ABSTRACT

EVALUATING THE COMMUNICATIONS CAPABILITIES OF THE GENERALIZED HYPERCUBE INTERCONNECTION NETWORK

by

Sanjay Krishnamurthy

This thesis presents results of evaluating the communications capabilities of the generalized hypercube interconnection network. The generalized hypercube has outstanding topological properties, but it has not been implemented in a large scale because of its very high wiring complexity. For this reason, this network has not been studied extensively in the past. However, recent and expected technological advancements will soon render this network viable for massively parallel systems.

We first present implementations of randomized many-to-all broadcasting and multicasting on generalized hypercubes, using as the basis the one-to-all broadcast algorithm presented in [3]. We test the proposed implementations under realistic communication traffic patterns and message generations, for the all-port model of communication. Our results show that the size of the intermediate message buffers has a significant effect on the total communication time, and this effect becomes very dramatic for large systems with large numbers of dimensions.

We also propose a modification of this multicast algorithm that applies congestion control to improve its performance. The results illustrate a significant improvement in the total execution time and a reduction in the number of message contentions, and also prove that the generalized hypercube is a very versatile interconnection network.

EVALUATING THE COMMUNICATIONS CAPABILITIES OF THE GENERALIZED HYPERCUBE INTERCONNECTION NETWORK

by

Sanjay Krishnamurthy

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Department of Electrical and Computer Engineering

January 1998

APPROVAL PAGE

EVALUATING THE COMMUNICATIONS CAPABILITIES OF THE GENERALIZED HYPERCUBE INTERCONNECTION NETWORK

Sanjay Krishnamurthy

Dr. Sotirios G. Ziavras, Advisor Date
Associate Professor of Electrical and Computer Engineering and
Computer and Information Science, NJIT

Dr. Jacob Savir, Committee Member Date
Professor and Director of Electrical and Computer Engineering, NJIT

Dr. Edwin Hou, Committee Member Date
Associate Professor of Electrical and Computer Engineering and
Computer and Information Science, NJIT

BIBLIOGRAPHICAL SKETCH

Author: Sanjay Krishnamurthy

Degree: Master of Science

Date: January 1998

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering
  New Jersey Institute of Technology, Newark, NJ, 1998

- Bachelor of Science in electronics and Communications
  The National Institute of Engineering, Mysore, India, 1995

Major: Computer Engineering

To my beloved and supportive family

ACKNOWLEDGEMENT

I sincerely express my gratitude and appreciation to Dr. Sotirios G. Ziavras, my thesis advisor and the main driving force behind the entire thesis. He has served as my research supervisor and has guided me with valuable ideas for the successful completion of this thesis. I thank Dr. Jacob Savir and Dr. Edwin Hou for their active participation in the committee.

Many of my fellow graduate students working in the Parallel Computing area do deserve credit for contributing innovative ideas and suggestions. Cooperation from Brenda Walker, Lisa Fitton and all other staff members of the Electrical Engineering department for their understanding and extension of their patient support over the past one and a half years has been abundant and admirable.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF TABLES
(Continued)

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The ever increasing demand for raw processing power to compute many of the age-old and new computational problems has taken the industry to limits in the design of single-processor computers with very high computing power. However, no matter what speed and/or computing power is obtained by a single-processor computer, a parallel computer with many processors could carry out computation-intensive jobs more effectively. This has lead to the development of massively parallel computers with thousands of processors.

1.1 Parallel Processing

Basically, two primary aspects will dominate the massively parallel processing field. These two aspects might as well be referred to as the parallel-computing primitives. One of these primitives is the development of high-level programming languages that could take into consideration the shared-memory space for DSM (Distributed Shared-Memory) implementations. The other aspect is the technique used to interconnect many powerful processors together in a scalable framework.

Many computation-intensive applications, like, among others, weather forecasting, simulation of physical phenomena, aerodynamics, simulation of neural networks, seismology, and real-time image processing all come under the purview of massively-parallel computers. The greater the computing power, the better are the results obtained (e.g., higher accuracy). The goal of building computers capable of PetaFLOPS performance (i.e., 10^{15} floating-point operations per second) by the year 2007 was identified recently by numerous federal agencies as being an absolutely essential task. Problems related to PetaFLOPS activities currently seem to be insurmountable, primarily because of the difficulties in developing low-complexity, high-bisection bandwidth, and low-latency interconnection networks capable of connecting thousands of processors together in the DSM framework [1, 4, 6, 12].

Current, feasible approaches to massively parallel processing use bounded-degree networks such as meshes with a low degree of connection (e.g., Intel Paragon and Cray Research MPP). The main obstacles with these approaches are the resulting large diameter and average inter-processor distance, and the small bisection bandwidth. To improve the topological properties of bounded-degree networks, switches may be incorporated in the design [11]. However, such approaches are not appropriate for very high-performance computing.

The generalized hypercube network [10] is better on all of the above properties but its very high VLSI (i.e., wiring) complexity is a Herculean task because of heavy scalability problems. Contrary to the popular direct binary hypercube [5] that contains only two nodes in each dimension, the generalized hypercube forms a fully connected subsystem with many nodes in each dimension. It is well known that the former is not scalable in practice [1, 6, 8], and therefore the latter (i.e., the generalized hypercube) has even more dramatic scalability problems. However, with an alternative to wiring, such as using hybrid electronic/optical interconnection technologies or electronic switches, the generalized hypercube seems to be an ideal interconnection network for the next generation of massively-parallel systems [9, 12, 13, 15]. An architecture capable of near-PetaFLOPS performance by the year 2005 was designed and analyzed, in terms of feasibility and performance, under a New Millenium Computing Point Design grant awarded jointly to NJIT by NSF, DARPA, and NASA [12, 13]. This architecture employs free-space optics for the implementation of a *2-D* generalized hypercube of *8*-processor cards, and contains a total of *10,368* processors. Other designs implement generalized hypercubes by substituting small switches [9] or optical fibers [16] for processor-to-processor wires in each fully connected subsystem.

This thesis investigates the implementation of important communications primitives, like broadcasting and multicasting, on generalized hypercubes. Broadcasting is the distribution of a message or a group of messages from one (or multiple) source processor(s) to all other processors. It can be considered a special case of multicasting, where a single (or multiple) source processor(s) distributes a message or a group of messages to a subset of the processors. All-to-all broadcasting is the distribution of a message or a group of messages from all processors to every other processor. All algorithms in this thesis assume *store-and-forward message (packet) switching* (i.e., an intermediate processor receives the entire message before attempting to forward it) and the *all-port model* where a processor is capable of using all of its communications ports at the same time for the same or different messages (i.e., a processor could communicate with all of its neighbors at the same time).

1.2 Interconnection Networks for Parallel Architectures

One of the most important criteria that decides on the effectiveness of a parallel computer is the technique or methodology used to interconnect the thousands of processing elements or processors into a tightly integrated unit. This is what is referred to as the interconnection network. Interconnection networks can broadly be divided as either *static* or *dynamic* interconnection networks. Static networks use direct links that are fixed once built. Message passing multi-computers usually rely on a static network (such as the hypercube interconnection network) and shared-memory multi-processor systems opt for dynamic interconnections, such as those implemented by a bus. Obviously an ideal interconnection network would interconnect together all pairs of processors in the system; however, such an interconnection network is neither practical nor feasible. Figure 1.1 shows a fully connected system with 16 processors. The VLSI implementation of such a large system using current wiring technologies is an impossible task.

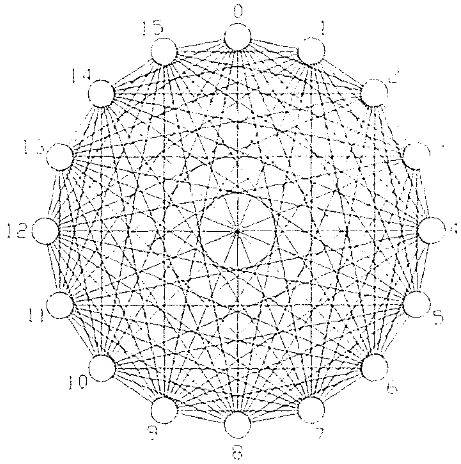

Figure 1.1: *Fully-connected system with 16 processors*

1.2.1 Mesh

A 3×3 mesh network is shown in Figure 1.2(a). This is a feasible popular architecture that has been implemented on computers such as the Good Year MPP, CRAY research T3D, and Intel Paragon. There exist many variations to the mesh interconnection network. In general, a symmetric k-dimensional mesh with $N = n^k$ nodes has an interior node degree (i.e. number of edges) of $2k$ and a network diameter of $k \times (n-1)$. Figure

1.2(b) shows a variation of the mesh allowing wraparound connections. This variation was implemented on the Illiac IV.

