Fault-Tolerant Communication with Partitioned Dimension-Order Routers

Rajendra V. Boppana, *Member*, *IEEE*, and Suresh Chalasani, *Member*, *IEEE*

Abstract—The current fault-tolerant routing methods require extensive changes to practical routers such as the Cray T3D's dimension-order router to handle faults. In this paper, we propose methods to handle faults in multicomputers with dimension-order routers with simple changes to router structure and logic. Our techniques can be applied to current implementations in which the router is partitioned into multiple modules and no centralized crossbar is used. We consider arbitrarily located faulty blocks and assume only local knowledge of faults. We apply our techniques for torus networks and show that, with as few as four virtual channels per physical channel, deadlock- and livelock-free routing can be provided even with multiple faults and multimodule implementation of routers. Our simulations of the proposed technique for 2D tori and mesh indicate that the performance degradation is similar to that seen in the case of cross-bar based designs previously proposed.

Index Terms—Cray T3D router, dimension-order router, fault-tolerant routing, multicomputer networks, message routing, torus networks, wormhole routing.

1 Introduction

Many recent experimental and commercial multicomputers and multiprocessors [29], [23], [12] use direct-connected networks with grid topology. A (k, n) network has an n-dimensional grid structure with k nodes (a node is a processor-memory-router element) in each dimension such that every node is connected to two other nodes in each dimension by direct communication links. The majority of these multicomputers use dimension-order or *e*-cube routing with *wormhole* switching [18]. Wormhole is a form of cut-through routing in which blocked messages hold on to the channels they already reserved.

In practice, the *e*-cube routing is implemented using multiple modules such that each module handles routing of messages in exactly one dimension. We refer to this implementation as the multimodule or partitioned dimension-order router (PDR) implementation [19], [14], [29], [23], [12]. For example, the Cray T3D uses a 3D torus network with each PDR implemented using three chips—one chip for each dimension module. An alternative router implementation is to use centralized crossbars or partitioned crossbars (if it is infeasible to implement the entire crossbar in a single chip) [25], [27] to handle the switching in each router. While crossbar implementations can offer adaptivity and more flexibility, each crossbar chip requires more numbers of pins than the module chips used as the building block for PDR implementations. Thus, for the same technology, a PDR implementation yields wider channels compared to the crossbar implementation.

While the *e*-cube is simple to implement and provides high throughput for uniform traffic, it cannot handle even simple node or link faults due to its nonadaptive routing. Adaptive, fault-tolerant cut-through routing algorithms have been the subject of extensive research in recent years [11], [20], [16], [22], [26], [1], [4], [9], [21], [2], [17], [6]. These results implicitly or explicitly assume routers with centralized crossbars. Therefore, such techniques are not suitable for multiprocessors with PDRs. Several other results (see, for example, [24], [28] and the references therein) exploit the rich interconnection structure of hypercubes and are not suitable for high-radix, low-dimensional tori. The current techniques to handle faults in torus and mesh networks require one or more of the following: 1) new routing algorithm, 2) substantial changes to router implementation, in particular crossbars connecting input channels in two or more dimensions to output channels in two or more dimensions, 3) global knowledge of faults, 4) restriction on the shapes, locations, and number of faults, 5) relaxing the constraint of guaranteed delivery, deadlock- or livelock-free routing.

In this paper, we propose a technique to incorporate fault tolerance into networks with PDRs. Our approach is to provide interprocessor communication among the fault-free nodes, rather than to recreate or simulate the original network topology. We have previously proposed similar techniques for fault-tolerant routing in multicomputer networks with crossbar based routers [4], [9], [7]. The main contribution of this work is to show that partitioned dimension-order routers also can be enhanced for fault-tolerant routing *without using crossbars*. We show that with a small increase in the resources and simple changes to the router organization and routing logic, multiple block faults can be handled without compromising livelock and deadlock freedom.

Our technique works with local knowledge of faults (each fault-free node knows only the status of its and its neighbors' links), handles multiple faults, and guarantees livelock- and deadlock-free routing of all messages. Our

- *R.V. Boppana is with the Division of Computer Science, Universit of Texas at San Antonio, San Antonio, TX 78249. E-mail: boppana@cs.utsa.edu.*
- *S. Chalasani is with TM Floyd & Company, 6133 N. River Rd., Rosemont, IL 60018. E-mail: surechalasani@hotmail.com.*

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 110037.

fault model allows multiple node and link faults as long as the fault regions are convex in shape—rectangular in 2D tori, cubic in 3D tori, etc. The convex fault model is simple enough to provide modular routing, yet powerful enough to model node and printed-circuit-board level faults. We apply our techniques to torus networks and show that, with as few as four virtual channels per physical channel, and some modifications to the interconnection of dimension-modules, PDRs can be used for fault-tolerant routing.

Section 2 gives an overview of dimension-order routers. Section 3 describes the fault-model. Section 4 describes the changes to the router required to handle faults. Section 5 gives the required modifications to the routing logic. Section 6 presents simulation results on the performance of mesh and torus networks under faults. Section 7 concludes this paper.

2 Partitioned Dimension-Order Routers

A (k, n)-torus has n dimensions—$\textsc{dim}0, \ldots, \textsc{dim}_{n-1}$, k nodes per dimension, and $N = k^n$ nodes. Each node is uniquely indexed by a radix-k n-tuple. 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 dimension i are $(x_{n-1}, \ldots, x_{i+1}, x_i \pm 1, x_{i-1}, \ldots, x_0)$, where addition and subtraction are performed modulo k. Each link provides full-duplex communication using two unidirectional physical channels. A link is said to be a *wraparound link* if it connects nodes $(x_{n-1}, \ldots, x_{i+1}, 0, x_{i-1}, \ldots, x_0)$ and

$$(x_{n-1}, \ldots, x_{i+1}, k-1, x_{i-1}, \ldots, x_0)$$

in dimension i, $0 \leq i < n$. Each node is a combination of processor, memory, and router. Since our interest in this paper is in the routing part of a node, we use node and router synonymously. To illustrate our technique, we use a 2D or 3D torus as a typical network. However, our results can be extended to multidimensional tori and meshes in a straight forward manner.

As per dimension order routing, each message completes the required hops (in a shortest path) in dimension $\textsc{dim}_i$ before taking any hops in $\textsc{dim}_j$, $0 \leq i < j < n$, where n is the number of dimensions in the network.

The Cray T3D [12] implements such a partitioned dimention-order router in each node using three identical router chips. A pair of 24-bit unidirectional lines (16-bit data + 8-bit control) interconnect appropriate dimension chips in adjacent nodes in the Cray T3D router. In addition, each chip has an input from the network interface (for injection of messages) or from previous dimension router chip and an output to the next dimension router chip or to the network interface (for delivery of messages). So, each router chip has three incoming 24-bit channels and three outgoing 24-bit channels. Not counting pins for power supply, ground, etc., each router chip requires at least 144 pins for data and control of virtual channels. For a crossbar based router implementation, one chip is used per router. Such a chip requires at least 336 pins ($2 * 6 * 24 = 288$ pins for internode-connections and $2 * 24 = 48$ pins for injection and consumption channels). Thus, PDR implementations have lower pin requirements per router chip. For the same number of pins per chip, PDRs can provide wider channels. The main disadvantages of PDRs are increased chip count and additional bottlenecks in the form of interchip links used by messages that need to change their dimensions.

Since channels are the resources for which messages compete in wormhole routing, cyclic dependencies, and deadlocks in a torus are avoided by simulating two virtual channels (denoted *high* and *low*) on each physical channel [18]. (The Cray T3D actually simulates four virtual channels to handle two distinct classes of messages with two virtual channels per class of messages.) In each dimension, a message starts with the *low* virtual channel and switches to the *high* virtual channel after taking a wraparound connection. This is one example of breaking deadlocks using virtual channels. Several variations of this are possible [18], [13].

3 Fault Model

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 routers are still connected and thus should be used for routing messages in the mean time. We assume that all faults are nonmalicious faults; that is, a failed component simply ceases to work. Therefore, messages are injected into the network by processors with nonfaulty router nodes. Furthermore, messages are destined only to processors with nonfaulty router nodes. We also assume that faults do not disconnect the network. (If the network is disconnected, the proposed results can be applied to the resulting subnetworks, which are meshes, with some restrictions.) These assumptions are commonly made in fault analyses in literature [11], [16].

