Performance Evaluation of a New Hardware Supported Multicast Scheme for K-ary N-cubes

Dianne R. Kumar
Computer Science & Eng.
University of Colorado at Denver
Denver, CO 80217-3364
dkumar@carbon.cudenver.edu

Walid A. Najjar
Computer Science & Eng.
University of California Riverside
Riverside, CA 92521-0304
najjar@cs.ucr.edu

Pradip K Srimani
Computer Science
Clemson University
Clemson, SC 29634-0974
srimani@cs.clemson.edu

Abstract

In this paper, we propose a new hardware tree-based routing algorithm (HTA) for multicast communication under virtual cut-through switching in k-ary n-cubes that out-performs existing software and hardware path-based multicast routing schemes. Simulation results are compared against several commonly used multicast routing algorithms and show that HTA performs extremely well under many different conditions.

1 Introduction

Efficient routing of multicast messages is extremely important to the performance of multiprocessors. Since most current multiprocessors only support unicast communication, multicast is therefore implemented as multiple unicast messages resulting in high message latencies. Hardware-based multicast support can greatly improve performance. Among the proposed hardware-based schemes for multicast are path-based and tree-based routing algorithms [1, 2, 3, 4, 5, 7]. Each one of these schemes uses multi-destination messages, which are messages that have more than one header flit. The main difference between these two types of multicast schemes lies in how the header flits in a multi-destination message are routed. Path-based techniques only require routing of the first header flit at each node while tree-based requires routing of all header flits.

In tree-based routing, no ordering of the destinations is required before the message is injected into the network and the shortest paths between the source node and all destinations are always taken. However, this type of routing suffers from a high probability of message blocking at intermediate nodes leading to higher deadlock probability. Path-based routing does not suffer from this high probability of message blocking. However, it does require destinations to be ordered at the source and does not always provide the shortest path between the source node and each destination node.

In this paper, we propose a hardware tree-based routing algorithm (HTA) which attempts to reduce the probability of message blocking, resulting in low message latencies. The probability of a message blocking is kept low by using virtual cut-through switching, independent virtual channels (VCs) for unicast and multicast messages, several VCs per physical channel (PC), an efficient deadlock detection and recovery scheme, and delayed header flit routing. The Hardware Tree-based routing algorithm (HTA) is a fully adaptive and minimal tree-based routing scheme for multicast routing. The scheme is fully compatible with existing unicast routing schemes. Although tree-based routing has not been seen in the past as very practical because of large blocking probabilities, we demonstrate that it can be made very competitive by judicial combination of existing techniques in addition to the new concept of delayed header flit routing.

2 Hardware Tree-Based Multicast Routing Algorithm (HTA)

HTA is a routing scheme that combines two distinct routing algorithms, one for unicast communication and one for multicast communication. Both routing algorithms are implemented within the same network and each message is assigned to the appropriate routing algorithm when input into the network. Although k-ary n-cube networks are simulated here, any direct network can be used with HTA as long as it is compatible with the unicast routing algorithm. The main characteristics of HTA are as follows: Virtual cut-through switching is used with distinct virtual paths for unicast and multicast messages, each path using three VCs per dimension (total of six VCs per dimension). Unicast messages are routed using the deadlock-free routing algorithm proposed in [11, 12] (briefly explained in Section 2.1). Any other unicast algorithm that is compatible with the network topology could also be implemented in HTA. Multicast messages are

routed using tree-based, fully adaptive routing along with a deadlock detection and recovery scheme (Sections 2.1 and 2.3) and delayed header flit routing (Section 2.2). Each message is composed of all header flits followed by all data flits. Each header flit holds one destination address and destinations do not need to be ordered within a multicast message.

2.1 The Routing Scheme

Our proposed HTA scheme consists of two separate routing algorithms for unicast and multicast messages, which are described below.

Unicast Communication Routing Algorithm. The deadlock-free routing algorithm proposed in [11, 12] is used for unicast communication in HTA and is an adaptive routing algorithm based on dimension-order routing. In this adaptive routing scheme, a message is routed on any adaptive channel until it is blocked. Once blocked, a message is routed using dimension-order routing if possible. A message may return to the adaptive channels in the following routing decisions if the adaptive channels are available.

