This is to certify that the

dissertation entitled

Modeling Collective Communication:
Design and Performance Evaluation

presented by

Yih-jia Tsai

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Computer Science

Major professor

Date 5/27/97

Modeling Collective Communication: Design and Performance Evaluation

By

Yih-jia Tsai

A Dissertation

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

DOCTOR OF PHILOSOPHY

Department of Computer Science

1997

Abstract

Modeling Collective Communication: Design and Performance Evaluation

By

Yih-jia Tsai

As advances in VLSI technology have enabled rapid improvements in commodity microprocessors, the same technology also has changed communication architectures dramatically. A communication subsystem is an integrated combination of both hardware and software. Efficient communication primitives can be achieved only through the careful exploitation of the underlying hardware architecture. Collective communication operations, which involve more than two participants, arise frequently in scientific applications. The primary objective of this work is to explore design and performance evaluation of collective communication operations. To this end, two models are proposed. The Extended Dominating Node (EDN) model is based on dominating sets in graph theory. The EDN model can be used to design collective operations in systems that support single-step multicast communication. We demonstrate the use of the EDN model on the design of collective communication algorithms for wormhole-routed massively parallel processors (MPPs). The second model, the τ-model, is a generalized communication cost model that facilitates the study and characterization of collective communication operations over a variety of networking technologies. We show that the τ-model can reveal relationships among algorithms that are not easily discernible through traditional methods.

To my family

ACKNOWLEDGMENTS

First of all, I would like to thank my thesis advisor, Dr. Philip K. McKinley, for his initial encourgement and inspiration on the subject that later led to this dissertation. The constant advice and guidance from him on the subject matter are enormous. I also would like to express great appreciation to all the members of my committee: Dr. Lionel M. Ni, whose insights and first class research program have always been an inspiring source for me; Dr. Abdol H. Esfahanian for stimulating my thinking in an initial course on graph theory; and Dr. T. Y. Li for his very useful discussion on the impact of communication performance to parallel scientific programs. The research projects that these faculty have conducted have served as wonderful example to me. It is a priviledge to be able to study and work in such an exciting environment.

In addition, much appreciation is due to Dr. Donald E. Weinshank for providing me with initial support to work at the Michigan State University, and to Dr. Edward N. Kashy for his support and discussions on the subject of teaching science in the University.

Finally, I would like to express my gratitude to members of the Communications Research Group for their inputs and feedbacks on my on-going research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

That which has been, it is that which shall be; and that which has been done is that which shall be done; and there is nothing new under the sun.

Ecclesiastes 1:9

1.1 Motivations

Communication and network efficiency play a key role in the pursuit of high performance scientific computation. Large-scale scientific applications are characterized by the demand for high computing power. Advancements in VLSI technology have increased raw computing power, while high-speed networks have offered an effective way to interconnect processors. Parallel computing provides one of the most economical ways to solve large scale problems. As the goal of "performance at all costs" has given way to economic considerations, however, fine tuning of underlying

hardware platforms and various software components is frequently used to improve performance of parallel applications at reasonable cost.

Communication is recognized by many researchers as an area in which such fine tuning can result in large performance gains. Communication operations can be generally classified into two broad categories, *point-to-point* and *collective*, according to the number of parties involved. The performance of point-to-point communication depends heavily on the underlying network bandwidth, network interface, communication pattern, message size, channel contention, protocol processing, and the current system load. These factors also play an important role in determining the performance of collective communication. In addition, the adverse effects of resource contention and message serialization are more important to collective operations than to point-to-point operations, due to the presence of multiple messages in the network concurrently.

During the past decade, communication operations have been extensively studied in the context of parallel computing environments. The consolidation of the Message Passing Interface (MPI) standard [131] has spawned a wave of implementations of various message-passing libraries for different architectures. The importance and usefulness of collective operations is evidenced by extensive discussion on this topic within the MPI Forum.

1.2 Thesis Statement

Several features of network interfaces and network architectures are critical to the performance of collective communication operations. We contend that a performance cost model, capable of accounting for these features can be used in the design of new algorithms for collective operations that outperform existing algorithms across a variety of parallel computing platforms. Moreover, such models can reveal relationships among algorithms and architectures that are not easily discernible through traditional methods.

1.3 Research Contributions

This research begins with the exploration of communication locality, and yields the *Extended Dominating Node* (EDN) model. This model has been used in the design of several collective operations in parallel computers [142, 143, 144, 145, 146, 147]. The results from these studies are used in the development of a general communication cost model, the τ-model, that covers collective operations and can accommodate different networking technologies. The contributions of this dissertation can be summarized as follows:

- The research has produced an abstract model for designing efficient communication operations for MPPs with dedicated wormhole-routed networks.
- Associated with the abstract model is a cost model to account for the network latency of communication operations. The cost model can be used to classify

and characterize communication algorithms, and thus provide a unified view of all communication operations. Moreover, the complexity of communication algorithms can be categorized in a more meaningful way.

- The generalized cost model can serve as a basis to characterize the effects of different architectural supports for communication operations across various platforms, including both MPPs and workstation clusters. By characterizing communication operations, we can gain a better understanding of the expected performance of parallel programs.

1.4 Dissertation Organization

The remainder of this dissertation is organized into the following chapters. Chapter 2 provides background and related work for this research. Algorithms and on-going research in collective communication primitives for both wormhole-routed MPPs and workstation clusters are reviewed, and several communication cost models used in the parallel processing community are discussed. The Extended Dominating Node model is described in Chapter 3. This model extends the dominating set concept in graph theory to support the design of collective communication primitives in wormhole-routed networks. Continued in Chapter 3 and in Chapter 4 are demonstrations of the use of the new model to design collective operations for two network topologies, the mesh and torus, respectively. Chapter 5 first describes the τ-model in general and its associated τ plot. Subsequently, we demonstrate the use of the τ-model to analyze communication operations in wormhole switching net-

works and network of workstations environments. Finally, Chapter 6 summarizes the dissertation.

Chapter 2

Background and Related Work

Where ... the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have 1,000 vacuum tubes and perhaps weigh just 1-1/2 tons.

Popular Mechanics, March 1949, p. 258

2.1 Introduction

The rapid advancements in VLSI technology and breakthroughs in the manufacturing industry have been major sources of innovation in parallel computation structures and networking architectures. From the early processor arrays, vector machines, and multiprocessors, the field of experimental parallel architectures has been very active. As man's quest for computing power to solve larger and finer-grained problems is unabated, so is the demand for economical solutions to satisfy this need. With the scaling of new technology, both in size and speed, and the vast

market for general-purpose computing chips, microprocessor-based systems have emerged as the most viable candidates for providing enormous computing power at an acceptable cost. Users now have at their fingertips processing power that was unimaginable fifty years ago, when the first electronic computer was invented.

The design of fast, powerful, and economical single chip microprocessors will be ultimately limited by physics, specifically, the heat transfer capability of silicon chips and the speed of light. One way to address these limitations is to interconnect multiple processors and program them to solve a problem cooperatively. Parallel processing is not a new idea. As early as the dawn of the computer, there were experiments connecting together these "electronic brains," and research projects in the 1960s and 1970s (Illiac-IV, NASA MPP, Cmmp, and many others) laid the foundation for parallel processing. However, it was Thinking Machines Corporation in the mid-1980s, with the Connection Machine, that symbolized the economical success of massive parallelism.

In recent years, however, the cost of silicon chip manufacturing equipment has risen so high that, without a commodity market, no such company can withstand the high risks and investment. The result is that future generations of microprocessors will be increasingly targeted at the mass market. Parallel computation architectures are already successfully competing with supercomputer architectures in the high performance computing market, and will likely continue to gain ground as inexpensive and powerful microprocessors are equipped with such features.

The introduction of CM-5 in 1991, which adopted a commodity microprocessor (SPARC), marked the end of the first generation of massively parallel computers

and a victory for off-the-shelf microprocessors over custom-made vector processors. Although the CM-5 used special-purpose hardware for communications, experimental microprocessor architectures are already incorporating features that can be used to better support communication operations [79]. With the widespread use of the Internet and network navigating applications, future microprocessor architectures are likely to be even more communication-oriented. The mass market will be the final judge. These forces have already resulted in the new trend of joining commodity workstations with off-the-shelf communication networks to perform parallel computations. The economics and ubiquity of these so-called *workstation clusters* may cause us to rephrase the previous technical jargon from "massively parallel machines" to "parallel machines for the masses."

2.2 Parallel Computation Structures

In retrospect, the introduction of the Intel 4004 single chip microprocessor [81] in 1972 opened a new era in computing. Since then, the pace of technological advancement has been so fast that single chip microprocessors are competitive with the vector machines. Such technological advancement has promised "to bring the power and flexibility of a dedicated general-purpose computer at low cost" [81]. In the early 1970s, Gorden Moore predicted that transistor count will be doubled every 18 months, since referred to as "Moore's law." Moore's prediction was based on his observation of tremendous potential advances in microprocessor manufacturing technology. These advances have been realized and the implications have been even

more far-reaching than predicted by Moore. With the size of transistors decreasing and the cost dropping, additional functionality, and even the whole processing unit, can be cheaply integrated into a single chip. As early as 1980, Patterson proposed a single-chip architecture that could be used as a node in a parallel computer [116]. The revolutionary development of the RISC architecture [115, 117] set the stage for more rapid performance improvement. Specifically, microprocessor performance has increased steadily at the rate of over 50 percent per year since the 1980s [70]. As mentioned earlier, parallel processing can be traced back to the origins of the computer itself: the ENIAC was equipped with multiple processing units for redundancy. Parallel computation architectures span a wide spectrum, from fine-grained bit-serial SIMD systems to multiprocessor vector machines. The Connection Machine marks one end of the spectrum, and the Cray X/MP marks the other end. Both architectures use custom-made processors, whose cost rises more rapidly than the computing power per processor. In the middle of this spectrum is the medium-grained multicomputer equipped with off-the-shelf microprocessors. Beginning with the Caltech Cosmic Cube project [129], several commercial multicomputers soon entered the market [11, 109, 120]. Evidence of the success of commercial microprocessors in parallel processing is seen in the computation architectures of the TMC CM-5, Intel TOUCHSTONE, Intel Paragon, Meiko CS-2, and IBM SP-1/SP-2. The trend toward using a commodity network is also clear. Examples include Myrinet [27], Asynchronous Transfer Mode (ATM) [114], and Fast Ethernet [84]. The careful design and integration of the network architecture and the computation architecture is a major issue in parallel systems. As the communi-

cation architecture of workstations improves, it is very likely that the dividing line between multicomputers and workstation clusters will eventually vanish. We can expect that the widespread use of commodity high-speed networks, combined with high performance commodity microprocessors, will continue to make the workstation cluster a popular parallel platform.

2.3 Communication Paradigms

The programming model used in a parallel system defines the tradeoff between exposing the underlying hardware platform to the programmer and preserving application portability and extensibility. Parallel programs have been long regarded as difficult to develop and even more difficult to debug. One possible reason is that programmers usually have a hard time visualizing the interaction among concurrent entities.

The classical dichotomy of parallel programming style has been shared memory versus explicit message passing. The shared memory paradigm can be supported in software, hardware, or both. Hardware-supported shared memory can be based on either a bus-based network or a multistage interconnection network. However, neither approach scales to a large number of nodes due to contention. Physically distributed, logically shared architectures must rely on underlying message-passing operations to move data among nodes. In one approach, the operating system supports data distribution and coherence protocols, and maintains a single virtual shared memory space for the user applications. This method is used in the MIT

Alewife [2, 3, 4], Stanford FLASH [68, 90], Princeton SHRIMP I, II [26], and Wisconsin Wind Tunnel [39]. In another approach, programming language constructs support the shared memory paradigm. The compiler generates communication stubs to support data distribution and sharing. Examples include Dataparallel C [67] and HPF [71].

Explicit message passing is often blamed for the difficulty to program and maintain parallel applications. Yet most of today's commercial multicomputers support this model, in part because it is more closely tied to the machine hardware and gives more control to the programmer. By exposing the communication hardware, this method relies on the programmer to prevent deadlocks in the parallel program. Of course, message passing can also serve as an underlying mechanism to support the shared memory programming paradigm.

No matter which upper level programming paradigm is used, all communication operations deal with movement and manipulation of data. Communication operations can be classified as either *point-to-point* or *collective*, depending on the number of parties involved. Numerous communications libraries have been developed in the past ten years by commercial and research organizations. The recently published Message Passing Interface (MPI) standard [131] is the result of an effort by both industry and the research community to improve the portability of parallel programs among different platforms. MPI defines the interfaces and semantics of both point-to-point and collective operations. The software layer that implements these operations is typically packaged as a library. Calls to these routines can be placed in application code developed by the user or inserted by the compiler. The

performance of communication operations has direct impact on the performance of the applications that use them. Carefully designed communication operations can increase program portability and efficiency. Portability implies that applications can run on different platforms with little or no modification, and efficiency means that no matter which platform is used, the communication operations can take advantage of the underlying architecture to achieve better performance.

2.4 Collective Communication Operations

A communication operation may involve exactly two, or more than two, parties. When more than two parties are involved, the communication operation is called collective. As defined in the MPI standard, the participating entity in these communications operation is a *process*, and the processes involved in a collective operation are defined as a *process group*. The efficient implementation of collective communication primitives can enhance the performance of programs that use them. According to the direction of data movements, we can classify collective communication operations as point-to-multipoint, multipoint-to-multipoint, and multipoint-to-point.

A point-to-multipoint operation is used to distribute one or more data items to a set of receiving processes. In a *broadcast* operation, the same data item is delivered to every receiving process. Another point-to-multipoint operation is called *scatter*, in which a different data item is distributed to each group member. The data flow for broadcast and scatter operations are depicted in Figures 2.1(a) and 2.1(b),

respectively.

If the same point-to-multipoint operation is performed by all members of a group, the result is a multipoint-to-multipoint collective operation. If all members perform a broadcast, then the resultant operation is called *all-gather*, or *gossiping*. When all members perform scatter operations simultaneously, it is called *complete exchange*, or *all-to-all personalized communication*. Depicted in Figure 2.1(c), and 2.1(d), respectively, are all-gather and all-to-all personalized communication operations.

In addition to data movements, computations can be performed on the data. The global computation can be as simple as a logical operation or as complicated as a user-defined function. If the resultant data resides at a single node, then the operation is multipoint-to-point. Depending on the operation performed over the data set, several different collective operations can be defined. As shown in Figure 2.1(e), *gather* refers to pure data movement from all members to a single member; concatenation is the operation performed on the data. In contrast, *combine* means that in addition to data movement, a logical or arithmetic operation is performed on the data set; the combine operation is depicted in Figure 2.1(f)

Multipoint-to-multipoint operations also come in several flavors. If the result of a combine operation is made available to all members, then the operation is called *global reduction*; see Figure 2.1(g). A special case, when the global operation synchronizes all participating members, is called *barrier synchronization*. Another special form of multipoint-to-multipoint operation is the *prefix-scan*, or simply *scan*, where the result at each node depends only on the contents of all its

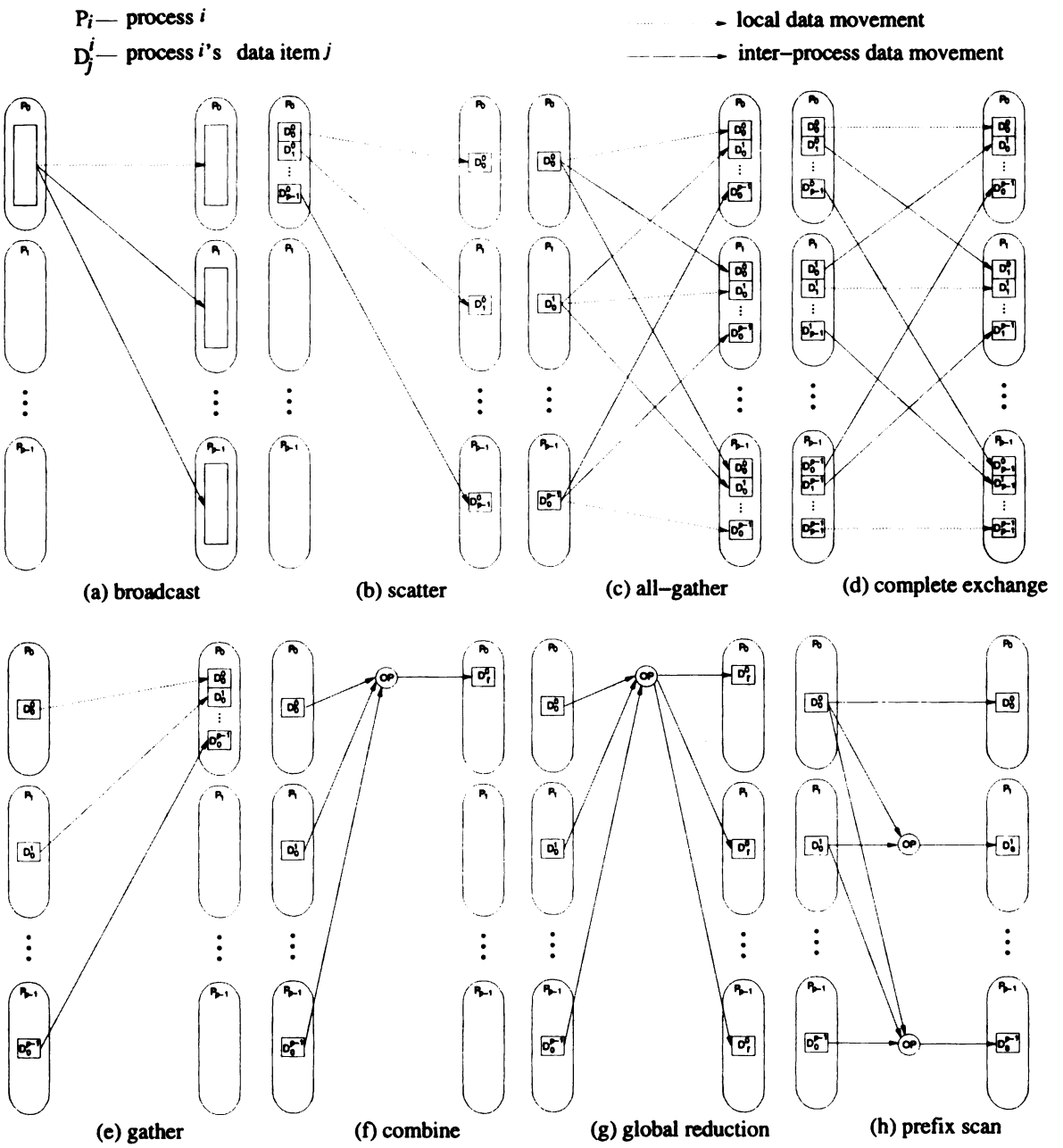


Figure 2.1: Collective operations.

predecessors in rank order (Figure 2.1(h)).

All the above collective operations are included in the Message Passing Interface (MPI) standard [104, 131], which defines a rich collection of both point-to-point and collective communication primitives. Key issues discussed in the MPI standard with regard to implementing these interfaces include sender buffering, receiver buffering, synchronization, data type, process group, communication context, and virtual topology. Most of these concepts are borrowed from existing message-passing libraries.

The importance of collective communication can be seen from the large number of pre-MPI communication libraries that include such operations. Vendors of parallel computers typically provide their own message passing libraries that are highly optimized with regard to their own architecture. Notable examples include IBM's EUI (External User Interface) [12], Thinking Machine's CMMD [139, 149], Intel's NX [119], and NCube's Vertex [126]. In the parallel processing research community, PVM [59] is perhaps the most widely used system for parallel computing over networks of workstations. Other prominent libraries include Argonne m4 descendants, p4[35, 36], PARMACS [37, 69], and TCGMSG, as well as PICL [60, 61] from Oak Ridge National Lab, Zipcode [130] from Caltech, Yale's Linda [6, 38], BLACS (Basic Linear Algebra Communication Subprograms), which is the communication library for ScaLaPACK [54], CHIMP [29] from University of Edinburgh, and LAM [33, 34] from Ohio State University. The NXLib, implemented by Lamberts et al. [91, 132, 133], is designed to emulate the Paragon message-passing environment on clusters of workstations.

All the above libraries implement some form of collective communication operations. From Table 2.1, we can see how certain libraries differ from the proposed MPI standard, although these libraries are now merging into the MPI mainstream. As indicated in the Table 2.1, most of the pre-MPI libraries do not have very extensive communication interfaces compared to MPI. Moreover, the workstation cluster version of most of these libraries is based on the point-to-point TCP/IP protocol suite. Clearly, careful implementation and fine tuning according to the system characteristics is necessary to achieve high performance, especially in high-speed networks where the overhead of conventional protocol stacks is especially detrimented to performance. Researchers such as von Eicken et al. [152] have used the active message approach [153] to reduce the software overhead in ATM networks, and Pakin et al. [108] have developed a fast messaging layer for Myrinet [27], a high-speed local area network.

2.5 Wormhole-Routed MPP Network Architectures

Given the importance of collective communication, the design and analysis of such operations on various parallel architectures has been a very active research area for the past decade [101]. Most early studies concentrated on the hypercube topology. First generation hypercubes, such as the Cosmic Cube, nCube/ten, and iPSC, all exhibited large communication latencies, on the order of milliseconds. This

	MPI	p4 1.4	PARMACS 6.0	TCGMSG	PICL	Zipcode	PVM 3.3
pt-to-pt	MPI_Send()	p4_send()	pm_send()			{1,y,z}_send() g{1,2,3}_send()	pvm_send()
	MPI_Bsend()						
	MPI_Ssend()	p4_sendr() p4_sendbr() p4_sendrx() p4_sendbrx()	pm_sendr()	p4_sendr()			
	MPI_Rsend()						
	MPI_Isend()						
	MPI_Ibsend()						
	MPI_Issend()						
	MPI_Irsend()						
	MPI_Recv()	p4_recv()	pm_recv() pm_recvr()			{1,y,z}_recvb() g{1,2,3}_recvb()	
	MPI_Irecv()					{1,y,z}_recv() g{1,2,3}_recv()	
	MPI_Sendrecv()						
	MPI_Sendrecv_replace()						
pt-to-mpt	MPI_Bcast()	p4_broadcast() p4_broadcastx()		BRDCST_()	bcast0() bcast1()	{1,z}_fanout() g{1,2,3}_fanout()	pvm_mcast() pvm_bcast()
	MPI_Scatter()						
	MPI_Scatterv()						
mpt-to-pt	MPI_Reduce()				gand0() gmax0() gmin0() gor0() gprod0() gsum0() gxor0()	{1,z}_fanin() g{1,2,3}_fanin()	pvm_reduce()
	MPI_Gather()						
mpt-to-mpt	MPI_Allgather()						
	MPI_Allgatherv()						
	MPI_Alltoall()						
	MPI_Alltoallv()						
	MPI_Allreduce()	p4_global_op()		DGOP_() IGOP_()		{1,z}_combine() {1,z}_pack_combine() g{1,2,3}_combine() g{1,2,3}_pack_combine()	
	MPI_Reduce_scatter()						
	MPI_Barrier()	p4_global_barrier()	pm_barrier()	SYNCH_()	barrier0()	{1,z}_sync()	pvm_barrier()
	MPI_Scan()					{1,z}_prefix() g{1,2,3}_prefix()	

Table 2.1: Comparison of some message passing libraries with MPI standard

characteristic was due to the inefficiency of communication software and the use of store-and-forward switching, which requires intermediate nodes on the message path to receive incoming data and then forward it to the next node in the path. As a result, the communication latency was proportional to the number of communication links traversed by the message. Suppose the software setup overheads are t_s for sending a message and t_r for receiving a message, and that the transmission delay per node per unit message is t_n, then the overall delay for sending an L-unit message h hops across the network in store-and-forward switching is $h(t_s + L * t_n + t_r)$.

With the advent of wormhole routing, in which a message is broken down into

small units and pipelined through the network, the overall latency of sending an L-unit message h hops across the network is reduced to approximately $t_s + h * L * t_n + t_r$. The advantage of this technique is twofold. First, communication latency is relatively distance-insensitive, a characteristic that has been demonstrated in experimental studies [107]. Second, the routing automata can be implemented in specialized hardware, called a *wormhole router*, relieving the processor of the burden of message buffering and routing. As a result, communication latency is reduced to the microsecond range.

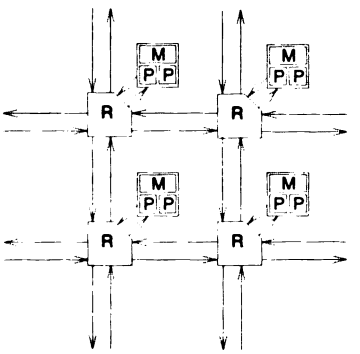


Figure 2.2: System architecture.

In a wormhole-routed system, the interconnection of routers form the topology of the multicomputer. Each processing node is attached to the router as illustrated in Figure 2.2. The link between routers is called *channel* and the link between router and processing node is called *port*. The number of channels between adjacent routers determines how many messages can be in transit simultaneously, while the number of ports is an indication of the communication capacity between the node and the network. The major difference between single-port and multi-port wormhole routers in functionality is the number of messages that can be sent (re-

ceived) by the computing node simultaneously. Due to this capability, a multi-port router can be important to the design and performance of collective communication operations, which often involve the transmission (reception) of multiple messages at a single node. Depicted in Figure 2.3(a) and (b) are different port models of a wormhole router. In Figure 2.4 is a detailed block diagram of the internal structure of a one-port 2D mesh wormhole router, which is composed of two 1D mesh routers. For comparison, Figure 2.5 shows the structure of an all-port 2D mesh wormhole router. The main element of this router is a crossbar.

Wormhole routing is the predominant switching technique used in most second generation multicomputers. Some systems, exemplified by the nCube/2, adopted the topology of their ancestors, namely, the hypercube. However, analysis by Dally [45] on the performance of wormhole routing under the assumption of constant bisection width for different k-ary n-cube topologies, showed a clear cost/performance advantage for low-dimensional topologies, such as meshes and tori. Consequently, Intel produced the DELTA and Paragon systems, which use a 2D mesh, and Cray designed the T3D and T3E, which use a 3D torus topology.

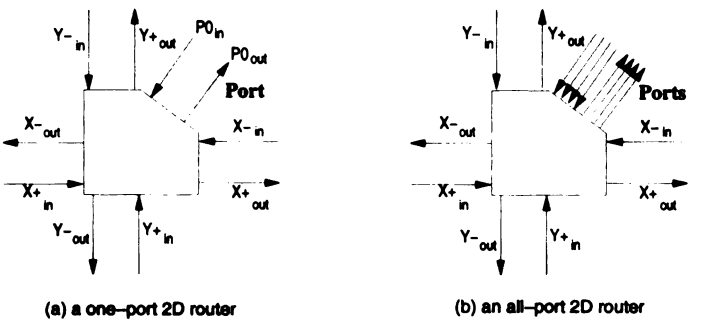


Figure 2.3: Wormhole routers.

Over the last several years, a number of researchers have sought to exploit the

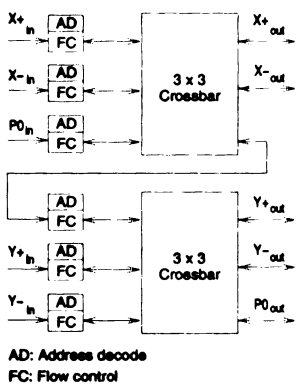


Figure 2.4: One-port 2D router.

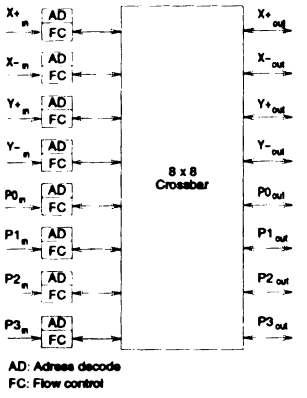


Figure 2.5: All-port 2D router.

distance-insensitivity of wormhole routing in the design of collective communication algorithms. A large fraction of that research has concentrated on broadcast and multicast for wormhole-routed systems. One-port mesh multicast (U-mesh algorithm) is given in [103], and a one-port mesh broadcast algorithm is presented in [17]. The former is a tree-based algorithm that can be used in both hypercubes and meshes. The latter uses a scatter-collect approach, in which the original message is split into pieces that one distributed to all nodes and later collected by the receiving nodes. Multicast for a one-port torus is described in [124]; similar to the U-mesh algorithm in one-port meshes [103], the U-torus algorithm is tree based. In addition to the tree-based approach to multicast and broadcast, a different method is to partition the message into pieces and send them through different routes in a pipelined fashion. This approach has been explored for one-port hypercubes [73] and one-port meshes [156].

All the communication algorithms mentioned above assume a one-port network architecture. In addition, software unicast algorithms have been designed for multiport systems. Communication algorithms based on the multiport assumption have

been studied quite extensively for hypercubes [21, 76, 85, 99]. Tree-based multiport hypercube multicast algorithms are presented in [76, 99, 123] and one-to-all personalized communication (scatter) is described in [85]. Broadcasting algorithms, based on our EDN model, for multiport meshes [147] and tori [146] comprise our preliminary studies and are described in detail in Chapter 3 and 4.

Another distinctive hardware feature that has been used to construct efficient communication operations is a path-like approach for multicast. This method relies on a special feature of the router, in which the message can be duplicated so that the node connecting to the router can receive a copy while the message is progressing toward subsequent destinations. The entire message visits all the destination nodes in a pipelined fashion. This path-based approach was first explored in [95, 96, 97]. Later, this concept was extended from single-phase algorithms to multi-phase collective algorithms for meshes [148], hypercubes [74], and tori [122]. The EDN model can also be used in the design of multi-phase path-based algorithms.

While most of the above mentioned communication operations focus on multicast and broadcast, other collective communication primitives have been developed specifically for wormhole networks. Examples include barrier synchronization [66, 110, 159] and global combine [151]. Several different approaches have been proposed for all-to-all exchange in meshes, including binary exchange, interleaved binary exchange, direct exchange [128], quadrant exchange [28], modified quadrant exchange [136], pairwise exchange, and indirect pairwise exchange [137, 138]. All these algorithms assume a one-port architecture. We [144] have used the EDN model to design global combine, gather, and matrix transposition algorithms for

multiport 2D meshes; details are given in Chapter 4.

These extensive research efforts have shown that, to design efficient collective communication operations, the characteristics of the underlying network should be carefully taken into consideration. This is particularly true in the parallel processing community, where communication latency is the major concern. In early studies of collective communication, the store-and-forward routing assumption is predominant, although both one-port and all-port architectures were addressed. With the advent of the wormhole-routed networks, the network interface characteristics play increasingly important roles in the design of communication algorithms. While a systematic approach to design efficient algorithms is very useful, more importantly, a unified model for characterizing and parameterizing the network is needed in order to compare and predict the performance of different algorithms in various environments.

2.6 The Trend Toward Clusters

The platforms used for parallel computing have continued to change over the past decade. In fact, technology has evolved to a point that the communication architectures of MPPs and workstation clusters bear more similarity than difference. In a paper bearing a similar title with the first paper introducing the RISC architecture [115], Anderson et al. [8] pushed forward a new era of the network of workstation (NOW) architecture. Using a collection of workstations to perform parallel computations is not restricted to scientific computing. For example, a

recently introduced search engine for the World Wide Web utilizes a set of workstations connected by Myrinet LAN [80]. Indeed, unprecedented computing power is possible when commercial microprocessors are coupled with high performance communication networks. However, how to harness such combined computing and communication capability is a challenging task. Past experiences with the parallel computers indicates that efficient parallel applications require not only careful layout the computation, but also the careful consideration of the underlying communication hardware. Given the economic and performance advantages of NOW platforms, many different high-speed networking technologies have been investigated in this context. Examples of such networking technology are asynchronous transfer mode (ATM), fast Ethernet, and Myrinet [27]. In order to take advantage of such technologies, network interface software must be streamlined. A number of research groups are investigating ways to reduce software overheads; examples include Berkeley NOW Fast Communication [125], Berkeley Active Messages [98, 153], Illinois Fast Messages (FM) [108], and Cornell U-Net [20]. To date, most of these efforts have focused on unicast communication, although collective communication is gaining increased attention.

Of course, the computer networking research community has already showed great interest in the study and implementation of collective communication, exemplified by efforts in designing multicast algorithms for various protocol stacks. The introduction of IP multicast [51] implementations made possible many such algorithms. Due to the unreliability of the underlying network, reliability is supported with a separate protocol. Example protocols proposed thus far include

ST-II [140] at the network layer; MTP [10], XTP [134, 157], TP++ [56], and VMTP [40] at the transport layer. The recently introduced Scalable Reliable Multicast (SRM) [57] emphasizes the functionality at both the application and session levels. The ISIS [23, 24, 25] and Horus [121, 155] systems have implemented group communication from a different perspective; they are designed to preserve message ordering in the presence of faults in the network.

Some communication packages designed to support parallel computing over conventional network technology are also based on the conventional Internet protocol stack. An important example is PVM [135]. The popularity of PVM has demonstrated that the use of off-the-shelf communication and computation components, including both software and hardware, is a viable approach to parallel processing. The implementation of MPI over workstation environments, for example, MPICH [64] and CHIMP/MPI [30], are natural extensions of this work. Several parallel processing projects have constructed communication systems based on IP multicast. This approach includes the Transis system [52] and the PCODE [31] library. All these packages support a subset of the collective operations described earlier.

Concurrent with the development of efficient communication operations over traditional network technology, the use of high-speed networks such as ATM has also received attention from the parallel processing community. Point-to-point communication performance over ATM network with different software interface implementations is studied in [94], and ATM point-to-point primitives have been used to improve the performance of PVM [160]. The exploration of low-level ATM mul-

ticast to support collective operations is presented in [77, 78]. This research has shown the potential of directly applying ATM networking technology to parallel computing. Meanwhile, a large contingent is focusing on the implementation of traditional Internet protocol stack over ATM [9].

