

# Evaluating the communications capabilities of the generalized hypercube interconnection network

SOTIRIOS G. ZIAVRAS\* AND SANJAY KRISHNAMURTHY

*Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA*

---

## SUMMARY

This paper 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 on 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 by Fragopoulou *et al.* (1996). 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. Copyright © 1999 John Wiley & Sons, Ltd.

## 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 led to the development of massively parallel computers with hundreds or thousands of processors. 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, such as 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.

\*Correspondence to: Professor Sotirios G. Ziavras, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA.  
Contract/grant sponsor: NSF/DARPA/NASA; Contract/grant number: ASC-9634775

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 computing 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 a DSM framework[1–4].

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 T3E). The main obstacles with these approaches are the resulting large diameter and average interprocessor distance, and the small bisection bandwidth. To improve the topological properties of bounded-degree networks, switches may be incorporated in the design[5]. However, such approaches are not appropriate for very high-performance computing. The generalized hypercube network[6] 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[7] 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[1,3,8], 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[4,9,10]. An architecture capable of near-PetaFLOPS performance by the year 2005 was designed and analyzed, in terms of feasibility and performance, under a New Millennium Computing Point Design grant awarded jointly to our group by NSF, DARPA and NASA[4,10]. This architecture employs free-space optics for the efficient 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[11] for wires in each fully connected subsystem.

This paper investigates the implementation of important communications primitives, like broadcasting and multicasting, on generalized hypercubes. One-to-all (or many-to-all) 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 algorithms in this paper that solve these problems 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 input and output 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).

The organization of this paper is as follows. Section 2 briefly presents the generalized hypercube interconnection network. Section 3 presents a case study for a system designed around the generalized hypercube. Section 4 summarizes algorithms presented in [12] for the implementation of one-to-all and all-to-all broadcasting on the generalized hypercube, and emphasizes realistic scenarios where these algorithms could result in

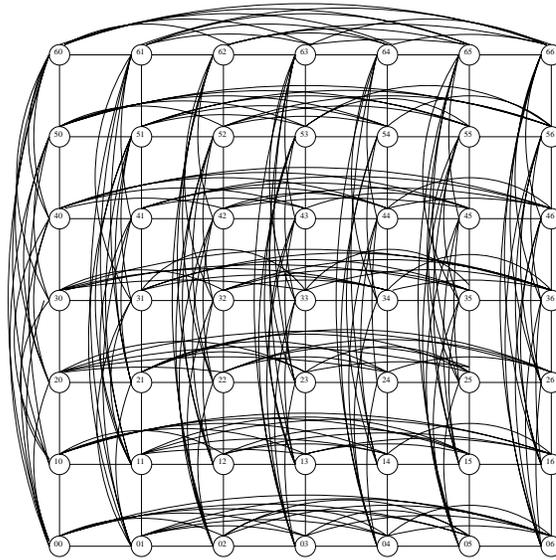


Figure 1. The  $GH_{2,7}$

significant communications bottlenecks. These are very common operations in parallel algorithms [13,14]. In Section 5, further investigation of these algorithms is made and algorithms for another communication primitive, namely multicasting, are proposed. Section 6 presents simulation results for realistic communication patterns and message generations. Section 7 is devoted to the improvement of the latter algorithms through adaptive routing. Relevant simulation results are also included. Finally, we present our conclusions in Section 8.

## 2. 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 [6]. The terms processor and node will be used interchangeably from now on. 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. its 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 shows the 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 much larger 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 implies that the generalized hypercube results in outstanding performance for large systems with thousands of processors and heavy inter-processor communication traffic.

Table 1. Comparison of interconnection networks, assuming full-duplex bidirectional data channels

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

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 [4,10], 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 compares the numbers of channels in the  $k$ -ary  $n$ -cube (i.e. the  $n$ -dimensional torus) and the generalized hypercube  $GH_{n,k}$  with the same number of processors (i.e.  $k^n$ ). For example, assuming bidirectional 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-dimensional  $GH_{3,22}$  with a diameter of 3.

