This is to certify that the

dissertation entitled

SCALABLE MULTICAST COMMUNICATION
IN MASSIVELY PARALLEL COMPUTERS

presented by

David F. Robinson

has been accepted towards fulfillment
of the requirements for

PhD degree in Computer Science

Betty H. C. Cheng
Major professor

Date 8/3/94

SCALABLE MULTICAST COMMUNICATION IN MASSIVELY PARALLEL COMPUTERS

By

David F. Robinson

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science Department

1994

ABSTRACT

SCALABLE MULTICAST COMMUNICATION IN MASSIVELY PARALLEL COMPUTERS

By

David F. Robinson

Efficient communication has long been considered the key to achieving ever greater performance from parallel processing. Recently, much attention has been focused on multicast communication, in which a single source node delivers a message to a group of destination nodes. Such operations have become recognized as crucial to the performance of many parallel algorithms. Given the important role played by multicast communication in parallel processing, the research reported in this dissertation addresses the effect on multicast communication of three key aspects of communication architecture in massively parallel computers, namely (1) port model; (2) virtual channels; and (3) intermediate message reception. This research shows that the performance of multicast communication can be significantly improved by considering these three architectural characteristics in the design of multicast operations.

Research into the effect of the port model focuses on the problem of multicast in wormhole-routed hypercubes. The system model allows a processor to send and receive data in all dimensions simultaneously. New theoretical results that characterize contention among messages in wormhole-routed hypercubes are developed and used to design new multicast routing algorithms. The algorithms are compared in terms of the number of steps required in each, their measured execution times when implemented on a relatively small-scale nCUBE-2, and their simulated execution times on larger hypercubes. The results indicate that significant performance improvement is possible when the multicast algorithm actively identifies and uses multiple ports in parallel.

Through the study of virtual channels, efficient algorithms are presented to implement multicast communication in wormhole-routed torus networks. By exploiting the properties of the switching technology and the use of virtual channels, a minimum-time multicast algorithm is presented for n-dimensional torus networks that use deterministic, dimension-ordered routing of unicast messages.

In order to study the third characteristic of communication architecture, research is presented that focuses on torus networks in which intermediate nodes on a message path are able to receive a copy of a message while simultaneously routing the message to subsequent destinations. In developing new multicast algorithms for such networks, this research examines the effects of intermediate message reception on multicast communication. The results of a simulation study show that, through the efficient use of special routing hardware, the performance of multicast communication in torus networks with unidirectional communication links can be significantly improved.

ACKNOWLEDGMENTS

It is with deep gratitude that I acknowledge my Ph.D. advisers, Dr. Betty H. C. Cheng and Dr. Philip K. McKinley. Because of their eagerness to help, whenever help was needed, my academic career has been extremely rewarding.

Drs. McKinley and Cheng were in a large part responsible for many of the opportunities available to me while at Michigan State University. Their knowledge and insight have been invaluable in defining the direction of my research. Through their advice regarding research topics and publication opportunities, and their subsequent help in my development, preparation, and refinement of technical papers, I have been able to pursue a very rewarding course of research. Their careful reading of numerous publication drafts, including earlier versions of this dissertation, has made my work more enjoyable and successful than I could have envisioned.

Their guidance, advice, and encouragement, along with a genuine and selfless concern for my interests, have made all the difference during my graduate education at Michigan State University.

I thank the members of my Ph.D. committee, Dr. Abdol H. Esfahanian and Dr. Marvin L. Tomber, for the very helpful and important work they have performed on my behalf.

Finally, I thank those faculty members at Michigan State University, who through their genuine interest and enthusiasm for education, have taught me much about excellence in teaching. The lessons I have learned from them will be valuable to me throughout my career.

TABLE OF CONTENTS

LIST OF FIGURES viii

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Thesis Statement 3
- 1.3 Research Contributions 4
- 1.4 Dissertation Organization 4

2 Background and Related Work 6
- 2.1 Multicast Communication 6
- 2.2 MPC Communication Architectures 9
  - 2.2.1 Network Topologies 9
  - 2.2.2 Switching Strategy 12
  - 2.2.3 Routing Algorithm 14
- 2.3 Port Model 15
- 2.4 Virtual Channels 18
- 2.5 Intermediate Message Reception 20

3 Unicast-Based Multicast in All-Port Hypercubes 24
- 3.1 Issues 25
- 3.2 Theoretical Foundations 28
  - 3.2.1 Notation and Definitions 29
  - 3.2.2 Useful Lemmas 31
  - 3.2.3 Arc-Disjoint Paths 32
  - 3.2.4 Avoiding Depth Contention 33
- 3.3 Algorithms Based on Dimension-Ordered Chains 36
- 3.4 An Algorithm Based on Cube-Ordered Chains 40
- 3.5 Performance Evaluation 50
  - 3.5.1 Stepwise Comparisons 50
  - 3.5.2 Implementations on an nCUBE-2 51
  - 3.5.3 Simulations of Larger Systems 53
- 3.6 Conclusions 54

4 Unicast-Based Multicast in One-Port Torus Networks 57
- 4.1 System Model 58
- 4.2 Unicast Routing Algorithms 60

4.2.1 Unidirectional Torus Routing 61
4.2.2 Bidirectional Torus Routing 63
4.3 Contention in Wormhole-Routed Torus Networks 65
4.4 Optimal Multicast Algorithm 74
4.5 Performance Evaluation 80
4.6 Conclusions 86

5 Path-Based Routing in Unidirectional Torus Networks 88
5.1 Issues 89
5.2 Hamiltonian Circuits 92
5.3 The Path Routing Function 96
5.3.1 Restrictive Routing 96
5.3.2 General Routing 98
5.4 Message Preparation 100
5.5 Correctness 103
5.6 Implementation Issues 111
5.6.1 Multi-Destination Message Format 111
5.6.2 The Flit Forwarding Algorithm 113
5.6.3 Boundary Identification 115
5.6.4 Implementation of the Path Routing Function 116
5.7 Conclusions 117

6 Path-Based Multicast in Unidirectional Torus Networks 118
6.1 The S-Torus Multicast Algorithm 119
6.2 The M-Torus Generalized Multicast Algorithm 120
6.3 The M_d-Torus Multicast Algorithm 124
6.4 The M_u-Torus Multicast Algorithm 125
6.5 Multi-Phase Implementation Issues 127
6.6 Performance Evaluation 130
6.7 Conclusions 134

7 Conclusions 142

BIBLIOGRAPHY 145

LIST OF FIGURES

2.1 MPC network topologies 10
2.2 Example message paths under dimension-ordered routing in a mesh . 16
2.3 Generic MPC node architecture [1] 17
2.4 Channel dependencies in a 1D torus with single channels 19
2.5 Router support for multicast communication 21
2.6 A multicast operation using message replication 22

3.1 An example of multicast in a 4-cube 26
3.2 Unicast-based software multicast trees 27
3.3 Conditions 2, 3, and 4 of Theorem 3.3 35
3.4 Multicast chain in a one-port 4-cube 37
3.5 Generalized multicast algorithm 39
3.6 Simple Maxport and U-cube comparison 39
3.7 Cube-ordered chains of dimension 4 41
3.8 The *WeightedSort* procedure 43
3.9 Examples of multicast communication 45
3.10 Stepwise comparisons on a 6-cube 51
3.11 Stepwise comparisons on a 10-cube 52
3.12 Average delay comparisons on a 5-cube 53
3.13 Maximum delay comparisons on a 5-cube 54
3.14 Average delay comparisons on a 10-cube 55
3.15 Maximum delay comparisons on a 10-cube 56

4.1 Examples of 2D torus networks 59
4.2 Virtual channels in one dimension of a unidirectional torus 62
4.3 Virtual channels in one dimension of a bidirectional torus 63
4.4 An example of multicast in a 2D torus 66
4.5 Unicast-based software multicast trees 67
4.6 Channels used by $P(u,v)$ and $P(x,y)$ in dimension d (Theorem 4.2) . 72
4.7 Channels used by $P(u,v)$ and $P(x,y)$ in dimension d (Theorem 4.3) . 74
4.8 A minimum-time multicast 75
4.9 The U-torus algorithm for multicast 77
4.10 Example multicast using the U-torus algorithm 78
4.11 Possible relations between unicasts produced by U-torus 78
4.12 Average communication steps (512 and 1024-node tori) 82
4.13 Maximum communication steps (512 and 1024-node tori) 83

4.14 Average communication steps (4096-node tori) 84
4.15 Maximum communication steps (4096-node tori) 85
4.16 Effects of physical link sharing 86
4.17 Comparison of 2-way and 3-way physical link sharing 87

5.1 A multicast operation in a 2D torus 90
5.2 Port-induced communication deadlock in a one-port system 92
5.3 Path-based multicast routing in a mesh using Hamiltonian Paths . . . 93
5.4 Hamiltonian circuit $\mathcal{H}$ in a 2D torus 94
5.5 Hamiltonian circuit $\mathcal{H}$ in a 3D torus 95
5.6 A (restrictive) multi-destination message in a 2D torus 98
5.7 A multi-destination message in a 2D torus 103
5.8 The message preparation algorithm for path-based routing 104
5.9 Multi-destination message format 112
5.10 The flit forwarding algorithm for path-based routing 114
5.11 Implementation of the path routing function $\mathcal{R}_{UTPR}$ 116

6.1 The M-torus algorithm for multicast 122
6.2 Compound message format 128
6.3 Multicast latency (4096-node 2D torus, high message startup latency) 136
6.4 Multicast latency (4096-node 2D torus, low message startup latency) 137
6.5 Multicast latency (4096-node 3D torus, high message startup latency) 138
6.6 Multicast latency (4096-node 3D torus, low message startup latency) 139
6.7 Multicast latency (4096-node 2D torus, 512 node multicast size) . . . 140
6.8 Total link usage (4096-node tori) 141

CHAPTER 1

Introduction

The recent trend in supercomputer design has been towards *scalable* parallel computers, which are designed to offer corresponding gains in performance as the number of processors is increased. Many such systems, known as *massively parallel computers* (MPCs), are characterized by the distribution of memory among an ensemble of processing nodes. Each node has its own processor, local memory, and other supporting devices. MPCs are scalable because, as the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase.

1.1 Motivation

In parallel scientific computing, data must be redistributed periodically in such a way that all processors can be kept busy performing useful tasks. Because they do not physically share memory, nodes in MPCs must communicate by passing messages through a communications network. Some communication operations are *point-to-point*, that is, they involve only a single source and a single destination. Other operations are *collective*, in that they involve more than two nodes. Examples of collective communication include *multicast*, *reduction*, and *barrier synchronization*.

Multicast communication, in which a single source node delivers a message to a group of destination nodes, is important to many MPC activities, including numeric algorithms, parallel simulation, and the implementation of data-parallel languages, such as High Performance Fortran. The performance of MPCs is thus highly dependent on the performance of the underlying multicast operations, which, in turn, depends on several characteristics of the MPC communication architecture.

Three characteristics that affect the performance of multicast communication are *network topology*, *message routing algorithm*, and *message switching strategy*. The network topology defines the pattern of interconnection that exists between nodes; for example, the nodes of an MPC may be connected to form a two-dimensional (2D) mesh network. The routing algorithm determines the path taken by a message between the source and destination nodes, and the switching strategy determines how messages are transferred between adjacent nodes on the message path. Early multicomputers used store-and-forward switching, in which the time taken to transmit a message is proportional to the distance between the source and destination nodes. In contrast, many current MPCs employ wormhole routing, where messages are pipelined through the network.

The characteristics of communication architecture noted above have been widely studied; these contributions are surveyed in Chapter 2. However, there are other system attributes that also have a large effect on the performance of multicast communication. One such attribute is the *port model*, which describes the number of (parallel) connections between a node processor and the communication network. In a *one-port* architecture, each processor is connected to the network by a single input/output channel pair, thereby effectively serializing all communication originating from, and destined for, that node. Some MPCs support a *multi-port* architecture, where processors are connected to the communication network by multiple pairs of input/output channels. In an *all-port* architecture, nodes are maximally connected to the network, so that simultaneous communication over all network links is possible.

Another characteristic affecting multicast communication is the use of *virtual channels*. Some network topologies require virtual channels, in which adjacent nodes are connected by more than a single logical link in one or both directions. The redundant communication paths provided by virtual channels are needed by these topologies in order to provide deadlock-free communication.

Multicast communication in MPCs is also influenced by *intermediate reception capability*. When a message is routed through an intermediate node on a path from the source node to the destination, the processor of the intermediate node is ordinarily unaffected by the message; in fact, message routing hardware is usually such that the processor of an intermediate node cannot access a passing message without interfering with its transmission. A system that allows a processor to simultaneously receive a message while it is being relayed through the node enroute to other destinations is said to have intermediate reception capabilities.

1.2 Thesis Statement

In this dissertation, we address three specific characteristics of MPC communication architecture, and how they relate to the performance of multicast communication. The thesis statement is:

> *The performance of multicast communication in MPCs can be significantly improved by exploiting specific properties of the following three characteristics in the design of the multicast operation: (1) port model; (2) virtual channels; and (3) intermediate message reception.*

1.3 Research Contributions

This dissertation offers three specific contributions to the current research in the area of multicast communication on MPCs, as follows:

1. We develop a new multicast algorithm for all-port hypercubes. This new algorithm is shown to perform significantly better on all-port architectures than the best known algorithm, which is optimal for one-port architectures.

2. We develop an optimal multicast algorithm for one-port torus networks. This algorithm accounts for the presence of virtual channels in torus topologies and is compatible with networks having either unidirectional or bidirectional communication links.

3. We develop new multicast algorithms for torus networks with intermediate message reception capabilities. These algorithms allow multicast communication to be efficiently implemented either wholly or partly in hardware, with a resulting performance gain over corresponding software implementations.

1.4 Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter 2, background is given for multicast communication, MPC communication architectures, and the influence of these architectures on the design of efficient multicast operations. Also in Chapter 2, the related research is reviewed, and differences from the research described in this dissertation are discussed. The three research contributions stated above are described in Chapters 3 through 6. Chapters 3 and 4 describe in detail the research in the area of efficient software-based multicast algorithms for MPCs. In Chapter 3, investigations into all-port hypercubes are presented, while in Chapter 4, we discuss our findings for one-port torus networks. The research presented in Chapters 5 and 6 addresses multicast methods for torus networks with intermediate reception capability and unidirectional communication links. In Chapter 5, a deadlock-free path-based routing method is described. In Chapter 6, this routing method is used as a basis for path-based multicast algorithms. Two classes of algorithms are presented: (1) single-phase, in which a single multi-destination message is used to perform the multicast operation, and (2) multi-phase, where a collection of multi-destination messages are generated during a sequence of communication phases. Finally, Chapter 7 presents the concluding remarks.

CHAPTER 2

Background and Related Work

This chapter presents background information needed to study multicast communication in MPCs. Multicast communication operations, MPC communication architectures, and the issues involved in the implementation of multicast operations are covered. The research of other investigators is reviewed, and differences from the research described in this dissertation are described.

2.1 Multicast Communication

Point-to-point, or *unicast*, communication involves a single source node and a single destination node. In collective communication, also termed *group communication*, more than two nodes are involved. Collective communication operations can be roughly decomposed into three categories: (1) those with a single source node and multiple destination nodes; (2) those with multiple source nodes and a single destination node; and (3) those with multiple source and multiple destination nodes.

In a *multicast* operation, a source node must deliver copies of a single message to each node in the destination group. A special case of multicast is *broadcast*, in which the destination group contains every node in the network (except the source). Unfortunately, the terms broadcast and multicast are often used interchangeably in the literature. Throughout our work, we use broadcast to refer only to the case where the destination set includes all processors in the MPC; other cases are referred to as multicast. Multicast is a fundamental collective communication operation, and is important in many parallel numerical algorithms, including matrix multiplication [2], matrix transpose [3], tridiagonalization [4], eigenvalue computation [5], Gaussian elimination [6], and LU factorization [7]. Efficient implementation of multicast is also useful in many other aspects of parallel computing, including support for barrier synchronization [8], memory updates and invalidation in distributed shared-memory systems [9], and global notification of events in parallel simulation.

The growing interest in the use of collective communication routines, including multicast, is evidenced by their inclusion in the *Message Passing Interface* (MPI) [10], an emerging standard for communication routines used by message-passing programs, and in many research and commercial communication libraries, including IBM's Collective Communication Library (CCL) [11] and MSU's ComPaSS project [6]. Besides message passing, multicast communication is also important in implementing data-parallel languages, such as High Performance Fortran [12], on distributed-memory systems.

Much research has been performed in the area of multicast communication for parallel computers. Many multicast and broadcast algorithms have been developed under the assumption of store-and-forward architectures [13, 14, 15, 16, 17], and multicast communication has been investigated in bus-based architectures [18]. Much of the recent work in this area has addressed software-based (unicast-based) multicast [19, 20, 21] and broadcast [22, 23, 24, 25, 26, 27, 28] communication in wormhole-routed MPCs. Some of the work has investigated the use of special routing hardware for the support of multicast and broadcast communication [29, 30, 31, 32, 33, 34, 35].

In much of the current research in the area of multicast communication, it is assumed that the group of destination nodes consists of all nodes in the MPC; that is to say, the operation is assumed to be broadcast rather than general multicast. However, we focus on multicast operations that allow the destination node group to be arbitrarily specified. Usually, such operations are more difficult to develop than those that place restrictions on the node groups. When designing a software-based multicast operation that will send a message to an arbitrarily specified group of destination nodes, only the processors of the source and destination nodes should be used in the operation. Furthermore, no *a priori* assumptions can be made regarding which nodes will and will not be included in the destination group; the operation must work for any and all destination groups.

Because the set of nodes allocated to an application by the operating system frequently consists of only a subset of the MPC (rather than the entire MPC), multicast communication operations that allow for arbitrary specification of the node groups are often needed even when the application algorithm calls for an operation that accesses all nodes, or some regular subset of nodes. For example, when an application program executing on an allocated subset of the nodes of an MPC requests a broadcast to all nodes, the appropriate operation is actually a selective multicast to the allocated set of nodes. As further evidence of the need for flexible operations, the MPI standard [10] specifies that all collective communication operations are performed on node groups specified arbitrarily by the programmer.

Multicast communication may be implemented in either hardware or software. However, most existing MPCs support only point-to-point, or *unicast* communication in hardware. In these environments, multicast communication must be implemented in software, typically in a communication library, by sending one or more unicast messages; such implementations are called *unicast-based* [19]. For example, a multicast operation may be implemented using *separate addressing*, in which a separate copy of the message is sent directly from the source to every destination. An alternative is to use a *multicast tree* [19] of unicast messages. The tree can be considered as a sequence of message-passing steps. In the first step, the source node actually sends the message to only a subset of the destinations. In the next step, each node holding a copy of the message forwards it to some subset of the destinations that have not yet received it. The sequence of message-passing steps continues until all destinations have received the message. Using this approach, the time required for the operation can be greatly reduced [19].

In order to prevent interference with computation on nodes not directly involved in a multicast operation, the implementation should not affect any local processors other than those explicitly involved in the operation. That is to say, only source and destination node processors should be required to handle the message.

2.2 MPC Communication Architectures

We now describe some of the important characteristics of MPC communication architecture that affect multicast communication. These characteristics are network topology, switching strategy, and routing strategy. Three additional architectural characteristics that affect multicast communication in MPCs are port model, virtual channels, and intermediate reception. These last three factors are central to the research presented in this dissertation, and are discussed in Sections 2.3, 2.4, and 2.5, respectively.

2.2.1 Network Topologies

Two major categories of MPC network topology are *direct networks* and *indirect networks*. In a direct network, each node includes a processor and local memory as well as switching hardware, and is directly connected by physical communication links to some set of nodes, called neighboring nodes. In an indirect, or *multistage* network, some nodes act only as switching elements and have no processing capabilities. Processing nodes are connected indirectly through switching nodes. For example, in a *Fat Tree* topology, such as the Thinking Machines CM-5 [36], the nodes are arranged as a tree, where only the leaf nodes are processing nodes; intermediate nodes in the tree are switching nodes used to deliver messages between processing nodes. We concentrate only on direct networks, which are used in most MPC architectures. Network topologies of commercial and research direct network MPCs vary widely. Some of these topologies are illustrated in Figure 2.1.

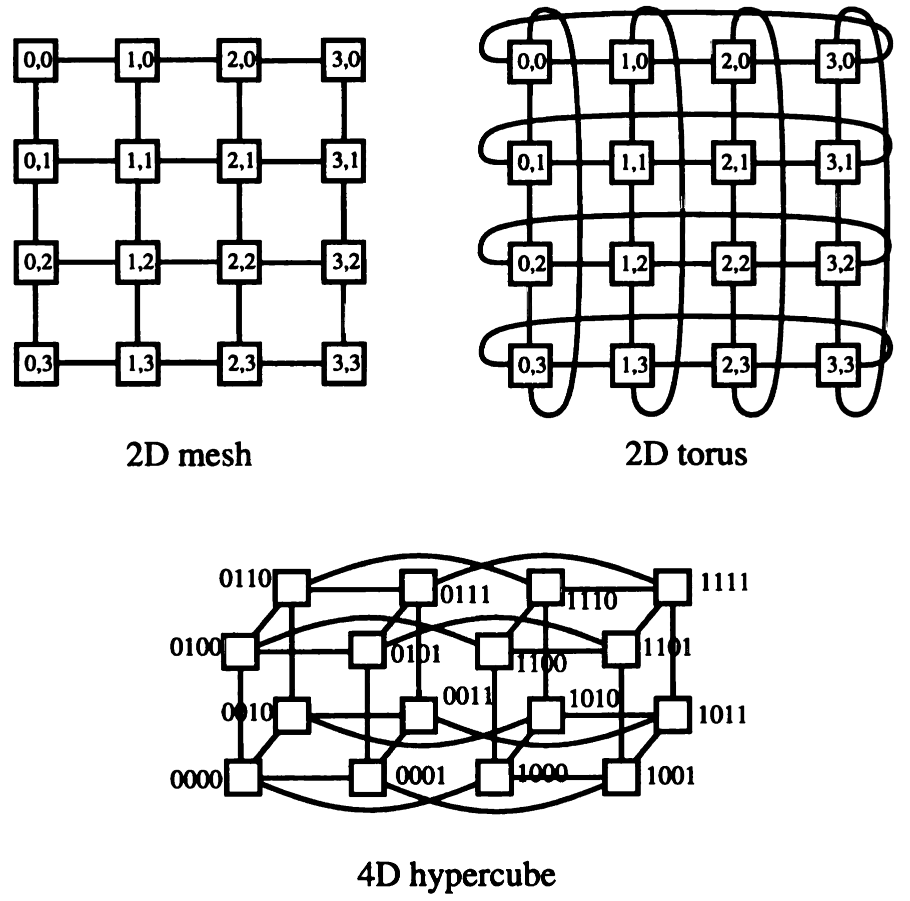


Figure 2.1. MPC network topologies

An n-dimensional *mesh* of width k contains k^n nodes. Each node of a mesh has an n-digit, radix-k address; each digit of the address specifies the coordinate of the node in the corresponding dimension. For example, in a 2-dimensional (2D) mesh, node addresses are the familiar 2D Cartesian coordinates. There exists a communication link between two nodes in a mesh if and only if their corresponding addresses are equal in every dimension except one, in which the address values differ by exactly one. Two nodes connected by a communication link are said to be *adjacent*, or *neighboring*, nodes. Examples of mesh architectures include the 2D Intel Paragon [37], the 2D Caltech Mosaic C [38], and the 3D MIT J-machine [39].

An n-dimensional *torus* is equivalent to an n-dimensional mesh in which each edge node is connected to the corresponding node on the opposite edge by a "wraparound" channel. Pairs of nodes on opposite edges of the network are made adjacent by these wraparound channels, thereby providing a shorter average path length than in a mesh network. Torus networks include the 2D Intel/CMU iWarp [40], the 3D Cray T3D [41], and the Torus Routing Chip [42], which can be used directly to construct 3D torus networks or cascaded to build tori of higher dimension.

An n-dimensional *hypercube*, or *n-cube*, contains 2^n nodes, each with an n-bit binary address. Two nodes x and y in a hypercube are adjacent if and only if their corresponding addresses differ in exactly one bit position. Commercial hypercubes include the nCUBE-2 [43] and nCUBE-3 [44]. An n-cube (hypercube) is a special case of an n-dimensional mesh or torus, where the width is 2. The general term *k-ary n-cube* has been used to refer to both mesh and torus networks of width k; thus, a *binary n-cube* is a hypercube [45].

Early systems that used store-and-forward switching often adopted a hypercube topology [46] because of the relatively dense interconnection network, which resulted in shorter message paths. However, in systems with wormhole routing, the distance between communicating nodes is less important; hence, the more easily constructed lower-dimension meshes and tori are often chosen as topologies for current MPCs [45]. Torus networks are sometimes favored over meshes, because with random communication, the communication links of a mesh are not evenly utilized, whereas a torus network achieves equal utilization of all links [41]. Also, because of the existence of wraparound channels, a torus with bidirectional communication links can provide shorter message paths on average than can a mesh, thus resulting in a more lightly loaded network. However, torus networks require the use of virtual channels in order to provide deadlock-free communication, whereas mesh and hypercube networks do not have this requirement.

2.2.2 Switching Strategy

The predominant switching technique in MPCs is *wormhole routing* [42], in which a message is divided into a number of *flits* that are pipelined through the network. The header flit of a message proceeds on the path to the destination node, followed by the remaining flits. If the header encounters a needed communication channel that is not currently available (due to use by another message), the header, along with those trailing flits already in the network, are blocked in place. When the required channel becomes available, the header flit, followed by the subsequent flits of the message, continue towards the destination.

Because of the way in which messages are blocked in place in a wormhole-routed network, only very small, fixed-size flit buffers are needed for each communication channel. These buffers can be as small as a single flit [43, 47], and are easily incorporated into the routing hardware at each node [42]. The term *network latency* refers to the elapsed time after the head of a message has entered the network at the source until the tail of the packet emerges at the destination. For long messages, the pipelining effect of wormhole routing reduces the effect of path length on network latency [1]. The *startup latency* is the time required for the system to handle the packet at both the source and destination nodes. For small messages, the startup latency often dominates unicast latency [19]. Therefore, in the absence of contention among messages for network resources, the latency of wormhole-routed messages is nearly distance-insensitive [1].

This behavior is in contrast to early *store-and-forward* systems [46], in which messages are transferred completely across each hop on the path before beginning travel on the next hop. Store-and-forward systems require adequate buffer space with each channel to accommodate the largest possible message. In addition, the amount of time required to deliver a message using store-and-forward switching is roughly proportional to the length of the message path. The distance-insensitive communication latencies and small buffer requirements associated with wormhole routing make this technique scalable, and thus well suited for use in MPCs.

Wormhole routing has been adopted in the Symult 2010, the nCUBE-2 and nCUBE-3, Intel/DARPA's Touchstone DELTA and the subsequent Intel Paragon, the MIT J-machine, Intel/CMU's iWarp, the Caltech Mosaic C, the Transputer IMS T9000 family, the Torus Routing Chip, the TMC CM-5, and the Cray T3D. A survey of the issues related to wormhole routing can be found in the literature [1].

Under wormhole routing, a message must acquire exclusive use of all channels on the path on which it travels. This extended retention of communication resources makes wormhole routing susceptible to *channel contention*, in which two or more messages simultaneously require the same communication channel. When channel contention occurs, the message that first requested the common channel proceeds, while other messages requiring the same channel are blocked in place in the network until the required channel has been relinquished. When messages are blocked in place by channel contention, they continue to hold the channels they have already acquired, thereby increasing the possibility of further channel contention. Thus, when designing multicast operations for wormhole-routed systems, it is important to avoid channel contention among the constituent messages of the operation.

2.2.3 Routing Algorithm

Nodes in current MPCs are connected by topologies that provide multiple paths between a given source and destination node; in fact, in all existing MPC topologies, there is more than one shortest path between many of the source/destination pairs. Thus, routing choices must be made when transmitting a message from source to destination. The way in which these routing choices are made is termed the *routing algorithm*.

Under *adaptive routing*, information about current network conditions such as traffic and defective nodes or links is used to determine message routes. This approach is in contrast to *deterministic routing*, in which the route between a given source and destination is unique and independent of current network conditions. Deterministic routing has also been termed *oblivious* routing. Although adaptive routing mechanisms for wormhole-routed networks have been the focus of recent research [48, 49, 50, 51, 52, 53, 54, 55], we study deterministic strategies because of their prevalence in both current and newly-announced architectures. In a deterministic routing method termed *dimension-ordered routing*, messages are routed first in the highest (lowest) dimension in which the source and destination nodes differ. Routing then proceeds on each required dimension, in descending (ascending) order of dimension, until the routing path reaches the destination. Routing in a particular dimension is always completed before routing in the next dimension begins. We assume, without loss of generality, that routing is performed in descending order of dimension.

Sullivan and Brashkow [56] introduced *E-cube* routing, which is essentially dimension-ordered routing for hypercubes. In 2D grid-like architectures such as a

2D mesh or torus, dimension-ordered routing is also termed *XY-routing* [1], since messages are routed first in the so called "X" dimension, and then in the "Y" dimension. Similarly, dimension-ordered routing in 3D topologies is often referred to as XYZ-routing. Many current and research MPCs use both wormhole routing as a switching strategy and dimension-ordered routing as a routing strategy, including the Symult 2010, the nCUBE-2 and nCUBE-3, Intel/DARPA's Touchstone DELTA and the subsequent Intel Paragon, the MIT J-machine, the Caltech Mosaic C, the Torus Routing Chip, and the Cray T3D.

The choice of routing algorithm also has a large influence in the design of multicast operations, because it determines how the constituent messages must be scheduled in order to avoid channel conflict. Channel conflict is undesirable in wormhole-routed networks because of the resultant message blocking. For example, consider the two unicast messages in a 2D mesh: message $\mathcal{U}_1$ from source node $(0,3)$ to destination node $(2,1)$, and message $\mathcal{U}_2$ from source node $(3,3)$ to destination node $(2,2)$. Although there are numerous minimum-length paths in a 2D mesh that would result in messages $\mathcal{U}_1$ and $\mathcal{U}_2$ traveling on arc-disjoint paths, the rules of dimension-ordered routing dictate the paths shown in Figure 2.2, which are unfortunately *not* arc-disjoint. Thus, in order to avoid channel contention in an MPC with dimension-ordered routing, a multicast operation must be implemented so as to produce messages that are pairwise either (1) arc-disjoint under the routing rules, or (2) temporally distinct.

2.3 Port Model

In wormhole-routed MPCs, communication among nodes is handled by a separate *router*. As shown in Figure 2.3, several pairs of *external channels* connect the router to neighboring routers, and are used for communication between those routers. The pattern in which the external channels are connected defines the network topology.

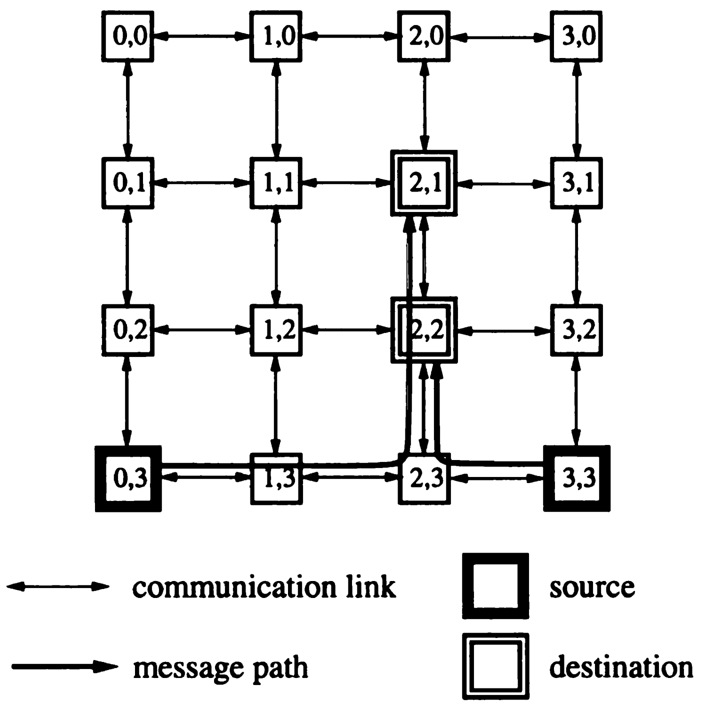


Figure 2.2. Example message paths under dimension-ordered routing in a mesh

Usually, the router can relay multiple messages simultaneously, provided that each incoming message requires a unique outgoing channel.

A router is connected to the local processor/memory by one or more pairs of *internal* channels. One channel of each pair is for input, the other for output. The *port model* of a system refers to the number of internal channels at each node. If each node possesses exactly one pair of internal channels, then the result is a so-called "one-port communication architecture" [14]. A major consequence of a one-port architecture is that the local processor must transmit (receive) messages sequentially. Although additional pairs of internal channels will increase communication capacity, the one-port architecture is characteristic of many existing systems. Architectures with multiple ports reduce this bottleneck. In the case of an *all*-port system, every external channel has a corresponding internal channel, allowing the node to send to and receive on all external channels simultaneously.

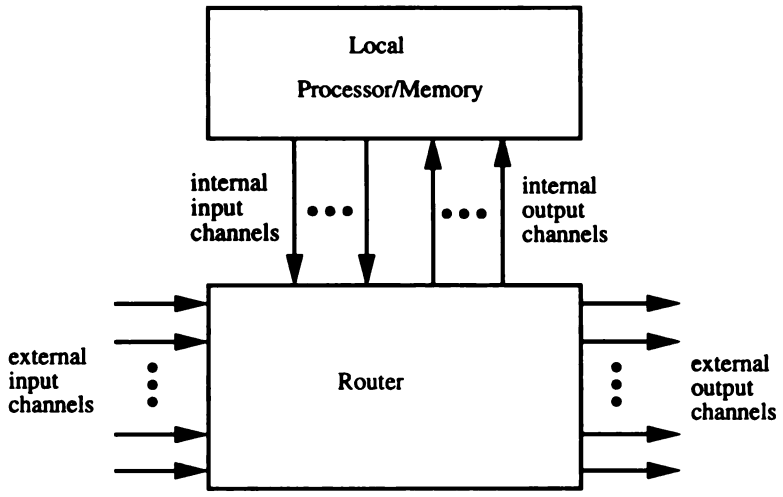


Figure 2.3. Generic MPC node architecture [1]

Some researchers have considered the effects of an all-port or *Multiple Link Availability* (MLA) architecture under the store-and-forward model. This work includes the study of broadcast communication in hypercube topologies [13, 14, 17]. Research into multicast communication on current wormhole-routed MPCs with multi-port architecture is just emerging, and includes work on broadcast in hypercubes [24, 25] and meshes [27]. Broadcast in all-port torus and mesh architectures is considered in [26], where a non-standard routing algorithm is assumed.

Our work in this area, which is described in Chapter 3, differs from the previous research in the following ways. As pointed out above, broadcast in all-port systems has been studied previously. However, the more general multicast problem has not been addressed. McKinley, *et al.* [19] developed the *U-cube* and *U-mesh* multicast algorithms, which are optimal for arbitrary multicast operations in *one-port* wormhole-routed hypercubes and meshes, respectively. By generalizing the U-cube algorithm, we provide a framework for studying all-port multicast algorithms, and then use this framework to develop the *W-sort* multicast algorithm, which performs better than U-cube in an all-port hypercube. We are the first to study multicast in all-port wormhole-routed MPCs.

2.4 Virtual Channels

In topologies with natural channel routing cycles, such as a torus, if only one communication channel is provided between each pair of neighboring nodes, then the network will not be deadlock free, since it would then be possible for a cycle of channel allocation and demand to exist among two or more unicast messages. Dally and Seitz [57] show that a network is deadlock-free under deterministic routing if and only if there are no cycles in the channel dependency graph. A ***channel dependency graph*** is a directed graph in which each vertex represents a channel of the network; there is an arc from channel c_i to channel c_j if and only if a message arriving on c_i might next be routed on c_j. We consider the case of a 1D torus, which is a simple ring. If there is only one unidirectional channel between each pair of neighboring nodes, then the channel dependency graph will consist of a single ring of channels (see Figure 2.4), which shows that the associated network is not deadlock-free. Similar cycles appear along each dimension in the channel dependency graphs of larger-dimension tori whenever pairs of nodes are connected only by single, unidirectional channels.

In order to enable deadlock-free routing in a torus network, multiple ***virtual channels*** can be multiplexed onto each physical communication link. These virtual channels share the bandwidth of the physical link, and provide multiple logical paths between neighboring nodes that are used by the routing algorithm to break cycles of channel dependency.

The issues related to the use of virtual channels to avoid deadlock in general topologies with natural channel dependency cycles are discussed by Dally and

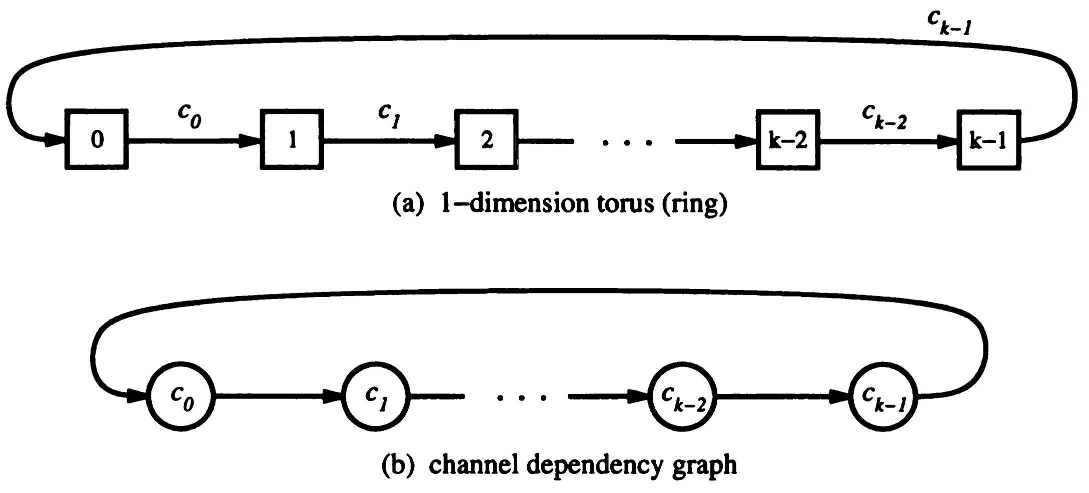


Figure 2.4. Channel dependencies in a 1D torus with single channels

Seitz [57]. Virtual channels are used to avoid deadlock in current torus MPC architectures, including the Cray T3D [41], and the Torus Routing Chip [42]. Besides providing deadlock-free routing in torus networks, the multiple logical paths associated with virtual channels have been used to support adaptive routing algorithms [49, 50, 51, 53, 54]. Dally [58] examines the use of virtual channels to improve network throughput in MPCs.

Our work in this area, which is described in Chapter 4, differs from the previous research in the following ways. As noted above, much research has focused on the use of virtual channels to provide deadlock-free and adaptive routing in MPCs. However, this work has generally included only point-to-point communication. Park, *et al.* [26] have studied broadcast in torus networks, but their work assumes a non-standard routing algorithm. The U-mesh multicast algorithm [19] is contention-free in one-port wormhole-routed *mesh* networks, but not in torus networks. In designing an optimal multicast algorithm for one-port wormhole-routed torus networks, we are the first to study unicast-based multicast on wormhole-routed torus architectures.