Detection and isolation of faults is done locally. Also the fault information is kept locally. That is each node knows only the faults of its and its neighbors' links. Each node is required to detect faults on its incoming physical channels and its router chips. A node can detect its faulty components, if any, using a suitable self-test sequence periodically. A healthy node sends status signals to its neighbors on its outgoing physical channels and monitors status signals sent by its neighbors on its incoming physical channels. Missing or incorrect sequences of signals indicate malfunction of the link or the node sending them. The nodes at the end of a malfunctioning link stop using that link. When a node detects a faulty link, it reports this fault to its neighbors reachable via the fault-free links. We develop fault-tolerant algorithms, for which it is sufficient if each nonfaulty node knows the status of the links incident on it and its neighbors. Another approach for fault detection is given in [2].

We consider router chip and internode link faults. Interchip link faults can be treated as pin faults on router chips, and are modeled as router-chip failures. Our fault model is such that in a router node, either 1) one router chip is faulty or 2) all router chips are faulty. If at most one of the router chips is faulty, then the internode links in the corresponding dimension are faulty. If two or more router chips are faulty, then the entire node is marked faulty.

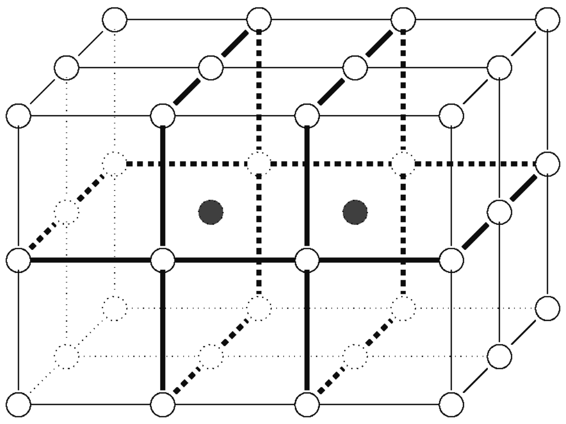

Fig. 1. Example of a block fault and the corresponding f-rings in a 3D torus. Shaded nodes and the links incident on them are faulty. For the 3D torus, wraparound links and faulty links are not shown for clarity and internal healthy links are indicated by dotted lines. Thick lines indicate the fault rings.

We model *block* or *convex* faults [11], [4]. In the block fault model, the set of faulty nodes or links can be partitioned into disjoint subsets such that each subset can be contained by an nD cube, the interior of which contains only the components of the fault-set. Block faults have the shape of a 3D cube in a 3D torus, a rectangle in a 2D torus, and so on. It is noteworthy that the fault set containing an entire row or column of a 2D torus is not a valid block fault. A two-node block fault in a 3D torus is shown in Fig. 1.

The block model is simple, yet models two common fault scenarios: single faults and multiple dependent faults, which can occur, for example, if a board (which has a block of nodes) loses its power-supply or is removed for repair. In addition, the routing techniques developed here can be used to provide a secure computation environment within a multiprogramming mode, where several users share the processors and the network. To provide a secure computing environment, a block of nodes may be allocated such that the nodes and the links among them are not used by other computations or messages resulting from them. By treating such a block of processors and links as faulty in routing the other messages, the proposed techniques can be applied for on-the-fly allocation and release of blocks of nodes for special-purpose computations.

A simple characterization of block faults is that a fault-free node may have at most one faulty link incident on it. Using this rule, any fault pattern can be *blocked*: If a node has more than one faulty link, it marks itself faulty. Thus, a fault is blocked within a finite number of steps, bounded by the diameter of the network.

3.1 Fault Rings

Consider a 2D torus with a block fault. This block fault is enclosed by a ring of nonfaulty nodes and links; the smallest such ring is called the *f-ring* for that fault [4]. In an nD torus, the block fault is an nD cube such that any 2D cross-section of the fault is a 2D block fault. Therefore, for a block fault in an nD torus, several fault rings are formed, one for each possible 2D cross section of the fault. Examples of fault rings for a 3D torus are shown in Fig. 1. An f-ring represents a two-lane path to a message that needs to go through the block fault contained by the f-ring. Thus, an f-ring simulates four paths to route messages in two dimensions.

A fault ring corresponding to each 2D cross-section of a fault-block can be formed in a distributed manner using a two-step process. In the first step, each node that detected a faulty neighbor sends this information to its neighbors in other dimensions. In the second step, based on the set of messages received in the first step, each node that is to be on the f-ring determines its neighbors on the f-ring [4].

There can be several fault rings, one for each f-region, in a faulty network with multiple faults. Up to two f-rings in a 2D torus may have a common link, and up to four f-rings may have a common node. Overlapped f-rings produce more channel dependencies, and require more classes of virtual channels to avoid deadlocks.

4 Modifications to Partitioned Dimension-Order Routers

In this section, we discuss the hardware modifications to provide additional connections between the router chips in a node. The next section discusses the changes to the routing logic.

For n-dimensional (nD) networks there are n router chips, one for each dimension from 0 to $n-1$. The following modifications to the router structure are necessary for nD networks (see Fig. 2):

1. Connections from injection channels to the inputs of all router chips.
2. Connections from the output of each router chip to a multiplexer; the output of this multiplexer is connected to the consumption channel of the processor.
3. Connections from the output of router chip i to the inputs of router chips $(i-1) \bmod n$ and $(i+2) \bmod n$, for $0 \leq i < n$.

Note that the two new interchip connections are in addition to the default i to $i+1$ connection in the original PDR. The i to $i-1$ connection is to facilitate adaptive routing in two adjacent dimensions. The i to $i+2$ connection is used to skip a failed DIM_{i+1} chip. With these connections, any single chip failure or faulty links in any one dimension can be tolerated. For $n = 2$ and $n = 3$ even fewer connections are required. For example, for a 2D torus, only the following new connections are required: a connection from DIM_1 to DIM_0 router chip, injection channel input to DIM_1 chip, and consumption channel output from DIM_0 chip. An example of this design is given in Fig. 3.

The proposed modifications increase the number of pins used for interchip channels (within a node). Prior to the modifications, each router chip has one interchip incoming (injection from previous dimension router chip) channel. With the proposed modification, we need extra pins to provide up to four (up to three for interchip and one for injection) incoming channels. It is noteworthy that this is the worst case increase in the pin count of a router chip, and is independent of the dimension of the torus. If desired, the pin count may be reduced by multiplexing multiple interchip channels (see Fig. 2) from different neighbor chips

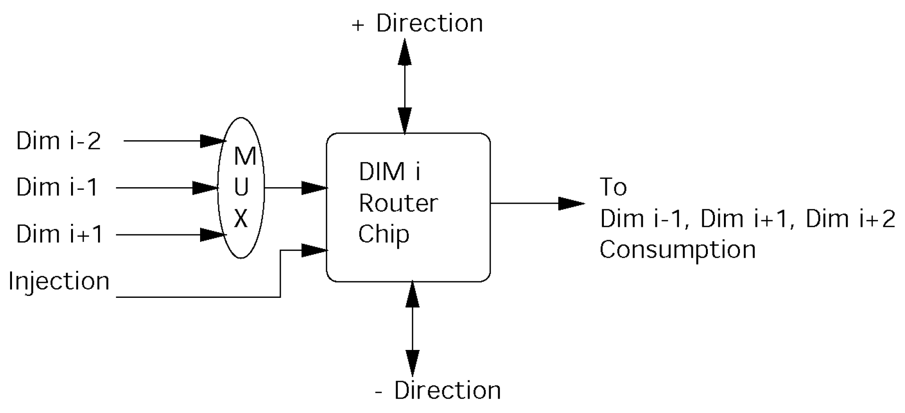

Fig. 2. Modifications to router chip i to support fault-tolerant routing in tori.

