Fault-Tolerant Routing with Non-Adaptive Wormhole Algorithms in Mesh Networks[*]

Rajendra V. Boppana
Div. of Math. and Computer Science
The Univ. of Texas at San Antonio
San Antonio, TX 78249-0664
boppana@ringer.cs.utsa.edu

Suresh Chalasani
Electrical & Comp. Engr. Dept.
Univ. of Wisconsin-Madison
Madison, WI 53706-1691
suresh@cauchy.ece.wisc.edu

Abstract. *We present simple techniques to enhance the e-cube algorithm for fault-tolerant routing in mesh networks. These techniques are based on the concept of fault rings, which are formed using fault-free nodes and links around each fault region. We use fault rings to enhance the e-cube to route messages in the presence of rectangular block faults. We show that if fault rings do not overlap with one another—the sets of links in fault rings are pairwise disjoint, then two virtual channels per physical channel are sufficient to make the e-cube tolerant to any number of faulty blocks. For more complex cases such as overlapping fault rings and faults on network boundaries, three or four virtual channels are used. In all cases, the routing guarantees livelock and deadlock free delivery of each and every message injected into the network. Our simulation results for isolated faults indicate that the proposed method provides acceptable performance with as many as 10 percent faulty links.*

Keywords: block faults, deadlocks, fault-tolerant routing, mesh networks, multicomputer networks, nonadaptive routing, wormhole routing.

1 Introduction

Point-to-point k-ary n-cube and related networks are being used in many recent experimental and commercial multicomputers and multiprocessors [1, 19, 18, 22]. 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 [23] has been widely used in the recent multicomputers [19, 18, 22]. 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 [17] and store-and-forward [16], 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 [12]. For networks with faults, a routing algorithm should exhibit the following additional features: graceful performance degradation, and ability to handle faults with only a small increase in routing complexity and local knowledge of faults—each non-faulty 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 non-adaptive.

Description of the problem. In this paper, we address the issue of incorporating fault-tolerance into nonadaptive wormhole routing algorithms such as the e-cube. We develop our techniques for the e-cube algorithm, which has been used in many recent parallel computers [19, 23, 1, 18, 22]. Because of its nonadaptive nature, the e-cube cannot handle faults. Thus, even for a single fault, many pairs of nodes cannot communicate under the e-cube algorithm. In this paper, we present a method to enhance the e-cube for fault-tolerant routing in meshes. The enhanced algorithm, called f-cube, tolerates multiple faults, while retaining the simplicity and livelock- and deadlock-free routing of the e-cube.

We believe that our methods to enhance the e-cube for fault-tolerant routing form the foundation for adaptive fault-tolerant algorithms. Because even with adaptivity, some messages may not have multiple paths to use. For example, with adaptive, minimal (shortest path) routing, messages between two adjacent nodes have no adaptivity. If the link connecting these two nodes is faulty, then the adaptive algorithm cannot route messages between them without violating the shortest-path constraint. The techniques used for the f-cube can be applied to handle such cases [6, 4].

When adaptive, nonminimal routing is used, livelocks may arise. To avoid livelocks, a backup nonadaptive routing algorithm is often used. For example, the dimension

[*]Rajendra Boppana's research has been supported by NSF Grant CCR-9208784. Suresh Chalasani's research has been supported in part by a grant from the Graduate School of UW-Madison and by NSF grants CCR-9308966 & ECS-9216308.

reversal schemes of Dally and Aoki [9] have the e-cube algorithm as the backup algorithm. Our techniques can be used, for example, when messages routed by backup algorithms encounter faults.

We consider routing methods that require only local knowledge of faults. Further, our techniques assume that faulty processors are confined to one or more rectangular blocks. With the current technology and anticipated advances in packaging, 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 accurately models faults at the chip, multichip module, and board levels.

For each fault region, there exist one or more paths that pass through fault-free nodes and links and encircle the fault. For a fault in a 2D mesh, there is an undirected ring of fault-free nodes and links. This is the *fault-ring* for that fault. In this paper, we show that fault-rings can be used to route messages around the fault regions using only local knowledge of faults, and without introducing deadlocks and livelocks. Further, we show, using simulations, that good performance may be achieved even with 10% of the links faulty.

Related results. Routing algorithms for WH and virtual cut-through switching techniques has been the subject of extensive research in recent years [7, 11, 14, 9, 15, 21, 2]. Several results have been reported for fault-tolerant routing in hypercubes. These results exploit the rich interconnection structure of hypercubes and are not suitable for high-radix, low-dimensional meshes.

Reddy and Freitas [20] use global knowledge of faults and routing tables to investigate the performance limitations caused by faults. Gaughan and Yalamanchili [13] use a pipelined circuit-switching (PCS) mechanism with backtracking for fault-tolerant routing. Glass and Ni [15] present a partially-adaptive algorithm, called *negative-first*, that 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 [3], and the number of faults tolerated is too few, for example, one for a 2D mesh. Dally and Aoki [9] have developed the dimension reversal (DR) algorithm to provide adaptive, fault-tolerant routing in meshes. The DR algorithm can tolerate faults of arbitrary shapes, but in the worst case, requires virtual channels proportional to the number of faults tolerated.