2.5 Intermediate Message Reception

In systems that support only point-to-point communication in hardware, multicast operations must be implemented in software by using unicast-based techniques such as multicast trees. In order to improve multicast performance and reduce software overhead, enhancements to the network routers have been proposed. These enhancements include two additional router features, *message replication* and *intermediate reception*, intended to provide some level of hardware support for multicast operations. Message replication refers to the ability to duplicate incoming messages onto more than one outgoing channel, while intermediate reception is the ability to simultaneously deliver an incoming message to the local processor/memory and to an outgoing channel.

In a unicast-based multicast operation, a message is replicated by the processor at intermediate destination nodes, and these multiple copies are then transmitted to subsequent destination nodes. A seemingly natural extension of a multicast tree is message replication, a tree-based approach using hardware support, where the router is enhanced so that an incoming message can be simultaneously transmitted on two (or more) outgoing links. That is, each flit of a message entering the router can be transmitted by the router on multiple outgoing channels, effectively producing a tree-like message worm. Figure 2.5(a) illustrates a message that is being replicated by a router in a 2D mesh (or torus), while Figure 2.6 shows how message replication can be used to perform a multicast operation in a 2D mesh. In the example shown in Figure 2.6, the message is replicated by the routers at nodes $(1,2)$, $(2,2)$, $(3,2)$, $(4,2)$, and $(5,2)$. Such message worms have headers on each branch of the resulting message tree.

A difficulty with this approach is that when any branch of the tree encounters a required channel that is unavailable, the entire message tree must be blocked, rendering all channels used by the tree unavailable for other communication. That is to say,

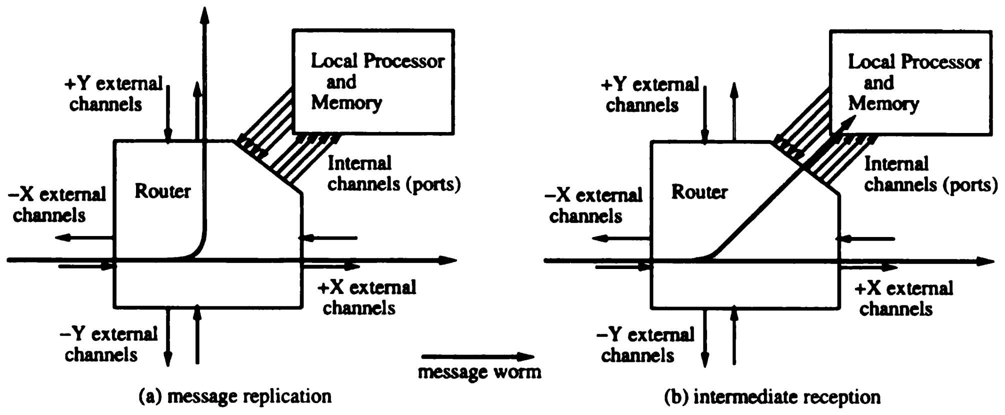


Figure 2.5. Router support for multicast communication

because of the pipelining effects of wormhole routing, when just one branch of the tree is blocked due to channel contention, progress on every branch of the tree is suspended. Even when such a tree is not blocked by channel contention, many communication channels are simultaneously held, and thus unavailable, during the operation. Multicast operations based on message replication are thus highly susceptible to channel contention especially for large destination sets, and must be carefully designed in order to avoid deadlock. Because of these disadvantages, message replication has not been widely supported. Lin, *et al.* [30] present, as a comparison, a multicast method for 2D meshes based on message replication, but do not recommend its use. The nCUBE-2 hypercube [43] includes hardware support for message replication, which is used to implement tree-based broadcast and limited multicast in cases where the destinations form a subcube, but this mechanism is not deadlock-free.

In order to avoid the disadvantages of the tree-like worms produced by message replication, and yet apply hardware support to the implementation of multicast operations, the technique of intermediate reception (IR) has been proposed [30, 31, 32, 33, 34, 59]. A router possessing IR capability is able to copy the flits of a message to the memory of the local processor as the message passes through the router enroute to other destinations, as illustrated in Figure 2.5(b). In this way, a

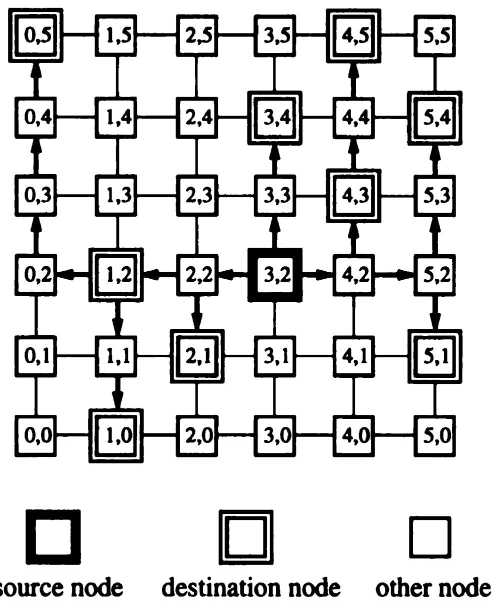


Figure 2.6. A multicast operation using message replication

message originating at a source node can be routed as a single worm through several destination nodes, depositing a copy of the message at each of the intermediate destinations as it passes through. Such communication methods are termed *path-based*, while the constituent messages are called *multi-destination worms*.

Most of the literature dealing with router enhancements for multicast communication is based on IR, and includes the following work. Lin, *et al.* [30] present a path-based multicast routing algorithm for 2D meshes based on Hamiltonian paths. Kim and Kim [31] propose methods to perform multicast communication in the support of parallel-prefix computations on meshes, which are, in turn, incorporated into a proposed matrix multiplication algorithm. Tseng and King [32] describe broadcast and multicast methods for tori. Panda and Singal [33] present methods for broadcast and all-to-all (simultaneous) broadcast in mesh and torus networks. These methods are a hybrid of hardware and software methods, in that they use multi-destination worms, but also employ the processors at intermediate destination nodes to start new multi-destination worms. Ho and Kao [35] have proposed a multi-step path-based broadcast method for hypercubes, in which all messages conform to E-cube routing rules. By adhering to dimension-ordered routing, deadlock is avoided; however, the routing rules are not sufficiently flexible to allow single-path broadcast operations. Panda and Prabhakaran [34] present a path-based multicast method that uses the underlying base routing algorithm, such as deterministic dimension-ordered routing or the adaptive *turn model* [50] routing.

Our work in this area, which is described in Chapters 5 and 6, differs from the previous research in the following ways. The work proposed in [34] is not a multicast routing algorithm, but rather an algorithm for creating multi-destination worms that are consistent with existing unicast routing algorithms. Given the constraints of dimension-ordered routing, certain destination sets cause this method to behave extremely poorly for multicast ($O(m)$ communication steps for $m-1$ destinations, as compared to $\lceil \log_2 m \rceil$ steps for existing software-based methods [19, 21]). Although a multicast routing algorithm for tori is proposed in [32], in order to avoid deadlock this method requires virtual cut-through routing, in which message-sized buffers are required for each channel. The methods of [33] are not deadlock-free in the presence of network traffic in addition to the single collective communication operation. In contrast to the above work, we present deadlock-free path-based multicast routing methods for wormhole-routed torus networks with unidirectional communication links.

CHAPTER 3

Unicast-Based Multicast in All-Port Hypercubes

In this chapter, the specific problem of efficient unicast-based multicast communication for all-port wormhole-routed hypercubes is addressed. Formally, a *hypercube* (or n-cube) consists of 2^n nodes, each of which has a unique n-bit binary address. For each node v, let v also denote its n-bit binary address, and let $\| v \|$ represent the number of 1's in v. A channel $c = (u, v)$ is present in an n-cube if and only if $\| u \oplus v \| = 1$, where $\oplus$ is the bitwise exclusive-or operation on binary numbers. The hypercube topology has been used in multicomputer design for many years [46]. The nCUBE-2 [43] hypercube supports wormhole-routing, as does the recently announced nCUBE-3 [44].

This chapter is organized as follows. Section 3.1 describes the issues and problems involved in supporting efficient multicast communication in all-port hypercube systems. Section 3.2 gives new theoretical results that provide the foundation for this work. Sections 3.3 and 3.4 present the new algorithms that have been designed to support multicast in all-port wormhole-routed hypercubes. Although the multicast problem has been studied previously for one-port architectures [19], the proposed methods improve performance by exploiting the presence of multiple ports. Section 3.5 compares the new algorithms using analysis, simulation, and implementations on a 64-node nCUBE-2, which possesses an all-port architecture. Finally, a summary is given in Section 3.6.

3.1 Issues

Although implemented in software, unicast-based multicast communication algorithms must exploit the underlying architecture in order to minimize their execution time. In a wormhole-routed system, the implementation should not only take advantage of the distance-insensitivity of unicast latency, but must also avoid channel contention, that is, no two messages involved in the operation should simultaneously require the same channel. Avoiding channel contention depends on the underlying unicast routing algorithm of the MPC; hypercubes often adopt *E-cube routing* [56], in which messages are routed through dimensions in either ascending or descending order. Also, the implementation should affect no local processors other than those explicitly involved in the operation. For example, in a multicast operation, only source and destination processors should be required to handle the message. Finally, the implementation should account for the port model, which affects the rate at which nodes can send and receive messages.

The following (small-scale) example illustrates the issues and difficulties involved in implementing efficient multicast communication in hypercubes. We consider the 4-cube in Figure 3.1, and suppose that a multicast message is to be sent from node 0000 to eight destinations {0001, 0011, 0101, 0111, 1011, 1100, 1110, 1111}. In this example and all subsequent examples, we assume that the E-cube routing algorithm resolves addresses from high order bits to low order bits. In the nCUBE-2, the opposite resolution strategy is used, but this difference does not affect any of the results presented.

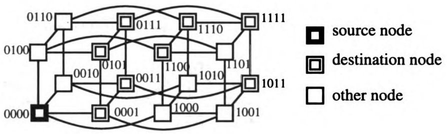


Figure 3.1. An example of multicast in a 4-cube

In early hypercube systems that used store-and-forward switching, the procedure shown in Figure 3.2(a) could be used to implement the multicast operation [60]. At step 1, the source sends the message to node 1000. At step 2, nodes 0000 and 1000 inform nodes 0100 and 1010, respectively. Continuing in this fashion, this implementation requires 4 steps to reach all destinations. In this example, five of the nodes that are required to relay the message (0010, 0100, 0110, 1000, and 1010) are not destinations themselves. Using the same routing algorithm in a one-port wormhole-routed network also requires 4 steps, as shown in Figure 3.2(b). In this case, however, only the *routers* at two of the non-destination nodes (0010 and 0110) are involved in forwarding the message. The message may be passed from node 0000 to node 0011 in one step because it is pipelined through the router at node 0010 rather than being relayed by the local processor at that node. However, because the message must be replicated and forwarded on multiple outgoing channels at nodes 0100, 1000, and 1010, the local processors at those nodes must still handle the message.

Figure 3.2(c) illustrates the result of using the *U-cube algorithm* [19] to solve the problem on a one-port wormhole-routed system. The U-cube algorithm, which was

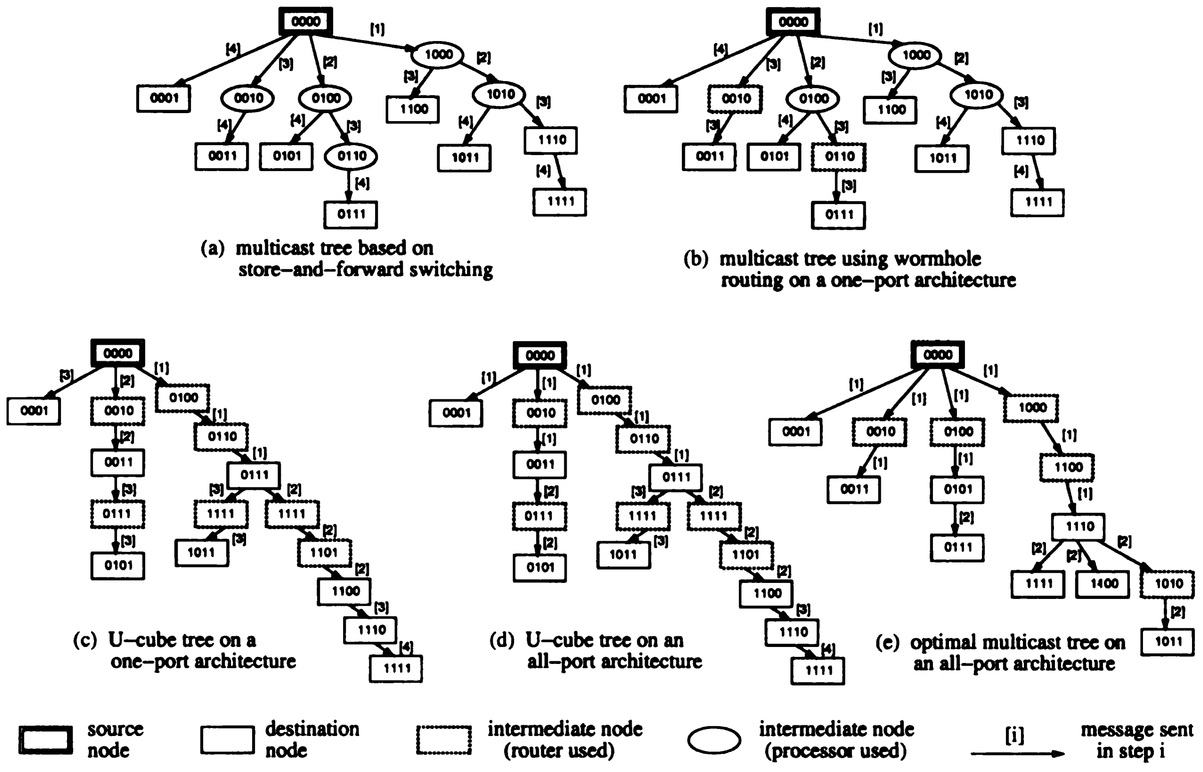

(a) multicast tree based on store-and-forward switching
(b) multicast tree using wormhole routing on a one-port architecture
(c) U-cube tree on a one-port architecture
(d) U-cube tree on an all-port architecture
(e) optimal multicast tree on an all-port architecture
source node
destination node
intermediate node (router used)
intermediate node (processor used)
[i] message sent in step i

Figure 3.2. Unicast-based software multicast trees

designed specifically for one-port wormhole-routed architectures, will be discussed further in Section 3.3. Using this algorithm, the only local processors required to handle the message are those at destination nodes. Furthermore, on a one-port architecture, all messages are guaranteed to be contention-free [19]. Although common channels are used between the 0111-to-1011 path and the 0111-to-1100 path, these messages are sent sequentially, so contention does not occur.

Since the U-cube algorithm was designed for one-port systems it makes no explicit attempt to take advantage of multiple ports between local processors and routers. That is to say, the U-cube algorithm does not actively seek out and use multiple ports in parallel. For example, if the algorithm were implemented on an all-port hypercube, it would still require four steps to complete the multicast in the above example, as illustrated in Figure 3.2(d). Some destinations are reached earlier than in Figure 3.2(c) simply because the algorithm inadvertently uses multiple ports at some nodes simultaneously. Notice that three steps are required to reach destination node 1011, since that unicast message must traverse a channel (0111, 1111) that lies along the path required to reach node 1100, thereby delaying its transmission.

Figure 3.2(e) shows a multicast tree that accounts for both wormhole routing and an all-port architecture. The algorithm requires only two steps, no local processors other than the source and destinations are involved, and contention among constituent messages is avoided. This particular tree is based on the methods presented in this chapter. In the next section, we develop the theoretical results necessary to guarantee that our new algorithms, presented in Sections 3.3 and 3.4, are contention-free.

Under the proposed system model, even the best known methods for broadcasting are heuristic [24], and since the multicast problem is a generalization of broadcast, it is at least as hard as broadcast with respect to computational complexity. We conjecture that generating optimal multicast solutions for an all-port wormhole-routed hypercube is an NP-hard problem. The methods presented in this chapter are therefore heuristic, and thus do not provide optimal solutions in every case (although the multicast tree shown in Figure 3.2(e) is, in fact, optimal for the given set of nodes).

3.2 Theoretical Foundations

In this section, we present new theoretical results that will serve as a basis for subsequent algorithms. First, we formally define terms related to routing and subcubes. We then state and prove several theorems that are useful in determining that certain pairs of paths are guaranteed to be arc-disjoint (and hence, contention-free). Finally, we formally define contention in an all-port hypercube architecture, and prove a related theorem.

3.2.1 Notation and Definitions

Bitwise *exclusive-or* is represented by the symbol $\oplus$; logical *and* and *or* are represented by $\wedge$ and $\vee$, respectively, and $\overline{v}$ is used to represent the bitwise complement of v. The symbol $\|v\|$ denotes the number of non-zero bits in v. We use N to represent the number of processors in the system. Since n represents the dimensionality of the hypercube, $N = 2^n$. The i^{th} bit of address v is denoted by $\sigma_i(v)$, $0 \leq i \leq n-1$, where $\sigma_0(v)$ represents the least-significant address bit; hence, address v can be written as $\sigma_{n-1}(v)\sigma_{n-2}(v) \ldots \sigma_0(v)$. For each node, v, the outgoing (and incoming) channels of node v are labeled 0 through $n-1$, where channel d connects node $v = \sigma_{n-1}(v)\sigma_{n-2}(v) \ldots \sigma_0(v)$ to node $\sigma_{n-1}(v) \ldots \sigma_{d+1}(v)\overline{\sigma_d(v)}\sigma_{d-1}(v) \ldots \sigma_0(v)$. We say that channel d is used to *travel* in dimension d.

Dimension-ordered routing is a minimal deterministic routing algorithm in which every message traverses dimensions of the network in a strict monotonic order. Under dimension-ordered routing, each routing step brings the message one hop closer to the destination, along the highest (alternatively, lowest) dimension in which the current node and the destination node differ. E-cube routing is the hypercube-specific case of dimension-ordered routing.

Definition 3.1 *Given distinct nodes u and v in an n-dimensional hypercube, let i be the highest dimension such that $\sigma_i(u) \neq \sigma_i(v)$. Under* E-cube routing*, a message sent from u to v will be routed first along dimension i to intermediate node $w = \sigma_{n-1}(v)\sigma_{n-2}(v)\ldots\sigma_{i+1}(v)\sigma_i(v)\sigma_{i-1}(u)\ldots\sigma_0(u)$, where $\sigma_i(w) = \overline{\sigma_i(u)}$. At node w, the same routing algorithm is invoked to determine the next intermediate node.*

The E-cube path from a source node u to a destination node v will be denoted $P(u,v) = (u; w_1; w_2; \ldots; w_p; v)$, where the nodes w_i, $1 \leq i \leq p$, are the nodes visited on the path. We note that $p + 1 = \|u \oplus v\|$. In any shortest path from a destination u to a source v, a message will travel exactly once in each dimension

d such that $\sigma_d(u) \neq \sigma_d(v)$. Traveling over these $\|u \oplus v\|$ dimensions in any arbitrary order will result in a shortest path between u and v. For example, the path from source node 0101 to destination node 1110 resulting from E-cube routing is $P(0101, 1110) = (0101; 1101; 1111; 1110)$. A unicast from node u to node v occurring at time step t is denoted $(u, v, P(u, v), t)$. The following definition simplifies references to the initial channel in a dimension-ordered route, that is, the first dimension in which a message will travel.

Definition 3.2 *The symbol $\delta(u,v)$ represents the highest-ordered bit position in which u and v differ. Formally, $\delta(u,v) = \max\{i : 0 \leq i \leq n-1 : \sigma_i(u) \neq \sigma_i(v)\}$. If $u = v$, then $\delta(u,v)$ is undefined.*

In order to identify a *subcube* of the nodes of a hypercube, we may explicitly state some of the n address bits, and allow the other address bits to range over all possible values. In this chapter, we need to work only with subcubes in which the explicitly-stated address bits are the high-order bits, and the free-ranging address bits are the low-order bits. We refer to such subcubes as *$\mathcal{S}$-cubes.*

Definition 3.3 *An $\mathcal{S}$-cube $S = \langle b_{n-1}b_{n-2} \ \dots \ b_{n_S}\rangle$ is defined by a* dimensionality *$n_S \in \{0, \ \dots, \ n\}$, and a $(n - n_S)$-bit* mask *$\langle b_{n-1}b_{n-2} \ \dots \ b_{n_S}\rangle$. Informally, S consists of those nodes whose address is of the form $\langle b_{n-1}b_{n-2} \ \dots \ b_{n_S} * * \ \dots \ *\rangle$, where the $*$'s represent arbitrary bit values. Formally, for any node v, $v \in S$ if and only if $\sigma_i(v) = b_i$, for $n_S \leq i \leq n-1$.*

For example, the $\mathcal{S}$-cube $\langle 0\,1\rangle$ in a 4-dimension hypercube contains the four nodes 0100, 0101, 0110, and 0111, while the $\mathcal{S}$-cube $\langle 1\,1\,0\rangle$ in a 6-dimension hypercube contains the eight nodes 110000, 110001, 110010, 110011, 110100, 110101, 110110, and 110111.

3.2.2 Useful Lemmas

This section contains lemmas that will be used to facilitate the proof of subsequent theorems. These lemmas and their proofs are also useful in understanding later sections of the chapter.

Lemma 3.1 *Let $P(u,v) = (u; w_1; w_2; \ldots; w_p; v)$ be any E-cube path (For clarity, let $w_0 = u$ and $w_{p+1} = v$.), and let $(w_i, w_{i+1}) \in P$ be any arc in $P(u,v)$. Let d be the dimension over which (w_i, w_{i+1}) travels, $\sigma_d(w_i) = \overline{\sigma_d(w_{i+1})}$. Then the following conditions hold:*

1. For all $j \in \{1, \ldots, i\}$ and for all $k \in \{0, \ldots, d\}$, $\sigma_k(w_j) = \sigma_k(u)$
(Before traveling in dimension d, a message does not travel in dimensions less than d.)

2. For all $j \in \{i+1, \ldots, p\}$ and for all $k \in \{d+1, \ldots, n-1\}$, $\sigma_k(w_j) = \sigma_k(v)$
(After traveling in dimension d, a message does not travel in dimensions greater than, or equal to, d.)

3. $\sigma_d(u) \neq \sigma_d(v)$
(A message travels in dimension d only if the source and destination node addresses differ in dimension d.)

Proof: All three assertions follow directly from the behavior of dimension-ordered routing in hypercubes. □

Lemma 3.2 *For any three nodes u, v, x, and for any $\mathcal{S}$-cube S, if $u, x \in S$ and $u \leq v \leq x$, then $v \in S$. (The node addresses within any $\mathcal{S}$-cube are contiguous.)*

Proof: Let S be represented by $\langle b_{n-1} b_{n-2} \ldots b_{n_S} \rangle$, let u, v and x be as specified, and assume that $v \notin S$. Then we have $\sigma_i(u) = \sigma_i(x) = b_i$ for $n_S \leq i \leq n-1$; and $\sigma_j(v) \neq b_j$ for some j, $n_S \leq j \leq n-1$. Let k be the value of the largest such j. Assume, without loss of generality, that $\sigma_k(v) > b_k$. Then for $k+1 \leq i \leq n-1$, $\sigma_i(v) = b_i = \sigma_i(x)$; and $\sigma_k(v) > b_k = \sigma_k(x)$. It follows that $v > x$, a contradiction.

(If $\sigma_k(v) < b_k$, then we conclude similarly that $v < u$, also a contradiction.) □

3.2.3 Arc-Disjoint Paths

In implementing a unicast-based multicast algorithm, whenever the paths of two constituent unicast messages share an arc (channel), care must be taken to ensure that the paths do not attempt to use the shared arc simultaneously, otherwise contention will arise. When two paths have no arc in common, of course, contention between these two particular paths is always avoided. Paths with no common arc are said to be *arc-disjoint.*

Each of the following theorems state sufficient conditions on two paths such that any two paths meeting these conditions are arc-disjoint. Each theorem is stated formally. Where needed for clarity, theorems are stated informally within a parenthetical block of text.

Theorem 3.1 *Consider any two paths $P(u,v)$ and $P(u,y)$ originating from a common source node u in a hypercube. If $\delta(u,v) \neq \delta(u,y)$, then $P(u,v)$ and $P(u,y)$ are arc-disjoint. (Paths leaving a common source on different channels are arc-disjoint.)*

Proof: Without loss of generality, assume that $\delta(u,v) > \delta(u,y)$, and let $d = \delta(u,v)$. Now suppose that there is some node $r \neq u$ contained in both paths: $r \in P(u,v) \wedge r \in P(u,y)$. Since $r \in P(u,v)$ and $\delta(u,v) = d$, then by Lemma 3.1, $\sigma_d(r) \neq \sigma_d(u)$. But since $r \in P(u,y)$ and $\delta(u,y) < d$, then $\sigma_d(r) = \sigma_d(u)$, which is a contradiction. So there cannot exist any node $r \neq u$ contained in both paths. Since paths $P(u,v)$ and $P(u,y)$ do not share any node (except the source node u), they are arc-disjoint. □

Theorem 3.2 *Consider any two paths $P(u,v)$ and $P(x,y)$ in a hypercube. If there exists an S-cube S such that $u,v \in S \ \wedge \ x,y \notin S$, then $P(u,v)$ and $P(x,y)$ are arc-disjoint. (A path with source and destination within S-cube S is arc-disjoint from any path with source and destination outside S.)*

Proof: Suppose there is an arc (w,z) common to both paths. Let d be the dimension in which w and z differ, that is, $\sigma_d(w) = \overline{\sigma_d(z)}$, and $\sigma_i(w) = \sigma_i(z)$ for all i, $0 \leq i \leq n-1$, where $i \neq d$. Let n_S be the dimensionality of S; $S = \langle b_{n-1}b_{n-2} \ \ldots \ b_{n_S}\rangle$.

Case 1: $d \geq n_S$. Since $(w,z) \in P(u,v)$, then $\sigma_d(w) = \sigma_d(u)$ and $\sigma_d(z) = \sigma_d(v)$ (dimension-ordered routing). Since $\sigma_d(w) = \overline{\sigma_d(z)}$, then $\sigma_d(w) \neq \sigma_d(z)$, hence, $\sigma_d(u) \neq \sigma_d(v)$. But since $u,v \in S$ and $d \geq n_S$, we have $\sigma_d(u) = \sigma_d(v)$, a contradiction.

Case 2: $d \leq n_S$. Since $(w,z) \in P(u,v)$ and $d \leq n_S$, then $z \in S$ if and only if $v \in S$. Likewise, since $(w,z) \in P(x,y)$, $z \in S$ if and only if $y \in S$. So $v \in S$ if and only if $y \in S$. But $v \in S$ and $y \notin S$, a contradiction.

Thus, there is no arc common to both paths. □

3.2.4 Avoiding Depth Contention

As previously stated, any two unicasts having arc-disjoint paths are contention-free. One may suspect that two unicasts sent in different steps of a multicast algorithm would also be contention-free, whether or not they are arc-disjoint. However, unicast messages sent in different steps may actually be transmitted concurrently depending on the value of *startup latency*, which includes the system call time at both the source and destination nodes. If startup latency is large, then the steps of a multicast tree may become staggered, causing unicasts in different steps to actually be sent simultaneously. This condition is possible in commercial systems, where the sending and receiving latencies may be much greater than the network latency of a message.

In order to study contention between messages sent in different steps, the definition of the *reachable set* is needed.

Definition 3.4 [19] *Given a multicast implementation, a node v is in the reachable set of a node u, denoted R_u, if and only if one of the following conditions holds:*

1. *$v = u$, or*

2. *There exists a unicast $(x, v, P(x, v), t)$ in the implementation such that $x \in R_u$.*

If the multicast implementation is considered to be a directed tree of unicast messages rooted at the source node d_0, then the reachable set of a node u is the set of nodes in the subtree rooted at node u. In Figure 3.2(e), for example, $R_{1110} = \{1110, 1011, 1100, 1111\}$. Using this definition, the properties of an implementation necessary to avoid contention between messages sent in different steps can be characterized. A multicast implementation is said to be *depth contention-free* if, regardless of overlap in message passing steps caused by startup latency, the constituent messages are contention-free. The following theorem gives sufficient conditions for a multicast implementation to be depth contention-free.

Theorem 3.3 [19] *A multicast implementation is depth contention-free if at least one of the following four conditions holds for every pair of unicasts $(u, v, P(u, v), t)$ and $(x, y, P(x, y), \tau)$ in the implementation, where $t \leq \tau$.*

1. *$P(u, v)$ and $P(x, y)$ are arc-disjoint.*

2. *$x = u$*

3. *$x \in R_v$.*

4. *$x \in R_w$ and the implementation contains the unicast $(u, w, P(u, w), t + k)$, for some node w and positive integer k.*

Proof: We need to show that contention does not arise between any pair of unicast messages in the implementation. We consider two arbitrary unicasts $(u, v, P(u,v), t)$ and $(x, y, P(x,y), \tau)$, with $t \leq \tau$.

Condition 1. If the two paths of the messages, $P(u,v)$ and $P(x,y)$, are arc-disjoint, then the two unicasts are contention-free.

Condition 2. If $x = u$, then we must consider two cases depending on whether or not destination nodes v and y are reached through the same outgoing channel from source node u. If $\delta(u,v) = \delta(x,y)$, then $t < \tau$ since u must send the messages sequentially. Figure 3.3(a) illustrates the situation. Node u sends the message to v before sending it to y. Even if $\tau = t + 1$ and the sending latency is 0, contention will not occur. If, on the other hand, $\delta(u,v) \neq \delta(x,y)$, then by Theorem 3.1, the two unicasts are arc-disjoint, and hence contention-free.

Condition 3. If $x \in R_v$, as shown in Figure 3.3(a), then the u-to-v unicast must be completed before the x-to-y unicast begins, so they are contention-free.

Condition 4. As shown in Figure 3.3(c), node u sends the message to v prior to sending it to node w, which is either an ancestor of x or perhaps x itself. Clearly, node v will have received the message prior to node x, thus preventing contention. □

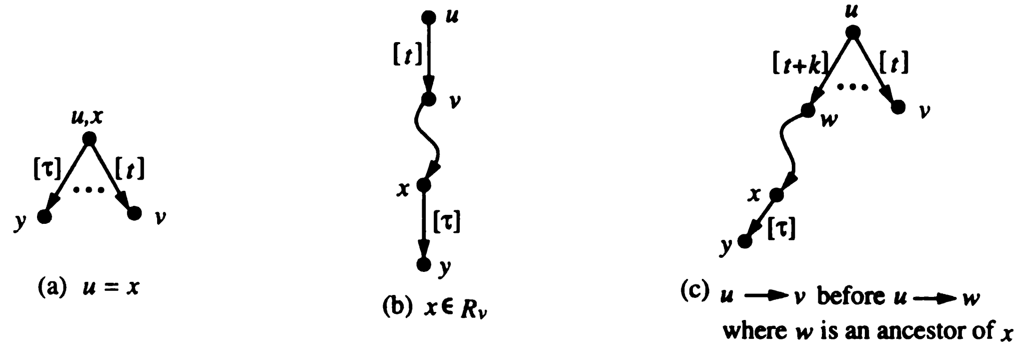

u,x
[τ]
[t]
y
v
(a) $u = x$
u
[t]
v
x
[τ]
y
(b) $x \in R_v$
u
[t+k]
[t]
w
v
x
y
[τ]
(c) $u \longrightarrow v$ before $u \longrightarrow w$
where w is an ancestor of x

Figure 3.3. Conditions 2, 3, and 4 of Theorem 3.3

In the next two sections, we define several multicast algorithms for all-port hypercubes. All the algorithms are depth-contention free. Their performance, which is compared in Section 3.5, depends largely on how well they take advantage of the presence of multiple ports.

3.3 Algorithms Based on Dimension-Ordered Chains

The algorithms considered in this section are extensions of the U-cube algorithm [19], which was mentioned in Section 3.1. The new algorithms were constructed by modifying the U-cube algorithm so as to make better use of multiple ports between each node and its router.

We begin with a brief review of the U-cube algorithm. This algorithm, designed for one-port architectures, produces multicast trees on such systems that are of minimum height and are guaranteed to be contention-free. The U-cube multicast algorithm relies on the binary relation "dimension order," denoted $<_d$, which is defined between two nodes u and v as follows: $u <_d v$ if and only if either $u = v$, or there exists a j such that $\sigma_j(u) < \sigma_j(v)$ and $\sigma_i(u) = \sigma_i(v)$ for all i, $j+1 \leq i \leq n-1$. A sequence $\{d_0, d_1, d_2, \ldots, d_p\}$ of source and destination addresses in which all the elements are distinct and $d_i <_d d_j$ for all $0 \leq i < j \leq p$ is called a *dimension-ordered chain* [19]. A sequence $\{d_1, d_2, \ldots, d_m\}$ is called a *d_0-relative dimension-ordered chain* if and only if $\{d_0 \oplus d_1, d_0 \oplus d_2, \ldots, d_0 \oplus d_m\}$ is a dimension-ordered chain.

If address resolution is performed from highest (left) to lowest (right), then dimension order is the same as the usual increasing order. For example, dimension ordering of 10100, 00010, and 10010 results in the chain: {00010, 10010, 10100}, since 00010 $<_d$ 10010 $<_d$ 10100. Alternatively, on systems in which addresses are resolved from lowest to highest, the dimension-ordered chain is: {10100, 00010, 10010}.

In the U-cube algorithm, the source d_0 and the destination addresses are sorted into a d_0-relative dimension-ordered chain, denoted Φ, at the time when the multicast is initiated. The source node successively divides Φ in half and sends a message to the first node in the upper half of the chain. That destination node is responsible for delivering the message to the other nodes in the upper half, using the same U-cube algorithm. At each step, the source deletes from Φ the address of the receiving node and those nodes in the upper half of the chain. The source continues this procedure until Φ contains only its own address.

Figure 3.4 gives an example of this method in a one-port 4-cube. The source node 0100 is sending to a set of eight destinations {0001, 0011, 0101, 0111, 1000, 1010, 1011, 1111}. Taking the exclusive-or of each destination address with 0100 and sorting the results produces the dimension-ordered chain Φ = {0000, 0001, 0011, 0101, 0111, 1011, 1100, 1110, 1111}. (The reader will notice that this chain Φ represents the same multicast operation examined in Figure 3.2.) The corresponding U-cube tree is shown in Figure 3.4; it takes 4 steps for all destination processors to receive the message.

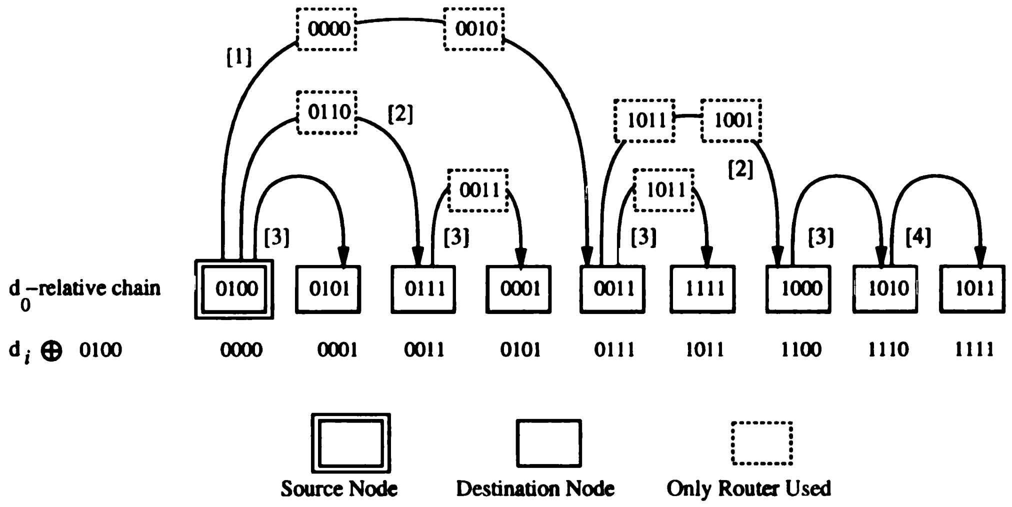


Figure 3.4. Multicast chain in a one-port 4-cube

It has been previously shown that message transmission in a U-cube tree is contention-free regardless of startup latency and message length [19]. Furthermore, the U-cube algorithm executes in minimum time for a one-port architecture by requiring only $\lceil log_2(m+1) \rceil$ time steps for m destinations. For details of the theory underlying the U-cube and the accompanying U-mesh algorithm, please refer to [19].

Since the U-cube algorithm was designed for one-port architectures, it makes no attempt to parallelize message transmissions from a given sender by using multiple ports. When executed on an all-port hypercube, the algorithm will often fail to take advantage of that architectural property. In the tree shown in Figure 3.2(d), for example, step 1 of the algorithm "mistakenly" selects node 0111 as the first destination to which the message is transmitted. This decision leaves node 0111 responsible for delivering the message to four nodes, all of which differ from 0111 in the highest dimension. Better message-forwarding decisions, shown in Figure 3.2(e), result in a tree of height two instead of four.

This observation leads to two variations on the U-cube algorithm called *Maxport* and *Combine*. Both algorithms differ from U-cube in a single statement, which determines the degree to which they exploit the all-port capability of the system. Figure 3.5 gives the generalized multicast algorithm, which encompasses all three algorithms. If Step 4(a) is executed, then the algorithm is identical to the U-cube algorithm [19].

In the *Maxport* algorithm, a sender transmits (in parallel) to the maximum number of destinations permitted by the architecture and the specific destination set. Step 4(b) in the body of the main loop of the generalized multicast algorithm is executed, setting $next = highdim$, rather than $next = center$. This choice can sometimes lead to performance worse than U-cube, however. For example, if node 0000 is the source of a multicast to nodes 1001, 1010, and 1011, then the resulting Maxport "tree" will require three steps, as shown in Figure 3.6(a). The U-cube solution shown in Figure 3.6(b) requires only two steps.

```
Algorithm 1: Generalized Multicast Algorithm
Input: Dimension ordered address sequence
       $\{d_{left}, d_{left+1}, \ldots, d_{right}\}$, where $d_{left}$
       is the local address, and a message $M$.
Output: Send out one or more copies of message M
Procedure:
  repeat
    1. Set $k = \delta(d_{left}, d_{right})$, the position of the first bit
       difference
    2. Let $d_{highdim}$ be the leftmost destination in the
       chain such that $\delta(d_{left}, d_{highdim}) = k$
    3. Set $center = left + \lceil \frac{right-left}{2} \rceil$
    4. Set $next$ according to algorithm variation
       a. $next = center$ /* U-cube */
       b. $next = highdim$ /* Maxport */
       c. $next = \max(highdim, center)$ /* Combine */
    5. $D = \{d_{next}, d_{next+1}, \ldots, d_{right}\}$;
    6. Send a copy of message $M$ to node $d_{next}$ with
       the address field $D$
    7. $right = next - 1$
  until ($left = right$)
```

Figure 3.5. Generalized multicast algorithm

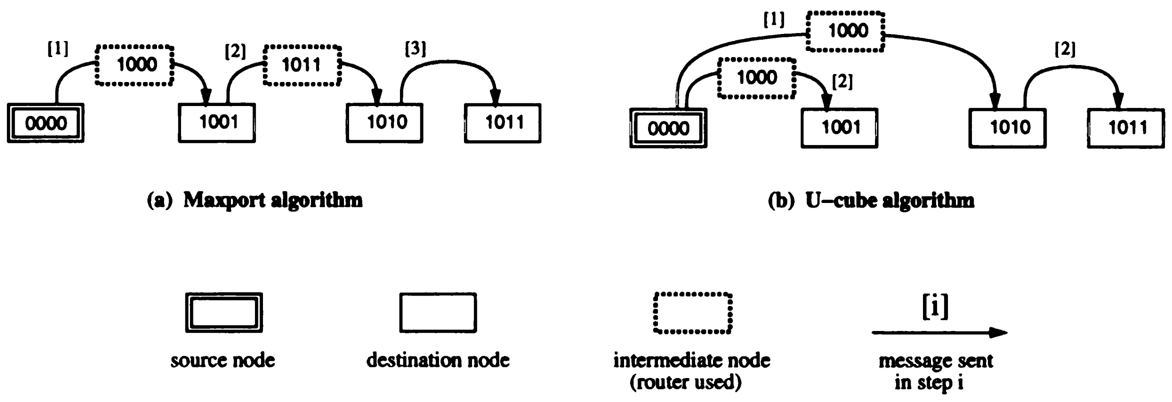


Figure 3.6. Simple Maxport and U-cube comparison

Just as U-cube does not account for dimension, neither does Maxport account for the number of destinations for which each node is responsible. A simple modification to the algorithm addresses this problem. As the name implies, the *Combine* algorithm exhibits characteristics of both the *U-Cube* and *Maxport* algorithms. This algorithm attempts to use multiple ports, but not at the expense of leaving a single node responsible for a large subset of the destinations. In order to obtain the *Combine* algorithm, Step 4(c) in the body of the main loop is executed, setting $next = \max(highdim, center)$. The performance of all three algorithms is compared in Section 3.5.