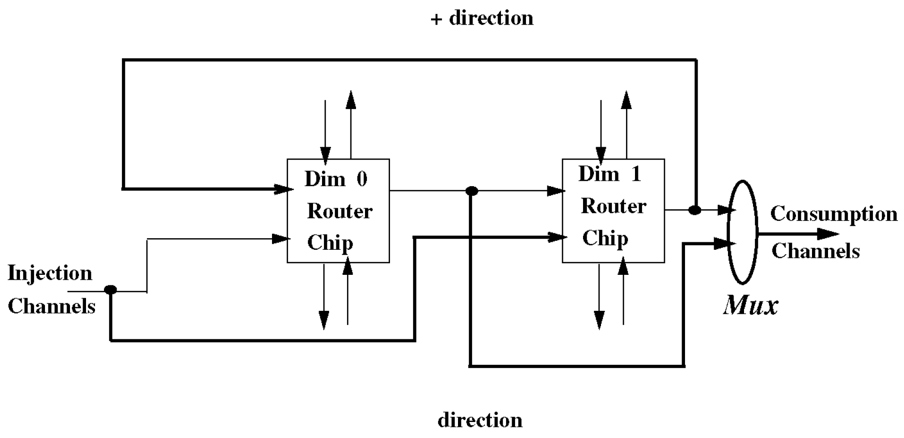

Fig. 3. A modified 2D dimension-order router to support fault-tolerant routing.

onto one single incoming channel of a chip. There are two possible ways of doing this. One idea is very similar to the sharing of a bus by multiple processors in a symmetric multiprocessor. This will require a few additional pins for request and grant, but will avoid two additional sets of data and other control pins. The arbitration logic is simple and not likely to use up too much of space inside a chip. If this is not feasible, an external multiplexer may be used, but it has the undesirable effect of increasing the chip count. When there are no faults, there is no reduction of bandwidth for interchip communication. Only for nodes on an f-ring the interchip channels are used more. Also, the attempts to save pins incur more delay because of bus arbitration or multiplexers. But, this additional delay is not too significant compared to the delay through the router chip or the queuing delay at moderate to high traffic loads. Also, this additional delay does not affect the network throughput, since the router chips operate in a pipelined fashion.

5 Fault-Tolerant Dimension-Order Routing

We begin with modifications to the routing logic in each router chip. Our approach is to use the nonadaptive dimension-order routing if a message path is not blocked by faults. For messages blocked by faults, we provide alternative paths around the faults. The techniques are livelock- and deadlock-free and guarantee delivery of each and every message injected into the network. We present our technique for 2D tori and extend it to higher dimensional tori.

5.1 Fault-Tolerant Routing in 2D Tori

At any time, the next hop of a message is specified by the default (shortest-path, nonadaptive) *e*-cube, or the fault-tolerant routing logic. We distinguish between the two types of hop specifications.

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

Message types. To route messages around the f-rings, messages are classified into one of the following types: DIM_{0-}, DIM_{0+}, DIM_{1+}, or DIM_{1-}. The type of a message can be determined by an intermediate node using the message's destination and status (discussed below). For easier description, we call $\text{DIM}_{0\pm}$ messages collectively as *row* messages, and $\text{DIM}_{1\pm}$ messages as *column* messages. With the normal dimension-order routing, a row message can turn into a column message, but not vice versa. When there are faults, we preserve this property by avoiding a message's type determination while it is being misrouted.

Normal versus misrouted messages. To handle misrouting of messages blocked by faults, we introduce a 2-bit

1. http://www.cs.utsa.edu/faculty/boppana/papers.htm.

```
// Messages are injected with normal status and undefined f-ring direction
// When a node receives a message meant for it, the node consumes the message
// Otherwise, it applies the following algorithm to determine
// the next node in message's path

1.  IF ($M$'s status is normal) // $M$ is currently normal. See if it should continue to be normal
    THEN
            Determine the dimension of the normal hop for M. Let it be $i$.
            IF ($M$'s normal hop is on a $\text{DIM}_{i+}$ channel) THEN
                    set $M$'s type to $\text{DIM}_{i+}$
            ELSE set $M$'s type to $\text{DIM}_{i-}$
            ENDIF
            IF ($M$'s normal hop is on a faulty link) THEN
                    set $M$'s status to misrouted
                    set $M$'s f-ring direction to clockwise or counter-clockwise using Table 1
            ENDIF
2.  ELSE // $M$ is being misrouted. See if it should become normal
            IF (($M$'s type is $\text{DIM}_{1+}$ AND $M$'s normal hop is a $\text{DIM}_{1+}$ hop)
               OR ($M$'s type is $\text{DIM}_{1-}$ AND $M$'s normal hop is a $\text{DIM}_{1-}$ hop)
               OR (($M$'s type is $\text{DIM}_{0+}$ OR $\text{DIM}_{0-}$) AND $M$'s normal hop is not blocked))
            THEN
                    set $M$'s type to Normal
                    set $M$'s f-ring direction to undefined
            ENDIF
    ENDIF
// Route $M$ using $e$-cube or misrouting logic
3.  IF ($M$'s status is normal) THEN
            use the normal hop to route the message
    ELSE use the hop in the f-ring direction specified in its status field
    ENDIF
```

Fig. 4. Pseudocode of fault-tolerant routing logic used by nodes. C++ style comments and block letters for keywords are used.

status field in the message header. The first bit indicates the status—misrouted or normal—of the message. For a message not currently blocked by a fault, this bit is cleared. If a message is blocked by a fault, then its status bit is set until it is misrouted to the extent that the original dimension-order routing can be used. There are two paths—clockwise and counter-clockwise—on the f-ring to misroute a blocked message. When a normal message is blocked by a fault, an appropriate orientation is chosen based on its type and its destination. The specified orientation is indicated in the second bit of the status field. A misrouted row message travels on a coulmn of the corresponding f-ring, and becomes normal when it reaches a corner node of the f-ring. Also, the row hops as specified by the dimension-order routing are unavailable for a misrouted row message until it reaches a corner node of the f-ring. A misrouted column message becomes normal only when it reaches the other row of the current f-ring such that it is in the same column as its destination.

Fault-tolerant routing logic. The fault-tolerant version of the dimension-order routing is given in Fig. 4. A message, say M, comes into an intermediate node in its path as a normal or misrouted message and leaves it as a normal or misrouted message. The incoming status is used to determine M's type and status for the next hop. It is easier to understand the algorithm if it is assumed that a message's type is indicated in its header using an additional type field. (But, the type of a message can be determined by each intermediate node unambiguously without having to maintain it in the message header. For a normal message, its type is determined by the dimension-order routing logic. For a misrouted message, its type can be determined by noting the current dimension of travel and virtual channel used.) The "if" conditions in step 2 of the algorithm in Fig. 4 ensure that the misrouted message has traversed far enough on the f-ring so that the original dimension-order routing can be used again. An implementaion of this algorithm is may be found in the simulation code available via the web.[1].

Example. In Fig. 5, node $(1,2)$ and link $<(3,2),(4,2)>$ are faulty, which give rise to two nonoverlapping f-rings. Message M originates at $(1,0)$ and is destined for $(4,2)$. It begins at its source as a DIM_{0+} message and travels to $(1,1)$. At $(1,1)$ its e-cube path is blocked by the faulty node $(1,2)$. This message then travels to $(2,1)$ as a misrouted message. It becomes normal again at $(2,1)$ and travels to $(2,2)$. M becomes a DIM_{1+} message at $(2,2)$, and is misrouted from $(3,2)$ to $(4,2)$.

TABLE 1
Directions to be Used for Misrouting Messages on Nonoverlapping f-Rings

Message Type	Position of Destination	F-Ring Orientation
DIM_{0+}	In a row above its row of travel	Counter Clockwise
DIM_{0+}	In a row below its row of travel	Clockwise
DIM_{0-}	In a row above its row of travel	Clockwise
DIM_{0-}	In a row below its row of travel	Counter Clockwise
DIM_{1+}	(don t care)	Clockwise
DIM_{1-}	(don t care)	Counter Clockwise

5.2 Fault-Tolerant Routing in nD Tori

The fault-tolerant algorithm described above for 2D tori can be extended to n-dimensional (nD) tori using the planar-adaptive routing (PAR) technique [11]. A block fault in an nD torus satisfies the condition that each 2D cross-section of the fault is rectangular in shape (see Section 3). Each such fault-ring sits in a 2D plane. Let $\text{DIM}_i - \text{DIM}_j$ denote the 2D plane involving dimensions i and j, where $0 \leq i, j \leq n-1$ and $i \neq j$. The routing algorithm still needs only four virtual channels per physical channel. The key issue is how virtual channels and planes are used to route messages.

