Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Efficient Hypercube Communications

by

Shreyas R. Bhatt

Thesis submitted to the Faculty of the Graduate School of
the New Jersey Institute of Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering.
Jan. 1992

APPROVAL SHEET

Title of Thesis: Efficient Hypercube Communications

Name of candidate: Shreyas R. Bhatt
Master of Science in Electrical Engineering, 1991

Thesis and abstract approved: _____ 12/19/91

Dr. S. G. Ziavras **Date**
Assistant Professor
Department of Electrical and
Computer Engineering
New Jersey Institute of Technology

_____ 12/19/91

Dr. J. Carpinelli **Date**
Assistant Professor
Dept. of Electrical and
Computer Engineering
New Jersey Institute of Technology

_____ 12-19-91

Dr. E. Hou **Date**
Assistant Professor
Dept. of Electrical and
Computer Engineering
New Jersey Institute of Technology

Contents

List of Figures

List of Tables

ABSTRACT

EFFICIENT HYPERCUBE COMMUNICATIONS

Shreyas R. Bhatt, Master of Science, Electrical and Computer Engineering Department, 1991

Thesis directed by : Dr. Sotirios G. Ziavras, Assistant Professor

Hypercube algorithms may be developed for a variety of communication-intensive tasks such as sending a message from one node to another, broadcasting a message from one node to all others, broadcasting a message from each node to all others, all-to-all personalized communication, one-to-all personalized communication, and exchanging messages between nodes via fixed permutations. All these communication patterns are special cases of many-to-many personalized communication. The problem of many-to-many personalized communication is investigated here.

Two routing algorithms for many-to-many personalized communication are presented here. The algorithms proposed yield very high performance with respect to the number of time steps and packet transmissions. The first algorithm yields high performance through attempts to equibalance the number of messages at intermediate nodes. This technique tries to avoid creating a bottleneck at any node and thus reduces the total communication time. The second algorithm yields high performance through one-step time-lookahead equibalancing. It chooses from the candidate intermediate nodes the one which will probably have the minimum number of messages in the next cycle.

The proposed algorithms are compared with two existing algorithms. Simulation results for all these algorithms are given for the purpose of comparison. Analytical results for some special types of communication patterns are also presented.

Chapter 1

Introduction

1.1 Hypercubes and some applications

An *n-cube* or *n-dimensional hypercube* consists of 2^n nodes each with its own memory and directly connected with n neighboring nodes [1]. If distinct n-bit binary addresses are assigned to the nodes, then two nodes are directly connected with each other if and only if their binary addresses differ by a single bit. A 3-cube can be represented as an ordinary cube in three dimensions where the vertices are the $8 = 2^3$ nodes of the 3-cube; see Fig. 1. More generally, one can construct an n-cube as follows. First, the 2^n nodes are labelled by the 2^n binary numbers from 0 to $2^n - 1$. Then a link between two nodes is drawn if and only if their binary numbers differ by one and only one bit.

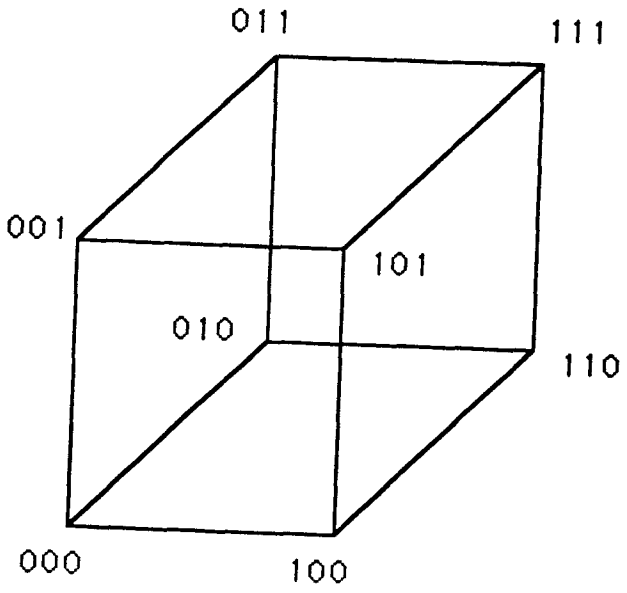

Figure 1.1: A 3-cube

Any connected multiprocessor system should allow for its processors to exchange data in all possible combinations. Let A and B be any two nodes of the n-cube and consider the problem of sending data from node A to node B. The way in which this is achieved in ensemble architectures is to move the data along a path from A to B crossing a (possibly minimum) number of processors. By definition, the length of a path between two nodes is simply the number of edges in the path. The *Hamming* distance between a pair of nodes in an n-cube, which is the shortest distance between the two nodes, is equal to the number of bits the two nodes' binary addresses differ. It can be shown that there exist j parallel paths (paths which do not have common nodes) of length j between any pair of nodes with Hamming distance equal to j. If we relax the restriction that the length must be Hamming, then as many as n parallel paths can be found between any pair of nodes with the length of each path being at most Hamming length plus 2.

Mapping other topologies into the hypercube is important due to the following reasons.

1. Some algorithms may be developed for other architectures for which they fit perfectly. Then one might wish to implement such an algorithm on a hypercube-based system with little additional programming effort. If the original architecture could be embedded into the hypercube, this will be easy to achieve.

2. A given problem may have a well-defined structure which leads to a particular pattern of communication. When running the algorithm on a hypercube-based system, the efficient mapping of the structure may result in substantial savings in communication time.

Efficient embeddings into the hypercube have been developed for several simple configurations. Rectangular meshes have been embedded by Chan and Saad [14], Ma and Tao [15], and Johnsson and Ho [16]. Algorithms for embedding binary trees into hypercubes have been developed by Wu [17], Bhatt and Ipsen [20]. and Bhatt *et. al.* [22]. Algorithms for embedding rings into hypercubes have been suggested by Chan and Lee [18]. Algorithms for embedding pyramids into hypercubes have been proposed by Patel and Ziavras [13], Stout [23], Miller and Stout [24], and Chang *et. al.* [25]. Finally, the embedding of d-dimensional grids into hypercubes has been suggested by Chan [19].

The hypercube network has been used in a wide variety of application domains, including image processing. Image processing is concerned with the manipulation of pictorial data. An image is generally represented as a two-dimensional array of numbers (pixels) each of which corresponds to the brightness or color value at some point in the image. There are many operations commonly performed on images for specific purposes: filtering, level transformations and thresholding, edge and region enhancements, feature recognition, statistical measurements, and so on. Such operations are efficiently performed on hypercubes.

Another application of the hypercube is for some computation-intensive numeric

applications, like matrix manipulations.

1.2 Hypercube communications

When algorithms are executed in a network of parallel processors, it is often necessary to exchange some intermediate information between the processors. The interprocessor communication time which is considered overhead, may be substantial relative to the time needed exclusively for computations, so it is important to carry out the information exchange as efficiently as possible. Information is transmitted along the hypercube links in groups of bits called *packets.* In our algorithms we assume that the time required to cross any link is the same for all packets and is taken to be one unit of time. Thus, our analysis applies to communication problems where all packets have roughly equal length.