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 paper, we first make use of the spanning graphs for the generalized hypercube proposed in [12] for one-to-all broadcasting, for performance assessment under realistic communications traffic. Relevant work from [12] is summarized in Section 4. Further work on multicasting is also presented later.

### 3. A CASE STUDY FOR VERY HIGH-PERFORMANCE COMPUTING

An architecture capable of near-PetaFLOPS performance by the year 2005 was designed and analyzed, in terms of feasibility and performance, under a New Millennium Computing Point Design grant awarded jointly to our group by NSF, DARPA and NASA[4,10]. Our architecture encompasses a 2-D interconnection network that employs electrical and optical technologies. Subsection 3.1 presents the structure of the 1-D building block (BB) and issues related to its implementation. The 2-D complete system is constructed by repeating this 1-D BB in two dimensions, and also incorporating additional glue logic. Subsection 3.2 describes the 2-D complete structure.

#### 3.1. 1-D building block (BB)

Our basic design takes advantage of free-space optical technologies to produce a 1-D fully connected, scalable BB capable of implementing bit-parallel communications channels. The first objective is to produce a low-cost, powerful, free-space, reliable, point-to-point communications system of low packaging complexity that incorporates guided-wave concepts. Free-space interconnects possess an energy–bandwidth product which is larger than their electronic counterpart.

---

The BB is a 1-D array of 8-PE cards attached to an inexpensive clear plastic/glass bar that provides alignment for optical transmissions. Each card carries the entire processing and memory power of eight processors fully interconnected via an electronic crossbar. This approach was chosen because of the high efficiency of small electronic crossbars. Each card is interfaced with optical transmitter/receiver modules and attached prismatic elements for inter-card data transfers, and the destination address for a data transfer is decoded to determine the prismatic element and associated modules to be used for the appropriate path.

In an optical cycle, 32 bits of information can be sent in parallel from one card to another via a card-to-card (i.e. point-to-point) color-coded interconnect, using 32 distinct colors (i.e. by the WDM technique); these 32 colors are the same for all of the cards. Actually, each inter-card channel is 128 bits wide because of the chosen format for messages, and therefore 128-bit information is transmitted each time (i.e. during a PE's cycle) by utilizing four (i.e. 128/32) optical cycles (i.e. by the TDM technique). All eight PEs on a card share the same lasers and receivers for inter-card transmissions (i.e. by the TDM technique). To summarize, the WDM and TDM techniques are used as follows:

- WDM with 32 wavelengths for bit-parallel transmissions involving 32 bits
- TDM with four communication cycles to implement 128-bit transmissions, where each of these four cycles is of the aforementioned WDM type implementing 32-bit transmissions
- TDM with eight communication cycles, so that the eight PEs on a card can share the optical transmit/receive modules assigned for the exchange of information with another 8-PE card.

The chosen prismatic element for a data transfer determines a specific optical path via the set of reflectors used between the transmitting and receiving cards. Since any two cards communicate via dedicated prismatic elements, multi-access node communication is available. Common colors from different cards are detected by different detector arrays at different locations on the card's interface. Separation among the messages sent to a given card from other cards is made by separating the fields of view, and therefore activating different detectors on the receiving card. The receiver demultiplexes the information and sends it to the destined PE on the given card.

### 3.2. Complete 2-D system

Extension of the 1-D fully connected BB into a 2-D configuration is now in order. In addition to interfacing a horizontal clear plastic bar (used for interconnects in the first dimension), each 8-PE card now also belongs to a similar 1-D structure in the second dimension. Therefore, each card also interfaces a vertical plastic bar. The clear plastic columns are patterned with small metallic reflectors and prismatic interfaces, as for the horizontal bars. All in all, the system may be viewed as a 2-D array, with rows and columns containing fully interconnected PEs. It behaves like a 2-D generalized hypercube; each node of the generalized hypercube contains eight fully interconnected PEs (they are fully interconnected via an on-card electronic crossbar network).

