Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms

Rajendra V. Boppana, *Member*, *IEEE*, Suresh Chalasani, *Member*, *IEEE*, and C.S. Raghavendra, *Fellow*, *IEEE*

Abstract—We show that deadlocks due to dependencies on consumption channels are a fundamental problem in wormhole multicast routing. This type of resource deadlocks has not been addressed in many previously proposed wormhole multicast algorithms. We also show that deadlocks on consumption channels can be avoided by using multiple classes of consumption channels and restricting the use of consumption channels by multicast messages. We provide upper bounds for the number of consumption channels required to avoid deadlocks. In addition, we present a new multicast routing algorithm, column-path, which is based on the well-known dimension-order routing used in many multicomputers and multiprocessors. Therefore, this algorithm could be implemented in existing multicomputers with simple changes to the hardware. Using simulations, we compare the performance of the proposed column-path algorithm with the previously proposed Hamiltonian-path-based multipath and an *e*-cube-based multicast routing algorithms. Our results show that for multicast traffic, the column-path routing offers higher throughputs, while the multipath algorithm offers lower message latencies. Another result of our study is that the commonly implemented simplistic scheme of sending one copy of a multicast message to each of its destinations exhibits good performance provided the number of destinations is small.

Index Terms—Consumption channels, deadlocks, multicasts, multicomputers, routing algorithms, wormhole routing.

1 Introduction

In a parallel processor, routing algorithms provide mechanisms for communication between processors executing a parallel program. The efficiency of routing algorithms is important for achieving high performance in massively parallel processors (MPPs). The interprocessor communication functions in an MPP are usually handled by a router which receives data from incoming channels and transmits data to outgoing channels using a suitable routing algorithm. The routing functions are often implemented in a distributed manner so that all routers work independently to accomplish the desired communications. Many commercially available parallel computers use hypercube and mesh networks for interprocessor communication with a processor and router module at each node.

Routing algorithms for hypercubes and *k*-ary *n*-cubes have been extensively studied in the context of *one-to-one* or *unicast* communication. In unicast communication, each source sends its message to precisely one destination. Routing algorithms for unicast communication are usually implemented as system calls in parallel machines. More powerful communication primitives that are useful in the execution of parallel programs are *broadcast* and *multicast*, which allow data to be transferred from one source node to many destinations. Broadcasts and limited forms of multicasts are supported on some commercial machines. For example, nCUBE-2 supports global broadcast to all nodes and selective multicast when all destinations form a subcube. In other machines, such as the Intel Paragon [12], users can send a multicast to multiple destinations by sending a copy of the message to each destination node.

Multicast messages are useful for efficient execution of parallel programs. For example, blocked matrix multiplications require broadcasting of matrix blocks in the rows of a mesh (see, for example, chapter 5 of [13]). In addition, multicast messages occur in several contexts; for example, when a master node dispatches information to several slave nodes, synchronization of multiple nodes, cache invalidations, etc. [13]. An important question is whether to support multicast communication in a parallel machine or implement multicast communication by other alternatives; these alternatives include using multiple unicast messages for multicast communication and using a broadcast message to accomplish multicasting.

There are several studies on the performance of multicast communication in multicomputer networks [20], [15], [16], [17], [24], [23], [9], [18], [10]. Lin et al. [17], [16] present two multicast algorithms, *dual-path* and *multipath*, based on the Hamiltonian paths in the interconnection network. These are especially suited for *wormhole* switching in low-dimension mesh and torus networks. (Wormhole switching is a form of cut-through switching in which blocked messages are not buffered [7]. Many recent multicomputers and multiprocessors use this form of routing [1], [5], [19], [12].) The previous studies on wormhole multicast communication addressed and solved the

- *R.V. Boppana is with the Division of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249-0667. E-mail: boppana@cs.utsa.edu.*
- *S. Chalasani is with the Dept. of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706-1691. E-mail: suresh@cauchy.ece.wisc.edu.*
- *C.S. Raghavendra is with the Aerospace Corporation, P.O. Box 92957, Los Angeles, CA 90009-2957. E-mail: raghu@aero.org.*

Manuscript received 27 July 1995.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 104619.

issue of deadlocks arising from dependencies on communication channels. For wormhole multicast routing, deadlocks can also occur due to dependencies on consumption channels [18], [24], [3].

In this paper, we show that deadlocks due to dependencies on consumption channels are a fundamental problem in wormhole multicast routing. An example of such a deadlock is shown in Fig. 4, with the presence of two multicast messages and a cyclic dependency for consumption channels. This example is explained in more detail in a later section. To solve this problem of deadlocks on consumption channels, we propose to provide multiple consumption channels in each node. Multiple consumption channels can be implemented using a single physical consumption channel on which several *virtual* consumption channels are time-multiplexed. Our results presented here expand on our earlier work in [4], [3]. Liu and Duato [18] and Panda et al. [24], [23] have independently discovered the consumption channel deadlock problem and have given solutions to specific algorithms. In this paper, we give more general results on this problem and provide sufficiency conditions for the minimum required consumption channels to avoid deadlocks.

Another issue we address in this paper is the compatibility between unicast and multicast routing algorithms. A unicast routing algorithm and a multicast routing algorithm are said to be compatible with each other if there are no deadlocks among the unicast and multicast messages routed by them. Certain unicast routing algorithms are incompatible with some multicast routing algorithms. For example, the *e*-cube algorithm [7] for unicasts is incompatible with the dual-path and multipath algorithms for multicasts by Lin et al. [16]. One approach to handle this problem is to use the same multicast algorithm for unicast routing also. Another approach is to design multicast algorithms that are compatible with the existing unicast algorithm. This approach has been pursued in a few earlier works [24], [3], [11].

In this paper, we present a multicast routing algorithm, called the *column path* algorithm, which is compatible with the *e*-cube algorithm. Since the *e*-cube is a popular routing algorithm used in several recent parallel computers, such as Intel Paragon [12] and Cray T3D [5], designing multicast routing algorithms that are compatible with the *e*-cube algorithm is important. The column path method uses at most two message copies for each column in the mesh that contains one or more destinations of a multicast communication. Messages are routed using the row-column or *e*-cube routing method. The column path algorithm is an example of the path-based, multidestination wormhole routing methods, in which a single message is sent to multiple destinations by specifying the destination addresses in the message header [16], [24]. The column path algorithm is a compromise between the naive technique of sending one copy to each destination of the message and the specialized Hamiltonian-path based multicast techniques, which send a few message copies each with many destinations.

The multicast algorithms proposed in [24], [23], [10] are similar to the proposed column path algorithm. Panda et al. [24], [23] present a general theoretical framework which includes the *e*-cube based multicast routing. It is more flexible than the proposed column path (see Section 2.3) but requires more resources. So, we have compared the performances of this algorithm with the column path using simulations to see the performance implications. Fleury and Fraigniaud [10] have proposed a routing algorithm which groups multiple columns of a mesh into a block and sends one copy of the message to each such block. This result could be used to derive *e*-cube compatible multicast techniques by choosing partial columns as blocks. Fleury and Fraigniaud do not consider, however, deadlocks resulting from consumption channels.

Using simulations, we study the performance of various multicast algorithms with multiple multicasts and with a mixture of multicasts and unicasts. We compare the performance of our column path routing scheme with the multipath [17] and the *e*-cube based multicast algorithm [24]. Our results show that for multicast traffic, the column path routing offers high throughputs, while the multipath algorithm offers lower message latencies. Another result of our study is that the performance of the naive multicast technique is comparable to the sophisticated schemes if the number of destinations is small.

The rest of this paper is organized as follows. Section 2 describes the various multicast algorithms in detail. Section 3 shows several situations in which deadlocks can occur when previously proposed multicast algorithms are used; this section also presents possible solutions for avoiding these deadlocks. Section 4 presents performance results of various algorithms. Section 5 summarizes the results reported in this paper.

2 Multicast Routing Algorithms

In this paper, we consider *k*-radix, two-dimensional meshes with wormhole switching technique. But all the results and discussions can be applied to multidimensional tori and meshes with suitable modifications.

The two dimensions of the mesh are denoted as DIM_1 and DIM_0. The rows of a 2D mesh are numbered from top to bottom 0, 1, …, $k-1$, and the columns are numbered from left to right 0, 1, …, $k-1$. Node x, $0 \le x < k^2$, in a 2D mesh may be represented by a two-tuple (x_1, x_0), where x_1 is the node's row number and x_0 the node's column number in the 2D grid. The hops taken by a message in a row correspond to hops through processors in DIM_0 and hops in a column correspond to hops in DIM_1. In addition, a hop may be a "+" or a "–" hop, depending on the indices of the current node and the next node in the dimension of travel. For example, DIM_{1+} hops correspond to column hops from top to bottom. A communication channel from node x to y is denoted by $[x, y]$.

Since multicast is a complex form of communication, many researchers have addressed the issue of deadlock-free multicast routing for mesh and other networks. The main emphasis of these earlier works has been to devise multicast schemes which ensure that dependencies on communication channels are acyclic. For this purpose, multiple *virtual* channels are sometimes multiplexed on a single physical channel to create many directed virtual networks. The tree-based multicast schemes require multiple virtual channels on each physical channel for deadlock free routing [16].