Chien and Kim [7] 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 methods yield several routing schemes depending on the positions of faulty blocks. If the faults do not lie on network boundaries and the fault-rings do not overlap—that is, the sets of links in fault-rings are pairwise disjoint— then two-channels are enough for fault-tolerant routing. Three channels are sufficient to handle faults on network boundaries. Four virtual channels are sufficient to handle more complex situations such as overlapping fault-rings and faults on network boundaries.

Organization of the paper. Section 2 describes the fault-model and fault-rings. Section 3 presents the main result: f-cube, the fault-tolerant version of e-cube. Section 4 investigates the performance of the f-cube using simulations of 2D meshes with faults. Section 5 summarizes the results and presents directions for future work.

2 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 $\mathrm{DIM}_0, \ldots, \mathrm{DIM}_{n-1}$, and $N = k^n$ nodes. Each node is uniquely indexed by an n-tuple in radix k. Each node is connected via communication links to two other nodes in each dimension. The neighbors of the node $x = (x_{n-1}, \ldots, x_0)$ in DIM_i, $0 \leq i < n$, 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 wraparound 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 wraparound connections missing. The well-known binary hypercube is the $(2,n)$-mesh. In this paper, we consider (k,n)-mesh networks for small n, large k, and with 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.

A message that reaches its destination is consumed in finite time. Following Dally and Seitz [10], we use 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 [8, 5]. We always specify the number of virtual channels on per physical channel basis.

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 four sides of a 2D mesh as North, South, East and West.

2.1 The fault model

We consider both node and link faults. All the links incident on a faulty node are considered faulty. We assume that failed components simply cease to work and that messages are generated among nonfaulty processors only.

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. We develop fault-tolerant algorithms which assume that each non-faulty processor knows only the status of its neighbors, since maintaining the global knowledge of faults is difficult in a massively parallel system.

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,

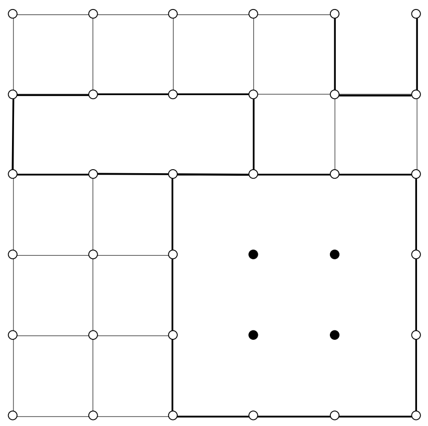

Figure 1: Examples of f-rings and f-chains in a 6×6 mesh. Faulty nodes are shown as filled circles, and faulty links are not shown.

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. Figure 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 Figure 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 [7]. However, their model does not effectively deal with faults on the network boundary. For example, to handle the faulty-link $<(0,4),(0,5)>$ with their fault model, Chien and Kim label all nodes in row 0 faulty. This causes a significant loss of computational power, even when the number of faulty components in the network is small. Our fault model treats such faults as rectangular and handles them without labeling other working components as faulty.

2.2 Fault rings and fault chains

Conceptually, fault regions may be considered as lakes of faults in a terrain of communication channels and nodes. In the same manner an automobile is driven around a lake to reach the other side, 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 links 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 Figure 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), and (3,2) on its boundary. 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. In a (k,n)-mesh, a link may be common to up to $2(n-1)$ 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 Figure 1 overlap with each other, since they share link $<(2,2),(2,3)>$.

When an f-region touches one or more boundaries of the network (e.g., F_3 in Figure 1), a *fault chain*, f-chain, rather than an f-ring is formed. There are four basic types of fault chains that are formed when an f-region touches exactly one of the (2D) network boundaries—top and bottom rows, leftmost and rightmost columns. 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.

An f-ring (respectively, f-chain) represents a two-lane (respectively, one-lane) path to a message that needs to go 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 on fault-rings creates additional possibilities for deadlocks.

When a fault occurs, the corresponding f-ring (or f-chain) can be formed in a distributed manner using a two-step process. In the first step, each processor that detected a faulty link sends this message to its neighbors in other dimensions. Based on the set of messages received, each node determines its position on the f-ring. There are eight possible positions for a processor to be in an f-ring: North West corner, North, North East corner, East, South East corner, South, South West corner, and West. More details are given in [4].

3 The fault-tolerant e-cube algorithm

The e-cube routes messages deadlock free when there are no faults in the mesh network. With multiple virtual channels simulated on each physical channel, the e-cube can achieve impressive network utilizations for uniform traffic. It cannot handle faults, however, due to its nonadaptive nature.