#### 4. COMMUNICATIONS OPERATIONS ON THE GENERALIZED HYPERCUBE

One technique often used to implement communications operations on interconnection networks is to embed spanning trees specially designed for each network. [12] presents a novel way of implementing the one-to-all and all-to-all broadcast communications primitives 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. Taking advantage of the fact that the generalized hypercube is a symmetric network, the authors create at static time a spanning tree 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.

Let us summarize the procedure for creating a spanning tree rooted at the (source) node  $s = 0^n$  in the  $GH_{n,k}$ ;  $0^n$  denotes a string of  $n$  consecutive 0s. 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  $v = v_{n-1} \dots v_{i+1} v_i v_{i-1} \dots v_0$  in the  $GH_{n,k}$  produces the node  $R(v)$  given by

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

where

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

For example, if  $v = 342$  and  $u = 023$ , in the generalized hypercube  $GH_{3,5}$ , 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 a given distance  $i$  from the node  $0^n$  in the  $GH_{n,k}$ , where  $1 \leq i \leq n$ , are collections of necklaces. More definitions are pertinent [12]. The *binary correspondent* of a node is the binary number derived by substituting a 1 for each non-zero digit in its  $n$ -digit 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 nodes in its necklace. An *unfolded necklace* contains  $n \times (k-1)$  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 shows the unfolded necklaces of the generalized hypercubes  $GH_{3,3}$  and  $GH_{3,4}$ .

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 *parent* 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,

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

Table 2. The unfolded necklaces of the  $GH_{3,3}$  and  $GH_{3,4}$

The necklaces of $GH_{3,3}$		The necklaces of $GH_{3,4}$	
Distance	Nodes	Distance	Nodes
$d = 0$	[000, 000, 000, 000, 000, 000]	$d = 0$	[000, 000, 000, 000, 000, 000, 000, 000, 000, 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, 212, 121, 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]

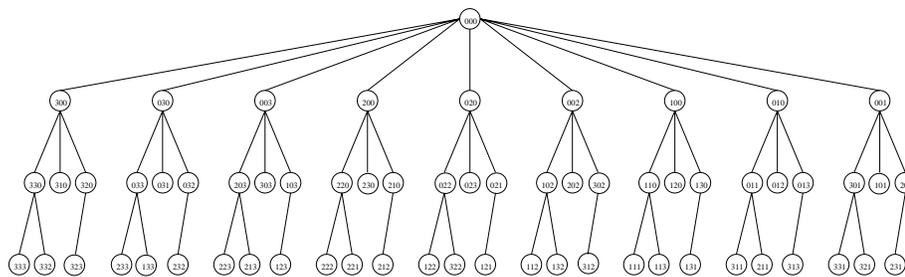


Figure 2. The  $BST_{0^3}$  of the  $GH_{3,4}$

The spanning tree rooted at  $0^3$  of the  $GH_{3,4}$  is shown in Figure 2. 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 3 shows the  $BST_{0^3}$  and  $BSG_{0^3}$  of the  $GH_{3,3}$ . The  $BST_{0^3}$  and  $BSG_{0^3}$  are identical for the  $GH_{3,4}$ . 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 communications cycles.

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; this may result in many broadcasts being generated at different times and being simultaneously present. Such an investigation is carried out in this paper. We also

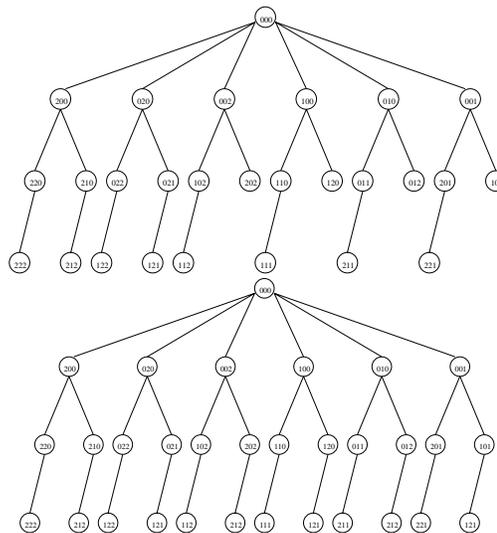


Figure 3. The  $BST_{03}$  and  $BSG_{03}$ , respectively, of the  $GH_{3,3}$

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. Incidentally, one-to-all broadcasting can be viewed as a special case of multicasting with a single source, where all processors are destinations.