When a message is routed using dimension-order routing, it is routed along decreasing dimensions with a dimension decrease occurring only when zero hops remain in all higher dimensions. By assigning an order to the network dimensions, no cycle exists in the channel-dependency graph and the algorithm is deadlock-free. A minimum of three VCs per dimension is required for deadlock-free routing in k-ary n-cubes. Two VCs per dimension are used for dimension-order routing and all remaining VCs are used for adaptive routing.

Multicast Communication Routing Algorithm. Multicast messages are routed through the network using a tree-based, *delayed header flit routing* algorithm along with a deadlock detection and recovery scheme. The first header flit of a multicast message is routed to any free channel using the following *priority scheme*: the message first requests any free channel in the dimension in which it has the greatest distance left to travel. If more than one dimension has the same distance left to travel, a dimension is randomly selected. If there are no free channels within the selected dimension, then any free channel in the dimension with the next furthest distance left to travel is requested. This type of requesting continues until a channel has been assigned to the header flit of this message or until no free channels have been found. If no free channels are found, the header flit at the top of the queue blocks. No other header flits can be routed until this header flit has been routed. After the header flit is routed at the current node, it is then moved to the neighboring node's queue. Because *delayed header flit routing* is used (explained in Section 2.2), the header flit just routed remains at the neighboring queue until all remaining header flits in this multicast message have been routed at the current node.

After the first header flit is routed, all remaining header flits are routed in the same manner as the first header flit, with one exception. When each of the remaining header flits reaches the top of the queue, it is *first* routed (if possible) to any channel *already allocated* to this multicast message by any of the *preceding* header flits that have already been routed. If this header flit cannot be routed in any of the previously routed dimensions, then it is routed using the *priority scheme* described above for the first flit in the message. By trying to route the remaining header flits to already allocated channels for each multicast message, extra channels are only assigned to the multicast message when necessary. This keeps other channels available for other multicast messages in the network and reduces the probability of blocking since less number of channels are assigned per node for each multicast message.

Once all header flits for each multicast message are routed, the data flits for this message are moved simultaneously to all channels allocated to this multicast message. HTA allows full adaptivity for multicast messages since there is no channel routing restriction. To deal with potentially deadlocked situations the deadlock detection and recovery scheme described in Section 2.3 is used. The schematic of the HTA routing algorithm is shown in Figure 1.

2.2 Header Flit Routing

The scheme most commonly used for routing header flits [9, 4] is referred to here as *immediate header flit routing*. To increase performance a new type of scheme is proposed called *delayed header flit routing*.

Immediate Header Flit Routing. When a header flit is routed at the current node and moved to a neighboring queue, it is routed at this neighboring node *without waiting* for the remaining header flits in the message to be routed. Figure 3 shows an example of *immediate header flit routing*. In this figure, header flit $A.2$ is blocked while header flit $A.1$ continues to be routed, holding VCs and causing message B to block.

Delayed Header Flit Routing. HTA uses *delayed header flit routing* to lower the probability of messages blocking and to increase performance. In *delayed header flit routing*, a header flit at a neighboring node is prevented from being routed until all header flits at the current node have been routed.

Because in tree-based routing, the remaining header flits may not be immediately routed, it's more advantageous to keep all header flits within close proximity of one another using *delayed header flit routing*. This close proximity prevents header flits from being assigned to queues at downstream nodes before all flits in the message can use them. This keeps the downstream queues free so that they are