Our fault-tolerant variant of the e-cube, called f-cube, uses two virtual channels to handle nonoverlapping fault rings. The f-cube can handle more complex cases of f-chains and overlapping f-rings and f-chains using additional virtual channels. We first describe the f-cube for two-dimensional (2D) meshes, and then extend them to multidimensional meshes. It is assumed that each fault-free node knows the status of its links and its position on an f-ring if it has faulty links.

3.1 Routing messages in nonoverlapping f-rings

For this simple case, the *f*-cube requires two classes of virtual channels, c_0 and c_1, and is denoted *f*-cube2. Let *row hops* be hops in DIM_0 and *column hops* be hops in DIM_1.

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

In a fault-free network, a message takes row hops until it is in the same column as the destination and then takes column hops.

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 traveling West to East (respectively, East to West) is a WE (respectively, EW) message. Similarly, NS and SN messages are column messages that travel from North to South and South to North, respectively. 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.

3.1.1 The *f*-cube2 algorithm

The routing logic of the *f*-cube2 algorithm is given in Figure 2. Let M denote a message in the network. At each intermediate node, a message M is routed to the next node using the procedure Route-Message(M). 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 Figure 2. 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 status changes from normal to misrouted, its direction along the f-ring is set using the procedure Set-Direction; once a message's direction is set for the current f-ring, it stays the same throughout its journey on that f-ring (Step 1 of Set-Direction in Figure 2). The direction of a misrouted message is reset to null when it becomes normal.

A misrouted NS (respectively, SN) message's direction is set to clockwise (respectively, counter-clockwise) regardless of its current host, and is routed on a f-ring in the clockwise (respectively, counter-clockwise) direction only (Steps 2 and 3 in Figure 2). 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 same row, clockwise or counter-clockwise direction is chosen randomly (Step 4). A similar rule is used for WE messages (Step 5). The use of f-ring channels by row and column messages is illustrated in Figure 3.

Example. As an example, consider the routing of a message from (1,0) to (4,4) in Figure 4. Its normal *e*-cube path is

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

```
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 $c_1$ virtual 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.
```

Figure 2: Pseudocode of the *f*-cube2 algorithm.

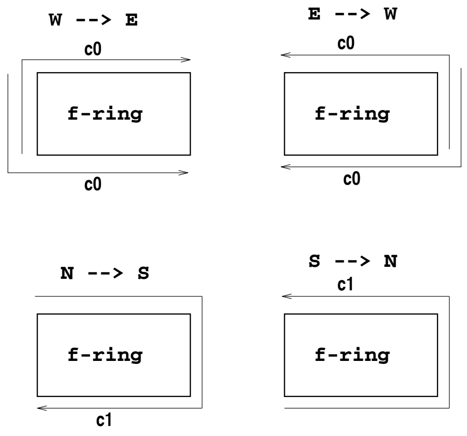

Figure 3: Usage of virtual channels in a f-ring. For each type of misrouted message, the class and sequence of virtual channels used is indicated.

M is normal at $(1,0)$. After the first hop, it is misrouted at $(1,1)$, since its e-cube hop, $(1,1) \rightarrow (1,2)$, is blocked by faulty node $(1,2)$. Because its destination, $(4,4)$, is south of $(1,1)$, M's direction is set to counter-clockwise and routed to $(2,1)$. For the next three hops, $(2,1) \rightarrow (2,4)$, M travels as a normal WE message, since the hops are on the e-cube path from $(2,1)$ to $(4,4)$. At node $(2,4)$, M becomes a NS message, since it only needs to take hops in the South direction. At $(3,4)$, M is blocked by the faulty link $< (3,4),(4,4) >$, which forces M to be misrouted; hence, M travels in the clockwise direction from $(3,4)$ to $(4,5)$. At $(4,5)$, M is normal again and takes its final hop to $(4,4)$. M uses c_0 channels up to $(2,4)$, where it changes into an NS, and c_1 for the remaining hops.

3.1.2 Proof of deadlock-free routing of *f*-cube2

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 algorithms provides correct and livelock- and deadlock-free routing in 2D meshes with any number of nonoverlapping f-rings.*

Proof. For the 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 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 cycle involving both row and column messages. Conceptually, the network may be considered as 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,

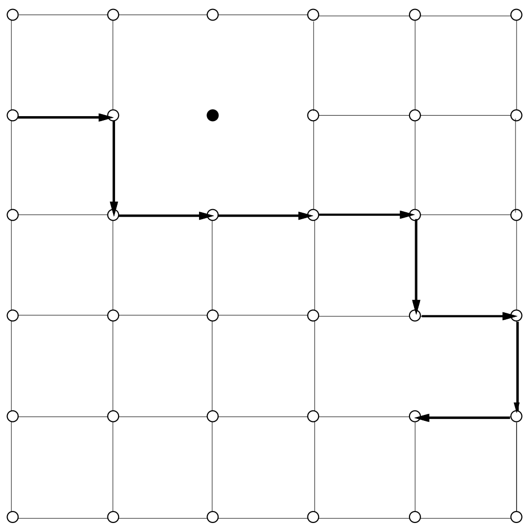

Figure 4: Routing of a message in a mesh with two nonoverlapping f-rings.

and virtual channels of class 0 on West columns of the f-rings in the network. The messages from East to West 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. The sets of physical channels and, hence, the set of virtual channels used by WE and EW are disjoint. (See Figure 5 for an example.) Therefore, there cannot be deadlocks among row messages.