Various types of communication transfers have been analyzed by Johnsson and Ho [2]. Their analysis includes one-to-all broadcasting, one-to-all personalized communication, all-to-all broadcasting, and all-to-all personalized communication. Contrary to broadcasting, where a data set is copied from one node to all other nodes or a subset of them, in personalized communication each sender sends different data sets to different destinations. Thus, personalized communication differs from broadcasting in the sense that there is no replication or reduction of data. Some special cases of routing with small buffers in the nodes were also analyzed by Kuszmaul [3]. His analysis assumes injective routing in which no two nodes are allowed to send messages to the same node at the same time. In addition, efficient routing using randomization for arbitrary permutations has been suggested by Valiant [4]. Stout and Wager have analyzed various communication patterns including broadcasting, opposite corner transfers, transfers between pairs of arbitrary nodes, and one-to-all personalized communication [5]. Their analysis assumes a link-bound model for hy-

percubes in which each node can fully utilize all of its bidirectional communication links simultaneously. Bertsekas, Ozveren, Stamoulis, and Tsitsiklis have analyzed some basic hypercube communication problems, including the problem of a single processor sending a different packet to each of the other processors, the problem of simultaneous broadcast of the same packet from every processor to all other processors, and the problem of simultaneous exchange of different packets between every pair of processors [7]. In their analysis, they assume that all incident links of a node can be used simultaneously for packet transmission (this is the Multiple Link Availability model).

1.3 Motivations and Objectives

As already noted in the previous section, several algorithms may be proposed for different communication patterns which are all special cases of many-to-many personalized communication. Analytical solutions for some special cases of communication transfers have been presented by Johnsson and Ho, and Stout and Wager. Their analyses present lower bounds on the communication time. Hence, we decided to present absolute total communication time for some special communication transfers. Chapter 2 is dedicated on the analytical treatment of some communication transfers.

Since there is no general analytical solution possible for the important problem of many-to-many personalized communication, we decided to develop heuristics to efficiently solve the problem. We developed two new routing algorithms which tend to yield reasonably small communication times. The main objective of both algorithms is to keep equal numbers of messages at neighboring intermediate nodes during each communication cycle. This is called the equibalancing principle. It avoids creating communication bottlenecks at intermediate nodes and thus reduces

the overall communication time. The second algorithm is a modified version of the first algorithm. Not only does the latter algorithm apply the equibalancing principle, but also looks ahead one step in the future. In addition, the performance of our algorithms is compared with the performance of two existing algorithms. Even though the computation complexity of the proposed algorithms is a little higher than the computation complexity of the latter two algorithms, simulation results show that the former pair of algorithms achieve higher performance.

Our analysis focuses on the following hypercube model:

- Each node may receive more than one message at a time on different channels, but may send only one message at a time to a single channel.
- The architecture operates in the SIMD (Single-Instruction stream, Multiple-Data stream) mode of computation.
- All channels are bidirectional and operate in the full duplex mode.
- Each node of an n-dimensional hypercube has n infinite capacity buffers attached to its n channels.
- All packets are of the same length.

Chapter 2

Analysis of some special communication transfers

2.1 Introduction

In this chapter, we present analytical results for some special communication transfers. The analysis includes one-to-one, one-to-many, and many-to-one transfers. Our analysis makes use of the model presented in the previous chapter. The problem of many-to-many communication is too complicated to be solved analytically. Heuristics are developed for this communication problem and are discussed in the remaining chapters.

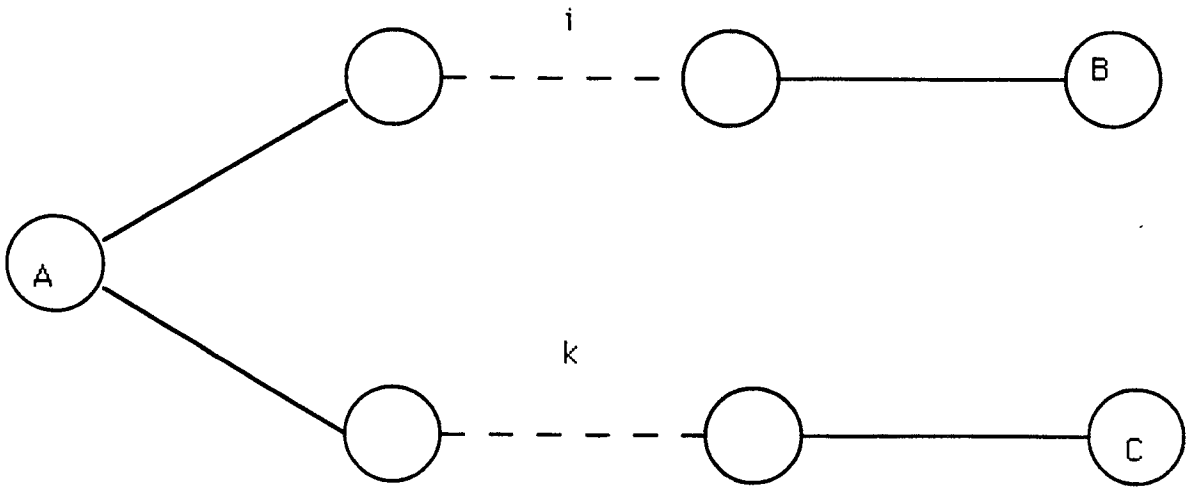

Figure 2.1: One-to-two personalized communication

2.2 One-to-many communication

We start with a very special case where one node transmits m messages to another node.

Proposition 1:

In an n-cube, if m messages are to be transmitted from one node to another node which is l links away from it, then the communication time T_c is $l + (m - 1)$.

Next, we consider one-to-two personalized communication in which one node transmits two different messages to two different nodes.

Proposition 2:

In an n-cube, two messages are to be transmitted from node A to two different nodes B and C located i and k links away from it (one to each node), where $i, k \leq n$ (Fig. 2.1).

(a) If $i < k$, in order to minimize the communication time, the message to node C should be transmitted first and the communication time is $T_c = k$.

(b) If $i = k$, any message can be transmitted first and $T_c = i + 1$.

(c) If $i > k$, the message to node B should be transmitted first and $T_c = i$.

Next, we consider the case where one node transmits m messages to m nodes (one message to each node).

Proposition 3:

If m messages are to be transmitted from one node to m different nodes (one message to each node) with Hamming distances $l_1, l_2, \ldots, l_m$ (after arranging these distances in ascending order), then the messages should be transmitted in a sequence such that the distances of the nodes are in descending order (i.e., the message to the node having maximum Hamming distance should be transmitted first).

(a) If none of the lengths is repeated, $T_c = l_m$.

(b) If some of the lengths are repeated, $T_c = l_1 + (m - 1)$.

