Communication in Multicomputers with Nonconvex Faults

Suresh Chalasani

Rajendra V. Boppana

Technical Report : CS-96-12

October 1996

The University of Texas at San Antonio
Division of Computer Science
San Antonio, Texas 78249-0667, USA

Communication in Multicomputers with Nonconvex Faults

Suresh Chalasani
Dept. of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706-1691

Phone: (608) 265-2639
Fax: (608) 262-1267
E-mail: suresh@cauchy.ece.wisc.edu.

Rajendra V. Boppana
Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249-0667

Phone: (210) 458-5692
Fax: (210) 458-4437
E-mail: boppana@cs.utsa.edu.

Abstract

A technique to enhance multicomputer routers for fault-tolerant routing with modest increase in routing complexity and resource requirements is described. This method handles solid faults in meshes, which includes all convex faults and many practical nonconvex faults, for example, faults in the shape of L or T. As examples of the proposed method, adaptive and nonadaptive fault-tolerant routing algorithms using four virtual channels per physical channel are described.

Communication in Multicomputers with Nonconvex Faults

Suresh Chalasani   **Rajendra V. Boppana**

1 Introduction

Many recent experimental and commercial multicomputers and multiprocessors use direct-connected networks with mesh topology [1, 14, 13, 7, 15]. These computers use the well-known dimension-order or *e*-cube routing algorithm in conjunction with *wormhole* switching [9] to provide interprocessor communication. In the wormhole technique, a packet is divided into a sequence of fixed-size units of data, called *flits* and transmitted from source to destination in asynchronous pipelined manner. The first flit of the message makes the path and the tail flit releases the path as the message progresses toward its destination.

The *e*-cube routing algorithm is simple and provides high throughput for uniform traffic. The *e*-cube achieves its simplicity by using, always, a fixed path for each source-destination pair, though the underlying network may provide many additional paths of the same length (in hops). Therefore, the *e*-cube cannot handle even simple node or link faults, because even one fault disrupts many *e-cube communication* paths.

Adaptive and fault-tolerant routing for multicomputer networks has been the subject of extensive research in recent years [2, 3, 4, 6, 8, 10, 11, 12, 16]. Most of the current techniques to handle faults in torus and mesh networks require one or more of the following: (a) new routing algorithms with adaptivity [4, 6, 8, 11, 12], (b) global knowledge of faults, (c) restriction on the shapes, locations, and number of faults [6, 8, 12, 3, 16], and (d) relaxing the constraints of guaranteed delivery, deadlock- or livelock-free routing [16].

In this paper, we present fault-tolerant routing methods that can be used to augment the existing fault-intolerant routing algorithms with simple changes to routing logic and with modest increase in resources. These techniques rely on local knowledge of faults—each fault-free node needs to know the status of only its links and its neighbors' links—and can be applied as soon as the faults are detected (provided the faults are of specific shapes). Messages are still delivered correctly without livelocks and deadlocks.

The fault model is a generalized convex fault model, called solid-fault model. In the convex-fault model, each connected set of faults has a convex shape (for example, rectangular in 2D meshes) [3, 6]. In the solid-fault model, a connected fault set is such that any dimensional cross section (defined in Section 2.1) of the fault region has contiguous faulty components. Fault regions with a variety of shapes, for example, convex, +, L, and T in a 2D mesh, are examples of solid faults.

Our approach in this paper is to demonstrate techniques to enhance known fault-intolerant routing algorithms to provide communication even under faults. To illustrate this, we apply our techniques to the non-adaptive *e*-cube and a class of fully-adaptive algorithms [10] for meshes with solid faults. The results presented in this paper expand on our earlier results for convex faults [3, 5].

The rest of the paper is organized as follows. Section 2 describes the solid fault model and the concept of fault-rings. Section 3 describes our fault-tolerance techniques for the nonadaptive *e*-cube

1

algorithm. Section 4 applies these techniques for fully-adaptive algorithms. Section 5 concludes the paper.

2   Preliminaries

We consider n-dimensional mesh networks with faults. A (k, n)-mesh 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. Each node is connected via bidirectional links to two other nodes in each dimension. Given a node $x = (x_{n-1}, \ldots, x_0)$, its neighbors in DIM_i, $0 \leq i < n$, are $(x_{n-1}, \ldots, x_{i+1}, x_i \pm 1, x_{i-1}, \ldots, x_0)$; if the ith digit of a neighbor's index is -1 or k, then that neighbor does not exist for x. If two nodes x and y are connected by a link, then the link is denoted by $x \leftrightarrow y$.

We assume that a message that reaches its destination is consumed in finite time. If a message has not reached its destination and is blocked due to busy channels, then it will continue to hold the channels it has already acquired and not yet released. Therefore, deadlocks can occur because of cyclic dependencies on channels. To avoid deadlocks, multiple logical or virtual channels are simulated on each physical channel and allocated to messages systematically [9]. When faults occur, the dependencies are even more common, and more virtual channels may need to be used or the use of channels may have to be restricted further. Using extra logic and buffers, multiple virtual channels can be simulated on a physical channel in a demand time-multiplexed manner. We specify the number of virtual channels on per physical channel basis and denote the ith virtual channel on a physical channel with c_i.

In the remainder of this section, we describe the fault model and the concept of fault-rings for 2D meshes. Our results can be extended to multidimensional meshes and torus networks with suitable modifications. We label the sides of a 2D mesh as North, South, East and West.