3.4 An Algorithm Based on Cube-Ordered Chains

In this section, we present an alternative approach to multicasting, in which the source node and destination nodes are considered as elements of $\mathcal{S}$-cubes.

Definition 3.5 *A chain* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *is a* cube-ordered chain of dimension n *if and only if*

1. For all $d \in D$, $0 \le d \le 2^n - 1$; and
2. For all $\mathcal{S}$-cubes S, and for all i, j, k where $first \le i \le j \le k \le last$, if $d_i, d_k \in S$, then $d_j \in S$.

(A chain D is cube-ordered if and only if the nodes of D within any $\mathcal{S}$-cube are contiguous.)

Figure 3.7 illustrates three cube-ordered chains in a 16-node hypercube. Although each chain contains the same set of node addresses, these addresses appear in three different orders. In Figure 3.7(a), the cube-ordered chain is $\{0, 1, 3, 5, 7, 11, 12, 14, 15\}$. Notice that for each $\mathcal{S}$-cube, of different sizes, the nodes of the chain within the

$\mathcal{S}$-cube are contiguous. In Figure 3.7(b), two halves of an $\mathcal{S}$-cube, each of which is in turn an $\mathcal{S}$-cube, have been interchanged. As can be seen, the resulting address sequence $\{0, 1, 3, 5, 7, 12, 14, 15, 11\}$ is also a cube-ordered chain. Figure 3.7(c) shows an additional interchange of $\mathcal{S}$-cube halves, resulting in a third cube-ordered chain $\{0, 1, 3, 5, 7, 14, 15, 12, 11\}$. The notion of interchanging $\mathcal{S}$-cube halves within a cube-ordered chain is important to an algorithm described later in this section.

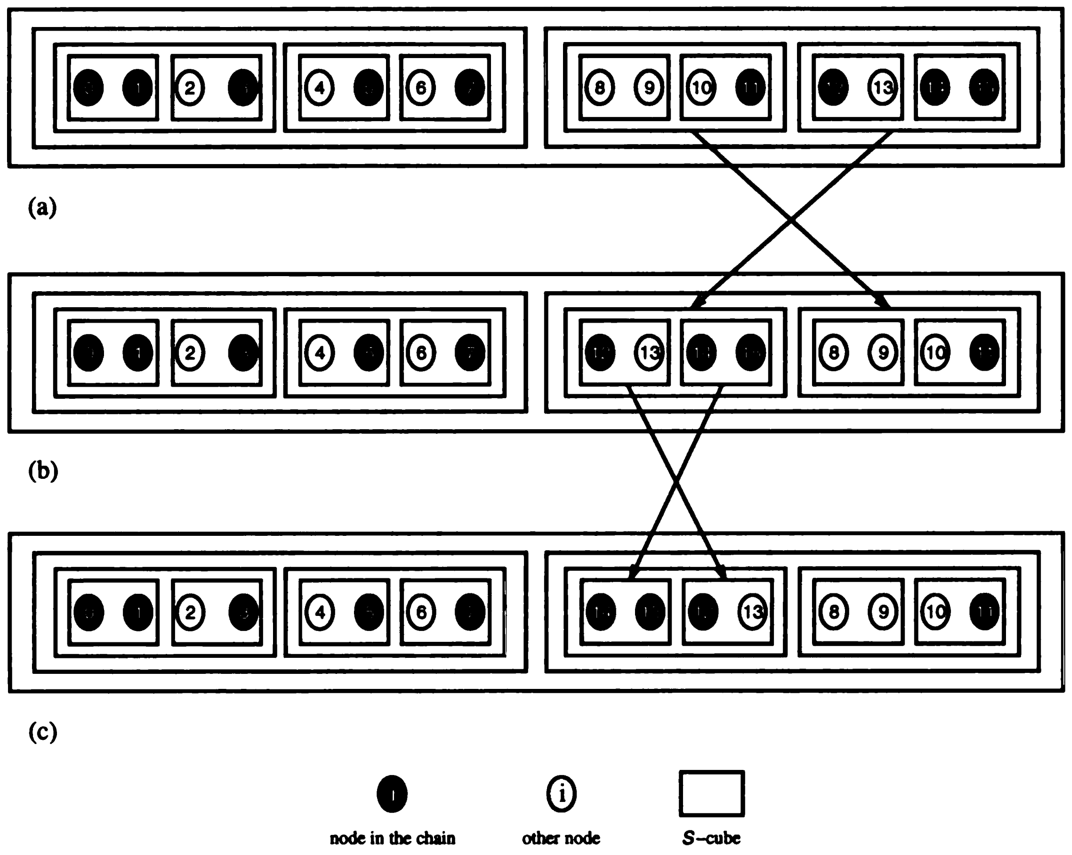


Figure 3.7. Cube-ordered chains of dimension 4

Theorem 3.4 *Every dimension-ordered chain is also a cube-ordered chain.*

Proof: Let $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ be any dimension-ordered chain. Thus, for all $i, j \in \{first, \ldots, last\}$, if $i < j$, then $d_i < d_j$. The theorem now follows directly from Lemma 3.2. □

In the Maxport algorithm, for each participating node, say v, the unicasts originating at node v are transmitted on different outgoing channels. In this approach, the message is always forwarded to different $\mathcal{S}$-cubes. When node v receives the message over channel d, it also receives a list of destination nodes, D, which are in the same d-dimension $\mathcal{S}$-cube as v, say $\mathcal{S}$-cube S. In turn, v issues one unicast into each $\mathcal{S}$-cube *within* S which (1) does not contain v, (2) is maximal, and (3) contains at least one destination node. As will be shown later, it is possible to input any cube-ordered chain, not just a dimension-ordered chain, to Maxport and still avoid contention among messages. An ordinary dimension-ordered chain may not be the most appropriate cube-ordered chain to use, however. In fact, performance increase may be gained by exchanging $\mathcal{S}$-cubes of the chain, where possible, so that source nodes (including intermediate source nodes in the multicast tree) always choose the most "crowded" destination node among available destination nodes.

Figure 3.8 shows the *WeightedSort* algorithm, which permutes a cube-ordered chain so that the most "crowded" node appears as the first node of each $\mathcal{S}$-cube. This task is accomplished by exchanging $\mathcal{S}$-cube halves (these halves are themselves $\mathcal{S}$-cubes) so that the most populated half occurs first in the chain. Notice that the *CubeCenter* function is applied to a cube-ordered chain of addresses that are contained within an $\mathcal{S}$-cube of dimension n_S. This function returns the starting position of the second $(n_S - 1)$ dimension $\mathcal{S}$-cube "half" of the input $\mathcal{S}$-cube. If one of the $(n_S - 1)$ dimension $\mathcal{S}$-cubes contains no destination nodes, then CubeCenter returns a value of $last + 1$.

```
Procedure: WeightedSort ($D$, $first$, $last$, $n_S$)
Input: Cube-ordered chain $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$
    and an $\mathcal{S}$-cube dimension $n_S$.
Output: Upon exit, $D$ is a weighted cube-ordered chain.
Procedure:
  if $last - first \geq 2$ then
    $center = CubeCenter\ (D, first, last, n_S)$
    WeightedSort($D$, $first$, $center - 1$, $n_S - 1$)
    WeightedSort($D$, $center$, $last$, $n_S - 1$)
    if ($first \neq 0$) $\wedge$
      (($center - first$) $<$ ($last - center + 1$)) then
      /* swap $\mathcal{S}$-cubes */
      $D = \{d_{center}, d_{center+1}, \ldots, d_{last},$
          $d_{first}, d_{first+1}, \ldots, d_{center-1}\}$
    endif
  endif
```

Figure 3.8. The *WeightedSort* procedure

In order to use the WeightedSort algorithm with Maxport, the list of destinations is first sorted according to dimension-order, then sorted using the weighted sort algorithm, and finally input to Maxport. We call the combination of these techniques the *W-sort* routing algorithm.

Figure 3.9 illustrates the advantage of the W-sort algorithm in a 4-cube. As shown in Figure 3.9(a), the set of destination nodes is $D = \{0, 1, 3, 5, 7, 11, 12, 14, 15\}$. (Their binary equivalents are given for reference.) Since the nodes of D are in ascending order, D is a cube-ordered address sequence, by Theorem 3.4. Figure 3.9(a) shows the U-cube algorithm executed on an all-port architecture, which requires four time steps to perform the multicast. Each arc represents a unicast, and is labeled with the time step in which it occurs. In this example, intermediate routers are not represented. Notice that node 7 cannot send to nodes 11 and 12 during the same time step, since both unicasts require the same outgoing channel.

Figure 3.9(b) shows the Maxport algorithm applied directly to address sequence D. In this example, the Maxport algorithm also requires four steps to reach all destination nodes. All unicasts with a common source node are transmitted on different outgoing channels, and thus can be sent during the same time step in an all-port architecture.

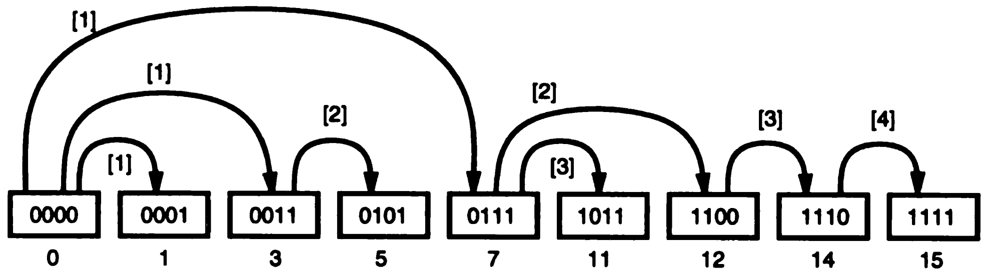
Now, we consider rearranging the nodes in the destination address sequence before beginning the multicast. As illustrated in Figure 3.7, applying the WeightedSort algorithm to address sequence D produces a new address sequence $D = \{0, 1, 3, 5, 7, 14, 15, 12, 11\}$. $\mathcal{S}$-cube $S = \langle 1 \rangle$ contains destination nodes $\{11, 12, 14, 15\}$. The two halves of $\mathcal{S}$-cube S, $S_0 = \langle 1\,0 \rangle$ and $S_1 = \langle 1\,1 \rangle$, contain destination nodes $\{11\}$ and $\{12, 14, 15\}$, respectively. Thus, the WeightedSort algorithm interchanges S_0 and S_1, since S_0 contains fewer destination nodes than S_1. This interchange results in the more populated $\mathcal{S}$-cube (S_1) receiving the message first. Continuing recursively, the two halves of $\mathcal{S}$-cube S_1 are also interchanged. Figure 3.9(c) shows the resulting W-sort multicast, which requires only 2 steps.

Theorem 3.5 *The WeightedSort algorithm applied to a cube-ordered chain* $D = \{d_{first}, d_{first+1}, \ldots, d_{last}\}$ *results in* $\hat{D} = \{\hat{d}_{first}, \hat{d}_{first+1}, \ldots, \hat{d}_{last}\}$, *where:*

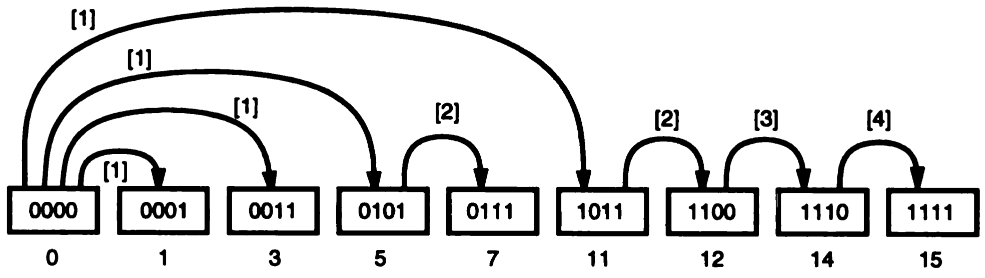
1. $\hat{D}$ is a cube-ordered chain;
2. $\hat{D}$ is a permutation of D; and
3. $\hat{d}_{first} = d_{first}$ (the source node remains in the first position).

Proof: With regard to the second assertion of the theorem, the algorithm contains only one statement that modifies the address sequence D. Since this statement does nothing more than permute the elements of D, the final result of the algorithm will be a permutation of D.

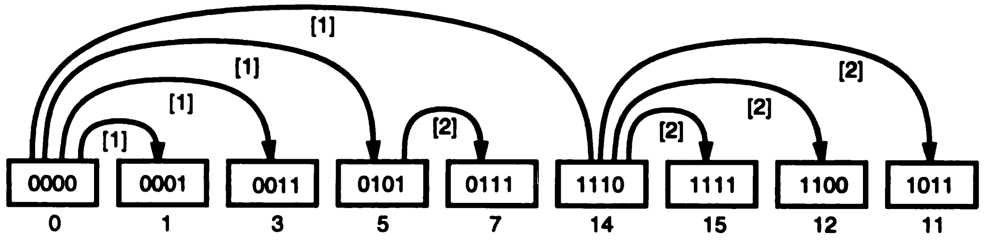
The third assertion of the theorem is confirmed by examining the second **if** statement of the algorithm: the "$(first \neq 0)$" guard clause prevents modification of any portion of D containing d_0. Thus, $\hat{d}_0 = d_0$.



(a) U–cube multicast algorithm



(b) Maxport multicast algorithm



(c) W–sort multicast algorithm

Figure 3.9. Examples of multicast communication

Upon application of the algorithm, the elements of $\{d_{first}, d_{first+1}, \ldots, d_{last}\}$ are members of $\mathcal{S}$-cube $S = \langle b_{n-1}b_{n-2} \ldots b_{n_S}\rangle$. S consists of two "halves," $S_0 = \langle b_{n-1}b_{n-2} \ldots b_{n_S}0\rangle$ and $S_1 = \langle b_{n-1}b_{n-2} \ldots b_{n_S}1\rangle$.

If $d_{first} \in S_0$, then let $S_{first} = S_0$ and let $S_{last} = S_1$. Otherwise ($d_{first} \notin S_0$), let $S_{first} = S_1$ and let $S_{last} = S_0$. Since D is a cube-ordered chain, members of S_{first} must appear contiguously in D. Likewise for S_{last}. So if $S_{first} \neq \emptyset$ and $S_{last} \neq \emptyset$, then there is a unique value $center \in \{first + 1, \ldots, last\}$ such that $d_{center-1} \in S_{first}$ and $d_{center} \in S_{last}$. The *CubeCenter* function provides this unique value of *center.*

Initially, the WeightedSort algorithm is invoked with the input parameter n_S equal to the cube dimension n, thus satisfying the input condition requiring that the destination nodes $\{d_{first}, d_{first+1}, \ldots, d_{last}\}$ contain only elements of an $\mathcal{S}$-cube of dimension n_S (since an $\mathcal{S}$-cube of dimension n corresponds to the entire hypercube). The algorithm is then called recursively for $\mathcal{S}$-cubes S_{first} and S_{last}, thereby preserving invariance of the input condition.

Because $\{d_{first}, d_{first+1}, \ldots, d_{center-1}\}$ and $\{d_{center}, d_{center+1}, \ldots, d_{last}\}$ represent the elements of D that are members of S_{first} and S_{last}, respectively, the statement that assigns to D a permutation of D only interchanges $\mathcal{S}$-cubes S_{first} and S_{last} within $\mathcal{S}$-cube S. (We note that S_{first} and S_{last} form two disjoint "halves" of S: $S_{first} \cup S_{last} = S$ and $S_{first} \cap S_{last} = \emptyset$.)

Clearly, the interchange of two $\mathcal{S}$-cubes (S_{first} and S_{last}) forming halves of a larger $\mathcal{S}$-cube (S) cannot alter the contiguity of elements within *any* $\mathcal{S}$-cube. Hence, the WeightedSort algorithm, when applied to a cube-ordered chain, results in a cube-ordered chain, and the first assertion of Theorem 3.5 is true. □

Theorem 3.6 *The W-sort algorithm applied to a cube-ordered chain $D = \{d_{left}, d_{left+1}, \ldots, d_{right}\}$ results in a contention-free multicast from source node d_{left} to destination nodes $\{d_{left+1}, d_{left+2}, \ldots, d_{right}\}$.*

Proof: First, we establish some facts about how the Maxport algorithm divides the address sequence $\{d_{left}, d_{left+1}, \ldots, d_{right}\}$ during each iteration of the **repeat-until** loop (Figure 3.5).

During the first iteration of the loop, the variable *highdim* acts to divide the list of destination nodes into two $\mathcal{S}$-cubes, S_{local} and S_{remote}, such that

$\{d_{left}, d_{left+1}, \ldots, d_{highdim-1}\} \subseteq S_{local}$ and $\{d_{highdim}, d_{highdim+1}, \ldots, d_{right}\} \subseteq S_{remote}$. After S is divided into S_{local} and S_{remote}, the local node (d_{left}) then sends the message to node $d_{highdim}$ with address field $\{d_{highdim}, d_{highdim+1}, \ldots, d_{right}\}$, thereby relinquishing to node $d_{highdim}$ responsibility for the destination nodes within S_{remote}. In subsequent iterations of the **repeat-until** loop, S_{local} is repeatedly divided into pairs of $\mathcal{S}$-cubes, until $S_{local} = \{d_{left}\}$, at which time the algorithm at node d_{left} terminates.

We make the following assertion: during each iteration of the loop, the address sequences $A_{local} = \{d_{left}, d_{left+1}, \ldots, d_{highdim-1}\}$ and $A_{remote} = \{d_{highdim}, d_{highdim+1}, \ldots, d_{right}\}$ are cube-ordered chains; furthermore, if S_{local} and S_{remote} are the minimal $\mathcal{S}$-cubes such that $A_{local} \subseteq S_{local}$ and $A_{remote} \subseteq S_{remote}$, then $S_{local} \cap S_{remote} = \emptyset$.

In order to see that the assertion is valid, we note that the input sequence to Maxport, $A = \{d_{left}, d_{left+1}, \ldots, d_{right}\}$, is a cube-ordered chain. Since address sequences A_{local} and A_{remote} are subsequences of A, then it follows from Definition 3.5 that A_{local} and A_{remote} are also cube-ordered chains.

In order to show that S_{local} and S_{remote} are disjoint, suppose there is some node $w \in S_{local} \cap S_{remote}$. Since $d_{highdim}$ is the *leftmost* node in the chain such that $\delta(d_{left}, d_{highdim}) = k = \delta(d_{left}, d_{right})$, then for all i, $left \leq i \leq highdim - 1$, $\delta(d_{left}, d_i) < k$. Similarly, for all j, $highdim \leq j \leq right$, $\delta(d_{highdim}, d_j) < k$. Since $w \in S_{local}$, and since S_{local} is the *minimal* $\mathcal{S}$-cube such that $\{d_{left}, d_{left+1}, \ldots, d_{highdim-1}\} \subseteq S_{local}$, then $\delta(d_{left}, w) < k$. Similarly, $\delta(d_{highdim}, w) < k$. But it then follows that $\delta(d_{left}, d_{highdim}) < k$, a contradiction. Thus, $S_{local} \cap S_{remote} = \emptyset$.

Let S^i_{local} and S^i_{remote} be the respective values of S_{local} and S_{remote} during the i^{th} iteration of the **repeat-until** loop, $i \geq 1$; and for notational convenience, let $S^0_{local} = A$.

Arc contention between constituent unicasts of the W-sort algorithm is avoided in the following way. We consider any two unicasts $\mathcal{U}_1 = (u, v, P(u, v), t)$ and $\mathcal{U}_2 = (x, y, P(x, y), \tau)$ produced by the W-sort algorithm. There are two cases to consider with regard to unicast $\mathcal{U}_1$: either (1) $\mathcal{U}_1$ originates from the source node d_{left} (that is, $u = d_{left}$), or (2) $\mathcal{U}_1$ is produced by one of the destination nodes relinquished by node d_{left} (that is, $u, v \in S^i_{remote}$, for some $i \geq 1$).

In the first case ($u = d_{left}$), if unicast $\mathcal{U}_2$ also originates from node d_{left} (that is, $u = x = d_{left}$), then Condition 2 of Theorem 3.3 ensures that $\mathcal{U}_1$ and $\mathcal{U}_2$ are contention-free. Otherwise, $\mathcal{U}_2$ must occur within one of the $\mathcal{S}$-cubes relinquished by d_{first} (that is, $x, y \in S^i_{remote}$, for some $i \geq 1$). Since $\mathcal{U}_2$ is in this case an ancestor of $\mathcal{U}_1$, by Condition 3 of Theorem 3.3, $\mathcal{U}_1$ and $\mathcal{U}_2$ are contention-free.

In the second case ($u, v \in S^i_{remote}$, for some $i \geq 1$), we must consider two forms of potential contention: contention *between* $\mathcal{S}$-cubes, and contention *within* $\mathcal{S}$-cubes.

In the case of potential contention between $\mathcal{S}$-cubes, we have the following situation: $u, v \in S^i_{remote}$ and $x, y \in S^j_{remote}$, where $i \neq j$. Since for all $i \geq 1$, $S^i_{remote} \subseteq S^{i-1}_{local}$, $S^i_{local} \subseteq S^{i-1}_{local}$, and $S^i_{local} \cap S^i_{remote} = \emptyset$, then for all $i, j \geq 1$ where $i \neq j$, $S^i_{remote} \cap S^j_{remote} = \emptyset$. Applying Theorem 3.2 shows that $\mathcal{U}_1$ and $\mathcal{U}_2$ are arc-disjoint, and thus contention-free.

Potential contention within $\mathcal{S}$-cubes involves the following situation: $u, v, x, y \in S^i_{remote}$ for some $i \geq 1$, in which case $\mathcal{U}_1$ and $\mathcal{U}_2$ are the product of the same recursive invocation of the Maxport algorithm, and by the above argument, such an invocation of Maxport produces only contention-free unicasts.

Having covered all possible cases, we now conclude that the unicasts of the W-sort algorithm are pairwise contention-free. $\square$

The W-sort algorithm places certain computational requirements on the source node processor. Recall that the W-sort algorithm requires that the source node (1) sort the list of destination nodes into a dimension-ordered chain, (2) invoke the WeightedSort algorithm to produce a cube-ordered chain, and (3) execute the Maxport algorithm.

Sorting the list of m destination nodes into a dimension-ordered chain can be done in $O(m\,log_2\,m)$ time. The worst-case computational complexity of the WeightedSort algorithm occurs when the input chain is split after the first element; that is, when the value of *center* is equal to *first* + 1 (Figure 3.8). The CubeCenter function can be implemented with a simple binary search, which executes in $O(log_2\,k)$ time on an input of size k. This approach gives a worst-case total of $O(m\,log_2\,m)$ comparison operations for the WeightedSort algorithm; while the statement that permutes D produces a corresponding total of $O(m^2)$ address copies. The resulting total worst-case computational complexity of W-sort is $O(m^2)$.

In many cases, the computational complexity of W-sort may not be important, particularly when a set of destination nodes remains constant over many multicast operations. In this case, WeightedSort can be executed once to produce the appropriate cube-ordered chain, and Maxport can then be executed repeatedly on this fixed chain. However, there may be cases in which a computational requirement less than $O(m^2)$ would be advantageous. Since the Maxport algorithm has a computational complexity of only $O(m)$ at the local node, it would be useful to distribute, and thus parallelize, some of the work of the WeightedSort operation, thereby reducing the computational requirements placed on the source node by the W-sort algorithm.

If, rather than using WeightedSort to permute the address sequence D, the source node merely *identifies* the appropriate destination nodes (and of course, transmits to these destination nodes the message along with a list of relinquished destinations), then the worst-case computational complexity of W-sort can be reduced to $O(m\,log_2\,m)$ at the local node. By Definition 3.5, the address sequence to be relinquished to any destination node will be contiguous in a cube-ordered chain; thus, no permutation of the addresses in D is required.

3.5 Performance Evaluation

In order to understand the relative performance of the algorithms presented in Sections 3.3 and 3.4, they have been compared in three ways on destination sets in which the nodes are randomly distributed throughout the hypercube. First, we compared their performance in terms of the maximum number of steps required to reach the destinations. Second, we compared the algorithms by implementing them on an nCUBE-2 and measuring the average and maximum delay, across destinations. Third, we simulated the performance of the algorithms using a simulation tool that has been validated against the nCUBE-2. Since we had access to a real system with only 64 nodes, only simulation allowed us to compare the algorithms on larger systems.

Each destination set was produced by selecting, from all nodes in the system, the appropriate number of unique nodes under a uniform distribution model, using a random number generator. The source node was always assumed to be node 0. Due to the symmetry of the hypercube topology, there is a homomorphism between the multicasts originating at a particular source node, and those originating at any other source.

3.5.1 Stepwise Comparisons

Figures 3.10 and 3.11 plot the averages, among random sets of destinations, of the maximum number of steps needed to multicast data in a 6-cube and a 10-cube, respectively. For each point in a curve, 100 destination sets were chosen randomly. In addition to reducing the number of steps, the new algorithms "smooth out" the staircase behavior of the U-cube algorithm. As shown in these plots, the W-sort algorithm performs significantly better than the other algorithms. This performance improvement is due to the actions of WeightedSort, which cause destination nodes within more populated $\mathcal{S}$-cubes to migrate toward the root of the multicast tree, thereby allowing the use of multiple ports to occur earlier in the multicast.

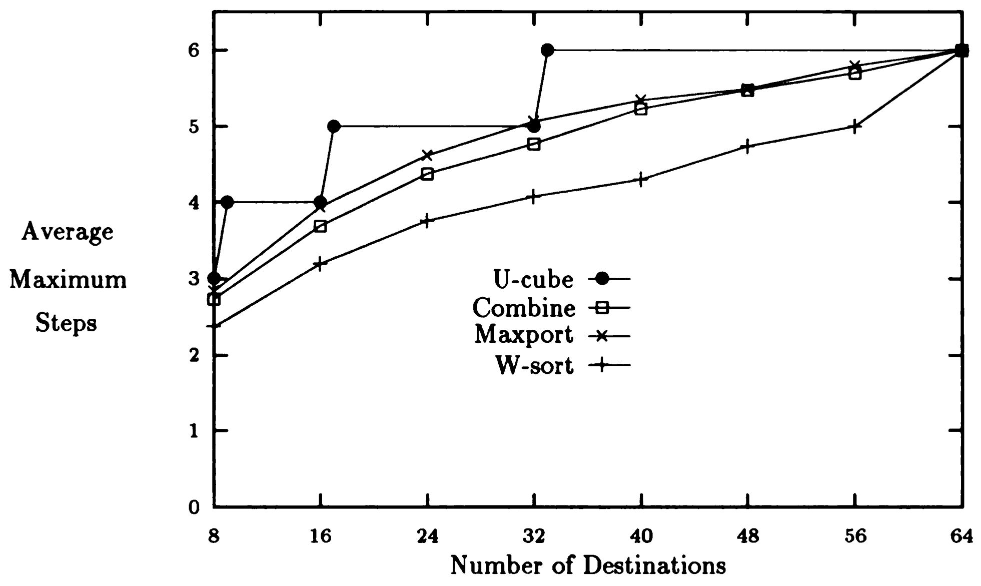


Figure 3.10. Stepwise comparisons on a 6-cube

All of the curves converge at the highest data point, which represents the special case of broadcast. This behavior results from the degeneration of all of the algorithms to the same algorithm in the case of broadcast, where the set of nodes involved in the multicast consists of every node in the system.

3.5.2 Implementations on an nCUBE-2

Figures 3.12 and 3.13 plot the average and maximum, respectively, among destinations, of the measured delay between the sending of a 4096-byte multicast message and its receipt at the destination. For each point in a curve, 20 destination sets were chosen randomly in a 5-cube. These plots show that all the algorithms designed to

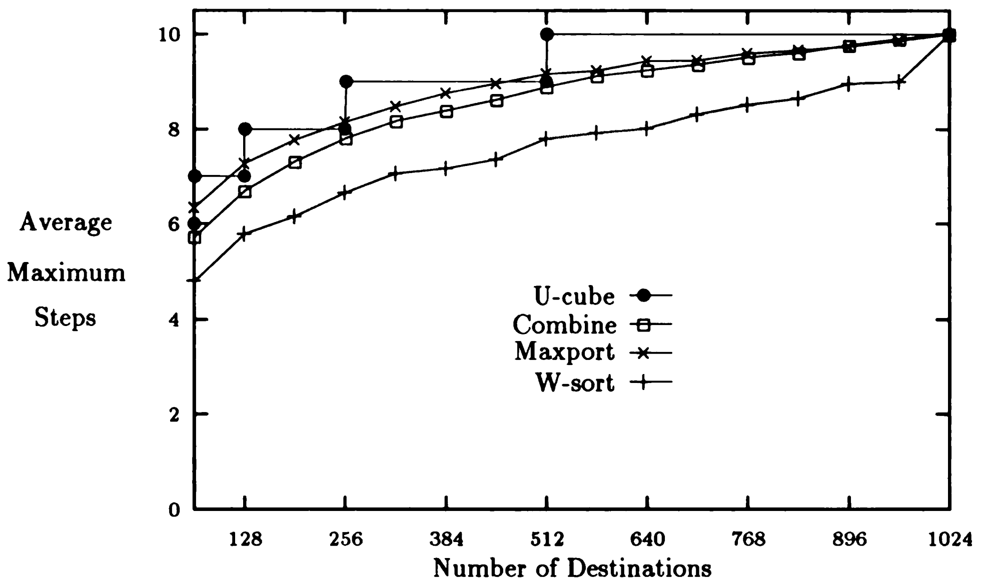


Figure 3.11. Stepwise comparisons on a 10-cube

take advantage of the all-port architecture offer some benefit over the U-cube algorithm. However, any advantage among Maxport, Combine, and W-sort, is unclear. Interestingly, Figure 3.12 shows that the average delay for U-cube is actually worse for multicast than for broadcast. This anomaly occurs because the algorithm sometimes transmits multiple messages along the same channel instead of taking advantage of multiple channels. In Figure 3.13, we see clearly the staircase behavior of U-cube. As predicted by the stepwise comparisons, the new algorithms tend to smooth the relative delays among various sized destination sets.

We infer that the relatively similar results among the new algorithms, and in particular, the lack of a clear advantage for the W-sort algorithm, is due to the startup latency of the nCUBE-2, which prevents the machine from taking full advantage of the all-port architecture. In the nCUBE-2, the sending latency is about 107 μsec and the receiving latency is about 80 μsec. For small messages, these latencies dominate the network latency; the system behaves much like a one-port architecture, and there

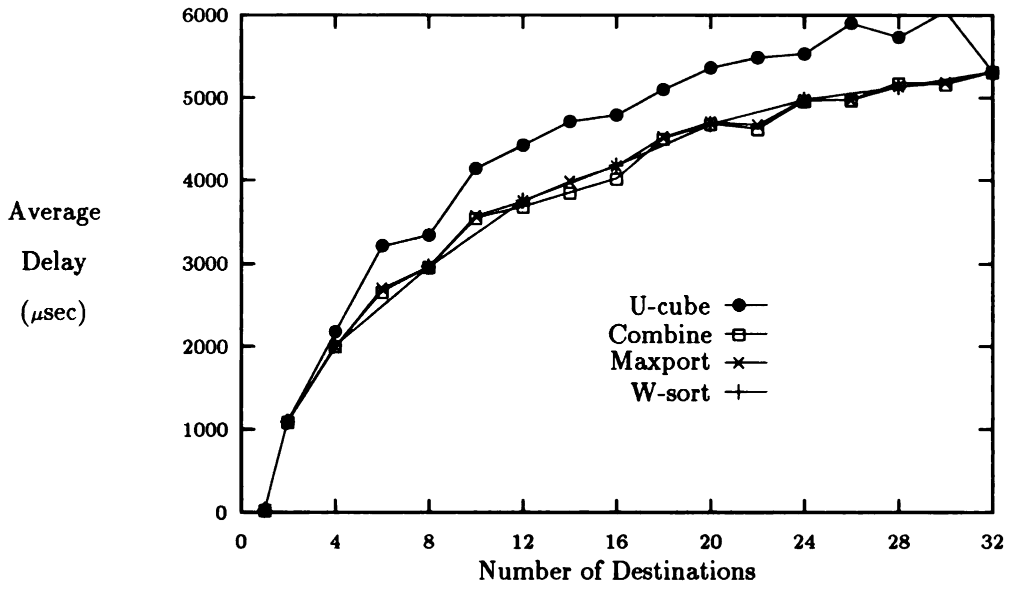


Figure 3.12. Average delay comparisons on a 5-cube

is less difference between the various multicast algorithms. For larger messages, the network latency becomes more significant compared to startup latency. One would expect the performance advantage of the W-sort algorithm to improve for all sizes of messages if the startup latency were reduced. It is therefore worth noting that the recently announced nCUBE-3 is claimed to exhibit a startup latency of only 5 μsec.

3.5.3 Simulations of Larger Systems

In order to compare the algorithm for larger hypercubes, we relied on simulation. McKinley and Trefftz [61] have developed a CSIM-based simulation tool, called *MultiSim*, which can be used to simulate large-scale multiprocessors. In particular, MultiSim uses novel methods to efficiently simulate wormhole-routed systems. In addition, the simulator has been validated against an nCUBE-2 hypercube multicomputer [61].

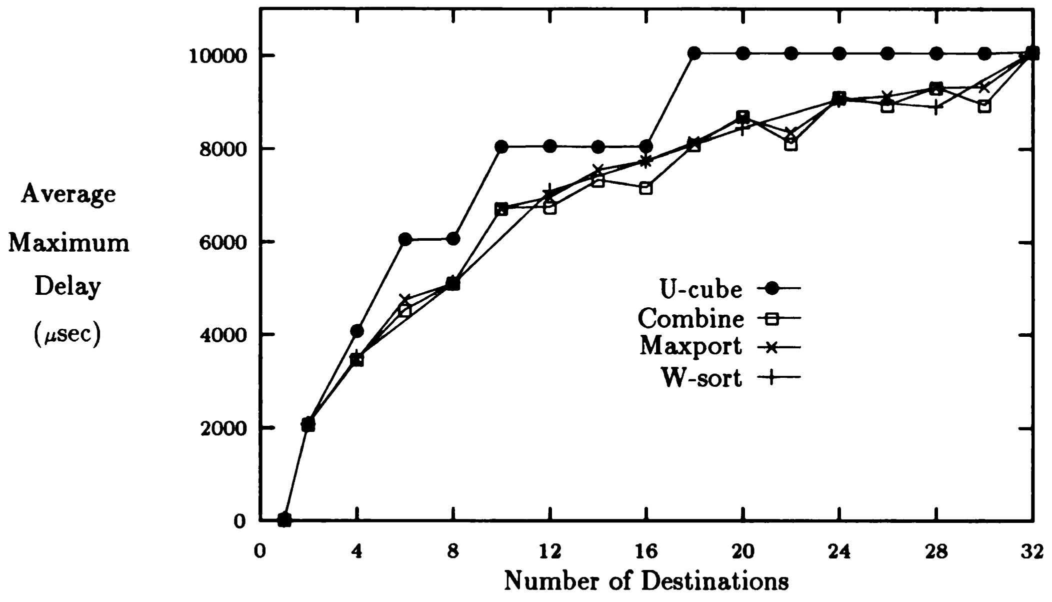


Figure 3.13. Maximum delay comparisons on a 5-cube

Figures 3.14 and 3.15 plot the average and maximum, respectively, among destinations, of the delay between the sending of a 4096-byte multicast message and its receipt at the destination. For each point in a curve, 100 destination sets were chosen randomly in a 10-cube. These plots show that all the algorithms designed to take advantage of the all-port architecture offer advantages over the U-cube algorithm. For the larger systems, the advantage of W-sort becomes more obvious in both the average and maximum cases.

3.6 Conclusions

Efficient data distribution is critical to the performance of new generation supercomputers that use massively parallel architectures. In this chapter, the problem of multicast in all-port wormhole-routed hypercubes has been addressed. It has been

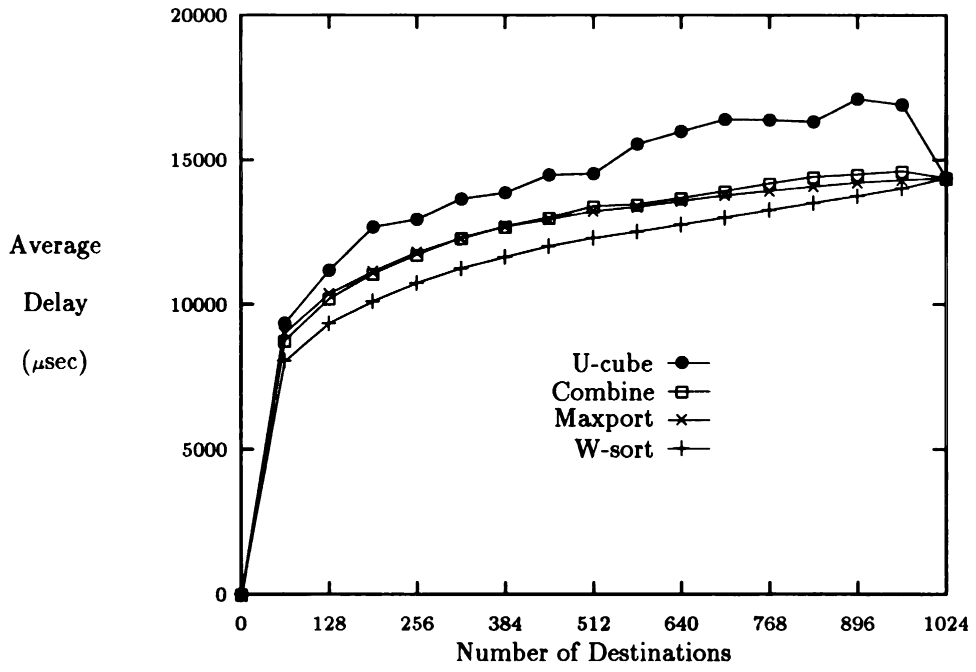


Figure 3.14. Average delay comparisons on a 10-cube

demonstrated why the U-cube multicast algorithm [19], which is optimal for one-port architectures, fails to take advantage of multiple ports when they are present in the system. New theoretical results regarding contention among messages in wormhole-routed hypercubes have been developed and used to design new multicast routing algorithms and to prove that these algorithms are contention-free. The algorithms were compared in terms of the number of steps required in each, their measured execution times when implemented on a relatively small-scale nCUBE-2, and their simulated execution times on larger hypercubes. The results indicate that significant performance improvement is possible when the multicast algorithm actively identifies and uses multiple ports in parallel.