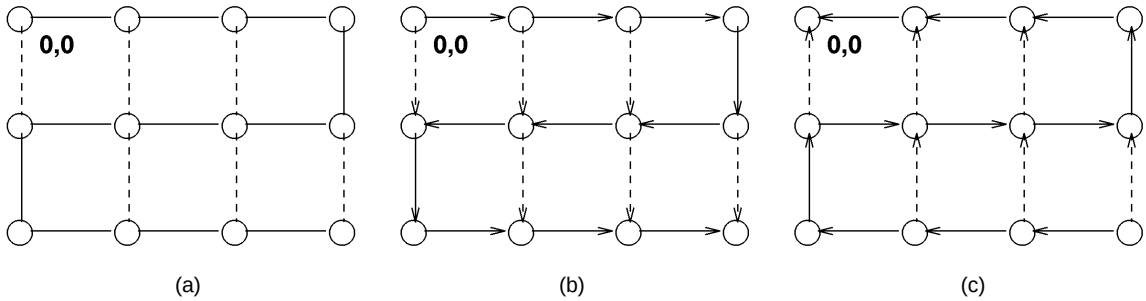

Fig. 1. Example of (a) an undirected Hamiltonian path and the corresponding (b) H_u and (c) H_l directed networks of a mesh. The solid lines indicate the Hamiltonian path and dashed lines indicate the links that could be used to reduce path lengths in message routing.

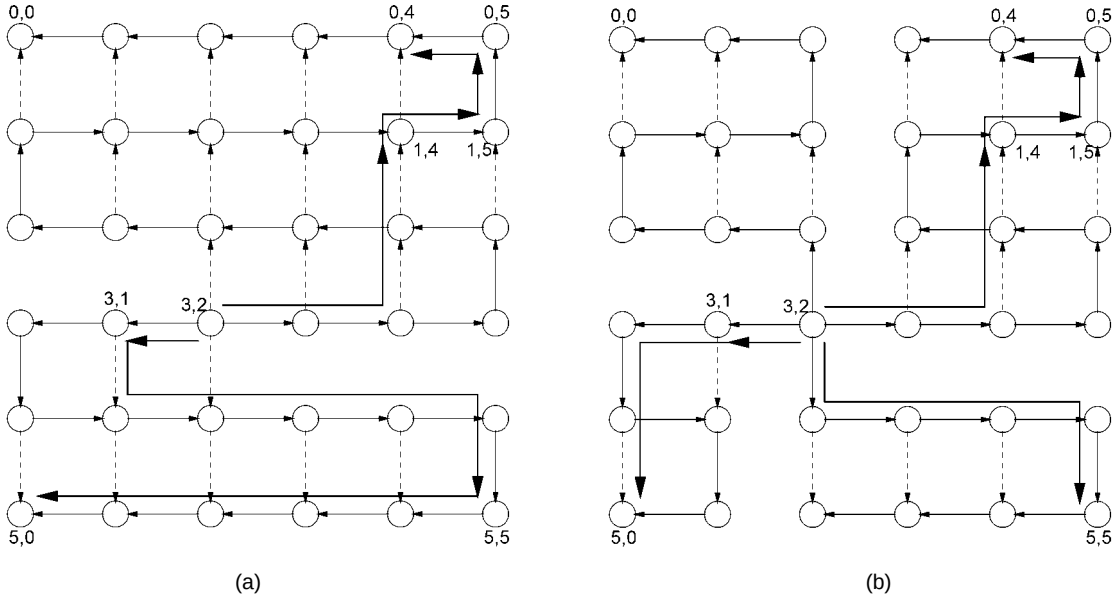

Fig. 2.The subnetworks of H_u and H_l used by multicast messages from node (3, 2) in (a) dual-path routing and (b) multipath routing. Also shown are the paths of an example multicast message from (3, 2). For clarity, the links unused by multicasts from (3, 2) are not indicated.

One naive algorithm for multicast routing is to generate several unicast messages, one for each destination of the multicast message, and route these unicast messages independently through the network. This algorithm, called *individual*, has the potential to perform well when the average number of destinations for each multicast message is small (see Section 4). In addition, routing of each individual message can take place using unicast routing and, hence, there cannot be additional deadlocks (for example, due to consumption channels) in this solution. Many recent distributed shared memory multiprocessors use this technique to perform cache invalidations in directory schemes [1], [14].

In this section, we describe a few important multicast algorithms that have been recently proposed and a new algorithm, which we call the *column-path* algorithm. Our discussion in this section is primarily on the avoidance of deadlocks due to dependencies on communication channels. The issue of deadlocks due to dependencies on consumption channels is addressed in the next section.

2.1 Hamiltonian-Path-Based Multicast Routing

In this method, first an undirected Hamiltonian path, which goes through each node exactly once, of the network is constructed. An example of an undirected Hamiltonian path, with node (0, 0) as an end node, is given in Fig. 1. From this, two directed Hamiltonian paths can be constructed: One starts at (0, 0), the H_u network, and another ends at (0, 0), the H_l network, as shown in Fig. 1. The links that are not part of the Hamiltonian path may be used in the appropriate directions to reduce path length.

The dual-path and multipath algorithms by Lin et al. [16], [17] are based on the acyclic directed networks thus formed. The destinations of a multicast message are partitioned into a small number of subsets (at most four), and a copy of the multicast message is sent to each subset of destinations. Each copy of the message visits its destinations in a predefined order, which is determined by the positions of the destinations in the Hamiltonian path. Different copies of a multicast message use disjoint sets of physical channels and are routed independently of one another. The path-based schemes attempt to alleviate the congestion and deadlocks introduced by tree-based multicast algorithms, albeit by using longer paths.

In the dual-path algorithm, multicast messages from a node are transmitted on appropriate parts of the H_u and H_l networks. Fig. 2a illustrates the portions of H_u and H_l networks used by node (3, 2) to send its multicast messages. Therefore, the destinations of a multicast message are placed into two groups. One group has all the destinations that can be reached from the source node using the H_u

network, and the other has the remaining destinations, which can be reached using the H_l network. As an example, consider a multicast message from node (3, 2) to destinations (5, 0), (3, 1), (0, 4), (1, 4), (0, 5), (1, 5), and (5, 5). In the dual-path method, two message copies are created: One copy services destinations (1, 4), (1, 5), (0, 5), and (0, 4) in sequence, and the other serves destinations (3, 1), (5, 5), and (5, 0). For shorter paths, vertical channels that are not part of the Hamiltonian paths are used appropriately. The use of short-cuts for the example multicast is shown in Fig. 2a.

The dual-path algorithm uses at most two copies of the message for multicast communication. This may increase the path length for some multicast messages. The multipath algorithm attempts to reduce path lengths by using up to four copies ($2n$ for n-dimensional meshes) of the original multicast message. As per the multipath routing algorithm, all the destinations of the multicast message are grouped into four disjoint subsets such that all the destinations in a subset are in one of the four quadrants when source is taken as the origin. The copies of the message are routed using the dual-path routing rules. For a complete description, see [16]. Fig. 2b indicates the routing of the example multicast message from (3, 2) using the multipath algorithm.

The dual-path and multipath schemes provide deadlock-free routing of multicast messages. Further, they also provide minimal routing of unicast messages, since vertical links are used for shortcuts. Therefore, either scheme can be used to route unicast and multicast messages simultaneously in a common framework.

2.2 Column-Path Multicast Routing

The dual-path and multipath schemes are not compatible with the well-known *e*-cube routing algorithm. In this paper, we are interested in designing multicast algorithms that are compatible with the *e*-cube routing. This led us to investigate other algorithms that use the *e*-cube as the base technique for unicast routing.

The column-path algorithm partitions the set of destinations of a multicast message into at most $2k$ subsets (k is the number of columns in the mesh), such that there are at most two messages directed to each column. If a column of the mesh has one or more destinations in the same row or in rows above that of the source, then one copy of the message is sent to service all those destinations. Similarly, if a column has one or more destinations in the rows below that of the source, then one copy of the message is sent to service all those destinations. One copy of the message is sent to a column if all destinations in that column are either below or above the source node; otherwise, two messages are sent to that column.

Let us consider the example multicast message from node (3, 2) to destinations (5, 0), (3, 1), (0, 4), (1, 4), (0, 5), (1, 5), and (5, 5), used for Hamiltonian-path-based algorithms. The paths used under the column path are shown in Fig. 3. Five copies of the message are used to achieve the desired multicast operation. Though destinations (0, 5), (1, 5), and (5, 5) are in the same column, two message copies are sent to this column, since two of the destinations are above the source node's row and the other below. Each of these messages is routed using the *e*-cube (or row-column) routing algorithm. Therefore, the column-path routing is compatible with the unicast routing method used in the current parallel computers.

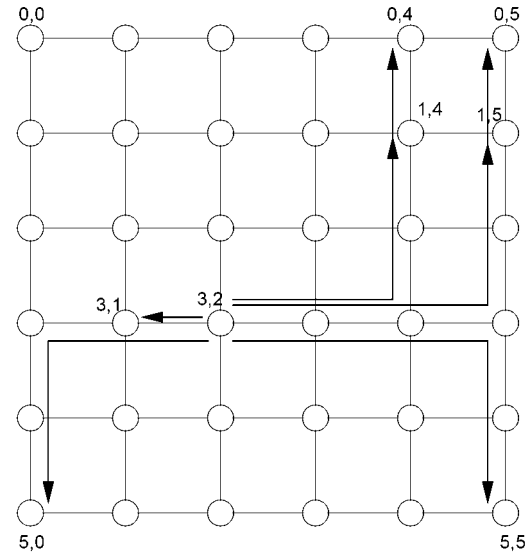

Fig. 3. Example of column path routing.

2.3 Multicast Routing Conforming to Base Routing

Panda et al. [24], [23] have proposed a general technique to provide multicast routing capability using the existing unicast routing method. The unicast routing algorithm can be the *e*-cube or an adaptive algorithm. As in the path-based algorithms described above, the set of destinations of a multicast message are partitioned such that each subset of the destinations can be serviced by a single message. Each such message visits its destinations in a specific order such that its entire path is a valid path for a unicast message between the source of the multicast and the last destination of the message.