2.1   The fault model

We consider both node and link faults. For fault detection, processors test themselves periodically using a suitable self-test algorithm. In addition, each processor sends and receives status signals from each of its neighbors. A link fault is detected by the processors on which it is incident by examining these status signals. A processor that fails its self-test stops transmitting signals on all of its links, which appears as link faults to its neighbors. A fault-free processor ignores the incoming signals on its links determined to be faulty. So, faulty nodes do not generate messages.

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 should be used for computations in the mean time. We develop fault-tolerant algorithms that can work with only local fault information—each node knows only the status of links incident on it and on its neighbors reachable via its fault-free links.

Before defining solid faults, we need to define cross sections of networks and faults. Each connected fault set describes a subnetwork of the original mesh. Given a subnetwork or network, all of its nodes that match with one another in all but one component of their n-tuple representations and the links among them form its 1-D cross section. For example, in a 2D mesh, each row and each column is a 1-D cross section of the network. The column cross sections of F_3 in Figure 1 are $\{(4,1),(3,1)\leftrightarrow(4,1),(4,1)\leftrightarrow(5,1)\}$ and $\{(3,2),(2,2)\leftrightarrow(3,2),(3,2)\leftrightarrow(4,2)\}$, and its row cross sections are

{(2,2)↔(2,3)}, {(3,2),(3,1)↔(3,2),(3,2)↔(3,3)}, and {(4,1),(4,0)↔(4,1),(4,1)↔(4,2)}. Sometimes a 1-D cross section of a fault contains just a single faulty link. In a similar manner, 2D and higher-dimension cross sections can be defined.

With local knowledge of fault information, handling situations where a message has to travel on a path and retract because it reached a dead-end is complicated. Such situations occur when the node connectivity of a 2D cross section of the network is one. (Node connectivity is the minimum number of nodes that need to be removed to disconnect a network.) This situation can be identified and corrected by disabling the pendant nodes in each 2D cross section. (A pendant node in a 2D plane has only one good link.) A pendant node marks itself faulty. This may result in more pendant nodes in the network. Then the new ones will mark themselves faulty. This can be repeated several times, bounded by the diameter of the network. At the end, we have a connected network with each node having two or more working links in each 2D cross section. Henceforth, we assume that when faults occur appropriate nodes mark themselves faulty as needed so that the resulting node connectivity of the network is greater than one.

A node fault is equivalent to making the links incident on that node faulty. Therefore, given a set F with one or more node faults and some link faults, we can represent the fault information by a set F_l which contains all the links incident on the nodes in F and all the links in F. Two faulty links $a = x \leftrightarrow y$ and $b = u \leftrightarrow v$ in F_l are *adjacent* if one of the following conditions hold:

1. a and b have different dimensions and are incident on a common node,

2. node x can be reached from u in one hop and node y from v in one hop, or

3. node x can be reached from v in one hop and node y from u in one hop.

A pair of links adjacent by the above definition are said to be connected. Two nonadjacent links $a_1, a_p \in F_l$ are connected if there exist links $a_2, \ldots, a_{p-1} \in F_l$ such that a_i and a_{i+1}, for $1 \leq i < p$, are adjacent. A faulty node and a faulty link a are connected if there is at least one link incident on the faulty node to which link a is connected. A set with a single faulty link represents a trivially connected fault set. A set of faulty links F_l with two or more components is connected if every pair of links in F_l is connected. A set F of faulty nodes and links is connected, if the corresponding set F_l of faulty links is connected. The fault sets $F_1 = \{(1,0) \leftrightarrow (1,1), (0,1) \leftrightarrow (1,1)\}$, $F_2 = \{(0,4) \leftrightarrow (0,5), (1,4) \leftrightarrow (1,5)\}$, $F_3 = \{(2,2) \leftrightarrow (2,3), (3,2), (4,1)\}$, and $F_4 = \{(4,4)\}$ in Figure 1 are examples of connected fault sets. F_2 is an example of the connected fault based on the last two adjacency rules given above.

A connected fault-set F, with all of its links given by the set F_l, indicates a *solid* fault region, or *f-region*, if the following condition is satisfied.

If two links $a, b \in F_l$ are in the same 1-D cross section, then all the nodes between a and b in the same 1-D cross section are also faulty.

A set of faults is valid if each connected fault in the set is a solid fault. All the faults in Figure 1 are examples of solid faults. The faults F_2 and F_4 are also examples of convex (rectangular block-shaped) faults. The faults F_1 and F_3 are not convex faults.

2.2 Fault rings

For each connected fault region of the network, it is feasible to connect the fault-free components around the fault to form a ring or chain. This is the fault ring, f-ring, for that fault and consists
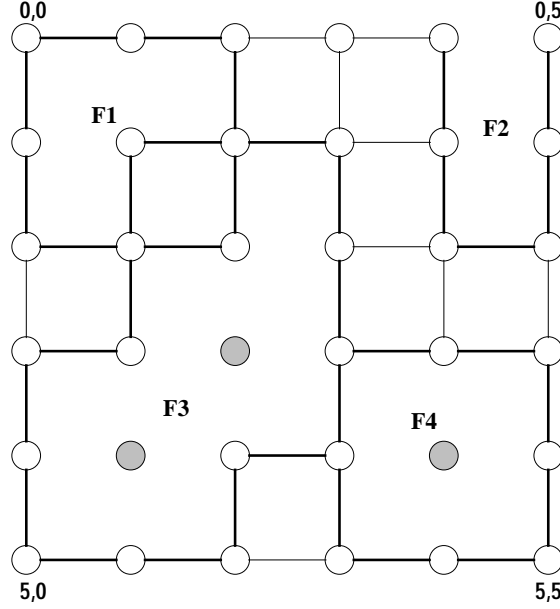
Figure 1: Examples of solid faults in a mesh. Faulty nodes are shown as filled circles, and faulty links are not shown. Thick lines indicate the corresponding fault rings.