A normal message that needs to travel in DIM_i, $0 \leq i < n$, as per the *e*-cube is a DIM_i message and is routed in a plane of the type $\text{DIM}_i - \text{DIM}_{i+1(\text{mod} n)}$. Depending on its direction of travel in DIM_i, a message can be further classified as DIM_{i+} or DIM_{i-}. A DIM_i message that completed its hops in the ith dimension 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 in an appropriate direction as determined in the case of a 2D torus. For DIM_i, $0 \leq i < n-1$, messages, the extent of misrouting around a fault is the same that in the case of DIM_0 messages in the 2D case presented above, but DIM_{n-1} messages are misrouted on three sides of an f-ring in the same manner as DIM_1 messages are misrouted in the 2D case. Fig. 6 illustrates the misrouting of messages around a fault in a 3D torus.

5.2.1 Allocation of Virtual Channels

In a fault-free network with dimension-order routing, each physical channel is used by a specific type of message. With faults, however, blocked messages are misrouted and some physical channels around f-rings are used by multiple types of messages. This creates cyclic dependencies. To break these new dependencies and to keep the routing deadlock-free, we use two new classes in addition to the original two classes required for deadlock-free routing in a fault-free torus. These additional virtual channels require additional flit-buffers in each router chip. Let the four classes of virtual channels be c_0, c_1, c_2, c_3. On each physical channel (internode as well as intranode—between router chips), a virtual channel of each class is simulated. Depending on the direction and dimension a message is traveling before being blocked by a fault, it can be one of $2n$ possible types: -DIM_{i+} and DIM_{i-}, $i = 0, \ldots, n-1$. The channel allocation is such that any new dependencies among these types of messages caused by sharing of the physical channels on f-rings are broken. DIM_0 messages use virtual channels of class c_0 before taking a hop on a DIM_0 wraparound channel and c_1-virtual channels thereafter. A DIM_0 message that has completed hops in DIM_0 but not reached its destination will turn into a DIM_1 message and continues routing as a DIM_1 message. Similarly, a DIM_1 message uses c_2 or c_3 virtual channels and turns into a DIM_2 message after completing hops in DIM_1. This continues for $\text{DIM}_0, \ldots, \text{DIM}_{n-2}$. The allocation of virtual channels is slightly different for the highest dimension message. For odd n, a DIM_{n-1} message uses c_0 or c_1 while traveling in DIM_{n-1} and c_2 or c_3 while traveling in DIM_0 (because of misrouting). For even n, a DIM_{n-1} message uses c_2 or c_3 while traveling in DIM_{n-1} and in DIM_0 (if misrouted). These rules for fault-tolerant routing of messages are summarized in Table 2. In Fig. 5, the message uses c_0 channels from $(1, 0)$ to $(2, 2)$ and c_2 channels thereafter.

A message that has completed hops in a dimension goes to the next dimension router chip using a virtual channel on the interchip link to that chip. The virtual channel used on the interchip link can be the one it will use in traversing the next dimension (as a normal message). If it does not need to travel in the next dimension, then it can use any virtual channel that can be used by a message of that dimension on the interchip link to the following dimension chip. For example, if a DIM_0 message has completed its hops, then it goes to DIM_1 chip using c_0 or c_1 channel on the interchip link. If it does not need to take any hops in DIM_1, then it uses c_2 or c_3 on the link from DIM_1 chip to DIM_2 chip.

5.3 Proof of Deadlock-Free Routing

The above routing algorithm works for routers implemented using a full crossbar which can connect any input channel of a node to any output channel. As mentioned earlier, the previous works on fault-tolerant wormhole routing algorithms implicitly assumed that the router in a node is implemented using a crossbar, which provides full switching capability among multiple dimensions [11], [16], [3], [8]. In a multichip dimension-order router, changing dimensions of travel by messages is complicated since interchip channels are shared among different types of messages. We prove below that these additional dependencies resulting from sharing interchip links do not cause deadlocks.

Lemma 1. *The proposed fault-tolerant routing algorithm is deadlock-free.*

Proof. There are $2n$ types of messages: DIM_{i+} and DIM_{i-}, $i = 0, \ldots, n-1$. A particular set of virtual channels are used for each message type. So, messages of a type travel in a particular virtual network formed by all the nodes and the set of virtual channels used by them. During the routing, a message of one type may change into another type (for example, a DIM_{0+} message may change into DIM_{1-} after completing its hops in DIM_0). Our proof technique relies on showing that there is a partial order among all the virtual channels of the network and

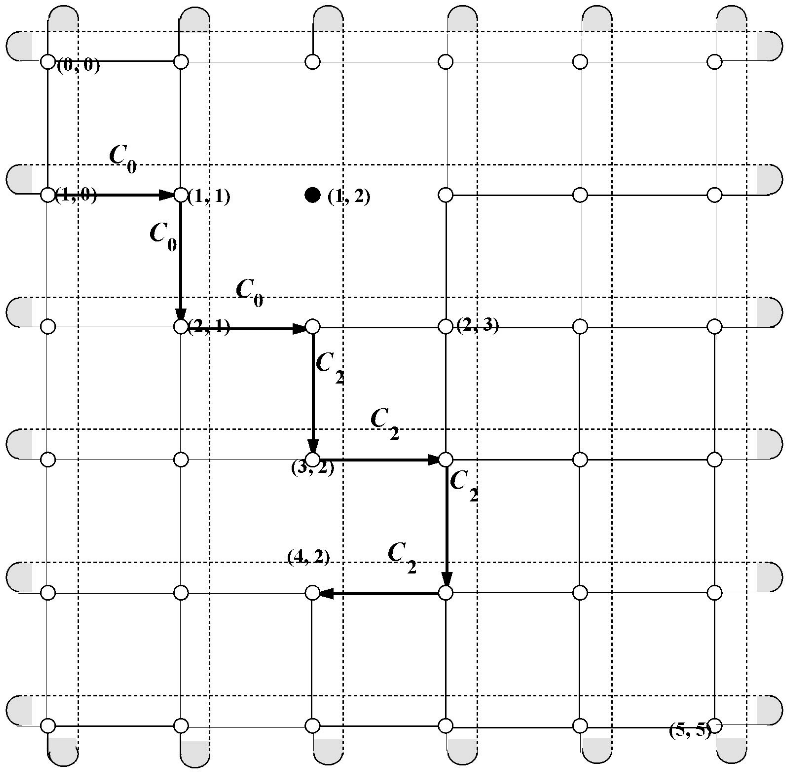

Fig. 5. Example of fault-tolerant dimension-order routing on nonoverlapping f-rings.

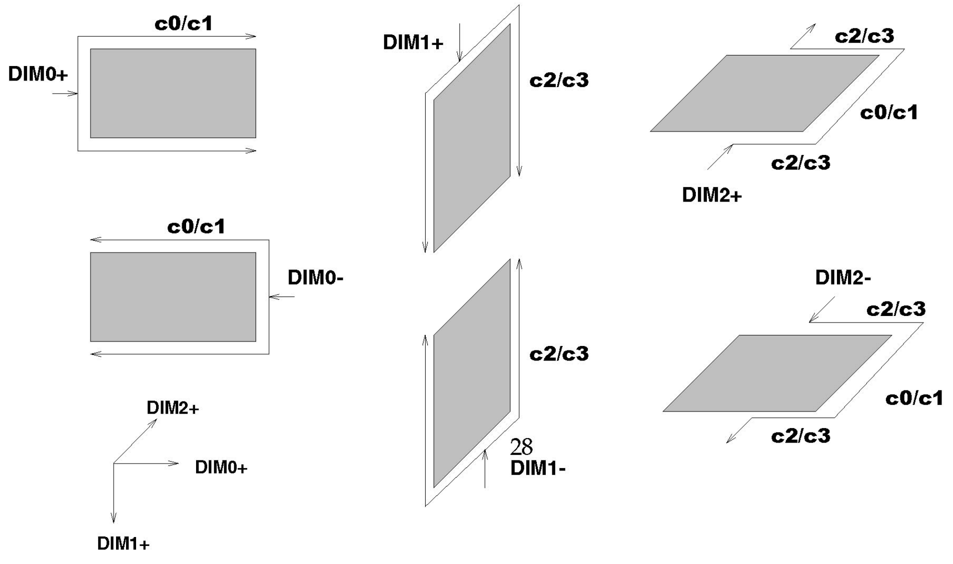

Fig. 6. Routing of six different message types around a fault in a 3D torus. The shaded area represents a faulty block, and directed lines indicate the paths of messages on the f-ring around the fault. The type of virtual channels used are also indicated.

messages acquire them in an increasing order of ranks [18]. For this we need to show that 1) the sets of virtual channels used by the various types of messages are pairwise disjoint, 2) the virtual network for each type is acyclic, and 3) the virtual networks are used by messages as per some partial order. The multiplexers do not cause any new dependencies since they simply connect inputs to outputs in a demand time-multiplexed manner.

