Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks

Rajendra V. Boppana and Suresh Chalasani

Abstract—**We present simple methods to enhance the current minimal wormhole routing algorithms developed for high-radix, low-dimensional mesh networks for fault-tolerant routing. We consider arbitrarily-located faulty blocks and assume only local knowledge of faults. Messages are routed minimally when not blocked by faults and this constraint is relaxed to route around faults. The key concept we use is a fault ring consisting of fault-free nodes and links can be formed around each fault region. Our fault-tolerant techniques use these fault rings to route messages around fault regions. We show that, using just one extra virtual channel per physical channel, the well-known *e*-cube algorithm can be used to provide deadlock-free routing in networks with nonoverlapping fault rings; there is no restriction on the number of faults. For the more complex faults with overlapping fault rings, four virtual channels are used. We also prove that at most four additional virtual channels are sufficient to make fully-adaptive algorithms tolerant to multiple faulty blocks in *n*-dimensional meshes. All these algorithms are deadlock- and livelock-free. Further, we present simulation results for the *e*-cube and a fully-adaptive algorithm fortified with our fault-tolerant routing techniques and show that good performance may be obtained with as many as 10% links faulty.**

Index Terms—**Adaptive routing, block faults, deadlocks, fault-tolerant routing, mesh networks, multicomputer networks, performance evaluation, wormhole routing.**

I. Introduction

Point-to-point k-ary n-cube and related networks are being used in many experimental and commercial parallel computers [1], [27], [24], [30]. In addition, k-ary n-cube based networks are becoming popular for reliable and high-speed communication switching [26]. A k-ary n-cube network has an n-dimensional grid structure with k nodes (processors) in each dimension such that every node is connected to two other nodes in each dimension by direct communication links.

The *wormhole* (WH) switching technique by Dally and Seitz [12] has been widely used in the recent multicomputers [27], [24], [30]. In the WH technique, a packet is divided into a sequence of fixed-size units of data, called *flits*. If a communication channel transmits the first flit of a message, it must transmit all the remaining flits of the same message before transmitting flits of another message. To avoid deadlocks among messages, multiple virtual channels are simulated on each physical channel and a pre-defined order is enforced on the allocation of virtual channels to messages. Alternatives to the wormhole switching are the virtual-cut-through [20] and store-and-forward [18] switching techniques, which require more storage at each routing node.

For fault-free networks, some of the most important issues in the design of a routing algorithm are high throughput, low-latency message delivery, avoidance of deadlocks, livelocks, and starvation, and ability to work well under various traffic patterns [14]. For networks with faults, a routing algorithm should exhibit a few additional features: graceful performance-degradation, and ability to handle faults with only a small increase in routing complexity and local knowledge of faults—each nonfaulty processor knows only the status of its neighbors.

The well-known e-cube algorithm routes messages in a strictly ascending order of dimensions; that is, a message takes hops in dimension 0 (if any), then in dimension 1 (if any), and so on to reach its destination. Thus, the e-cube algorithm uses a fixed path to route messages between a pair of nodes even when multiple shortest paths are available, and is termed nonadaptive. Routing algorithms that permit the use of all the paths between a source-destination pair by messages are known as fully-adaptive routing algorithms.

Problem definition. In this paper, we address the issue of incorporating fault-tolerance into both fully-adaptive and nonadaptive wormhole routing algorithms. We first develop our techniques for the e-cube algorithm, which has been used in many recent parallel computers [27], [33], [1], [24], [30]. Next, we develop a methodology to enhance fully-adaptive algorithms for fault-tolerant routing.

Our routing techniques require only local knowledge of faults and work correctly when faulty components are confined to one or more rectangular blocks. With the current technology and anticipated advances in packaging, it is reasonable to expect that each node (processor-memory-router combination) of a multicomputer could be implemented as a single chip or as a multichip-module, with several such nodes placed on a printed circuit board. The block-fault model used in this paper accurately models faults at the chip, multichip module, and board levels.

Related results. Routing algorithms for WH and virtual cut-through switching techniques has been the subject of extensive research in recent years [8], [13], [16], [11], [17], [29], [3]. Several results have been reported for fault-tolerant routing in hypercubes; see, for example, [22], [7], [31], [32] and the references therein. The results for hypercube exploit the rich interconnection structure of hypercubes and are not suitable for high-radix, low-dimensional meshes.

Manuscript received Dec. 7, 1993; revised May 31, 1994.
R.V. Boppana is with the Division of Computer Science, the University of Texas at San Antonio, San Antonio, TX 78249-0664; e-mail: boppana@ringer.cs.utsa.edu.
S. Chalasani is with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706-1691; e-mail: suresh@cauchy.ece.wisc.edu.
IEEECS Log Number C95068.

Reddy and Freitas [28] use global knowledge of faults—which is difficult to maintain in a massively parallel processor—and routing tables to study the performance limitations caused by faults. Gaughan and Yalamanchili [15] use a pipelined circuit-switching mechanism with backtracking for fault-tolerant routing. Glass and Ni [17] present the *negative-first* algorithm, which tolerates up to $(n - 1)$ faults in an n-dimensional mesh without any extra virtual channels. Unfortunately, the negative-first and its related algorithms do not have good performance for the fault-free case [4], and the number of faults tolerated is small, for example, one in a 2D mesh.

The works by Dally and Aoki [11] and by Chien and Kim [8] are the most relevant to our work. The similarities and differences between our work and theirs are explained below.

Chien and Kim [8] present a partially adaptive algorithm to handle block faults in meshes. Their method uses three virtual channels for fault-tolerant routing. However, their method cannot handle faults on the boundaries of mesh without excessive loss of computational power. For example, to handle a node fault in the top row of a 2D mesh, all other nodes in that row must be labeled faulty.

In contrast, our work applies to fully-adaptive and nonadaptive algorithms and handles faults on the network boundaries using four virtual channels. We also present a method that requires only two virtual channels for simpler fault cases.

Dally and Aoki [11] present fault-tolerant algorithms based on the concept of dimension reversal, which occurs whenever a message takes a hop in a dimension lower compared to that of the previous hop. A message can be routed adaptively if the number of dimension reversals it has taken is less than the number of highest virtual channel class (static algorithm) or if the message finds a free channel in other outgoing channels of the current host in a finite amount of time (dynamic algorithm). The main advantage of their algorithm is that arbitrarily shaped faults can be handled. With the dimension-reversal schemes of Dally and Aoki, a message may lose its adaptivity as described above. A message that has lost adaptivity is routed by the e-cube algorithm and is not guaranteed to be delivered to its destination if there are faults in the network. Thus the number of virtual channels needed and the number of faults tolerated is dependent on the location of faults and the misrouting logic—which determines when to change the dimension of travel—used.

In contrast, our algorithms can tolerate any number and combination of rectangular faulty blocks with simple logic using only a constant number of virtual channels. Further, each and every message that is injected into the network is guaranteed of delivery to its destination.

The rest of this paper is organized as follows. Section II describes the fault-model used in this paper. This section also introduces the concept of a fault ring. Section III presents a fault-tolerant e-cube algorithm that tolerates multiple faulty blocks in two-dimensional meshes. Section IV proves that any fully-adaptive routing technique can be modified to tolerate multiple faulty blocks in a two-dimensional mesh using four additional virtual channels per physical link. Section V gives the simulation results on the performance of these schemes in the presence and absence of faults. Section VI presents techniques to extend our results to n-dimensional meshes. Section VII summarizes the work reported in this paper and presents possible directions for future work.

II. Preliminaries

We use the following notation for mesh and torus networks. A (k, n)-torus (also called k-ary n-cube) has n dimensions, denoted $\text{DIM}_0, \ldots, \text{DIM}_{n-1}$, and $N = k^n$ nodes. Each node is uniquely indexed by an n-tuple in radix k [2]. Each node is connected via communication links to two other nodes in each dimension. The neighbors of the node $x = (x_{n-1}, \ldots, x_0)$ in DIM_i, $0 \le i < n$, are $(x_{n-1}, \ldots, x_{i+1}, x_i \pm 1, x_{i-1}, \ldots, x_0)$ where addition and subtraction are modulo k. A link is said to be a wrap-around link if it connects two neighbors whose addresses differ by $k - 1$ in DIM_i. A (k, n)-mesh is a (k, n)-torus with the wrap-around connections missing. The well-known binary hypercube is the $(2, n)$-mesh. In this paper, we consider (k, n)-mesh networks with small n, large k, and bidirectional links—implemented using two unidirectional physical communication channels. We denote the link between nodes x and y by $<x, y>$ and virtual channels of class i as c_i. We assume that a crossbar is used in each router, as in the Torus Routing Chip [9], to connect its input channels to its output channels.

A message that reaches its destination is consumed in finite time. Following Dally and Seitz [12], we use the concept of channel dependency graphs and multiple virtual channels to investigate deadlock properties of routing algorithms. Using extra logic and buffers, multiple virtual or logical channels can be simulated on a physical channel in a time-demand multiplexed manner [10], [5]. We always specify the number of virtual channels on per physical channel basis. The channel dependency graph is formed as follows. The virtual channels of the network form the nodes of the channel dependency graph; there is an edge from virtual channel u to virtual channel v if the routing algorithm allows the use of v after using u for any message.

In the remainder of this section, we describe the fault model considered in this paper and the concept of fault-rings, which are created by faults. To simplify presentation, we discuss these concepts for two-dimensional (2D) meshes. We label the sides of a 2D mesh as North, South, East and West. We generalize these concepts to multidimensional meshes in Section VI. These results can be extended to tori with suitable modifications [6].

A. The Fault Model

We consider both node and link faults. Link faults can also be used to model partial faults of routers, for example, when the buffer space associated with a channel is faulty in a node that is otherwise perfect. A node fault is modeled by making all links incident on it faulty. A node with no nonfaulty links incident on it is considered faulty. We assume that faults are nonmalicious—a failed component simply ceases to work.

Therefore, only nonfaulty processors generate messages. Furthermore, messages are destined only to fault-free processors. These assumptions are commonly made in fault analyses in literature.

We model multiple simultaneous faults, which could be connected or disjoint. We assume that the mean time to repair faults is quite large, a few hours to many days, and that the existing fault-free processors are still connected and thus should be used for computations in the mean time.

The fault information may be made global—each fault-free node knows all faults in the system—or local—each fault-free node knows the status of its neighbors only. The global-knowledge model requires some form of routing tables, which are unnecessary for fault-free routing. Distributing fault information to each and every node in a massively parallel processor is expensive. Also, routing algorithms that depend on global fault information should have alternate schemes to transmit fault status messages during the *transition* period—the interval between the occurrence of a fault and the time at which this fault is known globally. Therefore, we develop fault-tolerant algorithms that can work with local fault information.