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. For example, the f-rings for the various solid faults in Figure 1 are shown with thick lines. It is noteworthy 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. Fault rings provide alternate paths to messages blocked by faults.

A set of fault rings are said to overlap if they share one or more links. For example, the f-rings of F_3 and F_4 in Figure 1 overlap with each other on link $(3,3)\leftrightarrow(4,3)$. Also, if a nonfaulty node has both links in a dimension faulty, overlapping f-rings are formed. (if x is not faulty and has both links in a dimension are faulty, then it results in overlapping f-rings). 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_2 in Figure 1). In this case, a *fault chain*, f-chain, rather than an f-ring is formed around the f-region. In this paper, we do not consider solid faults that form f-chains or overlapping f-rings.

2.3   Formation of fault rings

Fault-rings are used to route messages around a fault region. Messages can be routed on an f-ring in one of two directions: clockwise (immediate boundary of the corresponding fault region is to the right of a message's path on the fault ring), and counter-clockwise. A node on an f-ring identifies its neighbors for each direction of traversal. For f-rings formed around solid faults, the neighbors are the same for both directions. In a general case, however, these neighbors can be different. Fault-rings can be constructed for each fault-region using a two-step distributed algorithm.

To see this, consider a single fault region in a 2D mesh.

In step 1, if a node has a DIM_0 faulty link incident on it, then it sends this information to its DIM_1 neighbors, and vice versa. It is noteworthy that if a node is fault-free and has both links in a dimension faulty, then it results in overlapping f-rings, which are not considered in this paper.

Table 1: Determining the neighbors of a node on an f-ring. Let x be the node whose neighbors are to be determined. N_x, E_x, S_x, W_x denote the nodes adjacent to x in the North, East, South, and West directions, respectively.

Faulty Links	Neighbors
East & South links of x	N_x, W_x
East & North links of x	S_x, W_x
West & South links of x	N_x, E_x
West & North links of x	S_x, E_x
East or West link of x (no faulty column links)	N_x, S_x
North or South link of x (no faulty row links)	E_x, W_x
North link of E_x or East link of N_x	N_x, E_x
South link of E_x or East link of S_x	S_x, E_x
North link of W_x or West link of N_x	N_x, W_x
South link of W_x or West link of S_x	S_x, W_x

In step 2, each node that has a faulty link or received a 'fault information' message from a neighbor determines its position on the f-ring using the rules given in Table 1. The first four cases apply when there are exactly two faulty links are incident on x, the node trying to determine its f-ring neighbors. The next two apply when there is exactly one faulty link incident on x. The remaining cases apply when x has no faulty links.

Even with nonoverlapping f-rings, a node may appear in up to 2 f-rings in a 2D mesh with solid faults. For example, nodes (2,1) and (1,2) appear in the f-rings of F_1 and F_3 in Figure 1. There can be at most two faulty links incident on a fault-free node even with multiple f-regions. If multiple faults occur simultaneously, a node may send or receive messages about multiple f-regions. Using the faulty link direction and dimension provided in each fault status message, it is feasible to separate the messages on faults for different f-regions. For multidimensional meshes, each solid fault creates multiple fault rings, one for each 2D cross section of the fault. In summary, f-rings are formed for any connected fault set using only near-neighbor communication among fault-free processors.

The definition of solid faults can be used to check if a fault can be characterized as a solid fault. In a 2D mesh, the boundary of a solid fault crosses each row and each column exactly zero times or twice. A special type of *shape-finding* message can be circulated around an f-ring and the number of times the worm crosses each row and column can be counted. If any row or column is visited more than twice, then the corresponding fault is not a solid fault; otherwise it is a solid fault. If a fault is not a solid fault, then disabling selected nodes and links so that the result is a solid fault requires more than local knowledge. Some techniques to do this are discussed in Appendix A.

If a fault is a solid fault, the routing techniques described in the remainder of the paper can be used to route messages without any further network reconfiguration.

3   Fault-Tolerant Nonadaptive Routing

We first show how to enhance the well-known *e*-cube routing algorithm to handle solid faults in 2D meshes. The *e*-cube routes a message in a row until the message reaches a node that is in the same

```
Procedure Set-Message-Type(M)
/* Comment: The current host of M is (a_1, a_0) and destination is (b_1, b_0). When a message is
generated, it is labeled as EW if a_0 ≥ b_0 and as WE otherwise. */
  If M is an 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.
```

```
Procedure Set-Message-Status(M)
/* Comment: Determine if the message M is normal or misrouted.
The current host of M is (a_1, a_0) and destination is (b_1, b_0). */

1 If M is a row — EW or WE — message and its e-cube hop is not blocked, then set the status
      of M to normal and return.

2 If M is a column — NS or SN — message and a_0 = b_0, and its next e-cube hop is not on a
      faulty link, then set the status of M to normal and return.

3 Set the status of M to misrouted,
      determine using Table 2 the f-ring orientation to be used by M for its misrouting.
```

Figure 2: Procedures to set the status and type of a message.

column as its destination, and then routes it in the column. For fault-free meshes, the e-cube provides deadlock-free shortest-path routing without requiring multiple virtual channels to be simulated. At each point during the routing of a message, the e-cube specifies the next hop, called e-cube hop, to be taken by the message. The message is said to be blocked by a fault, if its e-cube hop is on a faulty link. The proposed modification uses four virtual channels, c_0, c_1, c_2 and c_3, on each physical channel and tolerates multiple solid faults with nonoverlapping f-rings.