Finally, a number of studies have addressed how to parameterize collective operations for use in distributed environments. For cluster environments that are dedicated parallel machines, such as IBM SP systems, the CCL library [13] offers parameterized collective communication. In addition, Park et al. [112] have shown how to design multicast algorithms using a parameterized communication cost model that derives from the LogP model. Finally, considering a network environment spawned from the parallel processing community, the Myrinet LAN [62], a reliable multicast algorithm based on reliable point-to-point communication primitive and credit-based flow control mechanism is reported in [154].

2.7 Computation/Communication Models

In the design and performance assessment of parallel programs, an appropriate computation model is used to aid the understanding of their behavior. In this section, several popular computation models that are used in the design of efficient parallel programs are briefly reviewed. In addition, emerging communication models, proposed by various researchers to characterize communication behavior more accurately, are discussed. The difference between modeling parallel algorithms and sequential algorithms lies not only in the parallelism from multiple concurrent threads

of control, but more importantly, in the communication costs that be taken into account. Although the Turing machine is a theoretically interesting computation model that can effectively simulate arbitrary algorithms, this model is seldom used in the design and analysis of parallel algorithms. Numerous research endeavors have sought to design a more practical model for analysis of parallel algorithms. We review some of these efforts in the following.

PRAM and variants. In the area of modeling and analyzing the complexity of parallel algorithms, the classical Parallel Random Access Machine (PRAM) model [58] has been widely used. This model assumes P concurrent processors sharing a unlimited sized memory, and closely matches the uniform memory access shared-memory machines. Several refinements to the model accounting for the handling of simultaneous access of the same data item by several processors. Examples include the EREW (exclusive read exclusive write) PRAM, the CREW (concurrent read exclusive write) PRAM, and the CRCW (concurrent read concurrent write) PRAM. Due to the simplicity of the model and the lack of a detailed cost model for communication, several variants of the PRAM model have been proposed in an attempt to better model the behavior of parallel systems [5, 42, 63].

The PRAM models assume that communication is implemented using shared memory or explicit message-passing. In the case of message-passing, the network topology is either not considered or simply assumed to be a fully-connected graph. The reason for this is twofold: assuming a specific topology invalidates the generality of the model, and no "canonical" topology exists among real parallel architec-

tures. Moreover, even for small networks, the completely connected topology is far more dense than typical multicomputer topologies, such as meshes and hypercubes. Therefore, contention for network resources, such as link bandwidth and channel sharing, is not likely to be accurately accounted for by these abstract models.

Postal model. The Postal model [18], as opposed to *telephone model*, attempts to define a communication model in terms of a single parameter λ, which is a ratio of the network latency and the time that the sender engaged in preparing the message. By using the daily mail delivery system as an analogy to the communication network, this model can be used to analyze the performance of communication algorithms such as broadcast and multicast [19]. Specifically, if a node can send multiple messages before the first arrives at its destination, the structure of an optimal distribution tree is affected. Because of the long network latency and the relative short message preparation overhead, which the model asserts by assuming $\lambda \geq 1$, a complete graph for the underlying network is suitable for the analysis. However, for high speed networks such as ATM LANs, as observed in [86], the software startup latency is on the order of 400 to 1000 μsecs, while the switch latency is on the order of few to tens of microseconds. This effectively makes λ value less than one, and therefore the optimal algorithm developed under this model may not be the best choice for such systems.

BSP model. The Bulk Synchronous Parallel (BSP) architecture model [150] was designed to achieve the goal of developing architecture-independent parallel algo-

rithms. It is intended to bridge the programmer's perspective with the actual system architecture in order to achieve portable program design. The BSP architecture is a collection of processor/memory pairs, a point-to-point virtual network, and a mechanism to synchronize all or a subset of the processor/memory nodes. The computation is divided into sequences of parallel "supersteps," with each superstep separated by a global synchronization, before which all memory and network accesses should be completed. The cost model associated with the BSP model is expressed in the form of linear combination of communication costs, synchronization costs, and computations within each superstep. The communication cost is measured by a communication primitive called the h-relation [1], which means that each processor has at most h messages to send to different processors and each processor is due to receive at most h messages. Unfortunately, the BSP model does not describe how communication occurs, that is, the communication pattern is not scheduled precisely.

C^3 model. The C^3 model [88] is an attempt to capture the cost of computation, communication patterns, and network congestion due to communication for coarse-grained parallel algorithms. Five system parameters are used as metrics for evaluating the target algorithms for a particular system. They are the number of processors p, the network latency h, the bisection width of the communication network b, the startup cost per message s, and the length of a message ℓ. The latency h is modeled as the average distance between two processors. The overall communication cost is the sum of send/receive cost, link congestion, and processor congestion

costs. The send/receive cost depends on the underlying routing mechanism and the communication mode used to send/receive a message. The routing mechanism can be store-and-forward or wormhole routing and the communication mode can be either blocking or non-blocking. This model is designed specifically to characterize parallel computer networks, wormhole network in particular. Similar to the LogP model discussed later, this model mainly considered point-to-point communication and the possible underlying hardware supported multicast capability is not included in the model.

LogP model. The LogP model is an effort to model parallel and distributed system in which all participating nodes communicate through explicit point-to-point message-passing. Associated with each message are three parameters to account for the cost of message-passing. They are the overhead o, the time that the sender is busy preparing the message; the gap g, the total amount of time elapsed from the start of one message to the initiation of the next message; and the network latency L, the time that a message spends in the network. The model also assumes that the time the receiver engaged in reception of the message is equal to the overhead o. As with the Postal model, the LogP model assumes a completely connected graph as the underlying network topology. In fact, the LogP model can be viewed as a generalization of the Postal model. By setting parameters $o = 0$, $g = 1$, and $\lambda = \frac{L}{g}$, the LogP model is exactly the same as the Postal model [18]. Recent extensions to the LogP enable the gap between packets to be much smaller than the gap between messages [7], and allow for a hierarchical memory model [93]. Although the LogP

model was carefully designed to account for communication costs, with three out of four parameters are used to describe such costs, there exists many situations the model does not adequately cover. Examples include network congestion and network supported multicast.

Hockey's model. Some models have focused exclusively on communication. Hockney [72] recently proposed communication model that identifies two metrics as the most important performance indicators for modeling point-to-point communication on massively parallel processors such as Intel iPSC/860, Paragon, and Meiko CS-2. The startup time, t_0, is the latency to send a zero-length message, and the asymptotic bandwidth, r_∞, is the averaged elapsed time for sending an m-byte message through the network. The latency formula used to describe the performance of point-to-point communications is of the form $t_0 + m/r_\infty$. Despite its simplicity, this formula matches the measured point-to-point communication performance of many massive parallel architectures [53].

Xu and Hwang's model. In an attempt to generalize Hockney's cost model, Xu and Hwang [158] incorporated collective communication operations into a new model and introduced the idea that both t_0 and r_∞ are not constants for a specific architecture, but instead are functions of the number of participating parties n. Therefore, the latency formula used in this cost model assumes the form $t_0(n) + m/r_\infty(n)$. This cost model also possesses the virtue of simplicity, and yet is precise enough to capture the aggregate behavior of collective communications. Xu and

Hwang studied different implementations of collective communication operations on the IBM SP-2 and measured the different parameters $t_0(n)$ and $r_\infty(n)$. By identifying the order of complexity of both $t_0(n)$ and $r_\infty(n)$, they showed the model is able to identify the possible inefficiencies in different implementations.

The need for a general communication model. A universal parallel computing model that satisfies both the efficiency and universality requirements is difficult to construct, since these basic requirements are often conflicting. This difficulty has led to the separation of performance modeling, analysis, and algorithm development. Modeling the complexity of parallel algorithms is based on models that are rarely used to design and develop communication algorithms. Therefore, in the design of efficient, portable parallel algorithms, the usual approach is to adopt some frequently used communication primitives and implement these primitive operations on a per-platform basis.

In order to design efficient communication primitives, an extensive communication cost model is needed. The model should be general enough to capture the characteristics of different communication platforms and capable of reflecting costs associated with resource conflicts. In the following chapter, we will show how the proposed EDN model provides an intuitive model that is powerful enough to accommodate many communication operations on various platforms.

Chapter 3

Extended Dominating Nodes in Mesh Networks

There may be nothing new under the sun, but permutation of the old within complex systems can do wonders.

Stephen Jay Gould 1977

As discussed in Chapter 2, development of efficient collective algorithms requires that the underlying hardware characteristics be taken into account. This approach has been following in the application of the Postal model and the LogP model to the design of collective algorithms for various environments. However, an important feature that is not reflected in either of these models is the ability of one node to transmit a message to multiple destinations simultaneously.

In this chapter, we develop the Extended Dominating Node (EDN) model, which explicitly models this "single-step multicast" property. Such a capability may be

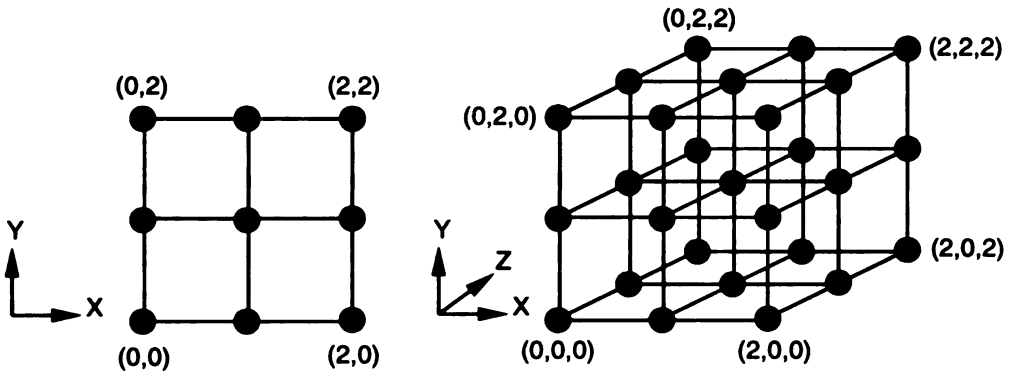
realized in different ways, for example, broadcast on a shared medium (such as Ethernet) or path-based multicast in wormhole networks. In this chapter and the follow chapter, we develop collective algorithms for wormhole-routed networks in which single-step multicast is possible due to a multi-port network interface architecture. In Chapter 5, we discuss the application of EDN to ATM networks, where single-step multicast is supported by multicast virtual channels.

3.1 System Model

Before we formally introduce the definition of Extended Domination and its application in wormhole networks, we discuss some major characteristics of the systems studied. This chapter addresses MPCs that are characterized by five properties described below.

First, their topologies are low-dimensional (2D or 3D) meshes. Formally, an $m \times n$ 2D mesh consists of $N = m \times n$ nodes; each node has an associated integer coordinate pair (x, y), $0 \leq x < n$ and $0 \leq y < m$. Two nodes with coordinates (x_i, y_i) and (x_j, y_j) are connected by a communication channel if and only if $|x_i - x_j| + |y_i - y_j| = 1$. A 3D mesh is defined similarly. Figure 3.1(a) shows a 3×3 2D mesh, while Figure 3.1(b) shows a $3 \times 3 \times 3$ 3D mesh.

Second, communication is handled by *routers*, one at each node. Several pairs of *external channels* connect each router to neighboring routers; the pattern in which the external channels are connected defines the network topology. Typically, a router can relay multiple messages simultaneously, provided that each incoming



(a) 3 x 3 2D mesh (b) 3 x 3 x 3 3D mesh

Figure 3.1: Mesh topologies

message requires a unique outgoing channel. In addition, two messages may be transmitted simultaneously in opposite directions between neighboring routers. As discussed in earlier chapter, a router is connected to the local processor/memory by *internal* channels, or *ports*. One or more of the ports is for input, and one or more is for output.

Third, these systems use the wormhole routing switching strategy [48]. Wormhole routing has been adopted in many new generation MPCs, including the MIT J-machine [47] (3D mesh); the nCUBE-2 [106] and nCUBE-3 [55] (hypercube); and the Intel Paragon [82] (2D mesh); and the Cray T3D [87] (3D torus).

Fourth, these systems use the deterministic *dimension-ordered* routing algorithm [47], in which messages are routed through dimensions of the mesh in a predetermined order. In Figure 3.1(b), for example, a message sent from node (0,0,0) to node (2,2,2) under dimension-ordered routing would follow the path: (0,0,0), (1,0,0), (2,0,0), (2,1,0), (2,2,0), (2,2,1), (2,2,2). (Since wormhole-routing is assumed here, such a message would traverse only the routers at the five inter-

mediate nodes, and would not be handled by the local processors at those nodes.) Although research in adaptive routing algorithms is very promising, most commercial systems whose topologies are n-dimensional meshes presently use some form of dimension-ordered routing.

The final distinguishing characteristic of these systems, mentioned earlier, is that each node possesses multiple internal channels, or *ports*.

3.2 Single-Level Dominating Node Broadcast

We begin with the case of broadcast in all-port 2D mesh networks. We first apply the concept of dominating sets in graphs to direct network topologies for MPCs. The MPC is modeled as a graph, with nodes represented as vertices and communication links represented as edges.

Definition 1 *A set of dominating nodes D in a direct network is a subset of nodes such that every node in the network is either in D or is a neighbor of at least one node in D.*

The first broadcasting approach studied in this project, called the *single-level dominating (SLD)* algorithm, comprises two phases. First, the message is multicast from the source node to a set of dominating nodes of the network; this phase may require several message-passing steps. In the second phase, the dominating nodes deliver the message to all other nodes in the network by forwarding the message directly to the appropriate neighbors. Once all the dominating nodes

have received the message, this second phase can be completed in one additional message-passing step. The two primary issues that arise in using this approach are (*i*) how to find a set of dominating nodes with small cardinality, and (*ii*) how to efficiently send a multicast message to those dominating nodes.

3.2.1 Dominating Nodes in a 2D Mesh

The problem of finding a dominating set (or domination number) of minimal cardinality in a rectangular graph (or grid graph) has been shown to be NP-hard [83]. Since the objective of this project is to develop *practical* broadcast methods, however, any approximation algorithm that yields a dominating set will serve our purpose as long as the number of nodes within that set is a relatively small fraction of the total number of nodes.

According to the work by Cockayne [41], we can easily find such sets of dominating nodes for square meshes. This method is illustrated in Figure 3.2. We begin by replicating a basic 5-square pattern, illustrated in Figure 3.2(a), in order to form an "infinite" background pattern, represented in Figure 3.2(b). The shaded squares represent dominating nodes; every unshaded square is adjacent to exactly one shaded square.

In order to find a set of dominating nodes for a square mesh of a given size, we move a square frame of that size over the background pattern, seeking a position in which all squares in the frame are covered minimally. As illustrated in Figure 3.2(b), our search for a dominating set for a 5×5 mesh results in the placement of the

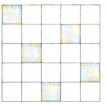
5-square frame as shown. The frame contains 5 dominating nodes, however, these are not sufficient to dominate all nodes that lie inside the 5×5 frame. As shown in Figure 3.2(c), we move two dominating nodes that lie outside the frame to its interior and adjust the positions of two of the dominating nodes that were already inside the frame, forming a set of 7 dominating nodes that covers all 25 nodes in the frame. Details of this method may be found in [41].

Also shown in Figure 3.2(b) is the optimal placement of a 4×4 frame. In this case, however, the 4 dominating nodes inside the frame represent a minimum dominating set. As shown in Figure 3.2(d), none of the nodes inside the frame are dominated by a node outside the frame, so no adjustment is necessary. In general, this approach yields a set of dominating nodes with cardinality of approximately one-fourth the number of nodes in that square mesh [41].

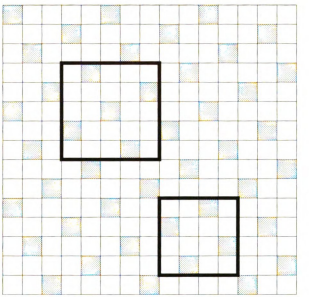
3.2.2 Multicast Algorithm

Once a set of dominating nodes has been found for a specific mesh, the first phase of the SLD broadcast algorithm can be executed, in which the source node delivers the message to all the dominating nodes in the mesh. Unicast-based multicast communication in wormhole-routed 2D mesh networks has been studied previously. For example, the U-mesh algorithm [103] was designed for one-port mesh networks, but can also be used on multi-port architectures.

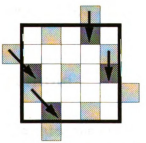
In the U-mesh algorithm, the source node and destination nodes are sorted lexicographically by their addresses into a list Φ. The source node successively



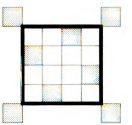
(a) basic 5–square pattern used to construct background pattern



(b) placing target squares on background



(c) adjusting dominating nodes for 5–square



(d) no adjustment necessary for 4–square

Figure 3.2: Examples of finding dominating nodes in a 5×5 mesh and a 4×4 mesh

divides Φ in half. If the source is in the lower half, then it sends a copy of the message to the first node in the upper half. In addition to the data, each message carries the addresses of the destinations for which the receiving node is responsible. The receiving node will be responsible for delivering the message to the other nodes in the upper half, using the same U-mesh algorithm in a distributed manner. If the source is in the upper half, then it sends a copy of the message to the last node in the lower half. At each step, the source deletes from Φ the receiving node and those nodes in the half not containing the source address. The source continues this procedure until Φ contains only its own address. On one-port architectures, the U-mesh algorithm is optimal in terms of the number of message-passing steps, requiring $\lceil \log_2(m+1) \rceil$ steps to reach m destinations. Moreover, the constituent unicast messages are guaranteed to be contention-free [103]. However, we emphasize that other multicast algorithms besides U-mesh could be used in the first phase of the SLD broadcast algorithm.

3.2.3 Example

Figure 3.3 illustrates the operation of the proposed SLD algorithm in a small, 4×4 mesh. Figure 3.3(a) shows the mesh itself. The four dominating nodes, (0,2), (1,0), (2,3), and (3,1), are shaded in the figure. In this example, node $(2,0)$ is the source of the broadcast. Figure 3.3(b) shows the broadcast "tree" rooted at node (2,0). In step 1, the source node transmits simultaneously to dominating nodes (1,0) and (2,3), which subsequently forward the message to dominating nodes (0,2) and (3,1),

respectively, in step 2. Although some of these messages pass through intermediate routers, the pipelining effect of wormhole routing and startup costs result in latency nearly identical to that of messages sent between neighboring nodes [107]. Since the network is assumed to be an all-port architecture, node (1,0) can also forward the message to node (1,1) in step 2, and node (2,3) can also forward the message to nodes (1,3) and (2,2) in step 2. The dominating nodes send the message to the remaining nodes in step 3.

3.2.4 Performance

In the SLD broadcast algorithm, the source node uses the U-mesh algorithm to multicast the message to the dominating nodes, which then complete the broadcast by passing the message to the appropriate neighboring nodes. Of course, the U-mesh algorithm itself can be used for broadcast, requiring $\lceil \log_2(N) \rceil$ steps, where N is the number of nodes in the network. The SLD algorithm requires approximately $\lceil \log_2(\frac{N}{4}) \rceil = \lceil \log_2(N) \rceil - 2$ message-passing steps to deliver the message to the dom-

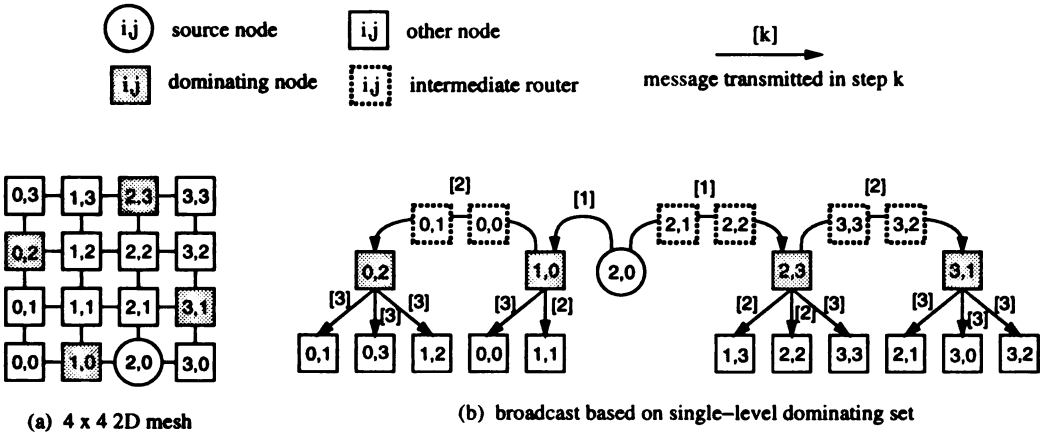


Figure 3.3: Example of SLD broadcasting in a 4×4 2D mesh

inating nodes. An additional step is required to complete the broadcast, bringing the total number of steps to $\lceil \log_2(N) \rceil - 1$. Although the SLD algorithm requires one less step than that of the U-mesh algorithm when used alone, the advantage may not be worth the added complexity of the algorithm. This observation led us to investigate a different approach to broadcast that relies entirely on the concept of dominating nodes.

3.3 EDN Broadcast in 2D Mesh Networks

A natural extension to the previous approach is to find some subset of the dominating nodes that are capable of sending the message to the other dominating nodes in a single message-passing step and without channel contention. Once each of these nodes has a copy of the message, then broadcast can be completed in two additional steps.

3.3.1 Extended Dominating Sets

The method described in this section applies such a method recursively until a very small set of nodes, which will be sent the message directly from the source, are used to begin the broadcast process. This approach broadens the concept of node domination by accounting for the distance-insensitivity of wormhole routing. We formally define the concept of an *extended* dominating set as follows.

Definition 2 *Consider an all-port wormhole-routed direct network with node set V, a routing algorithm R, a set D_1, $D_1 \subset V$, and a set D_2, $D_2 \subset D_1$. The*

set D_2 is said to be an extended dominating set (EDS) *of D_1 if and only if there exists a set of edge-disjoint paths $\mathcal{P}$ under R such that, for every node v in $D_1 - D_2$, there exists a node x, $x \in D_2$, and a path p, $p \in \mathcal{P}$, from x to v.*

Figure 3.4 illustrates four examples of extended domination in an 8×4 2D mesh network under XY routing. In each of the four figures, the 8 shaded nodes (two different shades are used) constitute a dominating set D_1 of all nodes in the mesh, and 2 of those 8 nodes form an extended dominating set D_2 of D_1. On an all-port architecture and using XY routing, the nodes in D_2 can transmit messages, as shown, to the other nodes in D_1 in a single step without channel contention.

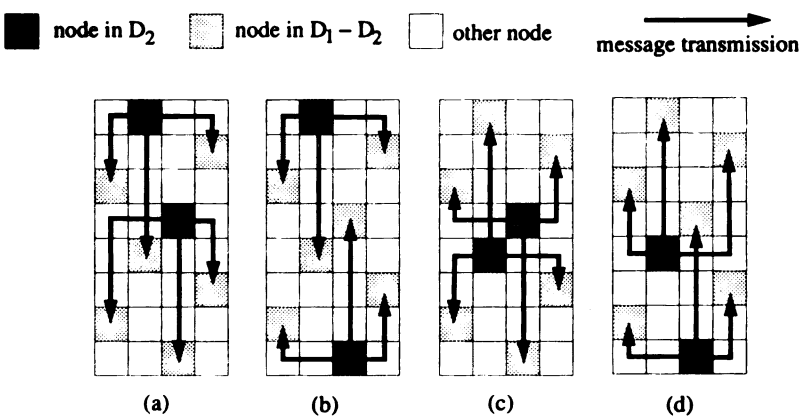


Figure 3.4: Examples of extended dominating sets in 8×4 2D meshes

In order to model multiple steps of a broadcast algorithm, we need to formally define the concept of *levels* of extended dominating sets. Hence, an EDS of a set of nodes X in a direct network will also be referred to as a level-1 EDS of X. For completeness, a level-0 EDS of a set X is just X itself.

Definition 3 *Consider an all-port wormhole-routed direct network with node set V, a routing algorithm R, a set $X \subset V$, and a set $D_i \subset X$. The set D_i is a*

level-i *EDS of* X *if and only if* D_i *is a level-1 EDS of a level-*$(i-1)$ *EDS of* X.

Given Definitions 2 and 3, the problem of performing efficient broadcast from a source node s in a wormhole-routed direct network with node set V can be stated as follows: Find a collection of multi-level EDSs, D_i, $i = 0, \ldots, t$ such that $\{s\}$ is a level-t EDS of V and t is minimum among all such collections. Once such a collection of sets has been found, broadcast can be performed from node s in a minimum number of steps, while avoiding contention among the unicast messages transmitted in each step.

Although numerous methods could be used to perform this task, we sought a method that will not only achieve low latency, but that will be simple to implement and applicable to meshes of different shapes and sizes. Specifically, it is desirable that the pattern of dominating nodes at each level be "regular," in order that the algorithm can be implemented easily and executed quickly. The generality of the approach will be important because, while determining the sets of multi-level dominating nodes for a physical architecture need be carried out only once, in some systems a given application may be allocated only a subset of the nodes on the network. If these nodes are organized as a (square or rectangular) submesh, then the application would benefit from a library broadcast routine that accommodates a large variety of possible submesh shapes and sizes.

3.3.2 Systematic Approach to Finding Multi-Level Extended Dominating Sets

Initially, we describe the proposed method in terms of meshes of size $2^k \times 2^k$; later, we show how the approach can be applied in other square and rectangular meshes. An optimal four-node dominating set for a 4×4 mesh was shown in Figure 3.2(d). Suppose that we wish to find a dominating set for an 8×8 mesh. Figure 3.5(a) depicts a minimum dominating set, comprising 16 nodes, for such a mesh. Figure 3.5(b) shows another dominating set for the 8×8 mesh, constructed by simply combining four solutions to the 4×4 problem. For each mesh, a level-2 dominating set comprising four nodes can be used to reach the level-1 dominating nodes. The advantage of the regular pattern in Figure 3.5(b) is that the number and pattern of messages transmitted by each level-2 dominating node is identical. That is to say, the four level-2 nodes can complete the broadcast in two steps using either pattern, but the actions are more regular for the pattern shown in Figure 3.5(b).

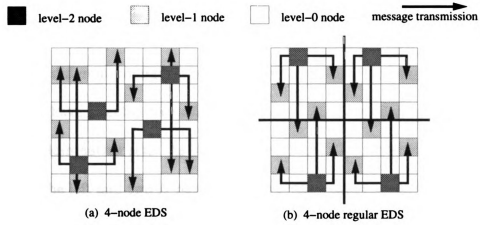


Figure 3.5: Level-1 and level-2 dominating nodes in 8×8 mesh

Using such regular patterns is advantageous to finding broadcast patterns for

larger meshes. In fact, by treating 4×4 meshes and their dominating sets as building blocks, we can construct level-2 dominating sets for any mesh of size $2^k \times 2^k$ for any $k \geq 2$. The problem becomes how to recursively find a higher level dominating nodes. In order to develop a regular approach to this problem, the orientation of the 4×4 dominating nodes must be considered.

There are essentially two different configurations for (non-extended) dominating nodes in a 4×4 mesh, as shown in Figure 3.6. We will refer to these patterns as A and B, respectively. Pattern B is the same as that shown earlier in Figure 3.2(d), and pattern A is the mirror image of pattern B across the vertical axis.

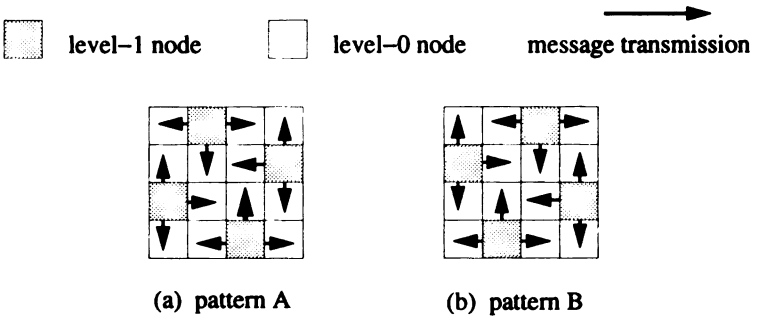


Figure 3.6: Two basic configurations for dominating nodes in a 4×4 mesh

By combining four instances of patterns A and B in all possible ways, we can construct 16 different configurations for an 8×8 mesh. Among these 16 configurations, we have selected two configurations to demonstrate the construction of higher-level dominating sets. As illustrated in Figure 3.7, pattern C places two A patterns on the left and two B patterns on the right, while pattern D places two A patterns on the right and two B patterns on the left. Patterns C and D are mirror images of one another, this time with respect to the horizontal axis.

It is straightforward to find four level-2 dominating nodes for 8×8 meshes with

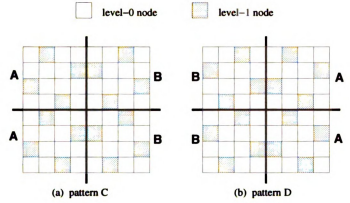


Figure 3.7: Two configurations of synthetic dominating nodes in a 8×8 mesh

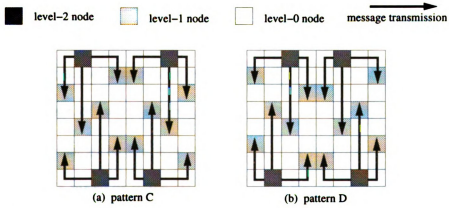


Figure 3.8: Level-2 dominating nodes sending to level-1 dominating nodes in 8×8 meshes

either pattern C or pattern D at level 1. Figure 3.8 illustrates four level-2 dominating nodes for each pattern. Under XY routing on an all-port architecture, the level-2 nodes can transmit messages to all of the level-1 nodes in a single message-passing step.

Figure 3.9 depicts the message passing from level-3 dominating nodes to level-2 dominating nodes in a 16×16 mesh. Since the pattern is formed by placing two C patterns above two D patterns, the reader will notice that the right half of the mesh is a mirror image of the left half, and that the lower half is a mirror image of the top half. Again, under XY routing, the message passing between level-3 nodes and level-2 nodes can be completed in a single step. This systematic approach can be applied to any mesh of size $2^k \times 2^k$. Essentially, the broadcast "tree" is constructed in a bottom-up manner. A dominating set of all nodes in the network is found and placed at level 1 of the tree; all non-dominating nodes are placed at level 0. A dominating set of the level 1 nodes is found and placed at level 2, and so on. This process continues until four nodes are placed at level $(k-1)$.

The resulting algorithm, which we call the *EDN* (extended dominating nodes) algorithm, consists of two distinct phases. In the *startup* phase, the source node sends out the message to the four dominating nodes at the highest level. In the second phase, the message is iteratively passed down the tree to lower-level dominating nodes, until, after the last step, all nodes in the network will have received a copy of the message.

The method used by the source node to deliver the message to the highest-level dominating nodes, as well as the number of message-passing steps required, depends

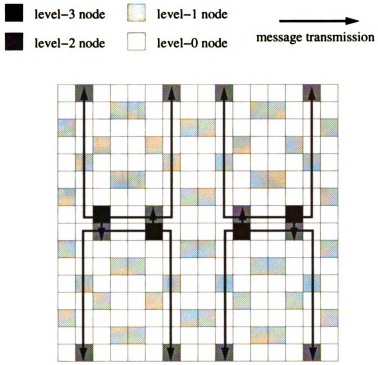


Figure 3.9: Level-3 dominating nodes sending to level-2 dominating nodes in a 16×16 mesh

on the position of the source relative to those nodes. Figure 3.10 illustrates two examples of the startup phase in an 8×8 mesh. In Figure 3.10(a), the source node sends the message to two of the level-2 dominating nodes in the first step, each of which forwards the message to another level-2 dominating node in the second step. The reader will notice that each of the level-2 dominating nodes that received the message in step 1 can also proceed to forward the message to two of the *level-1* dominating nodes in step 2. Since the architecture is assumed to be all-port, an overlapping of steps is permitted, thus allowing some level-0 nodes to receive the message *before* the last step. In Figure 3.10(b), since all the level-2 dominating nodes are to the right of the source node, under XY routing, the source node can transmit to only one of them in the first step. At the end of the second step, all

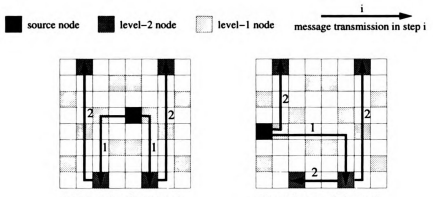


Figure 3.10: Startup message passing in 8×8 meshes

level-2 dominating nodes (and some level-1 dominating nodes) will have received the message.

As mentioned earlier, the multilevel dominating set approach to broadcast can be applied to square meshes whose widths are not powers of two, as well as to rectangular meshes. Of course, for mesh sizes other than powers of two, the basic building blocks are different than the 4×4 meshes shown in Figure 3.6. Figure 3.11 illustrates basic building blocks of size 5×5, 6×6, and 7×7, which were found using the method [41] illustrated in Figure 3.2. The performance of each is summarized in the next subsection.

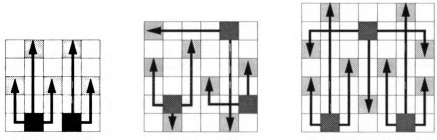


Figure 3.11: Basic building blocks for meshes not based on powers of two

In each of the cases described thus far, we have shown only one or a small

number of basic building blocks. One disadvantage of having a fixed pattern for all broadcasts is that, in the presence of a large amount of broadcast traffic, system resource usage may become imbalanced. However, it turns out that the patterns presented here are not the *only* ones that would suffice for broadcast. On the other hand, if broadcast were expected to be a relatively infrequent, but time-critical, operation, the proposed methods could be supported in hardware.

3.3.3 Theoretical Performance

As mentioned earlier, we evaluate the performance of the EDN from both a graph theoretical and from a systems perspective. Since several other projects evaluate the performance of multi-port collective algorithms by counting message-passing steps [21, 75, 111, 113, 118], we first assess algorithms in this manner. As in these related projects, it is assumed that a k-port node can send (receive) k messages simultaneously, that is, serialization of startup latencies is not reflected. In order to predict the performance of the EDN algorithms on actual systems, however, we also conducted simulation studies. The simulations account for sequential startups, as well as other system conditions and parameters.

