be easily performed when $M + K$ is small and the FTBB is symmetric. For example, when nodes 1 and 2 are removed from the FTBB of Fig. 6b, the cycle is formed by the nodes 6, 8, 10, 7, 9, and the nodes 3, 4, and 5 are directly connected to the cycle.

Another way to construct an FTBB is to start from a known module with M nodes, add K spares, $K < M$, to the module, and connect the K nodes in the FTBB in a way that satisfies Theorems 1 or 2. For example, for $M = 4$ and $K = 1$, we may start from a 4-node ring (a two-dimensional hypercube), and add one spare. The conditions of Theorem 1 will be satisfied if we connect the spare to at least two of the four nodes, and the conditions of Theorem 2 will be satisfied if we connect the spare to at least three of the four nodes. In Section IV.A, we will show that a simple two phase routing can be obtained when symmetry is preserved by connecting the spare to all the nodes in the module as shown in Fig. 7a.

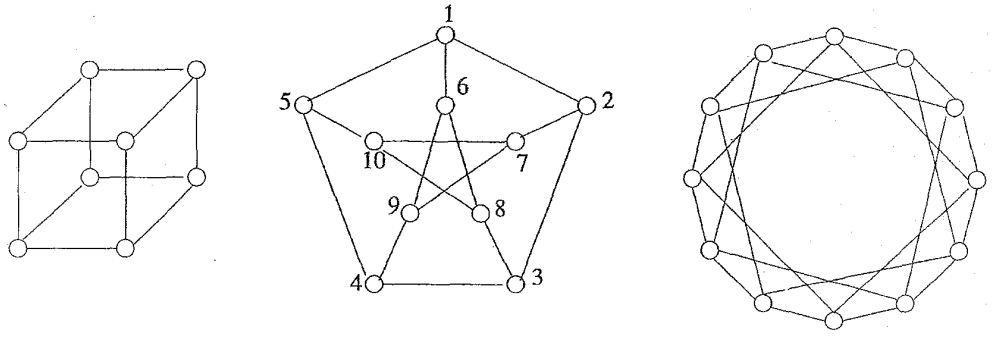
In Fig. 7b, we show an FTBB constructed by adding $K = 4$ spares, s_1, s_2, s_3, and s_4, to a three-dimensional hypercube module with $M = 8$ nodes. The spares are added to four faces of the cube such that each spare can replace any of the four nodes in its face. Links are added between each spare and the four primary nodes that it can replace. Specifically, using the notation in Fig. 7b, spare s_1 can replace any of the nodes 000, 001, 101, or 100. Spare s_2 can replace any of 100, 101, 111, or 110. Spare s_3 can replace any of 110, 111, 011, or 010, and spare s_4 can replace any of 010, 011, 001, or 000. It is shown in [2] that this spare allocation strategy maximizes the reliability under certain hardware restrictions.

The FTBB shown in Fig. 7b does not satisfy the conditions of Theorem 1 for $M = 8$ and $K = 4$. For instance, the removal of the four nodes 000, 001, 110, and 111 disconnects the FTBB into two disjoint components. The connectivity of the FTBB after the removal of any four nodes may be preserved if we add four links to connect the spares as shown in Fig. 7c. In fact, it will be shown in Section III.B, that the FTBB of Fig. 7c satisfies the more restricted conditions of Theorem 2.

In the next section, we will use the two FTBBs depicted in Fig. 7a and 7c to build fault-tolerant systems, and we will show that efficient two-phase routing algorithms may be designed for those systems.

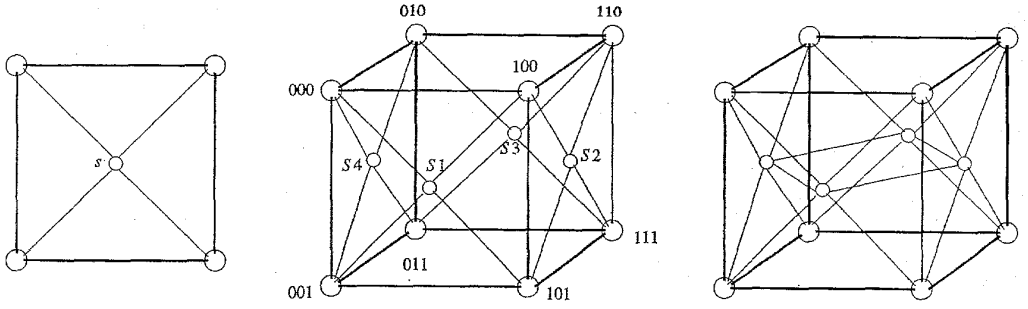
III. Application to Binary Hypercubes

Hypercubes are very modular systems. An n-dimensional binary hypercube, which has 2^n nodes and $n2^{n-1}$ links can be viewed as 2^{n-m} modules each containing 2^m nodes, for any $0 \le m \le n$. Each node, p, is given an n bit address, $p^{(n-1)} \cdots p^{(0)}$, 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



(a) M=6, K=2. (b) M=8, K=2. (c) M=10, K=2.