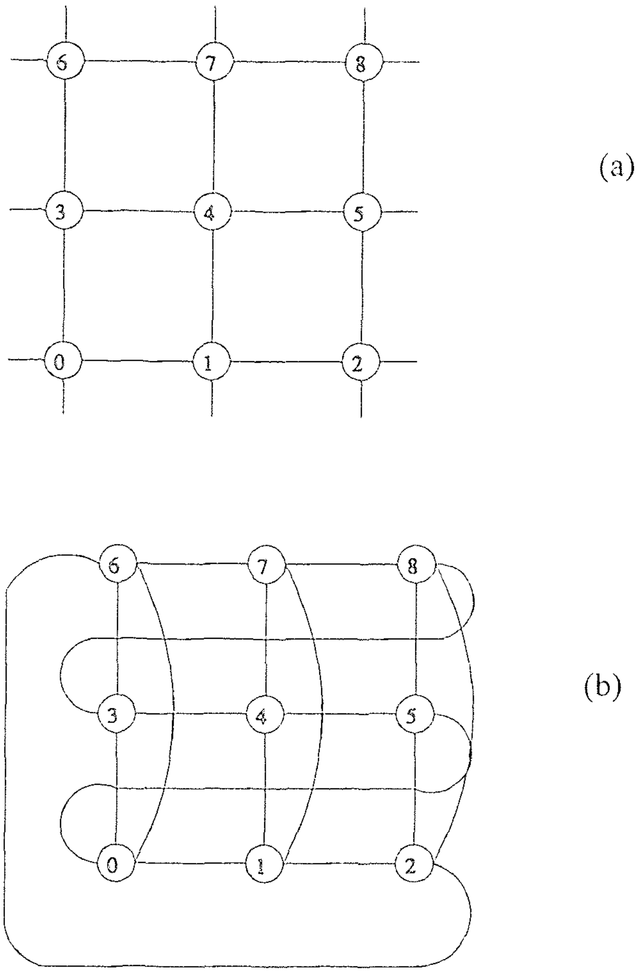

Figure 1.2 (a): *A 3×3 mesh.* **(b)** *A variation of the 3×3 mesh*

1.2.2 Torus

The torus is one of the most widely used inter connection networks in commercial parallel computers. A torus contains communication links that connect the smallest numbered node in a dimension directly to the largest numbered node in the same dimension. This type of connection forms a ring where information can be transferred unidirectionally from one node through all of the nodes in the same dimension, and back to the original node. In general, an $n \times n$ torus has a node degree of *4* and a diameter of $2\times \lfloor n/2 \rfloor$. Figures 1.3(a) - (b) show a *1-D* and *2-D* torus interconnection network, respectively.

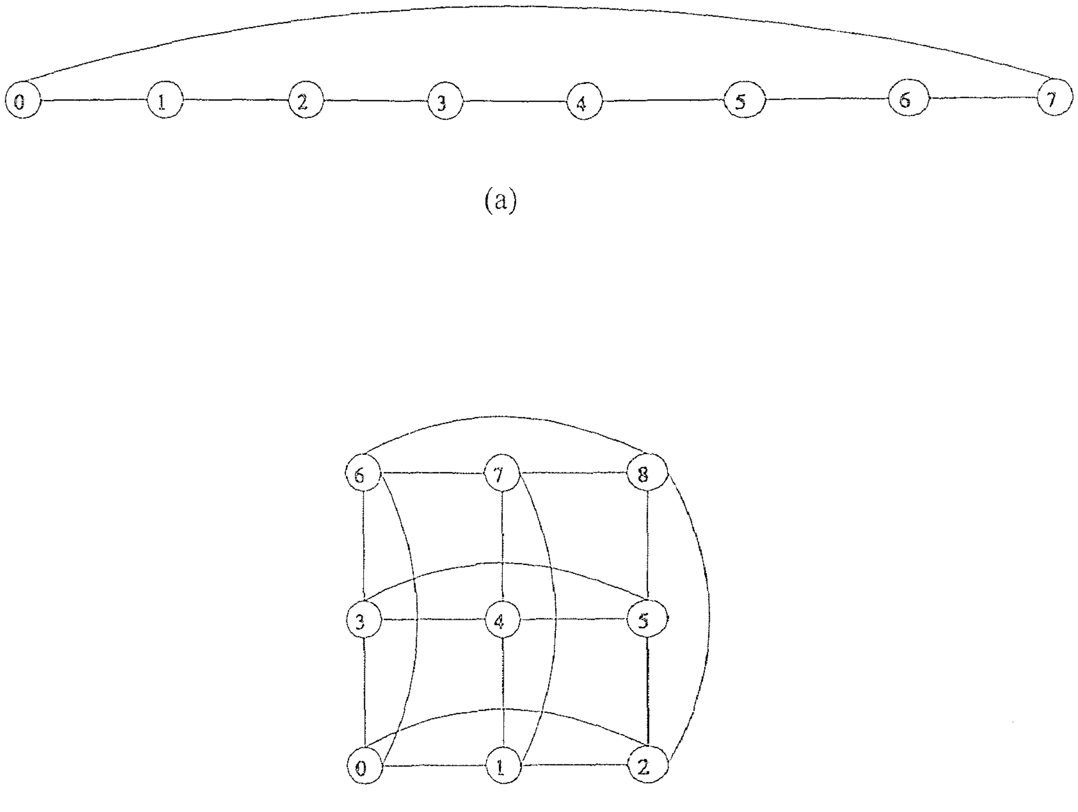

Figure 1.3 (a): *1-D torus.* **(b)** *2-D torus*

1.2.3 Hypercube

The hypercube is a binary n-cube architecture. An n-cube consists of $N=2^n$ nodes spanning n dimensions, with two nodes per dimension. Two nodes are neighbors in the n-cube if and only if their n-bit binary addresses differ in a single bit. A 3-cube with 8 nodes is shown in Figure 1.4. The node degree increases linearly with respect to the number of dimensions, making it difficult to consider the hypercube a scalable architecture. The main characteristic for an architecture to survive in future systems is packaging efficiency and scalability to allow modular growth.

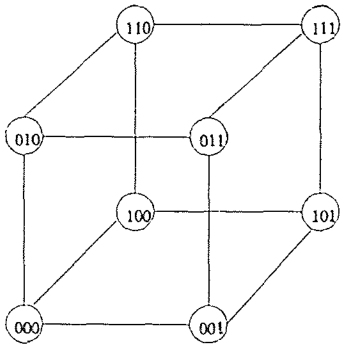

Figure 1.4: *3-cube*

1.2.4 The Generalized Hypercube Network

The symmetric k-ary n-dimensional generalized hypercube, denoted by $GH_{n,k}$, is a graph with $N = k^n$ nodes (processors), each one being represented by an n-digit number in radix-k arithmetic [10]. In this symmetric network, each processor is connected to $n \times (k-1)$ other processors. Any two directly connected processors are referred to as neighboring processors and their n-digit addresses differ in only one radix-k digit. Each processor in the generalized hypercube has a *degree* (i.e., the number of edges) of $n \times (k-1)$ and a diameter (i.e., the maximum shortest distance between any pair of processors) of n. Figure 1.5 shows the generalized hypercube $GH_{2,7}$.