We have calculated the number of message-passing steps required for EDN broadcast in square meshes of width $n = 4 \times 2^k, 5 \times 2^k, 6 \times 2^k, 7 \times 2^k$. The total number of steps consists of two parts, corresponding to the startup phase and message passing among multiple levels of EDNs. Tables 3.1 through 3.4 give these values. The number of steps needed in the startup phase depends on the position

of the source node relative to the highest-level EDNs.

Table 3.1: EDN steps for $n = 4 \times 2^k$

k	0	1	2	3	4	5	k
$n = 4 \times 2^k$	4	8	16	32	64	128	4×2^k
min startup	2	2	2	2	2	2	2
max startup	2	2	2	2	2	2	2
tree height	1	2	3	4	5	6	$k+1$
min total	3	4	5	6	7	8	$k+3$
max total	3	4	5	6	7	8	$k+3$

Table 3.2: EDN steps for $n = 5 \times 2^k$

k	0	1	2	3	4	5	k
$n = 5 \times 2^k$	5	10	20	40	80	160	5×2^k
min startup	1	1	1	1	1	1	1
max startup	2	2	2	2	2	2	2
tree height	2	3	4	5	6	7	$k+2$
min total	3	4	5	6	7	8	$k+3$
max total	4	5	6	7	8	9	$k+4$

Table 3.3: EDN steps for $n = 6 \times 2^k$

k	0	1	2	3	4	5	k
$n = 6 \times 2^k$	6	12	24	48	96	192	6×2^k
min startup	1	1	1	1	1	1	1
max startup	2	2	2	2	2	2	2
tree height	2	3	4	5	6	7	$k+2$
min total	3	4	5	6	7	8	$k+3$
max total	4	5	6	7	8	9	$k+4$

Table 3.4: EDN steps for $n = 7 \times 2^k$

k	0	1	2	3	4	5	k
$n = 7 \times 2^k$	7	14	28	56	112	224	7×2^k
min startup	1	1	1	1	1	1	1
max startup	2	2	2	2	2	2	2
tree height	2	3	4	5	6	7	$k+2$
min total	3	4	5	6	7	8	$k+3$
max total	4	5	6	7	8	9	$k+4$

In order to evaluate the EDN broadcast algorithm, we compare its performance to that of an algorithm based on *recursive doubling* (RD) and to a lower bound. The RD algorithm [102, 103] was designed for the general case of multicast in one-port n-dimensional meshes, and can also be used for the special case of broadcast. Figure 3.12 illustrates the operation of the RD algorithm on a multicast problem in a 6×6 2D mesh. The RD algorithm first sorts the source and destination addresses lexicographically into a list, denoted Φ. The source node successively divides Φ in half. If the source itself is in the lower half, then it sends a copy of the message to the first node in the upper half. That node will be responsible for delivering the message to the other nodes in the upper half, by invoking the same algorithm recursively. If the source is in the upper half, then it sends a copy of the message to the last node in the lower half. The source continues this procedure until Φ contains only its own address.

The RD algorithm doubles the number of nodes that hold the message in each

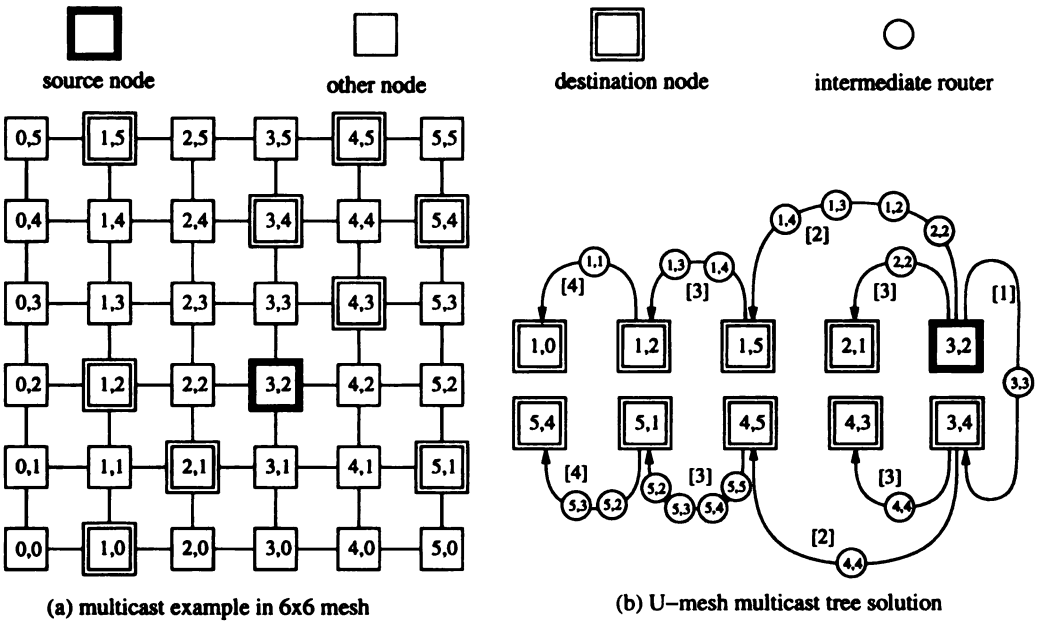


Figure 3.12: RD multicast solution in a 6×6 mesh.

step, and therefore is optimal on one-port architectures, requiring $\lceil \log_2(N) \rceil$ steps to complete a broadcast operation in an N-node network. Although the RD algorithm can be implemented on a multi-port architecture, it will often fail to take advantage of that architectural property, resulting in more message-passing steps than necessary. In fact, the number of steps is still $\lceil \log_2(N) \rceil$.

A simple lower bound on the number of message-passing steps needed for broadcast in a three-port 2D mesh is computed as follows. If, in every time step, every node that already holds a copy of the message is capable of forwarding the message to three other nodes, then the number of nodes that have received the message after time step t is 4^t. Given a mesh of size $n \times n$, this lower bound on the time complexity is derived to be $T \geq \lceil \log_4(n^2) \rceil$.

Figure 3.13 plots the number of steps needed to perform broadcast in $n \times n$ meshes, for $1 \leq n \leq 256$. The number of nodes in the mesh varies from 1 to

65,536. The number of steps required by the RD algorithm is equal to $\lceil \log_2(n^2) \rceil$, regardless of the location of the source node. The number of steps required by the EDN algorithm depends on the location of the source node, but the maximum number of steps is at most one more than the minimum number. As shown in Figure 3.13, in many cases the minimum number of EDN steps matches the lower bound of $\lceil \log_4(n^2) \rceil$, and in some cases even the maximum number of EDN steps achieves the lower bound.

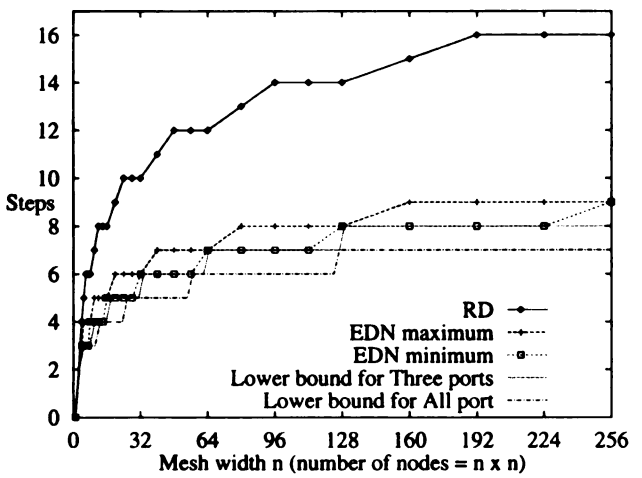


Figure 3.13: Comparison of EDN steps with RD and lower bound (2D mesh)

3.3.4 Analysis

In order to better understand the performance of these two algorithms, we develop an analytical model. Let α represent the message sending latency and γ represent the receiving latency. Let L be the message length, and let β denote the per-hop transmission time. If the message is long enough that we can ignore the number of hops traversed by each message, then we can approximate the message latency by $\alpha + \beta L + \gamma$.

In a $2^k \times 2^k$ mesh, the RD broadcast algorithm consists of $2k$ message-passing steps. In each step, every node holding the message sends it to one other node. Therefore, the total time to execute the algorithm can be approximated by:

$$\begin{aligned} \mathcal{L}_{\text{RD broadcast}} &= 2k(\alpha + \beta L + \gamma) \\ &= 2k\alpha + 2k\gamma + 2k\beta L. \end{aligned}$$

In each message-passing step of the EDN broadcast algorithm, other than the first two steps, a parent node sends three messages to its children. Again ignoring the number of hops traversed by each message, the time for this operation is approximated by $3\alpha + \beta L + \gamma$. For a $2^k \times 2^k$ mesh, $k - 1$ such steps are required to broadcast from the four highest-level EDNs, and two steps are needed to deliver the data from the source node to the four highest-level EDNs. Depending on the configuration (see Figure 3.10), one of the first two steps requires time $\alpha + \beta L + \gamma$ and the other step requires time $2\alpha + \beta L + \gamma$. The total time for the EDN broadcast

algorithm is approximated by:

$$\begin{aligned} \mathcal{L}_{\text{EDN broadcast}} &= (k-1)(3(\alpha)+\beta L+\gamma)+3\alpha+2\beta L+2\gamma \\ &= 3k\alpha+(k+1)\gamma+(k+1)\beta L. \end{aligned}$$

Comparing these two approximations, we see that they differ in the coefficients of α, βL, and γ. If we assume that sending and receiving latencies are approximately equal ($\alpha \approx \gamma$), then the sum of the first two terms is $(4k+1)\alpha$ for EDN and $4k\alpha$ for RD. In the RD algorithm, the coefficient of L is $2k\beta$, while in the EDN algorithm, it is $(k+1)\beta$. By combining these two results, the EDN algorithm can be expected to perform better than the RD algorithm when message length L is greater than $\frac{\alpha}{(k-1)\beta}$.

3.3.5 Simulation Study

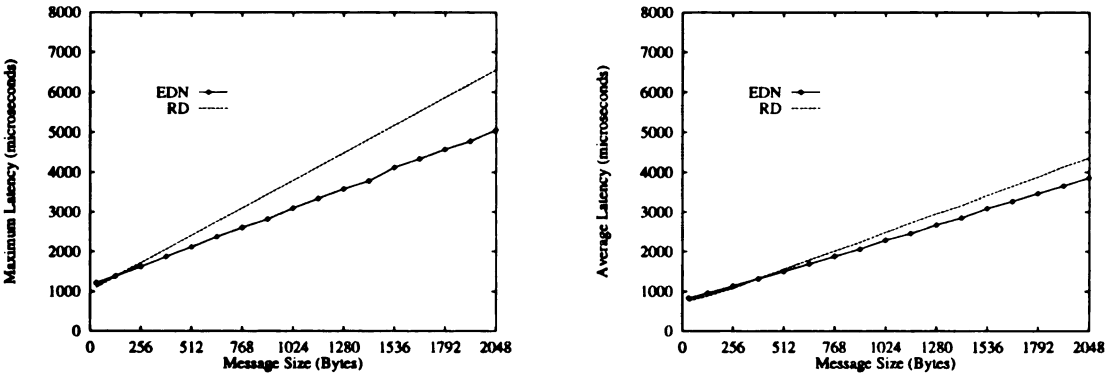
The previous two sections evaluated the performance of the EDN algorithm in terms of the number of message-passing steps and analysis based on sending/receiving overheads and per-byte transmission time. When implemented on an MPC, other factors such as path length and channel contention, affect the latency of unicast-based collective operations.

In order to compare the algorithms while accounting for such system characteristics, a simulation study was conducted. As part of an earlier project, we have developed a simulation tool called *MultiSim* [100] for the study of large-scale multiprocessors. MultiSim is based on an event-driven simulation package, CSIM [127],

which allows multiple pseudo-processes to execute in a quasi-parallel fashion and provides a convenient interface for writing modular simulation programs. MultiSim is designed to efficiently simulate wormhole-routed systems.

MultiSim was used to simulate broadcast operations in 3-port meshes of different sizes. For each mesh size, up to 100 different source nodes were selected randomly. For a given broadcast operation, the *average* broadcast latency is the mean of the broadcast latencies of all the destinations. The *maximum* broadcast latency is the largest broadcast latency among all destinations, that is, the time between when the source sends the message until the last destination receives it. Both values may be of interest, depending on how broadcast is used in a particular parallel application. Since EDN is assumed to be implemented in software, an intermediate node must fully receive the message before it forwards it to another node(s), and sending latency is incurred for each message transmitted. The computation time of a broadcast operation at an intermediate node is small (typically a table lookup determining to which nodes to forward the message) and is assumed to be part of the startup latency.

We first studied the performance of EDN broadcast on a multi-port 2D mesh with latency characteristics of the only commercial multi-port system, the nCUBE-2 hypercube. Specifically, the simulated channel rate 0.45 microseconds per byte, and a combined (sending and receiving) startup latency of 170 microseconds. As with all other simulations presented in this paper, sending latencies are simulated *sequentially*. For example, if a node is sending two messages in succession, the sending latencies do not overlap, even if the messages are transmitted on different



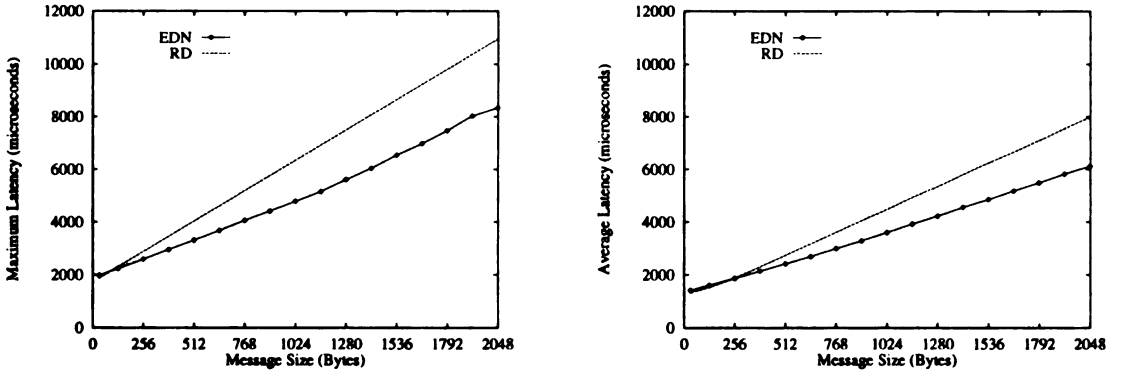
(a) maximum broadcast latency (b) average broadcast latency

Figure 3.14: Comparison of broadcast algorithms in 8×8 mesh (NCUBE-2 parameters)

ports and external channels. (The same is true of receiving latency, though this property does not affect the broadcast results.) Therefore, even though the system has an multi-port architecture, some staggering will occur among successive messages. This characteristic is consistent with our experience on the nCUBE-2 [99].

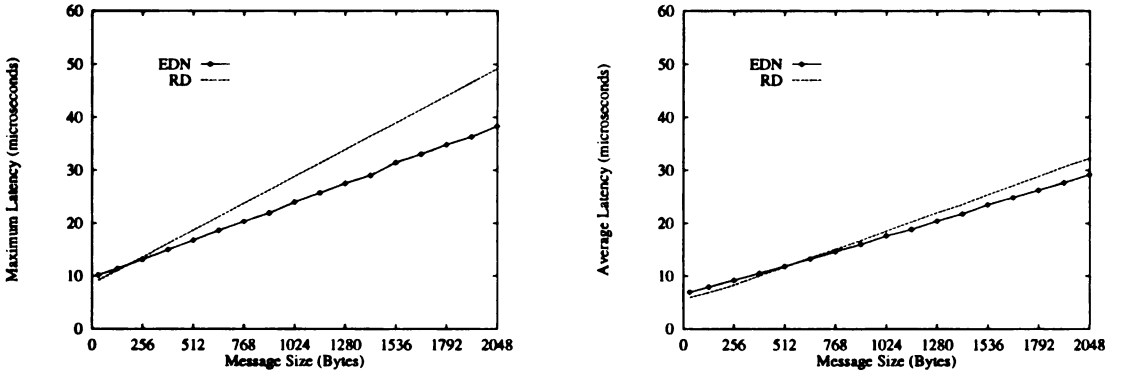
Figure 3.14 compares the maximum and average broadcast latencies of the two broadcast algorithms (EDN and RD) in an 8×8 mesh, and Figure 3.15 compares the algorithms in a 32×32 mesh. The message size is varied from 32 bytes to 2048 bytes. The advantage of the EDN algorithm is significant, and the advantage is greater in the larger mesh. In fact, inspection of the plots reveals that the maximum broadcast latency of the EDN algorithm in a 32×32 mesh is approximately equal to that of the RD algorithm in an 8×8 mesh.

We also wanted to evaluate the EDN algorithm on a newer architecture, since the nCUBE-2 architecture is now several years old. In the second set of tests, the values of the latency parameters are consistent with the Cray T3D. Specifically, the simulated channel rate is 300 MBytes/sec (0.0033 microseconds per byte, per hop)



(a) maximum broadcast latency (b) average broadcast latency

Figure 3.15: Comparison of broadcast algorithms in 32×32 mesh (NCUBE-2 parameters)

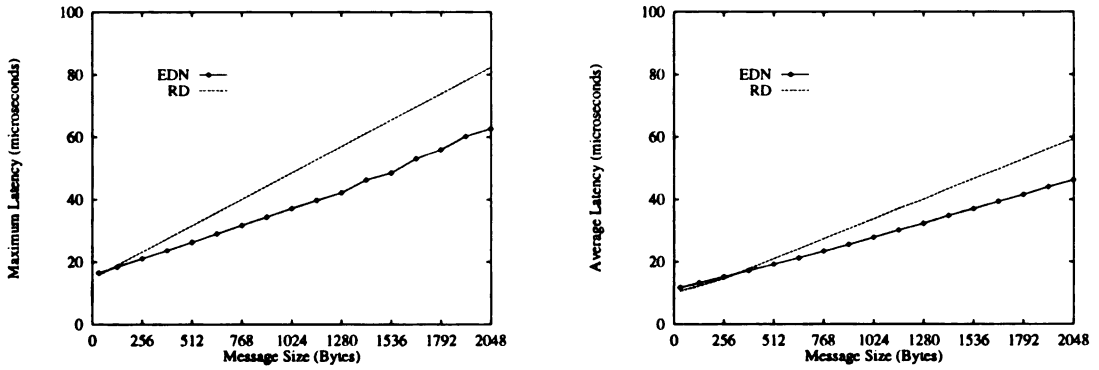


(a) maximum broadcast latency (b) average broadcast latency

Figure 3.16: Comparison of broadcast algorithms in 8×8 mesh (T3D parameters)

and the combined sending and receiving latency is 1.5 microseconds. Figures 3.16 and 3.17 compare the algorithms in 8×8 and 32×32 meshes, respectively. Again, the performance advantage of the EDN algorithm is significant.

Although the network latency in wormhole routing is relatively insensitive to the distance between the source and destination nodes, the total path lengths of the messages involved in a collective operation indicate the amount of communication resources that are consumed by the operation. A collective operation that consumes few resources may be less likely to negatively affect other network traffic. The link usage of the EDN algorithm is lower than that of the RD algorithm, due to the



(a) maximum broadcast latency (b) average broadcast latency

Figure 3.17: Comparison of broadcast algorithms in 32×32 mesh (T3D parameters)

divide-and-conquer nature of EDN. In the last step of EDN broadcast, for example, a full three-fourths of all the nodes receive the message from an immediate neighbor. In a 32×32 mesh, the average path length of a message in EDN is 1.86 hops, whereas the average for RD is 4.16 hops.

3.4 EDN Broadcast in 3D Mesh Networks

We illustrate the application of the EDN approach to 3D mesh network by way of an extended example. We first consider EDN patterns in 2D meshes, upon which the 3D mesh solutions are based. In a 4×4 network, several different level-1 EDN configurations can be found, each of which can serve as a building block pattern in larger networks. Figures 3.18(a) and (b) show four level-1 EDN configurations and their abstract representations, respectively. In the rest of this section, we will demonstrate a systematic approach for constructing multi-level EDNs for 3D meshes by starting from these four configurations.

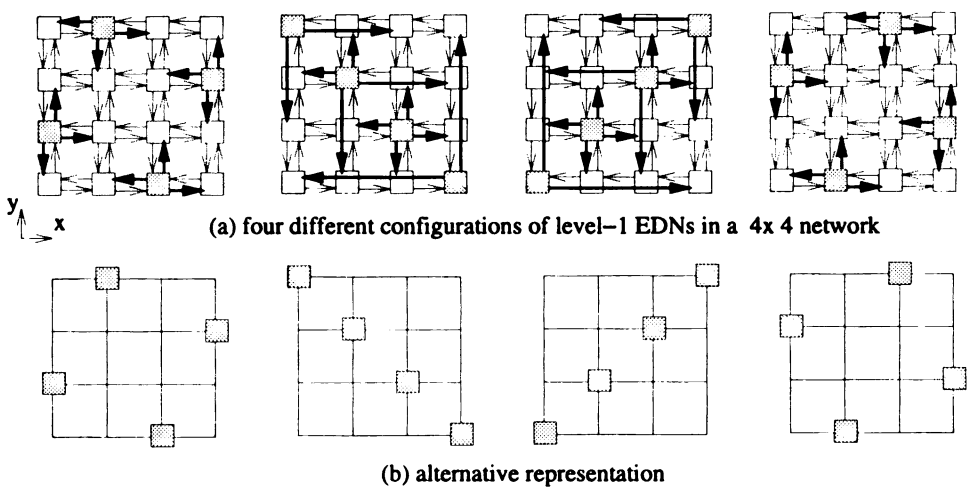


Figure 3.18: Different EDNs in 4×4 meshes

We begin with a $4 \times 4 \times 4$ 3D mesh network. Using any permutation of the above four configurations, one configuration in each of the four planes, gives a 16-node level-1 EDS for the entire network. Figure 3.19 shows one such configuration (for clarity, not all connections in the z dimension are shown). The configuration used in Figure 3.19 possesses an interesting feature, as shown in Figure 3.20. Under XYZ

routing, these 16 level-1 EDNs can together dominate an additional 4×4 plane. Therefore, the number of level-1 EDNs for a $4 \times 4 \times 5$ network is the same as that of a $4 \times 4 \times 4$ network.

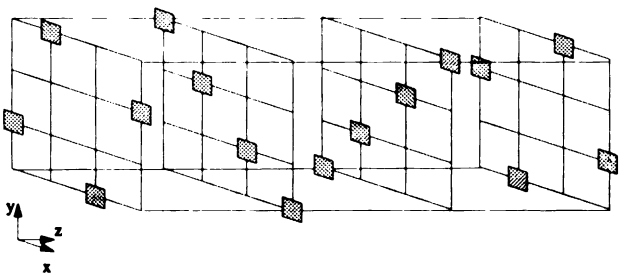


Figure 3.19: 16-node level-1 EDNs in a $4 \times 4 \times 4$ mesh

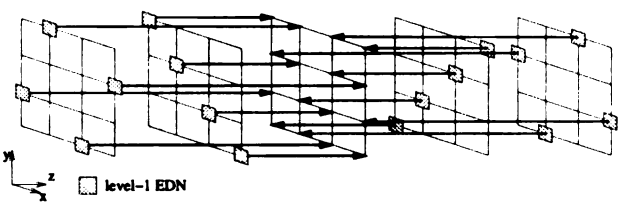


Figure 3.20: Combining EDN configurations to cover an additional plane

Next, we seek a set of level-2 EDNs that can reach the other level-1 EDNs in a single message-passing step and without channel contention. A set of four such nodes is shown in Figure 3.21(a). Also shown in the figure are the routes taken by the messages sent to the twelve other level-1 nodes; all messages follow XYZ paths. Given this configuration of the level-2 EDNs, any source node can deliver the message to them in two startup steps. Therefore, broadcast in a $4 \times 4 \times 5$ network can be completed in four message-passing steps.

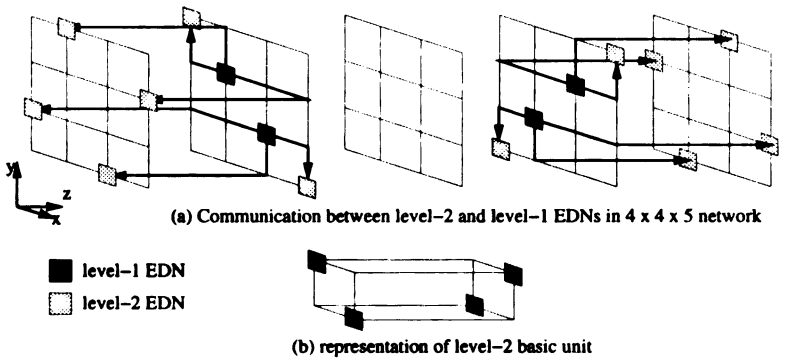


Figure 3.21: Level-2 EDNs in a $4 \times 4 \times 5$ network

3.4.1 Recursion in XY plane

We can represent the configuration of the four level-2 EDNs as shown in Figure 3.21(b). This particular configuration plays an important role in the recursive construction of multi-level EDNs in larger 3D mesh networks, and we refer to it as a *basic unit*. By combining four basic units as shown in Figure 3.22(a), we define the 16 level-2 EDNs of a larger network with four times the number of total nodes. By seeking a basic unit among these sixteen level-2 EDNs, as illustrated in Figure 3.22(b), four level-3 EDNs are defined. The paths followed by messages from level-3 EDNs to level-2 EDNs are shown in Figure 3.22(b). The twelve other nodes are reached while avoiding contention among the messages. As a basic unit, the orientation of the four level-3 EDNs is the same as the representation in Figure 3.21(b), with the pattern simply reversed in the Z dimension.

This method can be applied recursively to larger networks. In general, if the basic unit represents the highest-level EDNs in a network of size $\alpha \times \alpha \times \beta$, then four such basic units can be combined, and a new basic unit is found that represents the highest-level EDNs in a network of size $(2 * \alpha) \times (2 * \alpha) \times \beta$. Specifically, this basic

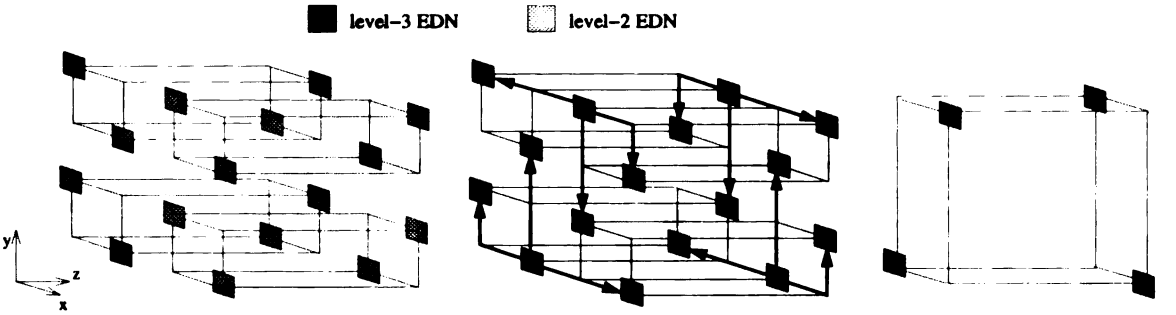


Figure 3.22: Level-3 EDNs in a $8 \times 8 \times 5$ network

unit can represent the highest-level EDNs in networks of size $4 \times 4 \times 4$, $8 \times 8 \times 4$, $16 \times 16 \times 4$, ..., $(4 * 2^k) \times (4 * 2^k) \times 4$ and $4 \times 4 \times 5$, $8 \times 8 \times 5$, $16 \times 16 \times 5$, ..., $(4 * 2^k) \times (4 * 2^k) \times 5$, where $k = 0, 1, 2$, and so on.

3.4.2 Recursion along the Z-axis

By positioning three basic units along the z-axis, as shown in Figure 3.23(a), we can create a network three times the original size. In Figure 3.23(b), we can see that 4 nodes in the middle section can serve as the highest-level EDNs in this $8 \times 8 \times 15$ network. Again, the basic unit of the level-4 EDNs in this network is the same as that of the level-3 EDNs in the $4 \times 4 \times 5$ network, as shown in Figure 3.22(c).

This approach can be extended to larger networks. In general, if the basic unit represents the highest-level EDNs in a $\alpha \times \alpha \times \beta$ network, then after the operation as described above, the same basic unit can be used to represent the highest-level EDNs in a $\alpha \times \alpha \times (3 * \beta)$ network. Combined with the earlier result, this basic unit can also represent the highest-level EDNs in a network of size $(4*2^k) \times (4*2^k) \times (4*3^m)$ and $(4 * 2^k) \times (4 * 2^k) \times (5 * 3^m)$ where $k = 0, 1, 2, \cdots$ and $m = 0, 1, 2$, and so on. The

techniques described here generalize to meshes of other shapes by using different basic units.

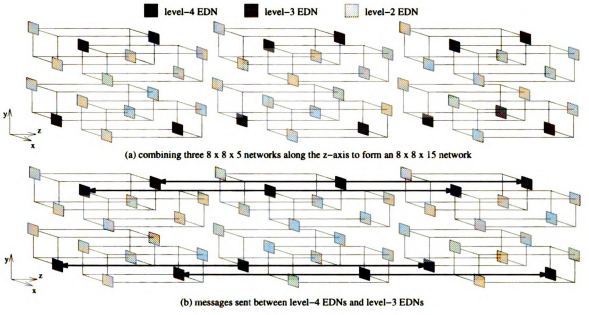


Figure 3.23: Example of concatenating networks in the Z direction

3.4.3 Theoretical Performance

The number of startup steps required to send the message from the source node to the four highest-level EDNs is two, regardless of the location of the source. This property can be seen from the orientation of the basic unit. The number of message-passing steps in networks of size $(4\times2^k)\times(4\times2^k)\times(4\times3^m)$ is $(k+2)+m+2 = k+m+4$, where $k \geq 0$ and $m \geq 0$. The numbers of steps required in specific networks are listed in Table 3.5 in increasing order of the total number of nodes in the network.

As with the 2D case, we compare the number of message-passing steps of the EDN algorithm with that of the RD algorithm and a theoretical lower bound. For a 3D mesh with four output ports per node, a lower bound is $\lceil \log_5(N) \rceil$. Figure 3.24

Table 3.5: Theoretical EDN performance in 3D mesh networks

Network size	Total nodes	Levels of EDNs	Total number of steps
$4 \times 4 \times 4$	64	2	4
$4 \times 4 \times 5$	80	2	4
$4 \times 4 \times 12$	192	3	5
$4 \times 4 \times 15$	240	3	5
$8 \times 8 \times 4$	256	3	5
$8 \times 8 \times 5$	320	3	5
$4 \times 4 \times 36$	576	4	6
$4 \times 4 \times 45$	720	4	6
$8 \times 8 \times 12$	768	4	6
$8 \times 8 \times 15$	960	4	6
$16 \times 16 \times 4$	1024	4	6
$16 \times 16 \times 5$	1280	4	6
$4 \times 4 \times 108$	1728	5	7
$4 \times 4 \times 135$	2160	5	7
$8 \times 8 \times 36$	2304	5	7
$8 \times 8 \times 45$	2880	5	7

compares the maximum broadcast latency (in message-passing steps) of the EDN algorithm and the RD algorithm for 3D meshes of specific sizes up to 5000 nodes. The advantage of the EDN algorithm over RD is clear; the performance of the EDN algorithm is within a small number of steps of the lower bound.

3.4.4 Analysis

Following the same notations of the previous analysis for 2D broadcast and combine, this section provides some analytical comparison of the performance of these two algorithms.

In a $(4 \times 2^k) \times (4 \times 2^k) \times (4 \times 2^m)$ mesh, the RD broadcast algorithm consists of $2k+m+6$ message-passing steps, with each receiving node combining the incoming

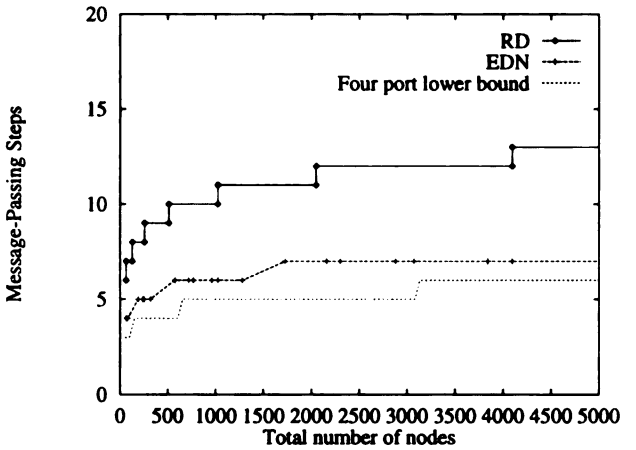


Figure 3.24: Comparison of EDN steps with RD and lower bound (3D mesh)

vector its local data. Therefore, the total time to execute the algorithm can be approximated by:

$$\begin{aligned} \mathcal{L}_{\text{RD broadcast}} &= (2k+m+6)*(\alpha+\beta L+\gamma) \\ &= (2k+m+6)\alpha+(2k+m+6)\gamma+(2k+m+6)\beta L. \end{aligned}$$

Although the number of message-passing steps of the EDN algorithm is the same for $(4\times 2^k)\times(4\times 2^k)\times(4\times 2^m)$ and $(4\times 2^k)\times(4\times 2^k)\times(4\times 3^m)$ meshes, it is more convenient in the case of the RD algorithm to analyze the $(4\times 2^k)\times(4\times 2^k)\times(4\times 2^m)$ mesh.