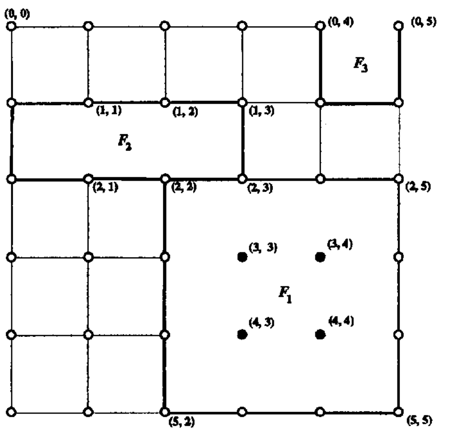


Fig. 1. Examples of f-rings and f-chains in a mesh. Faulty nodes are shown as filled circles, and faulty links are not shown. There are three f-regions, and the corresponding f-rings and f-chain are indicated by thick lines.

A *fault set* is a set of faulty nodes and links. A set F of faulty nodes and links indicates a (rectangular) fault block, or f-region, if there is a rectangle connecting various nodes of the mesh such that a) the boundary of the rectangle has only fault-free nodes and channels and b) the interior of the rectangle contains all and only the components given by F. A fault set that includes a component from one of the four boundaries—top and bottom rows, left most and right most columns—of a 2D mesh denotes a rectangular fault block, if the above definition is satisfied when the mesh is extended with nonfaulty *virtual* rows and columns on all four sides. Fig. 1 indicates three rectangular fault blocks: $F_1 = \{(3, 3), (3, 4), (4, 3), (4, 4)\}$, $F_2 = \{<(1, 1), (2, 1)>, <(1, 2), (2, 2)>\}$, and $F_3 = \{<(0, 4), (0, 5)>\}$.

We use the *block-fault* model, in which each fault belongs to exactly one fault block. Under the block-fault model, the complete set of faults in a 2D mesh is the union of multiple fault blocks. For example, the complete fault set for the network in Fig. 1 is $F_1 \cup F_2 \cup F_3$.

It can be easily verified that, under the block fault model, each fault-free node has faulty links incident on it in at most one dimension. Fault blocks that touch both row boundaries or both column boundaries disconnect the network and, hence, are not considered. If the network is disconnected, our results can be applied to each of the connected subnetworks.

It is noteworthy that Chien and Kim also consider block (*convex* in their terminology) faults in two- and higher-dimensional meshes [8]. Their model does not effectively deal with faults on the network boundary, however. For example, to handle the faulty-link <(0, 4), (0, 5)> in Fig. 1, Chien and Kim label all nodes in row 0 faulty. Our fault model treats such a fault as rectangular and handles it without labeling other working processors and links faulty.

B. Fault Rings and Fault Chains

Conceptually, fault regions may be considered as islands of faults in a sea of communication channels and nodes. In the same manner as a ship is navigated around an island, it should be feasible to route a message around fault regions. For this purpose, we use the concept of *fault rings*, denoted f-rings.

For each f-region in a network with faults, it is feasible to connect the fault-free components around the region to form a ring or chain. This is the fault ring, f-ring, for that region and consists of the fault-free nodes and channels that are adjacent (row-wise, column-wise, or diagonally) to one or more components of the fault region. The f-ring of an f-region is of rectangular shape. For example, the f-ring associated with the f-region F_1 in Fig. 1 has nodes (2, 2), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5), (5, 5), (5, 4), (5, 3), (5, 2), (4, 2), (3, 2) on its boundary. Notice that a fault-free node is in the f-ring only if it is at most two hops away from a faulty node. There can be several fault rings, one for each f-region, in a network with multiple faults. In a (k, n)-mesh, a link may be common to up to n f-rings and a node common to up to $2n$ f-rings. A set of fault rings are said to overlap if they share one or more links. For example, the f-rings of F_1 and F_2 in Fig. 1 overlap with each other, since they share link <(2, 2), (2, 3)>.

Forming a fault-ring around an f-region is not possible when the f-region touches one or more boundaries of the network (e.g., F_3 in Fig. 1). In this case, a *fault chain*, f-chain, rather than an f-ring is formed around the f-region. There are four basic types of fault chains that are formed when an f-region touches exactly one of the (2D) network boundaries. The fault chain of an f-region that touches more than one edge of the network can be synthesized from these four basic f-chains. The nodes at which an f-chain touches the network boundaries are the *end nodes* of the f-chain. Since the links in an f-chain are undirected, we can form a directed ring, which spans from one end of the f-chain to the other end and then back.

An f-ring (respectively, f-chain) represents a two-lane (respectively, one-lane) path to a message that needs to go

```
Procedure Form-Fring()
/* Comment: Creates an f-ring around an f-region.
Nodes on the f-ring are in one of the eight possible positions:
North West (NW) corner, North, North East (NE) corner, East, South East (SE) corner, South, South
West (SW) corner, and West. */
1 Each node, if it has faulty neighbors in DIM_0 sends one message (with information on its faulty
    neighbors) to each of its non-faulty neighbors in DIM_1, and vice-versa.
2 Each node that has a faulty link incident or has received one or more fault status messages determines
    its position on the f-ring based on the set of messages it received about the f-region and Table 1.
```

Fig. 2. An algorithm to form f-rings and f-chains in a 2D mesh.

TABLE I
NODE POSITIONS BASED ON STATUS MESSAGES FROM NEIGHBORS. LET x BE THE NODE WHOSE POSITION IS TO BE DETERMINED. N_x, E_x, S_x, AND W_x DENOTE THE NODES ADJACENT TO x IN THE NORTH, EAST, SOUTH, AND WEST DIRECTIONS, RESPECTIVELY

Node Position	Fault Status Messages Received
North	x's South link is down and one or both of the following occurred: 1. E_x's South link is down, 2. W_x's South link is down.
NE Corner	No faulty links are incident on x and one or both of the following occurred: 1. W_x's South link is down, 2. S_x's West link is down.
East	x's West link is down and one or both of the following occurred: 1. N_x's West link is down, 2. S_x's West link is down.
SE Corner	No faulty links are incident on x and one or both of the following occurred: 1. N_x's West link is down, 2. W_x's North link is down.
South	x's North link is down and one or both of the following occurred: 1. E_x's North link is down, 2. W_x's North link is down.
SW Corner	No faulty links are incident on x and one or both of the following occurred: 1. N_x's East link is down, 2. E_x's North link is down.
West	x's East link is down and one or both of the following occurred: 1. N_x's East link is down, 2. S_x's East link is down.
NW Corner	No faulty links are incident on x and one or both of the following occurred: 1. S_x's East link is down, 2. E_x's South link is down.

through the f-region contained by the f-ring. Thus, an f-ring simulates four paths to route messages in two dimensions. Depending on the size of the f-region, physical channels in an f-ring may need to handle a large amount of traffic compared to the other fault-free physical channels. Further, routing messages around one or more fault-rings creates additional possibilities for deadlocks. Hence, wormhole routing algorithms must be designed to handle the additional congestion and deadlocks caused by faults.

C. Formation of Fault Rings

We assume that nodes can test themselves using any one of the many popular self-test techniques [23]. Each node monitors the status of the links incident on it. When node x becomes faulty, each of its neighbors concludes that the appropriate link connecting itself and x is faulty. In the block fault model, each node has faulty links in at most one dimension. Therefore, if a node has faulty links in more than one dimension, it simply stops sending status signals to its neighbors. This procedure is iterated several times (bounded by the network diameter) until no new nodes or links are set faulty. The net faults after this process satisfy the block fault model.

First, let us consider a single fault region in a two-dimensional mesh. The formation an f-ring around this f-region is a two-step process as shown in Fig. 2. A node on an f-ring or f-chain, sends up to two messages and receives up to two messages. For example, for the f-region F_1 in Fig. 1, node (3, 2) sends messages to nodes (2, 2) and (4, 2) that its East link is down, and receives a message from (4, 2) that (4, 2)'s East link is also down. Node (2, 2) also receives a fault status message from (2, 3) that (2, 3)'s South link is down. Node (2, 2) determines that (3, 3) is faulty and that it is the NE corner node for the f-ring covering (3, 3). Node (3, 2) concludes that it is a West node, since links <(3, 2), (3, 3)> and <(4, 2), (4, 3)> are faulty.

Formation of f-chains is also done using the procedure for f-rings. The difference is two nodes on the f-chain mark themselves as the end nodes. The other nodes on the f-chain need not know this and may simply assume that they are on an f-ring. In fact, our fault tolerant routing logic for nodes (other than end nodes) on f-chains is the same as the one used for nodes on f-rings. Only the end nodes of f-chain use an additional rule to route messages.

If multiple faults occur simultaneously, a node may send/receive messages about multiple f-regions. There can be at most two faulty links incident on a node even with multiple f-regions. Hence, a node sends at most two messages to each of its non-faulty neighbors, which are at most $2(n - 1)$. Thus a node in a (k, n)-mesh may send up to $4(n - 1)$ messages to its neighbors and receive up to $4n$ messages in the formation of f-rings and f-chains. By fortifying the messages with information on faulty link direction and dimension, it is feasible to separate the messages on faults for different f-regions. Suppose the f-rings for F_1 and F_2 in Fig. 1 are to be formed simultaneously. Node (2, 3) may receive a fault status message from (2, 2) that (2, 2)'s North link is down. Since (2, 3) knows that its North link is up and did not receive any message from (1, 3), it is the corner node for the f-ring covering the North link of (2, 2). Now suppose that (2, 3) also detects that its South link is faulty and receives a message that the South link of (2, 4) is faulty at the same time it receives the fault status message from (2, 2). Node (2, 3) can separate the messages into two categories, since one set of messages inform about North links and the second set about South links.

In summary, f-rings and f-chains are formed using only near-neighbor communication among fault-free processors. Hence, the transition period from the occurrence of a fault to the formation of appropriate f-rings or f-chains is short. The actual details of implementation are dependent on the design of the OS and network interface, which are not addressed in this paper.

The fault-tolerant wormhole routing techniques presented in the next few sections use 1–4 additional virtual channels. The additional virtual channels required by our techniques can be used for routing messages in the absence of faults, to improve performance. However,. when faults occur, the messages using these additional virtual channels on the fault ring, may have to be first drained before the fault-tolerant routing algorithms begin to operate. If these messages are not drained in a specified amount of time, they may have to be dropped from the network. Similarly, messages that are currently in transit through a faulty link or node should also be dropped from the network. The sources of dropped messages retransmit the messages, if they do not receive the acknowledgement in a specified amount of time. This recovery process at the network level should be integrated with other mechanisms, such as checkpointing and roll-back recovery, which provide fault-tolerance at the system level.