As an example of this approach, let us consider the *e*-cube based multicast scheme (denoted, *e*-mcast). With the *e*-mcast algorithm, a message may service any destinations that may lie in its row path to its destinations in a column. For the example multicast in Fig. 3, the *e*-mcast algorithm uses just four message copies—the message to (5, 0) can service destination (3, 1), since it is in its row path. In contrast, no message in the column path scheme can service destinations in two different columns. This is the main difference between the column path and *e*-mcast algorithms. Because a multicast message in this scheme can service destinations in more ways, *e*-mcast tends to reduce the number of messages used compared to the column path. However, this also creates more dependencies on other resources, such as consumption channels.

In the next section, we describe another form of dependencies that are specific to multicasts in wormhole switching.

3 Consumption Channel Dependencies

A common assumption, called *consumption assumption* in [21] and also used here, is that when a flit of a message reaches its destination, it is consumed in finite time. With

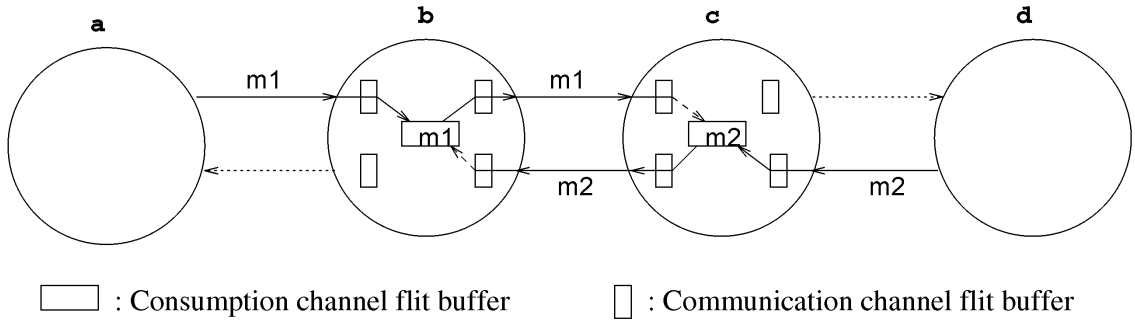

Fig. 4. Example of deadlocks on consumption channels.

this assumption, the proof of deadlock-free routing of unicast messages by a wormhole switching algorithm is reduced to showing that there do not exist cyclic dependencies on the communication channels during the routing process. There are some recent studies on the performance effects of consumption channels for wormhole switching of unicast messages [2]. However, little attention has been paid to the issue of consumption channels in wormhole multicast routing. We now show that wormhole multicast routing algorithms cause cyclic dependencies on the consumption channels.

3.1 Consumption Channel Deadlocks

To see the deadlocks on consumption channels, consider two-dimensional meshes in which each node has one consumption channel satisfying the consumption assumption. Assume that each router has buffer space to hold only a few (typically, one or two) flits of each message that is destined to its processor or passing through it. A multicast message that reaches one of its destinations and has more destinations to be visited can be handled in two ways:

1) The message reserves the consumption channel in the current destination and then reserves/awaits the communication channel in the path to its next destination;
2) The message is consumed (absorbed and removed from the network) by the router of the current destination node and then retransmitted to its next destination later.

The latter strategy is called absorb-and-retransmit. We show that deadlocks occur in both cases. First, we consider Method 1 of handling messages at intermediate destinations.

Fig. 4 illustrates the routing of two multicast messages in a linear chain, which is a subnetwork of some multicomputer network such as mesh or torus. The first message, $m1$, originates at node a and has destinations b, c; the second message, $m2$, originates at node d and has destinations b, c. Furthermore, nodes a, b, c are left neighbors to b, c, d, respectively. The following scenario is shown in Fig. 4. The message $m1$ obtains the communication channel from a to b, denoted $[a, b]$, consumption channel in b, denoted $Cons_b$, and communication channel $[b, c]$; $m2$ obtains the communication channel $[d, c]$, consumption channel in c, $Cons_c$, and communication channel $[c, b]$. The reservation of the consumption channel is shown by labeling the (flit) buffer associated to it with the name of the message that has reserved it.

Due to flit-level flow control in wormhole switching, node b can accept only the header flit (and a few data flits if buffer depth is more than one flit) of $m1$. Though the consumption channel in b is free, $m1$ cannot be consumed at b until it acquires the consumption channel in c also, since messages are not buffered at intermediate nodes during routing. A similar condition occurs with the consumption channel in c and message $m2$. Therefore, $m1$ waits for the consumption channel occupied by $m2$, and vice versa. This is a circular wait on consumption channels by $m1$ and $m2$, which leads to deadlock.

This discussion motivates the following result on multicast wormhole routing for multicomputer networks.

THEOREM 1. *Let M be a multicomputer network such that each node has a single consumption channel satisfying the consumption assumption. Let F be a wormhole multicast algorithm with the following properties*:

1) *A message can wait for a communication channel after reserving a consumption channel, and*
2) *There exists a set of two or more nodes that multicast messages visit as destinations in different orders.*

Then, the algorithm F can lead to deadlocks on consumption channels when it routes two or more multicast messages simultaneously.

The proof is similar to the discussion given for the deadlock in Fig. 4. The main difference is that b and c may not be neighbors in this general case, so additional dependencies on communication channels between b and c may have to be considered. But dependencies on consumption channels at b and c will be the same.

It is noteworthy that the proposed column path and virtually all the multicast algorithms proposed in the literature [16], [23], [9] satisfy the hypothesis of Theorem 1. The lone exception is the individual scheme.

COROLLARY 1. *The dual-path, multipath, and column path algorithms for mesh networks can cause deadlocks on consumption channels if two or more multicast messages are to be routed simultaneously, when there is only a single consumption channel satisfying the consumption assumption*.

We now show that deadlocks still occur if a message is absorbed and retransmitted at intermediate destinations. We still assume one consumption channel per node satisfying the consumption assumption. Any message that reserves the consumption channel releases the consumption channel only after it is absorbed. Let us consider Fig. 4 once

TABLE 1

Msg.	Source	Dest. List	Resources Obtained	Resources Awaiting
$m1$	a	b, c	$[a, b]$, $Cons_b$	$Retrans_b$
$m3$	a	b, c, d	$Retrans_b$, $[b, c]$	$Cons_c$
$m2$	d	b, c	$[d, c]$, $Cons_c$	$Retrans_c$
$m4$	d	a, b, c	$Retrans_c$, $[c, b]$	$Cons_b$

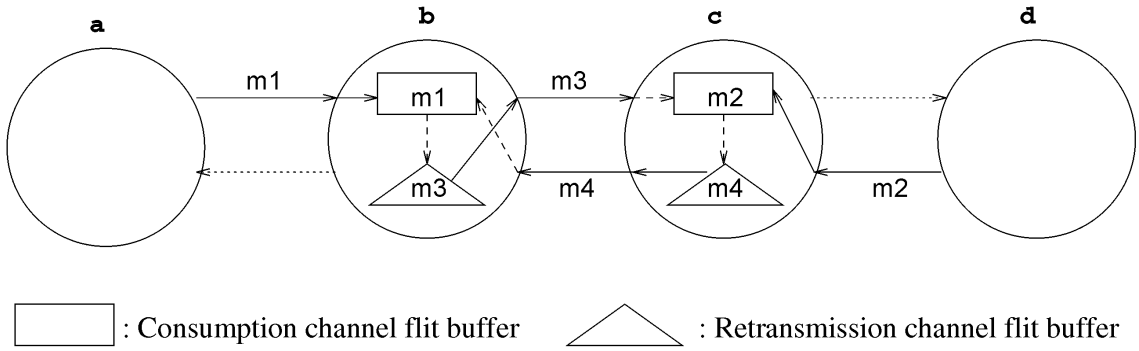

Fig. 5. Example of deadlocks with absorb-and-retransmit strategy. Communication channel flit buffers are not shown for clarity.

again. A plausible approach to avoid deadlocks on consumption channels may be to let a multicast message be absorbed by its destinations. If an absorbed message has more destinations to visit, then it may be retransmitted by the last destination node that absorbed the message. Therefore, a node can buffer two messages completely: one that is being consumed and another that is being transmitted. With the absorb and retransmit strategy for the situation in Fig. 4, $m1$ is absorbed at b and retransmitted to c later, when $m1$ obtains the consumption channel in c; similarly, $m2$ is absorbed at c and retransmitted to b later.

The motivation for absorb and retransmit strategy is that, while intermediate nodes may not have space to store and retransmit a message, it might be feasible to do this at message's destinations. For example, a multicast message may be absorbed at a destination through its consumption channel and retransmitted later via its injection channel.

Though the absorb and retransmit strategy avoids deadlocks for the example in Fig. 4, it does not work in general. Now, deadlocks occur on consumption and retransmission channels. We illustrate this in Fig. 5. There are four multicast messages that need to be routed. The information on source, destination, resources acquired, and resources requested by these messages are given in Table 1. The notations $Cons_x$, $Retrans_x$ are used to indicate, respectively, the consumption and retransmission channels in node x. The rectangle indicates the flit buffer associated with a consumption channel and the triangle indicates the flit buffer associated with the retransmission channel. Fig. 5 shows the following scenario: $m1$ has acquired the communication channel $[a, b]$ and consumption channel in b, and waits for the retransmission channel in b; $m3$ has acquired the retransmission channel in b, communication channel $[b, c]$, and waits for the consumption channel in c; $m2$ has acquired the communication channel $[d, c]$, consumption channel in c, and waits for the retransmission channel in c; $m4$ has acquired the retransmission channel in c, communication channel $[c, b]$, and waits for the consumption channel in b. There is a circular wait for resources by these four messages, and deadlock occurs.