To route messages around f-rings, messages are classified into one of the following types using Procedure Set-Message-Type (Figure 2): EW (East-to-West), WE (West-to-East), NS (North-to-South), or SN (South-to-North). A message is labeled as either an EW or WE message when it is generated, depending on its direction of travel along the row. Once a message completes its row hops, it becomes a NS or a SN message depending on its direction of travel along the column. Thus, EW and WE messages can become NS or SN messages; however, NS and SN messages cannot change their types. These rules are summarized in procedure Set-Message-Type. EW and WE messages are collectively known as *row* messages and NS and SN as *column* messages.

In addition to its type, each message also provides its current status information: normal or misrouted. A row message is termed **normal**, if its e-cube hop is not blocked by a fault. A column message whose head flit is in the same column as its destination is **normal** if its e-cube hop is not blocked by a fault. All other messages are termed **misrouted**. Procedure Set-Message-Status in Figure 2 gives these rules.

3.1 Modifications to the routing logic

Normal messages are routed using the base e-cube algorithm. A normal message blocked by a fault is treated as a misrouted message and routed on the corresponding f-ring using the logic given in Figure 5 *until it becomes normal again.* Sometimes a message may travel on the f-ring before being
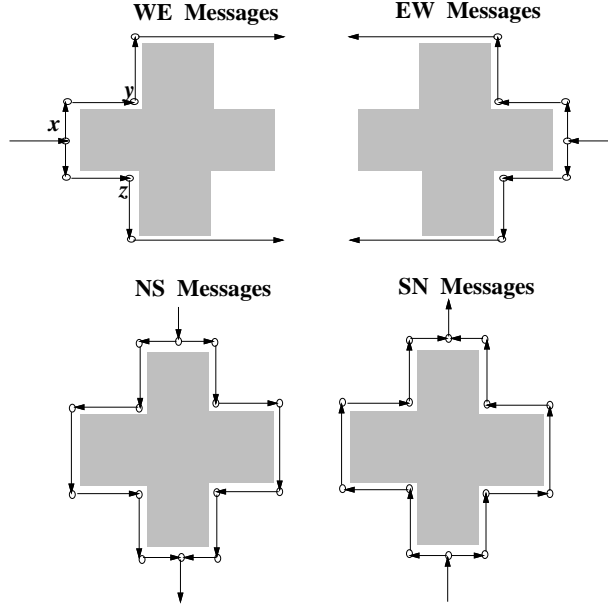
Figure 3: Routing of misrouted messages around fault rings.

blocked by the fault contained by the f-ring. In such cases, the message is forced to use the f-ring orientation compatible with its travel on the f-ring up to that point. For example, a WE message may be blocked at node y in Figure 3 after traversing a hop on the f-ring. In that case, the message should traverse the f-ring in the clockwise orientation (Rule 4 in Table 2) to get around the fault.

In other cases, a message is blocked the first time it arrives at a node on an f-ring (for example, node x for a WE message in Figure 3). In such cases, a message may use clockwise or counter clockwise orientation depending on other conditions. The orientations and conditions are given in Table 2.

If a message takes a normal hop on a link that is not on an f-ring, then the virtual channel to be used is given by the base e-cube algorithm. Under the e-cube, a message may use any virtual channel in its normal hop without deadlocks. (In fact, with e-cube routing, there can be only one type of messages using each physical channel that is neither faulty nor part of an f-ring.) Sometimes a message may travel on an f-ring using the base e-cube algorithm because its normal hop is on the f-ring. In addition, a message may travel on an f-ring because it is blocked by the corresponding fault. In both cases, messages traveling on f-rings can use only the following virtual channels: EW messages use c_0 for all hops on f-rings, WE messages use c_1, NS messages use c_2 and SN messages use c_3.

Consider a message M from (3,0) to (4,5) in the mesh with two solid faults in Figure 4. M begins as a WE message and is routed to (3,1), where its e-cube hop is blocked by the faulty node (3,2). It is misrouted in the counter-clockwise orientation to (4,1) to be compatible with its previous hop, which is on the f-ring. After routed by the base e-cube from (4,1) to (4,4), it is blocked by the faulty link $(4, 4) \leftrightarrow (4, 5)$, and is misrouted in the (randomly chosen) clockwise orientation. M is routed as an NS message for its final hop. The use of virtual channels per the enhanced routing logic is as indicated. The hop from (4,3) to (4,4) is by the e-cube on a link not on any f-ring; so, any of the four classes of virtual channels may be used for this hop as per the e-cube.
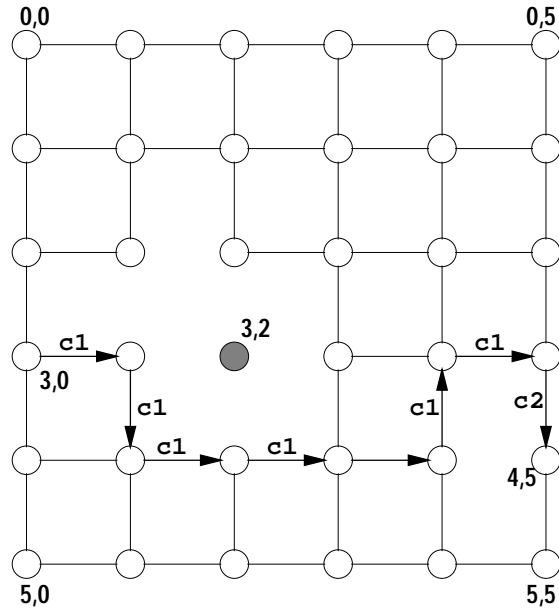
Figure 4: Example of nonadaptive fault tolerant routing.