Fig. 6. Examples of FTBBs that satisfy Theorems 1 and 2.



(a) M=4, K=1 (b) M=8, K=4 (c) adding links to satisfy Theorem

Fig. 7. FTBBs obtained by adding spares to two- and three-dimensional hypercubes.

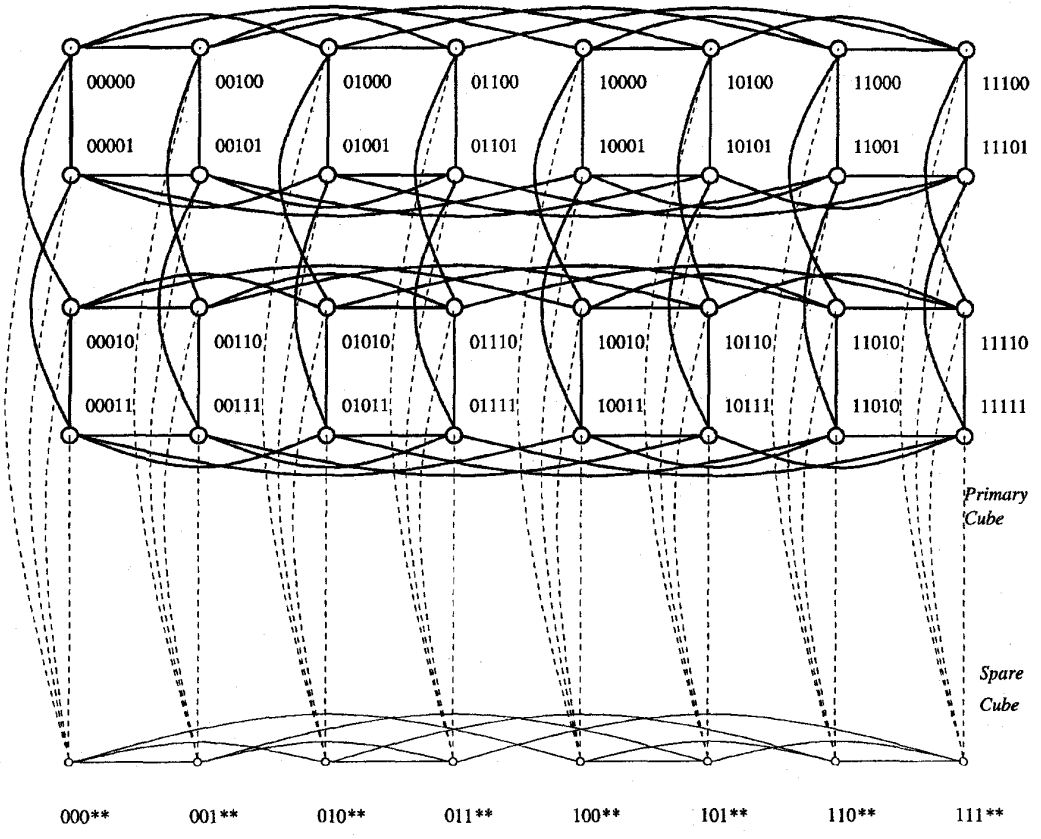


Fig. 8. The augmented hypercube, BH-I.

$d = d^{(n-1)} \cdots d^{(0)}$ then the present routing node, $p = p^{(n-1)} \cdots p^{(0)}$, executes the following routing algorithm ($x^{(i)}$ is the exclusive or of $d^{(i)}$ and $p^{(i)}$):

IF ($x^{(j)} = 0$, for all $0 \le j < n$) 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, we will generalize the FTBB of Fig. 7a such that a spare is added to each m-dimensional subcube in the system, $m \ge 2$ ($m = 2$ in the FTBB of Fig. 7a). 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, the second scheme, BH-II, uses the FTBB shown in Fig. 7c in which more than one spare is added to each module. 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.

A. BH-I: An Architecture with a Single Spare in Each FTBB

As mentioned before, an n-dimensional hypercube is the cartesian product of an m-dimensional hypercube (an FTBB) with an $(n-m)$-dimensional hypercube. A spare node is added to each FTBB and is connected to the 2^m primary nodes in the FTBB. As a result of taking the cartesian product of the resulting FTBB with an $(n - m)$-dimensional hypercube, the 2^{n-m} spare nodes become interconnected as a binary hypercube structure of dimension $n - m$, which is called the *spare cube*. Thus, 2^{n-m} spares and $2^n + (n - m)\, 2^{n-m-1}$ links are added to the original hypercube architecture. 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 an 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 Fig. 8, $n = 5$, $m = 2$ and an FTBB consists of a column of four primary nodes and a spare node (the same FTBB shown in Fig. 7a).