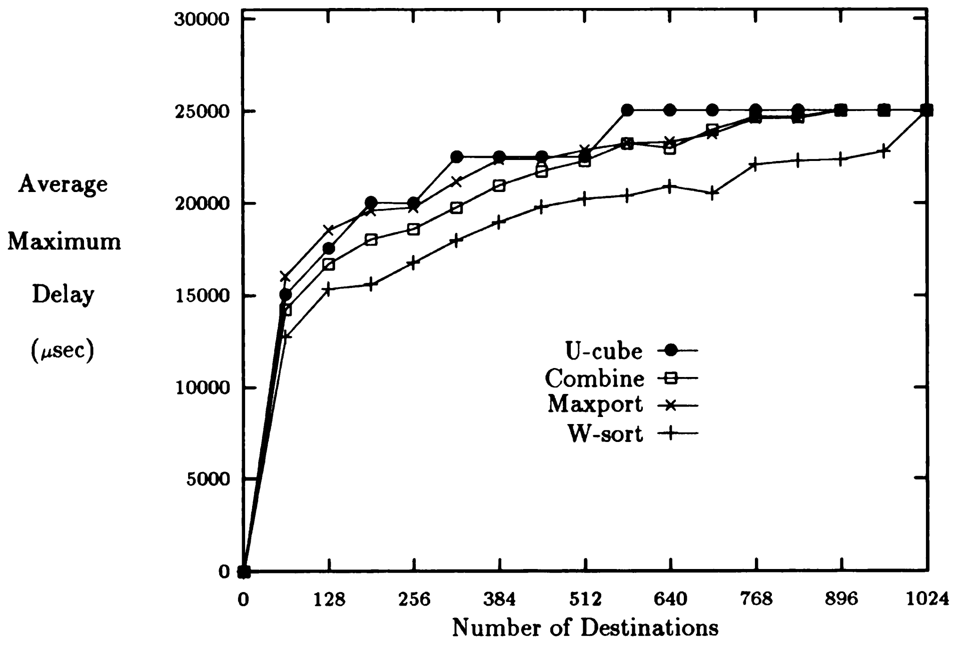


Figure 3.15. Maximum delay comparisons on a 10-cube

CHAPTER 4

Unicast-Based Multicast in One-Port Torus Networks

In this chapter, we develop a unicast-based multicast algorithm for one-port, wormhole-routed n-dimensional torus networks. This algorithm achieves the lower bound of $\lceil \log_2 m \rceil$ message-passing steps and avoids contention among the constituent unicast messages. Optimal multicast algorithms have previously been developed for meshes and hypercubes [19]; we generalize the earlier research to accommodate torus networks. The salient difference between the two topologies is the presence of wrap-around channels in tori, which affects the design of techniques for routing and switching of messages through the network in order to avoid deadlock. These torus-specific properties must be considered in the design of tree-based multicast algorithms in order to minimize the number of message-passing steps while avoiding contention.

The remainder of this chapter is organized as follows. Section 4.1 presents the system models under which we study multicast communication; we consider torus networks with both unidirectional and bidirectional communication links. In Section 4.2, we discuss the unicast routing algorithms for each architecture, upon which the multicast algorithm will be based. Section 4.3 develops theoretical results regarding channel contention in wormhole-routed torus networks, while in Section 4.4, we present an optimal unicast-based multicast algorithm for torus networks. In Section 4.5, we study the performance of the proposed multicast algorithm. Finally, a summary is presented in Section 4.6.

4.1 System Model

Formally, an n-dimensional torus has $k_0 \times k_1 \times \cdots \times k_{n-2} \times k_{n-1}$ nodes, with k_i nodes along each dimension i, where $k_i \geq 2$ for $0 \leq i \leq n-1$. Each node x is identified by n coordinates, $\sigma_{n-1}(x)\sigma_{n-2}(x)\ldots\sigma_0(x)$, where $0 \leq \sigma_i(x) \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes x and y are neighbors if and only if $\sigma_i(x) = \sigma_i(y)$ for all i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(x) \pm 1 = \sigma_j(y) \bmod k_j$. In this chapter, we will assume for purposes of discussion that a torus is *regular*, that is, that $k_i = k_j$ for all $0 \leq i, j \leq n-1$, and we refer to the *width*, or *arity*, of the torus as simply k; however, all of the results we present are also applicable to non-regular tori. We consider the problem of multicast on two classes of torus networks: those with *unidirectional* links, and those with *bidirectional* links. We refer to these network types as *unidirectional tori* and *bidirectional tori*, respectively.

In a unidirectional torus, neighboring nodes are connected by physical channels in one direction only. That is, if there is a channel from node r to node s, denoted (r, s), then there is not a channel (s, r). Specifically, a physical channel (r, s) is present if and only if there is a dimension d, $0 \leq d \leq n-1$, such that $\sigma_d(r) + 1 = \sigma_d(s) \bmod k$, and $\sigma_i(r) = \sigma_i(s)$ whenever $i \neq d$. Figure 4.1(a) shows the physical links associated with a 2D unidirectional torus. Also shown are the paths taken by two example unicast messages, one from source node $(0,0)$ to destination node $(2,1)$, and another from source node $(0,2)$ to destination node $(3,1)$. The paths shown result from a deterministic routing algorithm termed *dimension-ordered* routing [42]. In this approach, messages are routed first in the highest (lowest) dimension in which the source and destination nodes differ. Routing then proceeds on each required dimension, in descending (ascending) order of dimension, until the routing path reaches the destination. Routing in a particular dimension is always completed before routing in the next dimension begins. Although adaptive routing mechanisms for wormhole-routed networks have been the focus of recent research [51, 53], in this chapter we focus on the deterministic dimension-ordered routing strategy, which is widely used due to its simplicity [1].

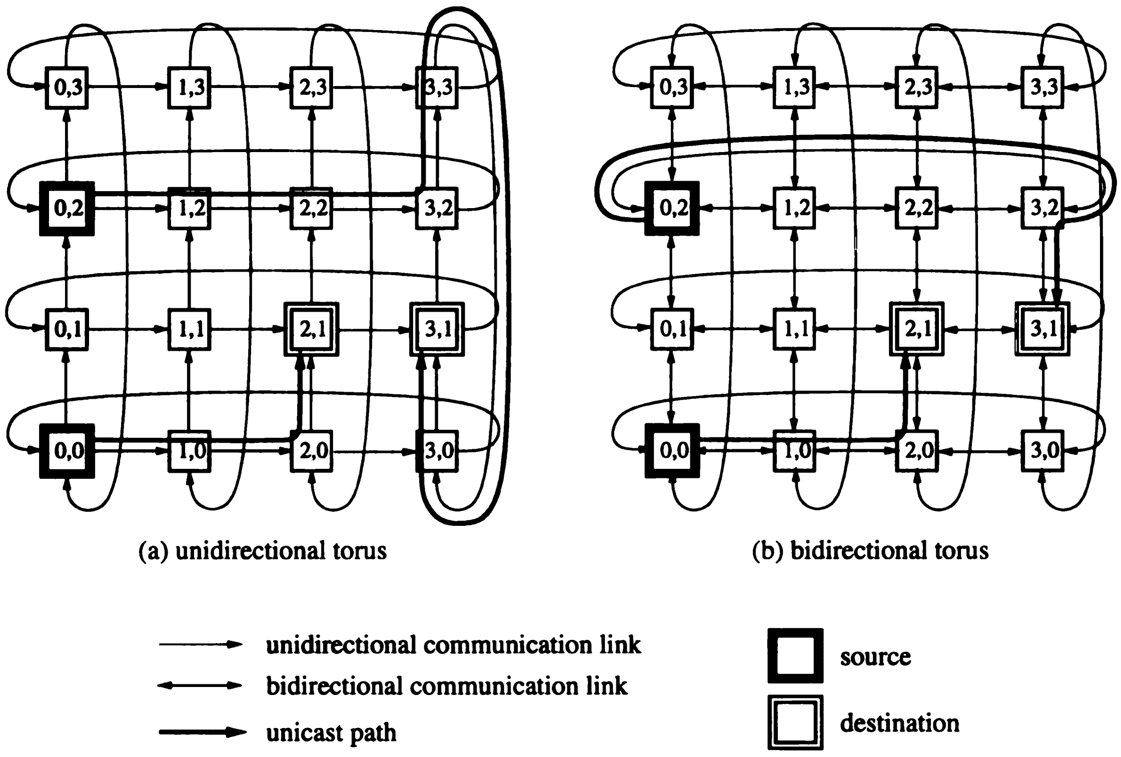


Figure 4.1. Examples of 2D torus networks

In a torus network with only unidirectional links, messages are often forced to travel a longer path than would otherwise be possible. Although message transmission time may be nearly distance-insensitive in a wormhole-routed network, it is still desirable to reduce path lengths whenever possible, since messages that travel on shorter paths use fewer channels, thereby reducing overall channel load, and hence decreasing the frequency of channel contention.

To this end, we also consider the problem of multicast in bidirectional tori, in which direct message transmission is possible in either direction between neighboring nodes. Formally, a physical link (r, s) is present in a bidirectional torus if and only if there is a dimension d, $0 \leq d \leq n-1$, such that $\sigma_d(r) + 1 = \sigma_d(s) \bmod k$ or $\sigma_d(r) - 1 = \sigma_d(s) \bmod k$, and $\sigma_i(r) = \sigma_i(s)$ whenever $i \neq d$. The physical links associated with a 2D bidirectional torus are shown in Figure 4.1(b). As in the case of the unidirectional torus in Figure 4.1(a), paths for two unicast messages are shown. In the case of the path from source node $(0, 2)$ to destination node $(3, 1)$, bidirectional links provide a shorter path than do unidirectional links (2 links versus 6). When a message is routed through a bidirectional torus, there are two possible directions of travel for each dimension. We consider systems in which the direction associated with the *shorter* path is always taken. For networks with even arity, ties are possible (that is, a message may need to travel exactly $k/2$ hops in a particular dimension). In the case of ties, we assume that the path *not* using a wraparound channel is selected. The neighboring nodes in a bidirectional torus may be connected by either single, bidirectional physical channels, or by pairs of unidirectional physical channels facing in opposing directions.

4.2 Unicast Routing Algorithms

Dally and Seitz [57] observed that torus networks with single (unidirectional or bidirectional) channels between neighboring nodes exhibit cycles of channel dependency between unicast messages that can cause deadlock, and that these channel-dependence

cy
ca
to
in
re
te

h
n
l
T
w

4
E
c
c
c
c
f
t
e
v
a
c
i

cycles can be broken by multiplexing ***virtual channels*** on a single physical communication channel. Each virtual channel has its own flit buffer and control [58]. In order to prevent message deadlock in a torus network, single channels between neighboring nodes are replaced with multiple virtual channels, thus allowing the underlying routing algorithm to choose among these multiple virtual channels in such a way as to eliminate cycles of channel dependency.

Virtual channels may be used in a variety of ways to eliminate deadlock, but how they are used has a significant effect on the design of efficient unicast-based multicast operations. We now describe two unicast routing algorithms that we will later consider in the context of their support of unicast-based multicast operations. The first routing algorithm is suitable for torus architectures with unidirectional links, while the second algorithm is applicable to systems with bidirectional links.

4.2.1 Unidirectional Torus Routing

For *unidirectional torus routing* (UTR), there are two parallel sets of virtual channels, called *p*-channels and *h*-channels. The fundamental idea behind UTR is that, for each dimension in which a message travels, *p*-channels ('*p*' for *pre*-wraparound) are used only by messages that will eventually use the wraparound channel in the current dimension; after using the wraparound channel, such messages use the *h*-channels ('*h*' for *high*-direction) for all remaining travel in the current dimension. Those messages that will not use the wraparound channel in a particular dimension use *h*-channels exclusively for travel in that dimension. Figure 4.2 illustrates the virtual channels within a single dimension, d, of a unidirectional torus. We use the notation of Dally and Seitz [57], where $c_{d\alpha x}$ represents the virtual channel leaving node x in dimension d, in the virtual channel set α where $\alpha \in \{\text{'p'},\text{'h'}\}$; that is, α indicates whether $c_{d\alpha x}$ is a *p*-channel or an *h*-channel).

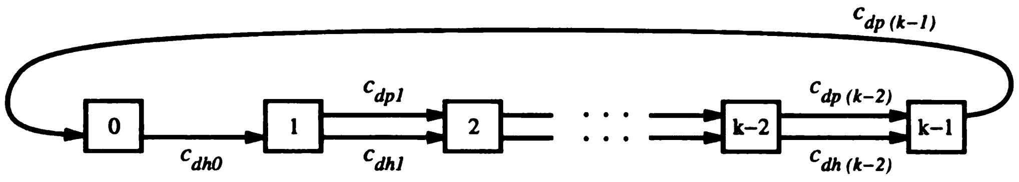


Figure 4.2. Virtual channels in one dimension of a unidirectional torus

Formally, for each dimension d, $0 \leq d \leq k-1$, and for each node x, let y be the node such that $\sigma_d(y) = \sigma_d(x) + 1 \bmod k$ and $\sigma_i(y) = \sigma_i(x)$ for all $i \neq d$. Then under UTR,

1. There is an h-channel, c_{dhx}, from x to y whenever $0 \leq \sigma_d(x) \leq k-2$, and
2. There is a p-channel, c_{dpx}, from x to y whenever $1 \leq \sigma_d(x) \leq k-1$.

As illustrated in Figure 4.2, there is no p-channel when $\sigma_d(x) = 0$, since the p-channels are used only by messages that will eventually use the wraparound channel, and clearly, no message will first visit node 0 and later use the wraparound channel in the same dimension, since to do so would constitute a cycle in the path. Similarly, when $\sigma_d(x) = k-1$, there is no (wraparound) h-channel, since h-channels are used only by messages that have already traversed the wraparound channel (there is no need to use a wraparound channel twice in the same dimension) and by messages that will not use the wraparound channel.

The UTR routing algorithm is described formally by the function $\mathcal{R}_{UTR} : N \times N \rightarrow C$, which maps a *(current node, destination node)* pair into the next channel of the routing path. In order to define $\mathcal{R}_{UTR}(x,y)$, let d be the highest-ordered dimension in which x and y differ, and let $\Delta = \sigma_d(y) - \sigma_d(x)$. Then

$$\mathcal{R}_{UTR}(x,y) = \begin{cases} c_{dpx} \text{ if } \Delta < 0 \\ c_{dhx} \text{ if } \Delta > 0 \end{cases} \tag{4.1}$$

A message is routed from a source node to a destination node by applying the $\mathcal{R}_{UTR}$ function, first at the source, and then at each router through which the message travels, until the destination is reached. Implementing the $\mathcal{R}_{UTR}$ function in a router is straightforward. The UTR routes unicast messages along only shortest paths (under the constraints of unidirectional links) and is deadlock-free [57].

4.2.2 Bidirectional Torus Routing

In a torus network with bidirectional links, cycles of channel dependency exist in both directions in each dimension; as with unidirectional tori, multiple virtual channels are necessary to provide a deadlock-free routing algorithm. For *bidirectional torus routing* (BTR), there are three sets of virtual channels: *p*-channels, *l*-channels, and *h*-channels. Each individual virtual channel carries messages in one direction only. The *p*-channels route messages that will eventually use the wraparound channel in the same dimension. The *l*-channels ('*l*' for *low*-direction) and *h*-channels are used after the wraparound channel has been traversed; the *h*-channels are also used by messages that will not use the wraparound channel in the current dimension. The *l*-channels are directed towards lower-address neighboring nodes, while higher-address neighbors are reached through *h*-channels.

The virtual channels along a single dimension, *d*, of a bidirectional torus with even width, *k*, are illustrated in Figure 4.3. The situation is similar when *k* is odd.

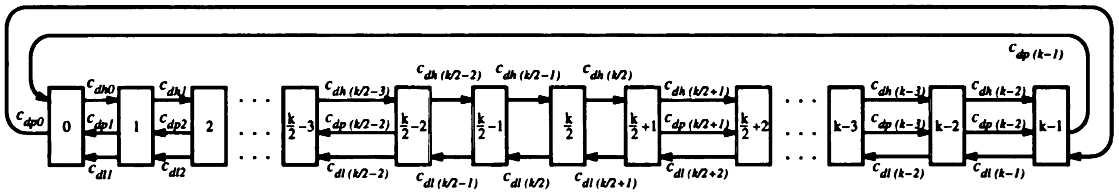


Figure 4.3. Virtual channels in one dimension of a bidirectional torus

Only one set of p-channels is needed, since a message that eventually uses the wraparound channel in a particular dimension will travel only *away* from the "center" of that dimension prior to using the wraparound channel; to do otherwise would result in non-minimum length routing. Formally, for each dimension d, $0 \leq d \leq k-1$, and for each node x, let y be the node such that $\sigma_d(y) = \sigma_d(x)+1 \bmod k$ and $\sigma_i(y) = \sigma_i(x)$ for all $i \neq d$, and let w be the node such that $\sigma_d(w) = \sigma_d(x) - 1 \bmod k$ and $\sigma_i(w) = \sigma_i(x)$ for all $i \neq d$. Then under BTR,

1. There is an l-channel, c_{dlx}, from x to w whenever $1 \leq \sigma_d(x) \leq k-1$,
2. There is an h-channel, c_{dhx}, from x to y whenever $0 \leq \sigma_d(x) \leq k-2$,
3. There is a p-channel, c_{dpx}, from x to y whenever $\lceil \frac{k-1}{2} \rceil + 1 \leq \sigma_d(x) \leq k-1$, and
4. There is a p-channel, c_{dpx}, from x to w whenever $0 \leq \sigma_d(x) \leq \lfloor \frac{k-1}{2} \rfloor - 1$.

As shown in Figure 4.3, some pairs of neighboring nodes require only two interconnecting virtual channels, rather than three. For example, p-channels, which are used by a message prior to wraparound, are not needed near the center of a particular dimension, since a message routed on a shortest path will never pass through the center of a dimension and also use a wraparound channel in that dimension. Wraparound channels are required only in the p-channel set, for reasons similar to those given for UTR.

The BTR routing algorithm is described formally by the function $\mathcal{R}_{BTR} : N \times N \rightarrow C$, which maps a *(current node, destination node)* pair into the next channel of the routing path. In order to define $\mathcal{R}_{BTR}(x,y)$, let d be the highest-ordered dimension in which x and y differ, and let $\Delta = \sigma_d(y) - \sigma_d(x)$. Then

$$\mathcal{R}_{BTR}(x,y) = \begin{cases} c_{dpx} \text{ if } |\Delta| > \frac{k}{2} \\ c_{dlx} \text{ if } -\frac{k}{2} \leq \Delta \leq -1 \\ c_{dhx} \text{ if } 1 \leq \Delta \leq \frac{k}{2} \end{cases} \tag{4.2}$$

Since $\mathcal{R}_{BTR}$ routes messages using a wraparound channel in each dimension, d, in which $|\sigma_d(y) - \sigma_d(x)| > k/2$, then BTR always selects a shortest path between source and destination nodes. BTR is deadlock-free, since there are no cycles of channel dependency between unicast messages routed by $\mathcal{R}_{BTR}$.

4.3 Contention in Wormhole-Routed Torus Networks

The unicast routing algorithm directly affects the design of multicast algorithms in wormhole-routed systems, because it determines how the constituent messages must be scheduled in order to avoid channel contention. Before a multicast operation begins, only the source node has a copy of the message. During the first message-passing step, the source can send the message to only one destination node on a one-port architecture. In each subsequent step, each node holding a copy of the message can send it to at most one new node. Therefore, the number of nodes that have a copy of the message can at most double during each step, leading to a lower bound of $\lceil \log_2 m \rceil$ on the number of steps required to complete a multicast to $m-1$ destination nodes. In order for the actual time required by this *recursive doubling* procedure to be proportional to the number of message-passing steps, contention between unicast messages must be avoided.

The following example illustrates the issues and difficulties involved in implementing efficient multicast communication in torus networks. We consider the 2D (5×5) torus in Figure 4.4, and suppose that a multicast message is to be sent from source node $(4,3)$ to six destinations. We define a multicast operation by a message, M, and a list of nodes, $\Phi = \{x_0, x_1, x_2, \ldots, x_{m-1}\}$, where x_0 is the source node and $\{x_1, x_2, \ldots, x_{m-1}\}$ are the destination nodes, listed in arbitrary order. In this example, $\Phi = \{(4,3), (0,0), (1,1), (2,1), (0,3), (1,3), (4,4)\}$.

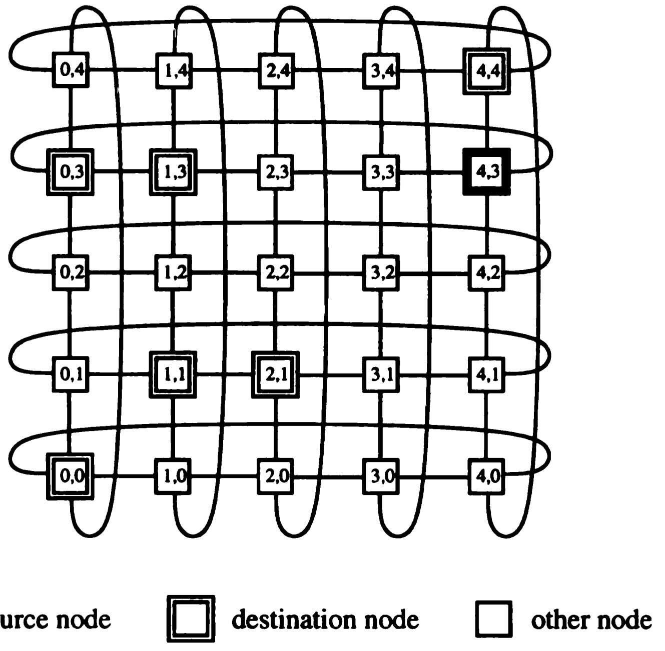


Figure 4.4. An example of multicast in a 2D torus

Figure 4.5 illustrates the multicast trees associated with three different implementations of the example multicast operation in a unidirectional torus (under UTR). The intermediate nodes and virtual channels (*arcs*) of each unicast are shown, as well as the communication step in which the unicast occurs. All arcs of a particular unicast share the same step label; because of the message pipelining associated with wormhole routing, a unicast message is considered to occur in a single step regardless of the number of hops on the associated path. In all three implementations, the local processors at only the source and the destination nodes are required to handle the message.

Although the multicast implementation depicted in Figure 4.5(a) appears to complete the multicast in 3 steps, the unicasts from node $(0,3)$ to node $(1,1)$, and from $(4,3)$ to $(1,3)$, both require the use of the h-channel from node $(0,3)$ to node $(1,3)$ during step 2. Since the unicasts are sent in the same step, there is said to be *stepwise* (a

co

ot

by

ei

wa

tio

me

ted

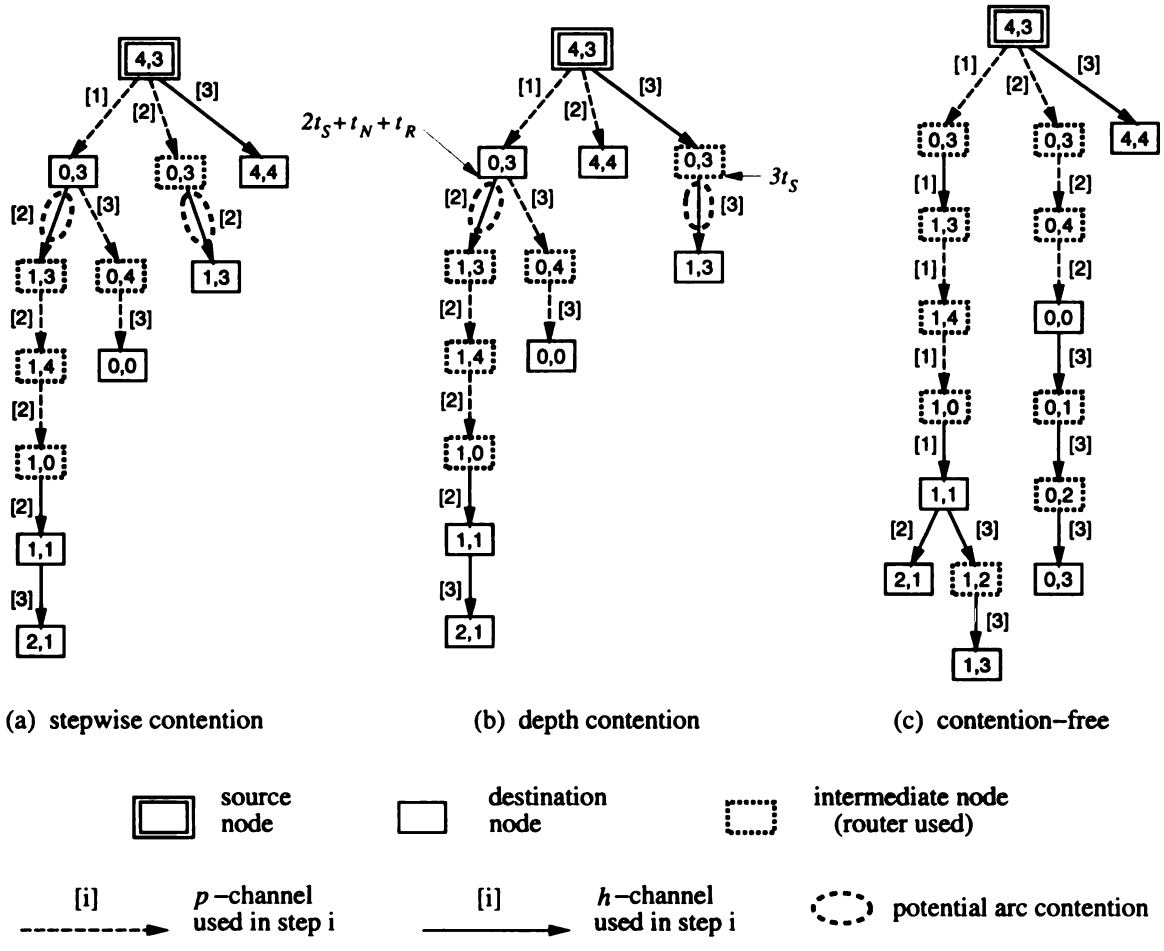


Figure 4.5. Unicast-based software multicast trees

contention between them, which will cause one of the unicasts to be blocked until the other has relinquished the shared arc. This contention will delay one of the unicasts by approximately one step, thereby causing a delay in the receipt of the message (at either nodes $(2,1)$ and $(0,0)$, or at node $(1,3)$, depending on which of the two unicasts was blocked) so that the multicast actually requires four steps to complete.

In Figure 4.5(b), we see an alternate implementation of the same multicast operation, but in this case without stepwise contention. However, under certain conditions, messages occurring in two *different* communication steps may actually be transmitted concurrently. This skewing of message-passing steps is due to the effects of the various communication latencies. As described in Section 2.2.2, the startup latency is the overhead incurred in handling a message at the source and destination nodes. Startup latency is composed of *sending latency*, t_S, and *receiving latency*, t_R. The network latency of a message, t_N, is dependent upon the message length.

In Figure 4.5(b), the message from node $(4,3)$ to node $(0,3)$ during the first step is completely received by node $(0,3)$ at time $t_S + t_N + t_R$. The message from node $(0,3)$ to node $(1,1)$ during the second step thus enters the network at time $2t_S + t_N + t_R$. The three messages originating at the source node $(4,3)$ enter the network at time t_S, $2t_S$, and $3t_S$, respectively. Given the above facts, suppose that the communication latencies are such that $t_S = t_N + t_R$ (for example, perhaps $t_S = 2t_N = 2t_R$). Then the message from node $(0,3)$ to node $(1,1)$ occurring in the second step, and the message from node $(4,3)$ to node $(1,3)$ occurring in the third step, both enter the network at time $3t_S$. Since these two messages each require the h-channel from node $(0,3)$ to node $(1,3)$, then the messages will exhibit channel contention under this scenario of communication latencies. Such channel contention occurring among messages in different communication steps is termed *depth contention*.

Figure 4.5(c) illustrates the use of techniques presented in this chapter, which provide contention-free multicast in a torus using the optimal number of steps. Although the p-channel from node $(4,3)$ to node $(0,3)$ is used by two unicasts (in steps 1 and 2, respectively), there can be no contention for this shared arc, since the first unicast is guaranteed to be complete before the second unicast begins.

Before formally studying contention between messages, we present some notation. The path from a source node u to a destination node v resulting from dimension-ordered routing in a torus network is denoted $P(u,v) = (u; c_0; x_1; c_1; x_2; c_2; \ldots ; x_q; c_q; v)$, where c_i is the virtual channel used to travel from node x_i to node x_{i+1}, for $0 \le i \le q$ (where $x_0 = u$ and $x_{q+1} = v$).

The
amp
rect
((0.
occu
T
able
set c
thro
is th
impl

Defi

if on

1.
2.

V
netw
ticas
const
C
gatec
in ge
corre

Theo

ing f
$(x, y,$

The sequence of nodes visited on the path is $(u; x_1; x_2; \ \ldots \ ; x_q; v)$. For example, the path from source node $(0,0)$ to destination node $(2,1)$ in a unidirectional torus, depicted in Figure 4.1(a), is represented by $P((0,0),(2,1)) = ((0,0); c_{1h(0,0)}; (1,0); c_{1h(1,0)}; (2,0); c_{0h(2,0)}; (2,1))$. A unicast from node u to node v occurring at step t is denoted $(u, v, P(u,v), t)$.

To help understand the structure of a unicast-based multicast operation, the *reachable set* [19] of a node, say node u, in a multicast implementation is defined to be the set of nodes in the multicast that receive the message, either directly or indirectly, through node u. If the multicast is viewed as a tree of unicast messages, then R_u is the set of nodes in the subtree rooted at u. As an example, in the multicast implementation shown in Figure 4.5(a), $R_{(0,3)} = \{(0,3),(1,1),(2,1),(0,0)\}$.

Definition 4.1 *A node v is in the reachable set of node u, denoted R_u, if and only if one of the following holds:*

1. $v = u$; or
2. The implementation contains a unicast $(w, v, P(w,v), t)$ such that $w \in R_u$.

We now present several theorems regarding contention in wormhole-routed torus networks. These theorems are important in verifying that the proposed torus multicast algorithm, presented in Section 4.4, produces only multicast operations whose constituent unicast messages are pairwise contention-free.

Contention in wormhole-routed, n-dimensional *mesh* networks has been investigated in a previous work [19], in which the following theorem regarding contention in general wormhole-routed networks is presented. We use this theorem to develop corresponding results about contention in torus networks.

Theorem 4.1 *Given a multicast implementation, if at least one of the following four conditions holds for every pair of resultant unicasts $(u, v, P(u,v), t)$ and $(x, y, P(x,y), \tau)$, where $t \leq \tau$, then the multicast is depth contention-free.*

1. $x \in R_v$.
2. $P(u,v)$ and $P(x,y)$ are arc-disjoint.
3. $x = u$
4. $x \in R_w$ and $(u, w, P(u,w), t+i)$ is a product of the multicast, for some node w and positive integer i.

In implementing a multicast algorithm, whenever the paths of two constituent unicast messages share an arc (virtual channel), care must be taken to ensure that the paths do not attempt to use the shared arc simultaneously. When two paths have no virtual channel in common, of course, contention between these two particular paths is always avoided. Paths with no common virtual channel are said to be *arc-disjoint*. We now develop two theorems (Theorems 4.2 and 4.3) that identify situations in a torus network, under UTR and BTR, respectively, in which pairs of unicast messages are arc-disjoint. We first give Lemma 4.1, which formalizes the basic notions of dimension-ordered routing and will be useful in later proofs.

Lemma 4.1 *Let $P(u,v) = (u; c_0; x_1; c_1; x_2; c_2; \ \ldots \ ; x_q; c_q; v)$ be any dimension-ordered path (For clarity, let $x_0 = u$ and $x_{q+1} = v$.), and let $c_{d\alpha x_i} \in P$ be any arc in $P(u,v)$. Then the following conditions hold:*

1. For all j, $0 \le j \le i$, and for all f, $0 \le f \le d-1$, $\sigma_f(x_j) = \sigma_f(u)$.
   (Before traveling in dimension d, a message does not travel in dimensions lower than d.)
2. For all j, $i+1 \le j \le q$, and for all f, $d+1 \le f \le n-1$, $\sigma_f(x_j) = \sigma_f(v)$.
   (Upon traveling in dimension d, a message does not travel in dimensions higher than d.)
3. $\sigma_d(u) \neq \sigma_d(v)$.
   (A message travels in dimension d only if the source and destination node addresses differ in dimension d.)

Proof: The lemma follows directly from the behavior of dimension-ordered routing.

□

Definition 4.2 formally describes a *dimension order*, denoted $<_d$, which is a lexicographical ordering (and thus, a total ordering) on the node addresses of a torus network. For example, given nodes $(2,1,5)$, $(2,1,8)$ and $(1,3,7)$ in a 3D torus, we have $(1,3,7) <_d (2,1,5) <_d (2,1,8)$.

Definition 4.2 [19] *The binary relation dimension order, $<_d$, is defined between two nodes x and y as follows: $x <_d y$ if and only if either $x = y$ or there exists an integer j such that $\sigma_j(x) < \sigma_j(y)$ and $\sigma_i(x) = \sigma_i(y)$ for $j+1 \leq i \leq n-1$.*

Lemma 4.2 *For any four nodes u, v, x, y in a torus network, if $v <_d x <_d y$ and paths $P(u,v)$ and $P(x,y)$ share an arc in dimension d under dimension-ordered routing, then $\sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$.*

Proof: Let (r,s) be an arc in dimension d, shared by paths $P(u,v)$ and $P(x,y)$. Since r lies along dimension-ordered paths to both v and y, then by Lemma 4.1, $\sigma_i(r) = \sigma_i(v) = \sigma_i(y)$ for all $i > d$. Since $v <_d x <_d y$, it follows that $\sigma_i(v) = \sigma_i(x) = \sigma_i(y)$, and $\sigma_d(v) \leq \sigma_d(x) \leq \sigma_d(y)$. Finally, because the path $P(x,y)$ travels in dimension d, $\sigma_d(x) \neq \sigma_d(y)$. □

We next present two theorems that give sufficient conditions under which message paths will be arc-disjoint in a torus network using the dimension-ordered routing algorithms described in Section 4.2. Theorem 4.2 applies to unidirectional networks, while Theorem 4.3 is applicable to bidirectional networks.

Theorem 4.2 *For any four nodes u, v, x, y, if $v <_d x <_d y$ then under UTR, paths $P(u,v)$ and $P(x,y)$ are arc-disjoint.*

Proof: The proof is by contradiction. Assume that there is an arc (r,s) shared by paths $P(u,v)$ and $P(x,y)$, and let d be the dimension in which (r,s) travels. Then by Lemma 4.2, $\sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$.

be

th

wi

(ca

by

Al

ur

pa

of

F

T

rathe

From the definition of $\mathcal{R}_{UTR}$ (Expression 4.1), path $P(x, y)$ uses only h-channels between $\sigma_d(x)$ and $\sigma_d(y)$ when traveling in dimension d, since $\sigma_d(x) < \sigma_d(y)$ (recall that h-channels are used after using the wraparound channel, and by messages that will not use the wraparound channel in a particular dimension).

There are two cases to consider with regard to $\sigma_d(u)$: (case 1) $\sigma_d(u) < \sigma_d(v)$; and (case 2) $\sigma_d(u) > \sigma_d(v)$. Figure 4.6 shows for each of these two cases the routes taken by paths $P(u, v)$ and $P(x, y)$ in dimension d, as prescribed by UTR (Expression 4.1). Although, in case 2, $\sigma_d(u)$ may, in fact, be farther to the right than shown in the figure, this difference would only cause *fewer* channels to be used by $P(u, v)$. As shown, paths $P(u, v)$ and $P(x, y)$ cannot share an arc in dimension d; thus, the assumption of the existence of such a shared arc must be false. □

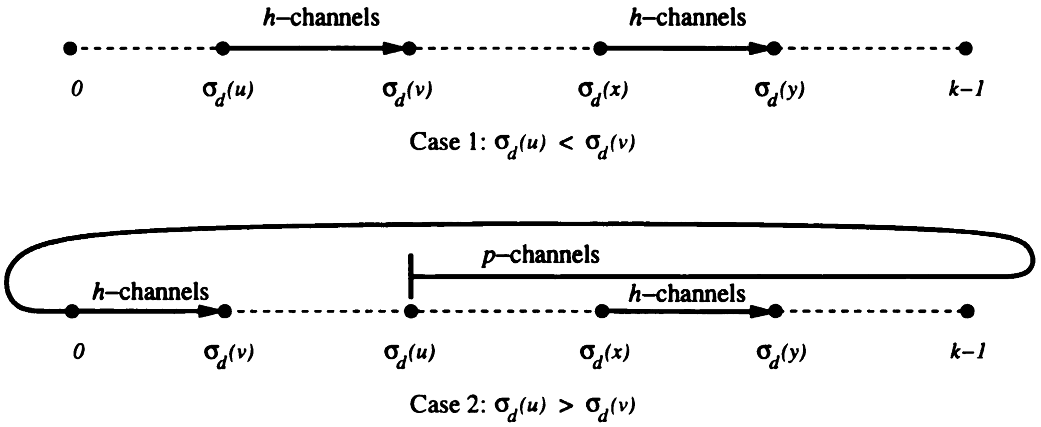


Figure 4.6. Channels used by $P(u, v)$ and $P(x, y)$ in dimension d (Theorem 4.2)

The next theorem is equivalent to Theorem 4.2, but is applicable to bidirectional, rather than unidirectional, torus networks.

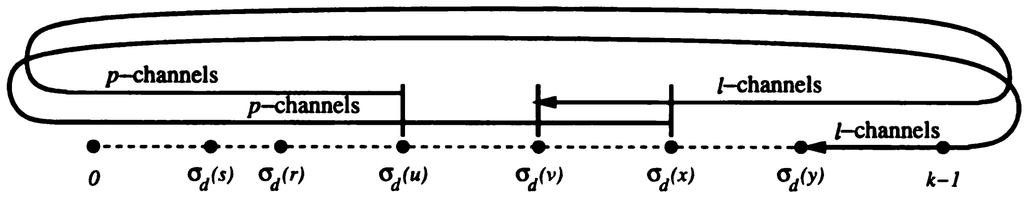
Theorem 4.3 *For any four nodes u, v, x, y, if $v <_d x <_d y$ then under BTR, paths $P(u,v)$ and $P(x,y)$ are arc-disjoint.*

Proof: The proof is by contradiction. Assume that there is an arc (r,s) shared by paths $P(u,v)$ and $P(x,y)$, and let d be the dimension in which (r,s) travels. Then by Lemma 4.2, $\sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$.