## 5. INVESTIGATION OF COMMUNICATIONS PRIMITIVES

The one-to-all and all-to-all broadcast techniques in [12] 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 randomized multicasting with many sources, where a random number of processors initiate a multicast operation, 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 any 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 store these messages. The buffer size cannot be infinite in practice, and hence the time taken by the communications operation to complete also depends on the buffer size. The effect of the buffer size on the total communication time is also studied in this paper, through simulation. We point out in the rest of this section potential message contention problems for the existing communications algorithms. We also present algorithms for the implementation of randomized multicasting on generalized hypercubes. Simulation results for all these algorithms are presented in the next section.

---

### 5.1. Randomized many-to-all broadcast

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 [12] 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 communications 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 cases that result in such message contentions.

In each cycle of our simulations, every processor calculates randomly the probability of initiating a message. A threshold value of  $2/3 \times (\text{Maximum Probability})$  was set for the Poisson distribution, and all processors which have 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 contentions.

### 5.2. Randomized 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. with one source processor, and all other processors are destinations) and all-to-all broadcasting (i.e. every processor broadcasts a message to all other processors in the system).

The spanning trees/subgraphs created in [12] 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.

#### 5.2.1. Brute-force clubbing algorithm

Given a group of destinations for multicasting from a single source, all destinations having the same displacement (as defined in [12] and Section 4) are clubbed together as they all belong to the same subtree. 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

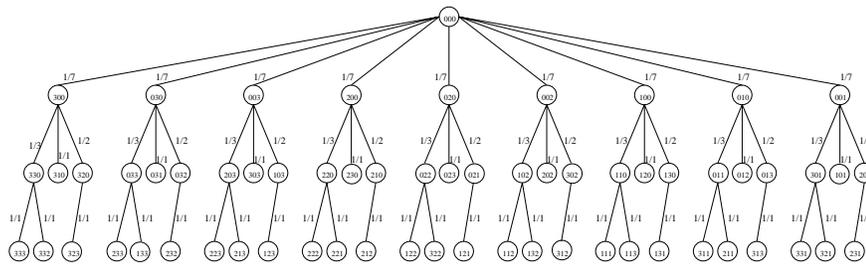


Figure 4. The process of clubbing messages to reduce the traffic for broadcasting on the  $GH_{3,4}$

multiple copies of the message to individual destinations, the source processor sends only one message to their 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 which 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 are again scuttled around to determine common ancestors closest to the source. Each processor which 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 4 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 for  $j$  destinations. As seen, the network traffic can be significantly reduced.

### 5.2.2. Multicast algorithm

Before we propose the basic multicast algorithm, a few definitions are pertinent. Let us first state 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 the 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 the 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  $0^n$ ; 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  $d = 0$  in the necklace  $[1010100, 0101010, 0010101, 1001010, 0100101, 1010010, 0101001]$  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:

$$\begin{array}{c} \text{0th digit} \\ \underbrace{\phantom{1}} \\ 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \end{array}$$

- The *leading zeros* in the address are:

$$1 \quad 0 \quad 1 \quad 0 \quad 1 \quad \underbrace{\phantom{00}}_{\text{2 leading zeros}}$$

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

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

Our multicast algorithm operates as follows. First, apply the inverse of the translation operation to each destination address to determine the displacement,  $d$ , of the inversely translated destination 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 given by  $t = T_s^{-1}(v)$ , so that  $t_i = (v_i - s_i) \bmod k$ , for  $0 \leq i \leq n - 1$ . The inverse translation is applied because the  $BST_s$  is obtained by translating all nodes in the  $BST_{0^n}$  by  $s$ .

*Step 1:* For the inverse-translated source processor (i.e. processor  $0^n$ ), modify its address digit with index  $(d \bmod n)$  from the left to equal the corresponding digit in the inverse-translated destination. Translate the resulting processor address with respect to  $s$  to obtain the node  $P_1$ . 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  $P_1$  for this destination.