The sets of virtual channels used by various types of messages are pairwise disjoint. First, we show that the set of virtual channels used by DIM_{i+} messages is disjoint from the set of virtual channels used by DIM_{i-} messages.

Consider DIM_{0+} and DIM_{0-} messages. If the sets of virtual channels used by DIM_{0+} and DIM_{0-} messages are not disjoint, then they share virtual channels on internode physical channels or interchip physical channels. The DIM_{0+} messages travel on positive direction DIM_0 physical channels and the DIM_1 physical channels among nodes on the left boundary columns of f-rings. Similarly, DIM_{0-} messages travel on negative direction DIM_0 physical

TABLE 2
Planes and Virtual Channels Used by Various Messages in an nD Torus with Fault-Tolerant PDRs

Message type	Plane type	Virtual channel classes
DIM_{0+}, DIM_{0-}	DIM_0-DIM_1	c_0 before reserving a wraparound link in DIM_0, c_1 after reserving a wraparound link in DIM_0
DIM_{1+}, DIM_{1-}	DIM_1-DIM_2	c_2 before reserving a wraparound link in DIM_1, c_3 after reserving a wraparound link in DIM_1
$\vdots$		
$\text{DIM}_{(n-1)+}$, $\text{DIM}_{(n-1)-}$, even n	DIM_{n-1}-DIM_0	c_2 (c_3) before (after) reserving a wraparound link in DIM_{n-1}
$\text{DIM}_{(n-1)+}$, $\text{DIM}_{(n-1)-}$, odd n	DIM_{n-1}-DIM_0	c_0 (c_1) while traveling in DIM_{n-1} before (after) reserving a wraparound link in DIM_{n-1}, and c_2 (c_3) while traveling in DIM_0 before (after) reserving a wraparound link in DIM_{n-1}

channels and DIM_1 physical channels among nodes on the right boundary columns of f-rings. This is illustrated in Fig. 7 for an f-ring in $\text{DIM}_0 - \text{DIM}_1$ cross-section of a 3D torus. Since f-rings do not overlap, the column channels are used by exactly one or none of the two classes of messages. So, DIM_{0+} and DIM_{0-} do not share virtual channels on internode physical channels since the physical channels they use are disjoint.

Because of multimodule implementation, DIM_{0+} and DIM_{0-} messages may share some virtual channels on interchip links. With our fault-tolerant routing logic, this cannot occur, however. Referring to Fig. 7 again, we note that only DIM_{0+} messages can reserve the c_0 channel from DIM_0 chip to DIM_1 chip in the middle nodes (node A in Fig. 7) on the left column and the c_0 channel from DIM_1 chip to DIM_0 chip in the corner nodes on the left boundary column of the f-ring. A DIM_{0-} message does not use them. Similarly, the c_0 channels between DIM_0 and DIM_1 chips in the nodes on the right boundary column of an f-ring are used only by DIM_{0-} messages. For messages that took hops on wraparound links in DIM_0, the above argument repeats with c_1 as the virtual channel. Hence, there cannot be sharing of virtual channels among $\text{DIM}_{0\pm}$ messages.

From the above argument, it is clear that there are no dependencies among DIM_{0+} and DIM_{0-} classes of messages. A normal message that completes its hops in DIM_0 becomes a DIM_1 message and moves to the DIM_1 chip in the current node. The use of a virtual channel into DIM_1 chip is similar to the use of the injection channel from processor into DIM_0 chip and does not cause any cyclic dependencies.

This argument can be repeated to show that DIM_{i+} and DIM_{i-}, $0 < i < n-1$, messages use disjoint sets of physical channels.

The argument to show that $\text{DIM}_{(n-1)+}$ and $\text{DIM}_{(n-1)-}$ messages use disjoint sets of physical channels must take into consideration that a DIM_{n-1} misrouted message travels on three sides of the f-ring. However, the principal argument is unchanged. As in the case of DIM_0 messages, the interchip channel between DIM_{n-1} and DIM_0 router chips in a node on an f-ring can be used by only one of the $\text{DIM}_{(n-1)+}$ and $\text{DIM}_{(n-1)-}$ types of messages. The interchip channel usage by a DIM_{2+} message on an f-ring of a 3D torus is shown in Fig. 8.

Now, consider messages of two different dimensions. By our virtual channel allocation given in Table 2, they use different classes of virtual channels on the physical channels they share. Only on disjoint physical channels may they use virtual channels of the same class.

The virtual network for each message type is acyclic. Consider DIM_{0+} messages. A normal DIM_{0+} message always progresses towards its destination. It uses c_0 channels before taking a hop on a DIM_0 wraparound link and c_1 thereafter. Since the original dimension-order routing is deadlock-free, there are no cycles among these virtual channels. A misrouted DIM_{0+} message travels on the left column of an f-ring reserving channels c_0 (or c_1) as per a linear order. Let $c_j(x, y)$ denote the virtual channel of class c_j on the physical channel that starts from node x and ends in node y, one of its neighbors. If y is a DIM_{i+} neighbor of x ($y_i = (x_i + 1) \bmod k$), then $x \to y$ is the DIM_{i+} physical channel between them. DIM_{i-} channels are defined similarly.

The linear order of c_0 channels on an f-ring is given by $c_0(x, y) \prec c_0(y, z)$ where x, y, z are nodes on the left column of the f-ring and both $x \to y$ and $y \to z$ are DIM_{1+} or DIM_{1-} physical channels. For example, for the f-ring in Fig. 7, $c_0(A', A) \prec c_0(A, B)$ and $c_0(B, A) \prec c_0(A, A')$. Now, a partial order can be defined among the virtual channels used by DIM_{0+} messages using the following rules:

- c_0 channels are ranked lower than c_1 channels.
- $c_0(w, x) \prec c_0(x, y)$ and $c_1(w, x) \prec c_1(x, y)$ if both $w \to x$ and $x \to y$ are DIM_{0+} or DIM_{0-} channels.
- $c_0(w, x) \prec c_0(x, y)$ and $c_1(w, x) \prec c_1(x, y)$ if 1) both $w \to x$ and $x \to y$ are DIM_{1+} or DIM_{1-} and 2) w, x, y are nodes on the left column of an f-ring.
- c_0 (respectively, c_1) channels on the left column of an f-ring in a $\text{DIM}_0 - \text{DIM}_1$ plane are ranked higher than the c_0 (c_1) channels on the DIM_{0+} physical channels that end in the left column nodes of the f-ring, and are ranked lower than the c_0 (c_1)

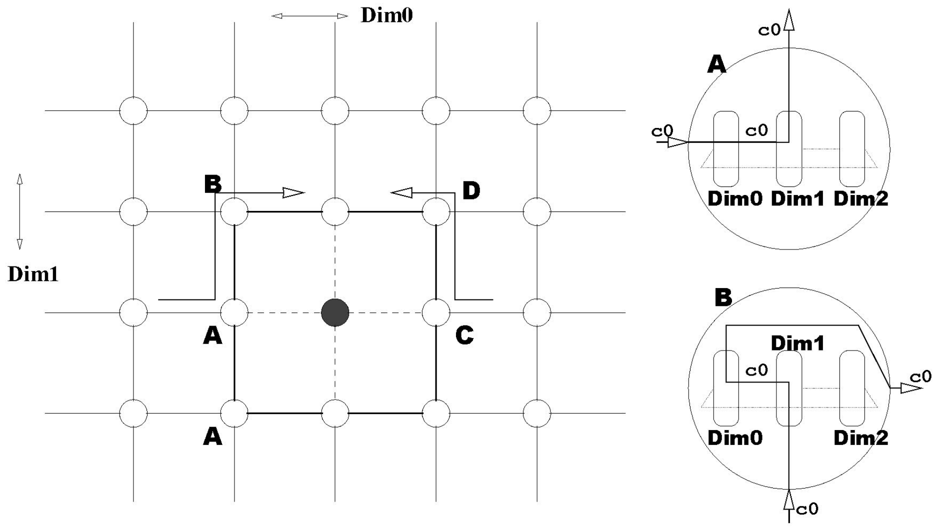

Fig. 7. Illustration of virtual channels used on interchip links. A misrouted DIM_{0+} message uses interchip links in nodes A and B and a misrouted DIM_{0-} message in nodes C and D. The interchip channels used are given by labeled, directed thick lines.

channels on the DIM_{0+} channels that start from the left column nodes.