In the first two steps of the EDN broadcast algorithm, a total time of $3\alpha+2\beta L+2\gamma$ is needed for the source to reach the four highest-level EDNs. Proceeding along the Z-direction, another m message-passing steps are needed, and for each step, two messages are sent out from each node. Thus, a maximum time of $m(2\alpha+\beta L+\gamma)$ is required. The broadcast algorithm now is executed on each XY-plane, where $k+2$

steps are needed. In the first $k+1$ steps, 3 messages are sent from each node holding the message, and in the last step, at most 4 messages are sent from an EDN. The total time for the EDN broadcast algorithm is approximated by:

$$\begin{aligned}\mathcal{L}_{\text{EDN broadcast}} &= (3\alpha + 2\beta L + 2\gamma) + m(2\alpha + \beta L + \gamma) + \\ &\quad (k+1) * (3\alpha + \beta L + \gamma) + (4\alpha + \beta L + \gamma) \\ &= (3k + 2m + 10)\alpha + (k + m + 4)\gamma + (k + m + 4)\beta L\end{aligned}$$

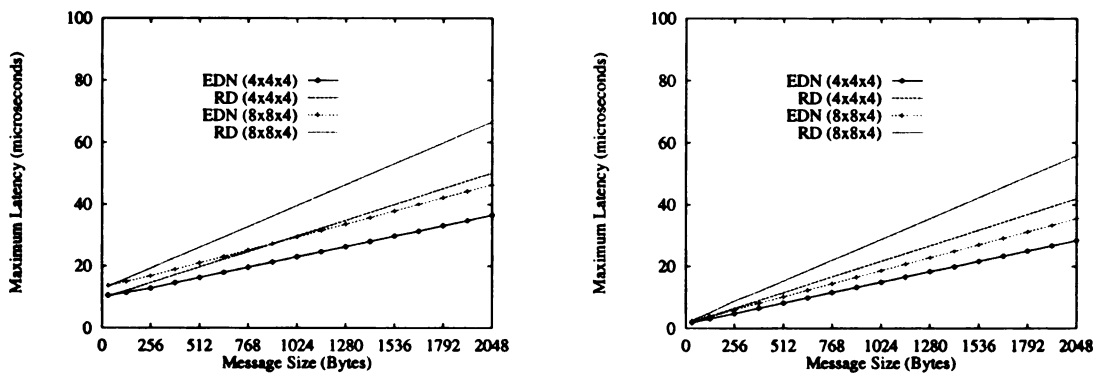
Again, if we assume that sending and receiving latencies are approximately equal ($\alpha \approx \gamma$), and combining these two approximations, the EDN algorithm is expected to perform better than the RD algorithm when the message length L is greater than $\frac{(m+3)\alpha}{(k+2)\beta}$.

3.4.5 Simulation Study

The MultiSim simulator was used to study the performance of EDN broadcast in 3D mesh networks under various system conditions. Two different mesh sizes were simulated. For each mesh size, different source nodes were selected randomly. The channel rate parameter is again consistent with the T3D, namely, 300 MBytes/sec. Different values of sending and receiving latencies, which together constitute startup latency, were used. As in the other simulations, sending (and receiving) latencies are simulated serially.

Figure 3.25(a) compares the broadcast latency of the EDN algorithm and the RD algorithm for meshes of size $4 \times 4 \times 4$ and $8 \times 8 \times 4$. The message size is varied

from 32 bytes to 2048 bytes. In this plot, the startup latency is 1.5 microseconds (0.75 for sending, and 0.75 for receiving). The advantage of the EDN algorithm is approximately 30 percent in the larger mesh and 27 percent in the smaller mesh. One may conclude that for tasks such as distribution of initial data and replication of arrays, the EDN algorithm is preferred to a recursive doubling approach. Even though the latter will inadvertently take advantage of a multi-port architecture, the EDN approach achieves better performance due to increased parallelism among the constituent messages.



(a) startup latency equals 1.5μsecs (b) startup latency equals 0.15μsecs

Figure 3.25: Comparison of EDN broadcast with RD broadcast in 3D meshes (simulation)

Reducing startup latency is a major goal of many ongoing research projects. For comparison, Figure 3.25(b) plots the broadcast latency of the EDN algorithm and the RD algorithm in the same meshes, except that the startup latency is decreased by an order of magnitude. Specifically, the sending latency is 0.075 microseconds, as is the receiving latency. Again, the message size is varied from 32 bytes to 2048 bytes. The broadcast times of both algorithms are reduced, but the relative advantage of the EDN algorithm increases (36 percent and 32 percent, respectively,

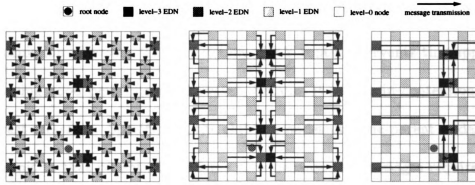


Figure 3.26: First three steps of reduction in a 16×16 2D mesh

in the large and small meshes). A lower startup latency increases the overlap among messages messages sent from the same node, and the EDN algorithm is better able to exploit this characteristic.

3.5 Reduction and Gather Operations

Besides broadcast, the EDN approach can be applied to collective operations such as reduction and gather [104]. In *reduction*, a commutative and associative operation, such as **maximum**, is performed on a set of data items that may be distributed across network nodes. In *gather*, one process receives a message from each of a group of processes. These operations, also known as global combine, have been studied for one-port meshes, for which at least $\lceil \log N \rceil$ message-passage steps are needed to complete the operation [159, 15].

EDN-based algorithms are designed to reduce this time by taking advantage of all-port network interfaces. Since the communication patterns of such operations are typically *many-to-one*, it may appear that an EDN broadcast tree, described

above, could be used to implement such operations with the direction of message transmission reversed. The restriction of XY routing prevents this approach from achieving good performance, however, because it results in contention among the unicast messages. Specifically, the children nodes of some parent nodes cannot all transmit to that parent at the same time because the messages require a common channel.

The solution lies in rotating the pattern of EDN nodes by 90 degrees clockwise or counter-clockwise, as shown in Figure 3.26. In the first step, all level-0 nodes send the message to their neighbor that is an EDN. After three steps, the four level-3 EDNs have received the data from their children. Two more *completion* steps are required to deliver the data to the root node.

Like broadcast, the total number of steps in EDN-based reduction consists of two parts, the steps in which messages are passed up the reduction tree, and the completion steps. Tables 3.6 and 3.7, respectively, give these values for $n = 4 \times 2^k$ and $n = 5 \times 2^k$, where the number of network nodes is $N = n \times n$. The values for $n = 6 \times 2^k$ and $n = 7 \times 2^k$, though not given, are the same as for $n = 5 \times 2^k$. In fact, although the number of steps in the reduction tree and the completion process differ among the four mesh types, the totals are the same for all types.

Table 3.6: EDN reduction steps for $n = 4 \times 2^k$

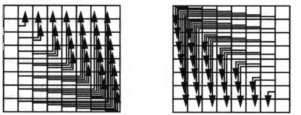
k	0	1	2	3	4	k
$n = 4 \times 2^k$	4	8	16	32	64	4×2^k
nodes N	16	64	256	1024	4096	$n \times n$
tree height	1	2	3	4	5	$k - 1$
min compl.	2	2	2	2	2	2
max compl.	2	3	3	3	3	3
min total	3	4	5	6	7	$k + 3$
max total	4	5	6	7	8	$k + 4$

Table 3.7: EDN reduction steps for $n = 5 \times 2^k$

k	0	1	2	3	4	k
$n = 5 \times 2^k$	5	10	20	40	80	5×2^k
nodes N	25	100	400	1600	6400	$n \times n$
tree height	2	3	4	5	6	$k + 2$
min compl.	1	1	1	1	1	1
max compl.	2	2	2	2	2	2
min total	3	4	5	6	7	$k + 3$
max total	4	5	6	7	8	$k + 4$

3.6 Matrix Transposition

We can also apply the EDN approach to the problem of transposing a square matrix $M_{m \times m}$. We assume that the elements of the matrix are distributed evenly among a 2D mesh processor network of size $N = n \times n$ with processor $p_{i,j}$ holding a block of submatrix of size $\frac{m}{n} \times \frac{m}{n}$, for $0 \leq i, j \leq n-1$. (In terms of High Performance Fortran [71], the matrix is decomposed in a $(block, block)$ manner.) Suppose that $b = \frac{m}{n}$ is an integer. In a *direct* implementation of transposition, depicted in Figure 3.27 for an 8×8 mesh, processor $p_{i,j}$ sends a message of length b^2 to processor $p_{j,i}$, for $0 \leq i, j \leq n-1$. Processors located in the major diagonal already hold the correct data, and do not send or receive any messages. Therefore, a total of $n^2 - n$ unicast messages are transmitted. Although all the messages are *theoretically* transmitted in one step, under XY routing, channel contention is a major drawback of this approach. As depicted in Figure 3.27, only two of the messages do not have to compete for at least one channel. In the EDN approach to transposition, although multiple message-passing steps are involved, contention among messages can be largely avoided, thereby reducing the latency of the operation.



(a) low to high (b) high to low

Figure 3.27: Channel contention problem in direct approach to matrix transposition

The basic EDN pattern used for transposition is shown in Figure 3.28(a) for

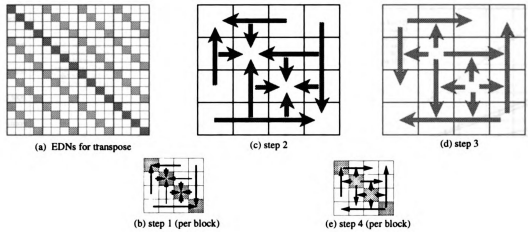
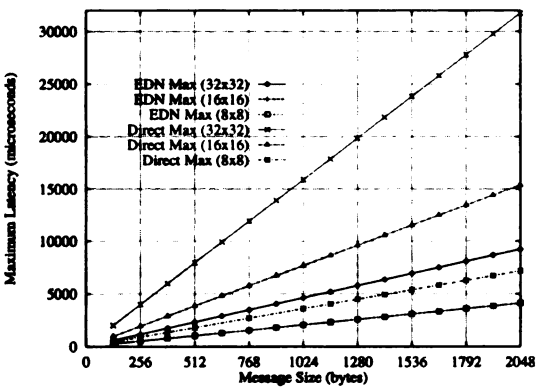


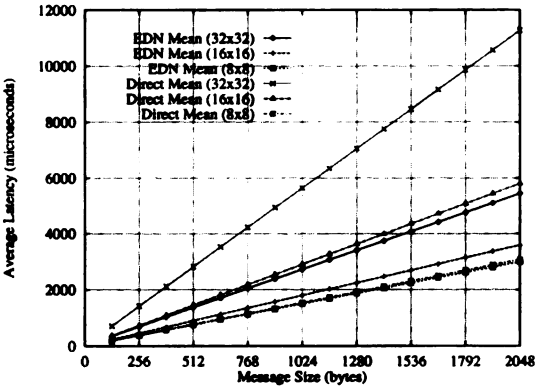
Figure 3.28: EDN matrix transposition in 16×16 mesh.

the 16×16 mesh. In order to implement transposition, the mesh is "divided" into 4×4 blocks. In the first message-passing step, nodes within each 4×4 block send their data to the appropriate EDN in that block, as shown in Figure 3.28(b). In the second step, corresponding EDNs on the diagonals in separate blocks send messages as illustrated in Figure 3.28(c). The third and fourth steps are similar to the second and first, respectively, except that the direction of the messages is reversed. After four steps, the transpose operation is complete, while avoiding channel contention.

We observe that the communication patterns in Figures 3.28(c) and (d) are identical to the communication pattern in Figure 3.28(b). That is, the "basic blocks" of this transpose pattern can be of size $((4 \times 4)^j \times (4 \times 4)^j)$, for $j \geq 0$. In general, transpose for a network of size $2^k \times 2^k$ requires k message-passing steps. For networks of size $2^k \times 2^k$, where k is even, this approach produces no channel contention for any communication steps throughout the process of transposition. However, when we use the same approach for a network of size $2^k \times 2^k$, where k is odd,



(a) maximum latency



(a) average latency

Figure 3.29: Comparison of EDN transpose with direct transpose

channel contention is possible in one of the transposition steps. Some messages in those blocks along the major diagonal may begin transmission at the same time that other messages from off-diagonal blocks are being sent across the network to other off-diagonal blocks. The amount of contention, though, is small as compared with the direct approach. Specifically, the number of messages involved in channel contention is $\frac{3n}{4}$, compared to $(n^2 - n - 2)$ in the direct approach.

In order to compare the transposition algorithms while accounting for such system characteristics, a simulation study was conducted using MultiSim [100]. Figures 3.29(a) and 3.29(b), respectively, compare the maximum and average transpose latency of the EDN transpose algorithm with the direct approach for meshes of size 8×8, 16×16, and 32×32. The message size is varied from 128 bytes to 2048 bytes. The advantage of the EDN transpose algorithm is clear. In fact, the maximum latency of the EDN algorithm is less than one-third that of the direct approach in a 32×32 mesh, and the average transpose latency of the EDN algorithm is approximately one-half that of the direct approach. Thus, while the direct approach can *theoretically* be completed in a single message-passing step, the EDN approach,

which involves k steps for network of size $2^k \times 2^k$, can achieve better performance by reducing or avoiding contention among the constituent unicast messages.

3.7 Conclusions

In this chapter, a communication model has been described that uses the concept of dominating sets to efficiently implement broadcast in all-port wormhole-routed 3D mesh networks. The EDN algorithm takes advantage of the distance-insensitivity of wormhole routing, allowing a node to "dominate" another node even though the two nodes may not be neighbors in the network. In addition, the algorithm is designed to exploit the presence of multiple ports between each local processor and its router. Using the EDN algorithm, the number of message-passing steps was shown to be at most three steps above a theoretical lower bound in systems with up to 60,000 nodes. Simulation results confirm the advantages of the algorithm under typical system conditions.

Chapter 4

EDN Broadcasting in Torus Networks

In this chapter, we use the EDN model to develop broadcast algorithms for multi-port wormhole-routed torus networks. We first consider 2D networks, followed by 3D networks.

4.1 System Model

The systems under consideration in this chapter are similar to those discussed in Chapter 3. They use wormhole routing, dimension-ordered routing, and possess a multi-port architecture. However, the topologies of these systems are 2D or 3D tori. 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 consider only *bidirectional* tori, in which direct message transmission is possible in either direction between neighboring nodes. Figures 4.1(a) and 4.1(b), respectively, show the physical links associated with a 2D unidirectional torus and a 2D bidirectional torus. Also shown in each figure 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 in Figure 4.1 result from dimension-ordered routing.

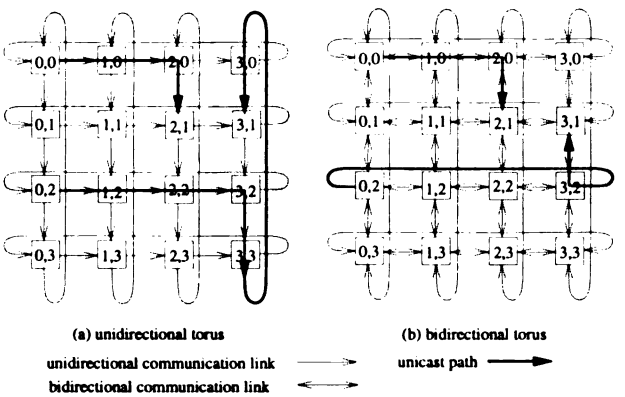


Figure 4.1: Examples of 2D 4×4 torus networks

The second distinguishing property of these networks is that they use *virtual channels* [46] in order to prevent deadlock. Each virtual channel has its own flit buffer and control lines. For wormhole routing in bidirectional torus networks, three sets of (unidirectional) virtual channels are required: *p*-channels, *l*-channels, and *h*-channels [49]. The virtual channels along a single dimension, d, of a bidirectional torus with even width, k, are illustrated in Figure 4.2; the situation is similar when k is odd. We use the notation of Dally and Seitz [49]: $c_{d\alpha x}$ represents the

virtual channel leaving node x in dimension d, in the virtual channel set α, where $\alpha \in \{p, l, h\}$.

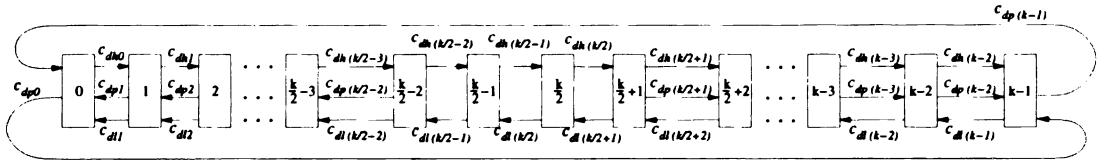


Figure 4.2: Virtual channels in one dimension of a bidirectional torus

4.2 EDN Broadcast in 2D Torus Networks

An important step in the development of an EDN broadcast algorithm is the selection of the *EDN configuration*, that is, the pattern of EDNs that will be used. We might initially consider using the EDN configuration of a 5×5 torus, since it possesses the property that only five level-1 EDNs are required to dominate the other nodes, and that all five EDNs use their entire set of outgoing channels in doing so; see Figure 4.3. However, we observe that at least three message passing steps are needed for broadcasting in a 5×5 torus under XY routing, since a single source node cannot reach the other 4 nodes in one step. We would rather base our algorithm on a more efficient "building block," such as a 4×4 torus.

As shown in Figure 4.4, it is possible to perform broadcast in a 4×4 torus in two message-passing steps under XY routing. In the first step, the source delivers the message to three level-1 EDNs, labeled A, B, and C, which together with the source deliver the message to the remaining nodes in the second step. In this case, the source is a level-2 EDN of all the nodes in the network. There are actually many EDN configurations that could be used to implement two-step broadcast in a 4×4

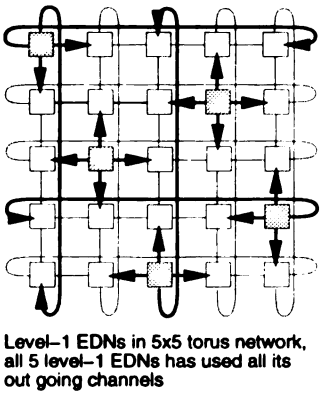


Figure 4.3: Five dominating nodes in 5×5 torus.

torus. The message-passing patterns shown in Figure 4.4 are particularly important because they can be used to design broadcast algorithms for larger networks as well. The pattern in Figure 4.4(a) is referred to as a *Y-pattern*, and the pattern in Figure 4.4(b) is referred to as a *T-pattern*.

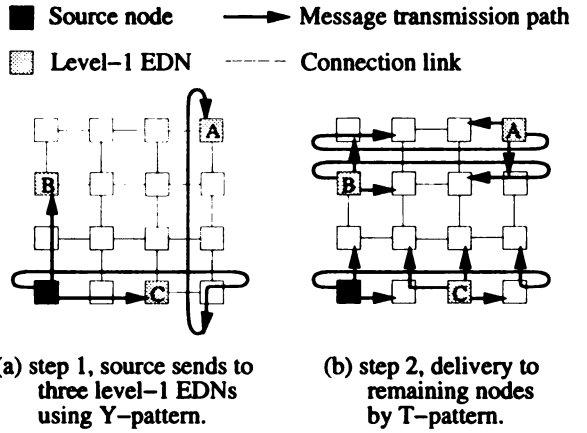


Figure 4.4: Two-step broadcast in 4×4 torus.

Figure 4.5 shows an abstract representation of the two patterns, which are used to identify the EDNs at each level for a given network. The Y-pattern is used by a level-t EDN to send a message to three level-$(t-1)$ EDNs. The T-pattern is used by four level-$(t-1)$ EDNs to send a message to twelve level-$(t-2)$ EDNs. In a torus, the paths may traverse wraparound channels. Associated with each instantiation of

a Y or T pattern is a parameter δ, which represents the relative distance between the source and destination of each message. Also, since the torus is a symmetric topology, exactly the same patterns can be used regardless of the location of the source node.

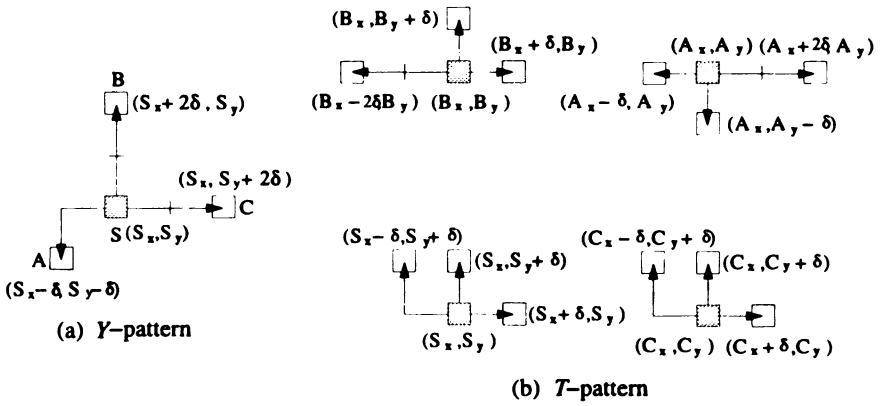
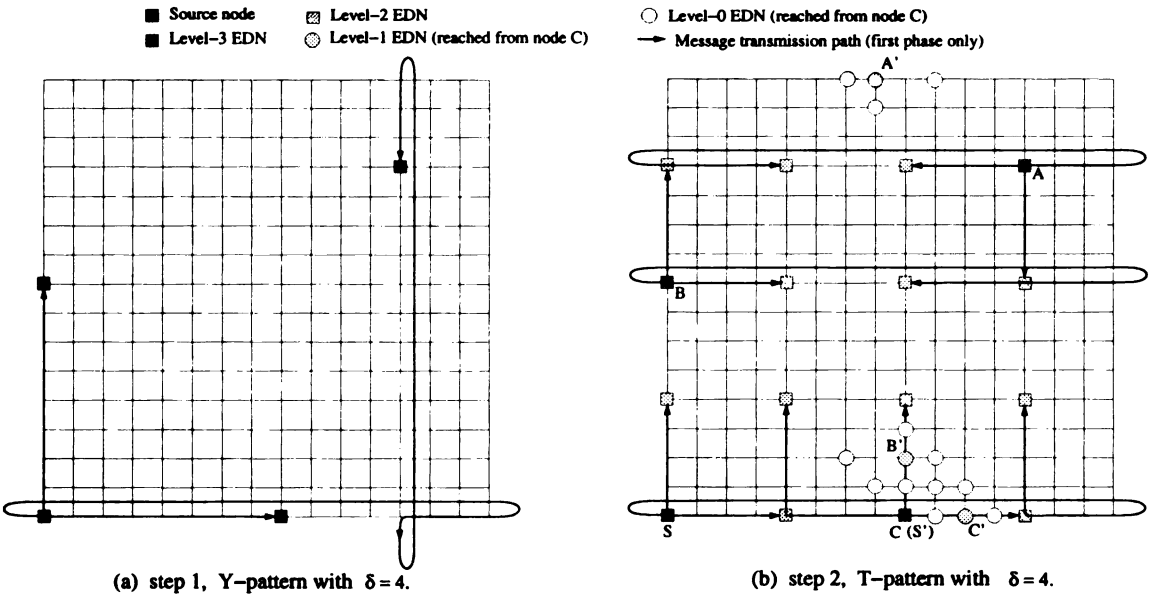


Figure 4.5: Y-pattern and T-pattern.

Given a single level-t EDN holding a message, execution of a Y-pattern followed by a T-pattern results in 16 level-t, level-$(t-1)$, and level-$(t-2)$ EDNs holding the message. These 16 nodes form a 4×4 grid and are equidistant from one another in each dimension, that distance being δ. We can therefore define a broadcast operation in a 4×4 torus as $\{YT\}$ with parameter $\delta = 1$. Let us refer to the combined $\{YT\}$ transmission pattern as a *phase*. For example, Figure 4.6 shows the first phase of an EDN broadcast operation in a 16×16 torus. The parameter δ for both steps is $16/4 = 4$. A second $\{YT\}$ phase, with $\delta = 1$, is required to complete the broadcast. We should emphasize that each $\{YT\}$-pattern in the second phase does not necessarily operate within a square 4×4 subnetwork, as illustrated in Figure 4.4. Rather, the patterns match those in Figure 4.5, with wraparound channels used as necessary. Figure 4.6 shows three level-1 EDNs and

the twelve level-0 EDNs that will be reached from node C in the second phase.



(a) step 1, Y-pattern with $\delta = 4$.

(b) step 2, T-pattern with $\delta = 4$.

Figure 4.6: First phase (two steps) of EDN broadcast in a 16×16 torus.

Figure 4.7 gives the general algorithm for EDN broadcast in a 2D 4×4 torus network; the algorithm is invoked by all nodes in parallel. The parameter δ is calculated from the size of the network. If the node is not the source, then it must first receive the message, and may be required to complete the T step of a $\{YT\}$ phase. Following this action, or if the node is the source itself, the node repeatedly sends messages according to the $\{YT\}$ pattern, reducing the value of δ by a factor of 4 following each phase. Each message carries two variables identifying the message type and the value of parameter δ. The message type can be one of Y_a, Y_b, Y_c, or T, which dictates to the receiving node which role it must play the subsequent message-passing step. Procedure `T_pattern(type,`δ`)`, given in Figure 4.8, generates three messages corresponding to the `type` parameter.

In order to implement broadcast in a network of size $(2 \times 4^k) \times (2 \times 4^k)$, we

```
Procedure EDN_bcast()          /* local address is (X,Y) */

  δ = 4^(k-1)
  if ( I_am_not_source )
    call msg_recv(type,δ);
    switch(type)
      case Y_a:  call T_pattern(Y_a,δ); break;
      case Y_b:  call T_pattern(Y_b,δ); break;
      case Y_c:  call T_pattern(Y_c,δ); break;
      case T:  break;
    end switch
    δ = δ/4
  end if
  while ( δ >= 1 )
    send {Y_a,δ} to node (X-δ,Y-δ); send {Y_b,δ} to node (X,Y+2δ);
    send {Y_c,δ} to node (X+2δ,Y);
    call T_pattern(Y_s,δ);  δ = δ/4 ;
  end while
```

Figure 4.7: Main EDN broadcast routine for a $4^k \times 4^k$ torus network.

```
Procedure T_pattern(type, δ)          /* local address is (X,Y) */

  switch (type)
    case Y_a:
      send {T,δ} to node (X+2δ,Y); send {T,δ} to node (X,Y-δ);
      send {T,δ} to node (X-δ,Y); break
    case Y_b:
      send {T,δ} to node (X-2δ,Y); send {T,δ} to node (X,Y+δ);
      send {T,δ} to node (X+δ,Y); break
    case Y_c or Y_s:
      send {T,δ} to node (X-δ,Y+δ); send {T,δ} to node (X,Y+δ);
      send {T,δ} to node (X+δ,Y); break
  end switch
```

Figure 4.8: Procedure T_pattern().

need to define a new message-transmission pattern. The *F-pattern* is shown in Figure 4.9. As mentioned above, after executing a $\{YT\}$ pattern of transmission from any source node, three levels of EDNs form a 4×4 grid with some internode spacing, call it δ_1. If each of these 16 EDNs executes an F-pattern transmission with parameter $\delta_2 = \delta_1/2$, the resulting 64 EDNs form an 8×8 grid with adjacent nodes spaced at distance δ_2.

The canonical example of F-pattern usage is broadcast in an 8×8 torus, illustrated in Figure 4.10. The entire broadcast procedure is denoted as $\{YTF\}$. The parameter δ for both the Y and T patterns is $8/4 = 2$, while $\delta = 1$ for the F-pattern. The operation of the F-pattern is illustrated in Figure 4.10(c), where it is used by each of the 16 EDNs holding the message to send to the remaining nodes in the network. The algorithm in Figure 4.7 can be extended easily to handle the networks of size $(2 \times 4^k) \times (2 \times 4^k)$ by defining an F-message type and by having all nodes that have not received an F message, invoke an F-pattern as their last action.

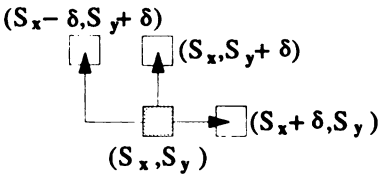


Figure 4.9: F-pattern of message transmission.

Theorem 1 *The EDN broadcast algorithm delivers a message exactly once to every node in a* $4^k \times 4^k$ *2D torus network in* $2k$ *message-passing steps, where* $k \geq 1$.

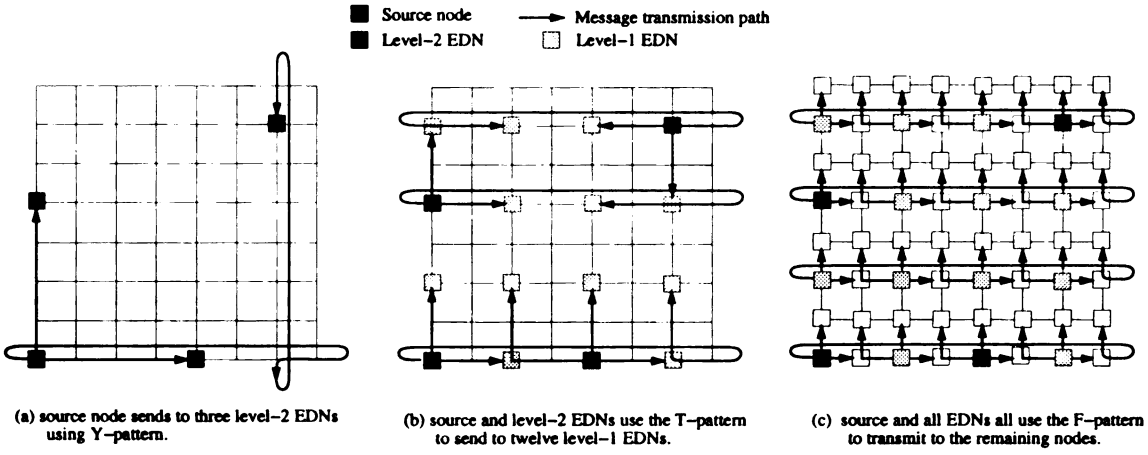


Figure 4.10: EDN broadcast in 8×8 torus, $\delta = 2$ for both Y and T patterns, and $\delta = 1$ for F-patterns.

Proof: The proof is by induction on k. For $k = 1$, the theorem holds by observation; Figure 4.4 shows the $\{YT\}$ pattern used to implement two-step broadcast in a 4×4 torus. Assume that the result is true for $k = \ell$, $\ell \geq 1$, that is, the EDN broadcast algorithm can reach every node in a $4^\ell \times 4^\ell$ torus in 2ℓ steps. Clearly, the same algorithm can be used to reach every fourth node (in each direction) in a $4^{\ell+1} \times 4^{\ell+1}$ torus by starting `EDN_bcast()` with $\delta = 4^\ell$ instead of $\delta = 4^{\ell-1}$.

An additional invocation of the $\{YT\}$ pattern (two steps) can be used to reach the remaining nodes in the $4^{\ell+1} \times 4^{\ell+1}$ torus. This can be seen by considering the the pattern of nodes reached by the $\{YT\}$ pattern, which is depicted in Figure 4.11(a). The source node is indicated by an open circle. As shown in Figure 4.11(c), this pattern can cover a plane, with the pattern repeating every four nodes in both directions. Therefore, after $2(\ell + 1)$ total steps, every node in the $4^{\ell+1} \times 4^{\ell+1}$ torus has received the message at least once. Finally, no node receives the message more than once, since in every step of the algorithm, every node holding the message sends it to 3 other nodes, so at most $4^{2(\ell+1)}$ nodes hold

the message after $2(\ell + 1)$ steps. ■

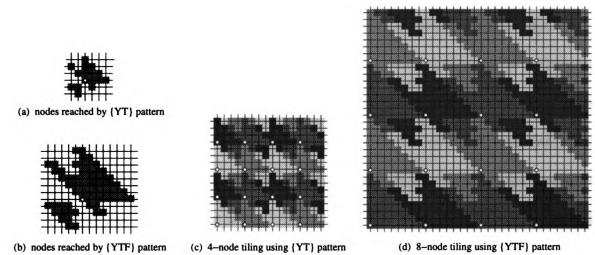


Figure 4.11: Tilings using $\{YT\}$ and $\{YTF\}$ patterns.

The pattern of nodes reached by the $\{YTF\}$ pattern is depicted in Figure 4.11(b). As shown in Figure 4.11(d), this pattern also can cover a plane, repeating every eight nodes in both directions. By a similar argument to that used in the proof of Theorem 1, we have the following corollary. Combining the two results yields Corollary 2.

Corollary 1 *The EDN broadcast algorithm delivers a message exactly once to every node in a* $(2 \times 4^k) \times (2 \times 4^k)$ *2D torus network in* $2k + 1$ *message-passing steps, where* $k \geq 1$.

Corollary 2 *The EDN broadcast algorithm requires* d *message-passing steps to deliver a message exactly once to every node in a* $2^d \times 2^d$ *torus.*

Due to the symmetry of the torus, the number of steps and patterns involved in a broadcast operation are independent of the location of the source node. Here,

we have considered only networks built from 4×4 networks. However, the EDN approach can be applied to torus networks of other shapes and sizes. Various patterns, applicable to both mesh and torus networks, are described in [143].

Besides the number of message-passing steps, an issue that must be addressed is channel contention among constituent messages. Two types of contention have been studied for collective communication in wormhole-routed direct networks [103]. *Stepwise contention* occurs when two or more messages sent in the same message-passing step contend for the same channel. In the absence of multiple virtual channels [46] for each unidirectional physical channel, one of the messages will be blocked until the other has relinquished the channel. The presence of virtual channels allows the messages to share the physical channel in a round-robin manner, though the resulting rate of progression of each will be indirectly proportional to the number of messages sharing the channel. The second type of contention is *depth contention*, which occurs when messages sent in two *different* communication steps may actually be transmitted concurrently. This skewing of message-passing steps can be caused by the delays at nodes, such as sending latency, receiving latency, and other computing tasks that delay the handling of messages [103].