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, F, may be connected by a cycle, Λ_F which, in the absence of faults in F, results from a Gray code embedding of a ring in the 2^m-node subcube [15]. If F contains a fault, then this fault may be bypassed in Λ_F using the spare node. Thus, the conditions of Theorems 1 and 2 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 $d = d^{(n-1)} \cdots d^{(0)}$ is routed to FTBB $d^{(n-1)} \cdots d^{(n-m)}$ first, and then is routed within that FTBB to

the destination node. If the primary node $d^{(n-1)} \cdots d^{(0)}$ is not faulty, then it is the destination node. Otherwise, the spare node in FTBB $d^{(n-1)} \ldots d^{(n-m)}$ 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.

The routing algorithm is described at a node $p = p^{(n-1)} \cdots p^{(0)}$ in FTBB F. Node p 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_F

Algorithm ROUTE_H-I

1) If p = the destination node, 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 $Cj(p)$
   else send the message to $next\ (p)$
4) else /* Phase 2 */
   If p is a spare node, then send the message to the destination node.
   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_H-I** is different from the general algorithm, Two-phase Route. Specifically, in the absence of faults, it does not send messages along Λ_F but rather uses the usual hypercube routing. With this change, the routing overhead is eliminated when the system does not contain any faulty nodes. The number of routing steps is, thus, bounded by $\min\{2\sigma + m, \sigma + f\}$, where σ is the shortest path between the source and the destination in the absence of faults, and f is the number of faults in the system.

B. BH-II: A Double Fault-Tolerant Architecture

In this scheme, we construct the augmented n-dimensional hypercube system by taking the cartesian product of the FTBB shown in Fig. 7c with an $(n-3)$ dimensional hypercube. Each FTBB is a three-dimensional hypercube augmented with four-spares, and 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. The total number of spares in the system is thus 2^{n-1}. Although that number is equal to the number of spares in the BH-I scheme with $m = 1$, each spare is shared by four nodes rather than two nodes. This improves the reliability of the system primarily because any two faults can be tolerated in BH-II, while some two-fault configurations in BH-I with $m = 1$ cannot be tolerated [2].

Dimensions $n-1$, $n-2$, and $n-3$ are chosen to span each module, and thus the eight primary nodes in each FTBB have $n-3$ identical high order bits. These are used to identify the FTBB. As a result of the cartesian product construction, the spare nodes are interconnected as a hypercube of dimension $n-1$, which we call the spare cube. The total number of links added to the nonfault-tolerant system is $(n-1)2^{n-2} + 4 \times 2^{n-1}$ The architecture of a system composed of two such FTBBs is shown in Fig. 9.

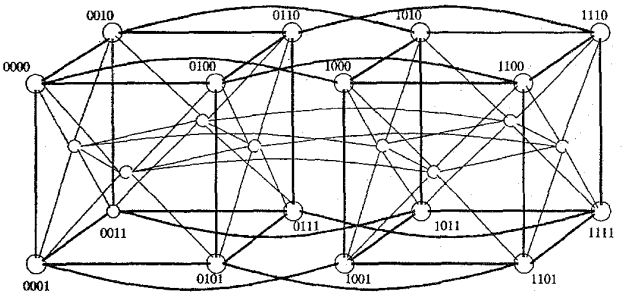


Fig. 9. A four-dimensional augmented hypercube.

In order to show that the FTBB of Fig. 7c satisfies the condition of Theorem 2, we consider the cycle Γ shown in Fig. 10a. Using the notation of Fig. 7b, Γ is the sequence of nodes 000, 001, s_1, 101, 100, s_2, 110, 111, s_3 , 011, 010, s_4, 000. This cycle spans the 12 nodes in the FTBB and may be used as the basis for defining a cycle Λ that spans any eight active nodes in the FTBB.

Let $\Gamma.next(p)$ and $\Gamma.next^{-1}(p)$ be the nodes following and preceding p in Γ and let $\Gamma.next^{j}(p)$ be the node following $\Gamma.next^{j-1}(p)$ in Γ for $j > 1$. Recall that a nonactive 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 nonactive. In fact, it allows three consecutive nodes, p_1, s, p_2, on Γ to be nonactive only if p_1 and p_2 are primary nodes and s is a spare node. With this observation, given any four nonactive nodes in an FTBB, Λ may be defined as follows:

Case 1. If no two nonactive nodes are consecutive in Γ, then Λ is the eight node cycle defined by specifying for each active node, p, the node $next(p)$, which follows p on Λ as follows:

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

An example of this case is given in Fig. 10c where the nodes s_1, 100, 111, and s_4 are nonactive (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 three-dimensional cube shown in Fig. 10b. This cycle is denoted by Λ_g.

Case 2. If three nonactive 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 eight node cycle specified at each active node, p, as follows.

$$next(p) = \begin{cases} \Gamma.next(p) & \text{if } \Gamma.next(p) \textit{ is active} \\ \Gamma.next^2(p) & \text{if } \Gamma.next(p) \textit{ is not active and} \\ & \quad \Gamma.next^2(p) \textit{ is active} \\ \Gamma.next^4(p) & \textit{otherwise} \end{cases}$$

An example of this case is given in Fig. 10d where the nodes 001, s_1, 101, and 110 are not active.