There are two types of column messages: NS and 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.

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. ∎

3.1.3 A more flexible *f*-cube2

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. This has two benefits for column messages: (i)

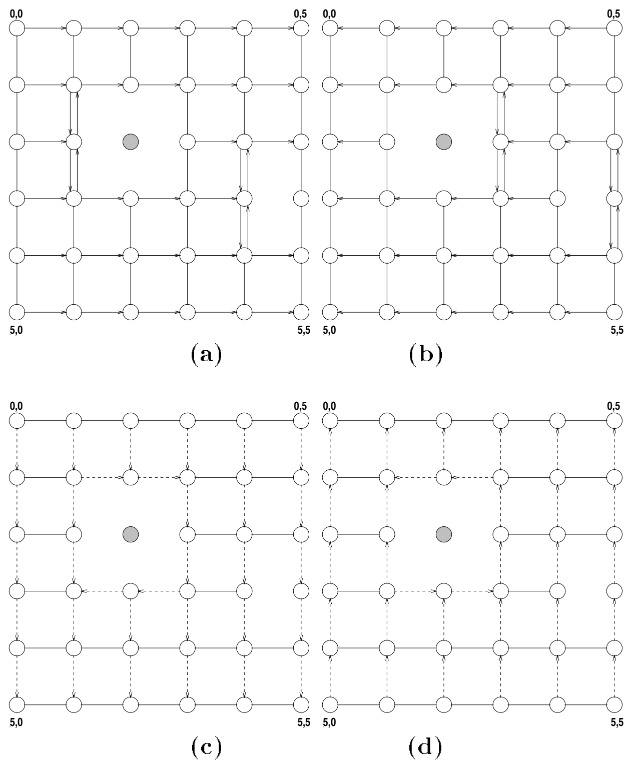

Figure 5: Acyclic directed networks used by the *f*-cube2 algorithm to route the four types of messages in a mesh with faults. The faulty links are not shown and the faulty node is the shaded node. The network in part (a) is used by West-to-East messages, part (b) by East-to-West messages, part (c) by North-to-South messages, and part (d) by South-to-North messages. Solid directed lines indicate virtual channels of class 0, dashed directed lines indicate class 1, and undirected lines indicate unused channels for a message type.

both orientations on f-rings are used, there by making the load on f-ring channels more balanced, and (ii) fewer f-ring channels are used. However, since we assume only local knowledge of faults, routing in shortest paths on f-rings 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.

3.2 Routing messages on f-rings and f-chains

We now extend the *f*-cube2 algorithm to handle faults on network boundaries, which cause f-chains, and overlapping f-rings and f-chains.

3.2.1 Nonoverlapping f-rings and f-chains

Because faults are known only locally, f-chains are treated as f-rings by all but the corresponding end nodes. Therefore, *f*-cube2 may route a message on an f-chain in the direction that actually leads to a dead end. As an example, consider the routing of a WE message, M, from s to d

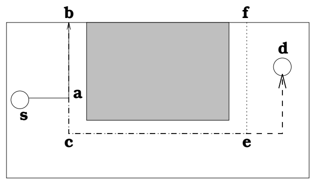

Figure 6: Example of u-turn on an f-chain. The shaded area indicates a f-region and dotted lines the corresponding f-chain. The path of the message after u-turn is indicated by a dashed line. The u-turn path overlaps with the path taken on f-chain from a to b before the u-turn.

on the f-chain in Figure 6. 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, an end node of the f-chain. The *f*-cube2 as described above cannot handle this situation unless the routing logic is modified. A similar scenario can be constructed for EW messages. For column messages f-regions that include nodes or links from the East boundary (rightmost column) of the mesh cause such difficulties. The following modification to the routing logic of *f*-cube2 solves this problem.

> If a misrouted message, M, reaches an end node of an f-chain and if its e-cube hop takes M to its previous host or is on a faulty link (M's destination is still on the other side of the f-region), then the direction of M is reversed (from clockwise to counter clockwise and vice-versa) and is routed as per *f*-cube2 routing rules in Figure 2.

Continuing with the example in Figure 6, the message takes a *u-turn*, as per the above modification, and travel on the f-chain in the opposite direction to reach the other side of the f-region. Finally when it reaches node e on the f-chain, the SE corner node, the message leaves the f-chain and completes its journey using the e-cube algorithm.