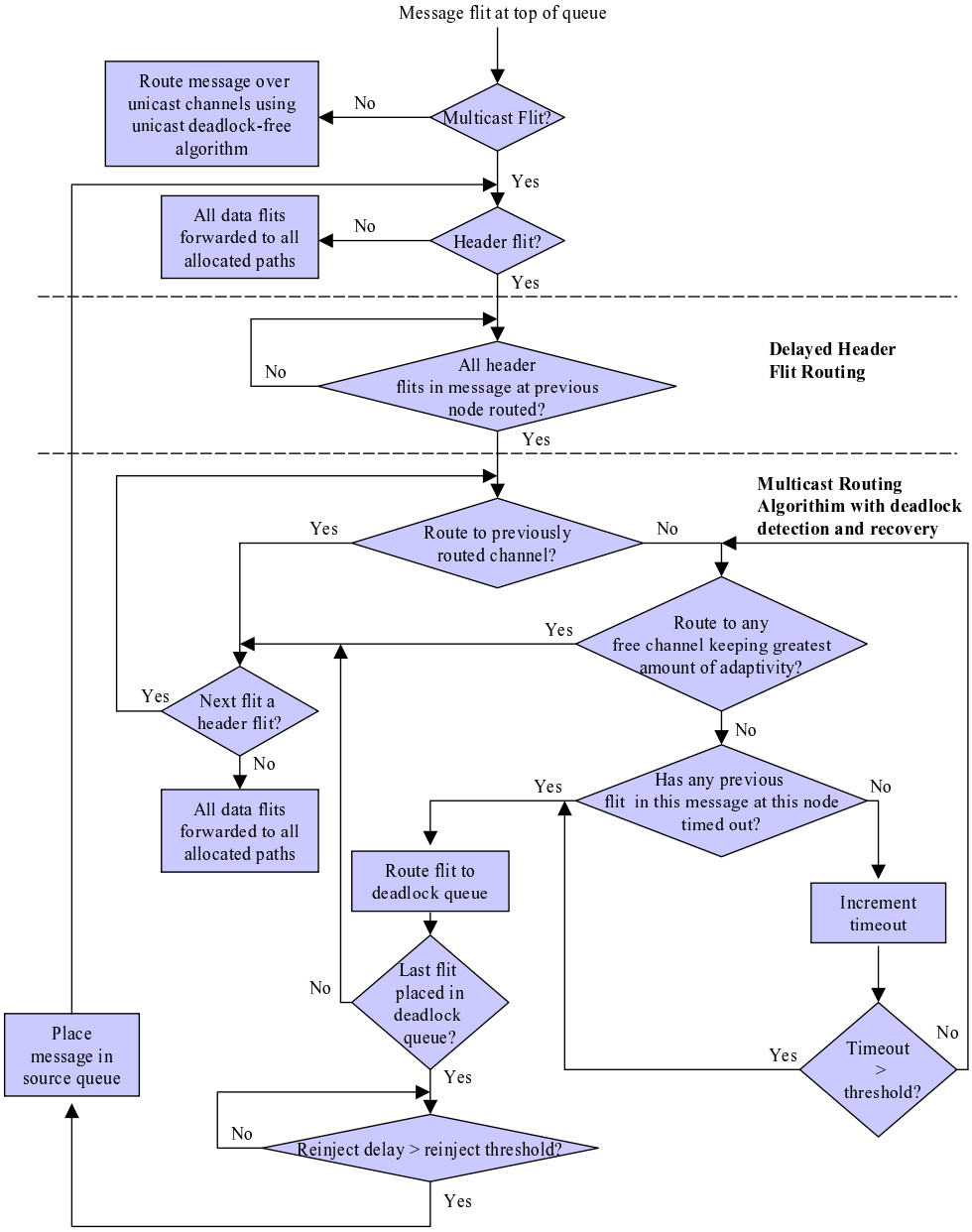

Figure 1. HTA Routing Algorithm for a multicast message

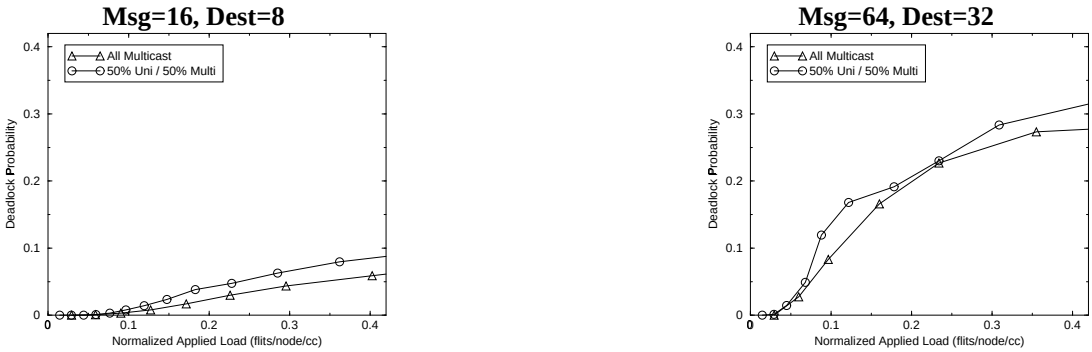

Figure 2. Deadlock Probabilities for an 8-ary 2-cube network

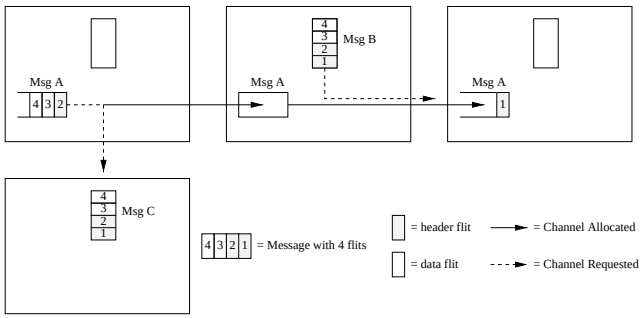

Figure 3. Immediate header flit routing

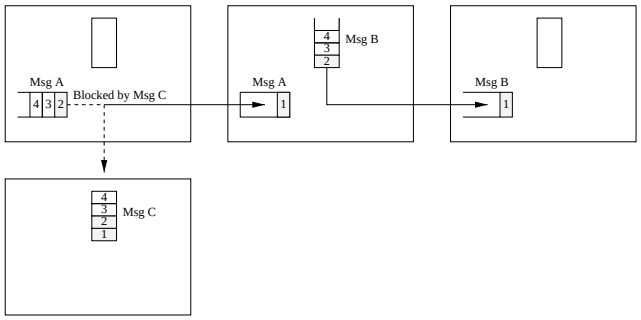

Figure 4. Delayed header flit routing

available for *other* messages in the network that can use them immediately. This scheme only requires a small additional amount of control logic to detect when all header flits at the current node have been routed and one extra control line per VC used to notify the header flits at the neighboring nodes when routing can continue. Figure 4 shows an example of *delayed header flit routing*. In this figure, header flit $A.2$ is blocked while header flit $A.1$ waits at the neighbor node until the last header flit in this message (flit $A.2$) is routed. Message B can now be routed. As the number of destinations grows, *delayed header flit routing* becomes increasingly important in reducing the probability of message blocking.

2.3 Deadlock Detection and Recovery Mechanism

In HTA, each node has a dedicated holding queue called the deadlock queue. If a header flit currently under consideration cannot be routed to a channel in a predetermined amount of time (timeout delay), the header flit is considered to be in a potential deadlock situation and is routed to the deadlock queue at the current node. This timeout delay value will further be explored in Section 3.

Once one of the header flits in a message has been assigned to the deadlock queue, all remaining header flits in the message must be routed as soon as they reach the top of the queue. If any remaining header flit cannot be immediately routed, it is also considered to be in a potential deadlock and is routed to the deadlock queue at the current node. Messages in the deadlock queue are re-injected into the network after a predetermined amount of time (re-injection delay). This re-injection delay will be further explored in Section 3.