Case 3. Only two of the four nonactive nodes are consecutive on Γ, while each of the other two is preceded and followed by active nodes. The latter 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 $\Gamma.next^{-1}(p_1)$ and $\Gamma.next(p_2)$. These two spares may be directly connected, thus bypassing p_1 and p_2. Finally, if the two nonactive 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 $\Gamma.next^{-1}(p_1)$ and $\Gamma.next(p_2)$. For example, if 001 and s_1 or s_1 and 101 are not active, then the three nodes 001, s_1, and 101 can be bypassed by making *next* (000) = 100 in Λ (see Fig.10e). 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 nonactive 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 Fig. 10e). 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, and 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 nonactive nodes using links not originally on Γ. Specifically, it is possible to go from $\Gamma.next^{-1}(p_1)$ to $\Gamma.next(p_4)$ via p_3 then p_2. An example is given in Fig. 10f where the nodes 001, s_1, s_2, and 110 are not active. Note that because of the spare allocation policy, only two of s_1, 101, 100, and s_2 may be nonactive.

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 $\Gamma.next^j(q)$, $j = 1, 2, 3, 4$ are inaccessible from q.* □

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

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

The requirement, in step 1, that $\Gamma.next(p)$ is not Γ-isolated is needed to implement **case 3** and ensure that if $\Gamma next^2(p)$ and $\Gamma.next^3(p)$ are not active, then $\Gamma.next(p)$ 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.

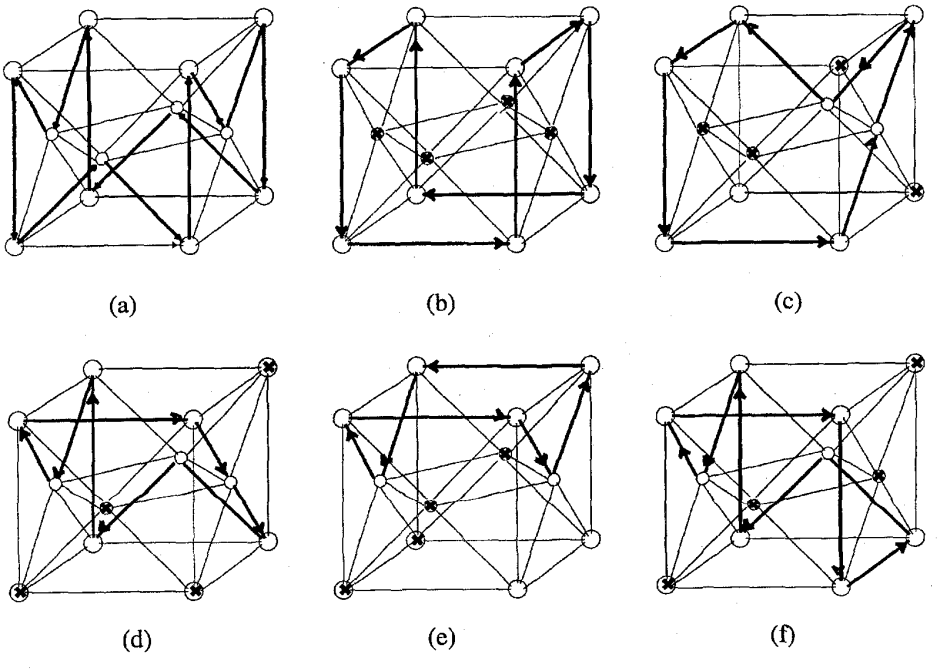


Fig. 10. (a) Γ, and (b – f) Λ for different configurations of active nodes.

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-I**, 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 guarantee 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 the direction opposite to the one from which the message was received.

Note that in the absence of faults, $\Lambda = \Lambda_g$ (see Fig. 10b). 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 Fig. 7b). 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 and in the presence of f faults, the number of routing steps is bounded by $\min\{5\sigma+4, \sigma+f\}$, where σ is the shortest path between the source and the destination in the absence of faults.

C. Experimental Analysis

The upper bounds on the number of routing steps in BH-I and BH-II assume a very pessimistic distribution of faults in the system. The probabilities of such distributions is very small, and the average number of routing steps is much less than this worst case bound. In order to backup this claim, a simulation software tool was designed and used 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 1,000 different FSs. For each FS, 1,000 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 Fig. 11 for $n = 7$. Obviously, the overheads are very small compared to the theoretical overheads of approximately 100% and 400% for BH-I and BH-II, respectively. The routing overhead increases with the number of faults in the system, which is expected.