*Step 2:* At any intermediate processor  $P_j$ , where  $1 \leq j \leq (l - 1)$ , inverse-translate  $P_j$  and the destination address  $d_l$  with respect to the source address  $s$ . We state 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  $(d \bmod n)$ . 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 5 demonstrates the multicast operation on the generalized hypercube  $GH_{3,4}$  assuming that the source is 000; the destinations are represented by shaded nodes. The multicast operation generates one message for every destination. For better performance, the technique of clubbing could be used (as described earlier). Figure 6 shows the same multicast operation with the clubbing of messages.

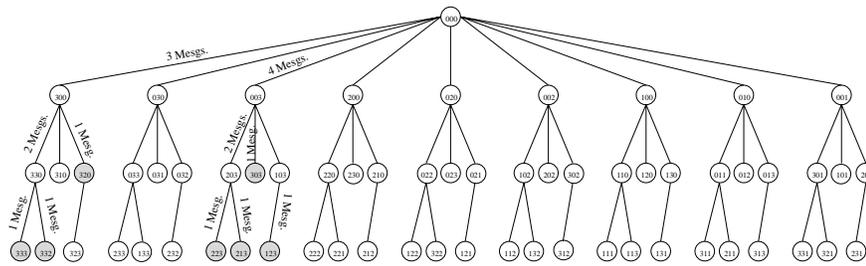


Figure 5. Multicasting on the  $GH_{3,4}$

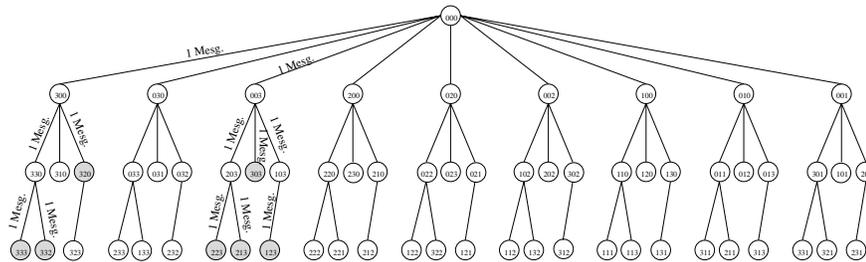


Figure 6. The process of clubbing messages for multicasting on the  $GH_{3,4}$

## 6. SIMULATION

### 6.1. 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 parallel systems 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 object/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 with address  $v = v_{n-1} \dots v_{i+1} v_i v_{i-1} \dots v_0$  in the generalized hypercube  $GH_{n,k}$ , with  $0 \leq v_i \leq (k-1)$  for all  $0 \leq i \leq (n-1)$ , 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 of the source processor  $0^n$ . For example, if the source processor has index 0 in the generalized hypercube  $GH_{2,4}$ , then the 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  $BST_{0^n}/BSG_{0^n}$  rooted at  $0^n$ . Each processor has an input message buffer, namely inbox, for data arriving from its parent(s) and  $n \times (k-1)$  (that is, 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, each processor  $s$  initiating a message 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 identified children. The propagation of messages continues till the leaf processors are reached. Each intermediate processor applies the inverse of the translation operation with respect to  $s$  to its own address (this operation subtracts  $s$  from the node's address) to find a new node address; the children of the intermediate node in the  $BST_s$  are then determined by applying the translation operation with respect to  $s$  to the children of the latter node in the  $BST_{0^n}$ .

In a variation of the one-to-all broadcast, called herein the 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 realistic traffic patterns; details follow in the next subsection. 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 Section 4.

The multicast operation makes use of the spanning tree  $BST_s$  for the transmission of messages originating at node  $s$ . As per the 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 to be  $N/32$ ,  $N/16$ ,  $N/8$  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  $((r_1 + i \times r_2) \bmod N)$ , where  $0 \leq r_1, r_2 \leq (N - 1)$ ;  $i$  ranges from 0 to  $\frac{N}{32} - 1$ ,  $\frac{N}{16} - 1$ ,  $\frac{N}{8} - 1$  and  $\frac{N}{4} - 1$ , respectively. 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 communications bottlenecks arising as a result of the increased, random traffic patterns.