When the deadlock queue is full and another message is potentially deadlocked, an interrupt is generated and the message is absorbed into the current node. When space is available in the router's deadlock queue, the message is prefetched from the local processing node and moved to the deadlock queue in the router. If a newly generated message and a potentially deadlocked message from the absorbed queue simultaneously request the source queue, the message from the absorbed queue is given priority. By allowing the overflow of messages to be stored in the local processing node, this deadlock queue becomes essentially infinite for all practical purposes without causing any additional delay in routing and eliminates the possibility of deadlock.

3 Performance Evaluation by Simulation

Extensive simulation experiments were carried out to compare the performance of our proposed HTA scheme with two commonly-used multicast schemes, e.g. *Software Multicast* (one unicast message is sent for every destination address in the multicast message) and *Column-Path* [1] (The destinations in a multicast message are placed into submulticast messages according to the column the destination is in. For example, in a unidirectional torus, at most k sub-multicast messages can be sent per multicast message, one sub-multicast message sent per column).

A discrete-time simulator was used for 8-ary 2-cube and 16-ary 2-cube networks. Message sizes varied from 16 to 64 flits and number of destinations per multicast message were randomly chosen and varied from 8 to 32. The buffer sizes used in the simulation are all equal to a single message length. All router implementations use six VCs per dimension. The Software Multicast and Column-Path algorithms both use two VCs per dimension for deterministic routing and four for adaptive. The timeout and re-injection delays for all message sizes and number of destinations per message simulated here for HTA are 16 and 50 clock cycles, respectively. Fifty cycles is a feasible delay because the deadlock detection and recovery scheme is not a software-based approach. Instead, the deadlock queue that holds potentially deadlocked messages in HTA is located in the router. The deadlock queue can hold one multicast message with all overflow messages being absorbed by the local processing node.