THEOREM 2. *Let the absorb and retransmit strategy be used in routing multicast messages by algorithm F in a 2D mesh. Let m be the maximum number of multicast messages a node can hold, at a time, absorbed messages for later retransmission. Then, deadlocks can occur if* $2m + 2$ *or more multicast messages are in progress simultaneously.*

The above discussion, which proves the theorem for $m = 1$, can be generalized easily.

The absorb and retransmit strategy has been used in the literature for multicasts and for fault-tolerant routing. For example, Suh et al. [25] propose a fault-tolerant routing algorithm in which messages affected by faults are absorbed by an intermediate node and later retransmitted to the final destination. The hierarchical multicast routing in [23] uses one or two levels of leaders which distribute the message data to the nodes in their group. Avoiding deadlocks solely using virtual channels requires a large number of them, since such tree form of communication is not suitable for wormhole switching [22]. So, multicast messages are absorbed by these leaders and retransmitted to their group members. However, the absorb and retransmit strategy is nothing but the well-known store-and-forward routing (with possibly cut-throughs allowed). The buffer space used for absorb and retransmit is a common pool of buffers used by the messages being consumed. It is well-known in store-and-forward routing that, when the finite buffer space is not partitioned and the network is undirected, deadlocks occur. The buffer space for the absorb-and-retransmit can be finite if the absorbed messages stay at the network interface-router portions of a node and do not use the node's local memory. If absorbed messages are allowed to be stored in nodes' local memories, then the deadlock problem is virtually solved; such a solution has the disadvantages of stealing valuable memory bandwidth from the processor and additional message delays. In

summary, if the buffer space is finite and not partitioned, absorb-and-retransmit strategy can lead to deadlocks in the presence of multicast messages.

The consumption deadlocks in wormhole multicast routing can be avoided by replicating and ordering the consumption buffer or channel resources. Since the motivation behind wormhole switching is to reduce the buffer space, we explore the issue of replicating and ordering consumption channels. One way to solve deadlocks on consumption channels is to provide as many or more consumption channels than the maximum number of messages that can be in a node at any time instant. By partitioning and systematically allocating consumption channels, however, deadlocks can be avoided using fewer consumption channels.

3.2 Prevention of Consumption Channel Deadlocks

The above results show that the cyclic dependencies caused by multicast messages on consumption channels is a fundamental limitation for deadlock free wormhole switching. As in the case of deadlocks on communication channels, which are resolved by multiple virtual communication channels, multiple (physical or virtual) consumption channels can be used to resolve deadlocks on consumption channels. We give below some general results and bounds on the number of consumption channels required. Then, we turn our attention to some specific multicast schemes.

We assume that the underlying multicomputer network has regular topology, such as mesh or torus. With the exception of a few boundary nodes, which may have a smaller node degree, most of the nodes have the maximum node degree, which is determined only by the topology. For now, let us assume that only the minimum required number of virtual channels are simulated on each physical channel to avoid deadlocks on communication channels in routing multicast messages by the given multicast routing algorithm. To keep the arguments consistent, we assume that one virtual channel is simulated on each physical channel if the routing algorithm is deadlock free without virtual channels.

DEFINITION 1. *Let IN_x and OUT_x be, respectively, the number of incoming and outgoing virtual channels of node x that can be used by multicast messages to reach or leave node x. Then, v_{in} = Maximum $\{IN_x \mid x$ is a node of the network$\}$ and v_{out} = Maximum $\{OUT_x \mid x$ is a node of the network$\}$.*

LEMMA 1. *Minimum of $\{v_{in}, v_{out} + 1\}$ consumption channels per node are sufficient to avoid deadlocks on consumption channels.*

PROOF. First, let us consider the dependencies by messages arriving at intermediate destinations and need to visit other destinations. We specify that such a message should reserve a consumption channel before reserving its next (communication) virtual channel. Let us associate one consumption channel with each of the v_{out} virtual channels that can be used by multicast messages. So, the dependency caused by a multicast message waiting for or using a consumption channel at one of its intermediate destinations has one-to-one correspondence to the dependency caused by the same message waiting for or using the associated outgoing virtual channel of this node. Since the algorithm is free of deadlocks on communication channels, these dependencies cannot lead to deadlocks. A message that arrives at its final destination node uses no outgoing virtual channels of the node. These messages can wait for or use the $(v_{out} + 1)$th consumption channel without creating any additional dependencies. So, $v_{out} + 1$ consumption channels are sufficient to avoid deadlocks.

Now, we can repeat this argument for the v_{in} incoming virtual channels used by multicast messages to arrive into a node. In this case, we can associate one consumption channel to each incoming virtual channel used by multicast messages, much like flit buffers are associated with incoming virtual channels. This avoids any additional dependencies caused by multicasts reserving consumption channels. So, by this argument, v_{in} consumption channels are sufficient.

Therefore, the minimum of $\{v_{in}, v_{out} + 1\}$ consumption channels are sufficient to avoid deadlocks. □

Lemma 1 gives an upper bound on the number of consumption channels per node and applies to all networks and routing algorithms. When more specific information about the routing algorithm is known, more precise bounds on the number of consumption channels can be obtained.

Suppose F is a multicast routing algorithm such that, for each message routed by F, one or both of the following conditions are true:

- It is possible to further route, without creating communication channel deadlocks, to some node after the message has reached its last destination.
- The last destination of the message is a node with out degree less than v_{out}.

As far as we know, all the proposed wormhole multicast routing algorithms for networks such as meshes satisfy the above property.

COROLLARY 2. *v_{suf} = Minimum $\{v_{in}, v_{out}\}$ consumption channels per node are sufficient for the routing algorithm F.*

PROOF. Let M be a message that has reached a destination, x. If M has further destinations to visit, then it will reserve the consumption channel corresponding to the outgoing channel it will use for its next hop. If x is the last destination of M, then M reserves a consumption channel as follows. If M can be further routed without creating deadlocks, then it can reserve the consumption channel associated with the outgoing channel it can use. It is noteworthy that the message will not be routed any further. The ability to route further is used only to determine the consumption channel it can reserve in its final destination. If $OUT_x < v_{out}$, then it can reserve a consumption channel that is not associated with any of the outgoing channels of x. Therefore, $v_{suf} \leq v_{out}$. The proof for $v_{suf} \leq v_{in}$ is the same as in Lemma 1. □

Given a network, several directed, acyclic networks (DANS) of virtual channels can be constructed. The path of a multicast message, after it reserved a consumption channel, can be partitioned into several segments, with each segment belonging to a specific DAN. Let S be a set of directed,

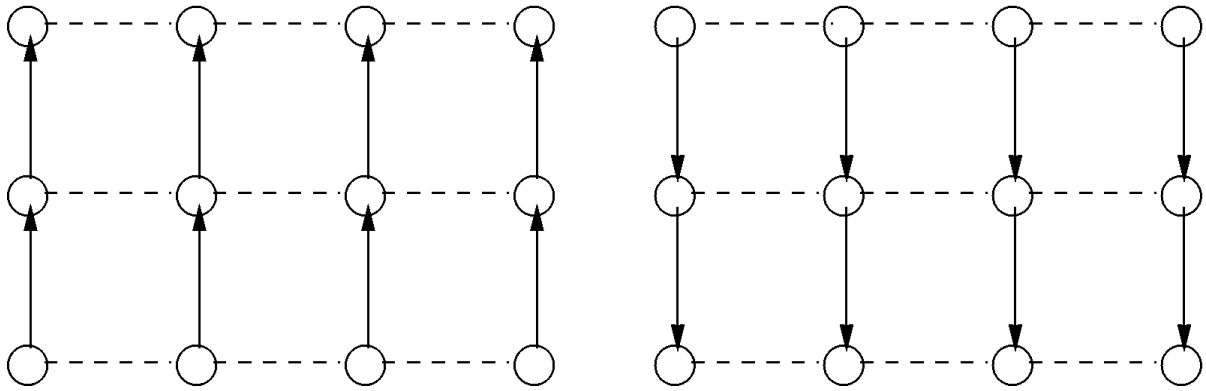

Fig. 6. Directed acyclic networks of the minimal set for column path routing in a mesh. Dashed lines indicate the row links used by messages prior to reserving consumption channels at their first destinations and solid lines with arrows indicate the virtual channels forming a specific DAN.

acyclic networks derived from the original network. We say that there is a dependency from DAN1 $\in S$ to DAN2 $\in S$ if there is a multicast message that needs to use a communication channel in DAN2 after using a communication channel in DAN1. It should be noted that the union of DANs need not contain all virtual channels of the network and that a message may traverse on a DAN even before reaching its first destination.

DEFINITION 2. *Let S be a set of l DANs* {*DAN*1, *DAN*2, …}, *and V be the set of all virtual channels that belong to the DANS of S. We say that S is a valid set if the following properties are satisfied*:

1) *A message that has reserved a consumption channel uses a channel in V for its next hop* (*if any*), *and*
2) *There exists a partial order among the DANs such that each multicast message visits DANs as per the partial order* (*that is, no multicast message violates the partial order while visiting DANs*).

There are several practical routing algorithms for which valid sets of DANS can be constructed easily. Examples of such algorithms include the column path, multipath, and *e*-mcast.

DEFINITION 3. *A valid set of DANS is minimal if it has the least cardinality among all valid sets.*

For a given network, the routing algorithm and the virtual channels used determine the valid and minimal sets of DANS. Often, many valid sets of DANS can be formulated. The number of minimal sets of DANS is much smaller, but a minimal set need not be unique. For 2D meshes and column path routing, the set of all virtual channels of the network is a valid set of DANS. Its minimal set is unique and consists of two DANS: One consisting of all the nodes and only the upward direction column communication channels, and the other consisting of all the nodes and downward direction of column communication channels (see Fig. 6). A multicast message, after reserving its first consumption channel, traverses in only one of the two acyclic networks.