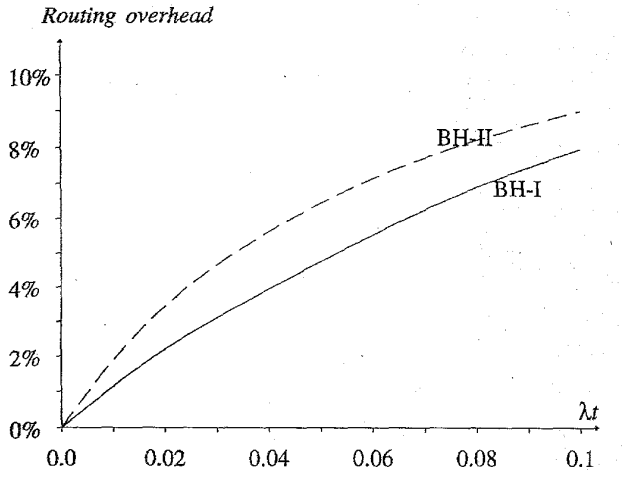


Fig. 11. Average routing overhead.

In Table I, we summarize the characteristics of the two schemes BH-I ($M = 4$ and $K = 1$) and BH-II ($M = 8$ and $K = 1$). The first two columns in the table give the total number of nodes and links in the system. The following three columns give the worst and average routing performance when $\lambda t = 0.1$. Given that the total number of nodes in the two systems is different, we show in the last column the number of faulty nodes when $\lambda t = 0.1$.

TABLE I
COMPARISON OF BH-I ($M = 4$ AND $K = 1$) WITH BH-II ($M = 8$ AND $K = 4$)

	no. of nodes	no. of links	max no. of routing steps	worst case overhead	avg overhead ($n = 7$, $\lambda t = 0.1$)	avg no. of faults ($n = 7$, $\lambda t = 0.1$)
BH-I	$5\times2^{n-2}$	$(5n+6)2^{n-3}$	$2\sigma+2$	100%	8%	16
BH-II	$6\times2^{n-2}$	$(3n+7)2^{n-2}$	$5\sigma+4$	400%	9%	19

IV. Application to Three-Dimensional Toroidal Systems

Consider an $n_1 \times n_2 \times n_3$ three-dimensional torus. The address of a node, p, in the torus is denoted by $[p^{(1)}, p^{(2)}, p^{(3)}]$, where $p^{(j)}, 0 \le p^{(j)} < n_j$, is the coordinate of a node along dimension j. The architecture of a $5 \times 4 \times 5$ torus is shown in Fig. 12. In this figure, the wraparound connections along dimensions 1 and 3 are not shown. The links in this architecture are bidirectional. The positive direction along dimension j from a node with jth coordinate $p^{(j)}$ is towards the

node with jth coordinate $p^{(j)} + 1 \bmod n_j$. For example in Fig. 12, the positive direction along dimension 3 is shown by the arrow labeled 3^+. The negative direction is the one opposite to the positive direction. If $n_j > 1$, a node p has two neighbors across dimension j along the positive and negative directions. They are denoted by $next_j^+(p)$ and $next_j^-(p)$, respectively.

There are many ways to divide a torus into modules that satisfy the strong neighbors property. In this section, we will consider an FTBB to consist of all the nodes that have the same value for the third coordinate. For example in Fig. 12, the nodes [0, 3, 0], [0, 3, 1], [0, 3, 2], [0, 3, 3], [0, 3, 4], and [0, 3, 5] form an FTBB.

Routing in a three-dimensional torus is simple. When a message destined to a destination $d = [d^{(1)}, d^{(2)}, d^{(3)}]$, arrives at an intermediate routing node p, that node calculates three values, δ_j, $j = 1, 2$, and 3, as follows.

$$\delta_j = \begin{cases} d^{(j)} - p^{(j)} & \text{if } |d^{(j)} - p^{(j)}| < n_j - |d^{(j)} - p^{(j)}| \\ n_j - |d^{(j)} - p^{(j)}| & \text{if } |d^{(j)} - p^{(j)}| \geq n_j - |d^{(j)} - p^{(j)}| \\ & \text{and } (p^{(j)} > d^{(j)}) \\ -(n_j - |d^{(j)} - p^{(j)}|) & \text{if } |d^{(j)} - p^{(j)}| > n_j - |d^{(j)} - p^{(j)}| \\ & \text{and } (p^{(j)} < d^{(j)}) \end{cases}$$

If $\delta_j > 0$, then the message should travel δ_j steps along the positive direction of dimension j and if $\delta_j < 0$, then the message should travel $|\delta_j|$ steps along the negative direction of dimension j.

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:

1) if $d = p$ then keep the message,
2) elseif ($\delta_1 \neq 0$) then route the message to $next_1^{dir_1}(p)$
3) elseif ($\delta_2 \neq 0$) then route the message to $next_2^{dir_2}(p)$
4) else route the message to $next_3^{dir_3}(p)$

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

A. Fault-Tolerant Three-Dimensional Toroidal Systems

One way to add redundant nodes to the torus architecture is to start with an $n_1 \times n_2 \times (n_3 + K)$ torus where $K \geq 1$, and use only $n_1 \times n_2 \times n_3$ nodes as active primary nodes and allocate the remaining nodes as spares. In other words, each FTBB will have n_3 primary nodes and K spare nodes that can replace any primary node within that FTBB. With this augmentation, $n_1 n_2 K$ spares are added to the $n_1 n_2 n_3$ primary nodes, and $3n_1 n_2 K$ links are added to the original $3n_1 n_2 n_3$ links. For example, a fault-tolerant version of the torus in Fig. 12 is the $5 \times 4 \times 6$ torus in Fig. 13 for the case $K = 1$.