Therefore, the virtual network of DIM_{0+} messages is acyclic. Similar rankings can be given to show that the virtual networks for other message types are acyclic.

The virtual networks are used according to a partial order. This directly follows from the dimension-order routing. A DIM_{i+}, $i = 0, \ldots, n-1$, message never uses the virtual network of a DIM_{i-} message and the virtual networks of DIM_{j+} or DIM_{j-} messages, $j < i$. It can be easily verified that this defines a partial order. □

Lemma 2. *The proposed fault tolerant routing algorithm is livelock-free and correctly routes all messages.*

Proof. To see that messages are correctly delivered without introducing livelocks in the faulty network, observe that:

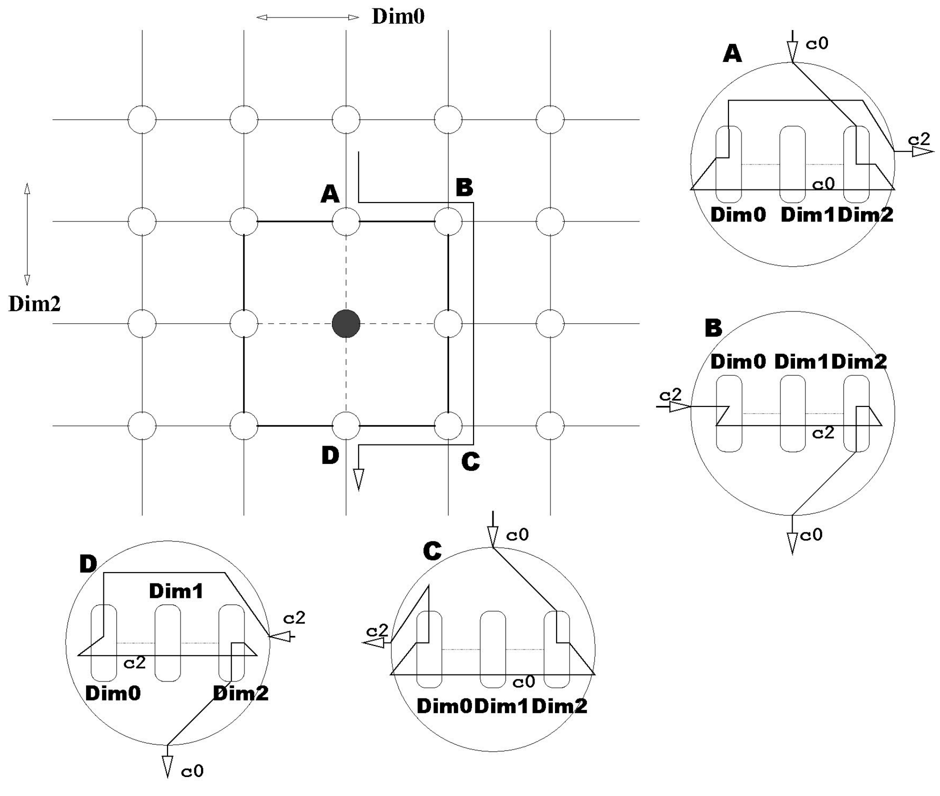

Fig. 8. Illustration of virtual channels used on interchip links. Nodes A, B, C, and D are the places where the misrouted DIM_{2+} message uses interchip links. The interchip channels used are given by labeled, directed thick lines.

TABLE 3
Virtual Channels Assigned to Messages to Handle Overlapping f-Rings

Virtual channel class	Used by message of type
c_0	DIM_{0+} for row hops before taking hop on DIM_0 wraparound link
c_2	DIM_{0+} for DIM_{1-} hops on f-rings (unused by DIM_{1+} messages)
c_3	DIM_{0+} for DIM_{1+} hops on f-rings (unused by DIM_{1-} messages)
c_0	DIM_{0-} for all hops before taking hop on DIM_0 wraparound link
c_1	DIM_{0+} for row hops after taking hop on DIM_0 wraparound link
c_1	DIM_{0-} for all hops after taking hop on DIM_0 wraparound link
c_2	DIM_{1+} for all hops before taking hop on DIM_1 wraparound link
c_3	DIM_{1-} for all hops before taking hop on DIM_1 wraparound link
c_4	DIM_{1+} and DIM_{1-} for all hops after taking hop on DIM_1 wraparound link

1. A message is misrouted only around an f-ring,
2. A message may visit an f-ring only once without changing its type,
3. There are a finite number of f-rings in the mesh,
4. A normal message progresses toward its destination with each hop, and
5. 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 that livelocks do not occur. □

5.4 Fault-Tolerant Routing in Meshes

The above techniques for fault-tolerant routing can be applied to mesh networks as well. For mesh networks, due to the absence of wraparound links, the required number of virtual channels is smaller than that for tori. For example, two virtual channels per physical channel are sufficient to handle nonoverlapping f-rings in meshes. However, in meshes, faults on the boundaries of the network (for example, topmost row in a 2D mesh) require special handling. The treatment of meshes is similar to that given in [3], [4]. The proof of deadlock-free routing is similar to that given above for torus routers.

5.5 Extensions to Handle Overlapping f-Rings

If a pair of f-rings overlap, then they have a common column, which is a left column for one f-ring and right column for the other, or a common row, which is a top row for one f-ring and bottom row for the other. We extend our routing method to handle such overlapped f-rings in 2D tori. Extensions to nD tori are similar to that given for the nonoverlapping f-rings case.

5.5.1 Routing Logic

We retain the routing logic used for the nonoverlapping f-rings case for both DIM_0 and DIM_1 messages. So, if a misrouted column message is immediately blocked by another fault region after traversing the current f-ring, either clockwise (for DIM_{1+} messages) or counterclockwise (for DIM_{1-} messages) orientation is used to route the message. This actually involves retracing some or all of its hops taken on that common row as part of misrouting on the previous f-ring. We use additional virtual channels and assign them to messages suitably to break the cycles caused by the overlaps of f-rings.

Row messages. The additional dependencies on row messages are somewhat simpler than those for column messages since they need to travel on only two sides of an f-ring in the absence of overlaps. In the nonoverlapping case, either DIM_{0+} or DIM_{0-} messages traversed on any column of an f-ring. With overlapped f-rings, both types of messages may travel on a column of an f-ring. To ensure acyclic dependencies, we give separate virtual channels for DIM_{0+} and DIM_{0-} messages when misrouted on columns of f-rings.

Column messages. For column messages, f-rings may be stacked one on top of another, causing some row physical channels of f-rings to be used by both types of column messages. So, additional dependencies due to overlaps occur only in the use of row channels of f-rings. It is noteworthy that our fault-tolerant routing logic is such that, once a message becomes a DIM_1 message, its DIM_1 hops always (even when it is being misrouted) take it closer to its destination.

First, consider the column messages that have taken hops on wrapaound links in DIM_1. These messages traverse in separate halves of the network because of shortest-path routing in DIM_1 and never share row channels of f-rings. So, we can use one class of virtual channels to route DIM_1 messages after they have taken DIM_1 wraparound links.

Now, consider the column messages that have not taken hops on wraparound links in DIM_1. Because of overlapping f-rings, they now share physical channels in the common rows of each pair of overlapping f-rings. Because DIM_{1+} and DIM_{1-} messages use the same class of virtual channels in the nonoverlapping case, cyclic dependencies occur. We break these cyclic dependencies by using separate virtual channel classes for each message type: c_2 for DIM_{1+} messages and c_3 for DIM_{1-} messages.

Table 3 gives the virtual channels used by each message type to ensure deadlock-free routing.