Finally, we consider a generalized case of one-to-many communication where the source node can transfer any number of messages to any destination.

Proposition 4:

If $m_1, m_2, \ldots,$ and m_i messages are to be transmitted from one node to nodes having Hamming distances $l_1, l_2, \ldots,$ and l_i respectively, then the messages should be transmitted in a sequence such that the Hamming distances of the nodes are in descending order ($l_i \geq l_{i-1} \geq \ldots \geq l_1$ in Fig. 2.2).

(a) If $l_i + (m_i - 1) > S_m + (l_1 - 1)$, then $T_c = l_i + (m_i - 1)$, where S_m is the total number of messages the given node must transmit.

(b) If $l_i + (m_i - 1) < S_m + (l_1 - 1)$, then $T_c = S_m + (l_1 - 1)$.

In the above case, if T_k is the transmission time for the node which is l_k links away from the source ($k = 1, 2, \ldots, i$) and ST_{k-1} is the sum of $T_1, T_2, \ldots, T_{k-1}$, then, if $T_k >> ST_{k-1}$, the order of transmission for nodes which are $l_1, l_2, \ldots, l_{k-1}$ links away from the given node can be altered.

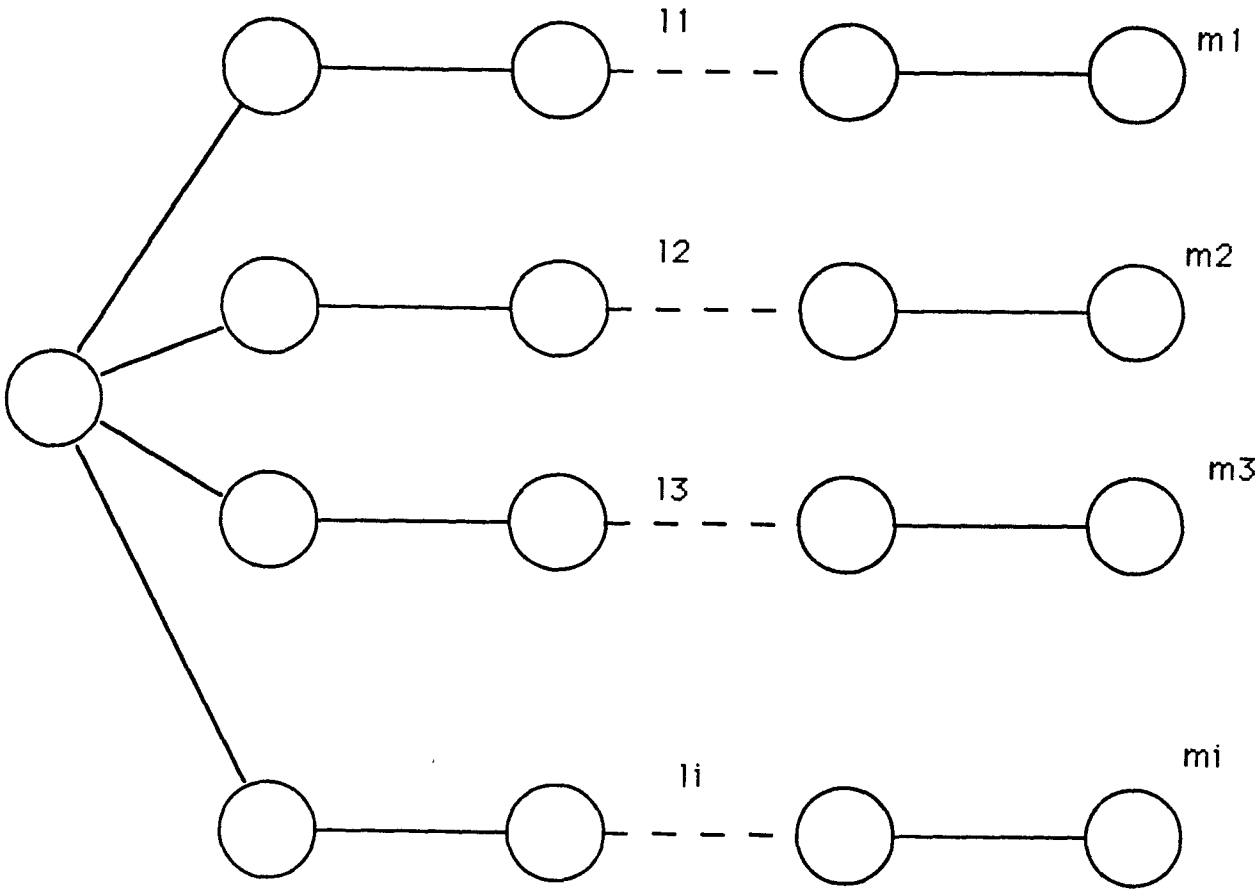

Figure 2.2: One-to-many communication

2.3 Many-to-one communication transfers

In this section, we will consider various cases where one node receives messages from more than one node and transmits to a single other node. While considering this type of transfer, all the paths to the given node should be parallel (non-intersecting and, of course, Hamming) in order to minimize the communication time. If it is not possible to select parallel paths, the amount of intersection should be kept to a minimum and the corresponding intersecting points should be near the destination node. In the case of parallel paths, the communication time is equal to the maximum distance traveled.

First of all, we consider the case where a node receives messages from two nodes and sends a single message to another node.

Theorem 1:

If a node receives m_1 and m_2 messages from two different nodes being l_1 and l_2 links away from it respectively and transmits a message to a node l_3 links away from it (see Fig. 2.3), then

(a) If $l_1 + (m_1 - 1) < l_2$, then $T_c = (l_2 + l_3) + (m_2 - 1)$.

(b) If $l_1 + (m_1 - 1) > l_2$, then $T_c = (l_1 + l_3) + (m_1 + m_2 - 1)$.

Proof:

(a) Let node C receive m_1 and m_2 messages from nodes A and B respectively and transmit a message to node D. Now, if the transmission time from A to C is less than the distance between B and C, node C would have transmitted all of m_1 messages to node D by the time the first message from B reaches C. Hence, in this case, the communication time will depend on the transmission time from node B to node D. But the transmission time from A to C is $l_1 + (m_1 - 1)$ and the transmission time from B to D is $(l_2 + l_3) + (m_2 - 1)$. This proves the theorem.