LEMMA 2. *If S is a valid set of DANS for a given routing algorithm, then* |*S*| *consumption channels per node are sufficient to avoid deadlocks on consumption channels.*

PROOF. In each node of the network, there are |*S*| consumption channels. Let us associate a consumption channel with each DAN of the set *S*. A multicast message, *M*, that arrives into one of its destination nodes, *x*, waits for or reserves the consumption channel associated with the DAN in which its most recent hop lies. This may not be possible if the message has not traversed on any DAN, in which case, *x* is the first destination of *M*. If *M* has to visit other destinations, then it will use a channel on a DAN for its next hop. *M* will reserve the consumption channel corresponding to this DAN. If *M* has no other destinations to visit, then it is a unicast message, since *x* is its first and last destination. A unicast message can wait for or reserve any consumption channel in *x*, since it will not create any additional dependencies after it reserves the consumption channel.

To show that deadlocks are avoided, we rank the consumption channels as follows. (The communication channels have been ranked by the routing algorithm, since it avoids deadlocks on communication channels.) A consumption channel of a node is ranked higher than the incoming virtual channels of the corresponding DAN and incoming channels that are not on any DAN and lower than the outgoing virtual channels of the DAN. Therefore, the set of virtual channels of a DAN and the associated consumption channels have a partial order according to which multicast messages acquire them. Since the message paths are acyclic within a DAN, there cannot be deadlocks within a DAN. Since different DANs have different sets of consumption channels used, and there is a partial order according to which messages use DANs, there cannot be deadlocks when all DANs are considered. □

COROLLARY 3. *Two consumption channels are necessary and sufficient for the column path routing algorithm*.

PROOF. Fig. 4 and the related discussion indicate that one consumption channel is not sufficient to avoid deadlocks for the column path algorithm. A valid set with two DANs can be constructed for the column path algorithm. One DAN consists of all the nodes and only the upward direction column communication channels, and the other DAN consists of all the nodes and downward direction column communication channels.

Fig. 6 indicates the DANs for an example mesh network. Hence, from Lemma 2, two consumption channels per node are sufficient to avoid deadlocks on consumption channels. □

The *e*-mcast algorithm (see Section 2.3) allows a message to service destination nodes in its row path while traveling towards a column of destinations. This flexibility seems to come at the cost of increased consumption channel requirement. Panda et al. [23] have shown that four consumption channels per node are sufficient to avoid deadlocks. We confirm this result by giving a valid set of four DANs: DAN1 contains all DIM_{0+} communication channels, DAN2 all DIM_{0-} channels, DAN3 all DIM_{1+} channels, and DAN4 all DIM_{1-} channels.

COROLLARY 4. *Two consumption channels are necessary and sufficient for the dual-path and multipath routing algorithms.*

PROOF. Both dual-path and multipath algorithms have a valid set of two DANs: H_u and H_l. So, from Lemma 2, two consumption channels per node are sufficient. Fig. 4 and the related discussion indicate that one consumption channel cannot avoid deadlocks. □

A unicast message can wait for or use any consumption channel without deadlocks, since it will release all of its resources in finite time.

3.2.1 Using Additional Virtual Communication Channels

Thus far, we assumed that only the minimum required number of virtual channels are used for multicast routing. However, often more than the minimum number of virtual channels are used to improve performance [6] and provide adaptivity [8], [9]. When extra virtual channels are used solely to reduce congestion [6], more than one virtual channel may be used to service messages that need to use a particular virtual channel class. For example, with two virtual channels per physical channel of a 2D mesh and column path routing, a message may use either one of the virtual channels on the physical channel specified by its routing algorithm for its next hop. Then, the number of outgoing virtual channels used by multicast messages is four. So, the number of outgoing virtual channels for multicast messages increases, but the actual number of classes of virtual channels that a message uses remains the same. Therefore, the number of consumption channels required to avoid deadlocks remains the same if each multicast message is required to reserve a consumption channel before reserving its next virtual channel at each of its intermediate destinations. To see the performance benefits of increased virtual channels, however, it may be desirable to increase the number of consumption channels so that there is no excessive congestion on consumption channels.

When extra virtual channels are used to improve the adaptivity of a routing algorithm, it often results in an adaptive algorithm with dynamic transitions [8]. These algorithms have the base algorithm with the required virtual channels to provide deadlock free routing and additional virtual channels with additional routing rules to use them adaptively. The complete channel dependency graphs for these algorithms have cycles, but the dependencies induced on the base virtual channels used for deadlock free routing are acyclic. From Lemma 1, deadlocks on consumption channels can be avoided if $v_{suf} = min\{v_{in}, v_{out} + 1\}$ consumption channels per node are used. The v_{in} and v_{out} are the maximum node in degree and out degree calculated with all virtual channels considered.

If the adaptive algorithm routes such that all destinations of a multicast message can be served using the base virtual channels only, then v_{in} and v_{out} in Lemma 1 are the maximum node in degree and out degree calculated with only the base virtual channels considered. For such algorithms, the concept of DANs may be applied with some modifications. These modifications are needed, since a message may use adaptive channels in between hops on the channels of a given DAN. To explain this, let us assume that a blocked message waits for a suitable channel from the base set only, but will use any suitable adaptive channel that becomes free in the mean time. (This assumption is consistent with the adaptive routing theory proposed by Duato [8]. However, to facilitate the description of modified DANs, we consider the current channel dependencies rather than the extended channel dependencies discussed by Duato.) The channel dependencies at a given instant of time consist of dependencies resulting from messages waiting for the base channels and the actual use of channels at that time instant. So, we define base DANs that contain virtual channels from the base set. The base DANs have a permanent structure. For each base DAN, we define a current DAN that contains all the channels of its base DAN and the adaptive channels that are *actually being used* by the messages traveling in the DAN at that instant of time. Since the routing algorithm is deadlock-free, the use of adaptive channels is such that they appear as alternative paths, without creating cycles, in the current DANs.

It is noteworthy that such adaptive routing does not reduce the number of message copies used, but can improve the performance by using adaptivity to reduce source congestion. For an example, consider the *e*-cube-based adaptive routing with two virtual channels per physical channel in a mesh network. One virtual channel is used for *e*-cube routing and the other for adaptive routing. When a message finds the channel given by the base *e*-cube routing busy, it may use an adaptive channel that takes the message closer to its destination. This does not induce cyclic dependencies on the base virtual channels, however. So, four consumption channels per node are sufficient to avoid deadlocks. If messages are created such that all destinations lie in a column, then only two consumption channels are sufficient.

We illustrate this using the example multicast in Fig. 3. With the column path as the base deadlock-free routing method, five messages are still used. However, each message copy can be routed adaptively from the source of the multicast to its first destination. With the *e*-mcast as the base deadlock-free routing method, four message copies are used (see Section 2.3). A message copy is routed adaptively from the last destination in its row path (or source if it has no row destinations) to its first destination in a column. For example, the message copy serving destinations (3, 1) and (5, 0) has adaptivity after reserving a consumption channel in (3, 1), while the message copy serving destinations (0, 4) and (1, 4) has adaptivity from the source to its first destination (1, 4).

4 Simulation Study

To study the performance issues, we have developed a flit-level simulator. This simulator can be used for wormhole switching in meshes and tori for unicast and multicast traffic. We present performance results for the multipath, individual, column-path, and *e*-mcast algorithms.

We have simulated an 8×8 mesh with uniform traffic pattern and message interarrival times using the geometric distribution. In addition, we have used message sizes of 20, 100, 500, and 1,000 flits. We have varied the number of destinations from 10 to 20 to 30. For deadlock free routing, one virtual channel per physical channel is sufficient for all the four algorithms simulated. We have conducted simulations with one, two, and four virtual channels per physical channel, since we are interested in finding performance improvements with multiple channels [6]. The virtual channels on a physical channel are demand time-multiplexed, and it takes one cycle to transfer a flit on a physical channel. In all cases, each physical channel is provided with eight flits of storage space. So, a virtual channel is provided with a buffer of size eight, four, or two flits, depending on one, two, or four virtual channels are simulated on a physical channel. We have used node delays of three cycles to process the header of a message, and two cycles for data flits cutting-through intermediate nodes in their paths.

To avoid consumption channel deadlocks, the *e*-mcast algorithm requires four consumption channels per node [23], the column path and multipath two, and the individual one. So, we have conducted simulations with two and four consumption channels per node. With two consumption channels per node, we have simulated only the multipath, individual, and column-path algorithms, since the *e*-mcast algorithm requires at least four consumption channels for deadlock avoidance. A multicast message can use any available consumption channel with the individual algorithm, but only a specific one, as given by its type, with the column path and multipath algorithms. (See Corollaries 3 and 4.) With four consumption channels, we have simulated all four algorithms. In the *e*-mcast approach, the four consumption channels are partitioned into four distinct classes: 0 through 3. Upon reaching a destination, a multicast message uses a specific class corresponding to the most recent physical channel used—DIM_{0+}, DIM_{0-}, DIM_{1+}, and DIM_{1-}. With multipath and column path algorithms, two of the four consumption channels are dedicated for the two distinct types of multicast messages. The remaining two may be used by either type of message depending on the availability.