In order to satisfy the first condition of Theorem 1 in the three-dimensional toroidal system, we should have $K < n_3$. Moreover, to satisfy the condition of Theorem 2, we assume that nonactive 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.

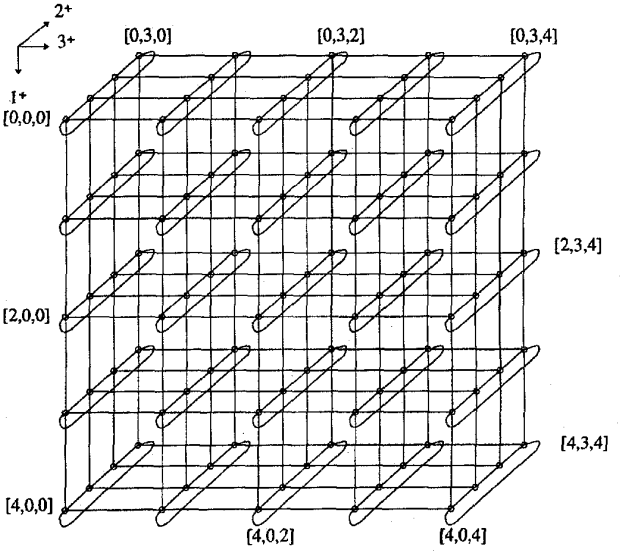


Fig. 12. A $5 \times 4 \times 5$ torus.

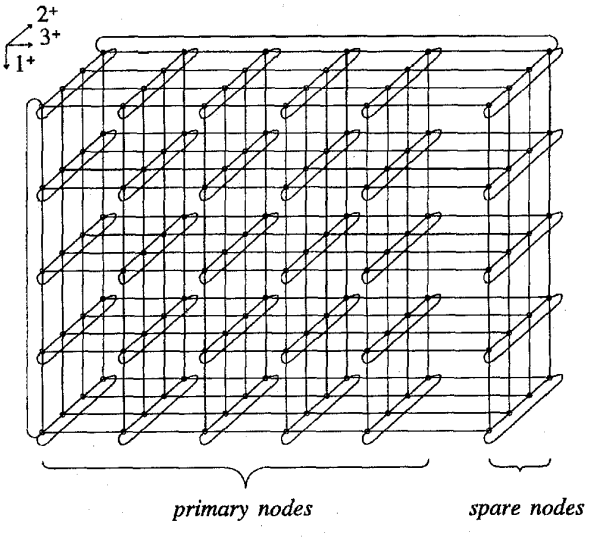


Fig. 13. A $5 \times 4 \times 6$ torus augmented with 20 spare nodes.

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 des-

tination 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* chooses 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 order to formally describe the routing algorithm at a node p, let *prev_dim* be the dimension from which p received the message. When p receives a message destined to d, it executes the following algorithm:

Algorithm ROUTE_T

if $p = d$ then keep the message, else

Phase 1:

1) compute δ_1, δ_2, and δ_3,
2) if ($\delta_1 \neq 0$ and $next_1^{dir_1}(p)$ is not faulty) then route the message to $next_1^{dir_1}(p)$
3) elseif ($\delta_1 \neq 0$ and $next_2^{dir_2}(p)$ is not faulty) then route the message to $next_2^{dir_2}(p)$
4) elseif ($\delta_1 \neq 0$ or $\delta_2 \neq 0$) then route the message to $next_3^{+}(p)$

Phase 2:

5) elseif ($|prev\_dim| \neq 3$) then /* entry node */
   if (p is a spare node) then route the message to $next_3^{+}(p)$
   else route the message to $next_3^{dir_3}(p)$
6) else route the message in the direction opposite to the one from which it is received.

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 $\Delta_j = \delta_j$ is calculated at the source node. In the presence of faults, it is straightforward to show that algorithm **ROUTE_T** routes a message to its destination through a cycle free path in at most $(K + 1)(\Delta_1 + \Delta_2) + (n_3 - 1)$ steps. 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.