The communication startup time required for ordering the messages in the Column-Path algorithm is *not* included in the simulations. The time required for creating and placing the messages in the source queue is also *not* included

for any of the routing algorithms simulated here. The simulations use a stabilization threshold of a 0.005 difference between traffic 1000 clock cycles apart to determine steady state. Traffic was varied from 0.1 until saturation was reached in 0.1 increments. Simulations were performed for traffic composed of only multicast communication, only unicast communication, and half unicast and half multicast communication.

To reduce the probability of deadlock near saturation, injection limitation schemes are often used [13, 14, 15]. In the simulations performed here, message injections were limited to three unicast and one multicast message in the source queue simultaneously. This back-pressure mechanism sometimes results in a fairly flat curve near saturation in the latency versus normalized applied load graphs.

All implementations use 12 sink channels. Although this is an unusually high number of consumption channels, the Column-Path routing algorithm requires this many channels for deadlock freedom since the adaptive routing algorithm proposed in [11, 12] is used for the base routing conformed path (as opposed to e-cube routing where less number of sink channels are required). For fairness, the Software Multicast and HTA are also simulated with 12 sink channels, although both only require one sink channel.

Figure 2 shows the probability of deadlock versus normalized applied load for HTA. Figures 5, 6, 7, and 8 show the message latencies versus normalized applied load plots of the three multicast routing algorithms for unicast and multicast traffic for various message sizes and number of destination nodes. When both unicast and multicast communication are simulated simultaneously, message latency includes both unicast and multicast latency. A discussion of these results is found in the following sections. Note that message latency for HTA includes the time that potentially deadlocked messages wait in the absorbed queue and that reinjection of potentially deadlocked messages are not counted as new throughput into the network.

3.1 Deadlock Probability

Figure 2 shows the probability of deadlock versus normalized applied load for HTA. The probability of deadlock is the total number of potentially deadlocked messages (PDM) divided by the number of messages that have reached their destinations. The probabilities are low except near saturation. Taking into account the differences in the simulations (e.g. time-out and re-injection delays, bidirectionality, switching type, message and network size), HTA's results are comparable to those reported for k-ary n-cubes under unicast traffic in [16, 17, 18, 19, 15].

In schemes such as DISHA, when a potential deadlock occurs one of the messages in the deadlocked set is removed from the network using additional buffers at each node to route the message directly and immediately to its destination and therefore the PDM does not have another chance of deadlocking [18, 16]. In HTA the PDM is removed from the network at the current node and is reinjected into the network after a given amount of time at the same current node in which it deadlocked. The re-injected PDM may potentially deadlock again along the path to its destination node. HTA does not require as much complex logic, is scalable, and does not have a single point of failure.

Increasing the number of destinations per message results in a greater probability that a destination will block, increasing deadlock probability. The greater the message size, the greater the deadlock probability because more resources are occupied. Increasing network size results in a longer path between source and destination and also results in greater deadlock probability.

3.2 Multicast Latency

Message Latency. Figure 5 shows that HTA performs best among all three algorithms at all utilization and for all message sizes, even without including the time for destination ordering in the Column-Path algorithm. This is because the probability of message blocking has been kept low, the deadlock detection and recovery algorithm is efficient, and because tree-based routing does not unnecessarily copy data flits when routing multicast messages.

As the number of destinations per message increases, the Column-Path algorithm performs better (although not as well as HTA) because more destinations can be grouped together, requiring less number of sub-multicast messages to be sent per multicast message.

HTA always performs better than Software Multicast. Even though HTA has greater message blocking probability at each node as well as fewer VCs per dimension than Software Multicast, the necessity of sending n completely separate messages for n different destinations in Software Multicast greatly outweighs these disadvantages of HTA (especially for large number of destinations). In addition, HTA keeps message blocking at each node to a minimum by routing all destinations through the same path as long as possible and by not restricting the use of *any* available channel (unlike deadlock prevention schemes).

Saturation Point. HTA always has the highest saturation point since channels are only used and data flits are only copied when necessary. Traffic in the network is kept low resulting in increased saturation points.

Effects of Network Size. HTA's performance increases as the 8-ary 2-cube network is increased in size to a 16-ary 2-cube network (Figure 6). Its performance is always better than the other two algorithms for all message sizes.

The Column-Path algorithm performance slightly suffers due to the lower probability that messages will fall in the same column and therefore uses more sub-multicast messages. The HTA is much more flexible with respect to topology and its latency remains low.