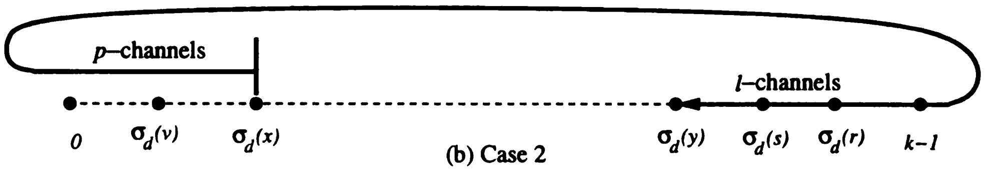
We must consider three possible cases with respect to the shared arc: either (r,s) is (1) a p-channel; (2) an l-channel; or (3) an h-channel.

Case 1. Arc (r,s) is a p-channel: Since the same (*pre*-wraparound) p-channel, (r,s), is used in the path from $\sigma_d(u)$ to $\sigma_d(v)$, and in the path from $\sigma_d(x)$ to $\sigma_d(y)$, then the relationship between $\sigma_d(u)$ and $\sigma_d(v)$ must be the same as the relationship between $\sigma_d(x)$ and $\sigma_d(y)$; that is, either $\sigma_d(u) < \sigma_d(v)$ and $\sigma_d(x) < \sigma_d(y)$, or $\sigma_d(u) > \sigma_d(v)$ and $\sigma_d(x) > \sigma_d(y)$. Thus, since $\sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$ is known, then $\sigma_d(u) < \sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$, as shown in Figure 4.7(a). But since paths $P(u,v)$ and $P(x,y)$ both use a p-channel in dimension d, then $\sigma_d(v) - \sigma_d(u) > k/2$ and $\sigma_d(y) - \sigma_d(x) > k/2$, which cannot be possible when $\sigma_d(u) < \sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$.

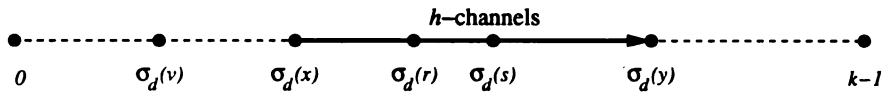
Case 2. Arc (r,s) is an l-channel: Since arc (r,s) is an l-channel (*low*-direction) used in the path $P(x,y)$, and since $\sigma_d(x) < \sigma_d(y)$, then $P(x,y)$ must first use a wraparound channel in dimension d before using arc (r,s). It then follows that $\sigma_d(x) < \sigma_d(y) \leq \sigma_d(s) < \sigma_d(r)$, and $\sigma_d(y) - \sigma_d(x) > k/2$, as shown in Figure 4.7(b). If $\sigma_d(u) > \sigma_d(v)$, then path $P(u,v)$ does not use a wraparound channel in dimension d, so $\sigma_d(v) \leq \sigma_d(s) < \sigma_d(r) \leq \sigma_d(u)$, but since $\sigma_d(y) - \sigma_d(x) > k/2$, then $\sigma_d(u) < \sigma_d(y)$ (otherwise, $\sigma_d(u) - \sigma_d(v) > k/2$, in which case $P(u,v)$ would use a wraparound channel in dimension d), from which follows that $\sigma_d(r) < \sigma_d(y)$, a contradiction. On the other hand, if $\sigma_d(u) < \sigma_d(v)$, we have $\sigma_d(u) < \sigma_d(v) \leq \sigma_d(x) < \sigma_d(y)$, and since $\sigma_d(y) - \sigma_d(x) > k/2$, then $\sigma_d(v) - \sigma_d(u) < k/2$, so path $P(u,v)$ uses only h-channels in dimension d.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 4.7. Channels used by $P(u, v)$ and $P(x, y)$ in dimension d (Theorem 4.3)

Case 3. Arc (r, s) is an h-channel: Since $\sigma_d(x) < \sigma_d(y)$, we know that $\sigma_d(x) \leq \sigma_d(r) < \sigma_d(s) \leq \sigma_d(y)$, as shown in Figure 4.7(c). If $\sigma_d(u) > \sigma_d(v)$, then $\sigma_d(r) < \sigma_d(s) \leq \sigma_d(v)$, hence $\sigma_d(r) < \sigma_d(x)$, a contradiction. If $\sigma_d(u) < \sigma_d(v)$, then it follows that $\sigma_d(u) \leq \sigma_d(r) < \sigma_d(s) \leq \sigma_d(v)$; and again, $\sigma_d(r) < \sigma_d(x)$, a contradiction.

Since each of the above cases leads to a contradiction, the assumption that paths $P(u, v)$ and $P(x, y)$ share an arc must be false. □

4.4 Optimal Multicast Algorithm

In this section, we use the theorems of Section 4.3 to develop a unicast-based multicast algorithm for wormhole-routed tori that use either UTR or BTR routing. We show that this algorithm produces pairwise contention-free unicast messages and completes the multicast in the minimum possible number of steps.

The algorithm uses the recursive doubling procedure described earlier. This method can be viewed in many ways; one possible view is depicted in Figure 4.8, where we assume for simplicity that m, the number of nodes involved in the multicast, is a power of 2. Figure 4.8 shows all unicasts occurring in the first three steps of an optimal multicast (labeled "[1]," "[2]," and "[3]," respectively), as well as two unicasts occurring in the final step (labeled "[$\log_2$ m]"). As shown, the source node, d_0, sends the message to destination node $d_{m/2}$ during the first step; this step partitions the multicast problem of size m into two problems, each of size $m/2$, with source nodes d_0 and $d_{m/2}$, respectively. This process continues recursively until all destination nodes have received the message.

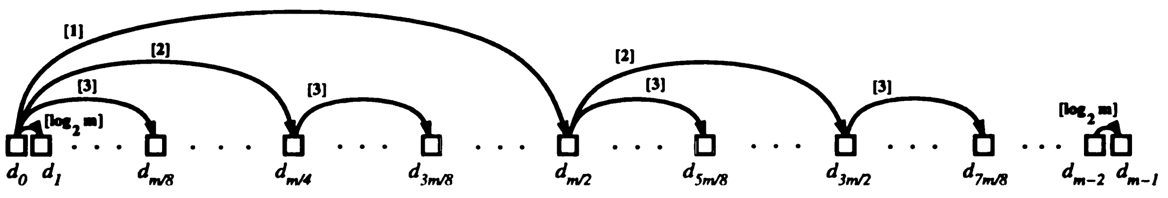


Figure 4.8. A minimum-time multicast

The key to avoiding contention among the constituent messages is the ordering of the destinations. For example, the recursive doubling technique illustrated in Figure 4.8 has previously been applied to meshes; the resultant algorithm is called *U-mesh* [19]. In order to prevent contention, the U-mesh algorithm orders the destination nodes according to the dimension order relation, $<_d$, which was discussed in the previous section. Such an ordered list is called a *dimension-ordered chain* [19], and is defined as follows.

Definition 4.3 *A sequence of nodes $\{x_0, x_1, x_2, \ldots, x_{m-1}\}$ is a dimension-ordered chain if and only if all the elements are distinct and $x_i <_d x_{i+1}$ for $0 \le i < m-1$.*

It turns out that using dimension-ordered chains does not prevent contention in torus networks; that is to say, the U-mesh algorithm is not contention-free when executed on a torus. However, we now show how an extension of the dimension-ordered chain can be used to avoid contention in a torus (and, incidentally, also on a mesh). The resultant *U-torus* algorithm produces contention-free, minimum-time multicast operations on a torus, under either UTR or BTR.

Definition 4.4 *If $\Phi = \{x_0, x_1, x_2, \ldots, x_{m-1}\}$ is a dimension-ordered chain and x_s is an element of Φ, then $\{x_s, x_{s+1}, \ldots, x_{m-1}, x_0, x_1, \ldots, x_{s-1}\}$ is an R-chain with respect to x_s.*

An R-chain is an end-around *rotation* of a dimension-ordered chain. Any dimension-ordered chain $\Phi = \{x_0,\ x_1,\ x_2,\ \ldots,\ x_{m-1}\}$ is an R-chain with respect to x_0; that is, any dimension-ordered chain is also an R-chain with respect to the first element. As an example of the construction of an R-chain, we consider the following multicast from source node $(8,4,5)$ in a 3D torus, where

$$\begin{aligned}\Phi \;=\;& \{\underline{(8,4,5)}, (4,9,3), (1,9,7), (1,0,2), (8,5,4),\\ & (4,8,9), (9,0,5), (3,5,5), (9,0,1), (8,0,5), (1,6,4)\}\end{aligned}$$

is the specified multicast operation, with source node $(8,4,5)$ underlined. First, we sort Φ according to a lexicographical ordering of the node addresses, to obtain the dimension-ordered chain

$$\begin{aligned}\Phi' \;=\;& \{(1,0,2), (1,6,4), (1,9,7), (3,5,5), (4,8,9),\\ & (4,9,3), (8,0,5), \underline{(8,4,5)}, (8,5,4), (9,0,1), (9,0,5)\}\end{aligned}$$

Next, we *rotate* the chain Φ' so that the source node, $(8,4,5)$, appears at the head of the list. This action results in the following R-chain,

$$\begin{aligned}\Phi'' &= \{\underline{(8,4,5)},(8,5,4),(9,0,1),(9,0,5),(1,0,2),\\ &\quad (1,6,4),(1,9,7),(3,5,5),(4,8,9),(4,9,3),(8,0,5)\}.\end{aligned}$$

Figure 4.9 gives the *U-torus* algorithm, which implements the recursive doubling process described above in a torus network. The algorithm takes an R-chain as input.

```
Algorithm 1: The U-Torus Algorithm
Input: R-chain $\{d_{left}, d_{left+1}, \ldots, d_{right}\}$,
   where $d_{left}$ is the local address.
Output: Send $\lceil \log_2(right - left + 1) \rceil$ messages
Procedure:
   while $left < right$ do
      $center = left + \lceil \frac{right-left+1}{2} \rceil$;
      $D=\{d_{center}, d_{center+1}, \ldots, d_{right}\}$;
      Send a message to node $d_{center}$ with the address field $D$;
      $right = center - 1$
   endwhile
```

Figure 4.9. The U-torus algorithm for multicast

Figure 4.10 illustrates the application of the U-torus algorithm to the R-chain Φ'' from the previous example. For clarity, intermediate routers in the unicast paths are omitted. As shown, the U-torus algorithm requires four steps in this example to reach all destination nodes. Since there are 10 destination nodes, the lower bound on the number of steps required to perform the multicast is also $\lceil \log_2 11 \rceil = 4$.

Theorem 4.4 *If $\Phi = \{d_0, d_1, \ \ldots \ , d_{m-1}\}$ is an R-chain, then the U-torus algorithm applied to Φ, under either UTR or BTR, performs a minimum-time, contention-free multicast from source d_0 to destinations $\{d_1, \ \ldots, \ d_{m-1}\}$.*

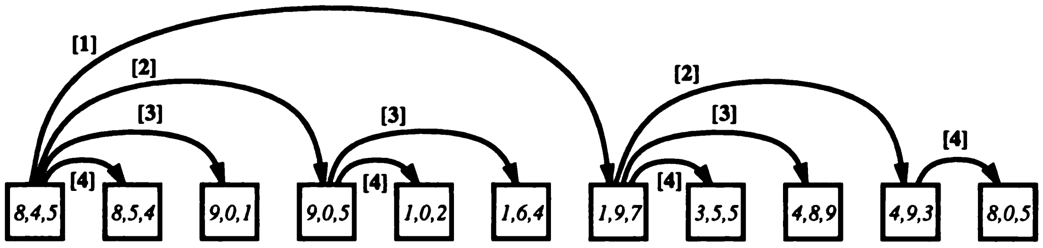


Figure 4.10. Example multicast using the U-torus algorithm

Proof: From the above description of the U-torus algorithm and the accompanying discussion, it is easy to see that the algorithm produces a multicast from the source to the intended destinations, and that barring contention, $\lceil \log_2 m \rceil$ steps are required to complete a multicast to $m-1$ destinations. The more difficult task is to show that the unicast messages produced by the algorithm are always pairwise contention-free.

Let $(u, v, P(u,v), t)$ and $(x, y, P(x,y), \tau)$ be any two unicasts produced by an invocation of the U-torus algorithm, and assume, without loss of generality, that $t \leq \tau$. There are three possible relationships between the two unicasts; these are depicted in Figure 4.11. In case 1, $x = u$, so by item 3 of Theorem 4.1, the unicasts are contention-free. In case 2, item 4 of Theorem 4.1 holds, so again, the unicasts are contention-free. We now show that the unicasts represented by case 3 are arc-disjoint, and hence, contention-free.

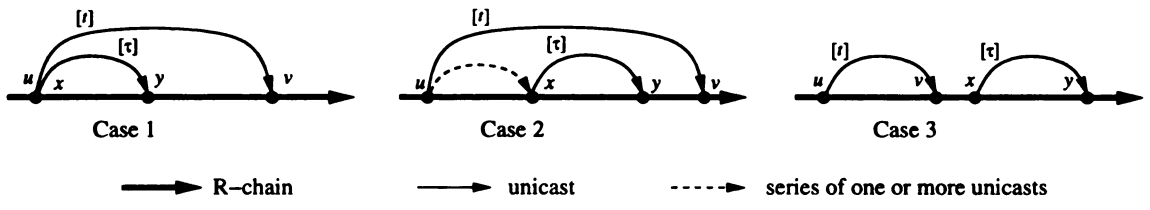


Figure 4.11. Possible relations between unicasts produced by U-torus

We note from Definition 4.4 that an R-chain, $\Phi = \{x_s, x_{s+1}, \ldots, x_{m-1}, x_0, x_1, \ldots, x_{s-1}\}$, consists of two concatenated sub-chains, $\Phi_a = \{x_s, x_{s+1}, \ldots, x_{m-1}\}$, and $\Phi_b = \{x_0, x_1, \ldots, x_{s-1}\}$. Since nodes u, v, x and y appear in the given order (u, v, x, y) in the R-chain, there are five possible points with respect to these four nodes where the partition between Φ_a and Φ_b might occur. These five subcases are enumerated as follows:

i. $u, v, x, y \in \Phi_a$

ii. $u, v, x \in \Phi_a$; $y \in \Phi_b$

iii. $u, v \in \Phi_a$; $x, y \in \Phi_b$

iv. $u \in \Phi_a$; $v, x, y \in \Phi_b$

v. $u, v, x, y \in \Phi_b$

We consider any two nodes u and v in an R-chain, where u occurs before v in the R-chain. From Definitions 4.3 and 4.4, if u and v belong to the same sub-chain (that is, if either $u, v \in \Phi_a$ or $u, v \in \Phi_b$), then $u <_d v$. Also, if the two nodes belong to different sub-chains, (that is, if $u \in \Phi_a$ and $v \in \Phi_b$), then $v <_d u$. Thus we can conclude, for each of the above subcases, respectively, the following:

i. $u <_d v <_d x <_d y$

ii. $y <_d u <_d v <_d x$

iii. $x <_d y <_d u <_d v$

iv. $v <_d x <_d y <_d u$

v. $u <_d v <_d x <_d y$

Since Theorem 4.2 (or Theorem 4.3, depending on whether the network being considered is unidirectional or bidirectional) applies to each of the above five subcases, we now conclude that, in case 3 of Figure 4.11, paths $P(u, v)$ and $P(x, y)$ are arc-disjoint, and therefore contention-free. $\square$

Let us now consider the computational demands placed on the source node processor by the U-torus algorithm. The list of destination nodes (along with the source node) must be arranged as an R-chain. This arrangement can be accomplished by sorting the node addresses, and then rotating the sorted list. The associated computational complexity is thus $O(m \log_2 m)$ for a multicast of size m. The computation performed by intermediate destination nodes of the U-torus algorithm consists of selecting the central node in the destination list; this selection is performed in constant time (see Figure 4.9). In cases where the same set of destination nodes is used for many multicast operations, such as in numerical algorithms, the list of node addresses need only be arranged into an R-chain once. This R-chain can then be used for all subsequent multicasts involving the same source and destination set.

4.5 Performance Evaluation

As shown in Section 4.4, the U-torus algorithm completes in a minimum number of steps, and the unicasts produced by the algorithm are pairwise contention-free (Theorem 4.4). As a practical consideration, however, we note that it is possible for the U-torus algorithm to generate pairs of unicasts that simultaneously use two *different* virtual channels joining the same pair of neighboring nodes. In Figure 4.2, for example, virtual channels c_{dp1} and c_{dh1} might be used simultaneously by two unicasts produced by the U-torus algorithm. If each virtual channel of a torus network corresponds to a distinct physical communication link, then the above situation has no effect on the performance of the algorithm, that is, unicast messages traveling between a particular pair of adjacent nodes will travel on separate physical links, and thus be unaffected by one another. However, we must also consider torus networks in which pairs of parallel virtual channels are multiplexed onto a single physical communication link [1].

When two virtual channels are multiplexed onto a single physical link, they share the bandwidth of that link. If one of the virtual channels is idle, then a message traversing the other virtual channel will use all the bandwidth of the physical link. When two unicasts simultaneously use virtual channels that are multiplexed onto a physical link, however, the speed at which these messages are delivered to their destination is reduced by half. This situation is quite different from arc contention, in which two (or more) messages require the same *virtual channel* simultaneously, and in which one message is blocked until the other has completely passed through the mutually-required channel.

In order to study the effect of virtual channel multiplexing on the performance of the U-torus algorithm, we examined its behavior when executed on destination sets in which the nodes are randomly distributed throughout a network. For this study, we assume that whenever two unicast messages in the same step use virtual channels that are multiplexed onto the same physical link, these two unicasts each require time equivalent to two message-passing steps, rather than one.

In the case of a unidirectional torus, we assume that both of the virtual channels between a particular pair of neighboring nodes are multiplexed onto a single physical link, while for a bidirectional torus, we consider virtual channels in the same direction to share a single physical link. For example, in Figure 4.3, nodes 0 and 1 are connected by two unidirectional physical links; one link supports virtual channel c_{dh0}, while virtual channels c_{dp1} and c_{dl1} are multiplexed onto the other physical link.

Given the above assumptions, Figure 4.12 plots the average, among destinations, of the number of message-passing steps required to reach the destination nodes in a U-torus multicast operation, while Figure 4.13 shows the maximum number of steps among destinations. Each point in these plots was produced by averaging over a large number of uniformly distributed destination sets. Both unidirectional and bidirectional networks are considered, as well as both 1024-node 2D (32×32) and

512-node 3D (8 $\times$ 8 $\times$ 8) topologies. Multicast set sizes from 8 through 64 nodes were examined. For both the average and the maximum case, the plotted lower bound is calculated as the number of steps required when physical link sharing is not considered.

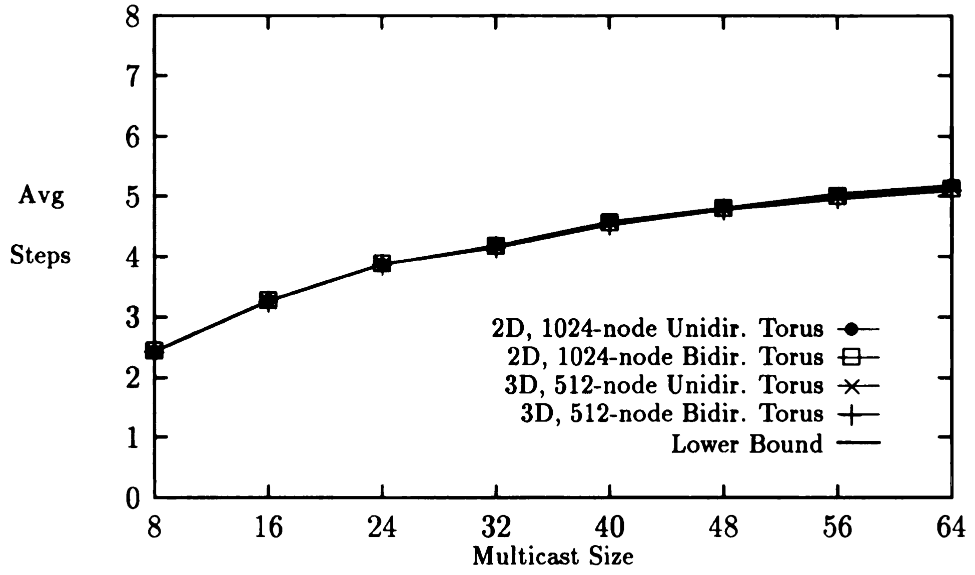


Figure 4.12. Average communication steps (512 and 1024-node tori)

We also examined the effects of physical link sharing on larger torus networks. Figures 4.14 and 4.15 plot the average and maximum number of steps, among destinations, for 4096-node 2D (64 $\times$ 64) and 3D (16 $\times$ 16 $\times$ 16) torus networks, with both unidirectional and bidirectional links. For these larger configurations, multicast set sizes from 64 through 512 nodes were considered.

As illustrated in Figures 4.12 and 4.14, the effect of virtual channel multiplexing on the average number of steps is small. In all cases, the number of steps is close to the theoretical lower bound for systems in which virtual channels do not share physical link bandwidth. In Figures 4.13 and 4.15, we see that the effect on the

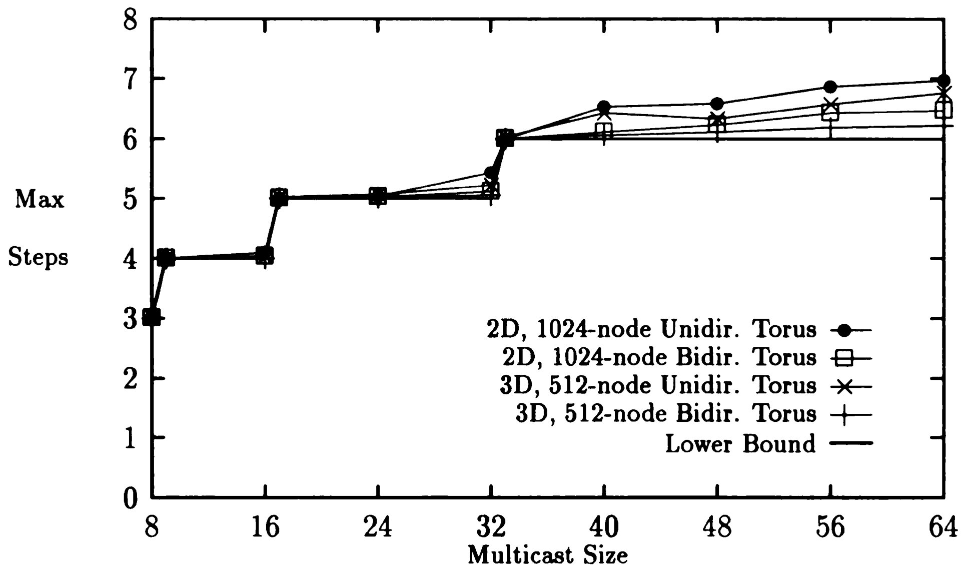


Figure 4.13. Maximum communication steps (512 and 1024-node tori)

maximum number of steps is greater than for the average case, but even this effect is limited. It is noted that, for a given cost, systems using fewer physical links will have individual links with greater capacity than systems in which each virtual channel is supported by a separate physical link. Thus, the modest effects of physical link sharing are likely to be more than offset by the greater capacity of each link resulting from virtual channel multiplexing.

To put the effects of physical link sharing into perspective, Figure 4.16 compares the worst-case observed performance of U-torus with the number of steps required when the source node performs the multicast operation. The worst observed case occurred with a 2D, 4096-node, unidirectional torus, as shown in Figure 4.15. As Figure 4.16 demonstrates, the difference between the performance of the U-torus algorithm, with and without the practical consideration of link sharing, is extremely small compared to the performance increase over a multicast operation performed using separate addressing, in which the source directly sends the message to every destination.

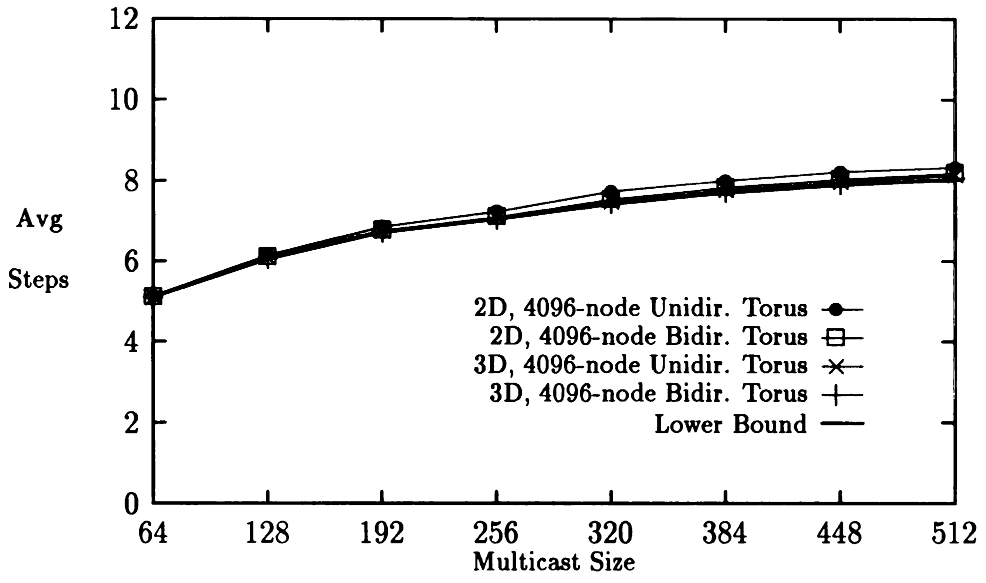


Figure 4.14. Average communication steps (4096-node tori)

In the case of a bidirectional torus, one might also consider designing a network in which all three virtual channels between a pair of neighboring nodes are multiplexed onto a single, bidirectional physical link. For example, in Figure 4.3, virtual channels c_{dh0}, c_{dp1}, and c_{dl1} might all be multiplexed onto a single physical link. This 3-way multiplexing would have two additional consequences beyond the 2-way multiplexing that we have already considered. First, two messages that travel between the same pair of nodes, but in opposite directions, would now share the bandwidth of a single physical link. Second, it would now be possible for three messages to simultaneously share a physical link. We examined the additional effects of the first item (2-way multiplexing between opposite-direction messages) on the U-torus algorithm and found an increase in the average and maximum number of steps of not more than 3 and 7 percent, respectively, over the values presented in the previous plots; in many cases, the effect was much smaller.

In order to better understand the effect of three unicast messages simultaneously sharing a physical link, we studied the *frequency* of such occurrences. Figure 4.17

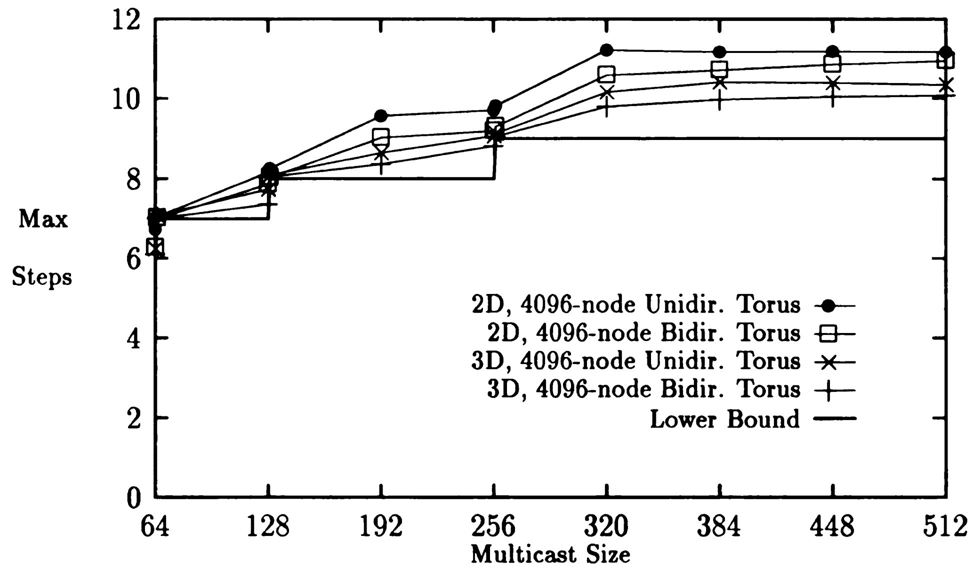


Figure 4.15. Maximum communication steps (4096-node tori)

compares, for 4096-node 2D and 3D bidirectional torus networks, the number of unicasts produced by the U-torus algorithm that are involved in the following two types of physical link sharing. Recall that the total number of unicasts in a multicast operation equals the number of destinations.

Type 1. Unicasts that share a physical link with another unicast in the same step, given the assumption that only virtual channels traveling in the same direction are multiplexed together.

Type 2. Unicasts that share a physical link with *two* other unicast in the same step, given the assumption that *all* virtual channels between a pair of adjacent nodes are multiplexed onto a single physical link.

We chose the above comparison because we know the effects of the first type of link sharing. As shown in Figure 4.17, the frequency of 3-way link sharing is very small compared to 2-way sharing. Since we have demonstrated that the effect of 2-way link sharing is not large, we conclude that the effect of 3-way link sharing is likely to be inconsequential.

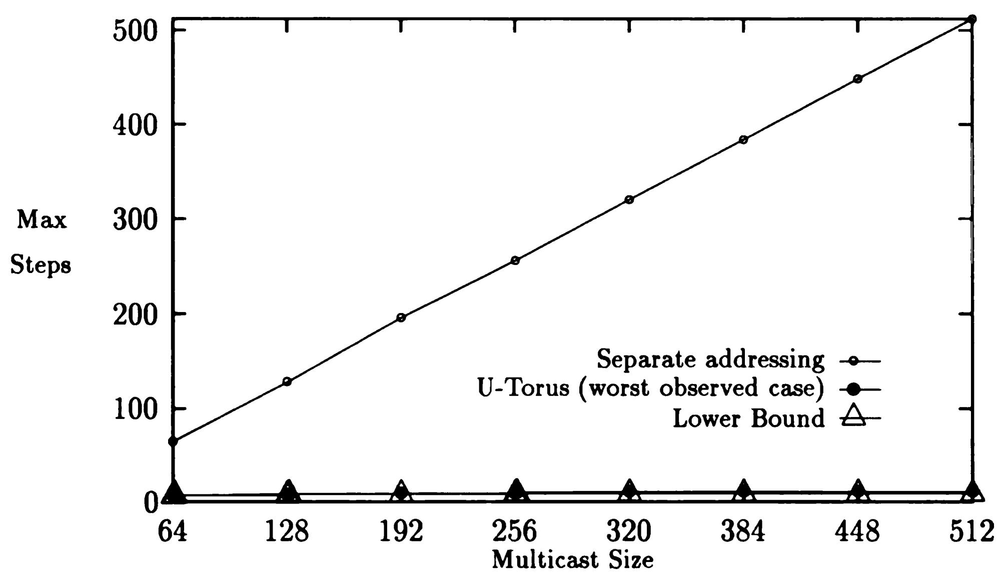


Figure 4.16. Effects of physical link sharing

Thus, the U-torus algorithm performs well in a variety of environments: those with either unidirectional or bidirectional communication links; and those with virtual channels implemented with either independent physical links, or by multiplexing either two or three (in the case of bidirectional tori) virtual channels onto a single physical link.

4.6 Conclusions

This chapter has presented an efficient algorithm for multicast communication on wormhole-routed torus networks. The U-torus algorithm applies to unidirectional and bidirectional tori of any dimension. The algorithm produces multicast trees in which the constituent unicast messages do not contend for the same channels, regardless of message length or startup latency. Moreover, the number of message-passing steps required to multicast data to $m - 1$ destinations is $\lceil \log_2 m \rceil$, which is optimal for one-port architectures. The results of a simulation study showed that the

Ur
Sh
Ph
L

prac

on d

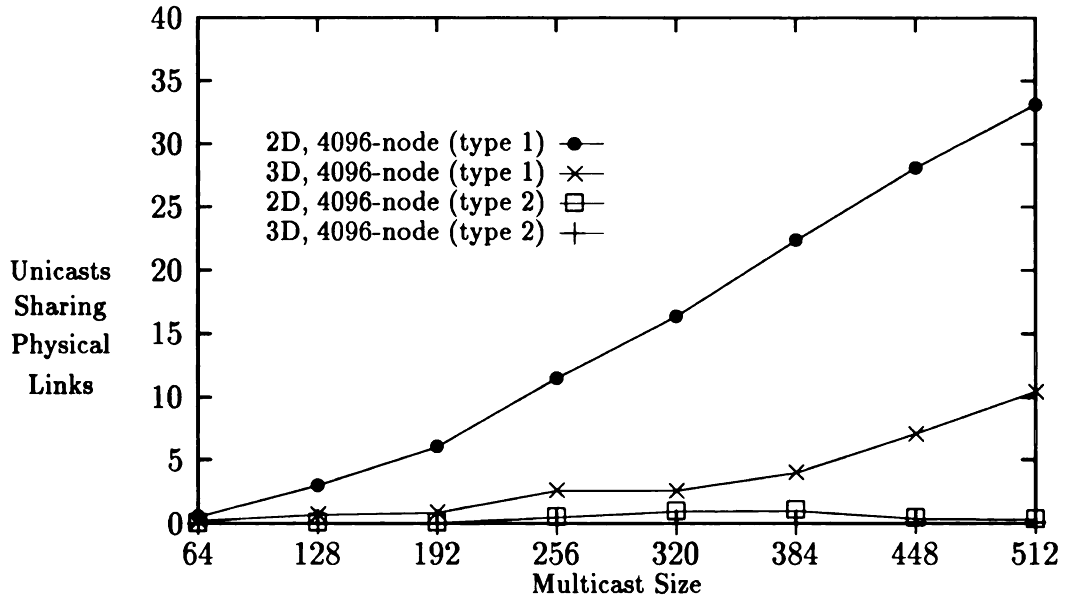


Figure 4.17. Comparison of 2-way and 3-way physical link sharing

practical consideration of physical link sharing by constituent messages transmitted on different virtual channels has little effect on performance.

V
w
d
in
o
de
ro
a
ca

pa
Se
wh
for
sag
des

CHAPTER 5

Path-Based Routing in Unidirectional Torus Networks

We now turn to research in the area of path-based message routing. In this chapter, we develop a general path-based message routing mechanism for wormhole-routed unidirectional torus networks with intermediate reception (IR) capability. As described in Section 2.5, a router with IR capability is able not only to route incoming messages onto outgoing channels without processor intervention, but can also simultaneously deliver a copy of a passing message to the local processor/memory. The path-based routing mechanism presented in this chapter will be used in Chapter 6 as a basis for a family of path-based multicast algorithms for unidirectional torus networks with IR capability.

The organization of this chapter is as follows: The major issues associated with path-based routing in unidirectional torus networks are discussed in Section 5.1. In Section 5.2, Hamiltonian Circuits in torus networks are presented as a means by which deadlock-free path-based routing can be achieved. The path routing function for multi-destination messages is described in Section 5.3, and in Section 5.4, a message preparation algorithm is described. This algorithm is used to order a list of destination nodes in such a way that when the path routing function is used to route from the source node through each destination node in the prescribed order, the resulting multi-destination message is deadlock-free in combination with any collection of such messages in the network. The correctness of the proposed message routing technique is verified in Section 5.5. The implementation issues associated with this path-based routing method are addressed in Section 5.6. Finally, conclusions are given in Section 5.7.

5.1 Issues

Multi-destination messages are subject to the same deadlock considerations as we have described for unicast messages. Since the progress of an entire multi-destination worm depends on the concurrent availability of all channels used by that worm, the routing rules must be applied to the worm as a whole, and not just individually to the segments of the worm that exist between destination nodes.

In an attempt to provide deadlock-free path-based routing, multi-destination worms may be subjected to the same routing rules that are known to be deadlock-free for unicast routing. However, in a communication operation involving an arbitrary destination set, such as multicast, multi-destination worms that are forced to adhere to existing deadlock-free unicast routing rules are often ineffective.

Figure 5.1 illustrates a multicast problem in a 2D (6×6) torus, where the source node $(3,2)$ is to deliver a message to 9 destination nodes. In this example, any single message following XY routing rules can reach at most two destination nodes. For instance, XY routing dictates that the path from source node $(3,2)$ to destination node $(4,3)$ is $((3,2),(4,2),(4,3))$. Upon reaching node $(4,3)$, this path has not passed through any additional destination nodes and has already traveled in the Y direction, so node $(4,5)$ is the only additional destination node that can be reached. In this case, the routing rules prevent a message enroute to any destination node from passing through a large number of the other destinations. The routing rules used for path-based routing must therefore be liberal enough to allow flexibility in message routing, so that many intermediate destination nodes can be visited by a single message. However, these routing rules must still avoid deadlock.

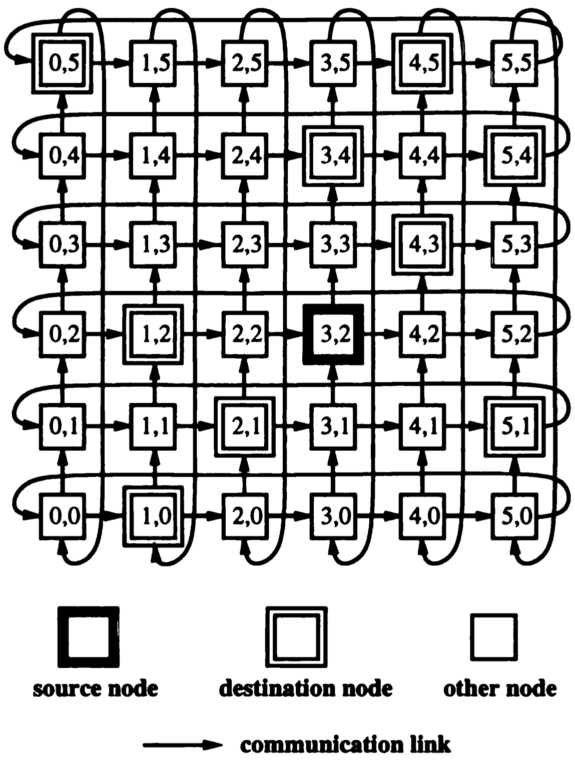


Figure 5.1. A multicast operation in a 2D torus

The way in which a multi-destination message is prepared for transmission can affect the channel dependencies created by the message, and hence, the deadlock properties of the system. For example, the order in which the destinations of a multi-destination worm are visited affects the channel dependencies produced by the message.

In general, the more flexible routing rules needed for path-based routing introduce additional channel dependencies over those that exist for unicast routing. In order to prevent deadlock in systems with deterministic routing rules, the channel dependency graph must be acyclic [57]. In order to avoid deadlock, the routing rules need only account for those messages that can actually be produced by the supported communication operations. For example, if a multi-destination message visiting destination nodes u, v, and w, in that order, will *never* be produced, then such a message need not be considered when analyzing the deadlock properties of a system. Therefore, it is the *combination* of the message preparation algorithm and the path routing function that must be considered when designing a deadlock-free system.

Throughout this chapter and the next, we assume that the system model is a regular unidirectional torus of width k and dimension n. The reader is referred to Section 4.1 for a definition of this system model. In addition, node routers are assumed to have IR capabilities, as described in Section 2.5.

In systems with one-port architectures, the single path from a router to the local node may itself induce communication deadlock when two or more multi-destination messages are issued concurrently. Figure 5.2 illustrates a scenario in which two multi-destination worms are each attempting to deliver a message to nodes u and v. However, each message is holding the single input port at one node, while waiting to use the input port at the other node, resulting in communication deadlock. So that such port-induced deadlock does not occur, we assume a system with an all-port architecture.