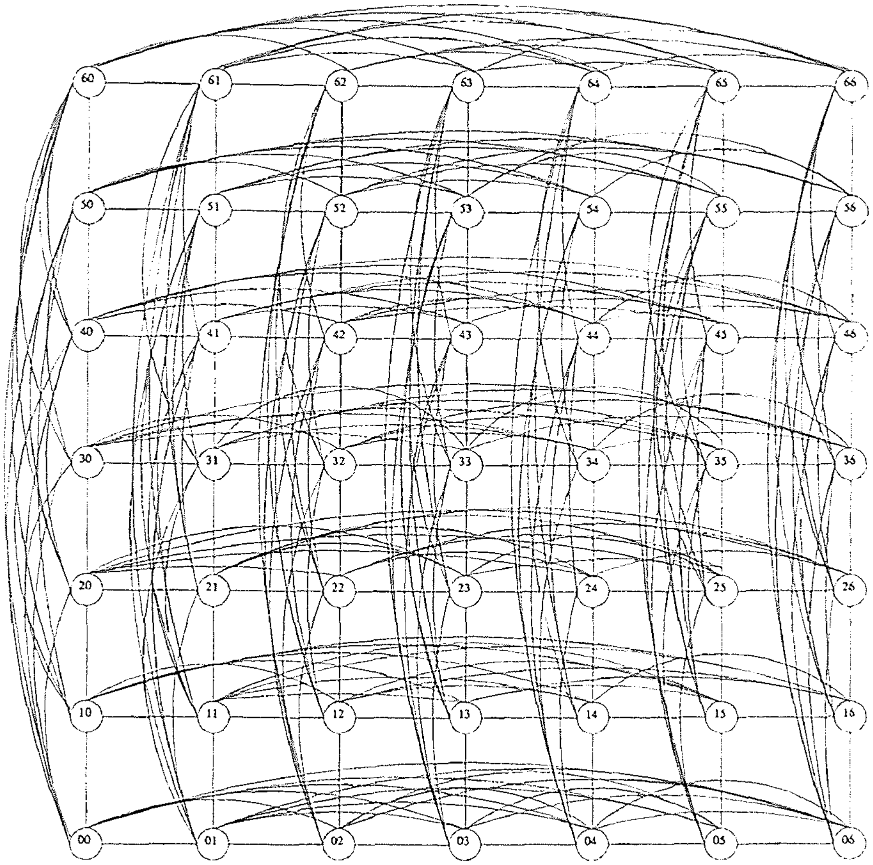

Figure 1.5: *Generalized Hypercube $GH_{2,7}$*

The generalized hypercube interconnection network has not only outstanding topological properties (e.g., a very small diameter) but also a very high bisection width (i.e., the minimum number of interconnections between two equal halves) when compared to the torus (i.e., the k-ary n-cube, which is the most widely used network in commercial systems nowadays) or the mesh with an equal number of processors. This means that the generalized hypercube results in outstanding performance for large systems with thousands of processors and heavy inter-processor communication traffic. Unfortunately, its implementation using only wires is impractical as the number of wires for data transfers increases exponentially with the number of processors. The system proposed in [12, 13], which will be capable of near-PetaFLOPS performance by the year 2005, has *10,368* processors. It makes use of hybrid electronic/optical technologies to implement a generalized hypercube. Table 1.1 compares the numbers of channels in the k-ary n-cube (i.e., the n-D torus) and the generalized hypercube $GH_{n,k}$ with the same number of processors (i.e., k^n). For example, assuming bi-directional channels for full-duplex communications and 64-bit data channels, systems with 10,648 processors (with $n=3$ and $k=22$) will have the following complexities:

- *4,088,832* wires for the *22*-ary *3*-cube with a diameter of *33*
- *42,932,736* wires for the *3*-D $GH_{3,22}$ with a diameter of 3

Table 1.1: *Comparison of interconnection networks, assuming full-duplex bi-directional data channels*

Network Model	**Number of Channels**	**Diameter**
k-ary n-cube	$2 \times n \times k^n$	$n \times \lfloor k/2 \rfloor$
$GH_{n,k}$	$(k-1) \times n \times k^n$	n

A common approach to designing communications algorithms for inter-processor communication networks, such as the generalized hypercube, the mesh, and the torus, has been the embedding of spanning (sub)graphs with special properties into these networks. In this thesis, we first make use of the spanning graphs for the generalized hypercube proposed in [3], for the performance assessment of corresponding one-to-all and all-to-all broadcast algorithms under realistic communication traffic. Relevant work from [3] is summarized in the next chapter. Further work on multicasting is also presented later.

1.3 Motivation and Objectives

As previously mentioned, most of the current commercial massively-parallel computers make use of the torus interconnection network due to the unavailability of an interconnection network with a greater number of connections that still offers efficient packaging. The parallel computing community has not yet implemented large generalized hypercubes because of scalability problems in terms of wiring. With recent advances in technology and the availability of alternatives to wiring, such as hybrid electronic/optical interconnection technologies and electronic switches, the generalized hypercube seems to be a very good interconnection network for scalable parallel computers. A recently introduced class of architectures employ the generalized hypercube as the basic building block; these building blocks are highly overlapped in each dimension to produce systems of reasonable hardware complexity and outstanding topological properties [15]. The latter architectures can be implemented feasibly even with current electronic technologies. Current designs have a small node degree (e.g. torus) and hence a lower performance. The generalized hypercube, on the other hand, has a higher node degree and can result in a dramatically better performance.

With advances in technology that will make systems using the generalized hypercube scalable and practical in the immediate future, it is imperative that a detailed study be made of the communications capabilities of such a system. While most of the available technical literature concerning hypercubes available tries to do a theoretical study on the aspects of the system, this thesis is projected to study in detail some of the capabilities of the system by actually simulating such a system.

CHAPTER 2

COMMUNICATIONS OPERATIONS ON THE GENERALIZED HYPERCUBE

2.1 Widely used Communications Operations

There are some operations that are so commonplace with all parallel computers that it is sometimes a good idea to hardwire (i.e. implement dedicated hardware circuitry) the interconnection to handle these operations effectively. On any parallel computer, some of the widely used communications primitives are broadcast, scatter and reduction operations. These are briefly summarized in the following subsections.

2.1.1 Broadcast Operation

A communication operation involving the transfer of the same message or group of messages from a source processor to all other processors in the system is referred to as a *broadcast*. This is referred to as one-to-all broadcasting when a single source processor initiates a message transfer to all other processors. A worst case scenario is when all the nodes in the system initiate a one-to-all transfer at the same time. This is a special case of the broadcast operation and is referred to as all-to-all broadcasting.

2.1.2 Scatter Operation

The scatter operation involves the transfer of different messages or groups of messages from a source processor to all other processors in the system. Similar to the broadcast operation, one-to-all scattering involves the transfer of different messages from a single source processor to all other processors in the system and the all-to-all scatter operation refers to the simultaneous transfer of different messages from all the processors in the system.

2.1.3 Reduction Operation

The *reduction* operation is the opposite of the broadcast operation and is the collection and combining of messages by a single processor from all other processors in the system. This kind of operation might be required in operations involving some kind of synchronization where all processors proceed with the next step in the execution cycle only when all of them have attained a particular value which is collected by the single processor which initiated the reduction operation.

2.2 Implementation

One technique often used to implement communications operations on any interconnection network is to embed spanning trees specially designed for each network. Fragopoulou in [3] presents a novel way of implementing the one-to-all and all-to-all broadcast communication primitives on the generalized hypercube using such a technique. The spanning tree is created with the source processor as the root and all other processors appear in subtrees of the spanning tree. However, taking advantage of the fact that the generalized hypercube is a symmetric network, the authors create at static time a spanning tree/subgraph rooted at the (source) processor with address zero, and at run time they can create another spanning tree rooted at any other processor through address transformation.

2.2.1 Binary Spanning Tree

Let us summarize the procedure for creating a spanning tree rooted at the (source) node $s=0^n$ in the generalized hypercube $GH_{n,k}$. 0^n denotes a string of n consecutive 0's. All processors appearing at the same minimum distance from the source processor, s, are grouped together and then in each group necklaces are created. A *necklace* is defined as an ordered group of processors, each one derived from the subsequent one in the same group cyclically, through rotation. The *rotation* of a node v, $v = v_{n-1} \ldots v_{i+1} v_i v_{i-1} \ldots v_0$ in the $GH_{n,k}$ produces the node $R(v)$ given by:

$$R(v) = v_{n-2} v_{i+1} v_i v_{i-1} \ldots v_0 r(v_{n-1})$$

where

$$r(v_{n-1}) = \begin{cases} 0 & if\ v_{n-1} = 0 \\ v_{n-1} \bmod (k-1) + 1 & if\ v_{n-1} \neq 0 \end{cases}$$

For example, for the generalized hypercube $GH_{3,5}$, if $v = 342$ and $u = 023$, then $R(v) = 424$ and $R(u) = 230$. Thus, all processors in a necklace are at the same distance from the source s. A necklace consists of at most $n \times (k - 1)$ processors. A *full necklace* contains $n \times (k - 1)$ distinct processors.

The nodes at each given distance i from node 0^n in the $GH_{n,k}$, where $1 \leq i \leq n$, are collections of necklaces. More definitions are pertinent [3]. The *binary correspondent* of a node is the binary number derived by substituting a *1* for each non-zero digit in its address. The *generator node* of a necklace is the node in the necklace with the largest binary correspondent. If more than one such node is found, we choose the one with the largest address. The *displacement*, $D(v)$, of a node v is the minimum number of rotations applied on v that produce the generator node. The *period*, $P(v)$, of a node v is the number of node s in its necklace. An *unfolded necklace* contains $n \times (k\text{-}1)$ ordered nodes, not necessarily distinct, where each node is obtained from its subsequent one through rotation. A full necklace is identical to its unfolded necklace. The unfolded necklace of a non-full necklace with P nodes is obtained by repeating the latter necklace $n \times (k-1)/P$ times. Table 2.1 shows the unfolded necklaces in the generalized hypercubes $GH_{3,3}$ and $GH_{3,4}$.