III. The Fault-Tolerant *e*-Cube Algorithm

The non-adaptive *e*-cube algorithm is the most commonly implemented wormhole routing algorithm in the recent parallel computers. Though it can achieve good network utilization for uniform traffic [10], it cannot handle faults due to its nonadaptive nature.

For fault-free networks, *e*-cube requires one virtual channel (per physical channel) for deadlock-free wormhole routing. Our fault-tolerant variant of the *e*-cube, called *f*-cube, uses one extra virtual channel for 2D meshes to handle nonoverlapping fault rings. The *f*-cube requires three extra virtual channels (four channels overall), however, to handle cases where f-rings and f-chains may occur and overlap with one another. First, we describe the *f*-cube in the context of two-dimensional meshes. The results for higher dimensional meshes are based on the results for 2D meshes and are given later. We first present the simpler case in which only nonoverlapping f-rings exist in the network.

A. Routing Messages on Nonoverlapping f-Rings

In the following treatment, *row hops* correspond to hops in dimension d_0, and *column hops* correspond to dimension d_1. Row messages that travel from West to East (respectively, East to West) are WE (respectively, EW) messages. NS messages and SN messages are column messages that travel from North to South and South to North, respectively. In the fault-free *e*-cube, a message travels in a row until it is in the same column as the destination and then takes column hops.

Definition 1 (E-cube hop). *At any given time, the path specified by* e-cube *from the current host to the destination of the message is called its* e-cube path; *the first hop in that path is its* e-cube hop *from the current host node.*

Definition 2 (Message type). *A message that has one or more row hops remaining is called a row message. A message that needs to travel only in a column to reach its destination is called a column message.*

A row message may eventually take column hops, but before doing that it changes itself into a column message. A column message never changes its type in *e*-cube routing.

The *f*-cube routing algorithm uses two virtual channels, c_0 and c_1 classes, to provide fault-tolerant deadlock-free routing with *e*-cube algorithm as the base algorithm. We call this algorithm two-channel *f*-cube, or *f*-cube2, to distinguish it from the final version of the *f*-cube, which uses four virtual channels and allows overlapping f-rings and f-chains.

A.1. The f-Cube2 Algorithm

The f-cube2 routing is governed by the set of rules given in Fig. 3. Each node applies this algorithm on each message passing through it. It is assumed that each node, knows the status of the links incident on it, and its position in the f-ring if any of its links is faulty.

Let M denote a message in the network. If the current host of M is its destination, then M is consumed. Otherwise, procedure `Route-Message` of Fig. 3 is used to determine the next hop for the message. For this purpose, we use a message status parameter which can be normal or misrouted. The criteria for setting the status of a message is given by procedure `Set-Status` in Fig. 3. The status parameter of a message indicates whether the message is blocked by a fault and needs to travel on the corresponding f-ring to reach its destination.

Each time a message's label changes from normal to misrouted, its direction along the f-ring is set using the procedure `Set-Direction` (Fig. 3); once a message's direction is set for the current f-ring, it stays the same throughout its journey around that f-ring (Step 1 of `Set-Direction`). The direction of a misrouted message is reset to null when it becomes a normal message. A misrouted NS (respectively, SN) message's direction is set to clockwise (respectively, counter-clockwise) regardless of its current host, and is routed on an f-ring in the clockwise (respectively, counter-clockwise) direction only (Steps 2 and 3 in `Set-Direction`). For an EW message, the direction is set to counter-clockwise (respectively, clockwise), if the destination is in a row above (respectively, below) the current host; if the current host and destination are in the

```
Procedure Set-Status($M$)

If the next e-cube hop is not blocked by a fault,
    then set the status of $M$ to **normal**
    and the direction of $M$ to *null*.
Otherwise, set the status of $M$ to **misrouted**
    and Set-Direction($M$).
```

```
Procedure Set-Direction($M$)
/* Comment: The current host of $M$ is $(a_1, a_0)$
and its destination $(b_1, b_0)$. */
1 If direction of $M$ is not null, return.
    /* $M$ is misrouted and had its direction set */
2 If $M$ is NS, set direction to *clockwise*.
3 If $M$ is SN, set direction to *counter-clockwise*.
4 If $M$ is EW, set direction to
    4.1 *clockwise* if $(a_1 < b_1)$,
    4.2 *counter-clockwise* if $(a_1 > b_1)$, or
    4.3 either orientation if $(a_1 = b_1)$.
5 If $M$ is WE, set direction to
    5.1 *clockwise* if $(a_1 > b_1)$,
    5.2 *counter-clockwise* if $(a_1 < b_1)$, or
    5.3 either orientation if $(a_1 = b_1)$.
```

```
Procedure Route-Message($M$)
/* Comment: The current host of $M$ is $(a_1, a_0)$
and its destination $(b_1, b_0)$. Row messages use $c_0$
virtual channels and column messages use $c_1$ vir-
tual channels. */
0 If $a_1 = b_1$ and $a_0 = b_0$, consume $M$ and return.
1 If $M$ is a row (EW or WE) message and $a_0 =$
    $b_0$,
    change its type to NS if $a_1 < b_1$ or SN if
    $a_1 > b_1$.
2 Set-Status($M$).
3 If $M$ is normal, use the e-cube hop.
4 Otherwise, route $M$ on the fault-ring in the
    specified direction.
```

Fig. 3. Pseudocode of the *f*-cube2 algorithm.

same row, clockwise or counter-clockwise direction is chosen randomly (Step 4). A similar rule is used for WE messages (Step 5).

At each intermediate node, a message M is routed using the procedure `Route-Message`(M). The use of channels of an f-ring by row and column messages is illustrated in Fig. 4.

EXAMPLE. Let us consider the message WE from (1, 0) to (4, 4) in a 6 × 6 mesh with fault set {(1, 2), <(3, 4), (4, 4)>} (see Fig. 5)..Its normal *e*-cube path is

$$(1, 0) \to (1, 1) \to (1, 2) \to (1, 3) \to (1, 4) \to (2, 4) \to (3, 4) \to (4, 4).$$

M is normal at (1, 0). However, it becomes misrouted at (1, 1) since its *e*-cube hop from (1, 1) to (1, 2) is blocked by the fault (1, 2). Since the destination (4, 4) is in a row below (1, 1), M's direction is set to counter-clockwise; M travels from (1, 1) to (2, 1) as a misrouted message. At (2, 1), M's *e*-cube path (from (2, 1) to (4, 4)) is not blocked by any fault and hence M becomes normal again. M travels as a normal message from (2, 1) to (2, 4). At (2, 4), M becomes a NS message, since it only needs hops in the North-South direction. At (3, 4), M is blocked by the faulty link <(3, 4), (4, 4)>, which forces M to become misrouted; hence, M travels in the clockwise direction from (3, 4) to (4, 5). At (4, 5) M becomes normal and travels to (4, 4). M reserves channels c_0 until (2, 4) (since it is a WE message) and channel c_1 from (2, 4) onwards (since it becomes NS at (2, 4)).

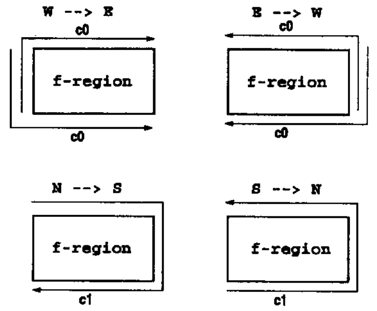


Fig. 4. Usage of virtual channels in f-rings.

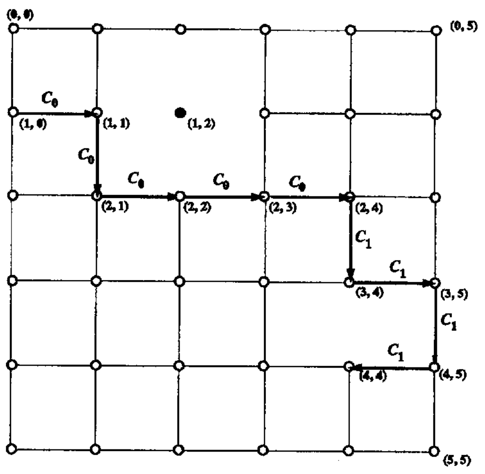


Fig. 5. Example of *f*-cube2 routing.

A.2. Proof of Deadlock-Free Routing on Nonoverlapping f-Rings

We now prove that *f*-cube2 provides deadlock free routing of messages in 2D meshes with nonoverlapping f-rings.

LEMMA 1. *The two-channel* f-*cube2 algorithm provides correct and livelock- and deadlock-free routing in 2D meshes with any number of nonoverlapping f-rings.*

PROOF. For a deadlock to occur, there has to be a cyclic dependency on the virtual channels acquired by the messages involved in the deadlock. Row messages may turn into column

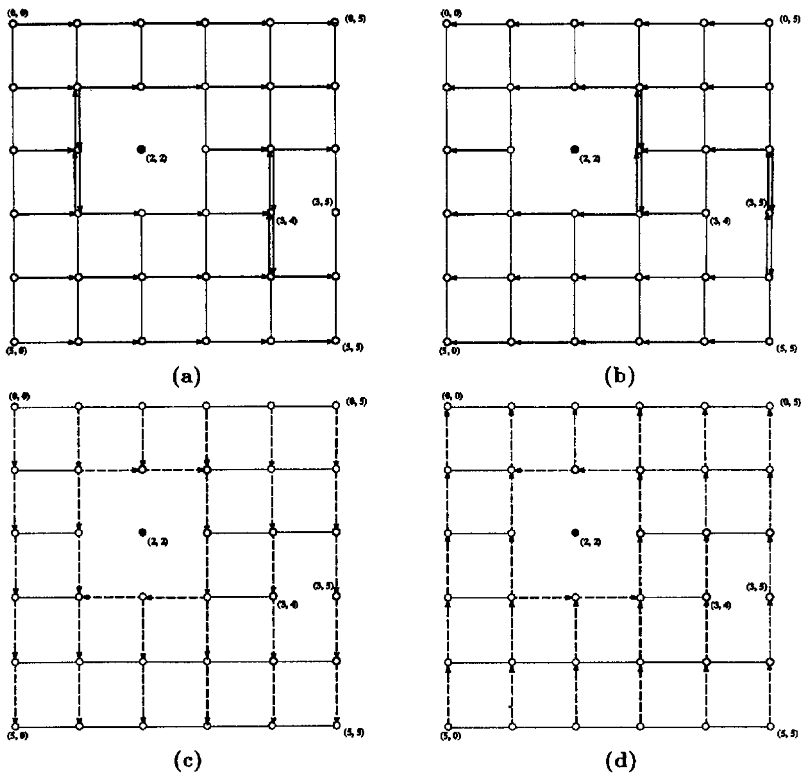


Fig. 6. Example of acyclic directed graphs used by *f*-cube2 in a 6 × 6 mesh with one node and one link fault. Node (2, 2) and link <(3, 4), (3, 5)> are faulty. Vitual channels of class 0 are shown as solid directed edges, virtual channels of class 1 as dashed directed edges, and unused channels for a message type as undirected edges. The channel graph in part (a) is used by West-to-East messages, part (b) by East-to-West messages, part (c) by North-to-South messasges, and part (d) by South-to-North messages.

messages after a few hops, but column messages never turn into row messages. Since row messages use only class 0 virtual channels and column messages use only class 1 virtual channels, there cannot be a deadlock involving both classes of virtual channels. Conceptually, the network may be considered a union of two planes, plane 0 with virtual channels of class 0, and plane 1 with virtual channels of class 1. A message may move from plane 0 to plane 1 but never in the opposite direction. Therefore, if there is a deadlock, then it is among the channels of class 0 or class 1 only.