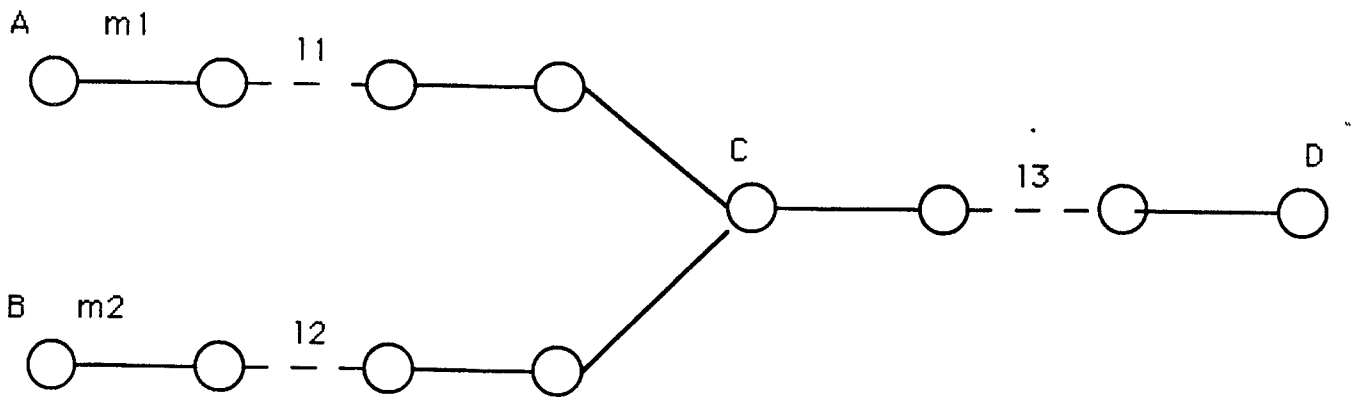

Figure 2.3: Two-to-one communication

(b) If the transmission time from A to C is greater than the distance between B and C, at time l_2 C will have transmitted $(l_2 - l_1)$ of its m_1 messages and will be left with $m_1 - (l_2 - l_1) + m_2$ more messages. Hence, $T_c = l_2 + l_3 + (m_1 - (l_2 - l_1) + m_2)$, or $T_c = (l_1 + l_3) + (m_1 + m_2 - 1)$. □

Next, we consider a special case where many nodes transmit different messages to a single destination node via an intermediate node located one hop away from all the source nodes. In this case, the intermediate node becomes a bottleneck for the communication. This situation is shown in Fig. 2.4. Source node S_i sends m_i messages to the destination node D via intermediate node I, where $1 \leq i \leq k$ and k is the number of source nodes. Also, messages are arranged in ascending order, i.e., $m_1 \leq m_2 \leq m_3 \leq \ldots \leq m_k$. Let l be the Hamming distance between the intermediate node I and the destination node D. Hence, the Hamming distance between each source node and the destination node D is $1 + l$. The communication time of this transfer can be minimized by properly utilizing the intermediate node.

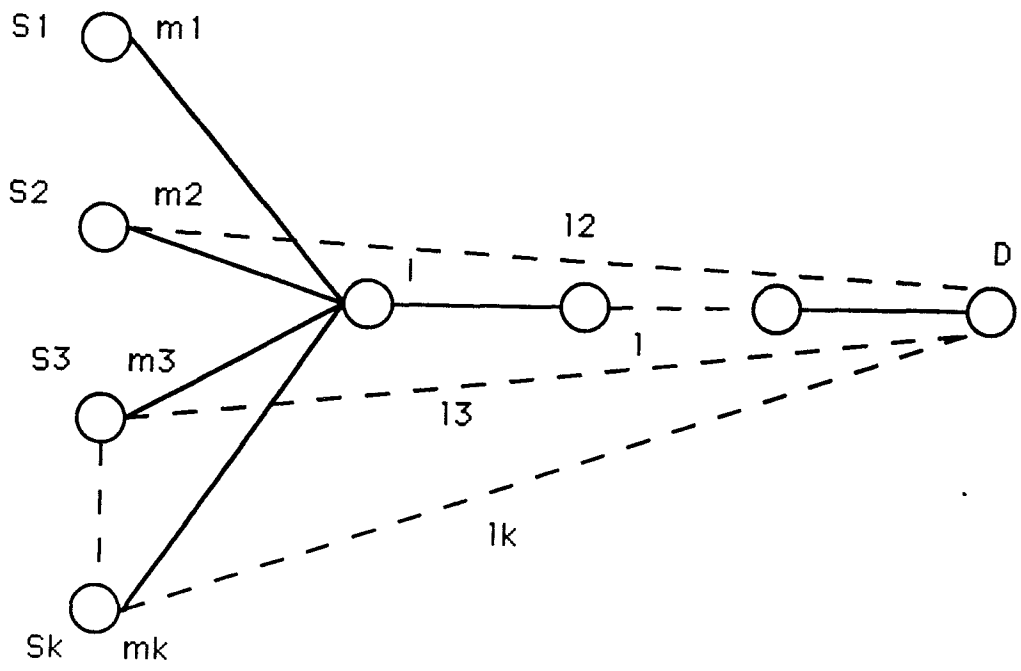

Figure 2.4: Many-to-one communication (I is one hop away)

Whenever a source node is utilizing this intermediate node, the rest of the source nodes should divert their messages onto other paths which may be longer than the Hamming path. By imposing some condition on the length of these diverted paths, it is possible to minimize the communication time of the transfer. This condition is investigated in the following theorem.

Theorem 2:

Let l_i be the length of the diverted path between the source node S_i and the destination node D, where $2 \leq i \leq k$. If $l_i = (m_i - m_{i-1}) + (1 + l)$, then the communication time will be minimum and will be equal to $(m_k - 1) + (1 + l)$.

Proof: Here, let us assume that node S_1 first starts transmitting via the intermediate node I. At this time, nodes S_i transmit to the destination node D via paths l_i, where $2 \leq i \leq k$. As soon as the node S_1 completes its transmission and I be-

comes available, node S_2 transmits the rest of its messages through node I and so on. Node S_1 will take $(m_1 - 1) + (1 + l)$ time for its transfer. For node S_2, m_1 messages out of m_2 will be transmitted on path l_2, the communication time for that will be $(m_1 - 1) + l_2$, and the rest of the $m_2 - m_1$ messages will be transmitted through the Hamming path of length $1 + l$, the communication time of which will be $m_1 + (m_2 - m_1) - 1 + (1 + l)$ or $(m_2 - 1) + (1 + l)$. Hence, the overall communication time for S_2 will be the maximum of these two quantities. Similarly, for node S_k, the communication time will be the maximum of $(m_k - 1) + (1 + l)$ and $(m_{k-1} - 1) + l_k$. Since m_k is the maximum number of messages, the overall communication time of the system is simply the communication time of the node S_k. Hence, for the communication time to be minimum, the following condition must be satisfied.

$(m_{k-1} - 1) + l_k = (m_k - 1) + (1 + l)$

or, $l_k = (m_k - m_{k-1}) + (1 + l)$. □

Corollary. If the distance between the source nodes and the intermediate node is 2 (see Fig. 2.5), then the condition for minimum communication time becomes $l_k = (m_k - m_{k-1}) + (l + 2)$.