Theorem 2 *The EDN broadcast algorithm for a $4^k \times 4^k$ 2D torus network is stepwise contention-free.*

Proof: EDN broadcast for a $4^k \times 4^k$ torus can be denoted as $\{YT\}_{\delta_1}\{YT\}_{\delta_2}\{YT\}_{\delta_3}\ldots\{YT\}_{\delta_k}$, where $\delta_m = 4^{k-m}, m = 1, 2, \ldots, k$. To prove that stepwise contention does not occur, we need only show that no two messages trans-

mitted in step $\{Y\}_\delta$, for $\delta \in \{\delta_i\}_{i=1}^k$, require the same channel, and that no two messages transmitted in step $\{T\}_\delta$, for $\delta \in \{\delta_i\}_{i=1}^k$, require the same channel.

By definition, the three messages sent by a single node $\{Y\}_{\delta_i}$ do not use a common channel. Consider step $\{Y\}_{\delta_i}$, $i > 1$, which follows step $\{T\}_{4\delta_i}$. All nodes holding the message are located at distance 4δ from one another. In the $\{Y\}_{\delta_i}$ step, each such node (x, y) sends messages to nodes $(x - \delta_i, y - \delta_i)$, $(x, y + \delta_i)$, and $(x + \delta_i, y)$. Since all channels used by these messages are within distance $2\delta_i$ of node (x, y), none of these messages may contend with the messages sent by a node that is distance $4\delta_i$ from (x, y). As for step $\{T\}_{\delta_i}$, $i \geq 1$, inspection of the tiling in Figure 4.11(c) shows that the T-pattern messages in neighboring regions do not require any common channels. ■

Corollary 3 *EDN broadcast in a 2D torus network of size* $(2 \times 4^k) \times (2 \times 4^k)$ *is stepwise contention-free.*

Proof: The algorithm requires a total of $2k + 1$ steps. The first $2k$ steps can be denoted as $\{YT\}_{\delta_1}\{YT\}_{\delta_2}\{YT\}_{\delta_3} \ldots \{YT\}_{\delta_k}$, where $\delta_m = 2 \times 4^{k-m}, m = 1, 2, \ldots, k$. The last step is a $\{F\}_{\delta=1}$ pattern. From the proof of Theorem 2, the first $2k$ steps are stepwise contention-free. After step $\{T\}_{\delta_k}$, all nodes holding the message are located at distance 2 from one another in each dimension. The three destination nodes of the $\{F\}_{\delta=1}$ messages sent from node (x, y) are located at $(x - 1, y + 1), (x, y + 1)$, and $(x + 1, y)$. Therefore, no channel is traversed by more than one message in $\{F\}_{\delta=1}$, completing the proof. ■

Besides stepwise contention, it can be shown that the EDN broadcast algorithm also avoids contention among messages sent in neighboring steps. Although depth contention is possible under certain conditions, simulation results presented in Section 3.4.3 indicate that the effect of such contention is likely to be small under conditions representative of commercial systems.

4.3 EDN Broadcast in 3D Torus Networks

We now describe the procedure for implementing EDN broadcast in 3D torus networks. As shown in Figure 4.12(a), a source node located in position (x, y, z) in a network of size $4 \times 4 \times 7$ can simultaneously send a message to six other nodes, one located in each of the XY planes other than its own. The six nodes are: $(x, y, z+3), (x+1, y, z+2), (x, y, z-2), (x, y-1, z+4), (x-1, y, z-1)$, and $(x, y-1, z+1)$. The broadcast process then proceeds in a divide-and-conquer manner, with each node holding the message performing a local broadcast within its XY plane using the 2D broadcast algorithm described earlier. Because the torus topology is symmetric, this method is applicable to any source node.

For networks in which the width of the Z dimension is greater than 7, additional strategies are needed. The proposed approach is to partition the network into at most 7 *zones*, each comprising contiguous XY planes, with the XY planes in the network assigned to the zones as evenly as possible. In the first message-passing step, the source node uses the message-passing pattern in Figure 4.12 to send a

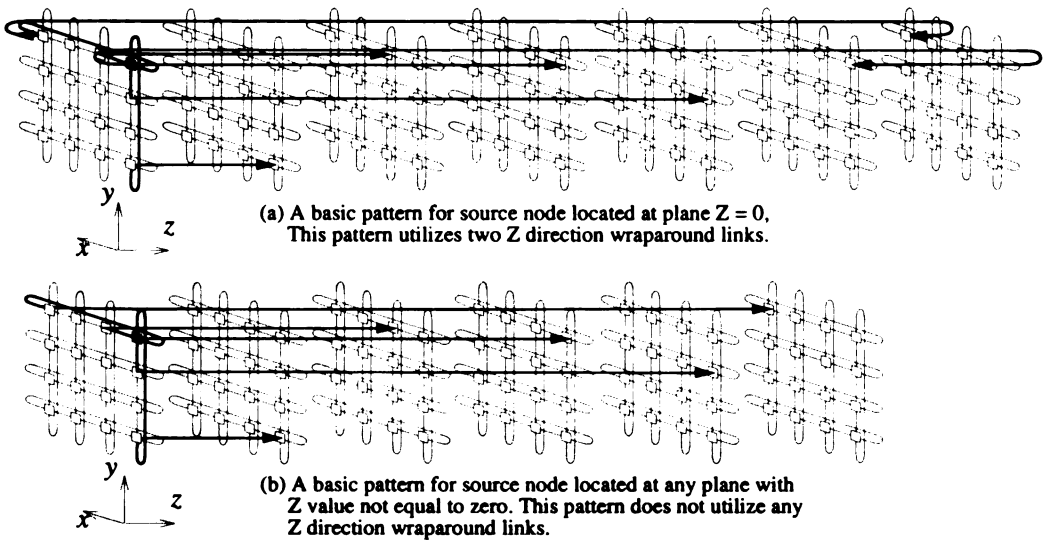


Figure 4.12: EDN configurations in 3D torus networks

copy of the message to one node in each zone. Continuing in a recursive fashion, each sub-zone is further partitioned, with each receiving node acting as a new source node. Eventually, the sub-zone is a single XY plane, in which case the node holding the message delivers it to the other nodes in the plane according to the 2D torus broadcast algorithm described earlier. In order to avoid the use of the wraparound channels in all but the first message-passing step, subsequent steps may partition a zone into at most 6 sub-zones. Figure 4.12(b) shows an example in which a single node sends to one node in each of the other planes without using wraparound channels in the Z dimension.

Theorem 3 *The EDN broadcast algorithm for 3D torus networks requires the following number of message-passing steps to deliver a message to every node in a* $2^d \times 2^d \times z$ *network, where* $d \geq 2$*: If* $z = 1$*, then the algorithm requires requires* d *message-passing steps. If* $2 \leq z \leq 7$*, then the algorithm requires requires* $d+1$ *message-passing steps. If* $7 \times 6^m + 1 \leq z \leq 7 \times 6^{m+1}$*, where* $m \geq 0$*,*

then the algorithm requires $d + m + 2$ *message-passing steps.*

Proof: In the following proof, let patterns (a) and (b) refer to Figure 4.12(a) and (b) respectively. If $z = 1$, then d steps are needed by Theorem 1 and Corollary 1. If $2 \leq z \leq 7$, pattern (a) is used in the first step to deliver the message to one node in each XY plane, after which these XY planes can then perform their own 2D torus broadcast in d steps, for a total of $d + 1$ steps. When $7 \times 6^m + 1 \leq z \leq 7 \times 6^{m+1}$, the (a) pattern is followed by $m + 1$ (b) patterns in order to distribute the message to one node in each XY plane. Including the d steps for broadcast in each plane, a total of $d+m+2$ message-passing steps is needed to reach all nodes. ■

Since the messages in the two patterns used to reach different XY planes do not produce channel contention (see Figure 4.12), and the algorithm used within each plane has already been shown to be stepwise contention-free, we have the following corollary.

Corollary 4 *EDN broadcast in a 3D torus network of size* $2^d \times 2^d \times z$, *where* $d \geq 2$ *and* $z \geq 1$, *is stepwise contention-free.*

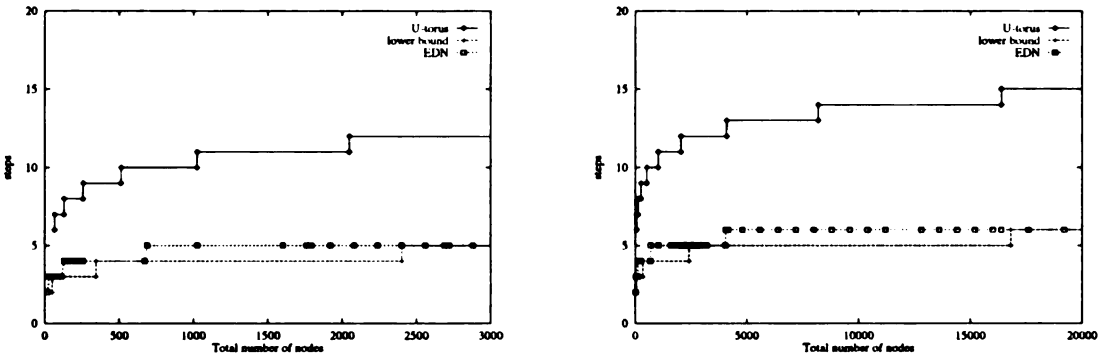
4.4 Performance Evaluation

The performance of EDN broadcast algorithm is evaluated in three ways: by comparing the number of message-passing steps to a lower bound, through timing analysis, and through simulation.

4.4.1 Number of Message-Passing Steps

A simple theoretical lower bound on the number of message-passing steps required for broadcast in an all-port n-dimensional torus is $\lceil \log_{2n+1}(N) \rceil$, where N is the total number of nodes in the network. In many cases, however, this bound may not be achievable because of the limitations imposed by deterministic routing. Specifically, it is not always possible that every node can send to $2n$ new nodes in every step of the algorithm.

Figure 4.13(a) compares the number of message-passing steps of the EDN algorithm for torus networks (both 2D and 3D) of various sizes up to 3000 nodes. The number of steps achieves the theoretical lower bound in many cases and is at most one step away from the lower bound in other cases. Figure 4.13(b) compares the number of steps for networks with up to 20,000 nodes. Again, the EDN algorithm again is close to, and in some cases equal to, the lower bound. As a reference point, both plots also show the number of steps needed by the *U-torus* algorithm [124]. The U-torus algorithm uses recursive doubling, in which the number of nodes holding the message is doubled in each step. Though optimal for one-port architectures, requiring $\lceil \log_2(N) \rceil$ steps to complete a broadcast operation in an N-node network, the U-torus algorithm will often fail to take advantage of a multi-port architecture, resulting in more message-passing steps than necessary. In fact, the number of steps is still $\lceil \log_2(N) \rceil$.



(a) medium-sized torus networks (b) large torus networks

Figure 4.13: Broadcast performance in torus networks.

4.4.2 Timing Analysis

Calculating the number of steps required is only one way to evaluate the performance of a broadcast algorithm. In order to better understand how the EDN algorithm may perform for specific values of system parameters, we analyze the time required when executed on an all-port torus. We use the following notation:

- α represents the sending latency incurred for each message.
- β is the time required to transmit a flit on a channel connecting two neighboring routers.
- γ is the receiving latency of a message.
- h denotes the number of hops, or distance, between the source and the destination of a message.
- L is the length of message, in flits.

We model the delay of a unicast message as the sum of the sending latency, the time it takes for the header flit to reach the destination node, the time for the message to be extracted from the network at the receiving node, and the receiv-

ing latency. That is, the unicast delay is $\alpha + \beta h + \beta L + \gamma$. When a node sends multiple messages in succession, we assume that their sending latencies are serialized, even though the other components may occur in parallel. Therefore, on a k-port architecture, the time needed for a node to transmit m ($m \leq k$) copies of the same message simultaneously to m different destinations, each of which is reached through a disjoint set of external channels, can be expressed as $m\alpha + h\beta + L\beta + \gamma$. This expression assumes that no channel contention occurs for any of these m outgoing messages and that the timing is measured from the beginning of the first message transmission until the last destination receives its message.

Using this formula as a basis, we can analyze the execution time of the EDN algorithm. In a Y-pattern transmission with parameter δ, for example, the last destination will receive its message at time $3\alpha + (2\delta)\beta + L\beta + \gamma$. As we have seen, for a $4^k \times 4^k$ torus, $k \geq 1$, the message transmission pattern is $\{YT\}^k$, with each stage comprising two message-passing steps. At stage i, the value of the parameter δ_i associated with the Y and T patterns is 4^{k-i}. Because $\sum_{i=1}^{k} \delta_i = \sum_{i=1}^{k} 4^{k-i} = \sum_{i=1}^{k} 4^{i-1} = (4^k - 1)/3$, after summing the times required for all stages, the total execution time of the EDN algorithm, $\mathcal{L}_{EDN}$, is computed as follows:

$$\begin{aligned} \mathcal{L}_{EDN} &= 2\sum_{i=1}^{k}\{3\alpha + \gamma + \beta(L + 2\delta_i)\} \\ &= 6k\alpha + 2k\gamma + 2kL\beta + \frac{4(4^k - 1)}{3}\beta \ . \end{aligned}$$

In other words, for a $2^d \times 2^d$ torus, $\mathcal{L}_{EDN} = 3d\alpha + d\gamma + dL\beta + \frac{2(2^{d+1}-2)}{3}\beta$, when d is even. Now let us consider a torus of size $(2 \times 4^k) \times (2 \times 4^k)$, in which the

broadcast pattern is $\{YT\}^k F$. There are k stages, plus one extra message-passing step. For stage i, the parameter $\delta_i = 4^{k-i-1}/2$. Because $\sum_{i=1}^{k} \delta_i = \sum_{i=1}^{k} 4^{k-i+1}/2 = \frac{1}{2}\sum_{i=1}^{k}(4^k) = 2(4^k - 1)/3$, we compute $\mathcal{L}_{EDN}$ for this case to be:

$$\begin{aligned}\mathcal{L}_{EDN} &= 2\sum_{i=1}^{k}\{3\alpha + \gamma + \beta(L + 2\delta_i)\} + 3\alpha + \gamma + \beta(L+2) \\ &= 3(2k+1)\alpha + (2k+1)\gamma + (2k+1)L\beta + \frac{2(4^{k+1}-1)}{3}\beta\ .\end{aligned}$$

To summarize, the time needed for EDN broadcast in a $2^d \times 2^d$ torus can be expressed as

$$\mathcal{L}_{EDN} = \begin{cases} 3d\alpha + d\gamma + dL\beta + \frac{2(2^{d+1}-2)}{3}\beta & , d \text{ is even} \\ 3d\alpha + d\gamma + dL\beta + \frac{2(2^{d+1}-1)}{3}\beta & , d \text{ is odd}\ . \end{cases}$$

Combining these two expressions, we obtain:

$$\mathcal{L}_{EDN} = 3d\alpha + d\gamma + dL\beta + \frac{2(2^{d+1} - 2 + \mathrm{mod}(d,2))}{3}\beta\ . \tag{4.1}$$

For comparison, the execution time of the U-torus algorithm for a $X \times Y$ torus can be derived in much the same way (see [124] for details of the algorithm). Assuming that both X and Y are powers of two, in the X dimension, $\lceil \log_2(X) \rceil$ message-passing steps are required to perform broadcast. At step i, the number of hops traversed by the message is $(X/2^i)$. The same argument can also be applied to the Y dimension. Because $\sum_{i=1}^{\lceil \log_2(X) \rceil} \frac{X}{2^i} = X - 1$, the total delay for broadcast

using the U-torus algorithm $\mathcal{L}_{U\text{-}torus}$ can then be calculated as follows:

$$\begin{aligned}\mathcal{L}_{U\text{-}torus} &= \sum_{i=1}^{\lceil \log_2(X) \rceil} \beta(\frac{X}{2^i}) + \sum_{i=1}^{\lceil \log_2(Y) \rceil} \beta(\frac{Y}{2^i}) \\ &\quad + \lceil \log_2(X) \rceil (\alpha + \gamma + \beta L) + \lceil \log_2(Y) \rceil (\alpha + \gamma + \beta L) \\ &= (\alpha + \beta L + \gamma)(\lceil \log_2(X) \rceil + \lceil \log_2(Y) \rceil) + \beta(X + Y - 2) \ .\end{aligned}$$

We rewrite the U-torus expression for $2^d \times 2^d$ torus as:

$$\mathcal{L}_{U\text{-}torus} = 2d\alpha + 2d\gamma + 2dL\beta + 2(2^d - 1)\beta \ . \tag{4.2}$$

In order to discover when the EDN algorithm will execute faster than the U-torus algorithm, we can set $\mathcal{L}_{\text{EDN}} \leq \mathcal{L}_{U\text{-}torus}$, eliminate common terms, and compare the two sides of the inequality for different values of parameters. Doing so for Equations 4.1 and 4.2 yields:

$$d\alpha \leq d\gamma + dL\beta + \frac{2(2^d - 1 - \text{mod}(d, 2))}{3}\beta \ . \tag{4.3}$$

For $d \geq 1$, the ratio $\frac{2(2^d - 1 - \text{mod}(d,2))}{3}$ is always greater than or equal to zero. Therefore, we can see immediately from this relation that, in those situations where the sending latency is less than or equal to the sum of the receiving latency and the network latency (that is, $\alpha \leq \gamma + L\beta$), the EDN algorithm will be faster than U-torus, assuming that $\beta > 0$, $L \geq 0$ and $d \geq 1$. Moreover, if $\alpha < \gamma$, then EDN will complete sooner for any message size L.

4.4.3 Simulation Study

In order to account for specific system dynamics, a simulation study was conducted using MultiSim [100]. MultiSim was used to simulate broadcasts of various-sized messages in networks of different sizes. For a given message size, a large number of different source nodes were selected at random to perform the broadcast operation. The maximum delays were measured and then averaged over these samples. The simulated channel rate was 0.5 microseconds per byte, a relatively modest rate consistent with the nCUBE-2, a wormhole-routed multicomputer with which MultiSim has previously been validated.

Figure 4.14 compares the two algorithms across different message lengths (128 bytes to 2048 bytes) in a 4×4 torus and a 32×32 torus. In Figure 4.14(a), the sending latency and the receiving latency are both set to 0. In Figure 4.14(b), the sending latency and the receiving latency are both set to 100 microseconds. In both cases, the advantage of the EDN approach is significant. Although the U-torus algorithm will sometimes happen to take advantage of the all-port architecture, the EDN algorithm achieves better performance because it is designed specifically to exploit the all-port architecture.

The results of the simulation study have been compared with the previous analysis. One reason to do so is that the analysis did not account for depth contention, while the simulation would accurately reflect any message delays due to this phenomenon. Figure 4.15 plots simulation versus analytical results for a 32×32 torus; the plots agree almost exactly. Such results indicate that, while depth contention is

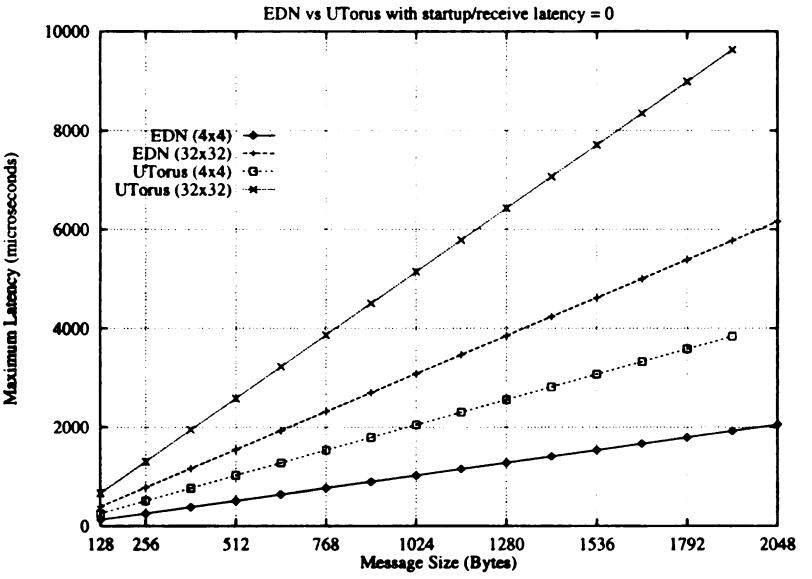


Figure 4.14: Comparison of EDN broadcast with U-torus for 4×4 torus networks.

possible in the EDN algorithm, the effect of this phenomenon may not be significant.

4.5 Conclusions

As one of the most fundamental collective communication operations, broadcast is highly demanded in parallel applications that are implemented on massively parallel computers. The EDN model is intended to formalize the concept of node domination among nodes that may be separated by multiple channels. In this chapter, an EDN-based algorithm has been described that uses the concept of dominating sets to efficiently implement broadcast in all-port wormhole-routed torus networks that use dimension-ordered routing. The performance advantage of the algorithm over recursive doubling has been evaluated through both analysis and simulation.

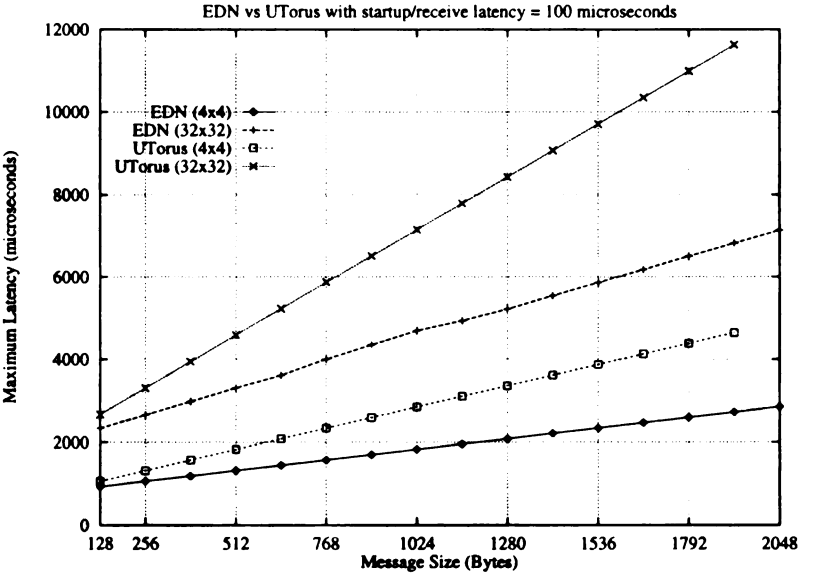


Figure 4.15: Comparison of EDN broadcast with U-torus for 32×32 torus networks.

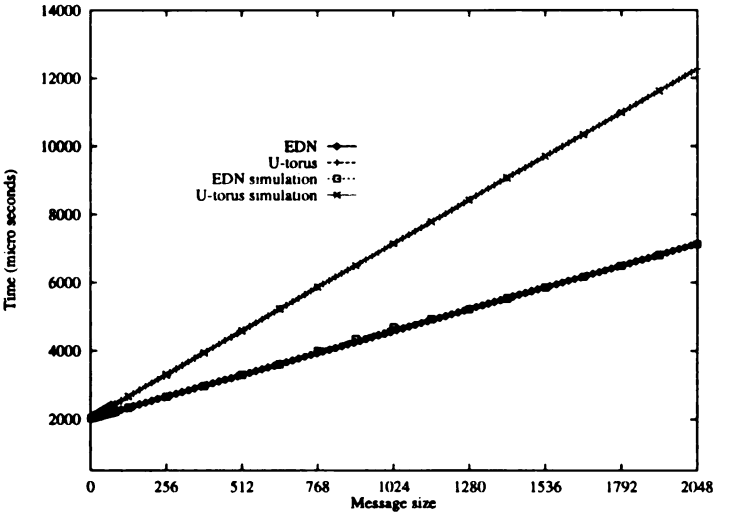


Figure 4.16: Comparison analytical and simulation results for a 32×32 torus.

Chapter 5

The τ-Model

As far as the Laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

Albert Einstein

Prediction is difficult, especially of the future.

Niel Bohr

As demonstrated in previous chapters, communication algorithms designed to exploit certain characteristics of a network architecture can improve performance. In this chapter, we introduce the τ-model, a generalized communication cost model that can assist in the comparison of different algorithms as well as different networking technologies, and the associated τ-plot, which facilitates the visualization of performance comparisons. Next, we demonstrate the use of the τ-model in evaluating the performance of wormhole broadcast algorithms, include EDN broadcast. Finally, we discuss the application of the τ-model to an increasingly popular parallel

platform, networks of workstations.

5.1 The τ-Model

In general, the point-to-point communication latency of a message can be represented as the sum of the message handling overhead and the network delay. Although this decomposition is represented in different ways in the literature, we will use in the following form:

$$T_s + T_n * L,$$

where L is the message size, T_s is the sum of software startup and receiving latencies, and T_n is the network latency per unit of message size. Empirical studies have shown, that T_s and T_n are not necessarily constants, but for a given network, depend on a number of system parameters [107, 86].

5.1.1 Generalized Latency Formula

The first part of the τ-model is a generalized latency formula describing both point-to-point and collective communication operations. Instead of explicitly asserting a limited number of parameters in the cost model, the formula incorporates all the factors that may have any effect on performance. The τ-model uses the following formula to represent the latency of a broad range of communication operations:

$$T = T_s(a_0, a_1, a_2, \ldots, a_{r-1}) + T_n(a_0, a_1, a_2, \ldots, a_{r-1}) * f(L) \qquad (5.1)$$

Although this formula appears rather complex, we can summarize it as the sum of a message-length independent term and a message-length dependent term. The parameters $\{a_i\}_{i=0}^{r-1}$ are specific to the underlying hardware and software of the system. They typically include such items as the software startup latency, reception latency, network bandwidth, communication network distance, system load, degree of network sharing, communication group size, and so on [89, 43]. Their values depend on the system configuration and network conditions under which the communication algorithm is executed. The function $f(L)$ satisfies the property that $f(0) = 0$ and is usually a non-decreasing, continuous polynomial function. The function $f(L)$ determines the *$\mathcal{L}$-complexity* of the algorithm. If $f(L) = L$, which is the most common situation, then we say that the algorithm is of *linear* $\mathcal{L}$-complexity.

5.1.2 The τ-Plot: Comparing Networks

The second part of the τ-model is a two-dimensional Cartesian plot that allows us to characterize and to compare different performance curves more easily than using traditional methods. From the above generalized latency formula, we define the parameter τ to be:

$$\tau = \frac{T_n(a_0, a_1, a_2, \ldots, a_{n-1})}{T_s(a_0, a_1, a_2, \ldots, a_{n-1})}. \tag{5.2}$$

The ratio τ is the reciprocal of the metric $m_{1/2}$, which in Hockney's model is the message length required to achieve half the asymptotic bandwidth [72].

Plotting the performance of a communication algorithm according to its coordinates (τ, T_s), results in a *τ-plot* for that algorithm. The use of the τ-plot is illustrated in the following analysis. Shown in Figure 5.1(a) are three lines marked 1, 2 and 3. Suppose that they represent measurements of the same communication operation for three different networks. Their individual latency formulas are $T_{s1} + T_{n1}L$, $T_{s2} + T_{n2}L$, and $T_{s3} + T_{n3}L$, respectively. Let us assume that $T_{s1} = T_{s2}$. In Figure 5.1(b), points 1 and 2 represent lines 1 and 2, respectively. They have the same T_s coordinate, but point 2 lies to the right of point 1, indicating that the slope of line 2 is greater than that of line 1 in original latency plot. Line 3 has a different T_s value and a different T_n value as compared with line 1. However, in the τ-plot, they have the same τ coordinate, indicating that the total time required to finish the operation in network 3 is a constant multiple of that in network 1, a relationship that is not obvious from the original latency plot. We can use this relationship to predict the performance of a communication operation in network 1 from its performance in network 3.

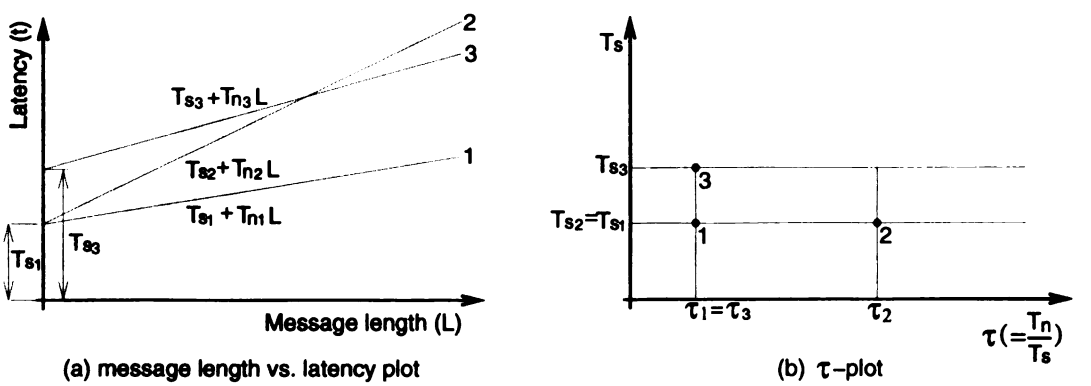


Figure 5.1: Example of the τ-plot.

This example illustrates the use of τ-plot for linear algorithms, that is, where the

algorithms discussed all exhibit linear $\mathcal{L}$-complexity. In the following, we demonstrate the use of the τ-plot for the more general communication cost model.

5.1.3 Use of the τ-Plot to Compare Algorithms

The use of the τ-plot to compare the performance of different communication algorithms under the same network conditions is illustrated in the following example. Suppose that three communication algorithms 1, 2, and 3, are described by the latency formulas $T_1(L) = T_{s1} + T_{n1} * f_1(L)$, $T_2(L) = T_{s2} + T_{n2} * f_2(L)$, and $T_3(L) = T_{s3} + T_{n3} * f_3(L)$, respectively. Let us further assume that these three algorithms have the same $\mathcal{L}$-complexity, that is, $f_1 \equiv f_2 \equiv f_3 \equiv f(L)$. Let τ_1, τ_2 and τ_3 be their respective τ-values; the τ-plot of these three algorithms is also represented by Figure 5.1(b).

Let us now derive the formal relationships among these three algorithms. The relationship between the performances of Algorithms 1 and 3 can be characterized as follows. Because $\tau_1 = \tau_3$ and $f_1 \equiv f_3$, we have, $T_{n1}/T_{s1} = T_{n3}/T_{s3}$. The latency formula of $T_1(L)$ can be rewritten as $T_1(L) = \frac{T_{s1}}{T_{s3}} * T_3(L)$.

The relationship between Algorithms 1 and 2 can be calculated from the fact that $T_{s1} = T_{s2}$ and $\tau_2 > \tau_1$. We can then derive the following equation:

$$T_2 - T_1 = \left(\frac{\tau_2}{\tau_1} - 1\right) T_{n1} f(L) \tag{5.3}$$

$$= \left(1 - \frac{\tau_1}{\tau_2}\right) T_{n2} f(L). \tag{5.4}$$

The relationship between Algorithms 2 and 3 can be derived directly by taking

the difference, $T_3 - T_2 = T_{s3} - T_{s2} + (T_{n3} - T_{n2})f(L)$. If $T_{n3} \neq T_{n2}$, then a specific length L^* can be defined as:

$$L^* = f^{-1}(\frac{T_{s2} - T_{s3}}{T_{n3} - T_{n2}}) \tag{5.5}$$

$$= f^{-1}(\frac{T_{s2} - T_{s3}}{\tau_3 T_{s3} - \tau_2 T_{s2}}). \tag{5.6}$$

Thus, we know that:

$$\begin{cases} T_3 > T_2, & \text{when } L > L^* \\ T_3 = T_2, & \text{when } L = L^* \\ T_3 < T_2, & \text{when } L < L^* \end{cases} \tag{5.7}$$

Because $T_{s3} > T_{s2}$, $f(0) = 0$ by definition, and f is an increasing function, if $T_{n2} > T_{n3}$, then $L^* > 0$, and if $T_{n2} < T_{n3}$, then $L^* < 0$. Since message size cannot be negative, if $T_{n2} < T_{n3}$, then for all message sizes $L > 0$, $T_2 < T_3$.

To summarize, the performance of a particular communication operation under a specific communication architecture is identified as a single point in the τ-plot, shown as point x in Figure 5.2. Four regions, marked by A, B, C, and D, can be identified by drawing two intersecting lines through point x, one line parallel to the horizontal axis and the other parallel to the vertical axis. Any other communication operation that falls in region A has better performance than point x, and point x outperforms any communication operation in region B. Algorithms in regions C and D can have better or worse performance than point x, depending on the message length. The crossover message length is calculated from Equation 5.7. In general,

algorithms in region C favor long messages as compared to algorithm x, and those in region D favor short messages as compared to algorithm x.

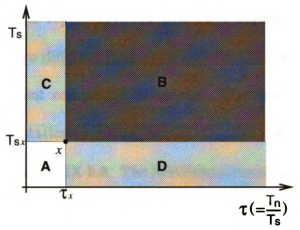


Figure 5.2: Comparing algorithms using the τ-plot.