Class 0 channels are used by two types of row messages: messages going from West to East (WE) and those going from East to West (EW). The WE messages use virtual channels of class 0 only on West to East physical channels, and virtual channels of class 0 on West columns of the f-rings in the network. The EW messages use virtual channels of class 0 only on East to West physical channels, and virtual channels of class 0 on East columns of the f-rings in the network. Therefore, they use disjoint sets of physical channels, and there cannot be deadlocks among row messages.

There are two types of column messages: North-to-South (NS), and South-to-North (SN). Each type uses two disjoint sets of physical channels, since they are routed in opposite directions on f-rings. Therefore, there cannot be deadlocks among column messages.

(An example of the acyclic directed networks of virtual channels used by the four types of messages is given in Fig. 6.)

To see that *f*-cube2 correctly routes messages without introducing livelocks in the faulty network observe that a) a message is misrouted only around an f-ring, b) a message, once it leaves an f-ring, will never revisit it, c) there are a finite number of f-rings in the mesh, d) a normal message progresses towards its destination with each hop, and e) the destination node is accessible, since all nonfaulty nodes are connected. Since a message is misrouted only by a finite number of hops on each f-ring and it never visits an f-ring twice, the extent of misrouting is limited. This together with the fact that each normal hop takes a message closer to the destination proves that messages are correctly delivered and livelocks do not occur. □

One source of performance loss with the *f*-cube2 is the unbalanced use of channels on f-rings by misrouted column messages, which never use channels in the West columns of f-rings. This can be avoided by allowing column messages to choose the orientation such that the paths traversed on f-rings are shortest paths. This effectively partitions the links of each f-ring into two groups. Because of the shortest-path constraint, each column message uses c_1 channels on the links from only one of these groups for its traversal on an f-ring; therefore, the routing is still deadlock free. However, since we assume only local knowledge of faults, routing along a shortest path on the f-ring may not be feasible except in one often-used case: isolated node and link faults. Therefore, for isolated node and link faults, *f*-cube2 can use either orientation to route blocked column messages on f-rings without creating deadlocks.

B. Routing Messages on Overlapping f-Rings and f-Chains

When two f-rings overlap along a column (respectively, row), some nodes in that column belong to the West (respectively, North) boundary of one f-ring and to the East (respectively, South) boundary of another f-ring. The proof of Lemma 1 is based on the fact that EW messages only use the East boundaries of f-rings and WE messages use only the West boundaries of f-rings. However, when f-rings overlap, this condition is no longer met, and the sets of physical channels used by WE and EW messages are no longer disjoint. Since both use class 0 virtual channels, additional constraints should be placed on how row messages are routed in overlapping f-rings to avoid deadlocks.

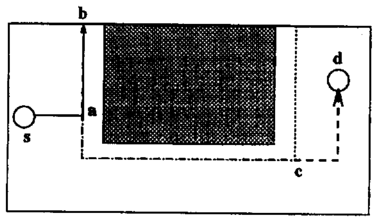


Fig. 7. Example of routing on an f-chain. The shaded area indicates an f-region and dotted lines indicate the corresponding f-chain. The path of the message after u-turn is indicated by a dashed line. The u-turn path overlaps with the path taken on the f-chain from a to b before the u-turn.

Now, consider the issue of routing messages on f-chains. Since we assume only local knowledge of faults, a message may be routed by *f*-cube2 in the direction of the f-chain that actually leads to a dead end. As an example, consider the routing of a WE message, M, from s to d on the f-chain in Fig. 7. When M touches the f-chain at node a, it is routed in NE orientation, since d is in a row above a. As a result, M reaches node b at one end of the f-chain; it needs to take a *u-turn* and travel on the f-chain in the opposite direction to reach the other side of the f-region. Finally when it reaches node c on the f-chain, the SE corner node, the message leaves the f-chain and completes its journey using the *e*-cube algorithm. A similar scenario may be constructed for EW messages. It is noteworthy that column messages are not blocked by this type of f-region; only normal column messages travel on this f-chain. When an f-region touches more than one boundary of the network, even fewer types of misrouted messages encounter the f-chain. For example, the only misrouted messages that travel on an f-chain of an f-region touching the North and East boundaries of the network are WE misrouted messages.

Dependencies that arise when f-rings and f-chains overlap cause severe problems for deadlock free routing. Therefore, to route messages on f-rings and f-chains in 2D meshes, we use four virtual channels, one each for WE, EW, NS, and SN message types. Let c_0, c_1, c_2, c_3 denote the virtual channel classes. The modified routing algorithm is the four-channel *f*-cube or *f*-cube4. The routing logic of *f*-cube4 are given in Fig. 8.

EXAMPLE. An example of *f*-cube4 routing is illustrated in Fig. 9. The network has two overlapping f-rings and an f-chain. The paths taken by two messages as per *f*-cube4 are shown in this figure. The first message is from node (3, 0) to (3, 4). It starts as a normal message and moves to (3, 1) at which point it becomes a misrouted message. The message chooses North orientation, and becomes normal at (1, 1) after two hops. After three more hops—that is, at node (1, 4)—it becomes a column message. As per the routing algorithm, it attempts to reach (3, 4) via (1, 5) but hits a dead end. The message takes a u-turn at this point, and reaches (3, 4). The message takes c_0 channels up to node (1, 4), and c_2 channels for the rest of the journey. The second message is from (4, 3) to (1, 0); it uses c_1 up to (3, 0) and c_3 for the next two steps.

LEMMA 2. *The four-channel* f*-cube4 algorithm provides correct and livelock- and deadlock-free routing of messages in 2D meshes with any number of* f*-regions.*

PROOF. First, we note that each type of message uses a disjoint set of virtual channels. The dependencies across classes of virtual channels are as follows. There are dependencies from c_0 virtual channels to c_2 and c_3, and from c_1 to c_2 and c_3. Note that c_0 virtual channels do not depend on c_1 channels and vice versa. Similarly, c_2 virtual channels do not depend on c_3 channels and vice versa. We also note that c_2 and c_3 never depend on c_0 or c_1 classes of virtual channels. Since, there cannot be cycles involving two or more classes of channels, it is sufficient to prove that there cannot be deadlocks due to dependencies of channels of one particular class only. We show this for WE messages, but similar arguments can be constructed for other types of messages.

First, no WE message travels in the East-to-West direction of the mesh (even the WE messages that take a u-turn can do so only along a column). Thus, there cannot be a deadlock between WE messages waiting in distinct columns. Thus, if there are cyclic dependencies among WE messages, they can only be for channels in the same column. However, for deadlocks to occur along a column, the following condition must be satisfied.

Condition 1. There must be two WE messages such that the first (respectively, second) reserved a North-to-South (respectively, South-to-North) channel and is waiting for a South-to-North (respectively, North-to-South) channel.

WE messages traveling on overlapping f-rings do not take any u-turns, since the orientation of the message alternates

Virtual channel usage

In all cases, WE messages use c_0 virtual channels, EW messages c_1 virtual channels, NS messages c_2 virtual channels, and SN messages c_3 virtual channels.

Routing rule for normal messages

All messages are routed as per the e-cube algorithm.

Routing rule for misrouted messages

If the current node is not an end node of the f-chain, then it is routed using the *f*-cube2 algorithm (see Section 3.1) with the following exception in choosing the direction of a column message:

If a blocked column message has not taken any row hops on the current f-ring prior to its blocking, then it may be routed in either orientation. Otherwise, it is routed using the orientation such that the current direction of travel on the row is continued.

Fault-chains. If a misrouted message hits one of the ends of an f-chain, and if its e-cube path takes the message to its previous host or goes through the f-region, then its direction is reversed (from clockwise to counter clockwise and vice-versa) and routed as per *f*-cube2 routing rules.

Fig. 8. Routing logic of the *f*-cube4 algorithm.

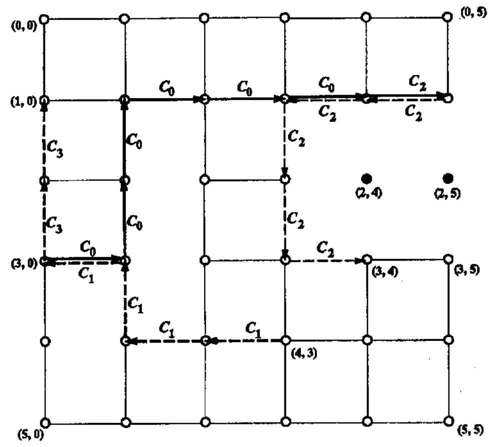


Fig. 9. Illustration of *f*-cube4 routing on f-rings and f-chains.

on successive f-rings; hence, messages traveling on overlapping f-rings cannot satisfy condition 1. A WE message can take hops in both North-to-South and South-to-North directions of a column, only if it was on an f-chain and took a u-turn at an end node of the f-chain in that column. However, Condition 1 requires two such messages in the same column, which is possible only if the West boundary of the f-chain spans a complete column, that is, only if a complete column of the mesh is faulty. If this occurs, the mesh is disconnected, which is a contradiction.