Proof: Node S_k transmits m_{k-1} messages through path l_k, the communication time for which is $m_{k-1} + l_k - 1$, and $m_k - m_{k-1}$ messages through the Hamming path of length $2+l$, for which the communication time is $m_{k-1}+(m_k-m_{k-1})+(2+l)-1$. For minimum communication time, $m_{k-1}+l_k-1 = m_k+1+l$ or $l_k = (m_k-m_{k-1})+(2+l)$. □

The analysis of more complex communication transfers is very difficult. For that, new routing algorithms are developed based on heuristics, which will be described in the following chapters.

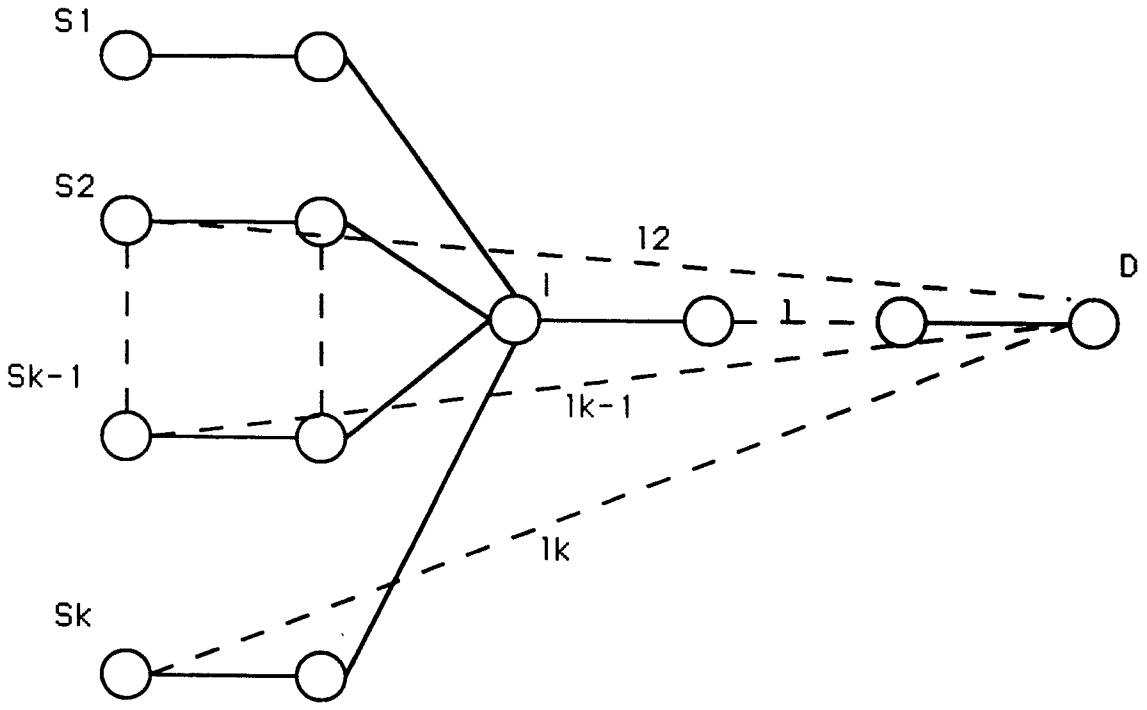

Figure 2.5: Many-to-one communication (I is two hops away)

Chapter 3

Existing Hypercube Communication Algorithms

3.1 Introduction

In this chapter, we present existing algorithms for some special communication transfers. Several algorithms have been suggested for many variations of broadcasting and personalized communications. Most of the algorithms give the lower bound on the communication time. Two existing algorithms, which are compared with the proposed ones in Chapter 5, are described here in detail.

3.2 Some Existing Algorithms

Various types of communication transfers have been analyzed by Johnsson and Ho [2]. Their analysis includes one-to-all broadcasting, one-to-all personalized commu-

nication, all-to-all broadcasting and all-to-all personalized communication. Some special cases of routing with small buffers in the nodes were analyzed by Kuszmaul [3]. He presented two formulations of the routing scheme. The first formulation delivered messages in $O(k^2)$ bit times using $O(k)$ bits of buffer space at each node in the k-dimensional hypercube. The second formulation assumed that there were several batches of messages to be delivered, and made certain assumptions about the cost of sending messages along the various dimensions of the hypercube. In this case, the latency for delivery time is still $O(k^2)$ bit times, but the throughput is increased to one set of messages every $O(k)$ bit times. For the first formulation, the routing was restricted to subsets of permutations (i.e., every node sends at most one message and receives at most one message). The second formulation indicated a way to perform routings which are subsets of H-permutations (i.e., every node sends at most H messages and receives at most H messages). His analysis assumed injective routing in which no two nodes are allowed to send messages to the same node at the same time.

Stout and Wager have analyzed various communication patterns including broadcasting, opposite corner transfers, transfers between pairs of arbitrary nodes, and one-to-all personalized communication [5]. Their algorithms are for link-bound hypercubes in which local processing time is ignored, communication time predominates, message headers are not needed because all nodes know the task being performed, and all nodes can use all communication links simultaneously. They present the lower bound on the communication time for each type of transfer. Their algorithms provide support to the belief that it is useful to build machines where all communication links can be used simultaneously.

Bertsekas, Ozveren, Stamoulis and Tsitsiklis [7] have analyzed basic communication problems for hypercubes, including the problem of a single processor sending

different packets to each of the other processors, the problem of simultaneous broadcast of the same packet from every processor to all other processors, and the problem of simultaneous exchange of different packets between every pair of processors. In their analysis, they assume that all incident links of a node can be used simultaneously for packet transmission (this is the Multiple Link Availability model). In addition, they assume that each of their algorithms is simultaneously initiated at all processors. This is a somewhat restrictive assumption, essentially implying that all processors can be synchronized with a global clock. The algorithms proposed are optimal in terms of execution time and commmunication resource requirements; that is, they require the minimum possible number of time steps and packet transmissions.

Adaptive fault-tolerant routing algorithms were presented by Chen and Shin [8]. They proposed some routing schemes for an *injured hypercube* with faulty links and/or nodes. To enable any nonfaulty node to communicate with any other nonfaulty node in an injured hypercube, the information on component failures has to be made available to nonfaulty nodes so as to route messages around the faulty components. They proposed a distributed adaptive fault-tolerant routing scheme for an injured hypercube in which each node was required to know only the condition of its own links. Despite its simplicity, this scheme was shown to be capable of routing messages successfully in an injured n-dimensional hypercube as long as the number of faulty components was less than n. Moreover, it was proved that this scheme routed messages via shortest paths with a rather high probability and the expected length of a resulting path was very close to that of a shortest path. Due to the insufficient information on faulty components, however, the paths chosen by the above scheme might not always be the shortest. To guarantee the routing of all messages via shortest paths, they proposed the inclusion in every node of more

information than that on its own links. The effects of this additional information on routing efficiency were analyzed, and the additional information to be kept at each node for the shortest path routing was determined.