If a message is destined to a faulty node, then it can be detected and removed from the network using our misrouting logic. A message, say, M, destined to a faulty node will eventually become a column message, say, an NS message, with our misrouting logic. Upon further routing, M will reach a point where it has just completed misrouting by reaching a south row of the f-ring, but its destination is directly above its current host node and its *e*-cube hop is on the faulty North link of its current host node. Upon detecting this anomaly, the message M can be removed from the network.

3.2 Proof of deadlock and livelock freedom

Lemma 1 *The algorithm* Fault-Tolerant-Route *routes messages in 2D meshes with solid faults and nonoverlapping f-rings free of deadlocks and livelocks.*

Procedure Fault-Tolerant-Route(Message M) /* Specifies the next hop of M */
  1. Set-Message-Type(M).
  2. Set-Message-Status(M).
  3. If M is **normal**, select the hop specified by the base algorithm.
  4. If M is **misrouted**, select the hop along its f-ring orientation.
  5. If the selected hop is on an f-ring link, route the message using virtual channel c_0 if M's type is EW, c_1 if WE, c_2 if NS, or c_3 if SN.
  6. If the selected hop is not on an f-ring link, route the message using the virtual channel specified by the base algorithm.

Figure 5: Fault-tolerant routing algorithm.

8

Table 2: Directions to be used for misrouting messages on f-rings.

Rule No.	Message Type	Traversed on the f-ring	Position of Destination	F-Ring Orientation
1a	WE	No	In a row above its row of travel	Clockwise
1b	WE	No	In a row below its row of travel	Counter Clockwise
1c	WE	No	In the same row	Either orientation
2a	EW	No	In a row above its row of travel	Counter Clockwise
2b	EW	No	In a row below its row of travel	Clockwise
2c	EW	No	In the same row	Either orientation
3	NS or SN	No	(don't care)	Either orientation
4	Any message	Yes	Don't care	Choose the orientation that is being used by the message

Proof. Each type of messages (EW, WE, SN, and NS) uses a distinct class of virtual channels. This can be easily seen for the virtual channels simulated on physical channels forming f-rings. For each physical channel not on any f-ring, there can be only one type of message using that physical channel because of *e*-cube routing. Therefore, in all cases, each message type has an exclusive set of virtual channels for its hops. Furthermore, row messages (EW and WE) can become column messages, but not vice-versa. Thus, deadlocks among two different types of messages cannot occur, since NS and SN messages do not depend on any other message type. Hence, to prove deadlock-freedom, it is sufficient to show that there are no deadlocks among messages of a specific type.

Deadlocks among NS messages. Deadlocks can be among NS messages waiting for virtual channels at nodes on a single f-ring only or at nodes on multiple f-rings. (The NS messages waiting for virtual channels at other nodes will be routed by the deadlock-free *e*-cube and cannot be part of deadlocks.) Furthermore, a NS message may use counter clockwise or clockwise orientation to travel on an f-ring. The set of physical channels used for each orientation are disjoint. Misrouted NS messages with clockwise orientation never use the channels on the west-most column of an f-ring. (For example, the link $(2,0) \leftrightarrow (3,0)$ constitutes the west-most column of the f-ring for F_1 in Figure 4.) Similarly, NS messages misrouted counter clockwise on an f-ring never use the east-most column of the f-ring (for example, the two links between nodes (2,3) and (4,3) for the f-ring of F_1 in Figure 4). Therefore, the paths used by NS messages on an f-ring are acyclic. So, a single f-ring does not cause deadlocks among NS messages.

Let us define a relation R on the set of f-rings as follows. Given two f-rings f_1 and $f2$, $f_1 R f_2$ if there exists a DIM$_1$ (column) cross section of the network containing nodes on both f-rings and the top-most node f_1 is above that of f_2.

If $f_1 R f_2$ and $f_2 R f_1$, then (a) both f-rings have the same top node in a DIM$_1$ cross section or (b) there exists two DIM$_1$ cross sections such that in one cross section the top node of f_1 is above that of f_2 and in the other f_2 has its top node above that of f_1. Case (a) is feasible if f_1 overlaps with f_2,

which is prohibited, or $f_1 = f_2$. Case (b) is feasible if the f-region contained by f_2 wraps around (in one or more columns) that of f_1. This is impossible, since faults are solid. So R is antisymmetric.

Suppose $f_1 R f_2$ and $f_2 R f_3$, for three f-rings $f_1, f_2,$ and f_3. If both f_1 and f_3 have a common column, then $f_1 R f_3$, since faults are solid. If f_1 and f_3 do not have a common column, however, then $f_1 \not{R} f_3$ and $f_3 \not{R} f_1$. This shows that R is a subset of some partial-order.

NS messages always use f-rings as per the relation given by R. So, f-rings are used acyclically by NS messages.