For any node u, and any dimension d, let u^d be the node adjacent to u in dimension d. Formally,

$$u^d = \sigma_{n-1}(u)\sigma_{n-2}(u) \cdots \sigma_{d+1}(u)[\sigma_d(u) + 1 \bmod k]\sigma_{d-1}(u) \cdots \sigma_0(u). \qquad (5.1)$$

5

O

th

E

ca

a

on

firs

up

co

ma

no

rou

tot

the

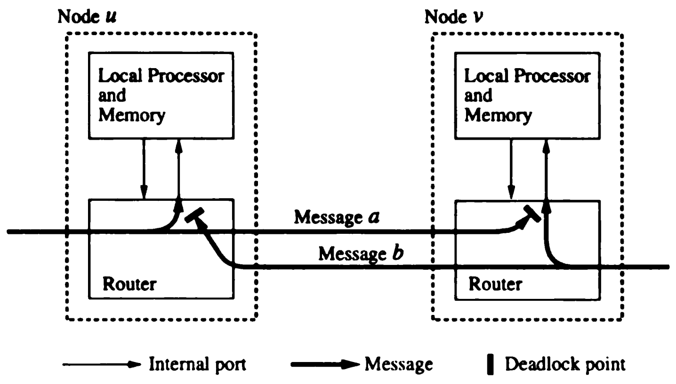


Figure 5.2. Port-induced communication deadlock in a one-port system

5.2 Hamiltonian Circuits

One way in which deadlock can be avoided in path-based routing is by ensuring that there are no cycles of channel dependency allowed by the routing algorithm. Ensuring an absence of channel dependence cycles, and thus freedom from deadlock, can sometimes be accomplished by means of a *Hamiltonian Path* (HP). An HP in a network is a path that traverses communication links, visiting each node exactly once, while a *Hamiltonian Circuit* (HC) is an HP whose last node is adjacent to the first node (thus completing the circuit).

An example of an HP in a 2D mesh is shown in Figure 5.3. The numbers near the upper-left corner of each node define a total ordering of the nodes. This ordering corresponds to an HP through the network. In general, a network may contain many HPs. If the routing algorithm can be designed so that messages always visit nodes (including the destination nodes and the intermediate nodes at which only the router is used) in the same order as those nodes appear on a particular HP, then a total ordering will be induced on the use of the communication channels (resources), thereby ensuring deadlock-free communication.

Figure 5.3 shows two message worms generated by a path-based multicast algorithm designed for mesh networks [30]. These message worms each travel in an order corresponding to the HP and together deliver the message, via IR, to the destination nodes. Since the communication channels that connect nodes in a forward direction with respect to the HP are disjoint from those that connect nodes in a reverse direction, the use of both message worms that travel forward, and those that travel backward, on the HP does not produce communication deadlocks.

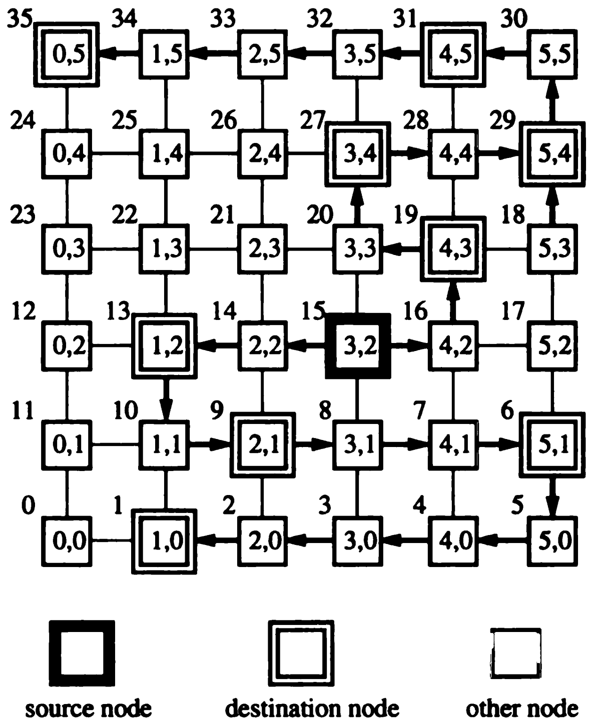


Figure 5.3. Path-based multicast routing in a mesh using Hamiltonian Paths

The path-based routing method for meshes, described above, is based on the existence of an HP with the following characteristic: it is possible to route a message from an arbitrary source node to an arbitrary destination such that the nodes visited on the route (including the source and destination) conform to the ordering of nodes defined by the HP. In a torus network with unidirectional communication links, an HP with the above characteristic clearly does not exist. Therefore, in order to provide deadlock-free path-based routing in this topology, we cannot rely on an HP alone.

Figure 5.4 shows an HC in a 2D (6×6) unidirectional torus. The numbers near the upper-left corner of each node correspond to one particular HC; this HC begins at node $(0,0)$, continues through the network, visiting every node. Although there is generally more than one HC in a unidirectional torus, we use only a particular HC such the one illustrated in Figure 5.4. We refer to this special HC in a particular torus, T, as $\mathcal{H}_T$, or simply $\mathcal{H}$ when it is clear which torus network is indicated. Informally, $\mathcal{H}$ begins at node 0 (node 0 is the node whose address value is 0 in every dimension); at each node u on $\mathcal{H}$, the next node is the neighbor, u^d, of u that minimizes d under the constraint that u^d does not already precede u on $\mathcal{H}$. Also shown in Figure 5.4 are *boundaries*, which are communication links that travel backwards in $\mathcal{H}$. Boundaries will be used when defining a path-based routing function for torus networks.

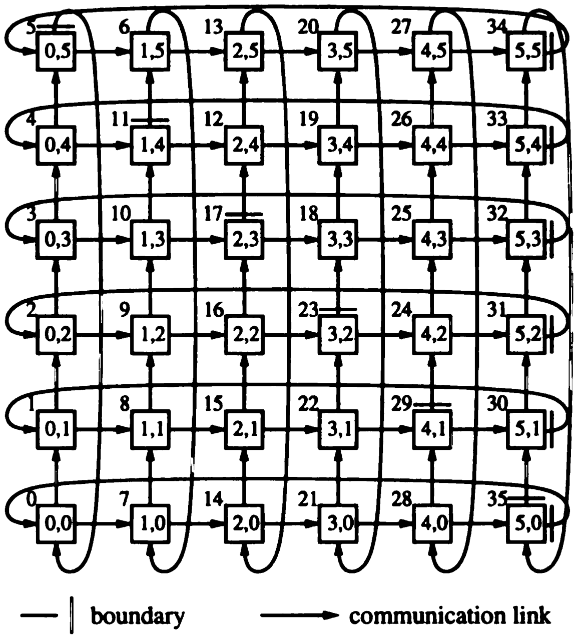


Figure 5.4. Hamiltonian circuit $\mathcal{H}$ in a 2D torus

Figure 5.5 shows the node orderings defined by $\mathcal{H}$ in a 3D ($4 \times 4 \times 4$) torus where, for clarity, the communication channels in dimension 2 (inter-plane) are not shown. In addition to the boundaries shown explicitly in Figure 5.5, every channel from plane 3 to plane 0 is also a boundary.

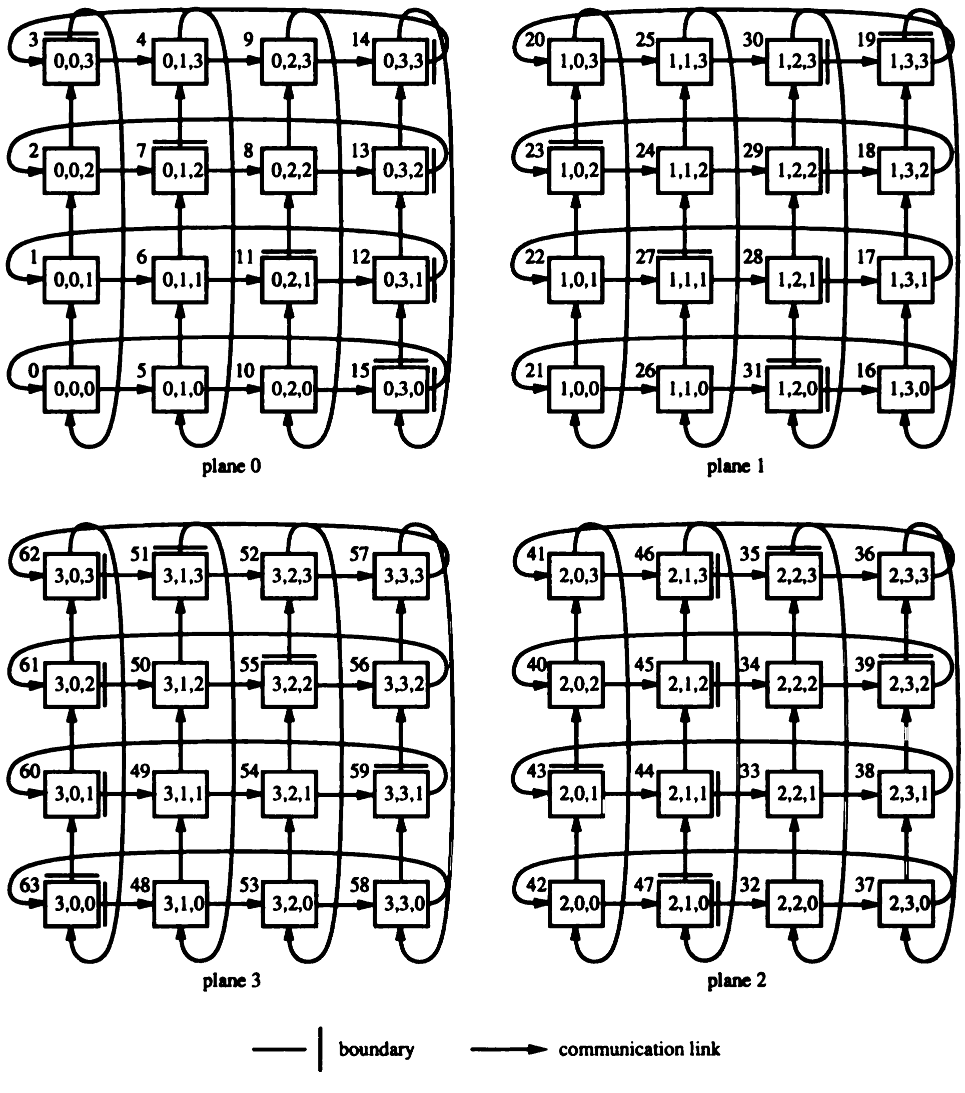


Figure 5.5. Hamiltonian circuit $\mathcal{H}$ in a 3D torus

We define notation $\ell_T(u)$ to be the ordinal position, or *label*, of a node u in torus T as determined by $\mathcal{H}_T$. Again, when it is clear which torus network is under consideration, we use notation $\ell(u)$. For example, in Figure 5.4, $\ell(0,0) = 0$, $\ell(0,1) = 1$, $\ell(1,0) = 7$, and so forth. We define $\mathcal{H}$ formally by defining the ordinal position, $\ell(u)$, of an arbitrary node, u. For a 1D torus, which is simply a ring, trivially, $\ell(u) = \sigma_0(u)$. For a 2D torus, as shown in Figure 5.4, we have

$$\ell(u) = [(\sigma_0(u) + \sigma_1(u)) \mod k] + k\,[\sigma_1(u)] \text{ for } n = 2$$

For the general case of a k-ary n-dimensional torus, $\mathcal{H}$ is defined by Equation 5.2:

$$\ell(u) = \sum_{i=0}^{n-1} \left[k^i \left(\left(\sum_{j=i}^{n-1} \sigma_j(u) \right) \mod k \right) \right] \text{ for } n \geq 1 \tag{5.2}$$

In Definition 5.1, node labels are used to formally define a boundary in a unidirectional torus.

Definition 5.1 *If u and v are two neighboring nodes (that is, if $u^i = v$ for some $0 \leq i \leq n-1$), then channel (u, v) is a boundary if and only if $\ell(u) > \ell(v)$.*

5.3 The Path Routing Function

In order to develop the path routing function, we first describe a path-based routing method for a restricted class of multi-destination messages in which the source node precedes, on $\mathcal{H}$, every destination node. We then extend this method to include arbitrary multi-destination messages.

5.3.1 Restrictive Routing

We consider a multi-destination message in which the destination nodes have been arranged in ascending order according to their position on $\mathcal{H}$. Such a message is defined by the node sequence $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$, where u_0 is the source node, $\{u_1, u_2, \ldots, u_{m-1}\}$ are the destination nodes, and $\ell(u_i) < \ell(u_j)$ for $0 \leq i < j \leq m-1$.

We can describe a multi-destination message path that begins at the source node u_0 and visits each destination node in order. At each step, the path routing function is applied to the current node and the next destination node, in order to determine the node adjacent to the current node to which the message is next routed. In order to provide minimal paths between destination nodes, we must restrict travel to those dimensions in which the addresses of the current node and the next destination node differ; these are called *useful dimensions*.

Given the above, we can define the path routing function (for this restrictive case) as follows: messages are always routed in the *lowest* useful dimension that does not cross a boundary. More formally, at the current node, u, the message is routed to the neighboring node, u^d, such that

1. The addresses of node u and the next destination node differ in dimension d;
2. Channel (u, u^d) is not a boundary; and
3. The value of d is minimal, given the above two restrictions.

For example, Figure 5.6 illustrates the application of this routing algorithm to the (restrictive) multi-destination message $\Phi = \{(3,2)^{23}, (4,3)^{25}, (4,5)^{27}, (5,1)^{30}, (5,4)^{33}\}$, where the first element, $(3,2)$, is the source node, and the destination nodes are ordered according to their labels (which have been placed as superscripts in the above list).

Since the multi-destination message path described above visits all nodes in an order that is consistent with the total ordering established by $\mathcal{H}$ (and does not utilize the cyclic property of $\mathcal{H}$), then all cycles of channel dependency are prevented, thereby ensuring deadlock-free routing.

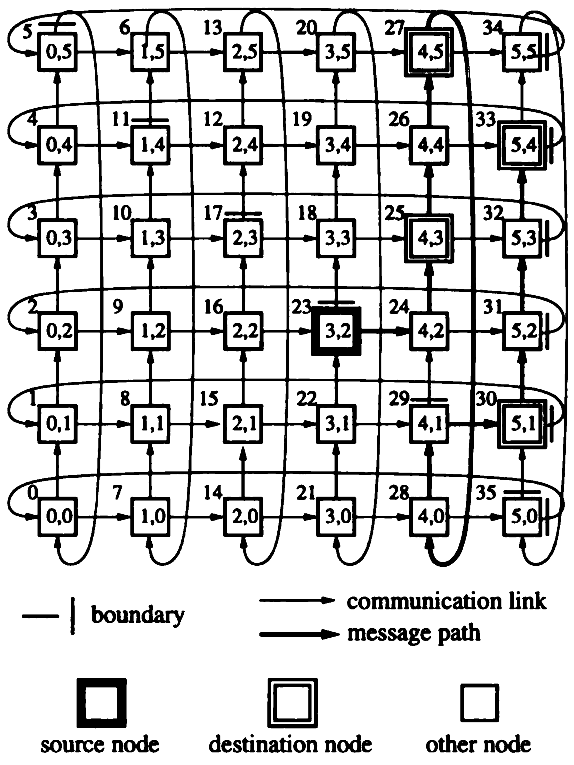


Figure 5.6. A (restrictive) multi-destination message in a 2D torus

5.3.2 General Routing

The restrictive path-based routing described above does not apply to messages in which one or more destination nodes precede the source node on $\mathcal{H}$. In order to provide a general path-based routing method that is deadlock-free, we extend the above restrictive routing mechanism through the use of virtual communication channels. It is known that deadlock-free deterministic routing cannot be achieved in a wormhole-routed torus network without the use of multiple virtual channel sets, even in the simplified case of unicast routing [57]. Thus, at least two virtual channel sets are required in order to provide deadlock-free deterministic path-based (or unicast) routing. We will, in fact, describe a path-based routing that requires only this lower bound of two virtual channel sets.

For ***unidirectional torus path-based routing*** (UTPR), there are two virtual channels sets, called *p*-channels and *h*-channels. Briefly, the *p*-channels ('*p*' for *pre*-boundary) are used by messages only prior to crossing a boundary; after crossing a boundary, messages use the *h*-channels ('*h*' for *high*-channel) for all remaining travel. Those messages that will not cross a boundary use *p*-channels exclusively. Each non-boundary physical communication link in the network has multiplexed onto it a *p*-channel and an *h*-channel. Only *h*-channels are required at boundary links. Formally, under UTPR, for each dimension d, $0 \leq d \leq k-1$, and for each node u,

1. There is a *p*-channel, c_{dpu}, from u to u^d whenever (u, u^d) is not a boundary, and
2. There is an *h*-channel, c_{dhu}, from u to u^d.

The rule for routing from node u to node v has been described above for the case where $\ell(u) < \ell(v)$. We must now consider the case where $\ell(u) > \ell(v)$. To route from a node u to a node v, where either $\ell(u) < \ell(v)$ or $\ell(u) > \ell(v)$, we use the following generalized routing rule: travel occurs in the *lowest* useful dimension that does not cross a boundary. If every useful dimension crosses a boundary, then travel occurs in the *highest* useful dimension.

By using *p*-channels prior to crossing a boundary, and *h*-channels thereafter, cycles of dependency among virtual channels, and thus deadlock, are impossible. We show in Section 5.5 that an arbitrary multi-destination message can be routed so that at most one boundary is crossed. Limiting a message to a single boundary is important, since with only two virtual channel sets, allowing messages to cycle through the network multiple times could cause deadlock, even if these messages are restricted to the cyclic order given by $\mathcal{H}$.

The path routing function for UTPR is described formally by the function $\mathcal{R}_{UTPR}$: $N \times \{\text{'p'},\text{'h'}\} \times N \rightarrow C$, which maps a *(current node, incoming virtual channel set, destination node)* triple into the next channel of the routing path. Let $\Delta(u, v)$ be the set of useful dimensions for routing from u to v. That is,

$$\Delta(u,v) = \{i \mid 0 \le i \le n-1 \text{ and } \sigma_i(u) \ne \sigma_i(v)\}$$

Let $\Delta_f(u,v)$ to be the useful dimensions that are not boundaries at u. Thus,

$$\Delta_f(u,v) = \{i \in \Delta(u,v) \mid (u,u^i) \text{ is not a boundary}\}$$

We define $\mathcal{R}_{UTPR}$, based on Δ and Δ_f, as follows.

$$\begin{aligned} \mathcal{R}_{UTPR}(u,\alpha,v) &= c_{u\beta d}, \text{ where} \\ d &= \begin{cases} \min(\Delta_f(u,v)), \text{ if } \Delta_f(u,v) \ne \emptyset \\ \max(\Delta(u,v)), \text{ otherwise} \end{cases} \\ \beta &= \begin{cases} \text{`p', if } \alpha = \text{`p' and } (u,u^d) \text{ is not a boundary} \\ \text{`h', otherwise.} \end{cases} \end{aligned} \tag{5.3}$$

A multi-destination message is routed from a source node to a destination node (or between successive destination nodes) by applying the $\mathcal{R}_{UTPR}$ function, first at the source, and then at each router through which the message travels, until the destination is reached. When a message enters the network, it is initially routed over a p-channel. As specified by Equation 5.3, the message continues to be routed over p-channels unless a boundary is crossed. After crossing a boundary, a message is routed on h-channels.

5.4 Message Preparation

The path routing function, realized in hardware within the node router, determines the route taken by a message between subsequent destination nodes (and between the source node and the first destination node). The *message preparation algorithm*, implemented in software within the source node processor, arranges the list of destination nodes in the message header, thereby determining the order in which the destination nodes are reached. The path routing function and message preparation algorithm must therefore be carefully designed in tandem, so that cycles of channel dependency, and thus deadlock, are avoided.

The path routing function $\mathcal{R}_{UTPR}$ defined in Equation 5.3 describes how messages can be routed between pairs of nodes; that is, between the source node and the first destination node, and between pairs of successive destination nodes of a multi-destination message. One way to maintain the deadlock-free properties of $\mathcal{R}_{UTPR}$ is to force the multi-destination message to visit the destination nodes in an order corresponding to $\mathcal{H}$. In addition, given the constraints of only two virtual channel sets, the message must be limited to one full cycle of $\mathcal{H}$.

In order to meet the above requirements, we introduce the notion of an *$\mathcal{H}$-cycle.* We then show how path-based routing can be applied to an arbitrary source and list of destination nodes that have been arranged into an $\mathcal{H}$-cycle. We first define an *$\mathcal{H}$-chain*, which is simply a sequence of nodes whose order is consistent with the *linear* (as opposed to cyclic) ordering established by $\mathcal{H}$.

Definition 5.2 *A sequence of nodes $\{u_0, u_1, u_2, \ldots, u_{m-1}\}$ is an $\mathcal{H}$-chain if and only if all elements are distinct, and $\ell(u_i) < \ell(u_{i+1})$ for $0 \leq i < m-1$.*

An $\mathcal{H}$-chain does not utilize the cyclic properties of $\mathcal{H}$ — the labels of the elements of an $\mathcal{H}$-chain are strictly increasing. An $\mathcal{H}$-cycle, on the other hand, is a sequence of nodes whose ordering is consistent with the *cyclic* ordering associated with $\mathcal{H}$, and is defined as an end-around *rotation* of an $\mathcal{H}$-chain.

Definition 5.3 *If $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$ is an $\mathcal{H}$-chain and u_s is an element of Φ, then $\{u_s, u_{s+1}, \ldots, u_{m-1}, u_0, u_1, \ldots, u_{s-1}\}$ is an $\mathcal{H}$-cycle with respect to u_s.*

For any $\mathcal{H}$-chain $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$, Φ is an $\mathcal{H}$-cycle with respect to u_0; that is, any $\mathcal{H}$-chain is also an $\mathcal{H}$-cycle with respect to the first element. As an example of the construction of an $\mathcal{H}$-cycle, we consider the multicast problem in a 2D (6×6) torus depicted in Figure 5.1, where the source node $(3,2)$ is to deliver a message to 9 destination nodes using a single multi-destination message. The problem is thus defined by the sequence

$$\Phi = \{\underline{(3,2)}^{23}, (0,5)^5, (4,5)^{27}, (3,4)^{19}, (5,4)^{33}, (4,3)^{25}, (1,2)^9, (2,1)^{15}, (5,1)^{30}, (1,0)^7\}$$

where the first element of Φ is the source node, and the destination nodes are listed in arbitrary order. For convenience, the source node is underlined, and the label, $\ell(u)$, of each node, u, is added as a superscript to the node address. (For reference, the node labels of a 2D (6×6) torus are illustrated in Figure 5.4.)

First, Φ is sorted according to labels of the node addresses to obtain the $\mathcal{H}$-chain

$$\Phi' = \{(0,5)^5, (1,0)^7, (1,2)^9, (2,1)^{15}, (3,4)^{19}, \underline{(3,2)}^{23}, (4,3)^{25}, (4,5)^{27}, (5,1)^{30}, (5,4)^{33}\}$$

Next, the $\mathcal{H}$-chain Φ' is *rotated* so that the source node, $(3,2)$, appears at the head of the list, resulting in the following $\mathcal{H}$-cycle

$$\Phi'' = \{\underline{(3,2)}^{23}, (4,3)^{25}, (4,5)^{27}, (5,1)^{30}, (5,4)^{33}, (0,5)^5, (1,0)^7, (1,2)^9, (2,1)^{15}, (3,4)^{19}\}$$

Finally, a single message can be routed, starting at the source node $(3,2)$, and then to each destination node in Φ'', in turn, according to the path routing function $\mathcal{R}_{UTPR}$ (Equation 5.3). Figure 5.7 illustrates the path of this multi-destination message as it is routed to the successive elements of the $\mathcal{H}$-cycle Φ''.

Figure 5.8 gives the message preparation algorithm, which is executed by the source node processor in order to build the multi-destination message header.

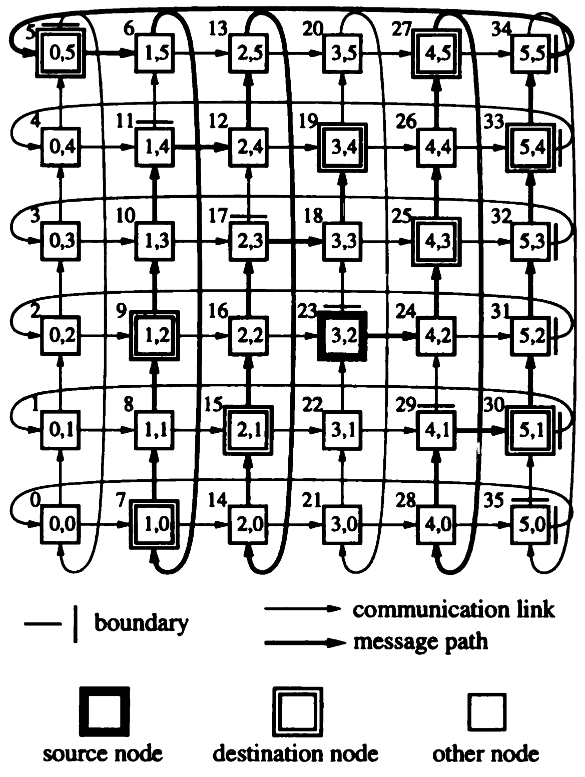


Figure 5.7. A multi-destination message in a 2D torus

5.5 Correctness

In order to implement the above path-based message routing, the source node applies the message preparation algorithm to the source and destination node addresses. The resultant $\mathcal{H}$-cycle is then used as the destination address list of a multi-destination message that will be routed, in turn, to each destination node, according to the path routing function $\mathcal{R}_{UTPR}$. Theorem 5.1 shows that this method provides an efficient, deadlock-free, path-based routing mechanism for unidirectional torus networks of arbitrary dimension.

Theorem 5.1 *If* $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$ *is an* $\mathcal{H}$*-cycle, then a multi-destination message, routed according to path routing function* $\mathcal{R}_{UTPR}$*, beginning at source node*

The Message Preparation Algorithm

```
Input: A sequence of nodes $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$,
    where $u_0$ is the source node.
Output: Message header $M$.
Procedure:
    1. Sort $\Phi$ according to an ascending ordering of the
       node labels $\ell(u_i)$ of each node $u_i$
    2. Rotate $\Phi$ so that $u_0$ is again the first element of $\Phi$
    3. $M = \Phi - \{u_0\}$
```

Figure 5.8. The message preparation algorithm for path-based routing

u_0 and routed through destination nodes $u_1, u_2, \ldots, u_{m-1}$, in that order, has the following properties.

1. All paths between successive destination nodes are minimal.
2. The network is deadlock-free under any and all combinations of such multi-destination messages.
3. The message uses all distinct physical channels.

In order to prove Theorem 5.1, we first present a series of lemmas. As a notational convenience throughout the following lemmas, we define ℓ_i to be the coefficient of the i^{th} term of Equation 5.2; that is, for any node u and for $0 \leq i \leq n-1$,

$$\ell_i(u) = \left(\sum_{j=i}^{n-1} \sigma_j(u)\right) \bmod k \tag{5.4}$$

Thus, we can write Equation 5.2 as

$$\ell(u) = \sum_{i=0}^{n-1} \left[k^i \, \ell_i(u)\right] \tag{5.5}$$

Lemma 5.1 *Let u be any node in a unidirectional torus, and let d be any dimension, $0 \leq d \leq n-1$. Then $\ell(u) > \ell(u^d)$ if and only if $\ell_d(u) > \ell_d(u^d)$.*

Proof: We consider the *maximum* amount that the value of $\ell(u)$ is effected by the first $d-1$ terms of Equation 5.2. Because of the modulo-k arithmetic, the value of $\ell_i(u)$ can change by at most $k-1$ as the value of u changes; thus, the corresponding aggregate change to the value of $\ell(u)$ due to terms 0 through $d-1$ of Equation 5.2 is bounded above by

$$\sum_{i=0}^{d-1}[k^i(k-1)] = k^d - 1.$$

Since k^d is the *minimum* amount by which a change in the d^{th} term of Equation 5.2 can effect the value of $\ell(u)$, and since terms $d+1$ through $n-1$ are not effected by the value of $\sigma_d(u)$, then the lemma is valid. □

Lemma 5.2 *Let u and v be any two distinct nodes in a unidirectional torus and let d be the greatest integer such that $\sigma_d(u) \neq \sigma_d(v)$. If channel (u, u^d) is a boundary, then $\ell(u) > \ell(v)$.*

Proof: By Definition 5.1, $\ell(u) > \ell(u^d)$. It then follows from Lemma 5.1 that $\ell_d(u) > \ell_d(u^d)$, and from the rules of modulo arithmetic, $\ell_d(u) = k-1$.

From the premise of the lemma, $\sigma_d(u) \neq \sigma_d(v)$ and $\sigma_j(u) = \sigma_j(v)$ for $j > d$, hence, $\ell_d(v) \neq k-1$ Therefore, due to the modulo-k arithmetic, $\ell_d(v) < k-1$. Finally, again by Lemma 5.1, $\ell(u) > \ell(v)$. □

Lemma 5.3 *Let u and v be any two distinct nodes in a unidirectional torus. If $\sigma_{n-1}(u) \neq \sigma_{n-1}(v)$ then $\sigma_{n-1}(u) > \sigma_{n-1}(v)$ if and only if $\ell(u) > \ell(v)$.*

Proof: From Equation 5.4, $\ell_{n-1}(u) = \sigma_{n-1}(u)$, so it suffices to show that $\ell_{n-1}(u) > \ell_{n-1}(v)$ if and only if $\ell(u) > \ell(v)$. By the same reasons given in the proof of Lemma 5.1, the value of $\sum_{i=0}^{n-2}[k^i\ell_i(u)]$ is bounded by

$$0 \leq \sum_{i=0}^{n-2} \left[k^i\ell_i(u)\right] \leq k^{n-1} - 1$$

Therefore, $\ell_{n-1}(u) > \ell_{n-1}(v)$ only if $\ell(u) > \ell(v)$, and similarly, $\ell_{n-1}(u) < \ell_{n-1}(v)$ only if $\ell(u) < \ell(v)$. Since $\ell_{n-1} \neq \ell_{n-1}(v)$ because $\sigma_{n-1}(u) \neq \sigma_{n-1}(v)$, then the lemma is proved. □

Lemma 5.4 *Let u and v be any two distinct nodes in a unidirectional torus. If $\ell(u) < \ell(v)$ then there exists a dimension d such that $\sigma_d(u) \neq \sigma_d(v)$ and $\ell(u) < \ell(u^d) \leq \ell(v)$. If $\ell(u) > \ell(v)$ then there exists a dimension d such that $\sigma_d(u) \neq \sigma_d(v)$ and either $\ell(u) < \ell(u^d)$ or $\ell(u^d) \leq \ell(v)$.*

Proof: Part I. $\ell(u) < \ell(v)$: The proof is by induction on n, the dimensionality of the torus. If $n = 1$, the torus is a ring, therefore $\ell(u) = \sigma_0(u) = u$; likewise, $\ell(v) = v$. The assertion is then trivially true. We now assume that the lemma is true for $n-1$ (although this assumption will only be needed in Case 1, below).

Case 1. $\sigma_{n-1}(u) = \sigma_{n-1}(v)$: In this case, all routing occurs in sub-tori of dimension $n-1$. If $\sigma_{n-1}(u) = 0$, then the Hamiltonian Circuit $\mathcal{H}$ within this sub-tori is equivalent to $\mathcal{H}_T$ within a torus, T, of dimension $n-1$. Otherwise, by the symmetry of the torus network, the sub-tori is isomorphic to T. Thus, by induction, there is a dimension d such that $\sigma_d(u) = \sigma_d(v)$ and $\ell(u) < \ell(u^d) \leq \ell(v)$.

Case 2. $\sigma_{n-1}(u) \neq \sigma_{n-1}(v)$: By Lemma 5.2, channel (u, u^{n-1}) is not a boundary, and by Lemma 5.3, $\sigma_{n-1}(u) < \sigma_{n-1}(v)$.

Case 2a. $\sigma_{n-1}(v) - \sigma_{n-1}(u) > 1$: Then $\sigma_{n-1}(u^{n-1}) < \sigma_{n-1}(v)$, and from Lemma 5.3, $\ell(u^{n-1}) < \ell(v)$. Since channel (u, u^{n-1}) is not a boundary, $\ell(u) < \ell(u^{n-1})$.

Case 2b. $\sigma_{n-1}(v) - \sigma_{n-1}(u) = 1$: This case is divided into two sub-cases, as follows.

Case 2b.i. There exists a dimension $d < n-1$ such that $\sigma_d(u) \neq \sigma_d(v)$ and channel (u, u^d) is not a boundary: Then $\sigma_{n-1}(u^d) = \sigma_{n-1}(u) < \sigma_{n-1}(v)$, so from Lemma 5.3, $\ell(u^d) < \ell(v)$. Since channel (u, u^d) is not a boundary, $\ell(u) < \ell(u^d)$.

Case 2b.ii. There is no dimension $d < n-1$ such that $\sigma_d(u) \neq \sigma_d(v)$ and channel (u, u^d) is not a boundary: In this case we have that for all $d < n-1$ such that $\sigma_d(u) \neq \sigma_d(v)$, channel (u, u^d) is a boundary, and hence, $\ell(u) > \ell(u^d)$; therefore, we must show that $\ell(u^{n-1}) \leq \ell(v)$.

Let d be any dimension such that $d < n-1$ and $\sigma_d(u) \neq \sigma_d(v)$. Since channel (u, u^d) is a boundary, then by Lemma 5.1, $\ell_d(u) = k-1$. Because channel (u, u^{n-1}) is not a boundary, then from Lemma 5.3, $\sigma_{n-1}(u^{n-1}) > \sigma_{n-1}(u)$, so $\sigma_{n-1}(u^{n-1}) = \sigma_{n-1}(u) + 1$. It then follows, from Equation 5.5, that $\ell_d(u^{n-1}) = 0$.

Since $\sigma_d(u) = \sigma_d(u^{n-1})$ for all $d < n-1$, we have that $\ell_d(u^{n-1}) = 0$ for all $d < n-1$ such that $\sigma_d(u^{n-1}) \neq \sigma_d(v)$. Therefore, from Equation 5.5, $\ell(u^{n-1}) \leq \ell(v)$.

Part II. $\ell(u) > \ell(v)$: If there is a dimension d such that $\sigma_d(u) \neq \sigma_d(v)$ and channel (u, u^d) is not a boundary, then trivially, $\ell(u) < \ell(u^d)$. Otherwise, we know that for all d such that $\sigma_d(u) \neq \sigma_d(v)$, channel (u, u^d) is a boundary, and we must show that for at least one such dimension, $\ell(u^d) \leq \ell(v)$.

For reasons similar to those given in Part I of this proof (Case 2b.ii), if we choose d as the greatest integer such that $\sigma_d(u) \neq \sigma_d(v)$, then $\ell_e(u^d) = 0$ for all $e < d$ such that $\sigma_e(u^d) \neq \sigma_e(v)$, and thus, $\ell(u^d) \leq \ell(v)$. □

Lemma 5.5 *Let u and v be any two distinct nodes in a unidirectional torus and let w be the first node (after node u) on the path from u to v, as determined by the path routing function $\mathcal{R}_{UTPR}$. If $\ell(u) < \ell(v)$ then $\ell(u) < \ell(w) \leq \ell(v)$. If $\ell(u) > \ell(v)$ then either $\ell(u) < \ell(w)$ or $\ell(w) \leq \ell(v)$.*

Proof: Let $c_{u\beta e} = \mathcal{R}_{UTPR}(u, \alpha, v)$ be the first channel on the path from u to v. Then $w = u^e$. Although the value of the virtual channel set, α, is not specified above, the dimension, e, produced by the $\mathcal{R}_{UTPR}$ function does not depend on α (Equation 5.3).

Part I. $\ell(u) < \ell(v)$: By Lemma 5.4, there exists a useful dimension, d, such that (u, u^d) is not a boundary and $\ell(u^d) \leq \ell(v)$. From Definition 5.1 and Lemma 5.1 follows that if i and j are two dimensions such that $i < j$ and neither channel (u, u^i) nor channel (u, u^j) is a boundary, then $\ell(u^i) < \ell(u^j)$. Since $\mathcal{R}_{UTPR}$ always selects the minimum dimension among useful non-boundary dimensions, then $\ell(u) < \ell(w) \leq \ell(v)$.

Part II. $\ell(u) > \ell(v)$: If channel (u, w) is a boundary then all channels in useful dimensions are boundaries, since $\mathcal{R}_{UTPR}$ always selects a non-boundary channel if one exists in a useful dimension. Thus, by Lemma 5.4, there exists a useful dimension, d, such that $\ell(u^d) \leq \ell(v)$. Since $\ell(v) < \ell(u)$, then (u, u^d) is a boundary.

From Definition 5.1 and Lemma 5.1 follows that if i and j are two dimensions such that $i > j$ and both channel (u, u^i) and channel (u, u^j) are boundaries, then $\ell(u^i) < \ell(u^j)$. Since $\mathcal{R}_{UTPR}$ always selects the maximum dimension among useful boundary dimensions, then $\ell(w) \leq \ell(v)$.

If, on the other hand, channel (u, w) is not a boundary, then by Definition 5.1, $\ell(w) > \ell(u)$. □

Lemma 5.6 *Let u and v be any two distinct nodes in a unidirectional torus. If $\ell(u) < \ell(v)$ then the path from u to v, as determined by the path routing function $\mathcal{R}_{UTPR}$, does not contain a boundary. If $\ell(u) > \ell(v)$ then the path contains exactly one boundary.*

Proof: Let $w_0, w_1, w_2, \ldots, w_q$ be the sequence of nodes on the path from u to v, where $w_0 = u$ and $w_q = v$.

Part I. $\ell(u) < \ell(v)$: By Lemma 5.5, $\ell(w_0) < \ell(w_1) \leq \ell(w_q)$, and by extension, $\ell(w_0) < \ell(w_1) < \ell(w_2) < \cdots < \ell(w_q)$. Hence, the path contains no boundaries.

Part II. $\ell(u) > \ell(v)$: Since $\ell(w_0) > \ell(w_q)$, then there exists an integer i $(0 \leq i \leq q-1)$ such that $\ell(w_i) > \ell(w_{i+1})$. Choosing the value of i as the least such integer gives $\ell(w_0) < \ell(w_1) < \cdots < \ell(w_i) > \ell(w_{i+1})$. By Lemma 5.5, $\ell(w_{i+1}) \leq \ell(w_q)$, and by extension, $\ell(w_{i+1}) < \ell(w_{i+2}) < \cdots < \ell(w_q)$. Hence, the path contains exactly one boundary, channel (w_i, w_{i+1}). □

Proof of Theorem 5.1 With the use of the above lemmas, the three assertions stated in the theorem are now proved, in turn.

Assertion 1 (minimal paths): Since the path routing function $\mathcal{R}_{UTPR}$ selects only useful dimensions, then all paths are minimal.

Assertion 2 (deadlock-free): In order to show that the network is deadlock-free, we first define a total ordering on the virtual channels of the network, and then show that all messages reserve virtual channels in an order that is consistent with this total ordering.