The proof of correct delivery and livelock-freedom is similar to the one given in Lemma 1. □

IV. Adaptive Fault-Tolerant Routing

We next present techniques using which any fully-adaptive routing algorithm can tolerate multiple rectangular fault regions in a 2D mesh. We distinguish between two distinct kinds of fully-adaptive algorithms: *strongly* adaptive algorithms and *weakly* adaptive algorithms. With a strongly adaptive algorithm, a blocked message can wait, indefinitely, for any of the *legal* links at an intermediate node; a link is termed legal if it takes the message closer to its destination. With a weakly adaptive algorithm, a blocked message can wait, indefinitely, for only a strict subset of the legal links at an intermediate node. Weakly adaptive algorithms are based on the result by Duato [13]. Most fully-adaptive algorithms proposed in literature are strongly adaptive. In this section, we concentrate mainly on the strongly adaptive routing algorithms. Henceforth, adaptive, fully-adaptive, and strongly adaptive are used synonymously.

At an intermediate node x, a message bound to node d is said to be *affected* by faults if there is no nonfaulty link connecting y such that the hop from x to y is along the shortest path from x to d. If a message becomes affected at node x, then it remains an *affected* message for the remainder of its journey. A message not affected by faults is said to be *normal.* We use the notion of affected rather than misrouted, since in *f*-cube4 a message may change its status from misrouted to normal and back. Here a message that is blocked by a fault and needs to be routed on the f-ring changes its status to affected permanently.

The following result forms the basis for the adaptive fault-tolerant routing algorithms. It does not hold for weakly adaptive algorithms.

LEMMA 3. *Consider a mesh with multiple f-regions (possibly overlapping f-rings and f-chains) and a fully-adaptive routing algorithm. A message with destination d is affected at some node x if and only if the addresses of x and d differ in exactly one dimension.*

PROOF. Assume that the message M is affected at x and that x and d differ in both dimensions. Under block fault model, a nonfaulty node can have faulty links in at most one dimension. If x has no link faults, then a hop in either dimension will take M closer to its destination. If x has one or both links in a dimension faulty, then one of the links in the other dimension takes M closer to its destination. In all cases, M can move closer to its destination and cannot be affected. This contradicts the hypothesis that M is affected at x. This proves the lemma. □

```
Procedure Fully-Adaptive-2D($M$)
/* Comment: Uses a generic fully-adaptive algorithm $\mathcal{F}$. Affected messages use one of the four additional
virtual channels depending on their types: $c_0$ for $0^+$ messages, $c_1$ for $0^-$ messages, $c_2$ for $1^+$ messages,
and $c_3$ for $1^-$ messages. */
0 If the current host is the destination of $M$, consume it and return.
1 If $M$ is normal and if its next hop as per $\mathcal{F}$ is not blocked by faults, then route $M$ using $\mathcal{F}$ and return.
    Otherwise, set $M$'s status to affected and determine its type and free dimension.
2 If $M$ has a legal hop, use it to route $M$ and return.
3   3.1 If $M$ has not taken any hops on this f-ring as an affected message, then choose either orientation
    for its travel on the f-ring.
    3.2 If the next node on the f-ring in the chosen orientation exists, then route $M$ to that node and
    return.
    3.3 /* The current node is an end node of an f-chain. */
    Reverse the orientation (from clockwise to counter clockwise and vice-versa) of $M$ and route it to
    the next node.
```

Fig. 10. Pseudocode of adaptive fault-tolerant routing in 2D meshes.

An affected message has only one minimal path and, hence, is nonadaptive. The routing technique used by f-cube4 to route column messages can be used to route affected messages in the adaptive case. To show this, we define a few terms.

DEFINITION 3 (Affected and free dimensions). *The dimension in which an affected message needs to travel at the time it is affected is its* affected *dimension, and the other dimension is its free dimension.*

DEFINITION 4 (Types of affected messages). *Let* DIM$_i$ *be the dimension of a message M affected at node x. If address of x in dimi is less than that of its destination, then M is an i^+ message. Otherwise it is an i^- message.*

There are four types of affected messages: 0^+, 0^-, 1^+, and 1^-.

DEFINITION 5 (Legal hop). *If the addresses of the current host and destination of a message match in its free dimension and if the hop that takes the message closer to its destination is on a fault-free channel, then that hop is the* legal *hop of the message at that position.*

If an affected message does not have a legal hop, then it is on an f-ring or f-chain.

The fault-tolerant version of a generic fully-adaptive algorithm $\mathcal{F}$ (to be denoted by $\mathcal{F}_f$) uses four virtual channels—c_0, c_1, c_2 and c_3—in addition to those used by $\mathcal{F}$ and routes messages using Procedure Fully-Adaptive-2D in Fig. 10. Each message is routed by the base adaptive algorithm until the message is affected by faults. The additional logic is used to route affected messages only. The journey of an affected message M to its destination is an alternating sequence of misrouting hops on an f-ring and some (≥ 0) legal hops. When M needs to go through an f-region to progress to its destination, Rule 3 of Fully-Adaptive-2D transports M to the other side of the f-region in a finite number of steps.

EXAMPLE. There are three f-rings and one f-chain in the mesh of Fig. 11. The message from (1, 5) to (1, 4) is affected at node (1, 5), and its type is set to 0^-. This message is sent to North using c_1 to avoid the faulty link <(1, 5), (1, 4)>. Since it has reached a dead end, its orientation is reversed and routed without any further incident to its destination. Another message from (4, 0) to (0, 2) touches two f-ring but slides around the fault because of adaptive routing using the channels provided for adaptive routing. The third message from (2, 4) to (5, 4) is affected at (3, 4) and is routed using virtual channels c_2 (since it is a 1^+-message).

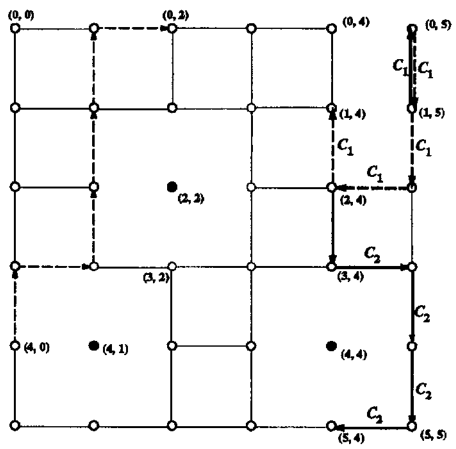


Fig. 11. Illustration of adaptive fault-tolerant routing.

THEOREM 1. *Assume that the fully-adaptive routing algorithm $\mathcal{F}$ correctly routes messages and is deadlock-free for fault-free 2D meshes. The corresponding fault-tolerant fully-adaptive routing algorithm $\mathcal{F}_f$, described in procedure Fully-Adaptive-2D, provides correct and livelock- and deadlock-free routing of messages in the presence of multiple fault blocks.*

PROOF. Algorithm $\mathcal{F}_f$ correctly routes normal messages between any pair of nonfaulty nodes, since $\mathcal{F}$ correctly routes messages. Since the virtual channels used for affected messages are different from those used for normal messages, the statement is true for all normal messages. To complete the proof, we need to show that affected messages are also routed correctly without deadlocks and livelocks.

An affected message is routed without adaptivity in a plane of appropriate class of virtual channels. With the exception of its status, the routing of a message by $\mathcal{F}_f$ is the same as the routing of a column message by the *f*-cube4. Hence, the proof of deadlock and livelock freedom and correct delivery can be constructed using the arguments of Lemma 2. □

A. Extensions

Fault-tolerant routing with two extra virtual channels. Procedure Fully-Adaptive-2D is the adaptive variant of *f*-cube4. It is feasible to devise a fault-tolerant adaptive scheme with only two extra virtual channels if faults are such that only nonoverlapping f-rings are formed. In this case, 0^+ and 0^- messages share virtual channels of class c_0, and 1^+ and 1^- share c_1 virtual channels; c_2 and c_3 virtual channels are unused and need not be provided. To avoid deadlocks, we need to ensure one of the following: a) i^+ messages, $i = 0, 1$, use a orientation different from that used by i^- messages, or b) orientation on each f-ring is chosen such that the hops in the free dimension are minimized. Constraint b) is ensured for isolated node faults. For larger fault blocks, the orientations used on f-rings should be restricted.

Fault-tolerant routing with weakly adaptive algorithms. In general, a weakly adaptive algorithm uses a deadlock-free base algorithm (which can be either nonadaptive or adaptive) and a few extra virtual channels, which can be used by messages for further adaptivity [13]. The advantage of weakly adaptive algorithms is full adaptivity can be provided using just one extra virtual channel. If its base algorithm is *e*-cube, our techniques can be used to enhance the weakly adaptive algorithm for fault-tolerant routing.

As an example, consider the following minimal, weakly adaptive algorithm based on *e*-cube for 2D meshes. Two virtual channels are used: one for *e*-cube algorithm and one for adaptive routing. A message can use adaptive virtual channels in all of its shortest paths to its destination at any time. In addition, a message can use the *e*-cube channel in the path specified by the *e*-cube algorithm. Before we discuss fault-tolerant routing with this algorithm, we show a deadlock situation that can occur with a weakly adaptive algorithm but not with a strongly adaptive algorithm. Fig. 12 indicates one such situation in a mesh with multiple faults. It is noteworthy that each and every message in the deadlock has at least one minimal path to its destination, which could be used by the weakly adaptive algorithm in the absence of other messages. This situation cannot occur with a strongly adaptive algorithm, since a message blocked due to congestion can wait for any channel that would be suitable in the absence of congestion.

A simple method to make this weakly adaptive algorithm fault-tolerant is to enhance the base algorithm, *e*-cube, into *f*-cube4. Thus, five virtual channels are required for the corresponding adaptive, fault-tolerant algorithm. At each hop, if a message obtains a free channel (within a finite period of time), it will be routed by the original weakly adaptive algorithm. If the message is waiting for a channel and its *e*-cube hop is not available because of faults, then *f*-cube4 will be used for the rest of its journey.

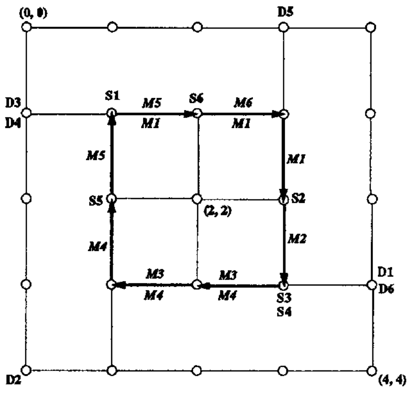


Fig. 12. Deadlock with a weakly adaptive algorithm. There are multiple link faults, which give rise to four f-rings. Faulty links are not shown, undirected lines indicate unused links, and directed lines indicate the physical channels used by the six messages in the deadlock. Message *Mi*, $1 \le i \le 6$, is from node *Si* to node *Di*. Messages using adaptive channels are indicated to the left of directed edges (outside the directed cycle) and messages using *e*-cube channels to the right.