Livelock freedom and correct delivery. A message is misrouted only by a finite number of hops on each f-ring, and it never visits an f-ring more than twice (at most once as a row message and once as a column message). So, 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. ∎

3.3 Extension to multidimensional meshes

We now consider solid faults with nonoverlapping f-rings in a (k, n)-mesh and show how to enhance the *e*-cube to provide communication. The *e*-cube orders the dimensions of the network and routes a message in dimension 0, until the current host node and destination match in dimension 0 component of their n-tuples, and then in dimension 1, and so on, until the message reaches its destination. From the definition of solid faults in Section 2.1, it is easy to verify that each 2D cross section (consists of all nodes that match in all but two components of their n-tuples and the links among them) of a solid fault in a (k, n)-mesh is a valid solid fault in a 2D mesh. Therefore, fault-tolerant routing in a (k, n)-mesh is achieved by using our results for 2D meshes and the planar-adaptive routing technique [6].

The routing algorithm to handle nonoverlapping f-rings still needs only four virtual channels per physical channel. Let A_i, where $0 \le i < n$, denote the set of all 2D planes (2D cross sections of the (k, n)-mesh) formed using dimensions i and $i + 1 \pmod{n}$.

A normal message that needs to travel in DIM$_i$, $0 \le i < n$, as per the *e*-cube is a DIM$_i$ message. A DIM$_i$ message that completed its hops in dimension DIM$_i$ becomes a DIM$_j$ message, where $j > i$ is the next dimension of travel as per the *e*-cube algorithm. A message blocked by a fault uses the f-ring in an appropriate 2D plane that contains the current host node for misrouting. So a DIM$_i$ message, $0 \le i \le n - 2$, uses a 2D plane of type A_i for routing and virtual channels of class $c_{2(i \bmod 2)}$ or $c_{2(i \bmod 2)+1}$ depending on its direction of travel in DIM$_i$. A DIM$_{n-1}$ message will use an A_{n-1} plane; it will use virtual channels of classes c_2 or c_3 if n is even, or c_0 or c_1 in DIM$_{n-1}$ and c_2 or c_3 in DIM$_0$, otherwise.

Table 3 indicates the types of planes and virtual channels used in routing various types of messages by the routing algorithm. For $n = 2$, this usage is the same as described before in this section.

There is a partial-order on the planes used and the sets of virtual channels used for these planes are pairwise disjoint. To see this, consider the following ranking of sets of virtual channels. Let n be even and $\nu(i, j)$ denote the set of virtual channels of class j on DIM$_i$ links.

$$\{\nu(0,0), \nu(0,1), \nu(1,0), \nu(1,1)\} \prec \{\nu(1,2), \nu(1,3), \nu(2,2), \nu(2,3)\} \prec$$
$$\{\nu(2,0), \nu(2,1), \nu(3,0), \nu(3,1)\} \prec \{\nu(3,2), \nu(3,3), \nu(4,2), \nu(4,3)\} \prec \dots$$
$$\prec \{\nu(n-1,2), \nu(n-1,3), \nu(0,2), \nu(3,3)\}$$

Table 3: Use of planes and virtual channels by various messages in an nD mesh by the fault-tolerant routing algorithm. First of the two virtual channels specified for each message type is used for hops in the '+' direction of the corresponding dimension and the second virtual channel class for hops in '-' direction.

Message type	Plane type	Virtual channel classes
DIM_0	$A_{0,1}$	c_0 and c_1
DIM_1	$A_{1,2}$	c_2 and c_3
DIM_2	$A_{2,3}$	c_0 and c_1
$\vdots$		
DIM_{n-1}, n even	$A_{n-1,0}$	c_2 and c_3
DIM_{n-1}, n odd	$A_{n-1,0}$	c_0 and c_1 in DIM_{n-1}, c_2 and c_3 in DIM_0

The virtual channels in each set above indicate the channels used by a particular dimension message in a 2D plane. Since the 2D plane routing is based on the 2D mesh routing given earlier, each set of virtual channels are used without creating cyclic dependencies. A rigorous proof of deadlock free routing is straight forward and is omitted.

4 Fault-Tolerant Adaptive Routing

The adaptive fault-tolerant routing algorithm described in this section uses the technique developed in the previous section. The fault intolerant version of the algorithm is based on the general theory developed by Duato [10]. The particular one we use here can provide adaptive routing with as few as two virtual channels: one for deadlock free *e*-cube routing and another for adaptive routing. Since we need four virtual channels for deadlock free routing under faults, we describe the base adaptive algorithm, $\mathcal{A}$, for fault-free networks using four channels (Figure 6). At any point of routing, a message has two types of hops: the *e*-cube hop and adaptive hops. The *e*-cube hop is the same as before: the hop specified by the *e*-cube algorithm. The adaptive hops are all other hops that take the message closer to its destination. Algorithm $\mathcal{A}$ first tries to route a message M using an *adaptive channel*—c_1, c_2, or c_3—along any of the dimensions that take M closer to the destination (Step 2). If this fails, $\mathcal{A}$ tries to route M using the non-adaptive channel c_0 on its *e*-cube hop (Step 3). If this step also fails, the same sequence of events is tried after a delay of one cycle.