Effects of Traffic Type (Unicast vs. Multicast). When traffic is composed of all unicast messages (Figure 7), the Column-Path and Software Multicast algorithms give similar performance because both these algorithms use the same unicast routing scheme and have the same number of VCs devoted to unicast messages. The slight variances are simply due to the random generation of messages.

At low utilization, HTA performs best because only three VCs per dimension are devoted to unicast communication while the other two algorithms devote all six VCs. Having fewer number of VCs per dimension means less message multiplexing, resulting in lower message latencies.

At high utilization, the Column-Path and Software Multicast algorithms perform better because each of these algorithms has six VCs per dimension for unicast messages. Six VCs provides greater adaptivity for messages to be routed around blocked messages at high utilization, resulting in higher saturation points.

When traffic is composed of half unicast and half multicast traffic (Figure 8), unicast and multicast latency versus normalized applied load graphs are shown. HTA performs comparable to or better than the other algorithms for all messages sizes and utilization.

4 Comparison of HTA with Existing Schemes

HTA differs in many important respects from the previously proposed tree-based routing algorithms [9, 4, 7, 10]. Below, we provide a detailed comparison of HTA with the existing schemes.

- In most tree-based routing algorithms, wormhole switching is implemented resulting in large blocking probabilities and poor performance. As shown in [18], virtual cut-through switching can greatly decrease the probability of deadlock for unicast communication over wormhole switching when a fixed number of virtual channels are used. Although in [7] cut-through switching is used, the switching is implemented using a common buffer pool and the buffer is located in the local processing node (not in the router itself). In HTA a buffer is implemented at every VC and every buffer is located in the router. This greatly increases performance.

- In [7] only one PC (with no VCs per PC) is used per dimension and only one sink channel is implemented. As shown in [18, 16], increasing the number of VCs for unicast communication increases routing freedom which in turn exponentially decreases the probability of deadlock and increases performance.

- Although [7] uses a deadlock detection and recovery scheme, the HTA scheme is more efficient. When a header flit blocks for a predetermined amount of time in HTA, the header flit is routed to the deadlock queue. All remaining header flits are then immediately routed to, first, any previously routed channels, then to any available and applicable channel and finally to the deadlock queue if no other routing option remains. The HTA scheme improves upon the deadlock recovery method in [7] in which all header flits (including all those that have already been routed at the current node and at any of the downstream nodes) are aborted. In addition, for their deadlock recovery scheme, the entire message is always copied to the local processing node when a multicast is split (whether deadlock occurs or not). When an abort does happen, the message is already stored at the local node and is ready to be reinjected into the network after a given amount of time. However, this method wastes valuable channel bandwidth and causes contention in the network if two multicast channels request the split channel.

- Choices for timeout values for deadlock detection and recovery schemes include timeouts equivalent to the size of the message [19], four times the message length [15], 8-16 cycles [20], and 800-1000 cycles [14]. Tree-based multicast communication timeout requirements are slightly different than those for unicast communication. In tree-based multicast, more than one header flit is usually routed at each node. If the timeout is too great for each header flit, message progress through the network will be very slow since data flits are not forwarded until all header flits are routed. If the timeout is too small, many false deadlocks will result. HTA uses 16 cycles for all message sizes and number of destinations per message.

- Reinjection delays for unicast communication on wormhole switching under k-ary n-cube networks are around 200 cycles [14, 15]. Deadlock detection and recovery techniques similar to DISHA [20] do not require re-injection delays because when messages are potentially deadlocked they are immediately routed using ”floating buffers” to their destination. HTA uses a re-injection delay of 50 cycles.

- It was shown in [18] that bidirectional networks have a lower deadlock probability than unidirectional networks. Deadlock recovery schemes using bidirectional channels include [19, 14, 18, 15]. Unidirectional channels are addressed in [18, 17]. HTA uses unidirectional channels.

- The QBM scheme [10] is a deadlock-free algorithm. This limits routing options to only those paths valid for deadlock freedom. The scheme also requires a startup delay for building a QBM tree at the beginning of a user-level multicast making it more suitable for bulk multicasts. HTA has more routing flexibility since any available path is a valid routing option in its deadlock

Figure 5. All multicast communication for an 8-ary 2-cube network with message size = 64 flits.

Figure 6. All multicast communication for a 16-ary 2-cube network with message size = 64 flits.