5.2 Characterization of Point-to-Point Communication

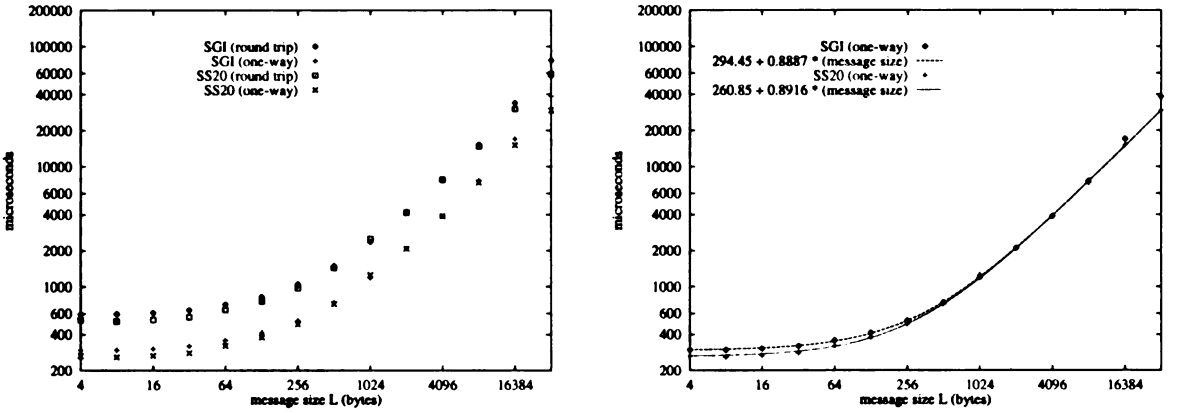
As mentioned earlier, using the τ-model, two kinds of comparisons can be made. The first is to compare different networks, which is made possible by identifying some communication operation that exhibits the same $\mathcal{L}$-complexity in these networks. The second is a quantitative comparison of different implementations of the same operation in a given network. In the same network, the system and algorithmic parameters $\{a_i\}_{i=0}^{r-1}$ are identical for all implementations of the operation. It is also true that the different implementations of the same operation usually exhibit the same $\mathcal{L}$-complexity in the same network architecture.

Point-to-point communication is one operation that exhibits same $\mathcal{L}$-complexity

for different networking technologies. The traditional way to measure point-to-point communication latency is the so-called *ping-pong* benchmark, in which two nodes exchange messages repeatedly. To provide a simple illustration of the τ-model, we conducted two different ping-pong experiments using the same network architecture, but different computing platforms. The first experiment uses two SPARCstation-20 workstations running Solaris 2.5, and the second uses two SGI Indy workstations running IRIX 5.3. The SPARCstation-20 has 32 MBytes of main memory and a clock rate of 75 MHz, whereas the Indy is running at 180 MHz and has a main memory size of 64 MBytes. According to the SPEC benchmark, the SPARCstation-20 has a SPECint95 rating of 3.11 and SPECfp95 of 3.10, while the Indy has a SPECint95 rating of 4.1 and SPECfp95 of 4.4. The results of 20,000 round-trip measurements were recorded for each specific message length, averaged, then divided by two to obtain the one-way latencies, which are plotted in Figure 5.3(a). These one-way latencies were then least-squared fitted to a straight line; the results (using a logarithmic scale) are shown in Figure 5.3(b). This method can be used to derive the values of T_s and T_n for point-to-point communication.

The ping-pong benchmark has been widely used to measure point-to-point performance, and many such results have been published in the literature [53, 86, 20]. We have used these results, and the method described above, to compute T_s, T_n and τ values for various configurations. Table 5.1 lists the results, and Figure 5.4 presents a τ-plot of the data listed in the table.

The τ-plot reveals several relationships that may not be evident in typical latency vs. message size and latency vs. bandwidth plots [53]. First, the effect of



(a) Round trip and one-way latencies.

(b) Least-squared fit one-way latencies.

Figure 5.3: Results from ping-pong test.

reducing the software and protocol processing time is reflected by a shift to the lower right in the τ-plot. This characteristic is illustrated in dramatic fashion by the relative locations of the two points marked SS20/UNet/TCP and SS20/UNet/UDP in the lower-right corner as compared to the points marked SS20/ASX200/TCP and SS20/ASX200/UDP in the upper-left corner. Although both sets of points represent measurements on Sun SPARCstation-20 workstations and a FORE ASX-200 ATM switch, the Active-Message based U-Net interface software is very effective in reducing T_s. Second, a similar shift occurs when using the same underlying physical networking technology, but faster processors. This property is exemplified in Figure 5.4 by the difference between the points SS10/Ethernet/TCP, SS20/Ethernet/TCP and SGI/Ethernet/TCP. Third, a constant factor increase/decrease in performance between networks for a particular communication operation is demonstrated in the τ-plot if their positions are located in the vicinity of a vertical line, which indicates that they have similar τ values. An example of this behavior in Figure 5.4 is the

Table 5.1: Point-to-point performance on various platforms.

system	software	$T_s(\mu s)$	T_n (μs/byte)	$\tau = T_n/T_s$	source
T3D	MAX 1.2.0.2 (SM)	3.0	0.00781	0.0026	[53]
Paragon	SUNMOS 1.6.2	25.0	0.00585	0.000234	[53]
Delta	NX 3.3.10	77.0	0.125	0.00162	[53]
iPSC/860	NX 3.3.2	65.0	0.333	0.00513	[53]
iPSC/2	NX 3.3.2	370.0	0.357	0.000965	[53]
SP-1	MPL	270.0	0.143	0.000529	[53]
SP-2	MPI	35.0	0.0286	0.000816	[53]
KSR-1	OSF R1.2.2	73.0	0.125	0.00171	[53]
CS-2	Solaris 2.3 (SM)	11.0	0.0250	0.00227	[53]
nCUBE-2	Vertex 2.0	154.0	0.589	0.00382	[53]
nCUBE-1	Vertex 2.3	384.0	2.5	0.00651	[53]
Cenju-3	End. Rel. 1.5d (SM)	34.0	0.04	0.00118	[53]
SGI	IRIX 6.1	10.0	0.0156	0.00156	[53]
CM-5	CMMD 2.0	95.0	0.111	0.00117	[53]
SS-20/ASX200	Solaris 2.4/UDP	716.0	0.16	0.000224	[86]
SS-20/ASX200	Solaris 2.4/TCP	817.0	0.13	0.000159	[86]
SS-20/Myrinet	Solaris 2.4/UDP	746.0	0.18	0.000241	[86]
SS-20/Myrinet	Solaris 2.4/TCP	807.0	0.152	0.000188	[86]
SS-20/UNet	SunOS 4.1.3/UDP	65.0	0.75	0.0115	[20]
SS-20/UNet	SunOS 4.1.3/TCP	75.0	0.85	0.0113	[20]
SS-10/Ethernet	Solaris 2.5/TCP	552.17	0.913	0.00165	*
SS-20/Ethernet	Solaris 2.5/TCP	260.85	0.892	0.00342	*
SGI Indy/Ethernet	IRIX 5.3/TCP	294.45	0.889	0.00302	*

* measured by the authors

τ values of the Paragon system and the SS-20s connected by the FORE ASX-200 ATM switch. The Intel Paragon system has a τ value of 0.00023, while the SS-20s have a τ value of 0.00022. The values of T_s for these points reveal that for point-to-point one-way communication operation, the ATM-connected SS-20s exhibit latencies that are approximately 28 times the latency of the Intel Paragon system, across the entire measured message length range. Another example pair is the CM-5 and Cenju-3, although in this case, the ratio is much smaller (2.78).

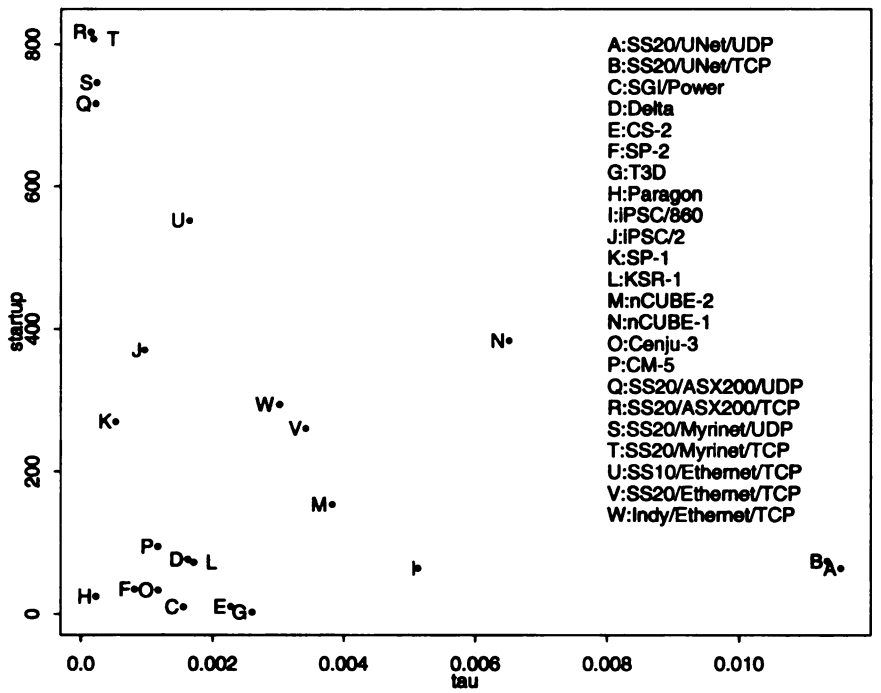


Figure 5.4: τ-plot for point-to-point communication benchmarks.

5.3 Analysis of Wormhole Broadcast Algorithms

In this section, we analyze and compare four broadcast algorithms for 2D wormhole-routed mesh networks. In Section 5.4 we use the τ-model to compare them. The algorithms under investigation are Recursive Doubling (RD) [16], Scatter Collect (SC) [17], Fibonacci Tree (FT) [50], and Extended Dominating Node (EDN) [147]. All these algorithms are based on point-to-point message passing. The following formula is used to describe the point-to-point communication latency in wormhole-routed networks. Suppose that a message traverses h hops before reaching its destination. Then the overall message latency is described by $\alpha+h\beta+\beta L+\gamma$, where α is the startup overhead, and β is the per-unit message transmission latency, and

γ is the reception overhead at the destination.

Probably the most well-known algorithm for multicast or broadcast is the Recursive Doubling algorithm. The source divides all participating nodes into two groups, the first containing itself and sends the message to one node in the second group. Those two nodes holding the message now acting as source nodes for their respective groups and proceed with the broadcasting process independently. When used as a broadcast algorithm for a wormhole-routed $2^n \times 2^n$ square mesh network, one implementation is to begin the broadcast operation with the distribution of the message along the row of the source node. Next, all nodes in that row perform the same operation along their respective columns. Figure 5.5 illustrates the operation of this algorithm in an 8×8 2D mesh. This algorithm, referred to as Dimensional Broadcast [16], is a special case of the more general U-mesh [103] multicast algorithm. For a $2^n \times 2^n$ mesh, a total of $2n$ steps is needed to complete the operation. Step i requires time $\alpha + \frac{2^n}{2^i}\beta + \beta L + \gamma$. Therefore, the overall time needed is $2n\alpha + 2\beta(2^n - 1) + 2n\gamma + 2n\beta L$.

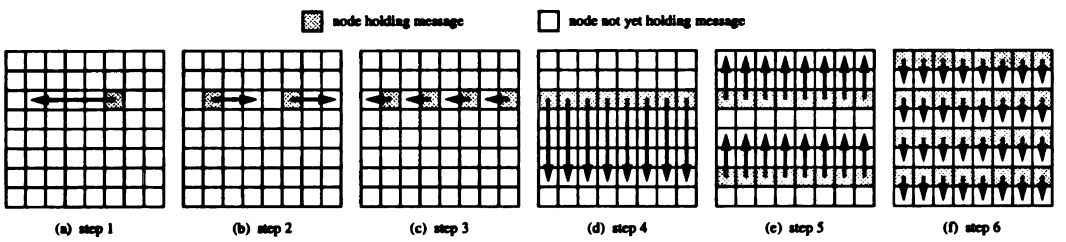


Figure 5.5: Dimensional broadcast in an 8×8 mesh.

The Scatter-Collect algorithm [17] is based on the idea of message fragmentation. The source node partitions the message and performs a scatter operation across its row, with each node receiving a particular segment of the message. Next, each node

holding part of the message performs a scatter in its column. In the third step, the nodes in each row form a logical ring and circulate segments around the ring until all nodes hold all segments in that row. Finally, a similar circulation in each column results in a copy of the entire message at every node. This algorithm avoids channel contention, and the pipelining of message segments offers good performance for broadcast of long messages [17]. Table 5.2 summarizes its performance for a $2^n \times 2^n$ mesh network.

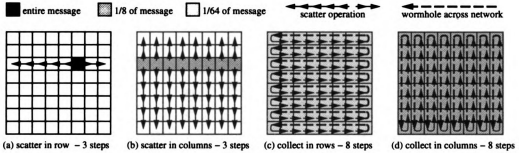


Figure 5.6: Scatter-collect broadcast operation in an 8×8 2D mesh.

Table 5.2: Analysis of Scatter Collect algorithm

scatter in row	$n * \{\alpha + \beta(2^n - 1) + \gamma + \frac{2^n-1}{2^n}\beta L\}$
scatter in column	$n * \{\alpha + \beta(2^n - 1) + \gamma + \frac{2^n-1}{2^n}\frac{\beta L}{2^n}\}$
collect in row	$(2^n - 1) * \{\alpha + \beta + \gamma + \frac{\beta L}{2^n}\}$
collect in column	$(2^n - 1) * \{\alpha + \beta + \gamma + \frac{\beta L}{2^n * 2^n}\}$

The Fibonacci Tree algorithm, proposed in [50], uses both message fragmentation and message pipelining. One requirement of the algorithm is that the tree used to broadcast is free of depth-contention [103]. As with the U-mesh algorithm, the tree is based on lexicographical ordering of nodes, which prevents depth contention. This means that messages will not contend for the same channel, even if

those messages are sent in different steps. The message is divided into k pieces and sent down the tree in a pipeline fashion. The structure of the tree depends on the parameter k and the number of nodes in the tree. Figure 5.7 illustrates the operation of the Fibonacci Tree algorithm for a partition parameter $k = 2$. The first part of the message is labeled as A and the second part as B; the messages are sent to the receiving nodes independently.

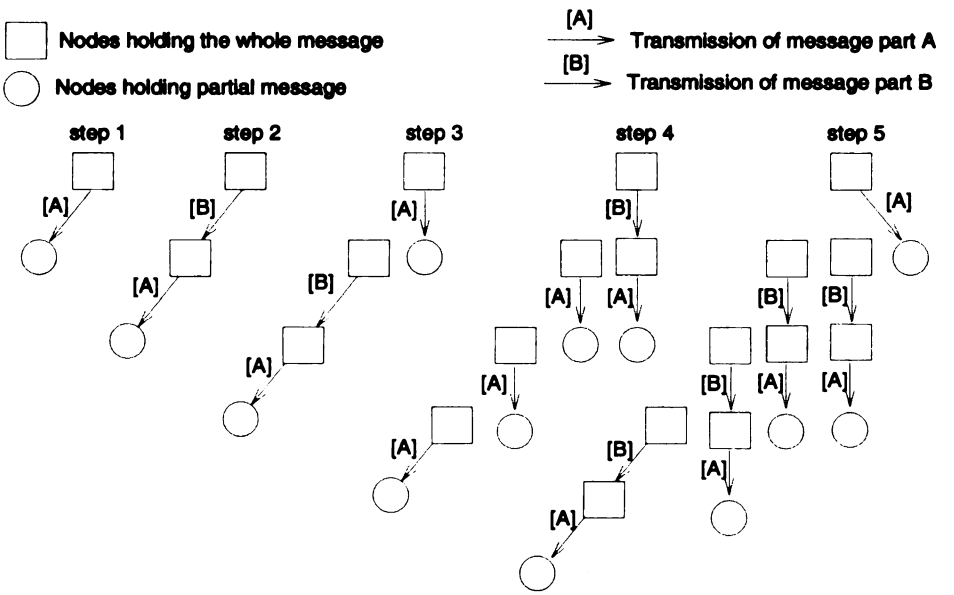


Figure 5.7: Fibonacci Tree algorithm for $k = 2$.

The relationship between the partition parameter k and the total number of nodes N that have received the entire message after t steps of message transmission can be described by the following recursion [50]:

$$\begin{cases} N(t,k) = N(t-k,k) + N(t-1,k) & \text{for } t \geq k \\ N(t,k) = 1 & \text{otherwise} \end{cases}$$

When implemented on a mesh network of size $P = 2^n \times 2^n$, the total number of hops traversed by the segmented message is k times that of the RD algorithm, if the original message is divided into k packets, each of size no more than $\lceil \frac{L}{k} \rceil$. We define the number of steps in the Fibonacci tree algorithm as $t(P,k)$ for group size P and message segmentation factor of k. A special case of the function $t(P,k)$ occurs when $k = 1$ and $P = 2^n \times 2^n$, specifically, $t(2^n * 2^n, 1) = t_1 = 2n$. We can approximate the latency of the Fibonacci tree algorithm by adding up the cost from the distance traveled by the message, $2k(2^n - 1)\beta$, to the cost computed by $t(P,k)$ steps of unicast message, $\alpha + \beta\frac{L}{k} + \gamma$. Therefore, the latency formula for the Fibonacci tree algorithm is approximated by

$$\mathcal{L}_{FT} = t(P,k)(\alpha + \gamma) + 2k\beta(2^n - 1) + t(P,k)\beta\frac{L}{k}.$$

The final algorithm that we consider is EDN broadcast. As described earlier, the EDN methodology designed to take advantage of a multiport architecture. We can compute its performance based on the analysis in Chapter 3. The approximate cost, without taking into account the distances traveled by the message, is $3n\alpha + (n + 1)\gamma + (n+1)\beta L$. The cost induced by the distance can be approximated by dividing the distance cost of the RD algorithm by half, as indicated by the simulation results in Chapter 3. Therefore, the overall cost of the EDN algorithm can be approximated by $3n\alpha + (n + 1)\gamma + (2^n - 1)\beta + (n + 1)\beta L$.

We summarize the performance of the four broadcast algorithms according to

the τ-model in Table 5.3. The overall latency formula takes the form $T_s(\alpha, \beta, \gamma, n) + T_n(\alpha, \beta, \gamma, n)L$, the T_s and T_n components are listed in the table.

Table 5.3: Analytical performance of the three broadcast algorithms

Algorithm	T_s	T_n
RD	$2n\alpha + 2\beta(2^n - 1) + 2n\gamma$	$2n\beta$
SC	$2(2^n - 1 + n)(\alpha + \gamma) + 6(2^n - 1)\beta$	$2(1 - \frac{1}{2^{2n}})\beta$
FT	$t(P,k)\alpha + 2k\beta(2^n - 1) + t(P,k)\gamma$	$\frac{t(P,k)\beta}{k}$
EDN	$3n\alpha + (n+1)\gamma + (2^n - 1)\beta$	$(n+1)\beta$

5.4 Comparisons Using the τ-Model

In this section, we demonstrate the use of the τ-model to analyze the four broadcast algorithms based on wormhole switching technology. More specifically, we use parameterized wormhole-routed square mesh network as the underlying networking technology to compare different implementations of broadcast algorithm.

5.4.1 Overall Comparison of Algorithms

Let us begin with two examples. We fix the size of the network by letting $n = 5$, and we assume that $\alpha = \gamma$. In the first example, we vary the value of α from 0.5 to 1.5 in increments of 0.1, while the value of β is fixed at 0.01. This represents a range of τ values from 0.01 to 0.003. In the second example, we vary the values of β from 0.001 to 0.2 in increments of 0.01, while we fix the value of α at 0.5. This results in a range of τ values from 0.001 to 0.2. Next, we calculate the T_s and T_n values and

the corresponding τ values for each algorithm. From those different (τ, T_s) pairs, the τ-plot for these four algorithms is constructed.

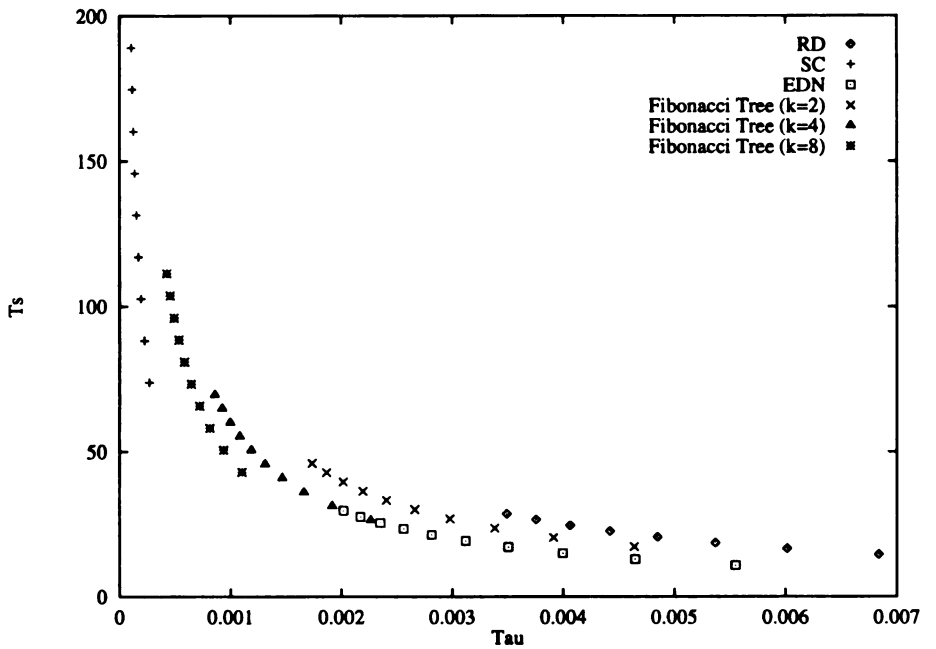


Figure 5.8: Fixed β with varying α.

As indicated in the Figure 5.8, with increasing α (τ decreases), the SC algorithm has more rapidly rising T_s component than the other three algorithms. From the closeness to the origin of the trace made by the SC algorithm, we know that it utilizes the largest aggregate network bandwidth, but suffers severely from the increased T_s component as the underlying network parameter α increases. The fact that the SC algorithm moves toward upper left corner as α increases indicates that long message sizes are more suitable for this algorithm. The FT algorithm can utilize more aggregate network bandwidth as k increases, however, with large k values, it is more sensitive to increased T_s as network parameter α increases. When compared with the SC algorithm, the FT algorithm will be more suitable for smaller message sizes. The EDN algorithm utilizes less aggregate network bandwidth as

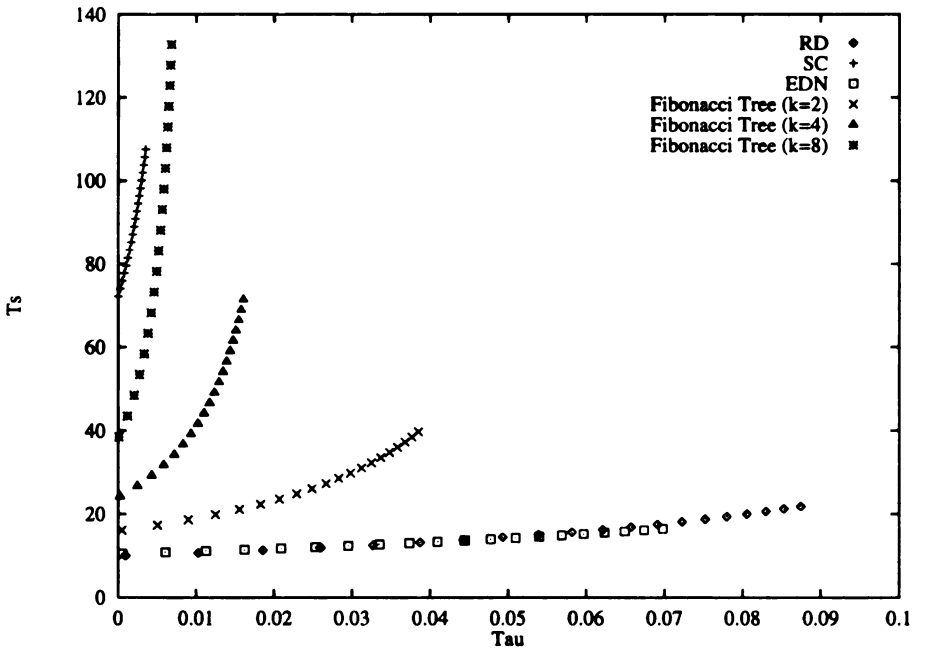


Figure 5.9: Fixed α with varying β.

α increases, the bandwidth is approximately the same as the FT algorithm with $k = 4$, and is less subjective to increased T_s. The RD algorithm is the lowest in network bandwidth utilization, and it has the least increase in T_s as α increases. However, it also has the largest τ values among the algorithms compared.

To summarize, the SC algorithm is more suitable for long message sizes and networks with larger τ values. The EDN algorithm will perform better for smaller message sizes and the FT algorithm with smaller k values is suitable for smaller message sizes, but compared with the EDN algorithm, the FT algorithm is faster for large message sizes.

Shown in Figure 5.9 is the τ-plot with increasing β (τ increases). The traces of these algorithms indicate that the upper bound of τ values in decreasing order is RD, EDN, $\text{FT}_{k=2}$, $\text{FT}_{k=4}$, $\text{FT}_{k=8}$, and SC when network parameter β increases. The SC algorithm will perform faster than the $\text{FT}_{k=8}$ algorithm as network param-

eter β increases. From the traces made by $FT_{k=2}$, $FT_{k=4}$, and $FT_{k=8}$, we can tell that as k increases, the FT algorithm will have characteristics similar to the SC algorithm, but performance will be worse than the SC algorithm as β increases. When comparing the RD algorithm with the EDN algorithm, we know that for this $\alpha = \gamma = 0.01$, the EDN algorithm will be faster than the RD algorithm. The characteristic of the EDN algorithm is similar to the RD algorithm, as compared with the FT, SC algorithms, the former are both suitable for short message sizes.

In summary, as the network τ value increases, the SC algorithm is better for long message sizes and the EDN algorithm is better for short message sizes. The FT algorithm has similar characteristic as the SC algorithm, but for large k and large τ values the SC algorithm will perform faster than the FT algorithm.

As demonstrated above, the comparison between broadcast algorithms through the use of the τ-plot can provide some qualitative information regarding the algorithms. In the following, detailed quantitative comparisons of between the SC, the FT and the EDN algorithms are made. This analysis is intended to give a thorough understanding about the behavior of these algorithms under any reasonable architectural assumptions.

5.4.2 Comparison of SC and FT

From the latency formulae of the SC and the FT algorithms, as indicated in Table 5.4, it is very difficult to extract an overall picture of how one algorithm performs as compared to the other under different parameter settings. With the aid of the

τ-model, we can establish these conditions and compare the algorithms easily. The parameters involved in the latency are the mesh size $P = 2^n \times 2^n$, the message size L, the segmentation count k for the FT algorithm, and the three unicast latency parameters, α, β, and γ.

Table 5.4: The latency formulae for the SC and the FT algorithms.

	T_s	T_n	τ
SC	$2(2^n - 1 + n)(\alpha + \gamma) + 6(2^n - 1)\beta$	$2(1 - \frac{1}{2^{2n}})\beta$	$\frac{(1-\frac{1}{2^{2n}})\beta}{(2^n-1+n)(\alpha+\gamma)+3(2^n-1)\beta}$
FT	$t(P,k)(\alpha + \gamma) + 2(2^n - 1)k\beta$	$\frac{t(P,k)\beta}{k}$	$\frac{t(P,k)\beta}{t(P,k)(\alpha+\gamma)k+2(2^n-1)k^2\beta}$

From our earlier description of the τ-model, we know that when $T_s[SC] < T_s[FT]$ and $\tau_{SC} < \tau_{FT}$, the SC algorithm performs better than the FT algorithm for all message sizes. The first condition leads to the inequality

$$\tau_0 > \frac{2(2^n - 1 + n) - t(P,k)}{2(2^n - 1)(k - 3)}, \tag{5.8}$$

provided that $k-3 > 0$, where the parameter τ_0 is defined to be $\beta/(\alpha+\gamma)$. Similarly, we can derive from the second condition that

$$\tau_0 > \frac{t(P,k)\{k(1 - \frac{1}{2^{2n}}) - 2^n + 1 - n\}}{(2^n - 1)\{3t(P,k) - 2k^2(1 - \frac{1}{2^{2n}})\}} \quad \text{when} \quad 3t(P,k) - 2k^2(1 - \tfrac{1}{2^{2n}}) > 0 \tag{5.9}$$

and

$$\tau_0 < \frac{t(P,k)\{k(1 - \frac{1}{2^{2n}}) - 2^n + 1 - n\}}{(2^n - 1)\{3t(P,k) - 2k^2(1 - \frac{1}{2^{2n}})\}} \quad \text{when} \quad 3t(P,k) - 2k^2(1 - \tfrac{1}{2^{2n}}) < 0. \tag{5.10}$$

If, on the other hand, the equation $3t(P,k) - 2k^2(1 - \frac{1}{2^{2n}}) = 0$ is true, by replacing $2k^2(1 - \frac{1}{2^{2n}})$ with $3t(P,k)$ in the inequality $\tau_{SC} < \tau_{FT}$, we can conclude that for any $k > 0$, the inequality $\tau_{SC} < \tau_{FT}$ holds. Therefore, the satisfaction of conditions $3t(P,k) - 2k^2(1 - \frac{1}{2^{2n}}) = 0$ and Equation 5.8, or Equation 5.8 and Equation 5.9, 5.10, will guarantee that the SC algorithm performs better than the FT algorithm for any given message length L.

In order to facilitate later derivations, we define three functions, $\omega(n,k), \epsilon(n,k)$, and $\mu(n,k)$, which depend only on parameter n and k, as follows;

$$\begin{aligned}
\omega(n,k) &= 3t(P,k) - 2k^2(1 - \frac{1}{2^{2n}}), \\
\epsilon(n,k) &= \frac{2(2^n - 1 + n) - t(P,k)}{2(2^n - 1)(k-3)}, \quad \text{and} \\
\mu(n,k) &= \frac{t(P,k)\{k(1 - \frac{1}{2^{2n}}) - 2^n + 1 - n\}}{(2^n - 1)\omega(k)}.
\end{aligned}$$

Because the parameter n is a constant when we consider a fixed-sized network of size $P = 2^n \times 2^n$, the only variable in the evaluation of these three functions is k. Once the mesh size is determined (n is fixed), we can compute a series of ω, ϵ, and μ values for varying k.

By comparing τ_0 with $\epsilon(n,k)$, $\mu(n,k)$, and $\omega(n,k)$, we can determine when the SC algorithm is better than the FT algorithm. Following a procedure similar to that outlined above, we know that when $T_s[SC] > T_s[FT]$ and $\tau_{SC} > \tau_{FT}$, the FT algorithm is faster than the SC algorithm. The first inequality leads to $\tau_0 < \epsilon(n,k)$ for $k > 3$ and the second inequality leads to $\tau_0 \gtrless \mu(n,k)$ when $\omega(n,k) \gtrless 0$.

In addition to the above two cases, according to the τ-model, there are cases

when one algorithm performs better for message size greater than a specific length. Specifically, when $T_s[SC] < T_s[FT]$ and $\tau_{SC} > \tau_{FT}$, the SC algorithm will perform better than the FT algorithm for any message size $L > L^*$, where $L^* = (T_s[SC] - T_s[FT])/(T_n[FT] - T_n[SC])$. This leads to the results that when $\tau_0 < \epsilon(n,k)$, and $\tau_0 \gtrless \mu(n,k)$ for $\omega(n,k) \gtrless 0$, the SC algorithm is better than the FT algorithm for message sizes greater than L^*, otherwise, the FT algorithm is better.

By combining these results, we can compare the SC algorithm with the FT algorithm for any architectural and algorithmic parameters as follows:

- From the equation $\omega(n,k) = 0$, we can solve for k. Suppose the solution is $k^* > 0$. We know that when $k = k^*$ the SC algorithm is better than the FT algorithm for any given network parameters α, β, and γ.

- For a given mesh network with parameters $\tau_0 > \epsilon(n,k)$, the SC algorithm is better than the FT algorithm for any $k > 3$.

- Given a mesh network with parameter τ_0, if $\tau_0 \leq \epsilon(n,k)$ and one of the conditions $\tau_0 \gtrless \mu(n,k)$ for $\omega(n,k) \gtrless 0$ holds, the SC algorithm is better than the FT algorithm for any message size L such that $L > L^* = (T_s[SC] - T_s[FT])/(T_n[FT] - T_n[SC])$. For any $L < L^*$, the FT algorithm is better than the SC algorithm.

- For a specified mesh network parameter τ_0, if $\tau_0 \leq \epsilon(n,k)$ and one of the conditions $\tau_0 \gtrless \mu(n,k)$ for $\omega(n,k) \lesseqgtr 0$ holds, the FT algorithm is better than the SC algorithm for any $k > 3$.

In order to illustrate the use of the above results, we select a mesh network with $n = 5$ (thus $P = 2^5 \times 2^5 = 1024$ nodes), and assume that $\alpha = \gamma = 1.5$, and $\beta = 0.009$, which are very similar to the T3D parameters [44]. This results in a τ_0 value of 0.003. The values of $\epsilon(5, k)$ and $\mu(5, k)$ for different values of k, are shown in Figure 5.10.

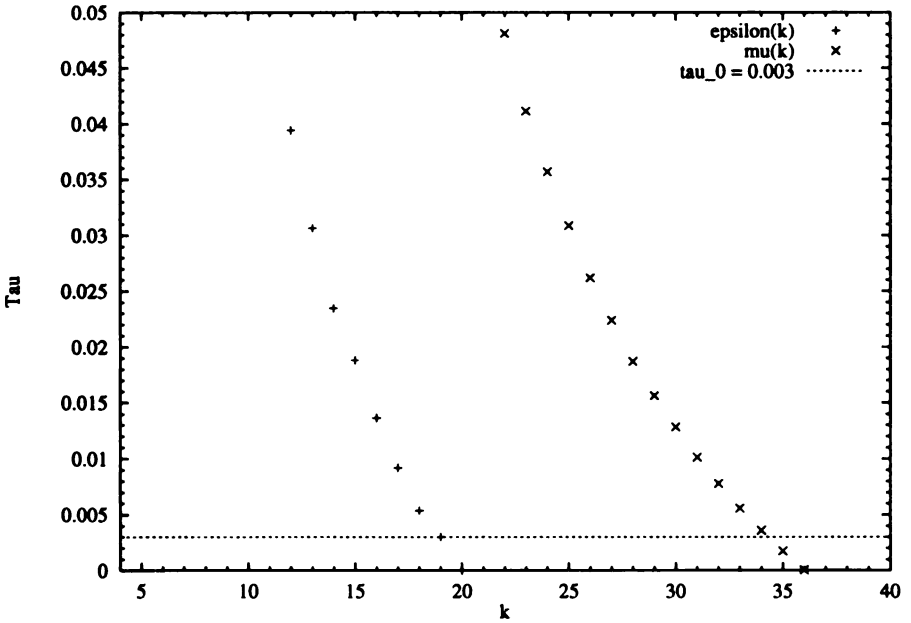


Figure 5.10: Plotting of $\epsilon(5, k)$ and $\mu(5, k)$ with respect to k.