The u-turns on f-chains cause additional channel dependencies. Recall that in *f*-cube2, EW and WE messages share c_0 virtual channels and NS and SN share c_1 virtual channels. When the same assignment is used for routing f-chains, deadlocks are avoided in all but one case. For misrouted row messages on all types of f-chains and misrouted column messages on f-chains corresponding to faulty blocks touching leftmost column of the mesh, the sets of virtual channels used are disjoint. The only case that causes deadlocks is misrouted column messages on f-chains with end nodes on the rightmost column of the mesh. On these f-chains, the virtual channels used by NS and SN messages are not disjoint because of u-turns.

To avoid these deadlocks, we use three virtual channels: c_0, c_1, and c_2. WE and EW messages share c_0, NS messages use c_1 exclusively, and SN messages use c_2 exclusively. Since NS (similarly, SN) messages use an exclusive class of virtual channels, they can be allowed to use either orientation—clockwise or counter clockwise—to travel on f-rings and f-chains. There is no restriction that the paths traversed on f-rings and f-chains should be shortest paths.

The f-chains corresponding to f-regions touching more than one boundary of the network block even fewer types of messages and are simpler to handle. For example, only WE messages need to be misrouted on the f-chain of an f-region touching the North and East boundaries of the network.

We call this the *f*-cube3 algorithm. It uses one more virtual channel than *f*-cube2, but handles f-chains and balances traffic on f-rings.

Lemma 2 *The* f-*cube3 provides correct and livelock- and deadlock-free delivery of each and every message injected into a mesh with nonoverlapping f-rings and f-chains.*

Proof sketch. Since the main arguments of the proof are similar to those for the *f*-cube2, we provide a proof sketch only. The main difference between *f*-cube2 and *f*-cube3 is the u-turns sometimes taken by messages on f-chains.

It can be easily verified that the path traversed by a type of messages on an f-chain is acyclic even with u-turns. For example, WE messages on the f-chain in Figure 6 traverse on whole or part of the acyclic path $c \to b \to c \to e$ and EW messages on $e \to f \to e \to c$. Similar observations hold for the u-turns by column messages. Thus the paths of a message type are still acyclic. Following the arguments of *f*-cube2, one can easily show that WE and EW messages use disjoint sets of physical channels and hence, disjoint sets of c_0 channels.

Column messages are more complex than row messages, since they traverse three sides of an f-ring or f-chain. For this reason, we provide an exclusive class of virtual channels for each type of column messages. The use of either orientation on f-rings and f-chains by column messages does not create any new dependencies. Using these facts and the arguments of Lemma 1 one can prove that *f*-cube3 is deadlock free. The issues of livelock-free and correct delivery are proved as in Lemma 1.

3.2.2 Overlapping f-rings and f-chains

When two f-rings overlap along a column (respectively, row), some nodes in that column (respectively, row) 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. Similar observations can be made for NS and SN messages, which share c_1 class channels in *f*-cube2. The *f*-cube can be used in such cases by providing one class of channels for each of four message types. This is the *f*-cube4 algorithm. The *f*-cube4 algorithm handles f-chains using the logic of *f*-cube3. More details are given in [4].

3.3 Routing in multidimensional meshes

The results we have developed for the two-dimensional meshes can be extended to n-dimensional (nD) meshes using the planar-adaptive routing (PAR) technique [7]. We illustrate this for the *f*-cube2 algorithm.

The block-fault model for nD meshes is as follows. An nD box has a base node $x = (x_{n-1}, \ldots, x_0)$ and apex node $y = (y_{n-1}, \ldots, y_0)$ and the set of nodes of the form $t = (t_{n-1}, \ldots, t_0)$ such that $x_i \le t_i \le y_i$, for $0 \le i < n$. If a fault set is contained in an nD box such that the interior of the box has only the faulty components and none on its exterior, then the fault-set represents a block-fault.

Now the extension of the *f*-cube2 to nD meshes follows naturally. The *f*-cube2 still needs only two virtual channels and handles cases where only nonoverlapping f-rings are formed. The key issue is how virtual channels and planes are used to route messages. Following Chien and Kim's notation we let $A_{i,j}$, where $0 \le i < j < n$, denote the set of all 2D planes formed using dimensions i and j. Further, $A_{i,j} = A_{j,i}$. For *f*-cube2, we use only planes in sets $A_{i,j}$, where $0 \le i < n$ and $j = i + 1 \pmod{n}$.

A normal message that needs to travel in DIM_i, $0 \le i < n$, as per the e-cube is an M_i message and is routed in a plane of the type $A_{i,j}, j = i + 1 \pmod{n}$. A M_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 blocked message uses the f-ring in its current 2D plane to get around faults. Table 1 indicates the types of planes and virtual channels used in routing various types of messages by the *f*-cube2. For $n = 2$, it is the same as described in Section 3.1. The proof of deadlock free routing is straight forward and is omitted.

Table 1: Use of planes and virtual channels by various messages in an nD mesh with *f*-cube2 routing.

Message type	Plane type	Virtual channel classes
M_0	$A_{0,1}$	c_0
M_1	$A_{1,2}$	c_1
M_2	$A_{2,3}$	c_0
$\vdots$		
M_{n-1}, n even	$A_{n-1,0}$	c_1
M_{n-1}, n odd	$A_{n-1,0}$	c_0 in DIM_{n-1}, c_1 in DIM_0

4 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 for the case of nonoverlapping f-rings.