V. Simulation Results

To study the performance issues, we have developed a flit-level simulator. This simulator can be used for wormhole routing in *k*-ary *n*-cubes (meshes and tori) with and without faults. In this section, we present simulation results for the performance of the fault-tolerant *f*-cube2 and a fully-adaptive algorithm for the case of nonoverlapping f-rings. The adaptive algorithm, described in [25], [9], is strongly adaptive and requires 2^{n-1} virtual channels to provide deadlock-free routing in a (k, n)-mesh. Our fault-tolerant version of it is denoted by LH2.

The virtual channels on a physical channel are demand time-multiplexed. Only the virtual channels that have messages to transmit use the physical channel in a round-robin manner [10], [5]. Each virtual channel, upon receiving permission to use the physical channel, transmits the next flit of its message and yields the physical channel to the next virtual channel in the queue. Idle virtual channels do not consume any bandwidth. It takes one cycle to transmit a flit between neighbors.

We have simulated a 16×16 mesh, since radix 16 has been used in many previous studies and in some recent multicomputers such as the Cray T3D [30]. Message length in flits is dependent on the grain of communication and width of links. We have used 20-flit messages, which could be suitable to transmit four 64-bit words together with header, checksum and other information when each link is 2-bytes wide as in Cray T3D. We have used uniform traffic pattern, since it facilitates comparison of results with other works (at least for the fault-free case). Furthermore, there is no consensus on the type and frequency of occurrence of nonuniform traffic that is representative of the traffic in the current parallel computers. Message interarrival times are geometrically distributed.

We use bisection utilization and average message latency as the performance metrics. The bisection utilization (ρ_b) is defined as follows.

$$\rho_b = \frac{(\text{Number of bisection messages delivered / cycle}) * \text{Message length}}{\text{Bisection bandwidth}}$$

The bisection bandwidth is defined as the maximum number of flits that can be transferred across the bisection in a cycle, and is proportional to the number of nonfaulty links in the bisection of the network—for example, the row links connecting nodes in the middle two columns of the 16 × 16 mesh. A message is a bisection message if its source and destination are on the opposite sides of the bisection of the fault-free mesh. The average message latency is the average time from the injection to consumption of a message.

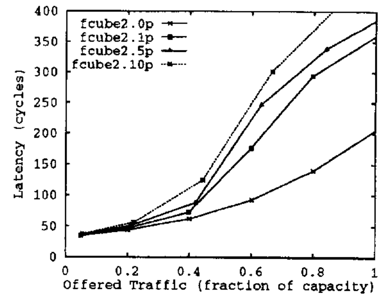
Dally [10] has shown that providing more virtual channels than those necessary for deadlock free routing improves the performance of the *e*-cube considerably. Therefore, we have used eight virtual channels per physical channel. For each required class, one virtual channel is provided. The extra virtual channels are placed in a free pool. If a message is supposed to use a virtual channel of class *v* for a hop and if that virtual channel is busy, then the message takes any idle virtual channel from the free pool, relabels it as virtual channel of class *v*, and uses it. A free pool virtual channel relinquished by a message is returned to the free pool. A message that finds the virtual channel of its class and all virtual channels in the free pool busy simply waits for one cycle and retries. Since the number of virtual channels for any class is at least one at all times, deadlock-free routing is preserved.

For *f*-cube2, two virtual channels on each physical channel are dedicated to implement c_0 and c_1 classes, required for deadlock-free routing, and six virtual channels are placed in the free pool. The fault-free version of the adaptive algorithm requires two classes of virtual channels. We have used the optimized fault-tolerant logic that uses only two extra virtual channels: affected messages in each dimension use virtual channels of a particular class without creating deadlocks (Section IV.A). Therefore, for LH2, four virtual channels are dedicated for deadlock-free fault-tolerant routing and the remaining four virtual channels are placed in the free pool. Furthermore, we allowed either orientation to be used on f-rings for both algorithms, since we simulated only isolated node and link faults.

To facilitate simulations at and beyond the normal saturation points for each routing algorithm, we have limited the injection by each node. This injection limit is independent of the message interarrival time; the motivation for injection control is due to Lam and Reiser [21]. After some experimentation, we set the injection limit to 3 for both algorithms.

A. Performance for Various Fault Cases

We have simulated a 16 × 16 mesh with 1%, 5%, and 10% of the total network links faulty. Specifically, for the 1% case, we have set, randomly, a node and link faulty; since four links are incident on a node, five of the 480 links in the network are faulty. For the 5% fault case, we have set four nodes and eight links faulty; for the 10% fault case, we have set eight nodes and 16 links faulty. In each case, we have randomly generated the required number of faulty nodes and links such that only nonoverlapping f-rings are formed.



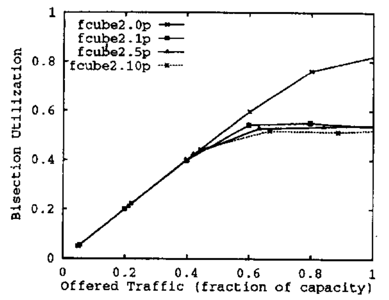
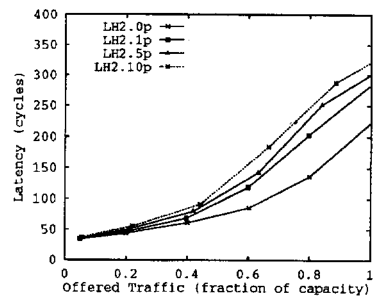


Fig. 13. Performance of the *f*-cube2 algorithm for uniform traffic in a 16 × 16 mesh with various faults. Extension *dp* indicates the *d*%-faults case.



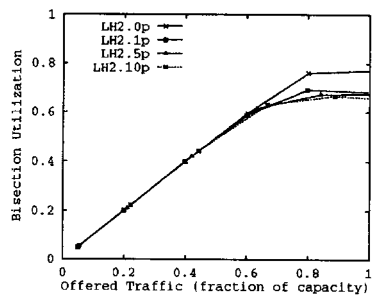


Fig. 14. Performance of the LH2 algorithm for uniform traffic in a 16 × 16 mesh with various faults. Extension *dp* indicates the *d*%-faults case.

The results for *f*-cube2 and LH2 are given in Figs. 13 and 14. The fault sets used for both algorithms are the same for a fault case. For each value reported, the simulation error—half-width of the 95% confidence interval of the value [19]—is less than 5% of the value.

For the fault-free network, *f*-cube2 is very attractive, with peak utilization 82%. Faults affect the performance of *f*-cube2 significantly. There is substantial decrease in utilization and increase in latencies starting at 40% offered traffic load. This can be explained as follows. There is heavy contention for channels on each f-ring, which makes each f-ring a hotspot. Because *f*-cube2 is nonadaptive, messages are unable to avoid f-rings in their row paths. This reduces throughput and increases latency even for normal messages which may be waiting for the channels reserved by misrouted messages prior to misrouting.

The fault-free performance of LH2 is slightly worse than that of *f*-cube2. The interesting point is that LH2 shows a graceful degradation of performance in the presence of faults. Furthermore, the performance of LH2 under faults is comparable to its fault-free performance for up to 60% load. Because LH2 is fully-adaptive, the probability for a normal message to wait for an affected message is small. Therefore, the hotspot effects observed for *f*-cube2 are less severe for the LH2 algorithm.

B. Peak Performance

To evaluate different fault cases more thoroughly, we have further simulated *f*-cube2 and LH2 for 1, 5, and 10 percent faults. For each case, we have simulated 10 different fault sets at the injection rate that would cause 90% load on a fault-free network. For each fault set of each fault case, we have sampled 100,000 delivered messages after the network reached its steady state. The values obtained from the ten different fault sets are averaged and shown in Fig. 15. The vertical bars indicate the corresponding 95% confidence intervals. This graph clearly shows that LH2 degrades more gracefully compared *f*-cube2. For faulty networks, message latencies of LH2 are 18-22% lower compared *f*-cube2. For quantitative comparisons let us set the base utilization as 80% (achieved by *f*-cube2 at 90% load) for both algorithms in the absence of faults. Network utilization with *f*-cube2 is reduced by 21% for 1% faults and by 34% for 10% faults. Thus even a single fault is likely to cause substantial loss of performance for *f*-cube2. The effect of additional faults is not that severe. On the other hand, LH2 is more graceful exhibiting 6% degradation with 1% faults and 20% for 10% faults. (Comparing with its fault-free performance—78% utilization, LH2 gives 4% and 18% lower utilizations for 1% and 10% faults, respectively.) Also LH2 has 18-22% lower latencies compared to *f*-cube2, under faults. Dally and Aoki [11] indicate an 18-19% decrease in utilization with 8% faulty channels, when the dimension-reversal based routing algorithm is used.

VI. Fault-Tolerant Routing in Multidimensional Meshes

In this section, we apply our fault-tolerant routing techniques to multidimensional meshes using the results for 2D meshes as the base cases. Given a (k, n)-mesh, $n > 2$, multiple 2D submeshes (planes) can be formed by removing links connecting neighbors in all but two selected dimensions. A plane such as (0, 1) is formed from links in DIM_0 and DIM_1 and interconnects a set of k^2 nodes whose addresses, in radix notation, differ only in the two least significant components.

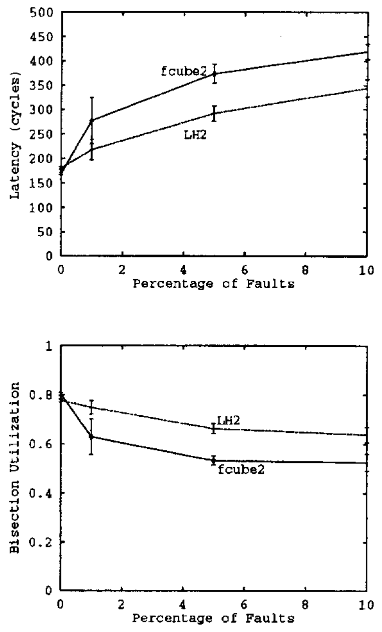


Fig. 15. Peak performance of *f*-cube2 and LH2 algorithms for various fault cases.