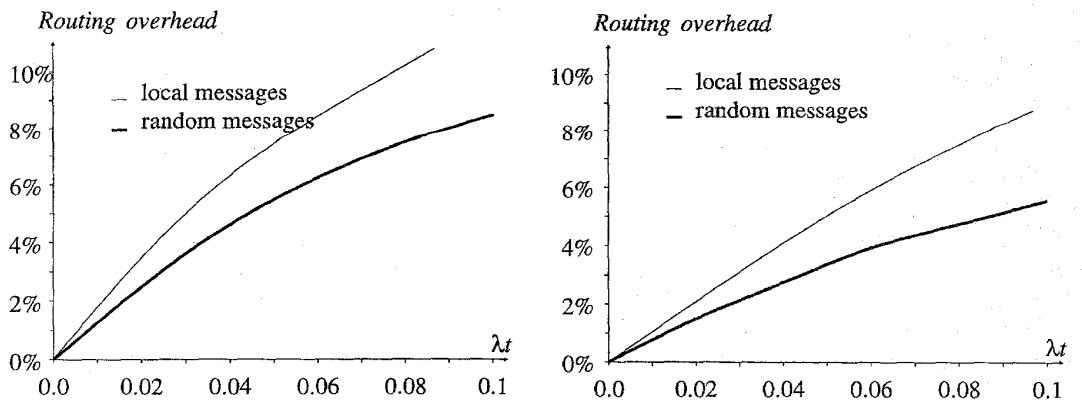
If messages are considered to be random (i.e., not local), then in the absence of faults, a message travels, on the average, $\sum_{j=1}^{3} \frac{n_j}{2}$ steps. For example, an average random message for the fault-tolerant $12 \times 12 \times (12 + 3)$ torus travels 18 steps. Messages can also be local, where the locality of a message is defined in terms of the distance it has to travel. Local messages have Δ_js bounded by $\Delta_j \leq L$, for some $1 < L \leq n_j$. For $n_1 = n_2 = n_3$, the worst case routing overhead for an average random message or a local message can be computed to be approximately $\frac{2}{3}K$.

A simulation study similar to the one described in the last section was used to estimate the average routing overhead. Different simulations were conducted for random and local messages. Random messages were generated by randomly selecting a source node and a destination node while local messages were generated by randomly selecting a source node and a destination node such that $\Delta_j \leq 2$, for $1 \leq j \leq 3$.

The results of the simulations are plotted in Fig. 14. These results show that the routing overhead for local messages is higher than that of random messages. This is because the effect of the inefficient second phase (relative to the first phase) of the routing algorithm is dominant for local messages. The average routing overheads are much less than the $\frac{2}{3}K$ worst case overhead.

V. 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



(a) 12×12×15 torus with K=3 (b) 10×10×11 torus with K=1

Fig. 14. Routing overhead for **ROUTE_T**.

the second phase the message is routed to the destination node within the destination FTBB. This approach 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.

The contribution of this paper is two-fold. First, it provides a general framework for adding fault tolerance to modular multiprocessor architectures in a way that is amenable to two-phase routing. Specifically, given a nonfault-tolerant modular architecture which satisfies the strong neighboring property, Theorem 1 indicates the maximum number of spares that can be added to each module. The connectivity of the nodes in each module is enhanced to satisfy Theorems 1 or 2, and the spares in different modules are connected such that the strong neighbor property is preserved. The second contribution of the paper is a detailed study of specific two-phase routing algorithms for certain spare-augmented hypercube and tori architectures.

Two fault-tolerant schemes that use the two-phase routing strategy in spare-augmented hypercube architectures are proposed. The first scheme applies a straightforward 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 nonfault-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 examples 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.

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation under Grant No. 8911303. A preliminary version of this paper appeared in the *Proceedings of the 22nd Fault-Tolerant Computing Symposium.*

REFERENCES

[1] M. Alam and R. Melhem, "Channel multiplexing in modular fault-tolerant multiprocessors," *J. Parallel and Distributed Computing*, vol. 24, no. 2, pp. 115-131, 1995.

[2] M.S. Alam and R.G. Melhem, "An efficient modular spare allocation scheme and its application to fault-tolerant binary hypercubes," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, no. 1, pp. 117-125, Jan. 1991.

[3] C. Aykanat and F. Ozguner, "A concurrent error detecting conjugate-gradient algorithm on a hypercube multiprocessor," *Proc. 17th Int'l Symp. Fault-Tolerant Computing*, pp. 204-209, Pittsburgh, Penn., July 1987.

[4] K. Belkhale and P. Banerjee, "Reconfiguration strategies for VLSI processor arrays and trees using a modified Diogenes approach," *IEEE Trans. Computers*, vol. 41, no. 1, pp. 83-96, 1992.

[5] D. Blight and R. McLeod, "Non-deterministic adaptive routing techniques for WSI processor arrays," *Proc. IEEE Int'l Workshop Defect and Fault Tolerance in VLSI Systems*, pp. 177-186, 1992.

[6] D. Blough and N. Bagherzadeh, "Near-optimal message routing and broadcasting in faulty hypercubes," *Int'l J. Parallel Programming*, vol. 19, pp. 405-423, 1991.

[7] J. Bruck, R. Cypher, and D. Soroker, "Tolerating faults in hypercubes using subcube partitioning," *IEEE Trans. Computers*, vol. 41, no. 5, pp. 599-605, 1992.

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

[9] M. Lchen and K. Shin, "Depth-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 152-159, Apr. 1990.

[10] E. Chow, H. Madan, and J. Peterson, "An adaptive message routing network for the hypercube computer," *Proc. 15th Symp. Computer Architecture*, pp. 90-99, 1988.