Because these three functions, $\epsilon(n, k)$, $\mu(n, k)$, and $\omega(n, k)$, are independent of τ, which indicates for a given network of size $2^n \times 2^n$, for any $\alpha > 0, \beta > 0$ and $\gamma > 0$, the resulting plots are identical. For a given τ_0, we can draw a horizontal line across the plot to indicate that it is independent of k, as shown in Figure 5.10. From the previous results, when $\tau_0 > \epsilon(n, k)$, the SC algorithm is better than the FT algorithm. As illustrated in Figure 5.10, by following the line indicating constant $\tau_0 = 0.003$, this condition holds only for $k \geq 20$. As to $3 < k < 20$, the condition $\tau_0 < \epsilon(n, k)$ holds, as well as $\tau_0 < \mu(n, k)$ for $\omega(n, k) < 0$ and $\tau_0 > \mu(n, k)$ for

$\omega(n,k) > 0$. This indicates that for $3 < k < 20$, the SC algorithm is better than the FT algorithm for any message size $L > L^* = (T_s[SC] - T_s[FT])/(T_n[FT] - T_n[SC])$. The computed L^* values vs. different k values are illustrated in Figure 5.11.

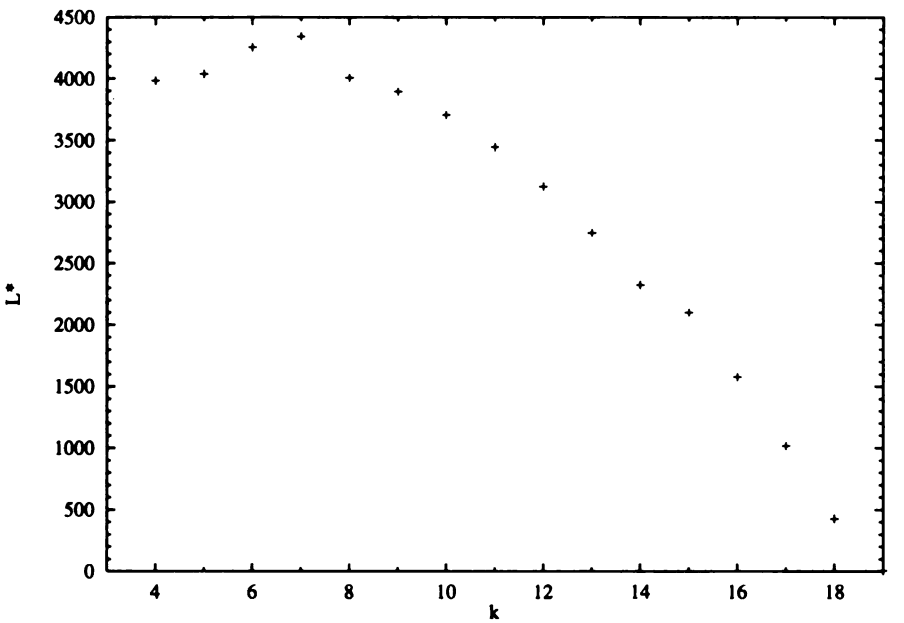


Figure 5.11: Different L^* values with respect to k.

Figure 5.12(a) compares the two algorithms by latency vs. message length as computed from the latency formula with specified α, γ and β values. For the FT algorithm, we select a set of k values, $k = 16, 18, 19, 20$, in the neighborhood of crossover point from Figure 5.10. The results agree with those predicted by the analysis according to the τ-model. When we keep the τ value fixed at 0.003 but change parameters (α, β, γ) from $(1.5, 0.009, 1.5)$ to $(0.5, 0.003, 0.5)$, the result is illustrated in Figure 5.12(b). The crossover message length for $k = 18$ can be calculated from the equation $L^* = (T_s[SC] - T_s[FT])/(T_n[FT] - T_n[SC])$. In the first case, when $\alpha = \gamma = 1.5$, $\beta = 0.009$, and $t(P, 18) = 67$, the crossover message

size is

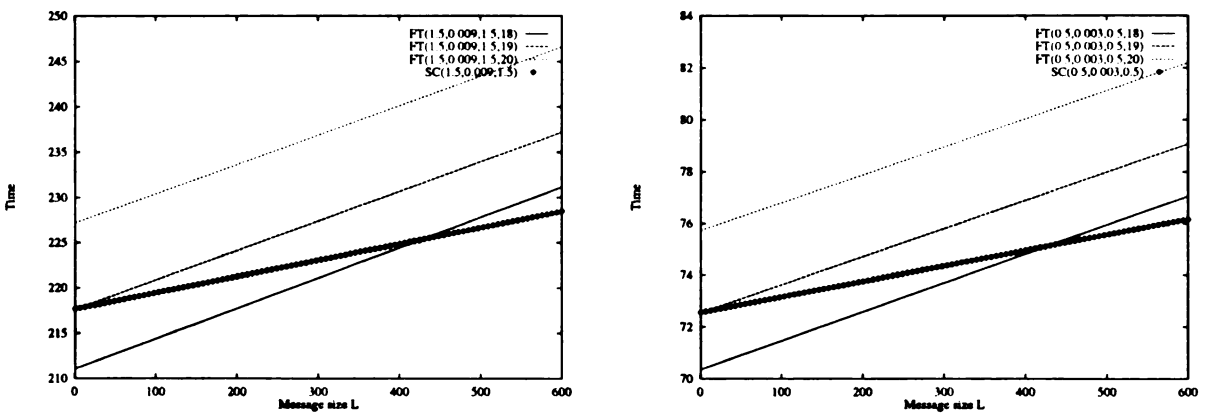
$$\frac{2(2^5-1+5)(1.5+1.5)+6(2^5-1)*0.009-67*(1.5+1.5)-2(2^5-1)*18*0.009}{(67*0.009/18)-2(1-1/2^{2*5})*0.009}$$

$$= \frac{6.603}{0.0155} = 425.5$$

In the second case, when $\alpha = \gamma = 0.5$, $\beta = 0.003$, and $t(P, 18)$ remains as 67, the crossover message size is

$$\frac{2(2^5-1+5)(0.5+0.5)+6(2^5-1)*0.003-67*(0.5+0.5)-2(2^5-1)*18*0.003}{(67*0.003/18)-2(1-1/2^{2*5})*0.003}$$

$$= \frac{2.21}{0.0051725} = 425.25$$



(a) $\alpha = \gamma = 1.5$, $\beta = 0.009$ (b) $\alpha = \gamma = 0.5$, $\beta = 0.003$

Figure 5.12: The SC algorithm compared with the FT algorithm with different k values.

When we compare Figure 5.12(a) with Figure 5.12(b), the difference is the scale

of the Y-axis, the crossover message length remains as predicted.

5.4.3 Comparison of EDN and SC

Next, we compare the EDN algorithm with the SC algorithm. The procedure is similar to the process illustrated in the previous discussion. Table 5.5 gives the latency formulae for both algorithms.

Table 5.5: Latency formulae for the SC algorithm and the EDN algorithm.

	T_s	T_n	τ
SC	$2(2^n-1+n)(\alpha+\gamma)+6(2^n-1)\beta$	$2(1-\frac{1}{2^{2n}})\beta$	$\frac{(1-\frac{1}{2^{2n}})\beta}{(2^n-1+n)(\alpha+\gamma)+3(2^n-1)\beta}$
EDN	$3n\alpha+(n+1)\gamma+(2^n-1)\beta$	$(n+1)\beta$	$\frac{(n+1)\beta}{3n\alpha+(n+1)\gamma+(2^n-1)\beta}$

We can compute the difference between $T_s[SC]$ and $T_s[EDN]$ as $T_s[SC] - T_s[EDN] = (2*2^n-2-n)\alpha+(2*2^n-3+n)\gamma > 0$ for $n > 1$. Thus, the inequality $T_s[SC] > T_s[EDN]$ holds for all $n > 1$. In addition, the difference between $T_n[EDN]$ and $T_n[SC]$ is $T_n[EDN]-T_n[SC] = (n+\frac{2}{2^{2n}}-1)\beta > 0$ for all $n > 1$ and $\beta > 0$. From these two conditions, we can conclude that the EDN algorithm is better than the SC algorithm for $L < L^*$. The length $L^* = (T_s[SC] - T_s[EDN])/(T_n[EDN] - T_n[SC])$. This indicates that independent of the underlying wormhole network parameters, the EDN algorithm is more suitable for short messages while the SC algorithm is better for long messages.

In Figure 5.13, the crossover message length is plotted against the network size. As the network size increases, the crossover size increases, and as the value of τ increases, the crossover message size decreases. This trend indicates that the SC

algorithm is more suitable for network with large τ values and large message sizes. However, the decrease of crossover message size becomes smaller as the values of τ increases. Take $n = 5$ for example, the crossover message size is 14 Kbytes when τ is 0.001 and it becomes 7000 bytes when τ is 0.002, the size decreases by a factor of 0.5. When τ is 0.003, the crossover message size is 4200 bytes, as compared with $\tau = 0.002$, the decrease factor is less than 0.5.

In summary, we can conclude that the EDN algorithm is faster for short message sizes, and the crossover message size increases as τ decreases.

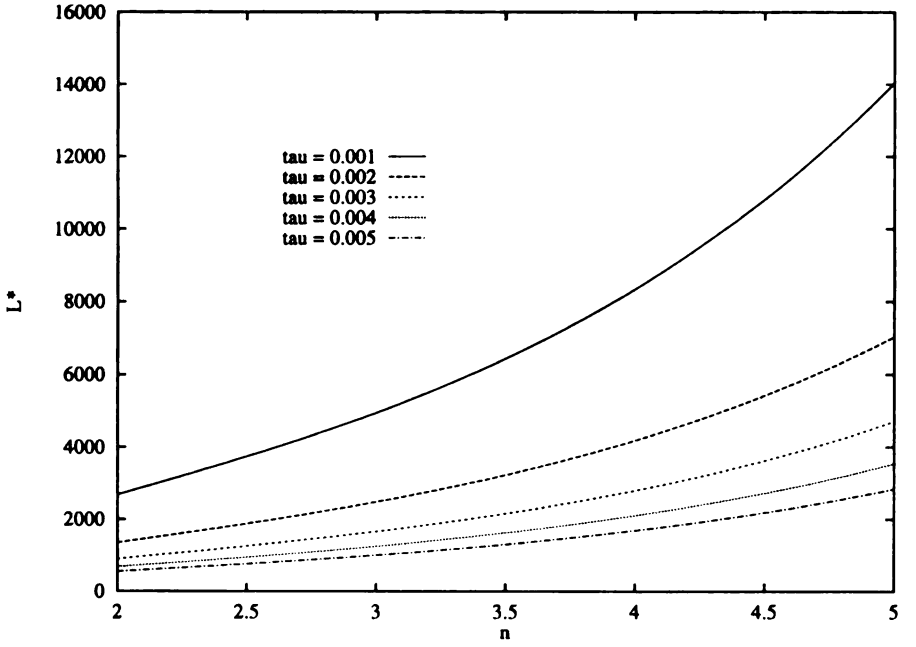


Figure 5.13: Different L^* values for various τ.

5.4.4 Comparison of EDN and FT

Following the same procedure used in comparing the SC algorithm with the FT algorithms, we can derive the conditions that will guarantee that the EDN algorithm performs better than the FT algorithm for any message length. That is, when both

inequalities $T_s[EDN] < T_s[FT]$ and $\tau_{EDN} < \tau_{FT}$ are true. From the first inequality, we can derive

$$\tau_0 > \frac{3n - t(P,k)}{2(k-1)(2^n-1)}.$$

Because $t(P,k) > 3n$ for $n = 5$ and $k > 3$, therefore, we know that the first inequality is valid for any $\tau_0 > 0$ due to the fact that $(3n - t(P,k))/(2(k-1)(2^n - 1)) < 0$. To facilitate the derivation of the second inequality, we define $\mathsf{R}(n,k)$ as $2k^2(n+1) - t(P,k)$. From the calculation of $t(P,k)$, we know that $\mathsf{R}(5,k) < 0$ for any $k > 2$. The following inequality is valid for any $k > 2$.

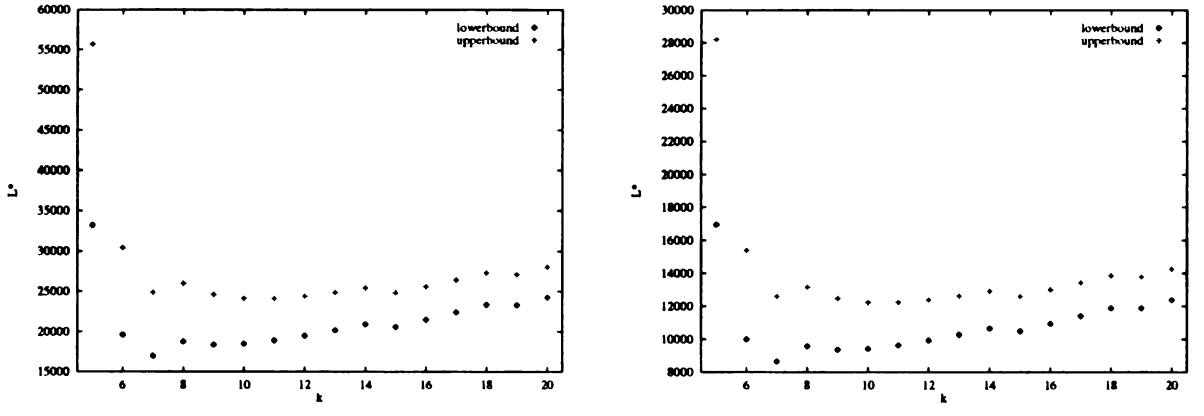
$$\begin{aligned} & t(P,k)\{3n\alpha + (n+1)\gamma\} - t(P,k)k(n+1)(\alpha+\gamma) \\ < \; & t(P,k)\{3n\alpha + 3n\gamma\} - t(P,k)k(n+1)(\alpha+\gamma) \\ & = t(P,k)(\alpha+\gamma)\{3n - k(n+1)\} < 0 \end{aligned}$$

Because $(2^n - 1)\beta * \mathsf{R}(n,k) > 0$ for all k, we know that $(2^n - 1)\beta\{2k^2(n+1) - t(P,k)\} > 0$. Furthermore, the following inequality is valid for all k.

$$(2^n - 1)\beta\{2k^2(n+1) - t(P,k)\} > t(P,k)\{3n\alpha + (n+1)\gamma\} - t(P,k)k(n+1)(\alpha+\gamma)$$

Rearranging the terms of the above inequality, we can conclude that $\tau_{EDN} > \tau_{FT}$.

Therefore, we know that both the inequalities $T_s[EDN] < T_s[FT]$ and $\tau_{EDN} > \tau_{FT}$ are true for $n = 5$ and $k > 2$. The EDN algorithm is faster than the FT algorithm for message sizes less than L^*. Similar to the previous comparison, we can calculate L^* for different $\tau_0 = \beta/(\alpha+\gamma)$. Figures 5.14(a), (b) illustrate the crossover message lengths for different τ_0 values. One observation from both Figure 5.14(a) and Figure 5.14(b) is that as k increases, the crossover message length does not increase very rapidly. In addition, as the τ_0 values increases, the crossover message length decreases. This indicates that architectures with lower τ_0 values will favor the EDN algorithm. This agrees with the observation from the τ-plot outlined previously.



(a) $\tau_0 = 0.001$ (b) $\tau_0 = 0.002$

Figure 5.14: The crossover message length with different τ_0 values.

To summarize, the process of comparing the performance of different communication algorithms over various architectural and algorithmic parameters is complicated. In the above analysis, we compare three broadcast algorithms, SC, EDN, and FT, through the use of the τ-model for a particular network topology and size. In

the beginning of this section, we use the τ-plot to facilitate the comparison visually. For a range of τ values, the τ-plot allows us to determine the qualitative characteristics of the algorithms under study easily. This is demonstrated by the comparison of the SC algorithm with the FT algorithm, and the different characteristics exhibited by the EDN algorithm and the SC algorithm. The SC algorithm is faster for long message lengths and favors machines with large τ values as compared with the EDN algorithm. Detailed analysis confirms the observation from the τ-plot. Nevertheless, to determine quantitatively the crossover message lengths, the computation from the τ-model is more appropriate. This approach can be applied to other networking topologies and sizes as well as different networking technologies, as demonstrated in the next section.

5.5 Application of the τ-Model to NOWs

As networking technology has evolved into different branches, and as new protocols and algorithms have been proposed over the past few years at a very fast pace, the need for a unified communication cost model is self evident. In this section, we present results of the application of the τ-model to networks of workstations, an increasingly popular parallel computing platform. In addition to broadcast algorithms, we use the τ-model to study all-to-all broadcast algorithms. Specifically, we use recent results from the literature to point out which aspects of the problem are most important, and speculate on how to improve implementations.

5.5.1 Broadcast Operations

In the following, we demonstrate the use of the τ-model to study the various results for broadcast on different network of workstation platforms. This demonstration focuses on the use of some empirical data from various research results and create a global view on those data. Thus allows us to understand better the performance comparison between those algorithms.

Table 5.6 gives the empirical data under study. The MPI-CCL library [32] is implemented atop unreliable hardware-supported multicast on Ethernet; the two entries are for group sizes of 4 and 8. These entries represent the performance of an MPI implementation over User-level Reliable Transport Protocol (URTP) on top of 10Mbps Ethernet [32]. The SS10/ASX100 entries represent software implementations of broadcast operations through FORE systems ATM application interface forming spanning binomial tree configuration, with group sizes of 4 and 8 [78]. The data for broadcast on networks such as Myrinet, ATM, and fast Ethernet are taken from [14]. There are two different implementations, the first is based on the Panda system [22] and the second is based on Fast Messages (FM) [92, 108]. For both cases, the group size is fixed at 8.

For the purpose of comparison, we also use data from implementations on parallel machines. The data for the IBM SP-2 are from Xu and Huang's study [158], in which two different communication libraries, MPI and MPL, were compared on a 512-node SP-2. The data for the Intel Paragon is from measurements of the InterComm collective communication library, also on a 512-node system [105].

Table 5.6: Performance of broadcast operation.

machine	software	$T_s(\mu s)$	T_n (μs/byte)	$\tau = T_n/T_s$	source
RS6000/10base-T	MPI-CCL/4	793.0	0.893	0.00113	[32]
RS6000/10base-T 10Mbps	MPI-CCL/8	2090.0	1.2	0.000573	[32]
SS10/ASX100	FORE API/4SBT	6080.0	1.87	0.000308	[78]
SS10/ASX100	FORE API/8SBT	12800.0	1.14	0.0000888	[78]
SPARC/100Base-T	Panda/8	311.3	0.105	0.000337	[14]
SPARC/100Base-T	FM/8	86.6	0.147	0.00170	[14]
SPARC/Myrinet	Panda/8	156.7	0.0838	0.000535	[14]
SPARC/Myrinet	FM/8	53.6	0.0901	0.00168	[14]
SPARC/ASX200-WG	Panda/8	370.3	0.106	0.000286	[14]
SPARC/ASX200-WG	FM/8	112.4	0.103	0.000916	[14]
SP-2	MPI/512	360.0	0.333	0.000925	[158]
SP-2	MPL/512	144.0	0.225	0.00156	[158]
Paragon	InterComm/512	3900.0	0.043	0.0000110	[105]

As illustrated in Figure 5.15, comparisons on various groups of data can be made using a τ-plot. Comparing algorithms over different networking technology are shown by the pairs (SPARC/100Base-T/Panda/8, SPARC/100Base-T/FM/8), (SPARC/Myrinet/Panda/8, SPARC/Myrinet/FM/8), and (SPARC/ASX200-WG/Panda/8, SPARC/ASX200-WG/FM/8). It is apparent that points SPARC/100Base-T/Panda/8, SPARC/Myrinet/Panda/8, and SPARC/ASX200-WG/Panda/8 have a relatively lower τ values than their counterpart SPARC/100Base-T/FM/8, SPARC/Myrinet/FM/8, and SPARC/ASX200-WG/FM/8. This tells us that algorithms based on the Panda system are relatively suitable for long messages than their counterparts based on the Fast Message. Based on the comparison of points SPARC/100Base-T/Panda/8, SPARC/Myrinet/Panda/8 and SPARC/100Base-T/FM/8, SPARC/Myrinet/FM/8, we can observe that SPARC/Myrinet/FM/8 is better than SPARC/100Base-T/FM/8 for all message sizes while SPARC/Myrinet/Panda/8 and SPARC/100Base-T/Panda/8 are not of similar relationship. This indicates that the algorithms based on the Fast Message have take advantage of the

networking technology (from Myrinet to fast Ethernet) while the algorithms based on the Panda system have not.

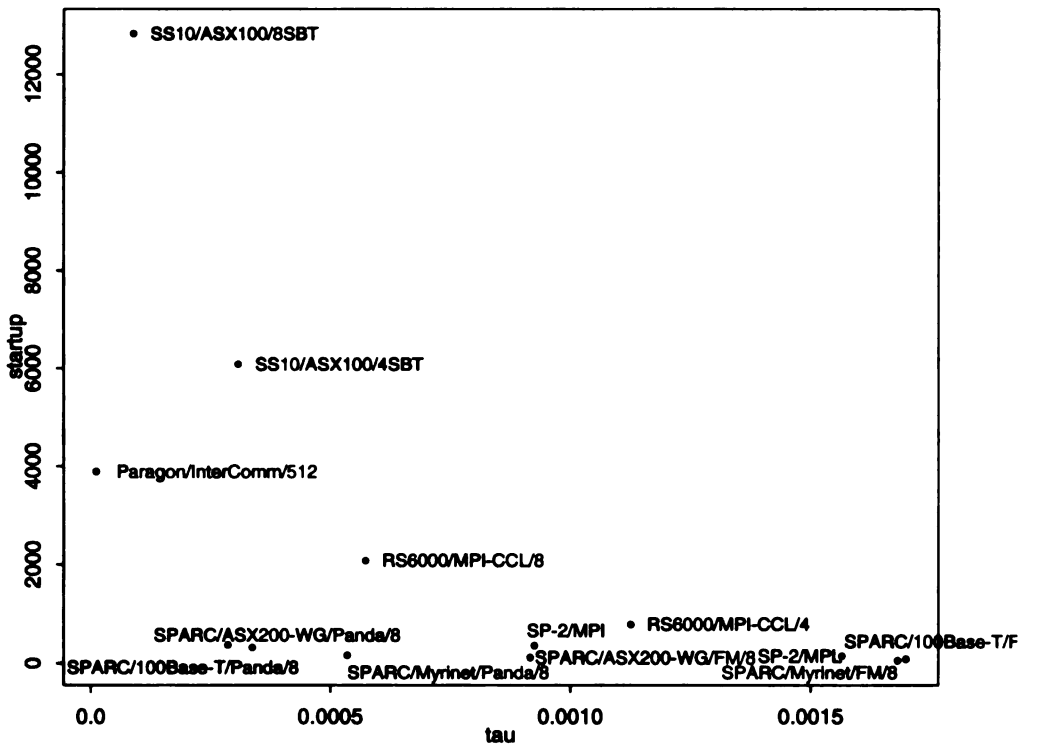


Figure 5.15: τ-plots for broadcast operations on different networking environments.

From the comparison of points SS10/ASX100/8SBT, SPARC/ASX200-WG/Panda/8, and SPARC/ASX200-WG/FM/8, they are all of the same group size, and are all based on ATM networking technology, what makes SS10/ASX100/8SBT deviates from SPARC/ASX200-WG/Panda/8 and SPARC/ASX200-WG/FM/8 are the high startup appeared in SS10/ASX100/8SBT. This is partly due to the fact that SS10/ASX100/8SBT uses point-to-point primitive to implement the multicast while both SPARC/ASX200-WG/Panda/8 and SPARC/ASX200-WG/FM/8

are built on top of the underlying un-reliable multicast mechanism.

One more observation from Figure 5.15 is that SPARC/ASX200-WG/FM/8 is better than SP-2/MPI for all message sizes. This indicates that for group size 8, the platforms based on commodity networks such as ATM have better performance than parallel machines such as SP-2

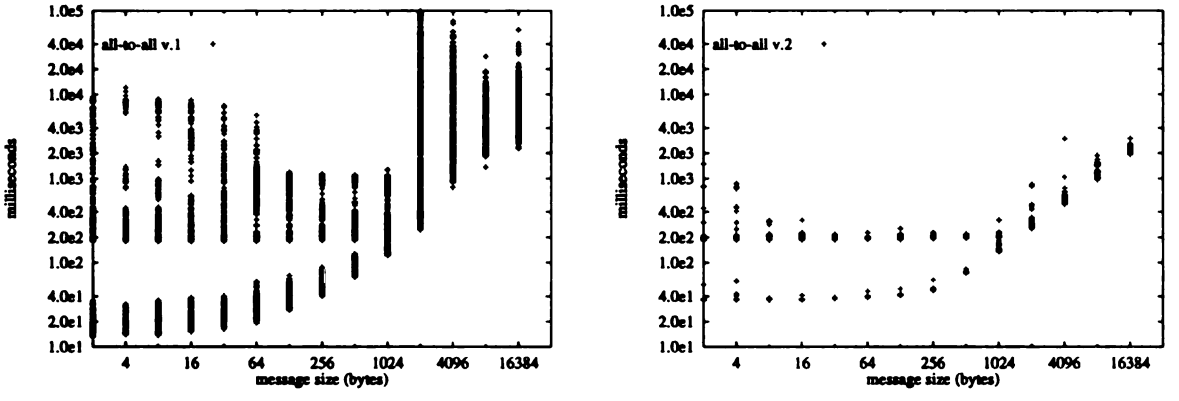
5.5.2 All-to-All Broadcast

For the all-to-all broadcast operation, we considered two different implementations, referred to as A and B, in a traditional Ethernet environment. The approach adopted in algorithm A is quite similar to that used in MPICH [65]: every participating node posts $(P-1)$ non-blocking indications to receive messages, and subsequently sends $(P-1)$ messages to their individual destinations. Due to the medium access control protocol of Ethernet, if every node starts the operation at the same time, the possibility of message collisions during initial and subsequent transmissions is very high.

The approach adopted in version B is to arrange the P participating nodes in a linear list, with each node successively transmitting its $(P-1)$ messages to the other nodes. The order in which nodes transmit is determined by their positions in the list. When one node has finished sending its messages, the next node must be triggered to begin sending. This synchronization is accomplished by arranging the message transmission pattern so that the last message sent by a given node is destined for the node that will start next. As an example, first node 0 sends $(P-1)$

messages to nodes $(P-1)$, $(P-2)$, ..., 1. Next, node 1 sends $(P-1)$ messages to nodes $0, (P-1), (P-2), \ldots, 2$. This process continues until node $(P-1)$ completes its message transmission.

A simple timing analysis of version B can be derived by assuming that the time needed to send a message of length L is $\alpha + \beta L$, in which case the overall cost is thus $P(P-1)(\alpha+\beta L)$. Therefore, one might assume that this approach would have much worse performance than version A. However, as indicated by the following experiments, the performance of this approach is in some sense quite competitive.



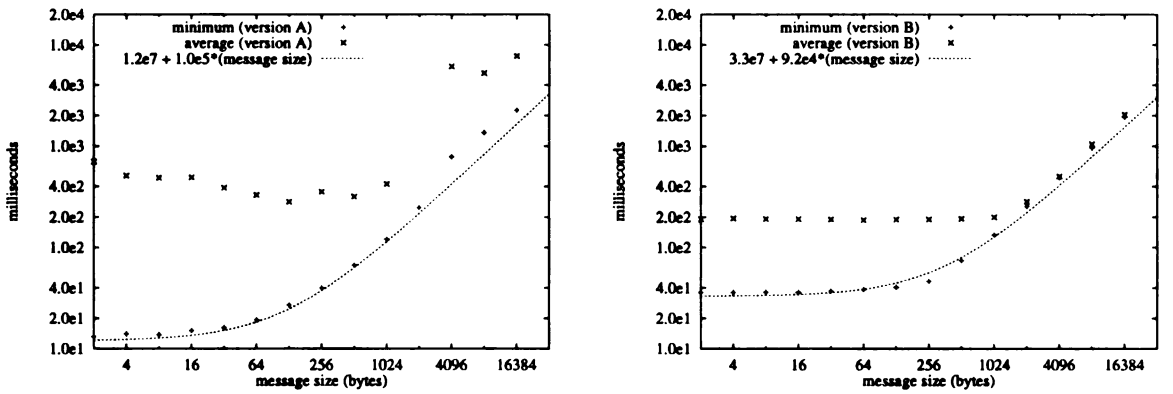
(a) version A.

(b) version B.

Figure 5.16: Results for all-to-all operation.

Figures 5.16(a) and (b) plot the results of executing all-to-all broadcast operations (1000 iterations for each message size) on 12 SUN SPARCstation-10 workstations interconnected by a shared Ethernet segment. The plots indicate that in this environment, completion times vary for both versions, but the variation is much greater for version A, likely due to network contention. Computing the average across 1000 runs shows that for message size less than 2048 bytes, version B is

faster than version A, despite the fact that the minimum value of version A is less than that of version B, as illustrated in Figure 5.17(a) and (b). In order to apply the τ-model to the performance plots of these two algorithms, we used the minimum values, which are the most optimistic estimation of algorithm performance.



(a) all-to-all version A.

(b) all-to-all version B.

Figure 5.17: Results of fitting the minimum value.

Following similar procedure, we measure the performance of these two all-to-all operations with group size 4 and 8. Table 5.7 summarizes the results of T_s and T_n values for these two algorithms with different group sizes.

Table 5.7: Performance of all-to-all operation.

machine	protocol/algorithm/size	$T_s(\mu s)$	$T_n(\mu s/\text{byte})$	$\tau = T_n/T_s$
SS10/Ethernet	TCP/A/4	3220.0	9.11	0.00283
SS10/Ethernet	TCP/B/4	5380.0	9.66	0.00180
SS10/Ethernet	TCP/A/8	6200.0	47.7	0.00769
SS10/Ethernet	TCP/B/8	14700.0	43.8	0.00298
SS10/Ethernet	TCP/A/12	12900.0	107.0	0.00830
SS10/Ethernet	TCP/B/12	32900.0	92.4	0.00281

From the data in Table 5.7, the τ-plot of all-to-all operations is constructed. As shown in Figure 5.18, one very distinct difference between these two versions of all-

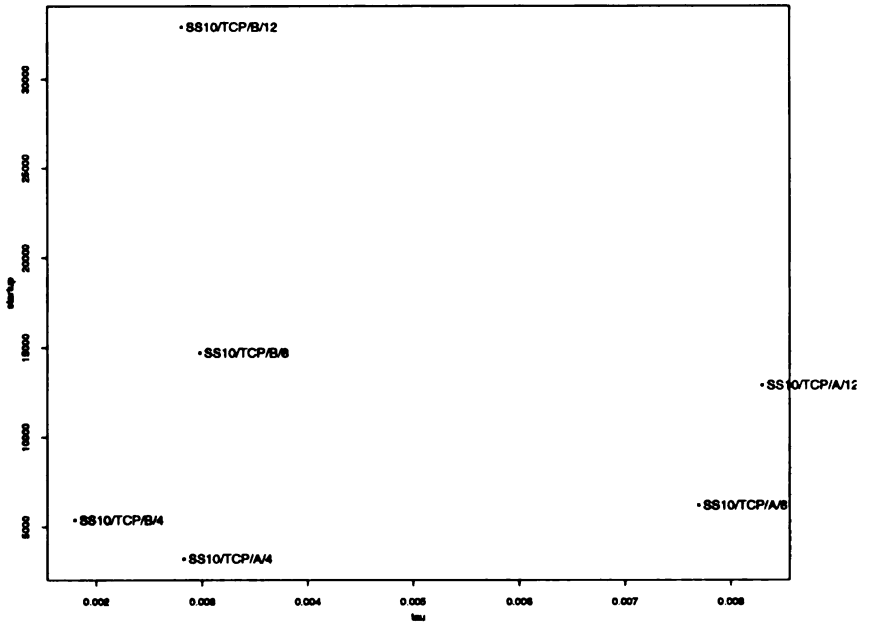


Figure 5.18: τ-plot of all-to-all operations.

to-all operation is the direction when we move from small group size to large group size. Version A tends to move to the right-hand direction while version B prefer to move up. Thus, the qualitative comparison of version A vs. version B is that for the same group size, version A prefers long messages, and this trend increases with increasing group size. Such a comparison is not obvious when we look at the latency-length or other similar performance plots. The second observation from the τ plot is the effect of increased group size. As the group size changes from 4 to 8, the horizontal shift made by version A is greater than that of B, which indicates that as group size increases, the rate of growth in the latency plot of A increases more rapidly than version B, however, the vertical shift of version B is greater than A, thus overshadowing the gain.

5.5.3 Summary

To summarize, we have demonstrated the use of the τ model to compare communication algorithms under different system parametric assumptions in NOW environments. This study allows us to gain better understanding of the behavior of any communication operation under different system parameters. It also illustrates that for a specific implementation, the effect of a particular system parameter on its performance can be quantified. From this study, we know that as the number of parameters increases, the task of detailed comparison between different algorithms is not trivial. Different networking environments as well as different algorithm implementations further complicated the issue. However, through the use of the τ-plot, we can capture the most prominent feature of the algorithm as compared with other algorithms. As seen from the study of communication operations in the network of workstations environment, the τ model approach provides a common platform to facilitate the understanding of the issues of communication performance.