## 6.2. Simulation results

The randomized multicast and broadcast algorithms 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 the predetermined threshold value presented in Sub-Section 5.1. The probability for  $k$  successes in the specified time interval is given by

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

where  $\alpha$  is the average number of initiations in the specified time interval of 20 cycles.  $k$  is a random positive integer generated each time using the system clock. The value of  $P[k]$  is maximum at  $k = \alpha$  and  $k = \alpha - 1$ , if  $\alpha$  is a positive integer and  $\alpha > 1$ . The value of  $\alpha$  has been chosen as 15 to have an increased probability of a processor initiating a message in any given cycle.

Table 3. Results of randomized many-to-all broadcasting

$GH_{n,k}$	No. of procs.	Execution time (cycles)							No. of mesgs.
		Buffer size (messages)							
		3	4	5	6	7	8	$\infty$	
$GH_{2,8}$	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

Table 4. Results of randomized multicasting, with one-quarter 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	$\infty$	
$GH_{2,8}$	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

All processors having in a given cycle a probability greater than the threshold value, which was preset to  $2/3 \times$  (Maximum Probability), are considered message initiators. For each processor that happens to be a message initiator, a translated spanning tree/subgraph is created dynamically in a distributed manner; the message is first distributed to all of the source's children in the case of broadcasting and to the appropriate set of its children in the case of multicasting.

Simulation results of randomized many-to-all broadcasting are presented in Table 3; only the first 20 cycles were assumed to generate messages for all simulations in this paper. '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 channel contentions if the basic algorithm from [12] is used repeatedly. Also, the buffer size has a very significant effect on the total time.

Tables 4–7 present results of randomized multicasting, where the number of destinations for each multicast is always one-quarter, 1/8th, 1/16th and 1/32nd, respectively, of the

Table 5. Results of randomized multicasting, with 1/8th 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	$\infty$	
$GH_{2,8}$	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	675	565	486	427	24	414020
$GH_{3,25}$	15625	742	562	454	382	330	292	23	458154

Table 6. Results of randomized multicasting, with 1/16th 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	$\infty$	
$GH_{2,8}$	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

total number of processors; the destination addresses are chosen randomly, as discussed earlier. 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 if the basic algorithm for broadcasting proposed in [12] is adapted for multicasting, this 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 Section, as well as respective performance results.

As the size of the generalized hypercube increases, the amount of information being exchanged among the processors in the system grows alarmingly. With a practical limit on the buffer size, it was noticed that generalized hypercube systems  $GH_{n,k}$  with a larger value for  $k$  and a smaller value for  $n$  seemed to have a smaller number of contentions than systems with almost the same number of processors having a larger value for  $n$  and a smaller value for  $k$ ; it is the result of the former's higher bisection width.

Table 7. Results of randomized multicasting, with 1/32nd 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	$\infty$	
$GH_{2,8}$	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

Table 8. Results of randomized multicasting, with 1/8th 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	$\infty$	
$GH_{2,8}$	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

To demonstrate in a more dramatic fashion the need for a better multicast algorithm, we show in Tables 8–10 results where the same destinations are always chosen for all multicasts, independently of the address of the source processor.

## 7. ADAPTIVE ROUTING

We present here an adaptive routing algorithm for randomized multicasting as well as relevant simulation results. In the case of adaptive routing in parallel systems, some messages do not follow the shortest paths to their destinations, in an attempt to avoid channel contentions[15].

In our adaptive routing algorithm, each sending/intermediate processor compares the numbers of messages in all its outboxes whenever deterministic routing may result in a

Table 9. Results of randomized multicasting, with 1/16th 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	$\infty$	
$GH_{2,8}$	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 10. Results of randomized multicasting, with 1/32nd 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	$\infty$	
$GH_{2,8}$	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	122	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

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. Despite the increase by one hop in the path length, the communication time is often reduced because the message will most probably reach its intended intermediate or destination node in the next cycle.