We have simulated a 16×16 mesh for the uniform traffic pattern with 20-flit messages. The virtual channels on a physical channel are demand time-multiplexed, and it takes one cycle to transfer a flit on a physical channel. The 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.

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

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.

Recent studies [5, 8] have shown that providing more virtual channels than those necessary for deadlock free routing improves the performance of the e-cube considerably. Therefore, we have experimented with two and four 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. For the simulations with two virtual channels, each of c_0 and c_1 classes are allocated one virtual channel. For simulations with four virtual channels, the extra two are placed in a free pool. Since the number of virtual channels for any class is at least one at all times, deadlock-free routing is preserved.

In all cases, each physical channel is provided with 16 flit buffers. For the two virtual channel case, each virtual channel has buffer depth of 8 flits; that is, up to 8 consecutive flits of a message using a virtual channel can be buffered for that virtual channel. For the four virtual channel case, each virtual channel has buffer depth of 4 flits.

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. After some experimentation, we set the injection limit to 2, which means that a node may inject a new message if fewer than two of its previous messages are still in the node. When there are faults in the network, the injection limit has little effect on the latency and throughput values prior to the saturation.

We have simulated a 16×16 mesh with 1%, 5%, and 10% of the total network links faulty. We have used a mixture of node and link faults. Node faults cause more severe congestions, since a node fault blocks both row and column messages while a link fault blocks only one type of messages. Specifically, for the 1% case, we have set, randomly, a node and link faulty; since 4 links are incident on a node, 5 of the 480 links—10 of the 960 physical channels in the network—in the network are faulty. For the 5% fault case, we have set 4 nodes and 8 links faulty; for the 10% fault case, we have set 8 nodes and 16 links faulty. In each case, we have randomly generated the required number of faulty nodes and links such that isolated faults with nonoverlapping f-rings are formed. Since we have created only isolated faults, we have simulated the more flexible version of *f*-cube2 (described in Section 3.1.3).

4.1 Performance for various fault cases

In this section, we present the performance of the *f*-cube2 for a fixed fault set in each fault case with varying load. The results for two- and four-channel cases are given in Figures 7 and 8. For each value reported in these graphs, the 95% confidence interval is within 10% of the value.

In the two-channel, fault-free case, the *f*-cube2 actually simulates the classical e-cube algorithm, since only

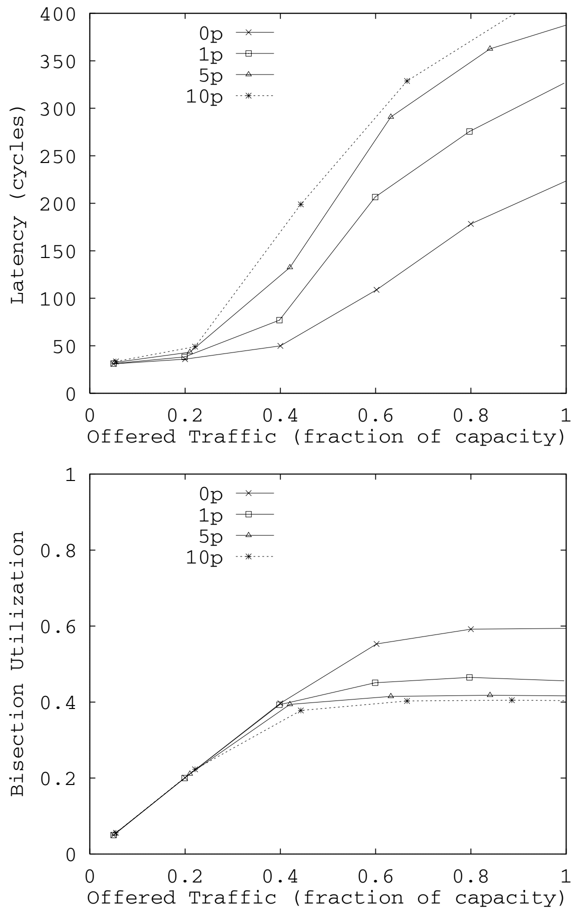

Figure 7: Performance of the *f*-cube2 algorithm with two virtual channels per physical channel. The label dp indicates results for d% faults.

one class of virtual channels in each physical channel are used in the absence of faults. The peak utilization is close to 60%. (This value differs from the classical result of 50% utilization for the e-cube [23], probably because of the buffer depth used. A separate set of simulations, not reported here, have indicated that when the buffer depth is 4, 50% utilization is obtained.) Similarly, the four-channel, fault-free case actually simulates the e-cube with three virtual channels.