Chapter 6

Conclusions and Future Research

Generally speaking, the purpose of a model is to capture the salient features of the object under study with clarity and defining details. Though many well known parallel *computation* models have been developed in recent years, these models are not necessarily well-suited for modeling *communication*. Since communication is a critical part of parallel applications, and since communication technologies and software interfaces are changing frequently, the need for an accurate communication model is immediate.

The research described in this dissertation has produced the following contributions: a general model for the design of efficient collective communication algorithms that could exploit the characteristics of certain network hardware architecture; the proper integration of a communication model and a cost model to support the design of communication primitives for various network environments. Moreover, this work improves the theoretical background used in characterizing the complexity of different communication operations and in comparing their implementations.

Our work in designing efficient communication primitives for wormhole routed systems generated insight into the proper cost model for collective communication. Specifically, the use of the EDN model in developing communication primitives illustrates a conceptual model, associated with the usual cost model, that aids the design of efficient communication operations while taking into account specific architectural features. The extension of the EDN model to workstation cluster environments can be considered quite natural, since many communication media used in those platforms provide multi-destination message passing. Examples include Ethernet, Fast Ethernet, and ATM switches, all of which support hardware multicast. We have conducted a preliminary study in this area [141]; additional studies are deferred to future research.

As an increasing number and variety of network architectures have been introduced into the commodity market, and as the trend of using such components for parallel computing continues, the task of understanding and comparing communication algorithms is increasingly complicated. The second model developed in this dissertation, the τ-model, ties together the various aspects of performance comparison and allows the comparison of algorithms and networking technologies to be done both qualitatively and quantitatively.

The future generation of parallel computing platforms is growing into an area where diverse networking components and workstations play the major roles. Understanding and comparing the performance of different implementations of the same operation, as well as different networking technologies, can assist in the refinement and selection of a better platform/algorithm combinations. Further re-

search can be conducted in the area of creating tools based on the τ-model. Such tools would facilitate a uniform representation of performance data for different communication operations and technologies as they come available.

Appendices

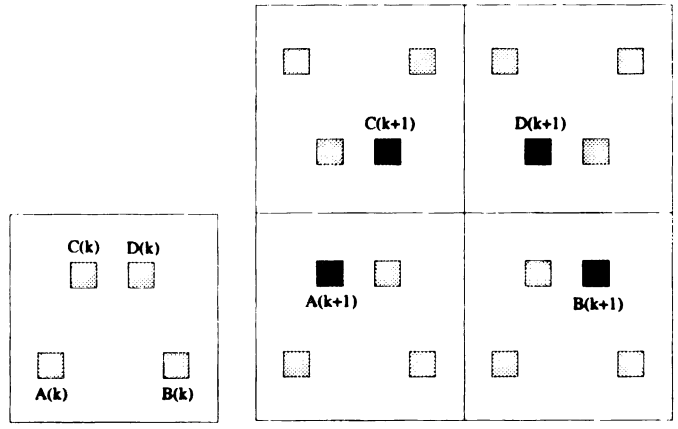
Appendix A

Formulae for EDNs in $2^k \times 2^k$ Meshes

We use the D pattern in Figure 3.8(b) to show how to compute the coordinates of the EDNs in a 2D mesh of size $2^k \times 2^k, k \geq 3$.

As illustrated in Figure -1(a), we denote the four highest level EDNs of a $2^k \times 2^k$ mesh as $A(k) = (A_x(k), A_y(k))$, $B(k) = (B_x(k), B_y(k))$, $C(k) = (C_x(k), C_y(k))$ and $D(k) = (D_x(k), D_y(k))$.

The construction of the next level EDNs is as shown in Figure -1(b). Let the corresponding four highest level EDNs in this $2^{k+1} \times 2^{k+1}$ mesh be $A(k+1), B(k+$



(a) the four highest level EDNs in a $2^k \times 2^k$ mesh

(b) the four highest level EDNs in a $2^{k+1} \times 2^{k+1}$ mesh

Figure -1: Representation of four highest level EDNs

$1), C(k+1)$ and $D(k+1)$. The following relation holds for any $k \geq 3$.

$$\begin{cases} A_x(k) = C_x(k-1) & , A_y(k) = C_y(k-1) \\ B_x(k) = 2^k - 1 - C_x(k-1) & , B_y(k) = C_y(k-1) \\ C_x(k) = 2^{k-1} - 1 - C_x(k-1) & , C_y(k) = 2^k - 1 - C_y(k-1) \\ D_x(k) = 2^k - 1 - D_x(k-1) & , D_y(k) = 2^k - 1 - D_y(k-1) = C_y(k) \end{cases}$$

The case of the 8×8 mesh serves as a boundary condition for the above recurrence. From Figure 3.8(b), the coordinates for these four level-2 EDNs in 8×8 mesh are:

$$\begin{cases} A_x(3) = 1 & , A_y(3) = 0 \\ B_x(3) = 6 & , B_y(3) = 0 \\ C_x(3) = 2 & , C_y(3) = 7 \\ D_x(3) = 5 & , D_y(3) = 7 \end{cases}$$

Now we solve the above recurrence for $C_x(k)$. Let $S(k) = 2^k - 1$. By expanding $C_x(k-1)$ with $C_x(k-2)$, $C_x(k-2)$ with $C_x(k-3)$ and so on until $C_x(3)$, we arrive at the following formula for $C_x(k)$:

$$C_x(k) = \begin{cases} \sum_{j=1}^{k-3}(-1)^{j-1}S(k-j) - C_x(3) & \text{when } k \text{ is even} \\ \sum_{j=1}^{k-3}(-1)^{j-1}S(k-j) + C_x(3) & \text{when } k \text{ is odd.} \end{cases}$$

Solving the summation and setting $C_x(3) = 2$, the following closed-form solution for $C_x(k)$ can be found:

$$C_x(k) = \begin{cases} \frac{2^k+5}{3} - C_x(3) = \frac{2^k-1}{3} & \text{when } k \text{ is even} \\ \frac{2^k-8}{3} + C_x(3) = \frac{2^k-2}{3} & \text{when } k \text{ is odd.} \end{cases}$$

The same procedure can also be used to derive the following equation for $C_y(k)$:

$$C_y(k) = \begin{cases} \frac{2^{k+1}+13}{3} - C_y(3) = \frac{2^{k+1}-8}{3} & \text{when } k \text{ is even} \\ \frac{2^{k+1}-16}{3} + C_y(3) = \frac{2^{k+1}+5}{3} & \text{when } k \text{ is odd.} \end{cases}$$

In a similar fashion, the formula for $D_x(k)$ can be written as:

$$D_x(k) = \begin{cases} \frac{2^{k+1}+13}{3} - D_x(3) = \frac{2^{k+1}-2}{3} & \text{when } k \text{ is even} \\ \frac{2^{k+1}-16}{3} + D_x(3) = \frac{2^{k+1}-1}{3} & \text{when } k \text{ is odd.} \end{cases}$$

Thus, we have the following formula to compute the coordinates of the four highest

level EDNs for a mesh of size $2^k \times 2^k$:

$$\left.\begin{array}{lll} A_x(k) = \frac{2^{k-1}-2}{3} & , A_y(k) = \frac{2^k+5}{3} \\ B_x(k) = \frac{3*2^k-2^{k-1}+1}{3} & , B_y(k) = A_y(k) \\ C_x(k) = \frac{2^k-1}{3} & , C_y(k) = \frac{2^{k+1}-8}{3} \\ D_x(k) = \frac{2^{k+1}-2}{3} & , D_y(k) = C_y(k) \end{array}\right\} \text{when } k \text{ is even}$$

$$\left.\begin{array}{lll} A_x(k) = \frac{2^{k-1}-1}{3} & , A_y(k) = \frac{2^k-8}{3} \\ B_x(k) = \frac{3*2^k-2^{k-1}-2}{3} & , B_y(k) = A_y(k) \\ C_x(k) = \frac{2^k-2}{3} & , C_y(k) = \frac{2^{k+1}+5}{3} \\ D_x(k) = \frac{2^{k+1}-1}{3} & , D_y(k) = C_y(k) \end{array}\right\} \text{when } k \text{ is odd.}$$

By using the above formulae to calculate the highest-level EDNs for submeshes and by performing coordinate transformations to larger meshes, we can generate all levels of EDNs within a mesh. Using this information, any node in the mesh can compute in advance the list of nodes that it should forward to upon receipt of a broadcast message. Some care must be taken in the startup phase, since a highest level EDN may be required to forward the message to one or more other EDNs at the highest level. The message transmission pattern in this startup phase can be either dependent on the location of source node or fixed.

Bibliography

Bibliography

[1] M. Adler, J. W. Byers, and R. M. Karp, "Scheduling parallel communication: The h-relation problem," in *Mathematical Foundations of Computer Science 1995, The* 20^{th} *International Symposium* (J. Wiedermann and P. Hájek, Eds.), pp. 1–20, Springer, August 28 – September 1 1995. Lecture Notes in Computer Science, Vol. 969.

[2] A. Agarwal, "Overview of the Alewife project." ALEWIFE SYSTEMS MEMO # 10, Laboratory for Computer Science, MIT, July 30 1990.

[3] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, "The MIT Alewife machine: Architecture and performance," in *Proceedings of the* 22^{nd} *Annual International Symposium on Computer Architecture*, pp. 2–13, June 22-24 1995.

[4] A. Agarwal, D. Chaiken, G. D'Souza, K. Johnson, D. Kranz, J. Kubiatowicz, K. Jurihara, B.-H. Lim, G. Maa, D. Nussbaum, M. Parkin, and D. Yeung, "The MIT Alewife machine: A large-scale distributed-memory mulitprocessor," in *Proceedings of the Workshop on Scalable Shared-Memory Multiprocessors*, June 1990.

[5] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *Theoretical Computer Science*, vol. 71, pp. 3–28, 1990.

[6] S. Ahuja, N. Carriero, and D. Gelernter, "Linda and friends," *IEEE Computer*, vol. 19, pp. 26–34, August 1986.

[7] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation," in *Proceedings of the* 7^{th} *Annual Symposium on Parallel Algorithms and Architectures*, pp. 95–105, July 1995.

[8] T. E. Anderson, D. E. Culler, and D. A. Patterson, "A case for NOW (networks of workstations)," *IEEE Micro*, vol. 15, pp. 54–64, February 1995.

[9] G. Armitage, "Support for multicast over UNI 3.0/3.1 based ATM networks." Internet-Draft, February 22 1996. draft-ietf-ipatm-ipmc-12.txt.

[10] S. M. Armstrong, A. O. Freier, and K. A. Marzullo, "Multicast transport protocol." RFC 1301, February 1992.

[11] R. Asbury, S. G. Frison, and T. Roth, "Concurrent computers ideal for inherently parallel problems," *Computer Design*, pp. 99–107, September 1 1985.

[12] V. Bala, J. Bruck, R. Bryant, R. Cypher, P. de Jong, P. Elustondo, D. Frye, A. Ho, C.-T. Ho, G. Irwin, S. Kipnis, R. Lawrence, and M. Snir, "The IBM external user interface for scalable parallel systems," *Parallel Computing*, vol. 20, pp. 445–462, April 1994.

[13] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir, "CCL: a portable and tunable collective communications library for scalable parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 154–164, February 1995.

[14] H. Bal, R. Hofman, and K. Verstoep, "A comparison of three high speed networks for parallel cluster computing," in *Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC'97)*, pp. 184–197, February 1997.

[15] M. Barnett, R. Littlefield, D. G. Payne, and R. van de Geijn, "Global combine on mesh architectures with wormhole routing," in *Proceedings of the* 7th *International Parallel Processing Symposium (IPPS '93)*, pp. 156–162, April 13–16 1993.

[16] M. Barnett, D. G. Payne, R. A. van de Geijn, and J. Watts, "Broadcasting on meshes with worm-hole routing," tech. rep., Department of Computer Science, University of Texas at Austin, November 2 1993.

[17] M. Barnett, D. G. Payne, R. A. van de Geijn, and J. Watts, "Broadcasting on meshes with wormhole routing," *Journal of Parallel and Distributed Computing*, vol. 35, no. 2, pp. 112–122, 1996.

[18] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the Postal model for message-passing systems," in *Proceedings of the* 4th *Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 13–22, June 29 – July 1 1992.

[19] A. Bar-Noy and S. Kipnis, "Designing broadcasting algorithms in the Postal model for message-passing systems," *Mathematical Systems Theory*, vol. 27, pp. 431–452, September/October 1994.

[20] A. Basu, V. Buch, W. Vogels, and T. von Eicken, "U-Net: a user-level network interface for parallel and distributed computing," in *Proceedings of the* 15th *ACM Symposium on Operating Systems Principles*, December 3-6 1995.

[21] D. P. Bertsekas, C. Özveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, "Optimal communication algorithms for hypercube," *Journal of Parallel and Distributed Computing*, vol. 11, pp. 263–275, 1991.

[22] R. Bhoedjang, T. Rühl, R. Hofman, K. Langendoen, and H. Bal, "Panda: a portable platform to support parallel programming languages," in *Symposium on Experiences with Distributed and Multiprocessor Systems IV*, pp. 213–226, September 1993.

[23] K. P. Birman, "The process group approach to reliable distributed computing," *Communications of the ACM*, vol. 36, pp. 37–53, December 1993.

[24] K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," *ACM Transactions on Computer Systems*, vol. 5, pp. 47–76, February 1987.

[25] K. P. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, vol. 9, pp. 272–314, August 1991.

[26] M. A. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina, "Virtual-memory-mapped network interfaces," *IEEE Micro*, vol. 15, pp. 21–28, February 1995.

[27] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, pp. 29–36, February 1995.

[28] S. H. Bokhari and H. Berryman, "Complete exchange on a circuit switched mesh," in *Proceedings of Scalable High Performance Computing Conference (SHPCC '92)*, pp. 300–306, April 26–29 1992.

[29] R. A. A. Bruce, J. G. Mills, and A. G. Smith, "CHIMP version 2.0 design," Tech. Rep. EPCC-KTP-CHIMP-V2-DESIGN 1.2, Edinburgh Parallel Computing Center, February 24 1994. version 1.2.

[30] R. A. A. Bruce, J. G. Mills, and A. G. Smith, "CHIMP/MPI user guide," Tech. Rep. EPCC-KTP-CHIMP-V2-USER 1.2, Edinburgh Parallel Computing Center, June 1994. version 1.2.

[31] J. Bruck, D. Dolev, C.-T. Ho, R. Orni, and R. Strong, "PCODE: an efficient and reliable collective communication protocol for unreliable broadcast domains," in *Proceedings of the* 9^{th} *International Parallel Processing Symposium*, pp. 130–139, April 1995.

[32] J. Bruck, D. Dolev, C.-T. Ho, M.-C. Rosu, and R. Strong, "Efficient message passing interface (MPI) for parallel computing on clusters of workstations," in *Proceedings of the* 7^{th} *Annual Symposium on Parallel Algorithms and Architectures*, July 1995.

[33] G. D. Burns, "The local area multicomputer," in *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, March 1989.

[34] G. D. Burns, R. B. Daoud, and J. R. Vaigl, "LAM: an open cluster environment for MPI," in *Supercomputing'94*, June 1994.

[35] R. M. Butler and E. L. Lusk, "User's guide to the p4 parallel programming system," Tech. Rep. ANL-92/17, Argonne National Laboratory, Mathematics and Computer Science Division, October 1992.

[36] R. M. Butler and E. L. Lusk, "Monitors, messages, and clusters: the p4 parallel programming system," *Parallel Computing*, vol. 20, pp. 547–564, April 1994.

[37] R. Calkin, R. Hempel, H.-C. Hoppe, and P. Wypior, "Portable programming with the PARMACS message-passing library," *Parallel Computing*, vol. 20, pp. 615–632, April 1994.

[38] N. J. Carriero, D. Gelernter, T. G. Mattson, and A. H. Sherman, "The Linda alternative to message-passing systems," *Parallel Computing*, vol. 20, pp. 633–656, April 1994.

[39] S. Chandra, J. R. Larus, and A. Rogers, "Where is time spent in message-passing and shared-memory programs?," in *The Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*, October 1994.

[40] D. Cheriton, "VMTP: Versatile message transaction protocol." RFC 1045, February 1988. Stanford University.

[41] E. J. Cockayne, E. O. Hare, S. T. Hedetniemi, and E. V. Wimer, "Bounds for the domination number of grid graphs," *Congressus Numerantium*, vol. 47, pp. 217–228, 1985.

[42] R. Cole and O. Zajicek, "The APRAM: incorporating asynchrony into the PRAM model," in *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 169–178, June 18–21 1989.

[43] M. Cosnard and D. Trystram, *Parallel Algorithms and Architectures*. International Thompson Press, 1995.

[44] Cray Research, Inc., *CRAY T3D System Architecture Overview Manual*, 1993.

[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] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[47] 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, 1992.

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

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

[50] L. De Coster, N. Dewulf, and C.-T. Ho, "Efficient multi-packet multicast algorithms on meshes with wormhole and dimension-ordered routing," in *Proceedings of the 1995 International Conference on Parallel Processing*, vol. III, pp. 137–141, August 1995.

[51] S. Deering, "Host extensions for IP multicasting." RFC 1112, August 1989.

[52] D. Dolev and D. Malki, "The Transis approach to high availability cluster communication," *Communications of ACM*, vol. 39, pp. 64–70, April 1996.

[53] J. J. Dongarra and T. H. Dunigan, "Message-passing performance of various computers," Tech. Rep. ORNL/TM-13006, Oak Ridge National Laboratory, February 1996.

[54] J. J. Dongarra and D. W. Walker, "Software libraries for linear algebra computations on high performance computers," *SIAM Review*, vol. 37, no. 2, pp. 151–180, 1995.

[55] B. Duzett and R. Buck, "An overview of the nCUBE 3 supercomputer," in *Proceedings of the Fourth Symposium on the FRONTIERS'92*, pp. 458–464, October 19–21 1992.

[56] D. C. Feldmeier, "An overview of the TP++ transport protocol project," in *High Performance Networks—Frontiers and Experience* (A. Tantawy, Ed.), ch. 8, pp. 157–176, Boston, MA: Kluwer Academic Publishers, 1993.

[57] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proceedings of the ACM SIGCOMM '95*, pp. 342–356, August 1995.

[58] S. Fortune and J. Wyllie, "Parallelism in random access machines," in *Proceedings of the* 10^{th} *Annual ACM Symposium on Theory of Computing*, pp. 114–118, May 1-3 1978.

[59] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine.* The MIT Press, 1994.

[60] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, "PICL: A portable instrumented communication library C Reference Manual," Tech. Rep. ORNL/TM-11130, Oak Ridge National Laboratory, July 1990.

[61] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, "A user's guide to PICL: A portable instrumented communication library," Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, August 1990.

[62] M. Gerla, P. Palnati, and S. Walton, "Multicasting protocols for high-speed wormhole-routing local area networks," in *SIGCOMM'96*, 1996.

[63] P. B. Gibbons, "A more practical PRAM model," in *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 158–168, June 18–21 1989.

[64] W. Gropp and E. Lusk, "User's guide for `mpich`, a portable implementation of MPI," Tech. Rep. ANL/MCS-TM-96-6, Argonne National Laboratory, 199.

[65] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," tech. rep., Argonne National Laboratory, 1996.

[66] S. K. S. Gupta and D. K. Panda, "Barrier synchronization in distributed-memory multiprocessors using rendezvous primitives," in *International Parallel Processing Symposium (IPPS'93)*, pp. 501–506, 1993.

[67] P. J. Hatcher and M. J. Quinn, *Data-Parallel Programming on MIMD Computers.* Cambridge, Massachusetts: The MIT Press, 1991.

[68] J. Heinlein, K. Gharachorloo, S. Dresser, and A. Gupta, "Integration of message passing and shared memory in the Stanford FLASH multiprocessor," in *Proceedings of the* 6^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 38–50, October 1994.

[69] R. Hempel, H.-C. Hoppe, and A. Supalov, *PARMACS 6.0 Library Interface Specification.* Institute for Foundations of Information Technology, German National Research Center for Computer Science, December 17 1992.

[70] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* San Francisco, CA: Morgan Kaufmann Publishers, Inc., second ed., 1996.

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

[72] R. W. Hockney, "The communication challenge for MPP: Intel Paragon and Meiko CS-2," *Parallel Computing*, vol. 20, pp. 389–398, 1994.

[73] C.-T. Ho and S. L. Johnsson, "Distributed routing algorithms for broadcasting and personalized communication on hypercubes," in *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 640–648, 1986.

[74] C.-T. Ho and M.-Y. Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," in *Proceedings of the 1994 International Conference on Parallel Processing*, vol. III, pp. 167–171, August 1994.

[75] C.-T. Ho and M. Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 100–204, February 1995.

[76] C.-T. Ho and M.-Y. Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 200–204, February 1995.

[77] C. C. Huang, E. P. Kasten, and P. K. McKinley, "Design and implementation of multicast operations for ATM-based high performance computing," in *Supercomputing '94*, pp. 164–173, November 1994.

[78] C. C. Huang and P. K. McKinley, "Communication issues in parallel computing across ATM networks," *IEEE Parallel & Distributed Technolog*, pp. 73–86, Winter 1994.

[79] H. H. J. Hum, K. B. Theobald, and G. R. Gao, "Building multithreaded architectures with off-the-shelf microprocessors," in *Proceedings of the* 8^{th} *International Symposium on Parallel Processing*, pp. 288–294, April 1994.

[80] Inktomi Corporation, *The INKTOMI Technology Behind HOTBOT: A White Paper*, 1996.

[81] "Computer on a chip." New Products, IEEE Computer, p. 71, January/February 1972.

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

[83] D. S. Johnson, "The NP-completeness column: An ongoing guide," *Journal of Algorithms*, vol. 5, pp. 147–160, 1984.

[84] H. W. Johnson, *Fast Ethernet: Dawn of a New Network.* Prentice Hall, 1996.

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

[86] K. K. Keeton, T. E. Anderson, and D. A. Patterson, "LogP quantified: The case for low-overhead local area networks," in *Hot Interconnects III: A Symposium on High Performance Interconnects*, August 10–12 1995.

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

[88] A. Khokhar, "c^3: An architecture-independent model for coarse-grained machines," in *Workshop on Models of Parallel Computation*, January 27 1995.

[89] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin-Cummings, 1994.

[90] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, M. R. Anoop Gupta, and J. Hennessy, "The Stanford FLASH multiprocessor," in *Proceedings of the* 21st *International Symposium on Computer Architecture*, pp. 302–313, April 1994.

[91] S. Lamberts, G. Stellner, A. Bode, and T. Ludwig, "Paragon parallel programming on Sun workstations," in *Sun User Group Proceedings*, pp. 87–93, December 1993.

[92] M. Lauria and A. Chien, "MPI-FM: high performance MPI on workstation clusters," *Journal of Parallel and Distributed Computing*, February 1997.

[93] Z. Li, P. H. Mills, and J. H. Reif, "Models and resource metrics for parallel and distributed computation," in *Proceedings of the* 28th *Annual Hawaii International Conference on System Sciences*, January 3–6 1995.

[94] M. Lin, J. Hsieh, D. H. C. Du, J. P. Thomas, and J. A. MacDonald, "Distributed network computing over local ATM networks," *IEEE Journal on Selected Areas on Communications*, vol. 13, no. 4, pp. 733–748, 1995.

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

[96] X. Lin, P. K. McKinley, and L. M. Ni, "Deadlock-free multicast wormhole routing in 2-D mesh multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 8, pp. 793–804, 1994.

[97] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," in *Proceedings of the* 18th *Annual International Symposium on Computer Architecture*, pp. 116–125, May 1991.

[98] R. P. Martin, "HPAM: An active message layer for a network of HP workstations," in *Proceedings of Hot Interconnects II*, August 1994.

[99] P. K. McKinley and C. Trefftz, "Efficient broadcast in all-port wormhole routed hypercubes," in *Proceedings of the 1993 International Conference on Parallel Processing*, vol. II, pp. 288–291, August 1993.

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

[101] P. K. McKinley, Y. Tsai, and D. F. Robinson, "Collective communication in wormhole-routed massively parallel computers," *IEEE Computer*, vol. 28, pp. 39–50, December 1995.

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

[103] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed direct networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 1254–1265, December 1994.

[104] Message Passing Interface Forum, "MPI: A message-passing interface standard," *International Journal of Supercomputing Applications*, vol. 8, no. 3/4, 1994. Special Issue on MPI.

[105] P. Mitra, D. Payne, L. Shuler, R. van de Geijn, and J. Watts, "Fast collective communication libraries, please," Tech. Rep. TR-95-22, Department of Computer Sciences, The Unversity of Texas, June 1995.

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

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

[108] S. Pakin, M. Lauria, and A. Chien, "High performance messaging on workstations: Illinois fast messages (FM) for Myrinet," in *On-line Proceedings of Supercomputing'95 at* `http://www.supercomp.org/sc95/proceedings/`, 1995.

[109] J. Palmer, "A VLSI parallel supercomputer," in *Proceedings of the First Conference on Hypercube Multiprocessors* (M. T. Heath, Ed.), August 26–27 1985.

[110] D. K. Panda, "Optimal phase barrier synchronization in k-ary n-cube wormhole-routed systems using multirendezvous primitives," in *Workshop*

on Fine-Grain Massively Parallel Coordination, in conjunction with International Symposium on Computer Architecture (ISCA '93), pp. 24–26, May 1993.

[111] J.-Y. L. Park and H.-A. Choi, "Circuit-switched broadcasting in tours and mesh networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 184–190, February 1996.

[112] J.-Y. L. Park, H.-A. Choi, N. Nupairoj, and L. M. Ni, "Construction of optimal multicast trees based on the parameterized communication model," in *The 1996 International Conference on Parallel Processing*, August 12–16 1996.

[113] J.-Y. L. Park, S.-K. Lee, and H.-A. Choi, "Fault-tolerant broadcasting in circuit-switched mesh," in *Proceedings of the sixth SIAM Conference on Parallel Processing for Scientific Computing*, pp. 887–890, 1993.

[114] C. Patridge, *Gigabit Networking*. Addison-Wesley, 1993.

[115] D. A. Patterson and D. R. Ditzel, "The case for the reduced instruction set computer," *Computer Architecture News*, vol. 8, pp. 25–33, October 15 1980.

[116] D. A. Patterson and C. H. Séquin, "Design considerations for single-chip computers of the future," *IEEE Transactions on Computers*, vol. C-29, pp. 108–116, February 1980.

[117] D. A. Patterson and C. H. Séquin, "RISC I: a reduced instruction set VLSI computer," in *The* 8^{th} *Annual Symposium on Computer Architecture*, pp. 443–457, May 1981.

[118] J. G. Peters and M. Syska, "Circuit-switched broadcasting in tours networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 246–255, March 1996.

[119] P. Pierce, "The NX message passing interface," *Parallel Computing*, vol. 20, pp. 463–480, April 1994.

[120] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, ch. 8 Commercial Hypercubes: A Performance Analysis, pp. 305–377. MIT Press, 1987.

[121] R. V. Renesse, T. M. Hickey, and K. P. Birman, "Design and performance of Horus: A lightweight group communications system," Tech. Rep. TR-94-1442, Department of Computer Science, Cornell University, August 1994.

[122] D. F. Robinson, *Scalable Multicast Communication in Massively Parallel Computers*. PhD thesis, Department of Computer Science, Michigan State University, 1994.

[123] D. F. Robinson, D. Judd, P. K. McKinley, and B. H. C. Cheng, "Efficient multicast in all-port wormhole-routed hypercubes," *Journal of Parallel and Distributed Computing*, vol. 31, pp. 126–140, 1995.

[124] D. F. Robinson, P. K. McKinley, and B. H. C. Cheng, "Optimal Multicast Communication in Torus Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 1029–1042, October 1995.

[125] S. H. Rodrigues, "Building a better byte stream," Master's thesis, University of California at Berkeley, 1996.

[126] M. Schmidt-Voigt, "Efficient parallel communications with the nCUBE 2S processor," *Parallel Computing*, vol. 20, pp. 509–530, April 1994.

[127] H. D. Schwetman, "Csim: A C-based, process oriented simulation laguage," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corporation, 1985.

[128] D. S. Scott, "Efficient all-to-all communication patterns in hypercube and mesh topologies," in *Proceedings of the* 6^{th} *Conference on Distributed Memory Concurrent Computers*, pp. 398–403, 1991.

[129] C. L. Seitz, "The Cosmic cube," *Communications of the ACM*, vol. 28, pp. 22–33, January 1985.

[130] A. Skjellum, S. G. Smith, N. E. Doss, A. P. Leung, and M. Morari, "The design and evolution of Zipcode," *Parallel Computing*, vol. 20, pp. 565–596, April 1994.

[131] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, Eds., *MPI: The Complete Reference.* Cambridge, Massachusetts: The MIT Press, 1996.

[132] G. Stellner, A. Bode, S. Lamberts, and T. Ludwig, "NXLib – a parallel programming environment for workstation clusters," in *PARLE'94 Parallel Architectures and Languages Europe*, no. 817 in Lecture Notes in Computer Science, pp. 745–748, Springer, 1994.

[133] G. Stellner, A. Bode, S. Lamberts, and T. Ludwig, "Developing application for multicomputer systems on workstations," in *High-Performance Computing and Networking, International Conference and Exhibition*, no. 797 in Lecture Notes in Computer Science, pp. 286–292, Springer, 1994.

[134] W. T. Strayer, M. J. Lewis, and R. E. Cline, Jr., "XTP as a transport protocol for distributed parallel processing," in *Proceedings of the USENIX Symposium on High-Speed Networking*, pp. 91–102, August 1-3 1994.

[135] V. Sunderam, "PVM: a framework for parallel distributed computing," *Concurrency: Practice and Experience*, vol. 2, December 1990.

[136] S. S. Takkella and S. R. Seidel, "Broadcast and complete exchange algorithms for meshes," in *Proceedings of the Conference on Scalable High-Performance Computing (SHPCC '94)*, pp. 422–428, May 23–25 1994.

[137] R. Thakur and A. Choudhary, "All-to-all communication on meshes with wormhole routing," in *Proceedings of the* 8th *International Parallel Processing Symposium (IPPS '94)*, pp. 561–565, April 1994.

[138] R. Thakur, A. Choudhary, and G. Fox, "Complete exchange on a wormhole routed mesh," in *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 131–135, January 31–February 2 1994.

[139] Thinking Machines Corporation, CMMD *and Active Messages*, 1993.

[140] C. Topolcic, "Experimental internet stream protocol, version 2 (ST-II)." RFC 1190, October 1990.

[141] Y. Tsai, Y. Huang, and P. K. McKinley, "Performance evaluation of barrier synchronization in ATM networks," in *Proceedings of the* 5th *International Conference on Computer Communications and Networks*, pp. 144–151, October 16–18 1996.

[142] Y. Tsai and P. K. McKinley, "Broadcast in all-port wormhole-routed 3D mesh networks using extended dominating sets," in *International Conference on Parallel and Distributed Systems*, pp. 120–127, December 19–23 1994.

[143] Y. Tsai and P. K. McKinley, "A dominating set model for broadcast in all-port wormhole-routed 2D mesh networks," in *Proceedings of the* 8th *ACM International Conference on Supercomputing*, pp. 126–135, July 11–15 1994.

[144] Y. Tsai and P. K. McKinley, "An extended dominating node approach to collective communication in wormhole-routed networks," in *Proceedings of the Scalable High Performance Computing Conference*, pp. 199–206, May 23–25 1994.

[145] Y. Tsai and P. K. McKinley, "A broadcast algorithm for all-port wormhole-routed torus networks," in *The Fifth Symposium on the Frontiers of Massively Parallel Computation*, pp. 529–536, February 6–9 1995.

[146] Y. Tsai and P. K. McKinley, "A broadcast algorithm for all-port wormhole-routed torus networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 876–885, August 1996.

[147] Y. Tsai and P. K. McKinley, "An extended dominating node approach to collective communication in wormhole-routed networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, pp. 41–58, January 1997.

[148] Y.-C. Tseng, D. K. Panda, and T.-H. Lai, "A trip-based multicasting model in wormhole-routed networks with virtual channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, pp. 138–150, February 1996.

[149] L. W. Tucker and A. Mainwaring, "CMMD: active messages on the CM-5," *Parallel Computing*, vol. 20, pp. 481–496, April 1994.

[150] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, pp. 103–111, August 1990.

[151] R. van de Geijn, "On global combine operations," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 324–328, 1994.

[152] T. von Eicken, A. Basu, and V. Buch, "Low-latency communication over ATM networks using active messages," *IEEE Micro*, vol. 15, pp. 46–53, February 1995.

[153] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active messages: a mechanism for integrated communication and computation," in *Proceedings of the* 19th *International Symposium on Computer Architecture*, pp. 256–267, May 1992.

[154] K. Verstoep, K. Langendoen, and H. Bal, "Efficient reliable multicast on Myrinet," in *Proceedings of the 1996 International Conference on Parallel Processing*, pp. III–156–III–165, August 12–16 1996.

[155] R. van Renesse, K. P. Birman, and S. Maffeis, "Horus, a flexible group communication system," *Communications of the ACM*, vol. 39, pp. 76–83, April 1996.

[156] J. Watts and R. van de Geijn, "A pipelined broadcast for multidimensional meshes," *Parallel Processing Letters*, vol. 5, no. 2, pp. 281–292, 1995.

[157] XTP Forum, *Xpress Transport Protocol Specification: XTP version 4.0*, March 1 1995.

[158] Z. Xu and K. Hwang, "Modeling communication overhead: MPI and MPL performance on the IBM SP2," *IEEE Parallel & Distributed Technology*, pp. 9–23, Spring 1996.

[159] 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, October 1992.

[160] H. Zhou and A. Geist, "Faster (ATM) message passing in PVM," in *The* 9th *International Parallel Processing Symposium: Workshop on High-Speed Network Computing*, April 1995.