Our simulations model the source congestion caused by the multicast communication accurately. In some of our simulations, we have modeled the message injection or preparation delay, which consists of partitioning the destination set into appropriate subsets and creating multiple copies of the message (as needed, depending on the algorithm); this requires modeling the delay in the processor, memory and network interface components of a node. To account for the message preparation time, we have run additional simulations with two sets of injection delays. In one case, the multipath, column-path, and *e*-mcast were simulated with injection delays of 25, 50, and 50 cycles, respectively. In the other case, they are simulated with 50, 100, and 100 cycles. In both cases, the individual algorithm has an injection delay of five cycles. These delays represent the relative times required by different algorithms to partition destinations into distinct subsets. The column path and the *e*-mcast algorithms have the most complicated procedures to partition the destinations, while the individual has the easiest procedure. The impact of creating multiple copies on message latencies depends to a large extent on how these copies are created. For example, if all copies must be generated before any of them can leave the source node, then all algorithms incur huge delays. The best among the four algorithms will be the multipath, since it requires fewer copies to be generated. On the other hand, one can assume that the time to prepare the first few copies (say, two) is significant, and that the other copies can be prepared while the first few copies are routed out of the source node. This is a reasonable assumption since, in any case, the later copies need to wait until the first few copies are routed out of the source node. In other words, preparation of later copies and routing of earlier copies can take place simultaneously to reduce congestion at the source node. Under this assumption, all four multicast algorithms will have very similar message copying delays. Hence, we did not model the delay to create message copies in our simulations.

We use throughput and average message latency as the performance metrics. Usually, throughput is the number of messages delivered per cycle (MPC) by the network. (A multicast message is said to be delivered only if each and every copy of the message has been consumed by their respective last destinations.) We have used a slightly modified version of the throughput to facilitate some form of comparison among simulations with varying number of destinations and message sizes. The throughput we have used is the product of MPC, average number of destinations, and message size. For example, if a multicast message of 20 flits with 10 destinations has been delivered to all of its destinations, then its contribution to the throughput is $20 \times 10 = 200$. The number of copies used or hops taken to achieve this multicast operation are not used in throughput computations. The message latency is defined as the number of cycles elapsed from the time a message is injected to the earliest time when each and every destination of the message has received a copy. The latency includes the time spent in the source by messages and captures the congestion at the source of a multicast message for algorithms such as the column path, which uses multiple copies.

Each latency and throughput value reported in the simulation results is obtained by averaging the values from at least four simulation samples, each with distinct random number sequences. The 95 percent confidence interval widths for latencies and throughputs prior to saturation are within 10 percent of the reported mean values.

4.1 Performance Under Multicast Traffic

We have conducted several simulations with multicast traffic only. This traffic pattern is not representative of any realistic communication in MPPs. The purpose is to see how well a routing algorithm performs under intensive multicast traffic and also to provide a basis for comparison with the results reported in literature.

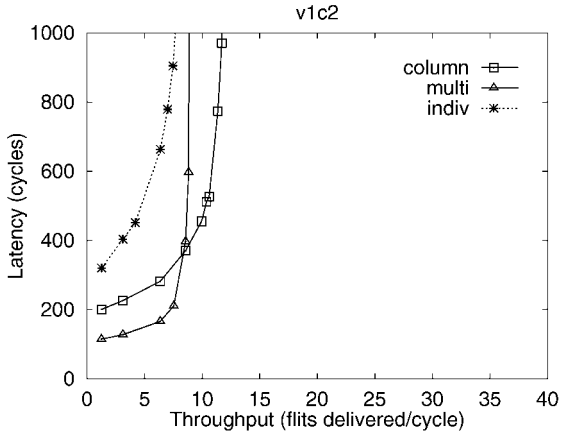

Fig. 7. Performance for all multicast traffic with one virtual channel per physical channel, two consumption channels per node, 20-flit messages, an average of 10 destinations per multicast, and no injection delays.

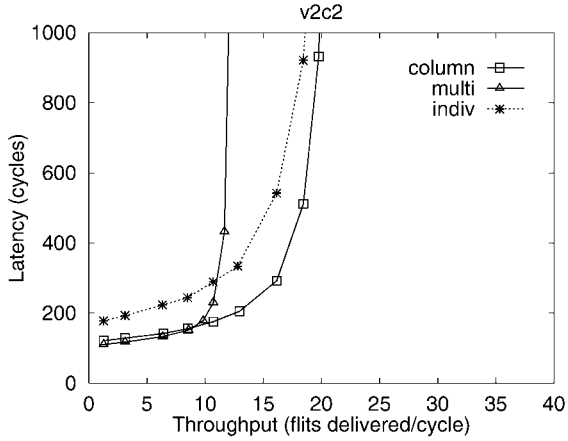

Fig. 8. Performance for all multicast traffic with two virtual channels per physical channel, two consumption channels per node, 20-flit messages, an average of 10 destinations per multicast, and no injection delays.

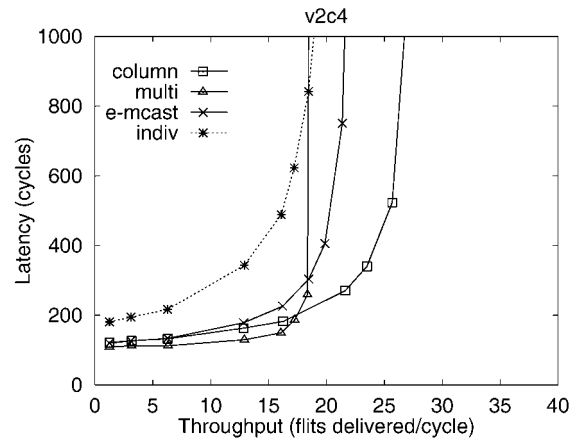

Fig. 9. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 20-flit messages, an average of 10 destinations per multicast, and no injection delays.

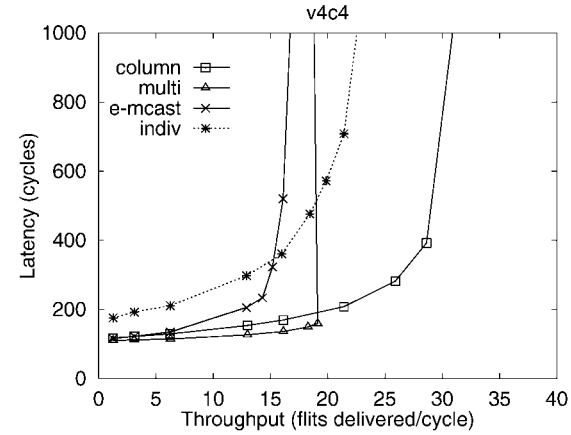

Fig. 10. Performance for all multicast traffic with four virtual channels per physical channel, four consumption channels per node, 20-flit messages, an average of 10 destinations per multicast, and no injection delays.

First, we have fixed the number of average destinations at 10—uniformly distributed with a range of 1 through 19—and the message size at 20 flits. The destinations are randomly selected. We did not use any injection delay for these initial simulations. We have used one, two, and four virtual channels per physical channel, and two, and four consumption channels per node. Figs. 7, 8, 9, and 10 contain the results of these simulations. The multipath algorithm has much lower latencies than the other three algorithms. The higher latencies for the other three algorithms are mainly due to source congestion.

We use the notation v*ic j* to indicate the simulation results with i virtual channels per physical channel and j consumption channels per node. Comparing the v1c2 (Fig. 7) and v2c2 (Fig. 8) cases, we see that increasing the virtual channels yields improved throughput for the individual and column path algorithms. To understand this observation, we define the v-c ratio as the ratio of the number of virtual channels to the number of consumption channels. In an 8×8 mesh, there are 64 nodes and 448 physical channels. For the v1c2 simulations, $64 \times 2 = 128$ consumption channels are used and $448 \times 1 = 448$ virtual channels are used. Thus, the v-c ratio for this case is $448/128 = 3.5$. For the v2c2 case, the provided v-c ratio is seven. The v-c ratio used by an algorithm is simply the average number of hops it uses to serve a destination. The individual, column path, *e*-mcast, and multipath algorithms use an average of 5.35, 3.76, 3.72, and 2.81 hops, respectively, to serve a destination. (These numbers are obtained from the simulations used in this study.) If the used v-c ratio differs substantially from the v-c ratio provided by the network, then either virtual channel or consumption channel congestion is likely to occur. We believe that the v-c ratio may be used as a rule of thumb in designing a network. However, the optimal ratio depends on the traffic that needs to be handled by the network.

For the v1c2 case (Fig. 7), the provided v-c ratio (which is 3.5) and the used v-c ratios indicate that the virtual channel congestion is more severe with the individual and column path algorithms. So, increasing the number of virtual channels provides the most benefit for these two algorithms. Similar observations hold for the v2c4 and v4c4 cases. The performance of *e*-mcast degrades with four virtual channels, since the v-c ratio changes from 3.5 to 7.0. Since each consumption channel in the *e*-mcast algorithm may be used only by a specific type of incoming message, congestion on consumption channels at destination nodes results in wastage

of the virtual channel bandwidth. This results in performance degradation. Additional experiments with lower provided v-c ratios indicate that *e*-mcast does improve its performance from the v1c4 case (not reported here), which has a v-c ratio of 1.75, to the v2c4 case. Destination congestion is not a severe problem for the column path, since it has two consumption channels used for breaking deadlocks and two consumption channels shared among all incoming messages.

The column path offers the following advantages. Compared to the multipath, message copies use shorter paths; so, resources are held for shorter times, which leads to better throughputs. Compared to the *e*-mcast, fewer consumption channels are needed to avoid deadlocks; so, it performs well even with a small number of consumption channels. Compared to the individual, fewer message copies are used; so, source congestion is reduced. Its disadvantages include longer header preparation time compared to the individual and multipath and more severe source congestion compared to multipath, which lead to higher latencies. The *e*-mcast is similar to the column path. However, it requires more consumption channels to see any performance benefits over the column path.