Now, let us consider the performance of *f*-cube2 under faults. There is a sharp drop in the performance even with 1% (five links) faults. For the two-channel case, the latencies increase starting at 20% offered load. The *f*-cube2 is more resistant to increase in latencies when four virtual channels are used. In terms of utilization, the *f*-cube2 is consistent with its fault-free performance for up to 40% load in both cases. But there is a large decrease in the utilization at high loads. With more faults, the *f*-cube2 shows graceful degradation of performance. In all cases, four channels give better performance than two channels.

The dramatic reduction in throughput and increase in latencies can be explained as follows. Since the e-cube is nonadaptive, each misrouted message in the network is

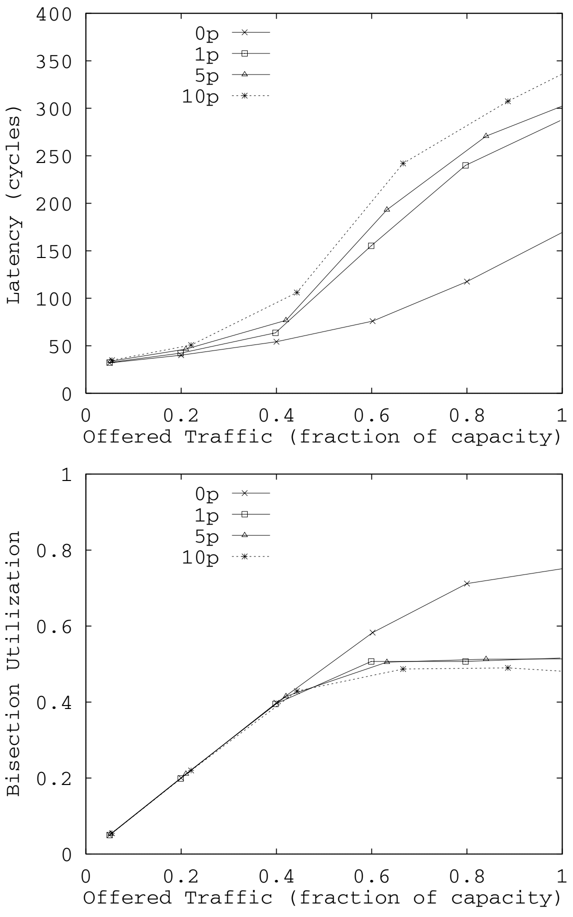

Figure 8: Performance of the *f*-cube2 algorithm with four virtual channels per physical channel. The label *d*p indicates results for *d*% faults.

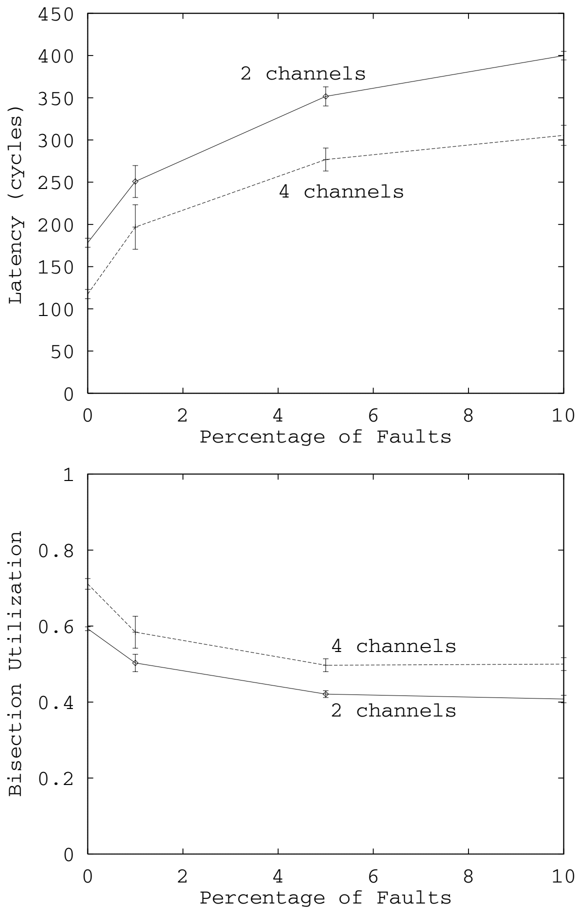

Figure 9: Comparison of peak performances of the *f*-cube2 for two- and four-channel cases.

routed around one or more f-rings. Thus there is heavy contention for channels on each f-ring in the network, which makes each f-ring a hotspot. This reduces throughput and increases latency even for normal messages which may be waiting for the channels reserved by the misrouted messages outside f-rings.

4.2 Peak performance

The results reported above are specific to a randomly generated fault set for each fault case. To see the performance limits of *f*-cube2, we have conducted further simulations with 15 different randomly generated fault sets for the 1, 5, and 10 percent fault cases. The injection rate is such that the load on the network would be 80% in the absence of faults. For each fault set of each fault case, we have sampled 100,000 delivered messages after the network reached its steady state. Then, we have computed the average of the samples for each metric. The results are given in Figure 9. For the sake of comparison, the fault-free performance at 80% load is also indicated. The vertical bars indicate the 95% confidence intervals. Four channels yield much better performance than two channels. For 1% faults, the latency is lower by 50 cycles and utilization is higher by 16% with two extra virtual channels. For 10% faults, the latency is lower by 90 cycles and utilization is higher by 22%.