For any channel $c_{u\alpha d}$, define $\lambda(c_{u\alpha d})$ as follows.

$$\lambda(c_{u\alpha d}) = \begin{cases} 0.\,\ell(u).d \text{ if } \alpha = \text{'p'} \\ 1.\,\ell(u).d \text{ if } \alpha = \text{'h'} \end{cases}$$

A lexicographical ordering of the three-part labels, $\lambda(c)$, of each channel, c, defines a total ordering of the virtual channels. All p-channels precede all h-channels in this ordering; among the same virtual channels set, the ordering is determined by the label, $\ell(u)$, of the source node, u. Finally, channels of the same virtual channel set and source node are ordered by the dimension, d, in which they travel.

We consider two cases with respect to the $\mathcal{H}$-cycle, Φ.

Case 1. Φ is an $\mathcal{H}$-chain: By Definition 5.1, $\ell(u_i) < \ell(u_{i+1})$ for $0 \leq i < m-1$, and by Lemma 5.6, the path from node u_0 to node u_{m-1} does not contain a boundary; thus, from Equation 5.3, the path contains only p-channels. Virtual channels are therefore used by the path from node u_0 to node u_{m-1} in an order that is consistent with the total ordering described above.

Case 2. Φ is not an $\mathcal{H}$-chain: Since Φ is an $\mathcal{H}$-cycle, and is not an $\mathcal{H}$-chain, then there exists an integer, q, $1 \leq q \leq m-1$, such that $\ell(u_0) > \ell(u_{m-1})$ and

$$\ell(u_0) < \ell(u_1) < \cdots < \ell(u_{q-1}) > \ell(u_q) < \ell(u_{q+1}) < \cdots < \ell(u_{m-1})$$

By applying Lemma 5.6 to each pair of consecutive nodes in Φ, the path from node u_0 to node u_{m-1} contains exactly one boundary. Thus, by Equation 5.3, the path uses p-channels prior to the boundary and h-channels thereafter, and from Definition 5.1, both the sequence of p-channels and the sequence of h-channels on the path visit nodes in an ascending order of node labels, $\ell(u)$. Thus, the order of all virtual channels on the path from node u_0 to node u_{m-1} is consistent with the total ordering described above.

Since all messages reserve virtual channels in an order that is consistent with the total ordering of virtual channels defined above by λ, then cycles of channel dependency, and thus deadlock, cannot occur.

Assertion 3 (distinct physical channels): From the above proof of Assertion 2, it is clear that the path from node u_0 to node u_{m-1} does not visit any node more than once, therefore, it cannot contain a multiple occurrence of a physical channel. $\square$

Because unicast communication is also essential, it should be supported efficiently and in a way that is compatible with other network communication such as multi-destination messages. Since a unicast message is a special case of a multi-destination message in which there is only one destination node, it follows that minimal, deadlock-free unicast routing is also provided by the above routing mechanism. Furthermore, all combinations of multi-destination and unicast messages can coexist without possibility of network deadlock.

5.6 Implementation Issues

Because the path routing function must be implemented in hardware, it must be simple. Thus, the path routing function must be designed so that the output channel on which an incoming message is to be forwarded can be quickly determined by examining only the first flit of the message header (which indicates the next destination), and perhaps also taking into account the channel on which the message is arriving. Also, the router must be able to quickly decide if the current node is a destination of the incoming message, and if so, whether there are additional destinations to which the message must be forwarded.

5.6.1 Multi-Destination Message Format

With unicast wormhole routing, the header flit of a message contains the address of the destination node (absolute addressing), or perhaps some indication of the direction and distance from the current node to the destination node (relative addressing). In path-based routing for messages with arbitrary destination sets, the message worm can be prefixed with a list of destination node addresses. The order of these addresses corresponds to the order in which the destination nodes will be visited. Once the message header reaches the first destination node in the list, the router at that node removes its own address from the head of the list and forwards the remainder of the message worm towards the next destination node, while simultaneously copying the message contents to the local host memory. The router at the last destination node copies the message to the local host memory, but does not forward the message.

In this way, the address of the next destination node, used by the router at each node to determine the next node in the path of a message, is always at the head of the message worm. Thus, as the head of the message advances through the network, each router at which the message arrives need only examine the first flit of the message in order to identify the outgoing channel over which the message will be routed. In the case of intermediate destination nodes, the router first identifies that the first flit of the message is the local address, and after discarding this flit, considers the next flit, which represents the address of the next destination, to determine message routing. Upon recognizing that the message is addressed to the local node, the router also prepares to copy the flits of the message body to the local host.

In order to identify the last destination address in the message header, and consequently, the beginning of the message body, the address of the last destination node can be duplicated in the message header. The occurrence of two consecutive and identical destination addresses then signifies the end of the destination address list. The message format corresponding to the $\mathcal{H}$-cycle $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$, which represents a multi-destination message from source node u_0 to destination nodes $u_1, u_2, \ldots, u_{m-1}$, is depicted in Figure 5.9.

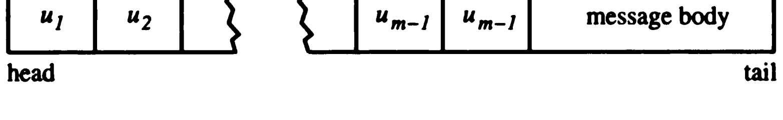


Figure 5.9. Multi-destination message format

This format represents the original message, as constructed by the host at the source node, u_0. As the message progresses through the network, the header becomes shorter as the routers at each destination node remove their respective addresses from the head of the message.

In order to reduce the message header length, destination encoding schemes have been proposed that avoid the need to list each destination address separately. For example, Kim and Kim [31] describe the use of address masks, which allow "don't-care" bits in the address field in order to match a group of destination addresses with a single entry in the message header. While such address encoding methods can be useful in reducing the length of the message header in cases where the destination nodes form a regular pattern, they add complexity to the routing hardware, and fail to offer an advantage with arbitrary destination sets.

5.6.2 The Flit Forwarding Algorithm

Figure 5.10 shows the flit forwarding algorithm implemented in a router in order to support the proposed path-based routing. The **recv** function reads the next incoming flit, while the **send** function transmits a flit over the specified outgoing channel. In cases where channel contention occurs so that a flit cannot be sent over the specified channel, the **send** function can be considered to block until the channel is available. The **send-local** function copies a flit to the local host memory. The flit forwarding algorithm, as shown in Figure 5.10, is invoked for each incoming message; thus, there may be concurrent invocations of the algorithm if there are messages arriving concurrently on different incoming channels.

The above discussion assumes the use of absolute addressing. In the case of relative addressing, rather than duplicating the address of the last destination node, the list of destination addresses can simply be appended with a zero offset (which, conceptually, is equivalent to duplicating the last address, since an offset of zero from

The Multi-Destination Flit Forwarding Algorithm

Input: Local node u and incoming virtual channel $c_{w\alpha d}$.
Output: Forward incoming message to outgoing channel and/or local host, as appropriate.
Procedure:

```
flit1 = recv ($c_{w\alpha d}$) ;
if flit1 = $u$ then                          // $u$ is a destination
    isDest = true ;
    flit2 = recv ($c_{w\alpha d}$) ;
    if flit2 = flit1 then                    // $u$ is the last destination
        isLastDest = true
    else
        isLastDest = false ;
        nextDest = flit2
    endif
else                                         // $u$ is not a destination
    isDest = false ;
    isLastDest = false ;
    nextDest = flit1
endif

if not isLastDest then
    $c = \mathcal{R}_{UTPR}(u, \alpha, \text{nextDest})$ ;   // set output channel
    send ($c$, nextDest)      // send first flit (address of next dest.)
endif

if isDest then
    while flit2 $\neq$ flit1 do    // skip to message body
        flit1 = flit2 ;
        flit2 = recv ($c_{w\alpha d}$) ;
        send ($c$, flit2)
    endwhile
endif

while not end-of-message do
    flit1 = recv ($c_{w\alpha d}$) ;
    if isDest then
        send-local (flit1) ;
    if not isLastDest then
        send ($c$, flit1) ;
endwhile
```

Figure 5.10. The flit forwarding algorithm for path-based routing

the last destination is, indeed, the last destination). In conjunction with this change, very minor adjustments to the flit forwarding algorithm are also required in order to support relative addressing.

5.6.3 Boundary Identification

In order for the router at a node, u, to implement the path routing function $\mathcal{R}_{UTPR}$ given in Equation 5.3, the outgoing channels at node u that are boundaries must be identified. Of course, one way for node u to accomplish this task is to compute the address of each neighbor of u using Equation 5.1, and to then compute the labels of these neighboring nodes using Equation 5.2. The boundary channels can then be identified by comparing the label of u with those of the neighboring nodes, as described in Definition 5.1. There is, however, a much simpler method of determining which of the routing dimensions correspond to boundary channels, as provided for by Lemma 5.7.

Lemma 5.7 *For any node u, and any dimension d $(0 \leq d \leq n-1)$, channel (u, u^d) is a boundary if and only if:*

$$\left(\sum_{j=d}^{n-1} \sigma_j(u)\right) \bmod k = k - 1$$

Proof: The lemma follows directly from Lemma 5.1. □

The configuration of boundaries in a network is completely static; thus, the boundary dimensions for each node can be identified once during system startup, and need never be recomputed. Alternatively, the identification of boundaries can be made a part of the node's static configuration, much the same as the local address of a node is part of that node's static configuration.

5.6.4 Implementation of the Path Routing Function

The formal definition of the path routing function $\mathcal{R}_{UTPR}$, given by Equation 5.3, is intended to convey the concept of how the routing path of a message is determined; it does not, however, represent an efficient *implementation* of the function. An algorithmic view of the function $\mathcal{R}_{UTPR}$, more suitable for implementation in a router, is given in Figure 5.11. The predicate **boundary** simply indicates whether an outgoing channel on the specified dimension is a boundary. For messages originating at the local node, u (as opposed to those that arrive on network channels), the input parameter α is set initially to 'p'.

The Path Routing Function

```
Input: Local node $u$, next destination node $v$,
   and virtual channel set of incoming message $\alpha \in \{$'p','h'$\}$.
Output: Outgoing virtual channel set and dimension.
Procedure:
   for $i = 0$ to $n - 1$ do
        if $\sigma_i(u) \neq \sigma_i(v)$ then        // useful dimension?
             if not boundary(i) then
                  return $(\alpha, i)$   // lowest non-boundary dimension
             else
                  $d = i$
             endif
        endif
   endfor
   return ('h', $d$)        // highest boundary dimension
```

Figure 5.11. Implementation of the path routing function $\mathcal{R}_{UTPR}$

In actuality, when the dimensionality, n, of a torus is small, as is the case with all current and proposed machines, the $\mathcal{R}_{UTPR}$ path routing function can be easily implemented with a simple combinational logic circuit.

5.7 Conclusions

In this chapter, the area of path-based message routing in unidirectional torus networks with IR capabilities has been studied. An efficient, deadlock-free path-based routing method was presented. This method is deadlock-free under all combinations of network traffic, provides minimal routing paths between subsequent destination nodes, and requires only the lower bound of two virtual channel sets. In the following chapter, this routing mechanism is used in the development of path-based multicast algorithms for torus networks with unidirectional communication links.

CHAPTER 6

Path-Based Multicast in Unidirectional Torus Networks

In this chapter, the path-based routing mechanism described in Chapter 5 is used as a basis for a family of efficient, deadlock-free path-based multicast algorithms for torus networks with unidirectional communication links.

In order to perform a multicast operation using multi-destination messages, one or more communication steps may be used. Methods that reach all destination nodes in one communication step are termed *single-phase*, while those that require more than one step are called *multi-phase*. During the first phase of a multi-phase multicast, the source node sends a single message to a subset of the destination nodes. During subsequent phases, some (perhaps all) of the nodes that have already received the message each send a multi-destination message to a distinct subset of the nodes that have not yet received the message. This process continues until the message has reached every destination node.

The remainder of this chapter is organized as follows: In Section 6.1 a single-phase multicast algorithm is presented. This algorithm follows directly from the path-based routing method presented in Chapter 5. A generalized multi-phase multicast algorithm is described in Section 6.2. This algorithm constitutes a family of path-based multicast algorithms. In Section 6.3, a specific instance of this generalized multicast algorithm is presented. By incorporating the topology of the torus network when partitioning the multicast destinations into individual multi-destination messages, the algorithm completes a multicast operation in n phases, where n is the number of dimensions in the network. Another instance of the generalized multi-phase multicast algorithm is described in Section 6.4. This algorithm partitions the destination nodes so that all messages produced by a multicast operation are addressed to a nearly equal number of destination nodes. Implementation issues specific to multi-phase multicast algorithms are addressed in Section 6.5. In Section 6.6, the results of a simulation study are presented. This study compares the performance of the multicast algorithms developed in this chapter, as well as a unicast-based multicast method based on the same routing mechanism. Conclusions are presented in Section 6.7.

6.1 The S-Torus Multicast Algorithm

The path-based routing method described in Chapter 5 provides a mechanism whereby any node can send a single multi-destination message to an arbitrary set of destination nodes within the network. When a single message is used in this way to perform a complete multicast operation, we call this process the *S-torus multicast algorithm* ('S' for *single* phase).

There are several advantages to implementing multicast communication with a single multi-destination message. The operation requires only one communication step. Hence, for long messages, full advantage is taken of the communication pipelining of wormhole routing. Also, the only processor involved in the operation is that of the source node. Each destination node receives the message via the node router, without the need to use the local processor to relay the message to other destination nodes.

However, the S-torus algorithm also suffers from some disadvantages. For example, a single message used to reach all nodes in the network, as in the case of a broadcast operation, will have a total path length of $N-1$. Such extremely large path lengths are likely to result in poor performance in large networks due not only to the length of the path traveled by the message flits from the source node to the last destination node, but also due indirectly to the network congestion caused by the many communication channels reserved by the message for the duration of the operation. In addition, a single-phase approach to multicast does not exploit the communication parallelism that is possible when concurrent messages are used to complete the operation.

6.2 The M-Torus Generalized Multicast Algorithm

Due to the limitations of the single-phase S-torus algorithm, we also consider multi-phase multicast algorithms. We describe a generalized multi-phase multicast algorithm that combines multi-destination messages in order to form a coverage of the destination nodes. Since this algorithm uses only messages that have been shown to result in a deadlock-free network, then the algorithm is also deadlock-free. We assume that a multicast operation is described by an $\mathcal{H}$-cycle, Φ, where the first element of Φ is the destination node. We define a *linear partitioning* of an $\mathcal{H}$-cycle, which will be used as a basis for the generalized multi-phase multicast algorithm.

Definition 6.1 *A linear partitioning of an $\mathcal{H}$-cycle $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$ is a set of non-empty sub-sequences $\Omega = \{\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_{r-1}\}$ such that $r \geq 2$ and $\Phi = \Phi_0 \parallel \Phi_1 \parallel \Phi_2 \parallel \ldots \parallel \Phi_{r-1}$, where the symbol '$\parallel$' represents concatenation of lists. Each element, Φ_i, of Ω is written as $\Phi_i = \{u_{i,0}, u_{i,1}, \ldots, u_{i,(m_i-1)}\}$.*

Lemma 6.1 *If $\Omega = \{\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_{r-1}\}$ is a linear partitioning of an $\mathcal{H}$-cycle Φ, then*

1. Each sub-sequence Φ_i is an $\mathcal{H}$-cycle, $0 \leq i \leq r-1$.
2. At least $r-1$ of the r sub-sequences are $\mathcal{H}$-chains. The remaining sub-sequence is either an $\mathcal{H}$-chain or an $\mathcal{H}$-cycle.

Proof: The lemma follows directly from Definitions 5.2, 5.3, and 6.1. □

Given an $\mathcal{H}$-cycle Φ that describes a multicast operation, a linear partitioning of Φ can be used to define the first phase of a multi-phase path-based multicast operation. In this first phase, the source node sends a multi-destination message according to the path-based routing method described in Chapter 5. The destination nodes of this message comprise the first elements of each of the partitions of Φ (except that the source node, which is always the first element of the first partition, is, of course, not included as a destination). After this first phase is complete, the multicast problem has, in effect, been partitioned into a set of smaller multicast problems, each corresponding to one of the partitions of Φ. The source node of each of these new multicasts is the first element, and the destination nodes are the remaining elements, of the respective partition.

The *M-torus multicast algorithm* is shown in Figure 6.1. This algorithm implements a multi-phase path-based multicast operation on an input sequence that has been arranged as an $\mathcal{H}$-cycle. As shown in Theorem 6.1, the resulting multicast operations are deadlock-free, while the constituent multi-destination messages are contention-free.

Theorem 6.1 *The M-torus algorithm applied to an $\mathcal{H}$-cycle $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$ results in a contention-free multicast from source node u_0 to destination nodes $u_1, u_2, \ldots, u_{m-1}$. Furthermore, the linear partitioning need not be consistent across*

```
The M-Torus Multicast Algorithm

Input: Message, $M$, and $\mathcal{H}$-cycle $\Phi = \{u_0, u_1, u_2, \ldots, u_{m-1}\}$,
      where $u_0$ is the local address.
Output: Performs a multi-phase, path-based multicast to
    destination nodes $u_1, u_2, \ldots, u_{m-1}$.
Procedure:
    if $|\Phi| > 1$ then
        1. Let $\Omega = \{\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_{r-1}\}$ be a linear partitioning of $\Phi$.
        2. Send a multi-destination message to the sequence of
           destination nodes $\{u_{1,0}, u_{2,0}, \ldots, u_{(r-1),0}\}$.
        3. Each node $u_{i,0}$ $(0 \le i \le n-1)$ invokes the M-torus algorithm,
           recursively, with the input $\mathcal{H}$-cycle set to $\Phi_i$.
    endif
```

Figure 6.1. The M-torus algorithm for multicast

the distributed invocations of the algorithm, nor over the successive invocations at any single node.

Proof: From definition 6.1, each partition is written as $\Phi_i = \{u_{i,0}, u_{i,1}, \ldots, u_{i,(m_i-1)}\}$. For each partition Φ_i, **range**(Φ_i) is defined as follows.

$$\textbf{range}(\Phi_i) = \begin{cases} \{u \mid \ell(u_{i,0}) \le \ell(u) \le \ell(u_{i,(m-1)})\} \text{ if } \Phi_i \text{ is an } \mathcal{H}\text{-chain} \\ \{u \mid \ell(u_{i,0}) \le \ell(u) \text{ or } \ell(u) \le \ell(u_{i,(m-1)})\} \text{ if } \Phi_i \text{ is not an } \mathcal{H}\text{-chain} \end{cases}$$

From Lemma 5.6 follows that the above sets are all disjoint, that is, **range**$(\Phi_i) \bigcap$ **range**$(\Phi_j) = \emptyset$ whenever $i \neq j$.

We now consider the multi-destination messages generated *within* a partition, say partition Φ_i. The first such message is generated by node $u_{i,0}$. Since the source and destination node list of each such message is ordered according to a sub-sequence (partition) of the $\mathcal{H}$-cycle Φ_i, then by Lemma 5.6, every node visited by the message is an element of **range** (Φ_i). We therefore conclude that, once the original multicast operation has been partitioned into r sub-problems, each corresponding to an $\mathcal{H}$-cycle, the messages generated by these sub-problems do not visit common nodes. That is to say, there is no contention *between* any two sub-problems.

Since the message generated in Step 2 of the algorithm (Figure 6.1) precedes all other messages of the multicast, then it cannot contend with any other message produced by the operation.

Recursively, each multicast sub-problem corresponding to one of the partitions, Φ_i, is itself contention-free. Therefore, the entire multicast operation is contention-free.

Since no assumptions have been made in this proof regarding the nature of any of the partitionings performed in Step 1 of the algorithm, except that they are linear partitionings, then the second assertion of the theorem is also valid. □

The M-torus algorithm actually represents a family of multi-phase multicast algorithms, whose specific instances are determined by the partitioning method used in Step 1. For example, if the input $\mathcal{H}$-cycle is partitioned into sub-sequences whose lengths are all one, then the M-torus algorithm will produce a single-phase multicast equivalent to the S-torus algorithm. On the other hand, if the input $\mathcal{H}$-cycle is always partitioned into exactly two subsequences, then all messages generated by the algorithm will have only a single destination, thus resulting in a unicast-based implementation.

Between these two extremes are a multitude of partitioning schemes, each resulting in a different version of the M-torus algorithm. We will examine two particular partitionings; namely (1) *dimensional partitioning* and (2) *uniform partitioning.*

6.3 The M$_d$-Torus Multicast Algorithm

By examining Statement 2 of the M-torus algorithm (Figure 6.1) and the definition of a linear partitioning (Definition 6.1), it can be seen that the multi-destination messages generated by the M-torus algorithm must reach destination nodes that may be widely distributed over the input $\mathcal{H}$-cycle.

Since higher-dimension channels traverse a greater span of $\mathcal{H}$ than do channels of lower dimension (we refer to Equation 5.2 and, for example, Figures 5.4 and 5.5), we consider partitionings that result in messages that cross higher-dimension channels between subsequent destination nodes. In this way, the path length of the constituent messages of the M-torus algorithm can be controlled. A *dimensional partitioning* is such a method. A dimensional partitioning of order d, in effect, partitions the nodes of an $\mathcal{H}$-cycle into their respective sub-tori of dimension d. For example, in a 3D torus, as shown in Figure 5.5, each element of a dimensional partitioning of order 2 corresponds to a *plane* of the network. Each plane of a 3D torus is, itself, a 2D torus. In a similar manner, a dimensional partitioning of order 1 partitions a network into 1D tori, or rings, corresponding to the columns of nodes in Figures 5.4 and 5.5.

Definition 6.2 *A linear partitioning Ω of an $\mathcal{H}$-cycle Φ is a dimensional partitioning of order d if and only if*

1. For each sub-sequence $\Phi_a \in \Omega$, and for any two elements $u, v \in \Phi_a$, $\sigma_i(u) = \sigma_i(v)$, for $d \leq i \leq n-1$; and
2. For any two sub-sequences $\Phi_a, \Phi_b \in \Omega$, where $a \neq b$, and any two elements $u \in \Phi_a$ and $v \in \Phi_b$, there exists some integer i, $d \leq i \leq n-1$, such that $\sigma_i(u) \neq \sigma_i(v)$.

By using dimensional partitionings, we create a specific instance of the M-torus algorithm, as follows (please refer to Figure 6.1). During the first communication phase, the partition, Ω, is a dimensional partitioning of order $n-1$. During the second phase, a dimensional partitioning of order $n-2$ is used, and so forth, until the

n^{th} and last phase, where Ω is a dimensional partitioning of order 0, which is simply a partitioning of Φ into individual nodes. This combination of the M-torus algorithm with dimensional partitioning is referred to as the *M_d-torus multicast algorithm.*

As an example, in a 3D torus, during the first phase of the M_d-torus algorithm, the destinations are partitioned into their respective 2D planes. A multi-destination message is sent by the source node to a single destination in each plane (among those planes that contain destination nodes). During the second phase, destinations reached during the first phase, as well as the original source node, each partition their respective 2D plane into 1D rings and send a multi-destination message that reaches one destination node in each of these rings. Finally, during the third phase, there is exactly one destination node in each 1D ring (the columns in Figure 5.5) that has received the message; each of these destinations sends a multi-destination message to cover the remaining destinations in the respective 1D ring.

6.4 The M_u-Torus Multicast Algorithm

The M_d-torus algorithm partitions the destination nodes based only on the structure of the underlying torus network, without regard to the actual destinations of a particular multicast operation (except that partitions corresponding to sub-tori without destination nodes are not created). In some cases, it is useful to consider the structure of the set of destination nodes when partitioning the associated $\mathcal{H}$-cycle. For example, limiting the number of destination nodes reached by any single message reduces the length of the message header (which contains the list of destination nodes addresses) and tends to also reduce the message path length.

A method that allows the number of destinations per message to be controlled is *uniform partitioning*, in which the $\mathcal{H}$-cycle is divided into a specified number of sub-sequences whose sizes are as nearly equal as possible. Uniform partitioning is defined as follows.

Definition 6.3 *A linear partitioning Ω of an $\mathcal{H}$-cycle Φ is a uniform partitioning of size r if and only if*

1. $|\Omega| = r$; and
2. For each partition $\Phi_a \in \Omega$, either $|\Phi_a| = \lceil m/r \rceil$ or $|\Phi_a| = \lfloor m/r \rfloor$, where $m = |\Phi|$ is the size of the multicast operation.

When the M-torus algorithm employs uniform partitioning, we term the result the *$\mathrm{M_u}$-torus multicast algorithm*. The $\mathrm{M_u}$-torus algorithm is parameterized by the number of partitions, r, which corresponds to one more than the number of destinations of each constituent message. During the last phase of the algorithm the multicast size, m, may be less than r. In this case, the $\mathcal{H}$-cycle is partitioned into m single-node partitions, resulting in a final message that is sent to $m - 1$ destinations rather than $r - 1$.

Since the number of destinations of each message produced by the $\mathrm{M_u}$-torus algorithm is $r - 1$ (except, perhaps, during the last phase), then the aggregate number of destinations reached will grow by a factor of r during each phase, leading to the following result: The $\mathrm{M_u}$-torus algorithm, with partitioning parameter r, applied to a multicast operation of size m, requires $\lceil \log_r m \rceil$ phases to complete the multicast.

In addition to the two specific methods covered above, many other linear partitionings are possible. As an example, one possible partitioning could combine sub-tori produced by dimensional partitioning whenever two or more adjacent sub-tori contain relatively few destination nodes, and could split single sub-tori that contain an over-abundance of destination nodes. This hybrid of dimensional and uniform partitioning would balance the purely network view imposed by dimensional partitioning with the destination-set view of uniform partitioning.

6.5 Multi-Phase Implementation Issues

In the M-torus multicast algorithm, the original source node sends a multicast message to a set of intermediate destination nodes, which, in turn, send the message on to other destination nodes. Depending on the number of phases used to perform the operation, these new destination nodes may also be required to relay the message, and so forth.

In some way, each intermediate destination node must be informed of the sequence of nodes to which it will send the message. The following are three mechanisms by which the dissemination of the multicast structure can be accomplished: (1) If a particular source node is to perform repeated multicasts to the same group of destination nodes, the necessary information can be distributed once at the time the communication group is formed. During subsequent multicasts to that group, the group identifier (GID) is attached to the message body so that each destination node can refer to the local information now associated with the group [6]; (2) The $\mathcal{H}$-cycle for which an intermediate destination node is subsequently responsible can be included in the multicast message body; and (3) The required information can be incorporated into the message header so that the router relays to each intermediate destination node the appropriate $\mathcal{H}$-cycle.

For succinct discussion, we introduce the following notation. Let Φ^u be the $\mathcal{H}$-cycle that is originally used as input when invoking the M-torus algorithm at node u. The $\mathcal{H}$-cycle Φ^u thus contains those destination nodes for which node u is responsible; that is, the destination nodes that receive the multicast message, either directly or indirectly, through the processor at node u. For example, if u is the original source node, then Φ^u represents the entire multicast operation.

As a practical matter, we note that when distributing the information in Φ^u to node u, the first element, which is the address of node u, need not be included. The notation $\tilde{\Phi}^u$ represents the value of Φ^u, after the address of node u has been removed; that is, $\tilde{\Phi}^u = \Phi^u - \{u\}$.

To support the third method, in which the message header contains information regarding the multicast structure, the header format shown in Figure 6.2 can be used.

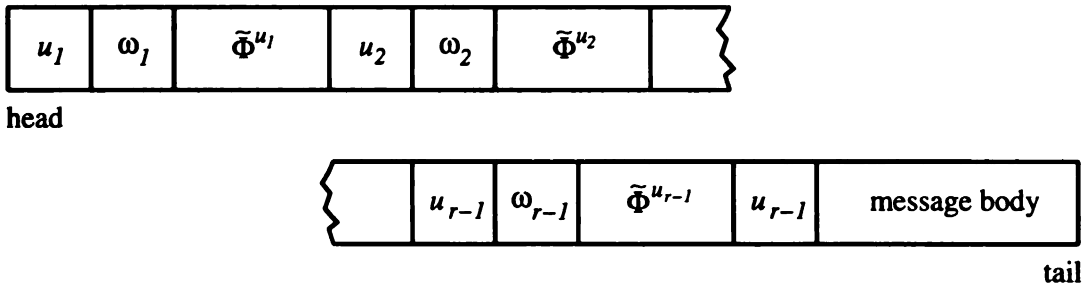


Figure 6.2. Compound message format

Figure 6.2 depicts a message that will be transmitted from source node u_0 to destination nodes $u_1, u_2, \ldots, u_{m-1}$, where node u_0 has partitioned the multicast problem represented by $\mathcal{H}$-cycle Φ into r sub-sequences $\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_{r-1}$, and where u_i is the first element of Φ_i $(0 \leq i \leq r-1)$. We term this structure a *compound message format*. Each destination node address, u_i, in the header of a compound message is followed by a count, $\omega_i = |\tilde{\Phi}^{u_i}|$, of the number of destination nodes for which u_i will in later phases be responsible, as well as the list, $\tilde{\Phi}^{u_i}$, of those node addresses. As in the standard multi-destination message header described in Section 5.6, the address of the last destination node, u_{r-1}, is duplicated to signify the end of the header. In this example, node u_0 may be the original source of the multicast operation, or it may be an intermediate destination node that has received the message and is now responsible for the destination nodes contained in Φ. In either case, the actions taken by node u_0 will be the same.

To process a compound message header, in addition to the activities required for a basic multi-destination message as shown in Figure 5.10, the router must perform two additional tasks: (1) When forwarding the message header to the next destination node, the included $\mathcal{H}$-cycles must also be forwarded; and (2) When the local node is a destination of the message, the $\mathcal{H}$-cycle immediately following the local node address must be forwarded to the local host.

After a compound message has been processed by the router at a node that is a destination of the message, the local host will have received, in addition to the message body, the $\mathcal{H}$-cycle needed as input when invoking the M-torus algorithm at the local node.

In the implementation of a system supporting path-based multicast, the choice between the above three methods of distributing the multicast structure depends on several factors. The first method, in which the structure information is distributed initially upon creation of a group of communicating nodes, is efficient only in cases where the same source and destinations will be involved in repeated multicasts. This method also requires that each destination node maintain a local group table in which information is stored for each group to which the node belongs.

If the multicast structure is added to the message body, as in the second method described above, one difficulty is that every destination node of a multi-destination message must receive the information intended for all of the other destination nodes. Without additional router support, the same message body must be delivered to each of the destination nodes of the message; therefore, there is no way to distinguish the structural information that is needed by a particular destination. Instead, the local host at each destination node must locate and use the correct $\mathcal{H}$-cycle from the sequence of $\mathcal{H}$-cycles received in the message body. An advantage of this method is that no additional hardware support is needed beyond that required for single-phase multicast.

Minor enhancements to the routing hardware, however, are required to support the compound message format. As stated above, the router must forward the correct $\mathcal{H}$-cycle, included in the message header, to the local host of a destination node; and must also forward all other $\mathcal{H}$-cycles to subsequent destination nodes. In return for this additional router capability each destination node will receive only the $\mathcal{H}$-cycle it requires in order to perform its portion of the multicast. The compound message format does add overhead in the case of simple multi-destination messages, which are used during the last phase of a multi-phase operation, and for single-phase methods, which may also be used on systems supporting multi-phase operations. When using the compound message format for a simple multi-destination message in which no multicast structure is being distributed, the $\mathcal{H}$-cycle for each destination node is null; however, in order for the router to process the message header, the associated $\mathcal{H}$-cycle size counts, all zero, must be included. Thus, the compound message format incurs an overhead of one additional header flit for each destination node when used for a simple (non-compound) message.

To efficiently support both compound and non-compound multi-destination message formats, it is possible to incorporate a message type indicator into the message header. This indicator would distinguish between the two types of multi-destination message formats, and could even identify a unicast message, so that the most efficient message format could be used for each type of message. However, in order to interpret the message type information and adjust the router functionality accordingly, even more extensive hardware support is required.

6.6 Performance Evaluation

In order to better understand the performance of the multicast algorithms presented in this chapter, a simulation study has been conducted in which these algorithms, as well as a unicast-based multicast algorithm, are examined. The system model for the simulation is the same as assumed throughout this chapter, that is, a k-ary n-dimension torus with unidirectional communication links and all-port architecture.

In order to evaluate the performance of the multicast methods through simulation, specific time values were chosen for the following message latencies. The software overhead at the message source and destination node are represented respectively by the message send latency, τ_S, and the message receive latency, τ_R. The combined send and receive latencies are referred to as the message startup latency. The time required for a message in the network to advance one flit is represented by the per-flit network latency, τ_n.

All simulations were performed for a 4096-node torus. Both 2D ($16 \times 16 \times 16$) and 3D (64×64) topologies were examined, as well as various message lengths, multicast sizes, and message latencies. To provide example cases for the simulation, each multicast set was produced by selecting, from all nodes in the system, the appropriate number of unique nodes under a uniform distribution model, using a random number generator. Statistics for each configuration are averaged over 400 trials.

Five specific multicast algorithms are simulated: the single-phase S-torus algorithm, the $\mathrm{M_d}$-torus algorithm, two instances of the $\mathrm{M_u}$-torus algorithm, and for comparison, a unicast-based multicast algorithm. To examine the effect of the partitioning parameter, r, on the $\mathrm{M_u}$-torus algorithm, two such parameter values have been chosen; they are $r = 8$ and $r = 64$. We refer to the corresponding algorithms as $\mathrm{M_u}$-torus(8) and $\mathrm{M_u}$-torus(64), respectively.

As a comparison to the path-based methods presented in this chapter, an efficient unicast-based multicast algorithm is also simulated. This algorithm uses the same minimum-path routing function, $\mathcal{R}_{UTPR}$, as do the path-based methods. The unicast-based algorithm is essentially the $\mathrm{M_u}$-torus algorithm with parameter value $r = 2$. To complete a multicast of size m, the algorithm therefore requires $\lceil \log_2 m \rceil$

phases, or communication steps, which has been shown in Chapter 4 to be optimal for unicast-based methods under the one-port model and is also the best known result for multicast in all-port torus architectures. Thus, the simulated unicast-based method is efficient and serves as a useful comparison to the studied path-based methods.

Figure 6.3 shows the performance of the various multicast methods on a 2D (64×64) torus with message latency values of $\tau_S = 95\mu sec$, $\tau_R = 75\mu sec$, and $\tau_n = 0.5\mu sec$. These values represent the relatively high message startup latencies associated with many of the current wormhole-routed computers. Both average and maximum multicast latencies are shown; the former being the elapsed time from the initiation of the multicast operation at the source node to the reception of the message at each destination processor, averaged over the destination nodes; and the later being the maximum time over all destinations of the multicast message. Results are shown for message lengths of 8, 512, and 16384 flits.

With this configuration, the results show clearly that all of the path-based methods perform better than the unicast-based operation, except for very small messages. Even for small messages, the $\mathrm{M_u}$-torus(8) algorithm performs much better than the unicast-based method. In general, as the message length is increased, methods that use fewer communication phases tend to perform better. This behavior is due to the pipelining of wormhole routing.

Among path-based algorithms, the method providing the best performance depends greatly on the message length. The S-torus algorithm performs very well for large messages, whereas the $\mathrm{M_u}$-torus(8) algorithm is better with short and medium message lengths.

For the results shown in Figure 6.4, lower message startup latencies were used to simulate the architecture of state-of-the-art MPCs. The respective latency values are $\tau_S = 10\mu sec$, $\tau_R = 8\mu sec$, and as before, $\tau_n = 0.5\mu sec$. Again, message sizes of 8, 512, and 16384 flits were simulated. Under this configuration, the unicast-based method exhibits performance that is nearly identical to the M_u-torus(8) algorithm for small messages, but still performs poorly, compared to the path-based methods, for medium and large messages. Again, with large messages, the S-torus algorithm shows the best performance.

Figures 6.5 and 6.6 show the results of simulating a 4096-node 3D ($16 \times 16 \times 16$) torus; the configurations are otherwise identical to those associated with Figures 6.3 and 6.4. Except for small messages, the performance of all algorithms is very similar to the corresponding 2D case. Compared to the unicast-based and M_u-torus(8) algorithms, the S-torus, M_d-torus, and M_u-torus(64) algorithms produce relatively few communication phases, and consequently, tend to generate individual messages that are addressed to a larger number of destination nodes. The performance of these latter three algorithms increases on the 3D torus, as compared to the 2D torus, due to the more dense interconnection network and resulting shorter path lengths of the 3D topology.

In Figure 6.7, the multicast latencies are plotted against the message length for a multicast size of 512 nodes. The results of using both the high and low message startup latencies described above are presented. The results show that the amount by which the algorithm is effected by an increase in the message size is directly related to the number of communication phases used by the algorithm. The S-torus algorithm, with only one phase, is least effected by the message length, while at the other extreme, the unicast-based method, requiring $\lceil \log_2 512 \rceil = 9$ phases, is most effected.

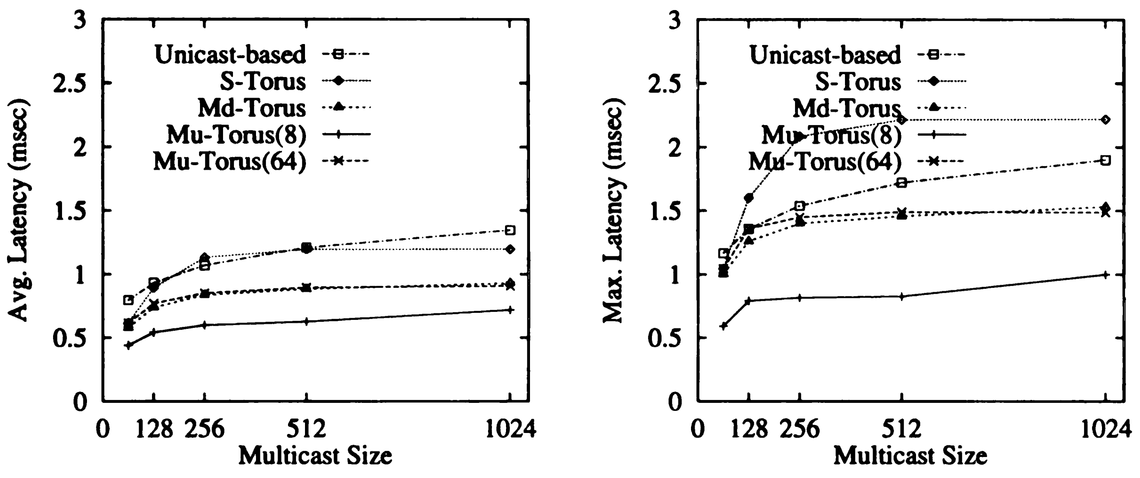
An important attribute of any communication operation is the resultant amount of network congestion. To investigate the amount of network traffic produced by the above multicast methods, the total number of *link visits* was recorded for each simulated multicast operation. Each link visit represents the use of one communication link by one message. Multiplying the number of link visits involved in a multicast operation by the message length and by the per-flit network latency, τ_n, produces a value equivalent to the summation, over all links in the network, of the total time during which the link is being used by the operation and is therefore unavailable.

Figure 6.8 depicts the resultant link usage for both 2D and 3D 4096-node torus networks, for various multicast sizes. As shown, the path-based algorithms require the use of fewer communication links than does the unicast-based method, especially with the 2D topology.