To see the performance variation with respect to the number of destinations, message sizes, and injection delays, we have conducted additional simulations by fixing the number of virtual channels per physical channel at two, and the number of consumption channels per node at four. We have used only the column path, *e*-mcast, and multipath algorithms in these simulations. Figs. 11 and 12 report the simulations with a message size of 20 flits, 0-cycle injection delay, and an average of 20 or 30 destinations (with uniform distribution) per message. As before, destinations are randomly selected. Compared to the case with 10 destinations given in Fig. 9, the throughput increases substantially, since fewer hops are used per destination. The multipath offers lower latencies, and the column path offers substantially higher throughputs.

In the next set of simulations, we have fixed the average number of destinations at 10 and varied the message sizes from 20 to 100, 500, and 1,000 flits. Fig. 9 gives the results for 20-flit messages, while Figs. 13, 14, and 15 report these results for 100-, 500-, and 1,000-flit messages, respectively. With increasing message sizes, the multipath becomes more attractive in terms of latencies. With respect to throughput, however, the column path or *e*-mcast are preferable.

To see the impact of injection delays on message latencies and throughputs, we have conducted simulations by modeling the header preparation time as an additional delay incurred at the time of injecting a multicast message into the network. Fig. 16 gives the results with injection delays for the 5-25-50 case (injection delays of 5, 25, 50, and 50 cycles are assumed for the individual, multipath, column path, and *e*-mcast algorithms). Fig. 17 gives the results for the 5-50-100 case (5, 50, 100, and 100 cycles of injection delay for the individual, multipath, column path, and *e*-mcast algorithms). All algorithms have latencies closer to the individual. If the header partition time is large but creating message copies is small or negligible, then individual is competitive with the more sophisticated multicast techniques. If creating message copies requires a large amount of time, then the multipath algorithm performs better than

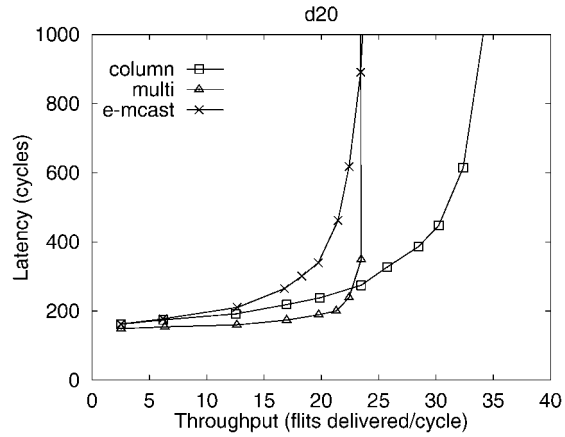

Fig. 11. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 20-flit messages, no injection delays, and an average of 20 destinations per multicast.

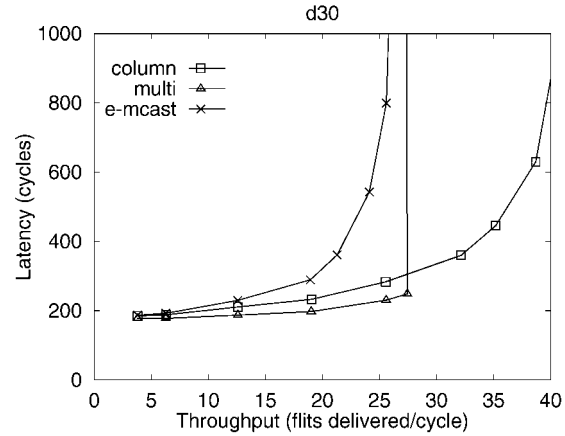

Fig. 12. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 20-flit messages, no injection delays, and an average of 30 destinations per multicast.

the other algorithms. If throughput is the main criterion of performance, the column path is a better choice.

4.2 Performance Under Unicast and Multicast Traffic

In the final set of simulations, we have considered a mixture of unicast and multicast traffic; 90 percent of all messages injected are unicast messages and the rest 10 percent are multicast messages. A multicast message has an average of five destinations (modeled using uniform distribution). This traffic pattern could be representative of communication in cache-coherent shared memory multiprocessors; the majority of traffic is due to remote fetches of cache blocks and the rest is a mixture of traffic for invalidations of shared cache blocks and synchronizations.

Unicast messages are also routed using the multicast technique simulated. Using one type of algorithm for unicast and another for multicast (for example, *e*-cube for unicasts and multipath for multicasts) could lead to deadlocks on virtual channels. We have used two virtual channels per physical channel, four consumption channels per node, 20-flit messages, and 5-25-50 injection delays. The corresponding simulation results are given in Fig. 18. In this case, all algorithms perform similarly, with multipath having a slightly lower latency and lower throughput. The individual

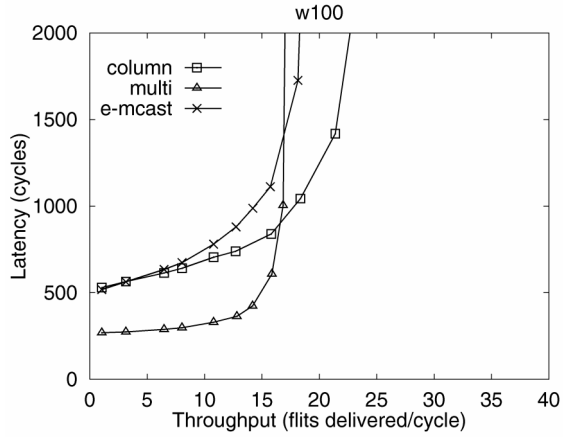

Fig. 13. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 100-flit messages, an average of 10 destinations per multicast, and no injection delays.

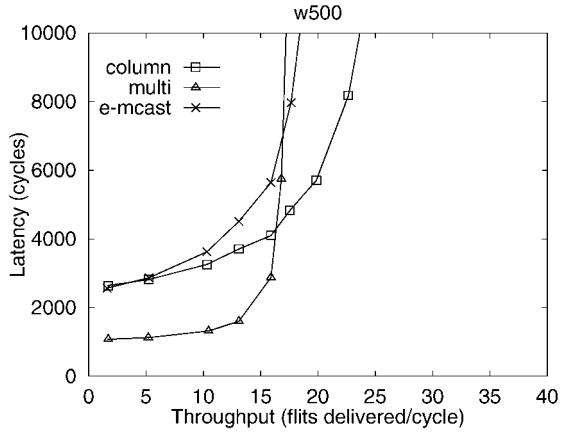

Fig. 14. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 500-flit messages, an average of 10 destinations per multicast, and no injection delays.

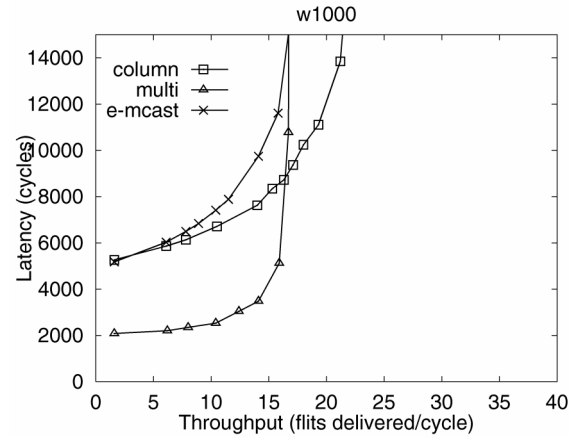

Fig. 15. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 1,000-flit messages, an average of 10 destinations per multicast, and no injection delays.

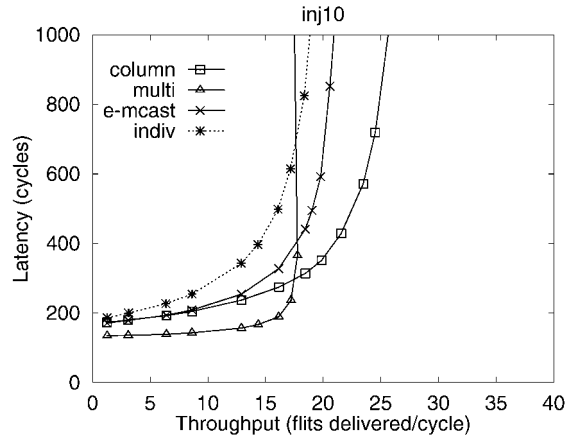

Fig. 16. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 20-flit messages, an average of 10 destinations per multicast and injection delays of 5, 25, 50, and 50 cycles for the individual, multipath, column path, and *e*-mcast algorithms.

is competitive with the column path and *e*-mcast algorithms in terms of both latency and throughput.

5 Summary and Concluding Remarks

Recently, much attention has been focused on designing deadlock-free routing algorithms for multicast communication. Thus far, only deadlocks due to dependencies on communication channels and buffers has been addressed. In this paper, we have shown that the dependencies on consumption channels are a fundamental issue in the design of multicast algorithms for wormhole networks. Whenever a multicast algorithm allows a message to hold consumption channels and reserve additional communication channels, deadlocks can occur.

We have also shown that deadlocks on consumption channels may be resolved using multiple *virtual* consumption channels time-multiplexed on a single physical consumption channel. This issue is similar to that of avoiding deadlocks on communication channels with multiple virtual channels. We have given upper bounds on the number of consumption channels required to avoid deadlocks.

Specialized multicast routing algorithms tend to be incompatible with *e*-cube routing and cannot take advantage of the results and techniques developed for unicast routing. Our approach in this paper has been to use simple techniques that are compatible with *e*-cube to provide multicast routing. We have considered two simple multicast routing algorithms for mesh networks that are *deadlock free* when two or more virtual consumption channels are used. One of them, called *individual*, uses one unicast message for each destination of a multicast message. This is the most commonly used method for multicast communication in the current parallel processors. In order to contain the problem of the number of copies generated for a multicast operation by the individual, we have proposed a new approach called the *column path* routing. The column path follows row-column or *e*-cube routing method and, hence, is compatible with the routing mechanism in, for example, the Intel Paragon and Cray T3D parallel computers.