The *f*-cube2 does not misroute a message before it is blocked by a fault. So, the *f*-cube2 is a nonadaptive algorithm with respect to congestions. Dally and Aoki [9] report that the fully-adaptive dimension-reversal method achieves a peak throughput of 0.54 with 8% faults when 16 virtual channels are used. Our simulations for isolated faults indicate that the *f*-cube2 provides similar throughputs (50% utilization with 10% faults) with only four virtual channels. For a more direct comparison, one should simulate both algorithms in the same simulation program.

5 Concluding remarks

We have presented techniques to enhance the nonadaptive *e*-cube algorithm for fault-tolerant wormhole routing in mesh networks. We have used the block-fault model in which faulty processors and links form multiple rectangular regions. The concept of fault-rings and fault-chains is 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 when there are multiple faulty blocks and faults on the boundaries of network.

Fault tolerance is always expensive. The cost of fault detection and isolation is common to every routing method that needs to handle faults at the network level. The additional cost of implementation of our proposed methods is small compared to many previously proposed routing methods. First, multiple virtual channels are required to provide fault-tolerant routing. When f-rings and f-chains do not overlap with one another, two or three virtual channels per physical channel are sufficient. Overlapping f-rings and f-chains can be handled using four virtual channels [4]. Second, a special bit the message header is needed to indicate message status: normal or misrouted. Third, the router logic should handle misrouting on fault rings and u-turns. This can be easily implemented by making the routing logic programmable. Finally, each node should have additional logic to send status messages to its neighbors and determine its position in fault rings. This can be achieved using a distributed two-step algorithm [4].

We have simulated *f*-cube, the fault-tolerant version of the *e*-cube algorithm. The first few faults are significant in terms of performance. The *f*-cube shows a significant drop in the throughput even for a small number of faults. However, the *f*-cube shows graceful degradation of performance with further increase in faults. Adaptive algorithms are less susceptible to this phenomenon and exhibit a more graceful degradation of performance [6, 4].

Though we have not considered, our techniques may be extended to tori using at most two times as many channels as used for meshes. Further, the proposed techniques can be used to make minimal, fully-adaptive algorithms fault-tolerant [4, 6]. The concept of fault rings and fault chains can be extended to faults with more complex shapes such as $\diamond$, 'L', and 'T', which may occur when multiple adjacent blocks are faulty. In such cases, fault rings and chains are not regular. Our preliminary investigations indicate that the proposed techniques can be applied to such faults with some changes. We are currently working on this problem.

References

[1] A. Agarwal et al. The MIT Alewife machine: A large-scale distributed 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. A comparison of adaptive wormhole routing algorithms. In *Proc. 20th Ann. Int. Symp. on Comput. Arch.*, pages 351–360, May 1993.

[4] R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. Technical Report CS-94-2, Div. of Math and CS, Univ. of Texas at San Antonio, May 1994.

[5] S. Borkar et al. iWarp: An integrated solution to high-speed parallel computing. In *Proc. Supercomputing '88*, pages 330–339, 1988.

[6] S. Chalasani and R. V. Boppana. Fault-tolerant wormhole routing in tori. In *Proc. 8th ACM Int. Conf. on Supercomputing*, July 1994.

[7] 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.

[8] W. J. Dally. Virtual-channel flow control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205, Mar. 1992.

[9] 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.

[10] 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.

[11] 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.

[12] S. A. Felperin, L. Gravano, G. D. Pifarré, and J. L. Sanz. Routing techniques for massively parallel communication. *Proceedings of the IEEE*, 79(4):488–503, 1991.

[13] P. T. Gaughan and S. Yalamanchili. Pipelined circuit-switching: A fault-tolerant variant of wormhole routing. In *Proc. Fourth IEEE Symp. on Par. and Distr. Processing*, pages 148–155, 1992.

[14] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proc. 19th Ann. Int. Symp. on Comput. Arch.*, pages 278–287, 1992.

[15] 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.

[16] K. D. Gunther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Trans. on Communications*, COM-29(4):512–524, April 1981.

[17] P. Kermani and L. Kleinrock. Virtual Cut-Through: A New Computer Communication Switching Technique. *Computer Networks*, 3:267–286, 1979.

[18] S. L. Lillevik. The Touchstone 30 Gigaflop DELTA prototype. In *Sixth Distributed Memory Computing Conference*, pages 671–677, 1991.

[19] 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.

[20] A. L. Narasimha Reddy and R. Freitas. Fault tolerance of adaptive routing algorithms in multicomputers. In *Proc. Fourth IEEE Symp. on Par. and Distr. Processing*, pages 156–161, 1992.

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

[22] W. Oed. The cray research massively parallel processor system, CRAY T3D. Technical report, Cray Research Inc., Nov. 1993.

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