Katseff has proposed routing algorithms for the incomplete hypercube [9]. Unlike hypercubes, incomplete hypercubes can be used to implement systems with any number of processors. However, the basic building block is the hypercube. The routing and broadcast algorithms were proved to be simple and deadlock-free. Lan, Esfahanian and Ni proposed algorithms for multicast (one-to-many) communication in hypercube multiprocessors [11]. They first proposed a graph theoretical model, the Optimal Multicast Tree (OMT), for interprocessor communication in distributed-memory multiprocessors. A heuristic Greedy multicast algorithm which guaranteed a minimum message delivery time was then proposed. It was proved by simulation results that the performance of the Greedy algorithm was very close to the optimal solution. Routing of multicast messages was done in a distributed manner. The hardware design of a VLSI router which supports all types of communications was also briefly discussed. In addition, efficient routing using randomization for arbitrary permutations has been suggested by Valiant [4]. Saad and Schultz presented optimal algorithms for single node scatter, multinode broadcast and total exchange problems [6]. Their analysis assumed hypercube links to be unidirectional.

All of these existing algorithms deal with some special cases of many-to-many communication transfers. Algorithms proposed by Johnsson and Ho deal with some special cases of the communication transfers that our proposed algorithms deal with and make similar assumptions. Hence, the next section describes their algorithms in some detail.

3.3 Hypercube communication algorithms by Johnsson and Ho

Johnsson and Ho have investigated broadcasting and personalized communication on hypercubes [2]. In broadcasting, a data set is copied from one node to all other nodes, or a subset of them. In personalized communication, a node sends a unique data set to all other nodes, or a subset of them. They consider broadcasting from a single node to all other nodes, *one-to-all* broadcasting, and concurrent broadcasting from all nodes to all other nodes, or *all-to-all* broadcasting. For personalized communication, they consider *one-to-all personalized communication* and *all-to-all personalized communication*. The difference between broadcasting and personalized communication is that in the latter no replication/reduction of data takes place. If a tree is used to represent the transfers, then the bandwidth requirement is highest at the root and is reduced monotonically towards the leaves. For single-source broadcasting and personalized communication, a *one-to-all communication graph* is required. Graphs of minimum height have minimum propagation time which is the overriding concern for small data volumes, or a high overhead for each communication action. For large data volumes, it is important to use the bandwidth of a Boolean cube effectively, in particular, if each processor is able to communicate on all its ports concurrently. Johnsson and Ho have proposed three new spanning graphs for Boolean n-cubes: one consisting of n edge-disjoint binomial trees (nESBT); one that consists of n rotated spanning binomial trees (nRSBT); and one *balanced tree* (SBnT), i.e., a tree with fan-out n at the root and approximately N/n nodes in each subtree. They prove some of the critical topological properties of the new one-to-all communication graphs and compare them to Hamiltonian paths and binomial tree embeddings. They derive lower bounds on communication time for

each type of communication they consider.

They generalize the one-to-all communication to all-to-all communications and study the interleaving of communications from different sources by defining *all-to-all communication graphs* as the union of *one-to-all communication graphs*. They investigate four types of communication transfers: 1) *one-to-all broadcasting;* 2) *one-to-all personalized communication;* 3) *all-to-all broadcasting*, and 4) *all-to-all personalized communication*. They show that for communications restricted to one port at a time, their spanning binomial tree scheduling results in communication times within a factor of two of the best known lower bounds for communications 2, 3, and 4. For case 1, they claim that the scheduling they define for the n edge-disjoint spanning binomial trees completes within a factor of four of the best known lower bound, also for concurrent communication on all ports.

Their scheduling discipline defines the communication order for each port, and the order between ports for every nonleaf node. They basically describe two types of sceduling disciplines: reverse-breadth-first and postorder scheduling disciplines. The Reverse-Breadth-First (RBF) scheduling discipline defines the order of communication for each node of the hypercube. For each sender, a minimum spanning tree of height h, which is the maximum Hamming distance for all transfers in the n-dimensional hypercube, is first constructed with the root being the sender. Then, the scheduling discipline is such that the root sends out the data for the nodes at level $h - p$ during the p^{th} cycle, where $0 \leq p < h$. The data received by an internal node are propagated to the next level during the next cycle, if the data are not destined for the node itself. Thus, higher priority is given to the transmission of messages going to nodes corresponding to larger Hamming distances. In the postorder scheduling discipline, each node sends out the entire data set to each of its children before accepting its own data. They have also proposed various schedul-

ing disciplines for the implementation of special types of communication transfers. For each type of transfer and scheduling discipline, they have found a lower bound on the communication time. Nevertheless, our comparative analysis involves only their algorithm for many-to-many communication that uses the RBF scheduling discipline. As discussed in the first chapter, the reason for this is that all other communication transfers are special cases of many-to-many communication.

3.4 Brute-Force Algorithm

Another existing routing algorithm uses a brute-force approach. The scheduling discipline is such that messages for destination nodes which are at Hamming distance $n-(i-1)$ have transmission priority i, where $1 \leq i \leq n$, and priority 1 is the highest priority. If j is the Hamming distance between a source node i and a destination node k, there exist j parallel paths between nodes i and k. Hence, there are j candidates for the next intermediate node to which i can forward a message destined for node k. In the brute-force approach, selection of this next intermediate node out of the j candidate nodes is done at random. Fig. 3.1 demonstrates this idea. Node A is a source node and node H is a destination node at Hamming distance 3 away from it. Nodes B,C, and D are next intermediate nodes to which A can forward messages destined for H. In the brute-force approach, node A selects one node out of B,C and D at random and forwards a message destined for node H to this node.

As it can be expected, this algorithm cannot give very good communication time. This is because of the selection procedure for the next intermediate node. There are all chances that the next intermediate node selected might be the busiest node (having the maximum number of messages to be transmitted). This will certainly result in poor total communication time. If the selection procedure for

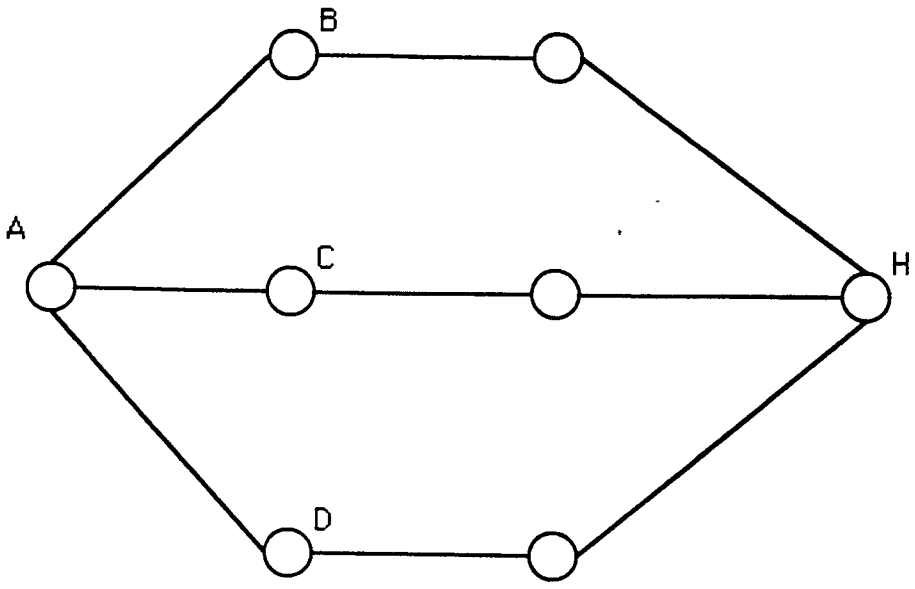

Figure 3.1: Brute-Force Algorithm

the next intermediate node can be improved, better overall communication time can be obtained. This is the main idea of our proposed algorithms which are presented in the following chapter.