Table 2.1: *The necklaces of $GH_{3,3}$ and $GH_{3,4}$*

Necklaces of $GH_{3,3}$		**Necklaces of $GH_{3,4}$**	
Distance	**Nodes**	**Distance**	**Nodes**
$d=0$	000	$d=0$	000
$d=1$	200, 020, 002, 100, 010, 001	$d=1$	300, 030, 003, 200, 020, 002, 100, 010, 001
$d=2$	220, 022, 102, 110, 011, 201 210, 021, 202, 120, 012, 101	$d=2$	330, 033, 203, 220, 022, 102, 110, 011, 301 310, 031, 303, 230, 023, 202, 120, 012, 101 320, 032, 103, 210, 021, 302, 130, 013, 201
$d=3$	222, 122, 112, 111, 211, 221 212, 121	$d=3$	333, 233, 223, 222, 122, 112, 111, 311, 331 332, 133, 213, 221, 322, 132, 113, 211, 321 323, 232, 123, 212, 121, 312, 131, 313, 231

Assume that the source node is $s = 0^n$. A shortest path, *balanced spanning tree* rooted at $s = 0^n$ and denoted by BST_0^n is now constructed using the following function. For processor v with displacement $D(v) = i$, let p be the position of its first non-zero digit cyclically to the left of the position $(n - 1 - i) \bmod n$. Then,

$$Parent^{BST_{0^n}}(v) = \begin{cases} 0 & if\ v = 0^n \\ v_{n-1} \dots v_{p+1} 0 v_{p-1} \dots v_0 & if\ v \neq 0^n \end{cases}$$

The spanning tree rooted at 0^3 of the $GH_{3,3}$ is shown in Figure 2.1.

2.2.2 Binary Spanning Graph

To derive the *spanning subgraph* BSG_0^n rooted at node 0^n, we replace each non-full necklace with its corresponding unfolded necklace in the BST_0^n. Figure 2.2 shows the BSG_0^3 of the $GH_{3,3}$. For the generalized hypercube $GH_{3,4}$ all the necklaces are full and hence there is no difference between the necklaces and the unfolded necklaces. Hence the BST_0^3 and BSG_0^3 for the $GH_{3,4}$ would be the same.

Table 2.2: *The unfolded necklaces of* $GH_{3,3}$

Unfolded Necklaces of $GH_{3,3}$	
Distance	Nodes
d=0	000
d=1	200, 020, 002, 100, 010, 001
d=2	220, 022, 102, 110, 011, 201 210, 021, 202, 120, 012, 101
d=3	222, 122, 112, 111, 211, 221 212, 121, 212, 121, 212, 121

Table 2.2 shows the unfolded necklaces of the $GH_{3,3}$. Nodes belonging to full necklaces have a single path to node 0^n in the BSG_0^n. In contrast, nodes with period P belonging to non-full necklaces have $n \times (k-1)/P$ paths. We use the BST_0^n for one-to-all broadcasting and the BSG_0^n for all-to-all broadcasting. The multiple paths in the BSG_0^n make room for data to be spread across channels, so that the bandwidth requirements of data channels can be reduced, which, in turn, reduces the total number of communication cycles.

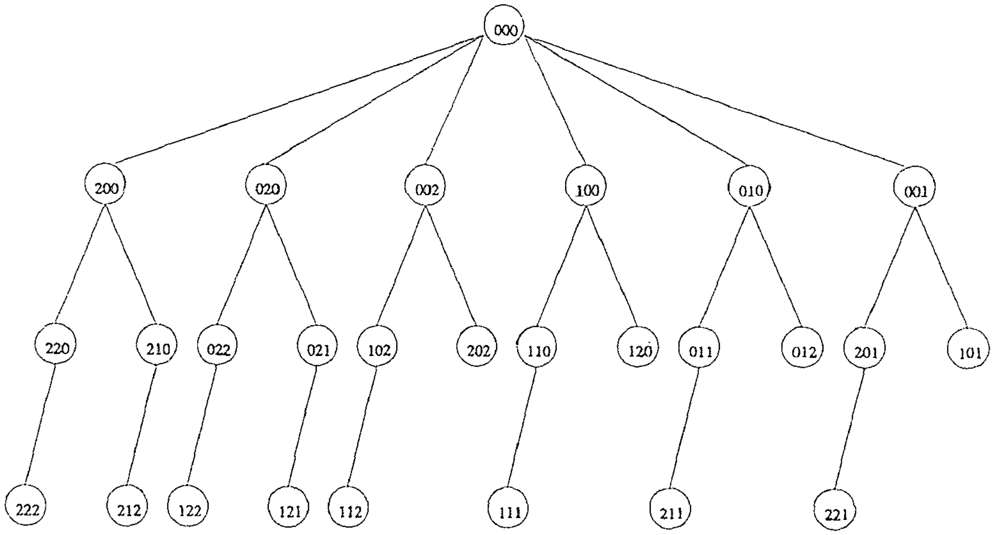

Figure 2.1: *The binary spanning tree* BST_0^3 *of the* $GH_{3,3}$

For communications operations originating at a processor other than the processor $s = 0^n$, the statically created tree/subgraph is translated with respect to the new source processor. The *translation* of a processor v with respect to s results in the processor $t=T_s(v)$, such that $t_i = (v_i + s_i) \bmod k$, where $0 \leq i \leq (n - 1)$. Both the rotation and translation operations preserve the distance between processors. This attribute helps in avoiding contention in all-to-all broadcasting. More specifically, messages are interleaved to completely avoid contention.

Whereas these algorithms for one-to-all and all-to-all broadcasting are asymptotically optimal, they may not perform well under realistic conditions where messages are generated randomly. Such an investigation is carried out in this thesis. We also investigate the performance of a technique that uses these spanning trees/subgraphs for the implementation of multicasting (i.e., one-to-many communication), again under realistic message generations. By the way, all-to-all broadcasting can be viewed as a special case of multicasting with a single source, where all processors are destinations.

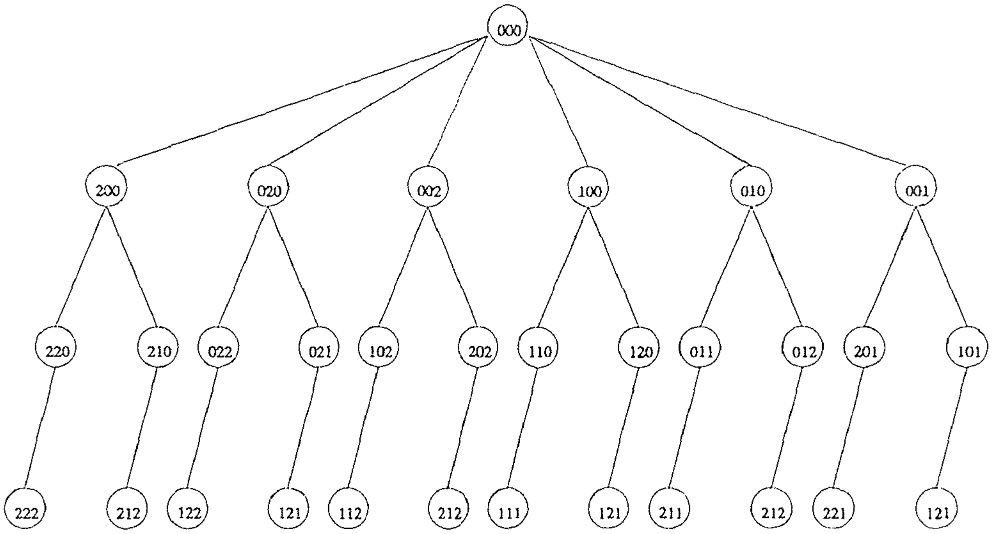

Figure 2.2: *The binary spanning graph BSG_0^3 of the $GH_{3,3}$*

CHAPTER 3

INVESTIGATION OF COMMUNICATIONS PRIMITIVES

The one-to-all and all-to-all broadcast techniques in [3] do not result in message contentions if no processor initiates a broadcast till all previous, if any, broadcasts have been fully completed. This offers a substantial limitation when dealing with practical systems where a random number of processors may initiate communications operations in any cycle. Under the latter scenario, there may be considerable numbers of contentions on the data channels. The same problem persists in the case of multicasting, where a random number of processors initiate a message transfer, the only difference here being that only a subset of the total number of processors receive the message.

Since only one message is allowed to traverse a given channel towards its destination at any time, any held up messages need to wait at the corresponding intermediate processor. An immediate consequence arising as a result of this complication is that the intermediate processor now must have buffer space to accommodate for these messages. The buffer size cannot be infinite in practice, and hence the time taken by the communications operation to complete now also depends on the buffer size. The effect of the buffer size on the total communication time is also studied in this thesis, through simulation. We point out in the rest of this chapter potential message contention problems for the existing communications algorithms. We also present algorithms for the implementation of many-to-many multicasting on generalized hypercubes. Simulation results for all these algorithms are presented in the next chapter.