Figure 7. All unicast communication with message size = 64 flits for an 8-ary 2-cube.

Figure 8. 50% unicast and 50% multicast communication for an 8-ary 2-cube network with message size = 64 flits.

detection and recovery scheme and HTA does not require any additional startup delay.

- Several machines already have some hardware support for multicast. The nCUBE-2 is a wormhole-switched hypercube which supports broadcast within each subcube. However, deadlock is possible if multiple multicasts exist [21]. The NEC Cenju-3 supports broadcast within each continuous region, but deadlock is once again possible if multiple multicasts exist. Finally, the Thinking Machines Corporation (TMC) CM-5 supports one multicast at a time via the control network.

References

[1] R. Boppana, S. Chalasani, and C. Raghavendra. Resource deadlocks and performance of wormhole multicast routing algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 9(6), Jun 1998.

[2] D. Robinson, P. McKinley, and B. Cheng. Path-based multicast communication in wormhole-routed unidirectional torus networks. *J. of Parallel and Distributed Computing*, 45:104–21, 1997.

[3] R. Kesavan and D. Panda. Minimizing node contention in multiple multicast on wormhole k-ary n-cube networks. In *Int. Conf. on Parallel Processing*, 1996.

[4] M. Malumbres, J. Duato, and J. Torrellas. An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors. In *8th IEEE Symp. on Parallel and Distributed Processing*, pages 186–9, Oct 1996.

[5] D. K. Panda, S. Singal, and P. Prabhakaran. Multidestination message passing mechanism conforming to base wormhole routing scheme. In *Proc. First Parallel Routing and Communication Workshop*, Seattle, Washington, 1994.

[6] X. Lin and L. M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *Int. Symp. on Computer Architecture*, pages 116–125, 1991.

[7] G. Byrd, N. Saraiya, and B. Delagi. Multicast communication in multiprocessor systems. In *International Conference on Parallel Processing*, volume 1, pages 196–200, Aug 1989.

[8] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, pages 62–76, 1993.

[9] X. Lin, P. McKinley, and L. Ni. Performance evaluation of multicast worm-hole routing in 2D mesh multicomputers. In *Int. Conf. on Parallel Processing*, pages 435–42, 1991.

[10] J. Yang and C. King. Efficient tree-based multicast in wormhole-routed 2D meshes. In *Inter. Symp. on Parallel Architectures, Algorithms, and Networks*, 1997.

[11] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. In *Symposium on Parallel and Distributed Processing*, pages 64–71, 1993.

[12] P. Berman, L. Gravano, G. Pifarre, and J. Sanz. Adaptive deadlock and livelock free routing with all minimal paths in torus networks. In *Proc. of the Symp. on Parallel Algorithms and Architectures*, pages 3–12, 1992.

[13] P. Lopez, J. Martinez, J. Duato, and F. Petrini. On the reduction of deadlock frequency by limiting message injection in wormhole networks. In *Parallel Computing, Routing, and Communication Workshop*, Jun. 1997.

[14] F. Petrini, J. Duato, P. Lopez, and J. Martinez. LIFE: a limited injection, fully adaptive, recovery-based routing algorithm. In *Fourth Inter. Conf. on High Performance Computing*, Dec. 1997.

[15] J. Martinez, P. Lopez, J. Duato, and T. Pinkston. Software-based deadlock recovery technique for true fully adaptive routing in wormhole networks. In *International Conference on Parallel Processing*, Aug. 1997.

[16] T. Pinkston and S. Warnakulasuriya. On deadlocks in interconnection networks. In *International Symp. on Computer Architecture*, pages 38–49, Jun. 1997.

[17] A. Folkestad and C. Roche. Deadlock probability in unrestricted wormhole routing networks. In *IEEE International Conference on Communications*, Jun. 1997.

[18] S. Warnakulasuriya and T. Pinkston. Characterization of deadlocks in interconnection networks. In *Int. Parallel Processing Symp.*, Apr. 1997.

[19] J. Kim, Z. Liu, and A. Chien. Compressionless routing. In *Proc. of ISCA*, Apr 1994.

[20] Anjan V. and T. Pinkston. An efficient, fully adaptive deadlock recovery scheme: DISHA. *Computer Architecture News*, 23(2), May 1995.

[21] L. Ni. Should scalable parallel computers support efficient hardware multicast. In *Int. Conf. on Parallel Processing*, 1995.