Chapter 4

Proposed Hypercube Communication Algorithms

4.1 Introduction

This chapter proposes two new routing algorithms for efficient many-to-many personalized transfers in an n-cube. The main objective of both algorithms is to maintain equal numbers of messages at neighboring intermediate nodes during each communication cycle. This is called the equibalancing principle. It avoids creating communication bottlenecks at intermediate nodes and thus reduces the overall communication time. The second algorithm is a modified version of the first algorithm. Not only does the latter algorithm apply the equibalancing principle, but it also looks ahead one step into the future.

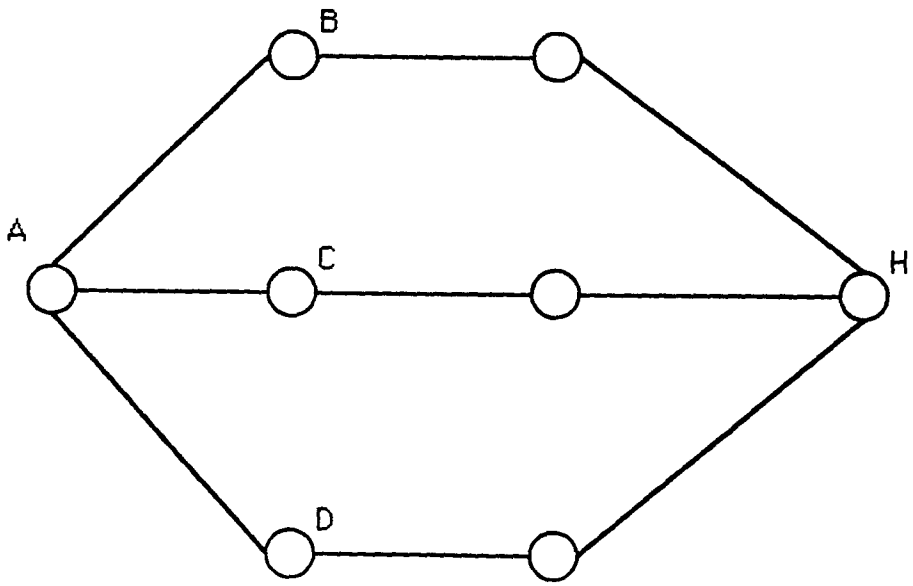

Figure 4.1: Equibalancing algorithm

4.2 Algorithm I: Equibalancing

This routing algorithm attempts to equibalance the numbers of messages in neighboring intermediate nodes all the time. The scheduling discipline is such that messages to destination nodes at Hamming distance $n-(i-1)$ have priority i, where $1 \leq i \leq n$, and priority 1 is the highest priority. For any given message that goes from a source node i to a destination node k, with j being the Hamming distance between the two nodes, there exist j parallel paths between nodes i and k. Hence, there exist j neighbors to i which are candidates to receive the message. Out of these j nodes, node i selects a node which currently has the smallest number of messages to transmit. Fig. 4.1 illustrates this idea. Node A is a source node and node H is a destination node at Hamming distance 3. Hence, there exist 3 parallel Hamming paths between A and H. Nodes B, C and D are intermediate nodes to which A can forward messages destined for H. The proposed algorithm selects out of these three nodes the node which has the minimum number of messages to transmit.

The algorithm is described below. It assumes that in the beginning of each communication cycle, the messages are already ordered in each node with respect to their Hamming distance. It also assumes that the head of the ordered list contains the message with the largest Hamming distance. The algorithm is:

Do in parallel for all nodes i, where $0 \leq i \leq 2^n - 1$:

Step 1: If the ordered list is empty then Stop, else go to Step 2.

Step 2: Let j be the Hamming distance for the message at the head of the ordered list. Find j possible next intermediate nodes for this message.

Step 3: Select out of the j nodes the one which currently has the minimum number of messages to transmit. If two or more nodes have this minimum number of messages, then select one of them at random.

Step 4: Transmit the message to this next intermediate node. Go to Step 1.

Hence, for an n-dimensional Hypercube, a queue of messages is formed at each node. Then, the messages at each queue are sorted and arranged with respect to their Hamming distances. The head of each queue contains the message with the largest Hamming distance. All the nodes then start transmitting messages to corresponding destination nodes. This process continues until all the queues become empty. Selection of a next intermediate node is based on the number of messages it has to transmit. The node with the minimum number of messages to be transmitted is selected as the next intermediate node. If more than one node has the same minimum number of messages to transmit, one node is selected at random out of these nodes as the next intermediate node. Hence, it can be seen

that the algorithm tries to keep an equal number of messages at each intermediate node. It avoids creating a bottleneck at any intermediate node. This results in a reduced overall communication time.

4.3 Algorithm II: One-step time-lookahead equibalancing

This algorithm is a modified version of Algorithm I which attempts equibalancing. Even though it employs the same scheduling discipline, it differs from the previous algorithm in the procedure that selects the next intermediate node. Algorithm I each time selects the node that has the minimum number of messages to transmit. The one-step time-lookahead algorithm chooses from the candidate intermediate nodes the one which will probably have the minimum number of messages in the next cycle. It makes predictions with regard to messages that nodes which are neighbors to these candidate nodes will also send in the next cycle. Since it cannot be determined in advance which neighbor will send a message to such a node, it is first determined how many neighbors have a message which can be transmitted to this node in the next cycle. This value is then multiplied by a threshold and the result is added to the number of messages that the node currently has in its buffer. The threshold represents a fraction of the maximum number of messages which the node may receive from its neighbors. The threshold may vary between 0 and 1. Fig. 4.2 demonstrates this idea. Node A is a source node and node H is a destination node at Hamming distance 3 away from it. Nodes B, C and D are next intermediate nodes to which A can forward messages destined for H. B1-B2, C1-C2 and D1-D2 are neighbors of B, C and D respectively. For each of these next intermediate nodes, it is first determined how many neighbors can transmit a message to it in the next cycle. This value is multiplied by the threshold and is added to the number of