3.1 Randomized Many-to-All Broadcast Operation

In the *randomized many-to-all broadcast*, each processor randomly tries to initiate a one-to-all broadcast in every cycle using the Poisson distribution. The Poisson distribution is widely accepted for message generation in simulations of parallel systems. The worst case would result if all the processors were initiating broadcasts, resulting in all-to-all broadcasting. Although the all-to-all broadcast algorithm in [3] deals with this worst case scenario in a way that avoids any message contention by using the spanning subgraphs, it guarantees this under the assumption that only communication activities related to a single all-to-all broadcast are present at any time. However, message contentions are possible if activities related to new and old (i.e., not yet completed) many-to-all and all-to-all broadcasts are simultaneously present. One of our objectives is to thoroughly study the cases that result in such message contentions.

In each cycle, every processor calculates randomly the probability of initiating a message. A threshold value of *2/3 × (Maximum Probability)* was set for the Poisson distribution and all processors having a probability value greater than this threshold initiate a message transfer. Since there is a high probability that more than one processor may initiate a message in a given cycle, and there may also be several message initiations in successive cycles (i.e., processors in the system need not wait till all messages generated in previous cycles have reached their destinations), there may be considerable channel contention.

3.2 Many-To-Many Multicast

Multicasting with a single source is the distribution of a message from a single processor to many, but not necessarily all, processors in the system. Many-to-many multicasting (i.e., multicasting with several sources) is several simultaneous multicasts of the former type, without necessarily the same set of destinations. Special cases of multicasting include one-to-all broadcasting (i.e., one source processor and all other processors are destinations) and all-to-all broadcasting (i.e., every processor is a source and broadcasts a message to all other processors in the system).

The spanning trees/subgraphs created in [3] for broadcasting may be used to selectively distribute the messages to the destination processors. Identical messages for several destination processors residing in the same subtree could be clubbed as one message as long as they follow the same path from the source processor. This could drastically reduce the network traffic. Such a clubbing algorithm and the main multicast algorithm are proposed in the following two subsections, respectively.

3.2.1 Brute-Force Clubbing Algorithm

Given a group of destinations for multicasting from a single source, all destinations having the same displacement (as defined in [3] and Chapter *3*) are clubbed together as they all belong to the same sub-tree. We assume that each transmitted message contains a header with the source address and a group of destination addresses. Destinations in the group with the smallest number of common digits in their addresses are determined. The system is then in a position to know the level closest to the root in the broadcasting tree where these identified destinations have a common ancestor. Thus, instead of transmitting multiple copies of the message to individual destinations, the source processor sends only one message to this ancestor, along with the list of the corresponding destinations (these destinations are in a subtree rooted at this ancestor). At this ancestor, say at level i, g_{i+1} copies of the message are made, where g_{i+1} is the number of its child processors at level *i+1* being destinations or having descendants that are destinations. One copy of the message is distributed to each one of these children at level *i+1*, along with the corresponding (sub) list of destinations.

When a particular destination processor is reached, its address is removed from the destination list. The group of remaining processors is again scuttled around to determine common ancestors closest to the source. Each processor that is in receipt of a copy of the message now initiates the above steps recursively till all the destination processors on the list are exhausted. This clubbing technique may drastically reduce the bandwidth required of data channels. The effect is more drastic for channels closer to the root of the tree. Figure 3.1 shows the process of clubbing the messages meant for different processors in the same subtree for the general case of broadcasting; the notation *i/j* denotes the transmission of *i* messages to *j* destinations. As seen, the network traffic can be significantly reduced.

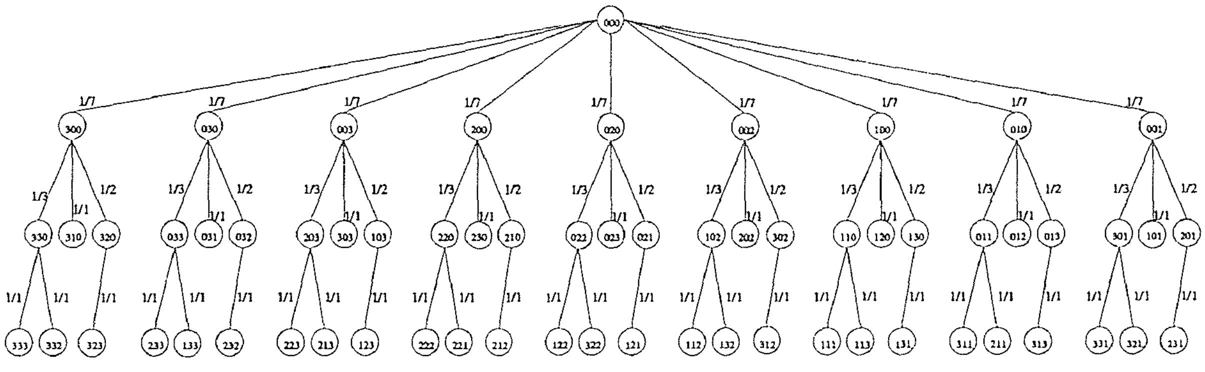

Figure 3.1: *The process of clubbing messages to reduce the traffic for broadcasting on the $GH_{3,4}$*

3.2.2 Multicast Algorithm

Before presenting the basic multicast algorithm, a few definitions are pertinent. Let us first reiterate that in the generalized hypercube $GH_{n,k}$ the total number of processors is $N=k^n$ and its diameter is n. The *depth*, D, of a node in a spanning tree is the number of radix-k digits in the node's address that differ from the source address, and in effect it is the minimum number of channels (hops) between the source and this node. The maximum depth corresponds to the leaf processors, which are at depth n (i.e., equal to the diameter of the generalized hypercube).

Given a node with displacement d in a spanning tree rooted at 0^n, the *leading zeros*, if present, in its address are found by the following procedure. Assuming that the most significant radix-k digit in the address has index 0, first find the digit with index $(d \bmod n)$. The leading zeros, if present, in the address are the maximal group of consecutive zeros just to the left of the latter digit, assuming a cyclic address. Leading zeros do not exist for the leaf nodes in the tree rooted at address 0^n; each node at any other level of this tree has children whose addresses differ from its own address in only one of its leading zero digits. For example, consider the processor with address *1010100* in the generalized hypercube $GH_{7,2}$, which has displacement $d = 0$ and depth $D=3$ in the tree rooted at 0^7. Starting with the most significant digit, corresponding to index 0, we go cyclically to its left to identify the two least significant digits in the address as the leading zeros. The details are shown below:

- The indicated digit position in the processor address *1010100*, for the spanning tree rooted at 0^7 in the generalized hypercube $GH_{7,2}$, corresponds to the displacement of that processor:

$$\overset{0-th\ digit}{1}\quad 010100$$

- The *leading zeros* of the address are:

$$10101 \quad \overbrace{00}^{2\,Leading\ Zeros}$$

Therefore, the number of child processors, M, for any non-leaf processor in the spanning tree created is given by

$$M \leq (k\text{-}1) \times (number\ of\ leading\ zeros)$$

Our multicast algorithm operates as follows. First, apply the inverse of the translation operation to any given destination address to determine the displacement, d, of the inversely translated destination processor in the spanning tree rooted at the given source s. This inverse translation of nodes is with respect to the source node s. The *inverse translation* of a node v with respect to node s is denoted by $t = T_s^{-1}(v)$, so that $t_i=(v_i - s_i)\ mod\ k$, for $0 \leq i \leq n\text{-}1$. The inverse translation is applied because the BST_s is obtained by translating all nodes in the BST_0'' by s.

Step 1: For the inverse-translated source processor (i.e., processor 0_n) modify its address digit with index $(d\ mod\ n)$ from the left to equal the corresponding digit in the inverse-translated destination. Translate the resulting processor with respect to s to obtain the node P_l. This is the first processor in the subtree enroute to the destination processor d_l at *depth l*. The message is then sent to this intermediate processor P_l for this destination.

Step 2: At any intermediate processor P_j, where $1 \leq j \leq (l\text{-}1)$, inverse-translate P_j and the destination address d_l with respect to the source address s. We reiterate that the source and destination addresses are contained in the message header. Identify the field of leading zeros in the inverse-translated P_j. In the corresponding field of the inverse-translated d_l, check for the first non-zero digit cyclically to the right of position *(d mod n)*. Modify the corresponding digit in the inverse-translated P_j to match this non-zero digit. Translate the result with respect to s. This is the next processor in the subtree enroute to the destination, d_l.

Step 3: Repeat the above step recursively till the current processor equals d_l.

Figure 3.2 demonstrates the multicast operation on the generalized hypercube $GH_{3,4}$. A label along a channel represents a message travelling from one processor to its neighboring processor enroute to the destination. The multicasting operation generates one message for every destination. For optimal performance, the technique of clubbing can be used (as described earlier). Figure 3.3 shows the same multicast operation with active clubbing of destination processors.