[11] S. Ohring and S. Das, "The folded Petersen cube networks: New competitors for the hypercube" *Proc. Fifth IEEE Symp. Parallel and Distributed Computing*, 1993.

[12] J. Gordon and Q. Stout, "Hypercube message routing in the presence of faults," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, vol. 1, pp. 318-327, 1988.

[13] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a hypercube in the presence of faults," *Proc. Symp. Theory of Computation*, pp. 274-284, May 1987.

[14] M. Garey and D. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Fransisco, Calif.: W.H. Freeman, 1979.

[15] S.L. Johnsson, "Communication efficient basic linear algebra computations on hypercube architecures," *J. Parallel and Distributed Computers*, vol. 4, pp. 133-172, 1987.

[16] S.Y. Kung, S.N. Jean, and C.W. Chang, "Fault-tolerant array processors using single-track switches," *IEEE Trans. Computers*, vol. 38, pp. 501-514, Apr. 1989.

[17] T. Lee and J. Hayes, "Routing and broadcasting in faulty hypercube computers," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, vol. 1, pp. 346-354, 1988.

[18] D. Linder and J. Harden, "An adaptive and fault tolerant routing strategy for k-ary n-cubes," *IEEE Trans. Computers*, vol. 40, no. 1, pp. 2-12, 1991.

[19] U. Manber, *Introduction to Algorithms, A Creative Approach.* Addison-Wesley, 1989.

[20] R. Negrini, R. Stefanelli, and M.G. Sami," Time redundancy in WSI arrays of processing elements," *Proc. Int'l Conf. Supercomputing Systems*, pp. 429-438, 1985.

[21] A. Olson and K.G. Shin, "Message routing in HARTS with faulty components," *Proc. 19th Int'l Symp. Fault-Tolerant Computing Systems*, pp. 331-338, 1989.

[22] D. Peleg and B. Simons, "On fault-tolerant routing in general networks," *Proc. Principles of Database Conf.*, pp. 98-107, 1986.

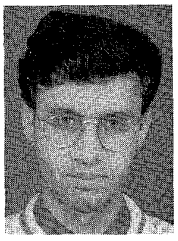
[23] M. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335-348, 1989.

[24] D.A. Rennels, "On implementing fault-tolerance in binary hypercubes," *Proc. IEEE Fault Tolerant Computing*, pp. 344-349, 1985.

[25] A. Singh, "Interstitial redundancy: An area efficient fault-tolerant scheme for large area VLSI processor arrays" *IEEE Trans. Computers*, vol. 37, no. 11, pp. 1,398-1,410, 1988.

[26] A. Singh, "A reconfigurable modular fault-tolerant binary tree architecture," *Proc. 17th Int'l Symp. Fault-Tolerant Computing*, pp. 298-304, June 1987.

[27] L. Valient and G. Brebner, "Universal schemes for parallel communication," *Proc. ACM Symp. Theory of Computing*, pp. 263-277, 1981.
[28] L. Valiant, "A scheme for fast parallel communication," *Siam J. on Computing*, vol. 11, no. 2, pp. 350-361, 1982.
[29] L. Valiant, "Optimality of a two-phase routing in interconnection networks," *IEEE Trans. Computers*, vol. 32, no. 9, pp. 861-863, 1983.
[30] M. Wang, M. Cutler, and S. Su, "Reconfiguration of VLSI/WSI mesh arrays with two-level redundancy," *IEEE Trans. Computers*, pp. 547-554, Apr. 1989.
[31] G. Chartrand and R. Wilson, "The Petersen graph," *Graphs and Applications*, F. Harary and J. Maybee, eds., 1985.



M. Sultan Alam graduated from the University of Petroleum and Minerals, Dhahran, Saudi Arabia with a bachelors degree in computer science and engineering in 1985. Alam completed his Masters and PhD degrees at the University of Pittsburgh in 1987 and 1991, respectively. Currently, he is a member of the technical staff at AT&T Bell Laboratories, Red Hill, New Jersey. He has worked as a performance and reliability consultant for different projects within AT&T. He is currently involved in designing highly available software systems to surveil and monitor the AT&T Bell Laboratories switching network. His research interests are in parallel processing, fault tolerance, and software reliability engineering.

Rami G. Melhem received his BE in electrical engineering from Cairo University in 1976; an MA degree in mathematics and an MS degree in computer science from the University of Pittsburgh in 1981; and a PhD degree in computer science from the University of Pittsburgh in 1983. He is a professor of computer science at the University of Pittsburgh. Previously, he was an assistant professor at Purdue University and an assistant and associate professor at the University of Pittsburgh. He has published numerous papers in the areas of systolic architectures, parallel computing, fault tolerance, and optical computing. Dr. Melhem has served on several program committees for conferences and workshops; he is on the editorial board of *IEEE Transactions on Computers*. He was guest editor for a special issue of the *Journal of Parallel and Distributed Computing* on Optical Computing and Interconnection Systems. Dr. Melhem is a member of the IEEE Computer Society, the Association for Computing Machinery, and the International Society for Optical Engineering.