6 Simulation-Based Performance Study

We have used a flit-level simulator to study the performance of the fault-tolerant PDRs proposed in this paper. We have simulated 16×16 mesh and torus networks for the uniform traffic pattern with geometrically distributed message interarrival times. In practice, fixed length messages give better manageability of resources, such as injection and consumption buffers, and small message sizes are more suitable for fine-grain computations. Hence, we have used fixed length messages of 20 flits, which could be suitable for transmitting four 64-bit words together with header, checksum, and other information on 16-bit wide physical channels.

For torus simulations, we have simulated four virtual channels on each internode and interchip physical channel, and, for mesh simulations, two virtual channels per physical channel. On physical channels that are neither faulty nor part of f-rings, all the simulated virtual channels are used to route normal messages. Since, on each such physical channel, only one-dimension messages travel, extra channels are available to reduce channel congestion. Recent studies [5], [15] have shown that using more virtual channels than those necessary for deadlock-free routing improves the performance of the *e*-cube considerably. On physical channels that are part of f-rings, each virtual channel is used for a specific type of message. The virtual channels on a physical channel (internode as well as interchip) are demand time-multiplexed and it takes one cycle to transfer a flit on a physical channel.

We have assumed that messages experience processing delays when passing through intermediate nodes. We used a delay of three cycles to process header flits, and a delay of two cycles to route data flits from an incoming channel to an outgoing channel of an intermediate node. This is in addition to any other delays that a flit may see because of either round-robin processing of one incoming header at a time by the router or virtual channel bandwidth allocation.

Each virtual channel has a buffer of depth four (holds four flits) to pipeline message transmission smoothly. Because of asynchronous pipelining of message transmission among nodes, bubbles are created with shallow buffers of depth 1 or 2. So, mesh routers have 32 flits of storage and torus routers 64 flits of storage.

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 have set the injection limit to two, which means that a node may inject a new message if fewer than two of its previously injected 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 use bisection utilization and average message latency as the performance metrics. The bisection utilization (ρ_b) is defined as follows:

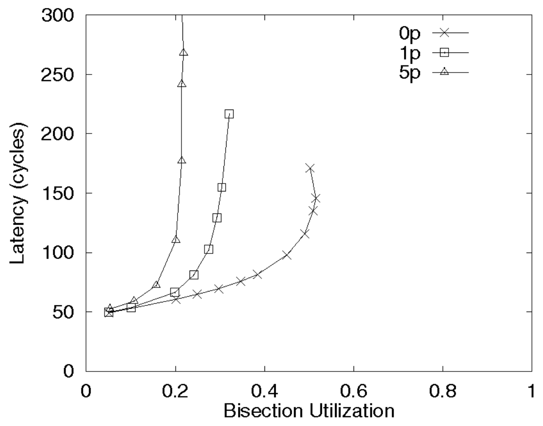

Fig. 9. Performance of the fault-tolerant PDR for a 2D torus network with four virtual channels per physical channel. The label dp indicates results for d percent faults.

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

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 a 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 network. The average message latency is the average duration from a message's injection to its consumption.

We have simulated the mesh and torus networks with no faults and with approximately 1 percent and 5 percent of the total network links faulty. We have used a mixture of node and link faults. Node faults cause more severe congestion since a node fault blocks both row and column messages, while a link fault blocks only one type of messages. We have set one node and one link faulty for the 1 percent-faults case, and four nodes and 10 links faulty for the 5 percent-faults case. 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.

6.1 Performance under Faults

Figs. 9 and 10 give the simulation results for torus and mesh networks, respectively. For each value reported in these graphs, the 95 percent confidence interval is within 10 percent of the value.

In each case, the performance for fault-free routing is much higher than the performance with faults. The peak utilization for torus PDR without faults is 52 percent, but drops to 32 percent with 1 percent faults and to 22 percent with 5 percent faults. Similarly, the peak utilization for mesh PDR without faults is 58 percent, but drops to 30 percent with 1 percent faults and to 27 percent with 5 percent faults. These results are consistent with the performance degradations seen for crossbar-based fault-tolerant dimension-order routers [4].

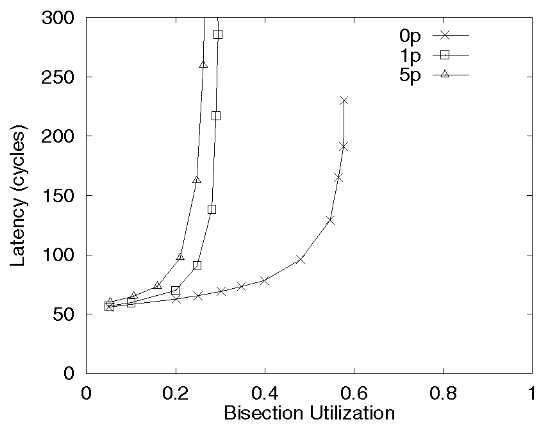

Fig. 10. Performance of the fault-tolerant PDR for a 2D mesh network with two virtual channels per physical channel. The label dp indicates results for d percent faults.

The high performance for fault-free networks is due to use of the extra channels to avoid congestion. Even with a single fault, the number of types of messages traveling on f-ring links is increased and severe congestion occurs. Thus, an f-ring becomes a hotspot causing performance degradation. Therefore, for graceful degradation of performance, some form of adaptivity should be considered. We believe that it should be feasible to provide limited adaptivity while retaining the multimodule implementation of the router.

The performances of torus and mesh networks are not directly comparable for several reasons. Mesh routers are simulated with 32 flits of storage, while torus routers are simulated with twice as much storage. Bisection utilization is a ratio of achieved throughput to bisection bandwidth, which is influenced by the topology. In terms of raw throughput, the fault-free torus delivered messages at the rate of 66 flits or 3.3 messages/cycle, while the fault-free mesh delivered at the rate of 36 flits/cycle.

6.2 Impact on Fault-Free Performance

Our approach for deadlock-free routing uses a few extra virtual channels to break cyclic dependencies among channels. If a fault-free network already uses virtual channels, then the impact of using a few more virtual channels to provide fault-tolerant routing is not too severe. Otherwise, adding virtual channels for fault tolerance may affect the fault-free performance. For example, dimension-order routing on a fault-free mesh is deadlock-free without using any virtual channels. Since we need two virtual channels per physical channel to provide fault-tolerant routing, the cost and speed of PDRs are affected. The cost is increased because of the additional switching and virtual channel controllers at the outgoing channels. The speed may be reduced because of the increased complexity of selecting an outgoing channel and additional delays through virtual channel controllers.

In this paper, we address the impact of the reduced speed on message delays and network throughput. For the sake of simplicity, assume that the node delays for flits is one cycle in the PDR without virtual channels. The reduction of router speed can be handled in one of the following two ways:

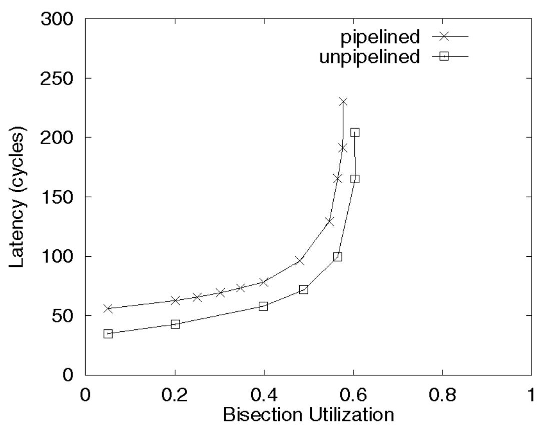

Fig. 11. Performances of unpipelined and pipelined PDRs in a (16, 2)-mesh with two virtual channels per physical channel. The unpipelined router is simulated with 1-cycle delay for flits going through an intermediate node. The pipelined router is simulated with 3-cycles delay for header flits and 2-cycles delay for data flits.

- Unpipelined routers: Transit time for each flit through an intermediate node is still one cycle. But the router operates with a slower clock.
- Pipelined routers: Clock rate of the router is kept the same. However, the transit time for a flit through a node equals multiple clock cycles.

Chien [10] analyzed several wormhole router organizations and concluded that adding virtual channels could increase the clock cycle time of a router substantially. This analysis is based on the assumption that routers are unpipelined. An unpipelined router has to examine the header of a message on an incoming channel, select an appropriate outgoing channel, and place the header on the selected outgoing channel (in the absence of contention) in one clock cycle. So, introducing virtual channels increases the delays seen by messages in their intermediate nodes.