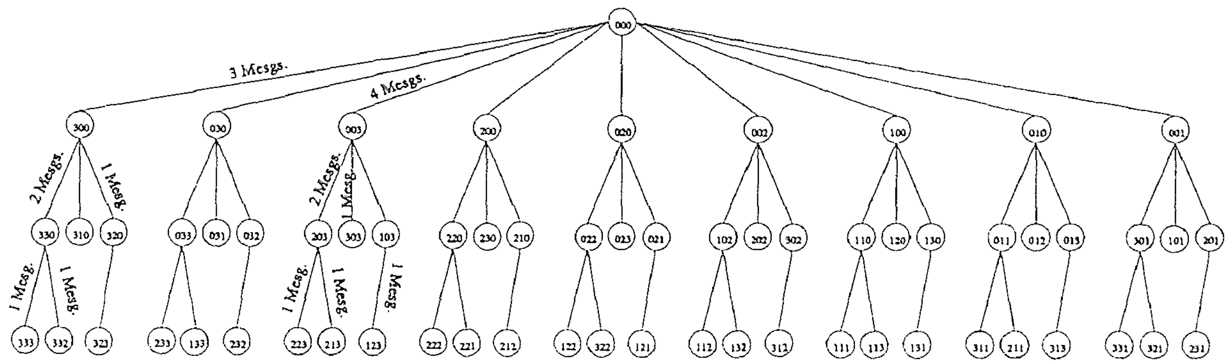

Figure 3.2: *Multicasting on the $GH_{3,4}$*

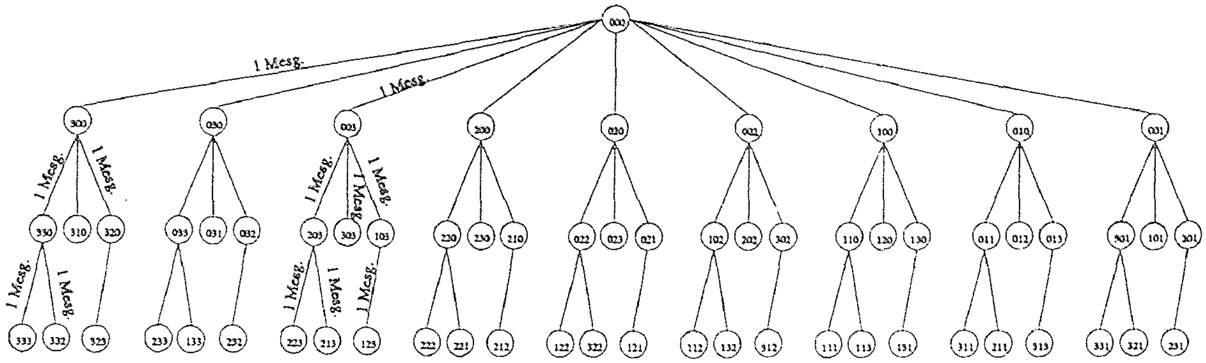

Figure 3.3: *The process of clubbing messages for multicasting on the* $GH_{3,4}$

CHAPTER 4

SIMULATIONS

4.1 Implementation

Simulation of the multicast and broadcast algorithms was carried out on sequential systems by generating the spanning trees/subgraphs as outlined earlier. The source code was written in C++. Despite the sequential simulation, the implementation here is described for a parallel system containing a generalized hypercube. The entire spanning tree/subgraph rooted at 0^n is created at static time (i.e., before the actual operations on the generalized hypercube commence). This has been implemented in the simulation by dynamically creating objects/array structures corresponding to each processor in the system. Each processor at static time creates the entire spanning tree and stores it in its local memory. The record of each node in the tree can be accessed in constant time in the local memory by using a simple hashing function involving the node's address. The processor in the generalized hypercube $GH_{n,k}$ with address $v=v_{n-1}{\dots}v_{i+1}v_iv_{i-1}{\dots}v_0$ corresponds to the index j in the array of node-records, where

$$j = R + v_{n-1} \times k^{n-1} + v_{n-2} \times k^{n-2} + \dots + v_1 \times k^1 + v_0 \times k^0$$

and R is the index for the source processor 0^n. For example, if the source processor has index 0 in the generalized hypercube $GH_{2,4}$, then processor with address 20 appears at index $j = 0 + 2 \times 4 + 0 = 8$. Each processor in this allocation has pointers to its child processors and also pointer(s) to its parent(s), as per the spanning tree/subgraph rooted at 0^n, also referred to as $BST_0{}^n/BSG_0{}^n$. Each processor has an input message buffer, namely inbox, for data arriving from its parent(s) and $n \times (k\text{-}1)$ (i.e. the number of its neighboring processors) output message buffers, namely outboxes, one for each of its children.

In the case of the one-to-all broadcast, when a processor s initiates a message, it identifies its child processors in the spanning tree BST_s by applying the translation operation with respect to s to the children of the node 0^n in the $BST_0{}^n$. The initiating processor then distributes the message to the appropriate inboxes of all its children. The propagation of messages continues till the leaf processors are reached.

In a variation of the one-to-all broadcast, called herein randomized many-to-all broadcast, a random number of processors may initiate one-to-all broadcasts in every cycle. This random number is determined in our simulation by the Poisson distribution that is often used to represent real-life traffic patterns. Thus, new messages may be initiated in any cycle of the simulation. In the case of the many-to-all broadcast, the spanning subgraph BSG_s is used for a source node s. Some of the processors with multiple paths to the root receive messages that are split across channels, as described in Chapter 3.

The multicast operation makes use of the spanning tree BST_s for the transmission of messages originating at node s. As in the case of randomized many-to-all broadcast, the source processors are determined in each cycle by using the Poisson distribution, and for each of these source processors random destinations are determined. Arbitrarily, the number of destinations has been chosen as $N/32$, $N/16$, $N/8$ and $N/4$, where N is the total number of processors in the system. A starting destination address, r_1, and a stride, r_2, are chosen each time using random number generators to determine the destination addresses $((r_1+i \cdot r_2) \ mod \ N)$, where $0 \leq r_1, \ r_2 \leq N\text{-}1$. When a message is initiated by a node s, copies of the same are made into the outboxes corresponding to the appropriate next level (in the BST_s) children for the multicast.

Each simulation was carried out *20* times and the results were averaged to give a clear picture of the communication bottlenecks arising as a result of the increased, random traffic patterns.

4.2 Simulation Results

The randomized multicast and broadcast operations were simulated using the Poisson distribution, where in any given cycle a processor may become the initiator of a message if and only if its probability is above a predetermined threshold value. The Poisson distribution probability for k successes in the specified time interval is given by

$$P[k] = \frac{\alpha^k}{k!} e^{-\alpha}$$

where α is the average number of initiations in the specified time interval of 20 cycles. k is a random positive integer generated each time by using the system clock. The value of $P[k]$ is maximum at $k = \alpha$ *and* $k = \alpha - 1$, if α is a positive integer and $\alpha > 1$.

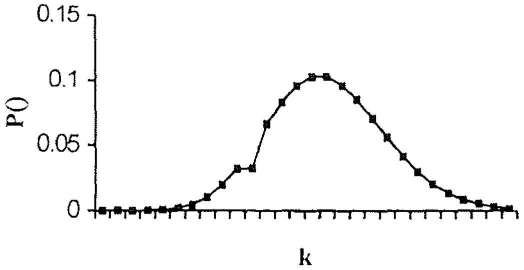

Figure 4.1: *The Poisson distribution curve obtained for a random processor.*

Figure 4.1 displays the Poisson distribution curve obtained for the processors initiating a communications operation in a given cycle, averaged over 20 cycles. The value of α has been chosen as 15 to have an increased probability of a processor initiating a message in any given cycle. All processors having in a given cycle a probability greater than the threshold value, set as *2/3 × (Maximum Probability)*, are considered message initiators. For each processor that happens to be an initiator, a translated spanning tree/subgraph is created dynamically in a distributed manner; the message is first distributed to all the source's children in the case of broadcasting and to the appropriate set of children in the case of multicasting.

Table 4.1: *Results of randomized broadcasting*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	71	55	44	38	33	30	22	3129
$GH_{2,16}$	256	57	44	36	31	28	25	22	8157
$GH_{3,8}$	512	62	48	40	35	31	29	23	18168
$GH_{6,3}$	729	71	56	46	42	36	33	26	34185
$GH_{4,7}$	2401	78	60	50	43	38	35	24	99257
$GH_{5,5}$	3125	74	57	48	42	37	34	25	141321
$GH_{4,8}$	4096	69	53	45	38	35	32	24	164406
$GH_{4,10}$	10000	77	60	50	43	38	35	24	425234
$GH_{4,11}$	14641	95	72	60	51	45	41	24	827566
$GH_{3,25}$	15625	96	73	60	51	44	40	23	926318

Simulation results of many-to-all randomized broadcasting are presented in Table 4.1. '*No. Of Mesgs.*' in the table represents the total number of one-to-one source-to-destination messages. The results show that randomized multicasting may result in a large number of contentions if the basic algorithm from [3] is used repeatedly. Also, the buffer size has a very significant effect on the total time.