The individual, Hamiltonian-path-based, dual-path, and multipath, and the column path algorithms offer different trade-offs in terms of performance and implementation. The individual is simplest to use but causes congestion at sources of multicast messages if each message has many destinations. The column path requires simple changes to

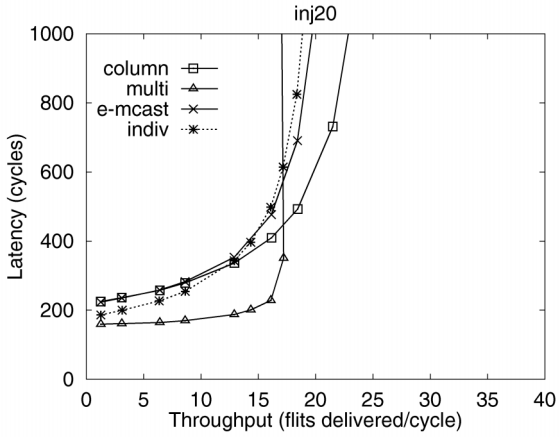

Fig. 17. Performance for all multicast traffic with two virtual channels per physical channel, four consumption channels per node, 20-flit messages, an average of 10 destinations per multicast and injection delays of 5, 50, 100, and 100 cycles for the individual, multipath, column path, and *e*-mcast algorithms.

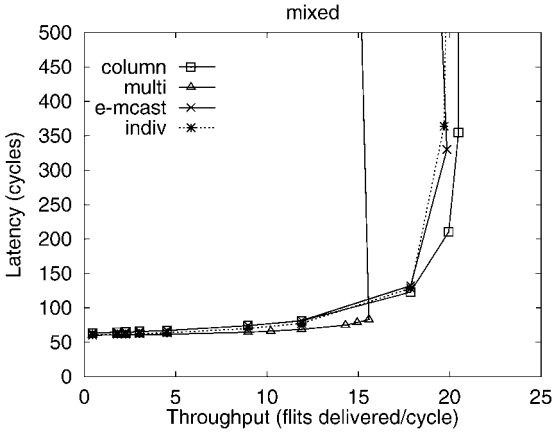

Fig. 18. Performance for "90 percent unicast + 10 percent multicast" traffic. Two virtual channels per physical channel, four consumption channels per node, 20-flit messages, an average of five destinations per multicast, and injection delays of 5, 25, 50, and 50 cycles for the individual, multipath, column path, and *e*-mcast algorithms are used.

e-cube routing and two virtual or physical consumption channels for deadlock free operation. But it reduces source congestion substantially. The dual-path and multipath algorithms virtually eliminate source congestion but tend to use long message paths. In the worst case, a 2D mesh is used as a collection of linear chains. The column path algorithm is a compromise between the individual and the Hamiltonian-path techniques.

We have investigated the performances of four algorithms— individual, multipath, and an *e*-cube based multicast (*e*-mcast)—for multiple multicasts and a mixture of multicasts and unicasts using simulations. Our results show that, in most cases, the column-path routing offers higher throughputs compared to the other routing algorithms. In terms of metrics, such as average additional traffic (used in [17]), individual, column path, and *e*-mcast are worse than the Hamiltonian-path-based, dual-path, and multipath algorithms. But in terms of metrics, such as network throughput, the column path performs better than the dual-path and multipath algorithms. (See [3] for a comparison of the column path and dual-path algorithms.) However, column path, *e*-mcast, and individual incur higher average latencies than multipath; this increase in average latencies for column path and *e*-mcast is partly due to the higher message preparation latencies (since destinations need to be sorted in a specific order), and partly due to source congestion resulting from the use of multiple message copies. So, the multipath algorithm may be a good choice when the average latency is important (as when invalidation messages in shared memory multiprocessors are sent as multicast messages). For infrequent multicasts with a small number of destinations, however, our simulations show that the simplistic approach of sending one copy to each destination performs as well as the other schemes studied in this paper. In such cases, implementing multicast protocols in hardware may not be necessary.

Our results can be easily extended to multidimensional meshes and tori. The column path algorithm is based on a form of multicast primitive: column broadcast. In future, we will consider other multicast primitives, such as subcube broadcasting to design better multicast algorithms. This work can be expanded on in several directions. Theoretical results on lower bounds for consumption channels needed to avoid deadlocks will be interesting. The focus of this paper is *e*-cube compatible multicast routing algorithms. So, an interesting direction for further work is the impact of adaptivity on the consumption channel requirements and on reducing the source congestion for the column path and related algorithms.

ACKNOWLEDGMENTS

Dr. Boppana's research has been partially supported by DOD/AFOSR Grant F49620-96-190472 and U.S. National Science Foundation Grants CCR-9208784 and CDA-9633299. Dr. Chalasani's research has been supported in part by a grant from the Graduate School of UW-Madison and U.S. National Science Foundation Grants CCR-9308966 and ECS-9216308. Dr. Raghavendra's research has been supoorted by U.S. National Science Foundation Grant MIP-9296043. The authors are grateful to Professor D.K. Panda of Ohio State University for his comments on an earlier version of this paper.

REFERENCES

[1] A. Agarwal et al., "The MIT Alewife Machine: Architecture and Performance," *Proc. 22nd Ann. Int'l Symp. Computer Architecture*, June 1995.

[2] S. Balakrishnan and D.K. Panda, "Impact of Multiple Consumption Channels on Wormhole Routed k-Ary n-Cube Networks," *Proc. Seventh Int'l Parallel Processing Symp.*, Apr. 1993.

[3] R.V. Boppana, S. Chalasani, and C.S. Raghavendra, "On Multicast Wormhole Routing in Multicomputers," *Proc. Sixth IEEE Symp. Parallel and Distributed Processing*, Oct. 1994.

[4] R.V. Boppana, S. Chalasani, and C.S. Raghavendra, "On Multicast Wormhole Routing in Multicomputers," manuscript, Jan. 1994.

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

[6] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, Mar. 1992.

[7] 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, May 1987.

[8] 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.

[9] J. Duato, "A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 9, pp. 976–987, Sept. 1995.
[10] E. Fleury and P. Fraigniaud, "Multicasting in Meshes," *Proc. 1994 Int'l Conf. Parallel Processing*, pp. III.151–III.158, 1994.
[11] C.-T. Ho and B. Kao, "Optimal Broadcast on Hypercubes with Wormhole and *e*-Cube Routings," *Proc. Int'l Conf. Parallel and Distributed Systems*, 1993.
[12] *Paragon XP/S Product Overview*. Intel Corporation, 1991.
[13] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Redwood City, Calif.: Benjamin Cummings, 1994.
[14] J. Kuskin et al., "The Stanford FLASH Multiprocessor," *Proc. 21st Ann. Int'l Symp. Computer Architecture*, pp. 302–313, 1994.
[15] Y. Lan, A.-H. Esfhanian, and L.M. Ni, "Multicast in Hypercube Multiprocessors," *J. Parallel and Distributed Computing*, vol. 8, pp. 30–41, 1990.
[16] X. Lin, P.K. McKinley, and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2D Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 8, pp. 793–804, Aug. 1994.
[17] X. Lin and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," *Proc. 18th Ann. Int'l Symp. Computer Architecture*, pp. 116–125, 1991.
[18] Z. Liu and J. Duato, "Adaptive Unicast and Multicast in 3D Mesh Networks," *Proc. 27th Hawaii Int'l Conf. System Sciences*, pp. 173–82, Jan. 1994.
[19] M.D. Noakes et al., "The J-Machine Multicomputer: An Architectural Evaluation," *Proc. 20th Ann. Int'l Symp. Computer Architecture*, pp. 224–235, May 1993.
[20] P.K. McKinley and J.W. Liu, "Multicast Tree Construction in Bus-Based Networks," *Comm. ACM*, vol. 33, pp. 29–42, 1990.
[21] 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.
[22] L.M. Ni, "Should Scalable Parallel Computers Support Efficient Hardware Multicast?" *Proc. Int'l Conf. Parallel Processing Workshop*, pp. 2–7, Aug. 1995.
[23] D.K. Panda, S. Singhal, and R. Keshavan, "Multidestination Message Passing in Wormhole *k*-Ary *n*-Cube Networks with Base Routing Conformed Paths," manuscript, Dec. 1995.
[24] D.K. Panda, S. Singhal, and P. Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme," *Proc. Parallel Computer Routing and Comm. Workshop, Lecture Notes in Computer Science 853*, Springer-Verlag, May 1994.
[25] Y.-J. Suh, B.V. Dao, J. Duato, and S. Yalamanchili, "Software Based Fault-Tolerant Oblivious Routing in Pipelined Networks," *Proc. 1995 Int'l Conf. Parallel Processing*, pp. I.101–I.105, Aug. 1995.

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.

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. Since 1991, he has been a faculty member of the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include parallel architectures, parallel algorithms, and fault-tolerant systems.

C.S. Raghavendra received the BSc (Honors) physics degree from Bangalore University in 1973, and the BE and ME degrees in electronics and communication from the Indian Institute of Science, Bangalore, in 1976 and 1978, respectively. He received the PhD degree in computer science from the University of California at Los Angeles in 1982. From September 1982 to December 1991, he was on the faculty of the Electrical Engineering-Systems Department at the University of Southern California, Los Angeles. From January 1992, he was on the faculty of the School of Electrical Engineering and Computer Science at the Washington State University in Pullman as the Boeing Centennial Chair Professor of Computer Engineering. Presently, he is with the Aerospace Corporation in El Segundo, California. His research interests are in high-speed networks, parallel processing, fault-tolerant computing, and distributed systems.