An alternative is to pipeline the message path within a router. By pipelining the message path, the clock rate need not be reduced when virtual channels are introduced. A message still sees larger node delays in the form of multiple clock cycles. For example, a message header may see a three-stage processing: buffering at input channel, selecting and switching to an appropriate outgoing channel, and virtual channel controller at the output channel. Once a path is established for a message, its subsequent data flits cut through each intermediate node in two stages: buffering at input channel and virtual channel controller at the output channel.

We have conducted simulations to compare message delays and throughputs for both types of routers. Fig. 11 gives the results. For the pipelined router, at each intermediate node visited, header flits see 3-cycle delays and data flits 2-cycle delays. For the unpipelined router, the node delays are constant—one cycle.

If the unpipelined router has the same clock rate as the pipelined router, then the former has about 30 cycles lower latency and 5 percent higher bisection utilization. If clock cycle time of the unpipelined router is about 30 percent more than the pipelined router, then both give rise to the same message delays. However, the pipelined router gives over 20 percent higher throughput in terms of bytes/second.

In our earlier simulations for a 16×16 mesh with crossbar-based routers, one virtual channel (same as no virtual channels) per physical channel, and channel buffer depths of 8, we obtained about 60 percent bisection utilization [3]. From the results in Fig. 11, a pipelined PDR with two virtual channels and channel buffer depth of 4 achieves similar throughputs. The main difference is message delays are higher in the pipelined router. In both cases, the router clock cycle time and amount of buffer space per physical channel are the same.

This shows that adding virtual channels to a network that does not already have virtual channels may not reduce the throughput. Pipelining the message path within a router is the key. Thus, adding additional virtual channels to provide fault-tolerant communication does not necessarily reduce the performance for the fault-free case.

If a network already uses virtual channels for the fault-free case, then adding a few more virtual channels causes only small increases to the cost and clock cycle time. In such cases, the throughput increased by adding a few extra virtual channels (as in the torus case) usually outweighs any small increases in message delays.

7 Concluding Remarks

We have presented a technique to enhance the nonadaptive dimension-order algorithm for fault-tolerant wormhole routing in torus networks. This technique requires simple changes to the routing logic and implementation, works with strictly local knowledge of faults, as per which each fault-free node knows only the status of its links, handles multiple faults, and guarantees livelock- and deadlock-free routing of all messages.

We have used the block-fault model in which faulty processors and links form multiple rectangular regions. The concept of fault-rings 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 provided a path exists. If the network is disconnected, then our techniques can be applied with some modifications to each subnetwork.

Particular attention has been paid to the applicability of proposed techniques for current multicomputers which use partitioned dimension-order routers (PDRs). Since PDRs do not have centralized crossbars, most of the previously proposed techniques for fault-tolerant routing cannot be implemented without redesigning the existing routers. With the proposed techniques, however, multiple faults can be tolerated while retaining the PDR implementation.

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 do not overlap with one another, four virtual channels per physical channel are sufficient. Overlapping f-rings can be handled using five virtual channels. Second, a special bit in the message header is needed to indicate message status: normal or misrouted. Third, the router logic should handle misrouting on fault rings. This can be easily implemented by making the routing logic programmable as in the Cray T3D implementation. 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].

Our previous simulation studies for mesh networks with crossbar-based dimension-order routers exhibited graceful degradation of performance under faults [4]. The simulation results presented in this paper also indicate that the proposed technique achieves similar graceful degradation of performance even when partitioned dimension-order routers are used.

The concept of fault rings can be extended to faults with more complex shapes, such as diamond and "L," which may occur when multiple adjacent blocks are faulty. In such cases, fault rings 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.

Acknowledgments

Rajendra V. Boppana's research was partially supported by DOD/AFOSR grant F49620-96-1-0472 and U.S. National Science Foundation grant CDA9633299. This work was performed while Suresh Chalasani was with the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison.

References

[1] K. Bolding and L. Snyder, "Overview of Fault Handling for the Chaos Router," *Proc. 1991 IEEE Int'l Workshop Defect and Fault Tolerance in VLSI Systems*, pp. 124-127, 1991.

[2] K. Bolding and W. Yost, "Design of a Router for Fault-Tolerant Networks," *Proc. Parallel Computer Routing and Comm. Workshop*, pp. 226-240, May 1994.

[3] R.V. Boppana and S. Chalasani, "Fault-Tolerant Routing with Non-Adaptive Wormhole Algorithms in Mesh Networks," *Proc. Supercomputing '94*, Nov. 1994.

[4] R.V. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Trans. Computers*, vol. 44, no. 7, pp. 848-864, July 1995.

[5] S. Borkar et al., "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proc. Supercomputing '88*, pp. 330-339, 1988.

[6] Y.M. Boura and C.R. Das, "Fault-Tolerant Routing in Mesh Networks," *Proc. 1995 Int'l Conf. Parallel Processing*, pp. I.106-109, Aug. 1995.

[7] S. Chalasani and R.V. Boppana, "Adaptive Fault-Tolerant Wormhole Routing Algorithms with Low Virtual Channel Requirements," *Proc. Int'l Symp. Parallel Architectures, Algorithms and Networks*, pp. 214-221, Dec. 1994.

[8] S. Chalasani and R.V. Boppana, "Fault-Tolerant Wormhole Routing in Tori," *Proc. Eighth ACM Int'l Conf. Supercomputing*, pp. 146-155, July 1994.

[9] S. Chalasani and R.V. Boppana, "Adaptive Wormhole Routing in Tori with Faults," *IEE Proc.: Computers and Digital Techniques*, vol. 142, pp. 386-394, Nov. 1995.

[10] A.A. Chien, "A Cost and Speed Model for k-Ary n-Cube Wormhole Routers," Presented at Hot Interconnects 1993, Mar. 1993.

[11] A.A. Chien and J.H. Kim, "Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, pp. 268-277, 1992.

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

[13] Cray Research Inc., *Cray T3D Technical Summary,* Oct. 1993.

[14] W.J. Dally, "Network and Processor Architecture for Message-Driven Computers," *VLSI and Parallel Computation,* R. Suaya and G. Birtwislte, eds., chapter 3, pp. 140-222, San Mateo, Calif.: Morgan-Kaufman, 1990.

[15] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, pp. 194-205, Mar. 1992.

[16] W.J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, pp. 466-475, Apr. 1993.

[17] W.J. Dally, L.R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers," *Proc. Parallel Computer Routing and Comm. Workshop,* pp. 241-255, May 1994.

[18] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers,* vol. 36, no. 5, pp. 547-553, 1987.

[19] W.J. Dally and P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," *Proc. Int'l Conf. Computer Design,* pp. 230-234, 1987.

[20] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 12, pp. 1,320-1,331, Dec. 1993.

[21] P.T. Gaughan and S. Yalamanchili, "A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 5, pp. 482-497, May 1995.

[22] C.J. Glass and L.M. Ni, "Fault-Tolerant Wormhole Routing in Meshes," *Proc. 23rd Ann. Int'l Symp. Fault-Tolerant Computing,* pp. 240-249, 1993.

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

[24] T. Lee and J. Hayes, "A Fault-Tolerant Communication Scheme for Hypercube Computers," *IEEE Trans. Computers,* vol. 41, no. 10, pp. 1,242-1,256, Oct. 1992.

[25] Z. Liu and A.A. Chien, "Hierarchical Adaptive Routing," *Proc. Sixth IEEE Symp. Parallel and Distributed Processing,* 1994.

[26] J.Y. Ngai and C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks," *Proc. First Symp. Parallel Algorithms and Architectures,* pp. 1-9, 1989.

[27] T. Pinkston, Y. Choi, and M. Raksapatcharawong, "Architecture and Optoelectronic Implementation of the WARRP Router," *Proc. Symp. Hot Interconnects V,* Aug. 1997.

[28] C.S. Raghavendra, P.-J. Yang, and S.-B. Tien, "Free Dimensions —An Effective Approach to Achieving Fault Tolerance in Hypercubes," *Proc. 22nd Annual Int'l Symp. Fault-Tolerant Computing,* pp. 170-177, 1992.

[29] C.L. Seitz, "Concurrent Architectures," *VLSI and Parallel Computation,* R. Suaya and G. Birtwistle, eds., chapter 1, pp. 1-84, San Mateo, Calif.: Morgan-Kaufman, 1990.

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

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