Tables 4.2 through 4.5 present results of randomized multicasting, where the number of destinations for each multicast is always *1/4-th*, *1/8-th*, *1/16-th* and *1/32-nd*, respectively, of the total number of processors; the destination addresses are chosen randomly. The results show that the larger the system, the larger the message buffers we need to have for better performance. The results also show that the basic algorithm for broadcasting proposed in [3] may result in large numbers of channel contentions under realistic conditions. For this reason, we present an adaptive routing algorithm for multicasting in the next chapter, as well as respective performance results.

Table 4.2: *Results of randomized multicasting, with 1/4-th of the processors being selected randomly as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	63	48	39	33	29	26	22	2389
$GH_{2,16}$	256	99	75	60	51	44	39	22	8036
$GH_{3,8}$	512	137	104	86	74	67	60	23	17752
$GH_{6,3}$	729	205	156	127	107	94	83	26	26427
$GH_{4,7}$	2401	467	354	286	240	208	184	24	100069
$GH_{5,5}$	3125	576	436	351	295	255	224	25	125050
$GH_{4,8}$	4096	753	568	456	382	329	290	24	155637
$GH_{4,10}$	10000	1720	1292	1036	864	742	650	24	367486
$GH_{4,11}$	14641	1744	1309	1048	874	750	657	24	726565
$GH_{3,25}$	15625	1445	1089	875	733	631	555	23	826213

As the size of the generalized hypercube increases, the amount of information being exchanged among the processors in the system grows alarmingly.

Table 4.3: *Results of randomized multicasting, with 1/8-th of the processors being selected randomly as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	34	27	24	24	23	23	22	1342
$GH_{2,16}$	256	57	43	36	32	30	28	22	4628
$GH_{3,8}$	512	93	71	58	50	43	39	23	9706
$GH_{6,3}$	729	108	83	69	59	52	48	26	14800
$GH_{4,7}$	2401	249	189	153	129	112	100	24	56477
$GH_{5,5}$	3125	310	234	190	161	140	124	25	69733
$GH_{4,8}$	4096	380	288	233	196	169	150	24	85583
$GH_{4,10}$	10000	868	653	524	438	377	331	24	212572
$GH_{4,11}$	14641	1115	840	775	565	486	427	24	414020
$GH_{3,25}$	15625	742	562	454	382	330	292	23	458154

With a practical limit on the buffer size, it was noticed that generalized hypercube systems $GH_{n,k}$ with a bigger value for k and a smaller value for n seemed to have a lesser number of contentions than systems with almost the same number of processors having a larger value for n and a smaller value for k.

Table 4.4: *Results of randomized multicasting, with 1/16-th of the processors being selected randomly as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	24	23	23	22	22	22	22	705
$GH_{2,16}$	256	33	29	27	25	24	22	22	2406
$GH_{3,8}$	512	51	40	36	33	31	30	23	5066
$GH_{6,3}$	729	60	48	41	37	34	33	26	7764
$GH_{4,7}$	2401	128	99	82	71	62	56	24	29576
$GH_{5,5}$	3125	161	124	102	87	76	69	25	36172
$GH_{4,8}$	4096	202	157	130	112	99	89	24	44664
$GH_{4,10}$	10000	437	330	265	223	192	169	24	118679
$GH_{4,11}$	14641	648	490	395	331	286	252	24	224242
$GH_{3,25}$	15625	384	293	239	203	177	157	23	243240

Table 4.5: *Results of randomized multicasting, with 1/32-nd of the processors being selected randomly as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	23	22	22	22	22	22	22	360
$GH_{2,16}$	256	26	24	23	23	23	23	22	1234
$GH_{3,8}$	512	32	29	27	26	25	25	23	2600
$GH_{6,3}$	729	39	33	31	30	30	29	26	3891
$GH_{4,7}$	2401	70	56	47	42	38	35	24	15232
$GH_{5,5}$	3125	89	70	59	51	46	42	25	18555
$GH_{4,8}$	4096	112	89	75	66	60	55	24	22898
$GH_{4,10}$	10000	222	169	137	115	100	89	24	62061
$GH_{4,11}$	14641	330	251	204	172	150	133	24	115614
$GH_{3,25}$	15625	203	157	130	112	89	78	23	127276

To demonstrate the need for a better multicast algorithm, we show in Tables 4.6 through 4.8 results where the same destinations are always chosen for all multicasts.

Table 4.6: *Results of randomized multicasting, with 1/8-th of the same processors being selected as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	41	38	33	28	24	24	22	1360
$GH_{2,16}$	256	97	74	60	51	44	39	22	5024
$GH_{3,8}$	512	154	116	94	79	68	60	23	10752
$GH_{6,3}$	729	205	155	124	105	91	81	26	18325
$GH_{4,7}$	2401	349	265	215	182	158	140	24	63300
$GH_{5,5}$	3125	327	249	201	170	148	132	25	75660
$GH_{4,8}$	4096	562	424	342	286	247	217	24	93184
$GH_{4,10}$	10000	601	454	366	307	265	233	24	258750
$GH_{4,11}$	14641	901	680	547	458	395	348	24	477630
$GH_{3,25}$	15625	1311	985	789	659	566	496	23	521451

Table 4.7: *Results of randomized multicasting, with 1/16-th of the same processors being selected as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	36	29	25	24	24	24	22	731
$GH_{2,16}$	256	88	67	54	46	40	35	22	2512
$GH_{3,8}$	512	118	93	73	61	53	47	24	5376
$GH_{6,3}$	729	150	114	92	78	68	60	26	9485
$GH_{4,7}$	2401	191	147	120	103	90	81	24	31650
$GH_{5,5}$	3125	209	159	130	110	96	85	25	37830
$GH_{4,8}$	4096	276	209	170	143	124	110	24	46592
$GH_{4,10}$	10000	367	277	223	188	162	143	24	129375
$GH_{4,11}$	14641	564	426	343	288	248	219	24	238815
$GH_{3,25}$	15625	659	496	398	333	287	252	23	260592

Table 4.8: *Results of randomized multicasting, with 1/32-nd of the same processors being selected as destinations for each transfer*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	22	22	22	22	22	22	22	390
$GH_{2,16}$	256	32	31	30	27	25	23	22	1224
$GH_{3,8}$	512	52	51	50	44	39	35	23	2688
$GH_{6,3}$	729	117	90	73	62	55	49	26	4820
$GH_{4,7}$	2401	159	117	99	84	73	65	24	15825
$GH_{5,5}$	3125	172	132	107	91	79	70	25	18818
$GH_{4,8}$	4096	172	133	110	93	81	72	24	23296
$GH_{4,10}$	10000	246	186	151	129	114	102	24	64584
$GH_{4,11}$	14641	321	244	198	167	145	129	24	119277
$GH_{3,25}$	15625	344	262	212	179	156	138	23	130296

CHAPTER 5

ADAPTIVE ROUTING

We have also simulated an adaptive routing algorithm for randomized multicasting. In the case of adaptive routing in parallel systems, some messages do not follow the shortest paths to their destinations, in order to avoid channel contentions [14]. In our adaptive algorithm, each sending processor compares the numbers of messages in all its outboxes whenever deterministic routing may result in channel contention (where a message will have to wait in an outbox for a future transfer). If it finds an empty outbox corresponding to a neighbor that is itself a neighbor to its intended child for that message, it sends the message to the former instead of sending it to the latter.

Table 5.1: *Results of randomized multicasting, with 1/4-th of the processors being selected randomly as destinations for each transfer. Adaptive routing.*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	63	48	39	33	29	26	22	2389
$GH_{2,16}$	256	99	75	60	51	44	39	22	3036
$GH_{3,8}$	512	137	104	86	74	67	60	24	17752
$GH_{6,3}$	729	204	156	126	107	93	82	29	26427
$GH_{4,7}$	2401	467	354	286	240	208	183	26	100069
$GH_{5,5}$	3125	576	436	251	295	25	224	26	125050
$GH_{4,8}$	4096	753	568	456	382	329	289	42	155637
$GH_{4,10}$	10000	1720	1292	1035	864	742	650	39	367486
$GH_{4,11}$	14641	1744	1309	1048	874	750	657	26	726565
$GH_{3,25}$	15625	1445	1089	875	733	631	555	24	826213

Table 5.1 shows results of such simulations, where the number of random destinations for each multicast is *1/4-th* of the total number of processors (as earlier for deterministic routing). Comparing the results with earlier results for deterministic routing,

we observe that adaptive routing reduces channel contentions and this often results in reduced execution times.