Fault model. To define the fault model precisely, we consider n-tuples of k symbols $\{0, \ldots, k-1\}$; we refer to this set as $\mathcal{P}$ and these correspond to nodes in a (k, n)-mesh. We refer to n-tuples of $k+2$ symbols $\{-1, 0, \ldots, k-1, k\}$ as $\mathcal{Q}$. A node $x = (x_{n-1}, \ldots, x_0) \in \mathcal{Q}$ is a *virtual* node, if $x_i = -1, k$ for some i. A link $<x, y>$ between nodes x and y is a *virtual link* iff at least one of x or y is virtual. Every virtual node and link is assumed to be nonfaulty.

A node $x = (x_{n-1}, \ldots, x_0)$ is said to be a *base-node* with respect to another node $y = (y_{n-1}, \ldots, y_0)$ if and only if $x_i \le y_i$ for all i; if x is a base node for y, y is said to be an *apex-node* for x. A block B_{xy} with base-node x and apex-node y contains all the nodes in the set

$$N_{xy} = \{(z_{n-1}, \ldots, z_0) | \, x_i \le z_i \le y_i, \, 0 \le i \le (n-1)\}$$

and every link that connects any two nodes in N_{xy}. A node $z = (z_{n-1}, \ldots, z_0)$ is said to be a *boundary node* of block B_{xy} if $z_i = x_i$ or $z_i = y_i$ for some i. A link $<z, z'>$ is a *boundary link* iff both z and z' are boundary nodes. The interior of B_{xy} contains all nodes and links that are not on the boundary of B_{xy}.

A set F of faulty nodes and links in a (k, n)-mesh is said to be a *faulty-block* iff there exist a base-node $x \in \mathcal{Q}$ and an apex-node $y \in \mathcal{Q}$ such that

TABLE II
FREE DIMENSIONS AND VIRTUAL CHANNELS USED BY AFFECTED MESSAGES IN ADAPTIVE ROUTING IN A 3D MESH

Message Type	Free Dim.	Planes of Routing	Virtual Channels
0^+	DIM_1	$(0,1)$	c_0 in both DIM_0 and DIM_1
0^-	DIM_1	$(0,1)$	c_1 in both DIM_0 and DIM_1
1^+	DIM_2	$(1,2)$	c_2 in both DIM_1 and DIM_2
1^-	DIM_2	$(1,2)$	c_3 in both DIM_1 and DIM_2
2^+	DIM_0	$(2,0)$	c_2 in DIM_0 and c_0 in DIM_2
2^-	DIM_0	$(2,0)$	c_3 in DIM_0 and c_1 in DIM_2

```
Procedure Fully-Adaptive-3D($M$)
/* Uses a generic fully-adaptive algorithm $\mathcal{F}$
and four additional virtual channels $c_0$, $c_1$, $c_2$, $c_3$ */
Routing rules are the same as in Procedure Fully-Adaptive-2D (Figure 11) with one exception: free
dimensions and virtual channels used by affected messages are as given in Table 2.
```

Fig. 16. Routing logic for adaptive fault-tolerant routing in 3D meshes.

1) the interior of B_{xy} contains all and only the components of F,
2) no boundary node or link of B_{xy} is faulty, and
3) no $k \times k$ submesh of the (k, n)-mesh is disconnected by faults, that is, a complete row or a complete column of faulty nodes in a $k \times k$ submesh is not allowed in this fault model.

A set F of faulty nodes and links in a (k, n)-mesh is a valid *block-fault* if F can be written as the union of disjoint subsets $F_1, F_2, \ldots, F_r$ such that each F_i is a faulty-block by itself. The following observation is immediate from the definition of faulty-blocks in nD meshes.

OBSERVATION 1. *Let F be a block fault in (k, n)-mesh and let $F_1, \ldots, F_r$ be the corresponding faulty blocks. If we consider any $k \times k$ submesh H of the (k, n)-mesh, the faulty nodes of F_i, $1 \le i \le r$ form a rectangular fault region in H.*

Observation 1 allows us to route in a (k, n)-mesh under block-faults by routing a message successively in several 2D meshes each with faulty-blocks of rectangular shapes. This concept of routing successively in overlapping planes was used by Chien and Kim [8] in designing the planar adaptive routing (PAR) algorithm. By combining the PAR technique with our algorithm *f*-cube4 for the 2D mesh (see Section III.B), we can design an algorithm that tolerates any combination of faulty blocks in an nD mesh using four virtual channels per physical link. However, we do not present this algorithm, since it is straightforward to combine the PAR technique and the *f*-cube4 algorithm.

A. Fault-Tolerant Fully-Adaptive Routing in nD Meshes

We now show that any strongly fully-adaptive routing algorithm for a (k, n)-mesh can be made fault-tolerant by using four additional virtual channels.

As in the two-dimensional case, we say that a message (with destination d) is *affected* by faults at an intermediate node x if there is no neighbor y such that link $l = \langle x, y \rangle$ is fault-free and the hop on l is along a shortest path from x to d. The following result is immediate from Observation 1 and Lemma 3.

LEMMA 4. *Suppose that a message is being routed from s to d in an n-dimensional mesh using a fully-adaptive routing algorithm. Further, suppose that the mesh has multiple faulty blocks. The message is affected at some node x if and only if the addresses of x and d differ in exactly one dimension.*

A message that needs to travel in DIM_i when it is affected is an i^+ message if its destination address is greater than that of the current host or an i^- message otherwise.

The strategy for routing affected messages is as follows. For each type of affected messages, 1) a suitable free dimension and a set of virtual channels are specified and 2) Procedure Fully-Adaptive-2D (Fig. 11) is used to route them in the plane formed by their affected dimension and the allocated free dimension.

To handle even and odd n, we first show how to route affected messages in 3D meshes. There are six types of affected messages, $0^\pm$, $1^\pm$, $2^\pm$, depending on the final dimension and direction in which they need to travel. The planes and virtual channels used to route affected messages are shown in Table II. The logic for 3D adaptive fault-tolerant routing is shown in Fig. 16.

LEMMA 5. *Let $\mathcal{F}$ be a fully-adaptive algorithm that routes messages correctly without livellocks and deadlocks in 3D meshes. Then the procedure Fully-Adaptive-3D with $\mathcal{F}$ as the base algorithm provides correct and livelock- and deadlock-free routing of messages in 3D meshes with multiple faulty blocks.*

PROOF. Since the fault-tolerant routing logic is used for affected messages only, we need to show that all affected messages are routed correctly and free of livelocks and deadlocks. Take any affected message, say M of type 0^+. The same argument applies with suitable modifications to other message types.

From Observation 1, there is a path from the point at which M is affected to its destination in a (0, 1) plane. Furthermore, all the faults within this plane are rectangular. Since Procedure

```
Procedure Fully-Adaptive-nD($M$)
/* Uses a generic fully-adaptive algorithm $\mathcal{F}$
and four additional virtual channels $c_0$, $c_1$, $c_2$, $c_3$ */
1 Route $M$ using algorithm $\mathcal{F}$ until $M$ either reaches its destination or is affected by faults.
2 If $M$ is affected, determine its type.
3 If ($n$ is odd and $i \in \{0,1,2\}$), route $M$ in the 3D submesh formed by DIM$_0$, DIM$_1$, and DIM$_2$ using
     Fully-Adaptive-3D algorithm.
  Else if (both $i$ $n$ are even or both $i$ and $n$ are odd) route $M$ in a 2D submesh formed by DIM$_i$ and
     DIM$_{i+1}$ using Fully-Adaptive-2D algorithm.
  Else if ($i$ is odd and $n$ is even or $i$ is even and $n$ is odd) route $M$ in a 2D submesh formed by DIM$_i$ and
     DIM$_{i-1}$ using Fully-Adaptive-2D algorithm.
```

Fig. 17. High-level description of adaptive fault-tolerant routing in nD meshes.

Fully-Adaptive-2D(M) is used to route this message in this plane, correct and livelock-free delivery is guaranteed.

From Table II, it can be verified that the sets of virtual channels used by various message types are pairwise disjoint. Therefore, deadlocks, if occur, should be among messages of a particular type. This is not feasible, since Fully-Adaptive-2D provides deadlock free routing for each type of messages when their virtual channels are not shared by other types of messages. □

Fully-Adaptive-2D and Fully-Adaptive-3D algorithms are used to provide adaptive fault-tolerant routing in nD meshes. A high-level description of the nD algorithm is given in Fig. 17. The correctness, deadlock-, and livelock-free properties of Fully-Adaptive-nD procedure follow from the corresponding proofs for Fully-Adaptive-2D and Fully-Adaptive-3D procedures.

Recall that block-fault model defined for (k, n)-meshes assumes that no $k \times k$ submesh is disconnected by faults. The Fully-Adaptive-nD algorithm can tolerate a more general type of block faults, which are not tolerated by nonadaptive fault-tolerant routing methods. If n is even, it is required that 2D planes of type $(2i, 2i + 1)$, $0 \leq i < n/2$, are not disconnected; if n is odd, it is required that 2D planes of type (0, 1), (1, 2), (2, 0) and $(2i - 1, 2i)$, $2 \leq i \leq (n - 1)/2$, are not disconnected.

VII. Concluding Remarks

We have presented techniques to enhance deterministic and fully-adaptive wormhole routing algorithms for fault-tolerant routing on k-ary n-dimensional meshes. We have used the block-fault model in which faulty processors and links are in the form of multiple rectangular regions of the network. The concept of fault-rings and fault-chains are used to route around the fault-regions. Our algorithms are deadlock- and livelock-free and correctly deliver messages between any pair of nonfaulty nodes in a connected component of the network even in the presence of multiple faulty blocks and when some faulty blocks contain boundary nodes and links of the network. The contributions of this paper are e-cube based fault-tolerant routing algorithms and methods to enhance any fully adaptive algorithm using extra virtual channels.

The increase in routing-complexity to achieve fault-tolerant wormhole routing is moderate. If a message is blocked by a fault, it can be detected by checking the fault status of the appropriate link. Based on this the status of a message can be determined easily. The direction of a misrouted message is determined by comparing a 2-tuple (of the current host) with the 2-tuple of the destination in 2D meshes. The status of a message and its direction on an f-ring can be maintained using a few bits in its header. The complexity of these functions scales linearly with the increase in the number of dimensions, because our routing algorithms for n-dimensional meshes, in turn, use the routing techniques developed for 2D meshes. Further, each nonfaulty node can determine its position in an f-ring or f-chain using a distributed algorithm based on exchanging messages with its neighbors.