To enhance this algorithm for fault-tolerant routing, we classify messages into normal and misrouted categories, as before. While normal messages may have adaptivity, misrouted messages do not. The top level description of our fault-tolerant adaptive routing is the same as that for the non-adaptive case in Figure 5, with the algorithm in Figure 6 as the base routing algorithm. An important amendment to the routing logic is for normal messages: (a) if a message's *e*-cube hop is on an f-ring, then adaptive hops cannot be used; or (b) if message's *e*-cube hop is not on an f-ring, but one or more of its adaptive hops are on f-rings, then those adaptive hops cannot be used. A message with its *e*-cube hop on a link that is faulty or part of an f-ring is routed exactly the same as in the *e*-cube case, because *e*-cube routing is the basis for deadlock freedom in the adaptive routing. The adaptivity is used only when the message does not have to travel on f-rings. Further, whenever a message travels on an f-ring, it reserves virtual channels as specified in Table 3; when a message is not traveling on an f-ring, it uses channel c_0 as the nonadaptive channel, and channels c_1, c_2, c_3 as

```
Procedure Adaptive-Algorithm($M$, $x$, $d$)
/* Current host is $x$ and destination $d$, $d \neq x$. */

1 Determine all the neighbors of $x$ that are along a shortest path from $x$ to $d$. Let $S$ be the set of
    such neighbors.

2 If a virtual channel in the set $\{c_1, c_2, c_3\}$ is available from $x$ to a neighbor $y \in S$, route $M$ from
    $x$ to $y$ using that virtual channel; return.

3 If virtual channel $c_0$ is available from $x$ to a neighbor $z$ along the e-cube hop of $M$, route $M$ from
    $x$ to $z$ using $c_0$; return.

4 Return and try this procedure one cycle later.
```

Figure 6: Pseudocode of the adaptive algorithm enhanced for fault-tolerance.
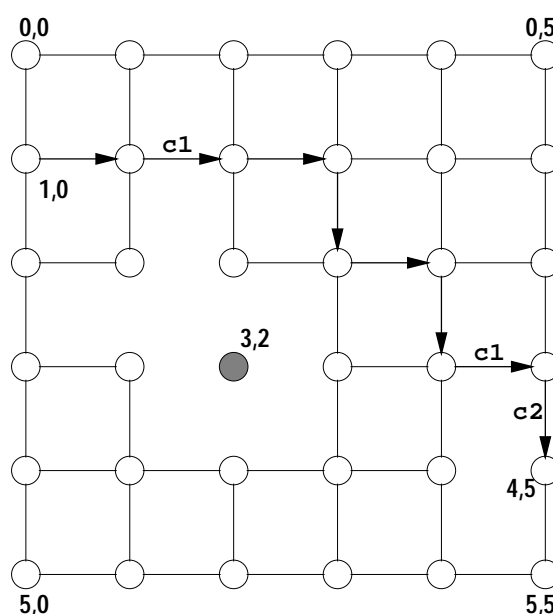

Figure 7: Example of fault-tolerant adaptive routing.

adaptive channels as in the original adaptive routing.

Once again the virtual channel allocation is crucial for deadlock avoidance. In the fault-tolerant adaptive routing, channels c_0 to c_3 are used for messages around f-rings. Virtual channels on links that are not on f-rings are used for normal routing by Adaptive-Algorithm. The four virtual channels on physical channels not on f-rings are partitioned into nonadaptive and adaptive subsets. The channels in the nonadaptive category are c_0 channels, which are used to ensure deadlock free routing. Furthermore, only one type of messages may use the nonadaptive channel on a physical channel not an f-ring. Thus, no virtual channel can be used for normal fault-free routing by one message and for misrouting routing by another message. Therefore, each message type has an exclusive set of virtual channels for deadlock free routing. Given this argument, the proof of deadlock-freedom is similar to that of the nonadaptive case. Figure 7 gives an example of our method. The message uses specific channels for its hops on the f-ring links $(1,1) \leftrightarrow (1,2)$, $(3,4) \leftrightarrow (3,5)$, and $(3,5) \leftrightarrow (4,5)$. For its other hops, the message uses virtual channels as per the original adaptive algorithm.

5   Concluding Remarks

We have presented a technique to enhance the nonadaptive and adaptive algorithms for fault-tolerant wormhole routing in mesh networks. This technique works with local knowledge of faults, handles multiple faults, and guarantees livelock- and deadlock-free routing of all messages. We have used the solid fault model, which generalizes the convex fault model used in previous studies. In the convex fault model, any 2D cross section of the fault has the shape of a rectangle. In the solid fault model, additional fault shapes such as +, T, L, and ⋄ can be handled. Thus, the definition of solid faults includes the convex fault model as a special case. With the current technology, a block of processors can be laid out on a single board. If a board fails or if adjacent boards fail, such failures can be modeled using the convex fault model. Our algorithms can handle any number and combination of convex faults as long as they faulty components do not lie on the boundary of the network. Our algorithms can be extended (using more virtual channels) to handle overlapping f-rings and f-chains in a straight forward manner. Our techniques can thus tolerate a significant number of fault patterns in mesh networks.

The concept of fault-rings is used to route around the fault regions. The main costs of the proposed fault-tolerant routing technique are (a) a special bit in message header to indicate the misrouted status, (b) additional routing logic, which is used by the nodes on f-rings, and (c) additional virtual channels to avoid deadlocks around f-rings. Also, some processing overhead is incurred in forming f-rings.

Our technique extends to related networks such as tori. The number of virtual channels required for tori is doubled, however, because of the wraparound connections. Designing routing algorithms that use more than local information and require fewer virtual channels will be an interesting direction for further work. Currently, we are evaluating the performance of the proposed technique and extending the results to more complex fault shapes and for more general network topologies.