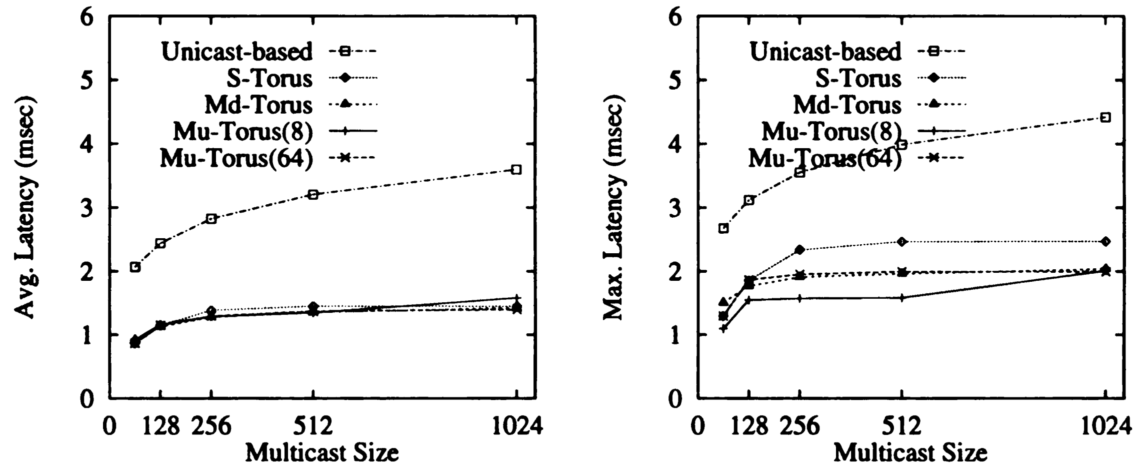
6.7 Conclusions

In this chapter, the area of multicast algorithms for unidirectional torus networks with IR capabilities has been studied. Specifically, several efficient path-based multicast algorithms were presented. The S-torus multicast algorithm uses a single multi-destination message to perform an arbitrary multicast operation. The S-torus algorithm was extended to the M-torus algorithm, a generalized multi-phase multicast algorithm, in which a combination of multi-destination messages is used to perform a multicast in one or more communication phases. Two specific instances of the M-torus algorithm, the M_d-torus and M_u-torus multicast algorithms, were presented. These algorithms produce contention-free multicast operations and are deadlock-free under all combinations of network traffic.

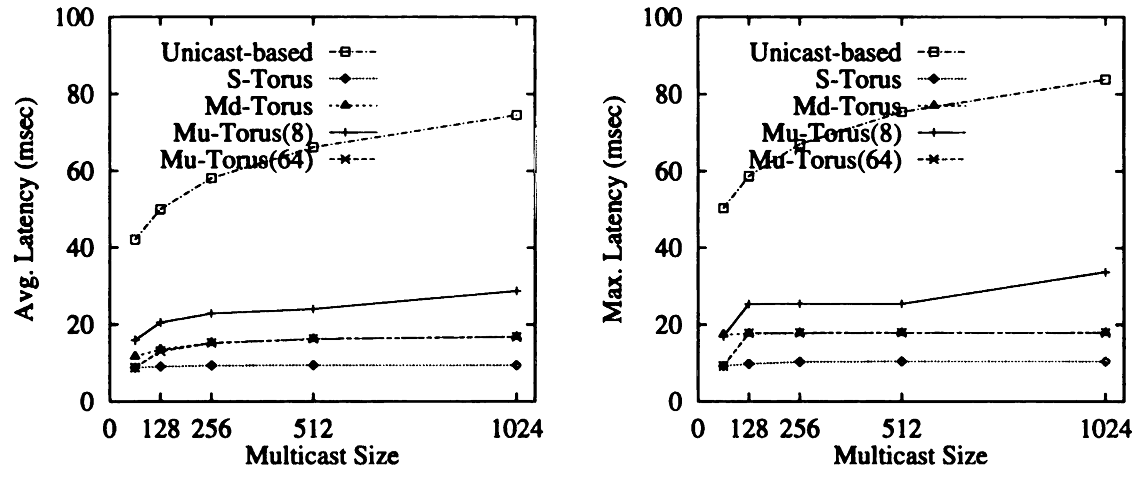
As a way to better gauge the real performance of the techniques presented in this chapter, a simulation study of the proposed multicast algorithms was conducted. The results of this study show that the path-based multicast algorithms presented in this chapter, together with the path-based routing method presented in Chapter 5, offer significant performance gains over unicast-based multicast techniques. By using the proposed algorithms, unidirectional torus systems with IR capability are able to perform efficient unicast and path-based multicast operations within a variety of environments.



(a) Message length: 8 flits

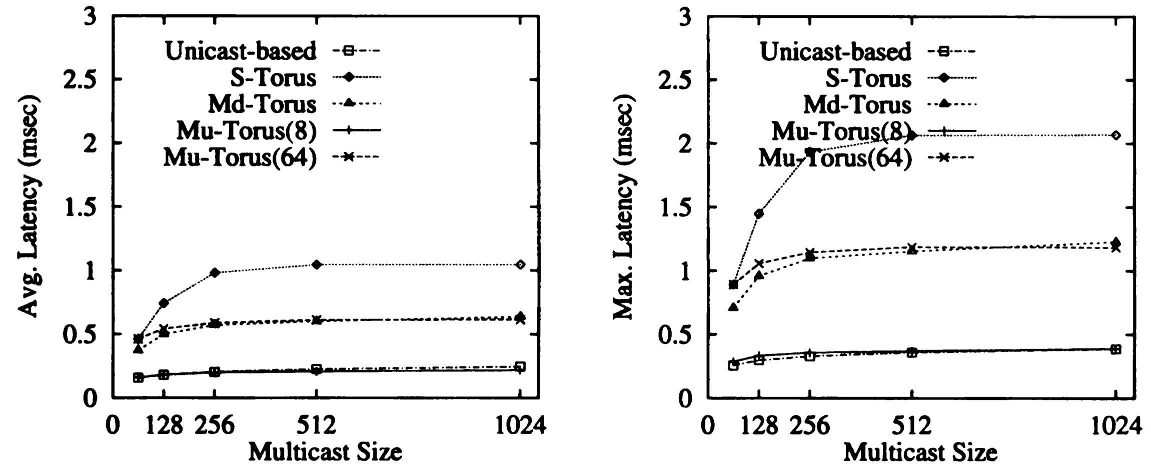


(b) Message length: 512 flits

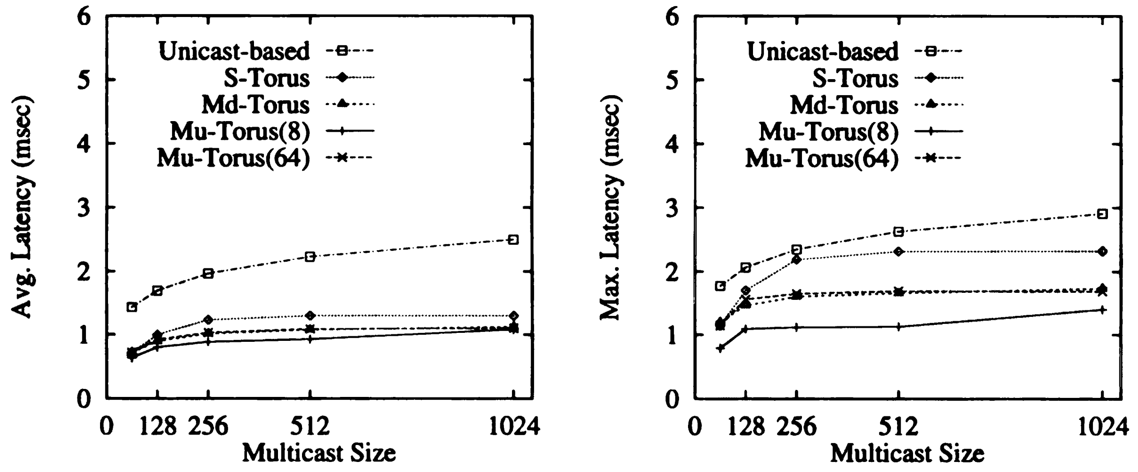


(c) Message length: 16384 flits

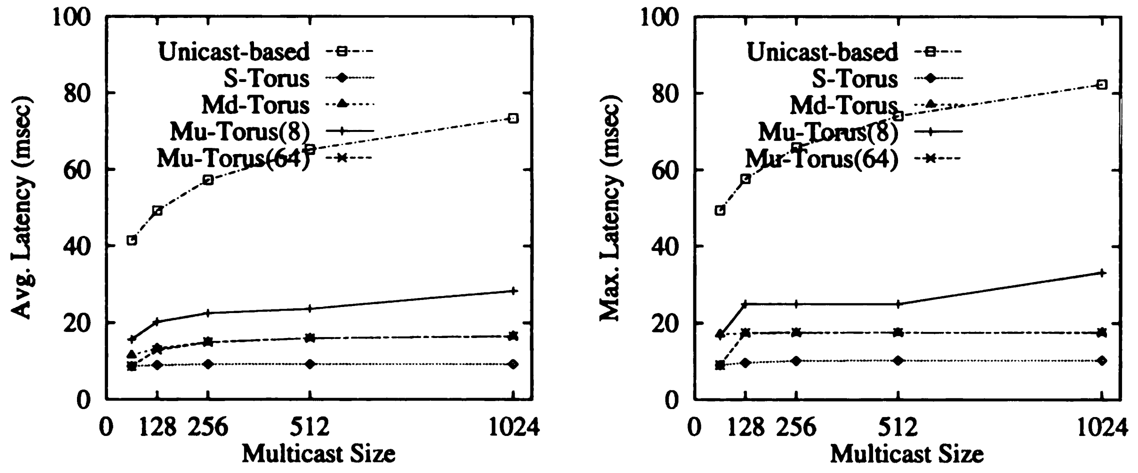
Figure 6.3. Multicast latency (4096-node 2D torus, high message startup latency)



(a) Message length: 8 flits

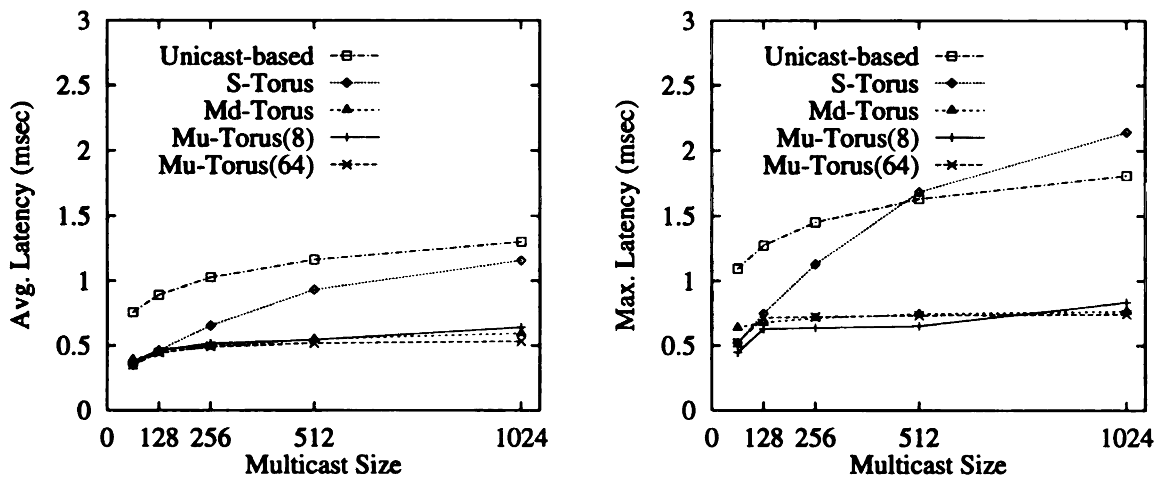


(b) Message length: 512 flits

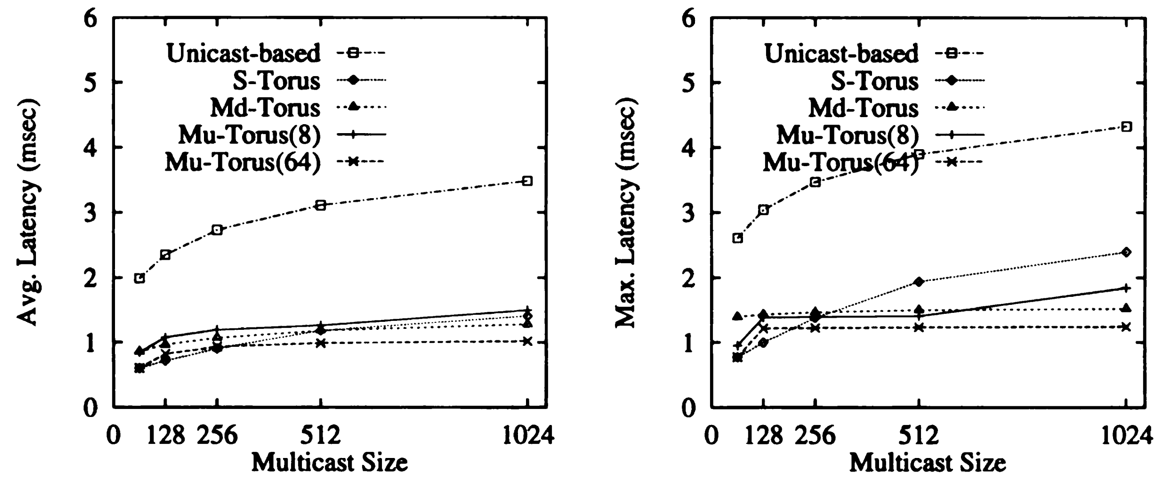


(c) Message length: 16384 flits

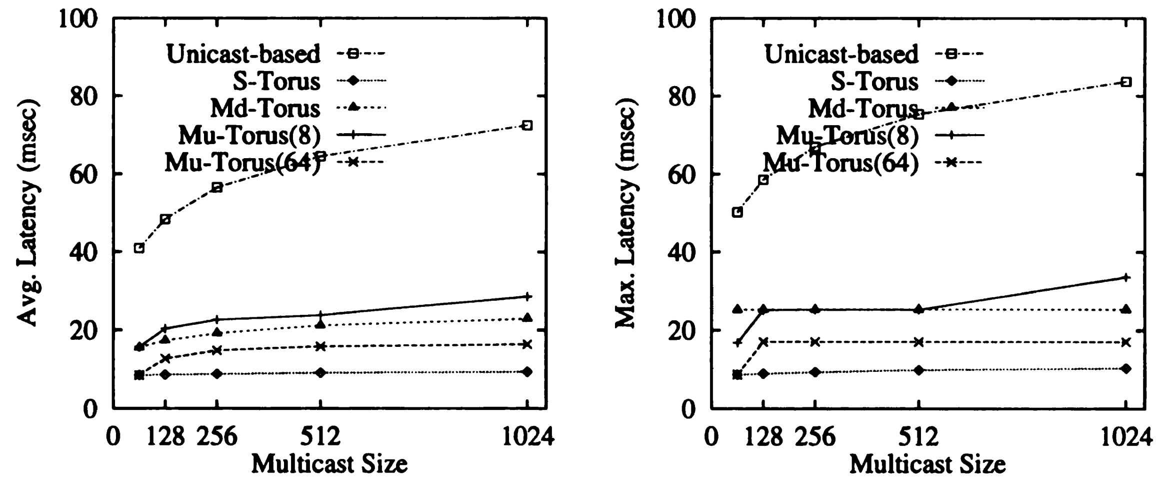
Figure 6.4. Multicast latency (4096-node 2D torus, low message startup latency)



(a) Message length: 8 flits

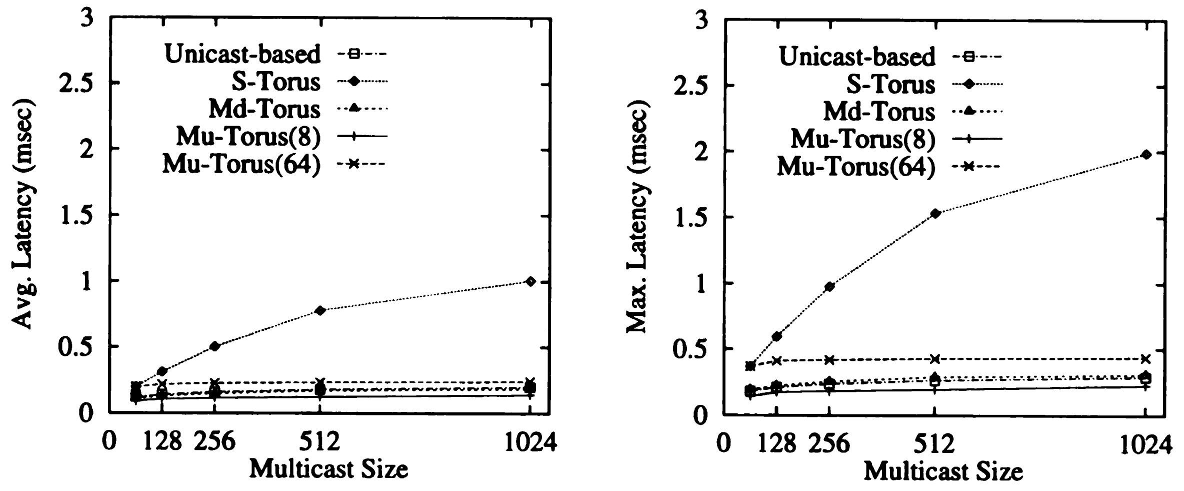


(b) Message length: 512 flits

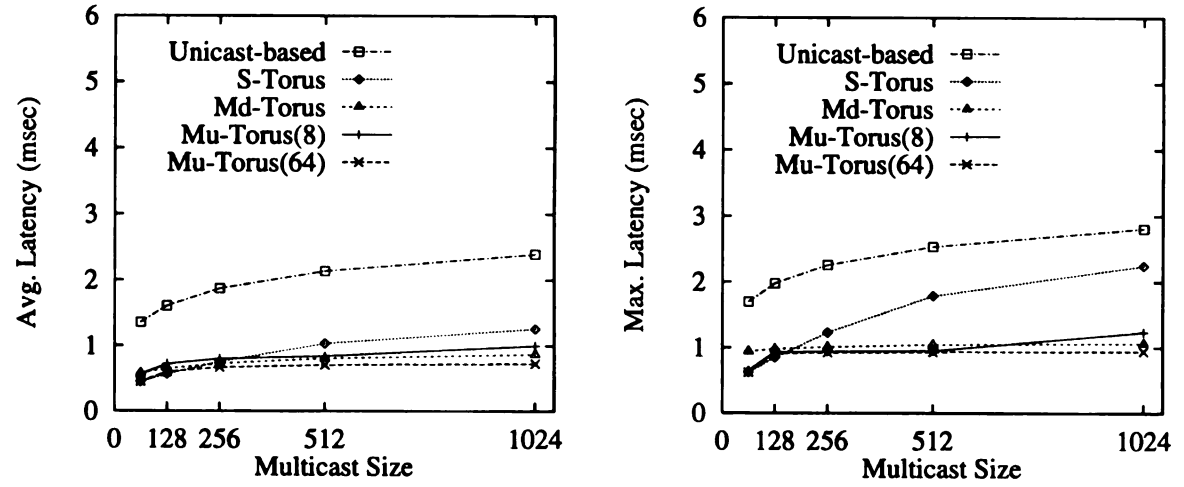


(c) Message length: 16384 flits

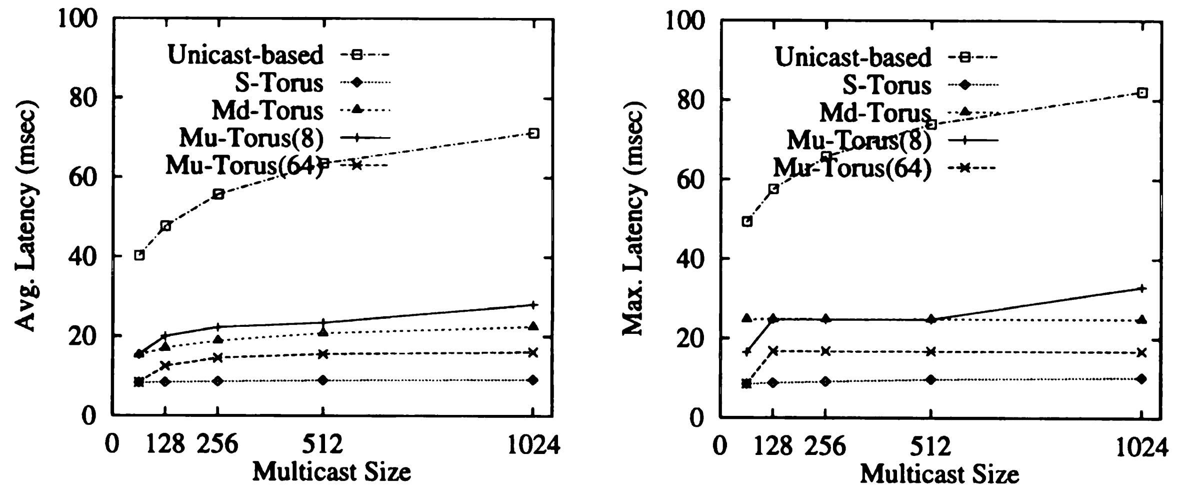
Figure 6.5. Multicast latency (4096-node 3D torus, high message startup latency)



(a) Message length: 8 flits

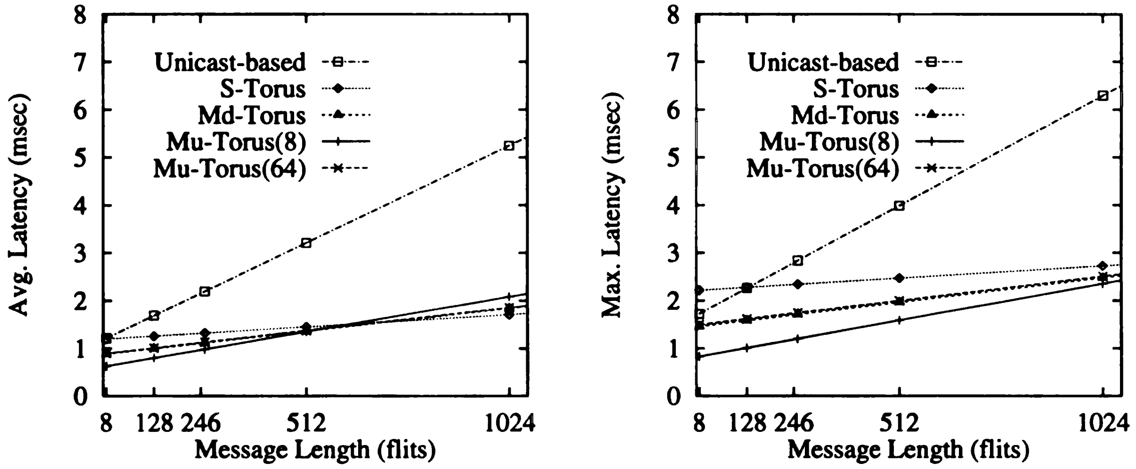


(b) Message length: 512 flits

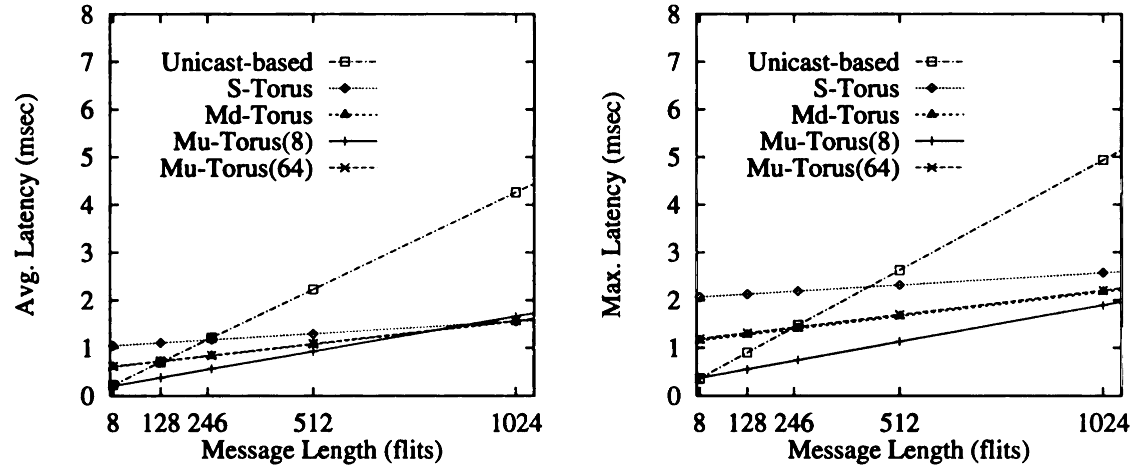


(c) Message length: 16384 flits

Figure 6.6. Multicast latency (4096-node 3D torus, low message startup latency)

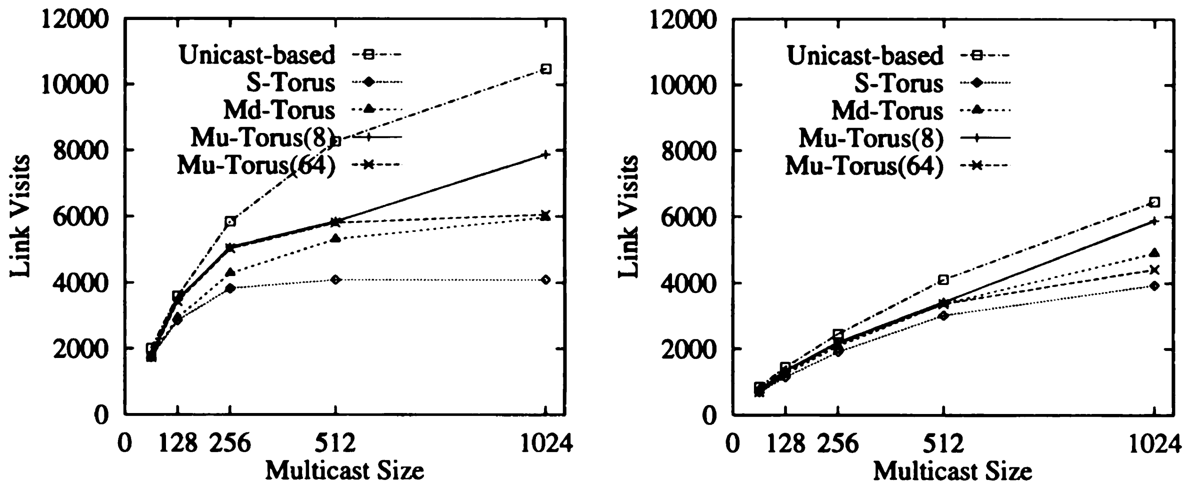


(a) High message startup latency



(b) Low message startup latency

Figure 6.7. Multicast latency (4096-node 2D torus, 512 node multicast size)



(a) 2D torus (b) 3D torus

Figure 6.8. Total link usage (4096-node tori)

CHAPTER 7

Conclusions

Efficient multicast communication is critical to the performance of new generation supercomputers that use massively parallel architectures. In this dissertation, we have presented research that focuses on three aspects of parallel communication architecture affecting multicast communication, namely (1) port model; (2) virtual communication channels; and (3) intermediate message reception. We have shown that the performance of multicast operations in current wormhole-routed parallel computers can be significantly improved by accounting for these three characteristics in the design of the operations.

In Chapter 3, we investigated the effects of all-port architectures on the performance of multicast communication in wormhole-routed hypercubes. It has been demonstrated why the U-cube multicast algorithm [19], which is optimal for one-port architectures, fails to take advantage of multiple ports when they are present in the system. New theoretical results regarding contention among messages in wormhole-routed hypercubes have been developed and used to design new multicast routing algorithms and to prove that these algorithms are contention-free. The algorithms were compared in terms of the number of steps required in each, their measured execution times when implemented on a relatively small-scale nCUBE-2, and their simulated execution times on larger hypercubes. The results indicate that significant performance improvement is possible when the multicast algorithm actively identifies and uses multiple ports in parallel.

In Chapter 4, we considered multicast communication in wormhole-routed torus networks, which require the use of virtual channels in order to provide deadlock-free message routing. The proposed U-torus algorithm applies to unidirectional and bidirectional tori of any dimension. The algorithm produces multicast trees in which the constituent unicast messages do not contend for the same channels, regardless of message length or startup latency. Moreover, the number of message passing steps required to multicast data to $m-1$ destinations is $\lceil \log_2 m \rceil$, which is optimal for one-port architectures. A simulation study was conducted which showed that the practical consideration of sharing of physical links by constituent messages transmitted on different virtual channels had little effect on performance.

In Chapters 5 and 6, we addressed architectures in which intermediate nodes on a message path are able to receive a copy of a message while simultaneously routing the message to subsequent destinations. In particular, in Chapter 5, by focusing on the torus topology, we investigated the interaction of intermediate reception capability with the use of virtual communication channels. We developed a routing method for unidirectional torus networks that not only supports efficient multi-destination messages, but also unicast communication. This routing method is deadlock-free for all combinations of multi-destination and unicast messages. In Chapter 6, this routing method was used to develop a family of path-based multicast algorithms. These algorithms are contention-free, and were shown through simulation to perform well in a wide variety of situations.

The research presented in this dissertation makes three primary contributions to the field of parallel computing: (1) the application of multi-port architectures to improve the performance of multicast communication in wormhole-routed parallel computers; (2) the use of virtual channels to provide efficient software-based multicast communication in wormhole-routed torus networks; and (3) the use of intermediate message reception to implement efficient path-based multicast communication in unidirectional wormhole-routed torus networks.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, pp. 62–76, Feb. 1993.

[2] J. Choi, J. J. Dongarra, and D. W. Walker, "PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," Tech. Rep. ORNL/TM-12252, Oak Ridge National Laboratory, Aug. 1993.

[3] J. Choi, J. J. Dongarra, and D. W. Walker, "Parallel matrix transpose algorithms on distributed memory concurrent computers," Tech. Rep. ORNL/TM-12309, Oak Ridge National Laboratory, Oct. 1993.

[4] J. Dongarra and R. A. van de Geijn, "Reduction to condensed form for the eigenvalue problem on distributed memory architectures," *Parallel Computing*, vol. 18, pp. 973–982, 1992.

[5] C. Trefftz, P. K. McKinley, T. Y. Li, and Z. Zeng, "A scalable eigenvalue solver for symmetric tridiagonal matrices," in *Proceedings of the Sixth SIAM Conference on Parallel Processing*, pp. 602–609, 1993.

[6] P. K. McKinley, H. Xu, E. Kalns, and L. M. Ni, "ComPaSS: Efficient communication services for scalable architectures," in *Proceedings of Supercomputing'92*, pp. 478–487, Nov. 1992.

[7] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, "ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers," in *Proceedings, Fourth Symposium on the Frontiers of Massively Parallel Computation*, pp. 120–127, IEEE Computer Society Press, 1992.

[8] H. Xu, P. K. McKinley, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 172–184, 1992.

[9] K. Li and R. Schaefer, "A hypercube shared virtual memory," in *Proc. of the 1989 International Conference on Parallel Processing*, vol. I, pp. 125–132, Aug. 1989.

[10] Message Passing Interface Forum, "Document for standard message-passing interface," Tech. Rep. CS-93-214, University of Tennessee, Nov. 1993.

[11] J. Bruck, R. Cypher, P. Elustondo, A. Ho, and C.-T. Ho, "A proposal for common group structures in a collective communication library," tech. rep., IBM Research Division, Almaden Research Center, San Jose, California.

[12] High Performance Fortran Forum, "Draft High Performance Fortran language specification." (version 1.0), May 1993.

[13] S. L. Johnsson and C.-T. Ho, "Algorithms for matrix transposition on boolean n-cube configured ensemble architectures," *SIAM Journal of Matrix Analysis and Applications*, vol. 9, pp. 419–454, July 1988.

[14] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, vol. C-38, pp. 1249–1268, Sept. 1989.

[15] D. W. Wall, *Mechanisms for Broadcast and Selective Broadcast.* PhD thesis, Department of Electrical Engineering and Computer Science, Stanford University, Stanford, California, 1980.

[16] A. Edelman, "Optimal matrix transposition and bit reversal on hypercubes: All-to-all personalized communication," *Journal of Parallel and Distributed Computing*, vol. 15, no. 11, pp. 328-331, 1991.

[17] D. P. Bertsekas, C. Özveren, G. D. Stamoulis, and J. N. Tsitsiklis, "Optimal communication algorithms for hypercubes," *Journal of Parallel and Distributed Computing*, vol. 15, no. 11, pp. 263–175, 1991.

[18] P. K. McKinley and J. W. S. Liu, "Multicast tree construction in bus-based networks," *Communications of the ACM*, vol. 33, pp. 29–42, Jan. 1990.

[19] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed networks," in *Proc. of the 1992 International Conference on Parallel Processing*, vol. II, pp. 10–19, Aug. 1992.

[20] D. F. Robinson, D. L. Judd, P. K. McKinley, and B. H. C. Cheng, "Efficient collective data distribution in all-port wormhole-routed hypercubes," in *Proceedings of Supercomputing'93*, pp. 792-801, Nov. 1993. Accepted to appear in the *Journal of Parallel and Distributed Computing.*

[21] D. F. Robinson, P. K. McKinley, and B. H. C. Cheng, "Optimal multicast communication in wormhole-routed torus networks," in *Proc. of the 1994 International Conference on Parallel Processing*, Aug. 1994. (accepted to appear).

[22] M. Barnett, D. G. Payne, and R. van de Geijn, "Optimal minimum spanning tree broadcasting in mesh-connected architectures." Technical Report.

[23] M. Barnett, D. Payne, and R. A. van de Geijn, "Broadcasting on meshes with worm-hole routing," Tech. Rep. TR-93-24, The University of Texas at Austin, Nov. 1993.

[24] C.-T. Ho and M.-Y. Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," in *Proc. of the 1994 International Conference on Parallel Processing*, Aug. 1994.

[25] P. K. McKinley and C. Trefftz, "Efficient broadcast in multi-port wormhole routed hypercubes," in *Proc. of the 1993 International Conference on Parallel Processing*, Aug. 1993.

[26] J.-Y. L. Park, S.-K. Lee, and H.-A. Choi, "Broadcasting in mesh and torus networks with circuit-switched communication." Technical Report, 1993.

[27] Y. Tsai and P. K. McKinley, "A dominating set model for broadcast in all-port wormhole-routed 2D mesh networks," Tech. Rep. MSU-CPS-ACS-23, Michigan State University, Department of Computer Science, East Lansing, Michigan, Sept. 1993.

[28] C.-T. Ho and M. T. Raghunath, "Efficient communication primitives on hypercubes," Tech. Rep. RJ 7932 (72915), IBM Almaden Research Center, Jan. 1991.

[29] Y. Lan, L. M. Ni, and A.-H. Esfahanian, "Distributed multi-destination routing in hypercube multiprocessors," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, pp. 631–639, Jan. 1988.

[30] X. Lin, P. K. McKinley, and L. M. Ni, "Performance evaluation of multicast wormhole routing in 2D-mesh multicomputers," in *International Conference on Parallel Processing*, vol. I, Architecture, pp. 435–442, Aug. 1991.

[31] D. Kim and S.-H. Kim, "$O(\log n)$ numerical algorithms on a mesh with wormhole routing," tech. rep., Pohang Institute of Science and Technology (POSTECH), Pohang, Korea, 1993.

[32] J.-S. Tseng and C.-T. King, "Efficient routing algorithms for torus networks." Technical Report, 1993.

[33] D. K. Panda and S. Singal, "Broadcasting in k-ary n-cube wormhole routed networks using path-based routing," Tech. Rep. TR36., Ohio State University, Sept. 1993.

[34] D. K. Panda and P. Prabhakaran, "Multicasting using multidestination-worms conforming to base routing schemes," Tech. Rep. TR37, Ohio State University, Sept. 1993.

[35] C.-T. Ho and M.-Y. Kao, "Optimal broadcast on hypercubes with wormhole and E-cube routings," in *International Conference on Parallel and Distributed Systems*, pp. 694–697, Dec. 1993.

[36] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells,

M. C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the connection machine CM-5," in *Proc. 4th ACM Symposium on Parallel Algorithms and Architectures*, pp. 272–285, June 1992.

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

[38] C. L. Seitz and W.-K. Su, "A family of routing and communication chips based on the Mosaic," in *Proceedings of the University of Washington Symposium on Integrated Systems*, 1993.

[39] W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Fyler, "The message-driven processor: A multicomputer processing node with efficient mechanisms," *IEEE Micro*, pp. 23–39, Apr. 1992.

[40] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWarp: An integrated solution to high-speed parallel computing," in *Proceedings of Supercomputing'88*, pp. 330–339, Nov. 1988.

[41] R. E. Kessler and J. L. Schwarzmeier, "CRAY T3D: A new dimension for Cray research," in *Proc. COMPCON*, pp. 176–182, Feb. 1993.

[42] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[43] NCUBE Company, *NCUBE 6400 Processor Manual*, 1990.

[44] B. Duzett and R. Buck, "An overview of the nCUBE 3 supercomputer," in *Proc. Frontiers'92: The 5th Symposium on the Frontiers of Massively Parallel Computation*, pp. 458–464, Oct. 1992.

[45] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, pp. 775–785, June 1990.

[46] C. L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, pp. 22–33, Jan. 1985.

[47] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W.-K. Su, "The architecture and programming of the Ametek Series 2010 multicomputer," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, vol. I, (Pasadena, CA), pp. 33–36, ACM, Jan. 1988.

[48] J. Duato, "On the design of deadlock-free adaptive routing algorithms for multicomputers: Theoretical aspects," in *Proc. Parallel Architectures and Languages Europe Conference (PARLE)*, pp. 234–243, 1991.

[49] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, pp. 2–12, Jan. 1991.

[50] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proc. of the 19th Annual International Symposium on Computer Architecture*, pp. 278–287, May 1992.

[51] P. Berman, L. Gravano, J. Sanz, and G. D. Pifarre, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," in *Proc. 4th ACM Symposium on Parallel Algorithms and Architectures*, pp. 3–12, June 1992.

[52] Z. Liu and H. Wu, "Performance evaluations of adaptive wormhole routing in 3D mesh networks," in *Proc. 26th Annual Simulation Symposium*, 1993.

[53] Z. Liu, J. Duato, and L.-E. Thorelli, "Grouping virtual channels for deadlock-free adaptive wormhole routing," in *Proc. Parallel Architectures and Languages Europe Conference (PARLE)*, pp. 254–265, June 1993.

[54] L. Schwiebert and D. N. Jayasimha, "Optimal fully adaptive wormhole routing for meshes," in *Proceedings of Supercomputing'93*, pp. 782–791, Nov. 1993.

[55] P. T. Gaughan and S. Yalamanchili, "A family of fault-tolerant routing protocols for direct multiprocessor networks." Technical Report.

[56] H. Sullivan and T. Brashkow, "A large scale homogeneous machine," *Proceedings 4th Annual Symposium in Computer Architecture*, pp. 105–124, 1977.

[57] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, May 1987.

[58] W. J. Dally, "Virtual channel flow control," *IEEE Transactions on Computers*, vol. 3, pp. 194–205, Mar. 1992.

[59] J. Duato, "A new theory of deadlock-free adaptive multicast routing in wormhole networks," in *Proc. of the 1993 IEEE Symposium on Parallel and Distributed Processing*, pp. 64–71, Dec. 1993.

[60] Y. Lan, A.-H. Esfahanian, and L. M. Ni, "Multicast in hypercube multiprocessors," *Journal of Parallel and Distributed Computing*, pp. 30–41, Jan. 1990.

[61] P. K. McKinley and C. Trefftz, "MultiSim: A tool for the study of large-scale multiprocessors," in *Proc. 1993 International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Networks (MASCOTS 93)*, pp. 57–62, Jan. 1993.