We have simulated the fault-tolerant versions of the e-cube and a fully-adaptive algorithm. Our simulation results indicate that the deterministic e-cube algorithm suffers from increased channel-contention along f-rings and performs poorly relative to the fault-free case. However, it still achieves 53% utilization with 10% faults. The fully-adaptive LH2 algorithm has better performance under faults.

Table III presents a comparison of the existing work on fault-tolerant wormhole routing and places the results presented in this paper in proper perspective. The number of faults, p_f, that dimension reversal schemes [11] can tolerate depends on the number of virtual channels, r, used. Exact relationship between p_f and r is unknown. For block faults, the dimension reversal methods tolerate up to $r - 1$ blocks with r channels in the worst case. Thus, the virtual channel requirements of these schemes grow linearly with the number of faulty blocks. The planar adaptive routing developed by Chien and Kim [8] also tolerates multiple faulty blocks using only three virtual channels per physical channel; however, when faulty blocks have nodes on the network boundary, their schemes do not work (or work by labeling a large number of nonfaulty nodes as faulty). Our schemes use four virtual channels and tolerate any combination of faulty blocks. Another contribution of this paper is the technique to enhance any fully-adaptive routing algorithm using four additional channels for tolerating any combination of faulty blocks. Our results complement and enhance the existing work in this area (Table III).

TABLE III
A COMPARISON OF THE TECHNIQUES FOR FAULT-TOLERANT WORMHOLE ROUTING

	Glass and Ni [17]	Dally and Aoki [11]	Chien and Kim [8]	This paper
Fault-model	Arbitrary	Arbitrary	Block faults (f-rings only)	Block faults (f-rings and f-chains)
Fault Knowledge	Local	Local	Local	Local
Routing Technique	Negative-first (partially adaptive)	Dimension reversal (adaptive)	Planar-adaptive	E-cube or any fully adaptive (FA)
Virtual channel requirements	1	$r \geq 2$	3	4 for e-cube or 4 extra for any FA
Faults tolerated	Up to $(n-1)$ in nD meshes	Exact number unknown, up to $r-1$ in the worst case	Multiple faulty blocks (f-rings only)	Multiple faulty blocks (f-rings and f-chains)

Our adaptive fault-tolerant routing methods are applicable to strongly adaptive algorithms. However, they can be used to provide fault-tolerant routing with the weakly adaptive algorithms based on Duato's theory [13]. We have shown how to achieve this for an example weakly adaptive algorithm using five virtual channels per physical channel. It may be possible to integrate Duato's adaptive routing techniques and the techniques proposed in this paper to provide adaptive, fault-tolerant routing with four or fewer virtual channels. We are currently working on this problem.

Though we have not considered k-ary n-cubes (n-dimensional tori), our techniques can be extended with suitable modifications to k-ary n-cubes [6]. Performance evaluation of these techniques for various routing algorithms on multi-dimensional tori is a topic for future research.

The concept of fault rings and fault chains can be extended to faults of arbitrary shape. In such cases, the fault rings are not rectangular. Using the concept of fault rings and chains, one arbitrary shaped fault of any size can be tolerated in a 2D mesh. The results presented here appear to be applicable to the case where there are multiple arbitrarily-shaped connected faults and each fault region can be inscribed in a rectangle on the mesh (n-dimensional box on an n-dimensional mesh) without including faults from other connected regions. Further work in this direction is needed.

AKNOWLEDGMENTS

R.V. Boppana's research was partially supported by National Science Foundation Grant CCR-9208784. S. Chalasani's research was partially supported by a grant from the Graduate School of the University of Wisconsin-Madison and National Science Foundation grants CCR-9308966 and ECS-9216308.

REFERENCES

[1] A. Agarwal, D. Chaiken, G. D'Souza, K. Johnson, D. Kranz, J. Kubiatowicz, K. Kurihara, B.-H. Lim, G. Maa, D. Nassbaum, M. Parkin, and D. Yeung, "The MIT Alewife machine: A large-scale distributed-memory multiprocessor," *Proc. Workshop on Scalable Shared Memory Multiprocessors*, Kluwer Academic Publishers, 1991.

[2] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Computers*, vol. 33, no. 4, 1984.

[3] K. Bolding and L. Snyder, "Overview of fault handling for the chaos router," *Proc. 1991 IEEE Int'l Workshop Defect and Fault Tolerance in VLSI Systems*, pp. 124–127, 1991.

[4] R.V. Boppana and S. Chalasani, "A comparison of adaptive wormhole routing algorithms," *Proc. 20th Ann. Int'l Symp. Computer Architecture*, pp. 351–360, May 1993.

[5] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P.S. Tseng, J. Sutton, J. Ubranski, and J. Webb, "iWarp: An integrated solution to high-speed parallel computing," *Proc. Supercomputing '88*, pp. 330–339, 1988.

[6] S. Chalasani and R.V. Boppana, "Fault-tolerant wormhole routing in tori," *Proc. Eighth ACM Int'l Conf. Supercomputing*, July 1994.

[7] M.-S. Chen and K. Shin, "Dept-first search approach for fault-tolerant routing in hypercube multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, pp. 152–159, Apr. 1990.

[8] A.A. Chien and J.H. Kim, "Planar-adaptive routing: Low-cost adaptive networks for multiprocessors," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, pp. 268–277, 1992.

[9] W.J. Dally, "Network and processor architecture for message-driven computers," *VLSI and Parallel Computation*, R. Suaya and G. Birtwislte, eds., ch. 3, pp. 140–222. San Mateo, Calif.: Morgan-Kaufman Publishers, Inc., 1990.

[10] W.J. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 194–205, Mar. 1992.

[11] W.J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 466–475, Apr. 1993.

[12] W.J. Dally and C.L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers*, vol. 36, no. 5, pp. 547–553, 1987.

[13] J. Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 1,320–1,331, Dec. 1993.

[14] S.A. Felperin, L. Gravano, G.D. Pifarré, and J.L. Sanz, "Routing techniques for massively parallel communication," *Proc. IEEE*, vol. 79, no. 4, pp. 488–503, 1991.

[15] P.T. Gaughan and S. Yalamanchili, "Pipelined circuit-switching: A fault-tolerant variant of wormhole routing," *Proc. Fourth IEEE Symp. Parallel and Distributed Processing*, pp. 148–155, 1992.

[16] C.J. Glass and L.M. Ni, "The turn model for adaptive routing," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, pp. 278–287, 1992.

[17] C.J. Glass and L.M. Ni, "Fault-tolerant wormhole routing in meshes," *23rd Ann. Int'l Symp. Fault-Tolerant Computing*, pp. 240–249, 1993.

[18] K.D. Gunther, "Prevention of deadlocks in packet-switched data transport systems," *IEEE Trans. Communications*, vol. 29, pp. 512–524, Apr. 1981.

[19] R. Jain, *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991.

[20] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, 1979.

[21] S.S. Lam and M. Reiser, "Congestion control of store-and-forward networks by input buffer limits—An analysis," *IEEE Trans. Communications*, vol. 27, pp. 127–133, Jan. 1979.

[22] T. Lee and J. Hayes, "A fault-tolerant communication scheme for hypercube computers," *IEEE Trans. Computers*, vol. 41, pp. 1,242–1,256, Oct. 1992.

[23] R. Leveugle, T. Michel, and G. Saucier, "Design of microprocessors with built-in on-line test," *20th Ann. Int'l Symp. Fault-Tolerant Computing*, pp. 450–456, 1990.

[24] S.L. Lillevik, "The Touchstone 30 Gigaflop DELTA prototype," *Sixth Distributed Memory Computing Conf.*, pp. 671–677, 1991.

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

[26] N.F. Maxemchuk and R. Krishnan, "A comparison of linear and mesh topologies—DQDB and the Manhattan Street network," *IEEE J. Selected Areas in Communications*, vol. 11, pp. 1,278–1,289, Oct. 1993.

[27] M.D. Noakes, D.A. Wallach, W.J. Dally, "The J-machine multicomputer: An architectural evaluation," *Proc. 20th Ann. Int'l Symp. Computer Architecture*, pp. 224–235, May 1993.

[28] A.L. Narasimha Reddy and R. Freitas, "Fault tolerance of adaptive routing algorithms in multicomputers," *Proc. Fourth IEEE Symp. Parallel and Distributed Processing*, pp. 156–161, 1992.

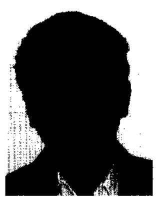
[29] J.Y. Ngai and C.L. Seitz, "A framework for adaptive routing in multicomputer networks," *Proc. First Symp. on Parallel Algorithms and Architectures*, pp. 1–9, 1989.

[30] W. Oed, "The Cray Research massively parallel processor system, CRAY T3D," Technical Report, Cray Research Inc., Nov. 1993.

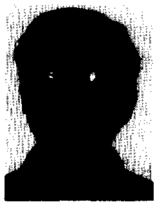
[31] M. Peercy and P. Banerjee, "Distributed algorithms for shortest-path deadlock-free routing and broadcasting in arbitrarily faulty hypercubes," *20th Ann. Int'l Symp. Fault-Tolerant Computing*, pp. 218–225, 1990.

[32] C.S. Raghavendra, P.-J. Yang, and S.-B. Tien, "Free dimensions—an effective approach to achieving fault tolerance in hypercubes," *22nd Ann. Int'l Symp. Fault-Tolerant Computing*, pp. 170–177, 1992.

[33] C. Seitz, "Concurrent architectures," *VLSI and Parallel Computation*, R. Suaya and G. Birtwislte, eds., ch. 1, pp. 1–84, San Mateo, Calif.: Morgan-Kaufman Publishers, Inc., 1990.



Rajendra V. Boppana received the BTech degree in electronics and communications engineering from Mysore University, India, in 1983, the MTech degree in computer technology from the Indian Institute of Technology, Delhi, in 1985, and the PhD degree in computer engineering from the University of Southern California in 1991. Since 1991 he has been a faculty member in computer science at the University of Texas at San Antonio. His research interests are in parallel computer systems, performance evaluation, computer networks, and fault-tolerant computing systems.



Suresh Chalasani received the BTech degree in electronics and communications engineering from J.N.T. University, Hyderabad, India, in 1984, the ME degree in automation from the Indian Institute of Science, Bangalore, in 1986, and the PhD degree in computer engineering from the University of Southern California in 1991. He is currently an assistant professor of electrical and computer engineering at the University of Wisconsin-Madison. His research interests include parallel architectures, parallel algorithms, and fault-tolerant systems.