A   Making an Arbitrary Fault into a Solid-Fault Region

If a fault is not a solid fault, it can be converted into one by disabling selected processors and links. We present below simple rules to detect and correct situations where a fault is not solid, is on the boundary of the network, or results in overlapping f-rings. To simplify our discussion, we consider 2D meshes.

Handling pendant nodes.   A pendant node has only one fault-free link incident on it. Such a node marks itself faulty, since it will result in a nonsolid fault otherwise.

Handling faults on network boundaries.   The boundaries of a 2D mesh are top and bottom rows, and left-most and right-most columns.

If a processor has a faulty link on the network boundary, then it sends a special "boundary fault-marker" message on the opposite direction of the faulty link. Soon, all the processors on the boundary where fault occurred will have the information and mark themselves faulty. The 1-D cross section containing the neighbors of the disabled processors forms the new boundary.

When a corner node of the mesh becomes faulty, it triggers this two-step procedure in both dimensions of the network. This results in one row and one column disabled, while it is sufficient

if either the row or column containing the faulty processor is marked faulty. This situation can be detected and rectified by enabling one of the two old boundaries minus the faulty node.

Handling overlapping fault-rings.   Overlapping f-rings can be detected using a simple rule. If a node has a faulty link, say, in dimension 0, and it or its neighbors (reachable in one hop) in dimension 1 have a faulty link in the opposite direction of dimension 0, then overlapping f-rings occur. When this is detected a node can disable itself.

Handling nonsolid faults.   We simply use the characterization of solid faults—all the nodes between two faulty links of a 1-D cross section of the fault are faulty—to remedy this situation. Let us define a corner of an f-ring as the node where a misrouting message changes the dimension of travel. An f-ring has two types of corner nodes: convex and concave corners. The nodes with links in both dimensions faulty form concave corners of an f-ring. The nodes with no links faulty but are on an f-ring form its convex corners.

If a message traverses on the boundary of a fault starting from a concave corner node and the first corner node it encounters is concave, then the entire path traversed is part of a concave fault, or all or some of the path is part of two overlapping f-rings. In either case, all the nodes in entire path can be marked faulty. The actual procedure to achieve this is as follows.

1. Each concave corner node sends a special *fault-marker* message on each of its good links.

2. If a node receives a fault-marker message and has a faulty link incident on it forwards the message to the next node (if exists) in the message's path.

   If a node could not forward a fault-marker message because its link to be used for forwarding the message is faulty, then it removes the message and sends its own fault-marker message (if not sent already) in the opposite direction.

   If a node receives a fault-marker message and has no faulty links incident on it, it removes the message from the network and sends an acknowledgement to the sender of the fault-marker message.

3. If a node receives fault-marker messages from both directions in a dimension, it marks itself faulty. For this purpose, the sender of a fault-marker message is assumed to have received its message and propagated it.

When a fault occurs, it may be result in a pendant node, boundary fault, or overlapping f-rings. So one or more of the above procedures may be triggered in addition to the f-ring formation algorithm in a distributed manner, when a fault occurs. Eventually, the network reaches a stable state where no more nodes are marked faulty, at which point the f-rings formed are suitable for fault-tolerant routing described in this paper.

References

[1] A. Agarwal et al. The MIT Alewife machine: A large-scale distributed-memory multiprocessor. In *Proc. of Workshop on Scalable Shared Memory Multiprocessors.* Kluwer Academic Publishers, 1991.

[2] K. Bolding and L. Snyder. Overview of fault handling for the chaos router. In *Proceedings of the 1991 IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pages 124–127, 1991.

[3] R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Trans. on Computers*, 44(7):848–864, July 1995.

[4] Y. M. Boura and C. R. Das. Fault-tolerant routing in mesh networks. In *Proc. 1995 Int. Conf. on Parallel Processing*, pages I.106–109, Aug. 1995.

[5] S. Chalasani and R. V. Boppana. Adaptive fault-tolerant wormhole routing algorithms with low virtual channel requirements. In *Proc. Int'l Symp. on Parallel Architectures, Algorithms and Networks*, pages 214–221, Dec. 1994.

[6] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. In *Proc. 19th Ann. Int. Symp. on Comput. Arch.*, pages 268–277, 1992.

[7] Cray Research Inc. *Cray T3D System Architecture Overview*, Sept. 1993.

[8] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

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

[10] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(12):1320–1331, Dec. 1993.

[11] P. T. Gaughan and S. Yalamanchili. A family of fault-tolerant routing protocols for direct multiprocessor networks. *IEEE Trans. on Parallel and Distributed Systems*, 6(5):482–497, May 1995.

[12] C. J. Glass and L. M. Ni. Fault-tolerant wormhole routing in meshes. In *Twenty-Third Annual Int. Symp. on Fault-Tolerant Computing*, pages 240–249, 1993.

[13] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[14] M. D. Noakes et al. The J-machine multicomputer: An architectural evaluation. In *Proc. 20th Ann. Int. Symp. on Comput. Arch.*, pages 224–235, May 1993.

[15] C. L. Seitz. Concurrent architectures. In R. Suaya and G. Birtwistle, editors, *VLSI and Parallel Computation*, chapter 1, pages 1–84. Morgan-Kaufman Publishers, Inc., San Mateo, California, 1990.

[16] Y.-J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili. Software based fault-tolerant oblivious routing in pipelined networks. In *Proc. 1995 Int. Conf. on Parallel Processing*, pages I.101–105, Aug. 1995.