Table 5.2: *Results of randomized multicasting, with 1/8-th of the same processors being selected as destinations for each transfer. Adaptive routing.*

GH$_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
GH$_{2,8}$	64	68	54	46	44	41	38	22	1360
GH$_{2,16}$	256	88	68	54	46	41	37	22	5024
GH$_{3,8}$	512	118	89	71	62	56	38	23	10752
GH$_{6,3}$	729	173	133	105	89	76	67	26	18325
GH$_{4,7}$	2401	305	234	188	162	141	127	25	63300
GH$_{5,5}$	3125	324	244	194	161	140	124	27	75660
GH$_{4,8}$	4096	562	423	339	283	244	215	25	93184
GH$_{4,10}$	10000	572	433	342	288	250	220	24	258750
GH$_{4,11}$	14641	850	642	510	428	370	326	24	477630
GH$_{3,25}$	15625	1311	985	789	659	566	496	25	521451

To demonstrate even more dramatic improvements due to adaptive routing, we present in Tables 5.2 through 5.4 results of randomized multicasting where the destinations are identical for all multicasts; the number of destinations is *1/8-th*, *1/16-th* and *1/32-nd*, respectively, of the total number of processors.

Table 5.3: *Results of randomized multicasting, with 1/16-th of the same processors being selected as destinations for each transfer. Adaptive routing.*

GH$_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
GH$_{2,8}$	64	35	30	25	24	24	23	22	731
GH$_{2,16}$	256	73	54	46	39	34	32	22	2512
GH$_{3,8}$	512	76	63	49	43	39	36	24	5376
GH$_{6,3}$	729	85	70	56	49	45	40	26	9485
GH$_{4,7}$	2401	124	105	85	76	69	64	25	31650
GH$_{5,5}$	3125	174	134	107	91	81	74	26	37830
GH$_{4,8}$	4096	274	209	168	142	123	109	24	46592
GH$_{4,10}$	10000	320	245	196	166	144	127	24	129375
GH$_{4,11}$	14641	471	365	286	238	206	182	24	238815
GH$_{3,25}$	15625	659	496	398	333	287	252	24	260592

Table 5.4: *Results of randomized multicasting, with 1/32-nd of the same processors being selected as destinations for each transfer. Adaptive routing.*

$GH_{n,k}$	No. Of Procs.	Execution Time (cycles)							No. Of Mesgs.
		Buffer Size (messages)							
		3	4	5	6	7	8	∞	
$GH_{2,8}$	64	27	25	24	23	23	23	22	390
$GH_{2,16}$	256	39	31	27	27	27	27	22	1224
$GH_{3,8}$	512	51	42	37	33	31	30	23	2688
$GH_{6,3}$	729	57	48	42	38	37	32	26	4820
$GH_{4,7}$	2401	90	71	61	54	49	46	24	15825
$GH_{5,5}$	3125	104	81	69	60	54	49	26	18818
$GH_{4,8}$	4096	146	111	91	78	68	61	24	23296
$GH_{4,10}$	10000	191	146	122	106	95	86	24	64584
$GH_{4,11}$	14641	238	184	153	132	116	104	24	119277
$GH_{3,25}$	15625	333	251	203	170	147	129	24	130296

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

We presented here results obtained by evaluating the communications capabilities of the generalized hypercube interconnection network. Recent and expected advances in electronic and hybrid wiring technologies will soon make the generalized hypercube a practical interconnection network for massively-parallel processing. The algorithm presented in [3] for broadcasting was tested under realistic conditions. The results show that this algorithm may not often produce good results. For this reason, an adaptive routing algorithm was proposed and tested. In addition, algorithms for multicasting were proposed and evaluated. The results prove the versatility of the generalized hypercube under heavy communications traffic.

More specifically, it was noticed that the execution time and the contentions on the channels were directly related to the intermediate processor's input buffer size. As the buffer size increases, the number of cycles required to complete the total task of communication reduces noticeably.

With adaptive routing, assuming that a message can be routed out of its normal route only a finite number of times, it was noticed that the effects were encouraging in a good percentage of the cases. It was seen that adaptive routing worked better than deterministic routing for systems with a large number of dimensions compared to equivalent systems with a smaller number of dimensions and more nodes per dimension. On the $GH_{5,5}$, which is the system that has been simulated with the highest number of dimensions, the use of adaptive routing improved the performance by a good margin. In smaller systems, like the $GH_{2,8}$ and $GH_{3,8}$, the use of adaptive routing actually had a small negative effect and the number of execution steps slightly increased.

In the general case (deterministic and adaptive routing), systems with a lower number of dimensions performed better than comparable systems (with the same number of processors) with more dimensions. This logically follows from the fact that the bisection bandwidth (given as $k^{n+1}/4$) in the case of the former is greater and hence tends to result in good communications behavior.

The entire simulation assumes the store-and-forward switching model. An alternate approach using wormhole routing would definitely produce better results, but in terms of implementation it would also be equally demanding. Redesigning the algorithm so as to support wormhole routing would be an interesting and challenging extension to this research work.

Additional research work should test the communications capabilities of the generalized hypercube for communication patterns derived from widely used application benchmarks.

Another improvement would be to assume realistic communication and message switching times.

REFERENCES

1. S G. Ziavras, "RH: A Versatile Family of Reduced Hypercube Interconnection Networks," *IEEE Trans. Paral. Distr. Systems*, Vol. 5, No. 11, Nov. 1994, pp. 1210-1220.

2. J. K. Antonio, L. Lin, and R. C. Metzger, "Complexity of Intensive Communications on Balanced Generalized Hypercubes," *Intern. Paral. Proces. Symp., 1993, pp. 387-394.*

3. P. Fragopoulou, S. G. Akl, and H. Meijer, "Optimal Communication Primitives on the Generalized Hypercube Network," *Journ. Paral. Distr. Comput. 32, 1996, pp. 173-187.*

4. W. Dally, "Network and Processor Architecture for Message-Driven Computers," in: *VLSI and Parallel Computation,* R. Suaya and G. Birtwistle (Eds.), *Morg. Kauf. Publ., 1990, pp. 140-222.*

5. L.D. Wittie, "Communication Structures for Large Networks of Multicomputers," *IEEE Trans. Comput. C-30(4), 1981.*

6. S.G. Ziavras, "Generalized Reduced Hypercube Interconnection Networks for Massively Parallel Computers," in: *Networks for Parallel Computations*, *Amer. Math. Soc.*, D.F. Hsu, A. Rosenberg, and D. Sotteau (Eds.), *1995, pp. 307-325.*

7. S.G. Ziavras and A. Mukherjee, "Data Broadcasting and Reduction, Prefix Computation, and Sorting on Reduced Hypercube Parallel Computers," *Parallel Computing 22, 1996, pp. 595-606.*

8. S.G. Ziavras, "On the Problem of Expanding Hypercube-Based Systems," *Journ. Paral. Distr. Comp. 16(1), 1992, pp. 41-53.*

9. S.G. Ziavras, "Scalable Multifolded Hypercubes for Versatile Parallel Computers," *Paral. Proc. Letts. 5(2), 1995, pp. 241-250.*

10. L.N. Bhuyan and D.P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Trans. Comput.33 (4), 1984, pp. 323-333.*

11. S.G. Ziavras, "Investigation of Various Mesh Architectures with Broadcast Buses for High-Performance Computing," *VLSI Design, Special Issue High Perf. Bus-Based Arch.*, R. Lin and S. Olariu (Eds.), accepted for publication, 1997.

12. S.G. Ziavras, H. Grebel, and A.T. Chronopoulos, "A Low-Complexity Parallel System for Gracious, Scalable Performance. Case Study for Near PetaFLOPS Computing," *6th Symp. Frontiers Massively Paral. Comput, Special Session New Millennium Computing Point Designs, 1996, pp. 363-370.*

13. S.G. Ziavras, H. Grebel, and A.T. Chronopoulos, "A Scalable/Feasible Parallel Computer Implementing Electronic and Optical Interconnections for 156 TeraOPS Minimum Performance," *PetaFLOPS Arch. Worksh.,1996, pp. 179-209.*

14. P.T. Gaughan and S. Yalamanchili, "Adaptive Routing Protocols for Hypercube Interconnection Networks," *IEEE Computer, May 1993, pp. 12-23.*

15. Q. Wang, and S.G. Ziavras, "Network Embedding Techniques for a New Class of Feasible Parallel Architectures Capable of Very High Performance," *Parallel Computing,* submitted for publication, Sept. 1997.

16. T. Szymanski, "A Fiber Optic Hypermesh for SIMD/MIMD Machines," *Supercomputing Conf., Nov. 1990, pp. 103-110.*

17. S. Lennart Johnsson and Ching-Tien Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans Comput.38(9), 1989, pp. 1249-1267.*

18. Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw Hill, New York, 1993

19. CRAY T3D System Architecture, Overview Manual, Cray Research, 1992.

20. S. G. Akl, *Parallel Computation – Model and Methods*, Prentice Hall, New Jersey, 1997.