Table 11 shows results of such simulations, where the number of random destinations for each multicast is one-quarter of the total number of processors. Comparing the results with earlier results for deterministic routing presented in Table 4, we observe that adaptive routing reduces channel contentions and this often results in slightly reduced execution times.

To demonstrate even more dramatic improvements due to adaptive routing, we present in Tables 12–14 results of randomized multicasting where the destinations are identical for

Table 11. Results of randomized multicasting, with one-quarter 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	$\infty$	
$GH_{2,8}$	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	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	351	295	255	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 12. Results of randomized multicasting, with 1/8th 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	$\infty$	
$GH_{2,8}$	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

all multicasts; the number of destinations is 1/8th, 1/16th, and 1/32nd, respectively, of the total number of processors. These results should be compared with relevant results for deterministic routing presented earlier in Tables 8–10. It becomes obvious that for large systems the proposed adaptive routing algorithm results in very significant performance improvements. For a small system with only two dimensions, the deterministic algorithm sometimes results in better performance because the multicast tree has only two levels.

## 8. CONCLUSIONS

We have 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

Table 13. Results of randomized multicasting, with 1/16th 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	$\infty$	
$GH_{2,8}$	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 14. Results of randomized multicasting, with 1/32nd 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	$\infty$	
$GH_{2,8}$	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

presented in [12] 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.

## ACKNOWLEDGEMENTS

The work presented in this research was supported in part by NSF and DARPA, also co-sponsored by NASA, under the New Millennium Computing Point Design Grant ASC-9634775.

---

**REFERENCES**

1. S. G. Ziavras, 'RH: A versatile family of reduced hypercube interconnection networks', *IEEE Trans. Parallel Distrib. Syst.*, **5**(11), 1210–1220 (1994).
2. W. Dally, 'Network and processor architecture for message-driven computers', in *VLSI and Parallel Computation*, R. Suaya and G. Birtwistle (Eds.), Morgan Kaufmann Publications, 1990, pp. 140–222.
3. S. G. Ziavras, 'Generalized reduced hypercube interconnection networks for massively parallel computers', in *Networks for Parallel Computations*, American Mathematical Society, D. F. Hsu, A. Rosenberg and D. Sotteau (Eds.), 1995, pp. 307–325.
4. 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.
5. S. G. Ziavras, 'Investigation of various mesh architectures with broadcast buses for high-performance computing', *VLSI Des.*, Special Issue High Perf. Bus-Based Arch., **9**(1), 29–54 (1999).
6. L. N. Bhuyan and D. P. Agrawal, 'Generalized hypercube and hyperbus structures for a computer network', *IEEE Trans. Comput.*, **33**(4), 323–333 (1984).
7. L. D. Wittie, 'Communication structures for large networks of multicomputers', *IEEE Trans. Comput.*, **C-30**(4), (1981).
8. S. G. Ziavras, 'On the problem of expanding hypercube-based systems', *J. Parallel Distrib. Comput.*, **16**(1), 41–53 (1992).
9. S. G. Ziavras, 'Scalable multifolded hypercubes for versatile parallel computers', *Parallel Proc. Lett.*, **5**(2), 241–250 (1995).
10. 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.
11. T. Szymanski, 'A fiber optic hypermesh for SIMD/MIMD machines', *Supercomputing Conf.*, Nov. 1990, pp. 103–110.
12. P. Fragopoulou, S. G. Akl and H. Meijer, 'Optimal communication primitives on the generalized hypercube network', *J. Parallel Distrib. Comput.*, **32**, 173–187 (1996).
13. J. K. Antonio, L. Lin and R. C. Metzger, 'Complexity of intensive communications on balanced generalized hypercubes', *Int. Parallel Process. Symp.*, 387–394 (1993).
14. S. G. Ziavras and A. Mukherjee, 'Data broadcasting and reduction, prefix computation, and sorting on reduced hypercube parallel computers', *Parallel Computing*, **22**, 595–606 (1996).
15. P. T. Gaughan and S. Yalamanchili, 'Adaptive routing protocols for hypercube interconnection networks', *Computer*, **26**(5), 12–23 (1993).