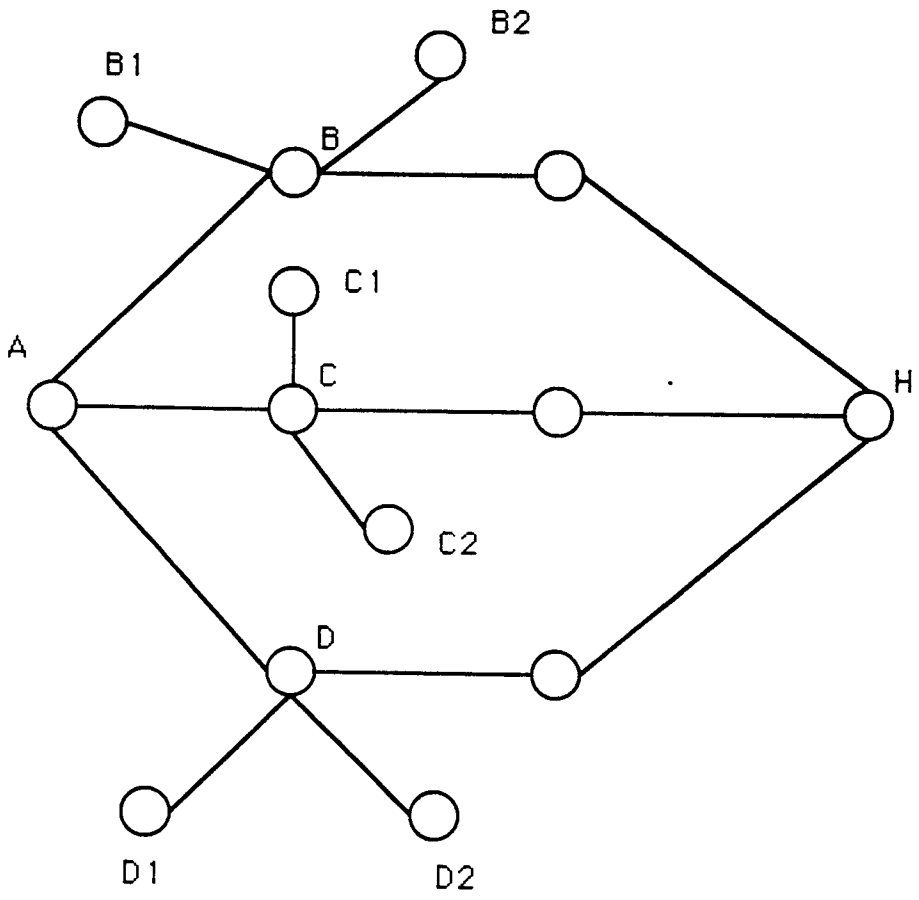

Figure 4.2: One-step time-lookahead equibalancing algorithm

messages for the next cycle. Node A will transmit the message destined for node H to the one of B, C and D which has the minimum predicted number. Of ourse, this routing algorithm can be applied only to deterministic algorithms. The sequence of operations to be performed by deterministic algorithms is known at static time.

Similarly to Algorithm I, an ordered list of messages is maintained for each node with respect to their Hamming distances. All nodes i, where $0 \leq i \leq 2^n - 1$, start executing the following routing algorithm.

Step 1: If the ordered list of messages is empty then Stop. else go to Step 2.

Step 2: Let j be the Hamming distance of the message at the head of the ordered list (i.e., j is the largest Hamming distance for the set of messages in the node). Find the j possible next intermediate nodes for this message.

Step 3: For each next intermediate node x, determine how many neighbors of x (except i) have a message that can be transmitted to node x in the current communication cycle. Multiply this number by the threshold and add it to the number of messages currently contained in the buffers of x to get m.

Step 4: Select that node x with the minimum value for m. If more than one nodes have the minimum value, then select one of them at random.

Step 5: Transmit the message to the selected node x. Go to Step 1.

As can be seen from the above, this algorithm looks one step in advance. The next intermediate node to be selected is the one that will probably have the minimum number in the next communication cycle. Although this algorithm makes only a prediction for selecting the next intermediate node, it should be expected to minimize the overall communication time.

Both of these proposed algorithms are simulated and their performance is compared with the existing ones. Simulation results and comparative analysis are pre-

sented in the next chapter.

Chapter 5

Simulation results and comparative analysis

5.1 Introduction

In this chapter, we simulate the two existing algorithms of Sections 3.3 and 3.4 and algorithms proposed in Chapter 4. Simulation results are tabulated and a comparative analysis involving the four routing algorithms is carried out. Simulation programs are attached in the appendix.

5.2 Simulation results and comparative analysis

All the four algorithms were simulated for an n-dimensional hypercube using the C language. The algorithms mainly differ in terms of their scheduling disciplines and the procedure that select the next intermediate node.

The following set of initialization parameters were used in our simulations.

- lnom1, lnom2 : The number of messages per node is uniformly distributed between these two limits, with $lnom1 \leq lnom2$.
- %senders : the percentage of nodes in the system that have messages to send.
- %dest : the percentage of destinations per sender. This represents a fraction of the total number of nodes to which a sender will send messages.
- n : the dimension of the hypercube.

com_time represents the total communication time, expressed in communication cycles, for given data sets.

Simulation results for the four algorithms are shown for the same data sets.

BR_FC: Brute_force ; RBF: suggested by Johnsson and Ho.

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	RBF
a.	1	1	100	100	201	205	213
b.	2	2	100	100	405	409	422
c.	3	3	100	100	609	613	623
d.	4	4	100	100	814	819	831
e.	5	5	100	100	1018	1022	1043

Table 5.1: Simulation results for a 6-cube (all-to-all communication).

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	RBF
a.	1	1	100	100	84	85	103
b.	2	2	100	100	167	170	179
c.	3	3	100	100	252	255	262
d.	4	4	100	100	337	340	348
e.	5	5	100	100	421	425	436

Table 5.2: Simulation results for a 5-cube (all-to-all communication).

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	RBF
a.	1	1	100	100	33	34	37
b.	2	2	100	100	67	68	68
c.	3	3	100	100	100	102	105
d.	4	4	100	100	133	135	141
e.	5	5	100	100	167	168	169

Table 5.3: Simulation results for a 4-cube (all-to-all communication).

Tables 5.1 through 5.3 show results for all-to-all communication and Tables 5.4 through 5.6 show results for many-to-many personalized communication. In the simulation results, the communication time for the one-step time-lookahead algorithm is shown for the optimum threshold value. It was found that at lower loads, the threshold has more effect on the communication time than at higher loads.

From the simulation results, it is apparent that the algorithm that attempts equibalancing performs much better than the two existing algorithms, i.e., the algorithm that applies a brute force approach and the algorithm that applies reverse breadth first scheduling (as suggested by Johnsson and Ho). The reason is that the equibalancing algorithm avoids the creation of bottlenecks at intermediate nodes and thus tends to reduce the total communication time. The poor performance of Johnsson's algorithm is due to its scheduling discipline. Recall that in reverse breadth first scheduling, the root node (sender) sends out data for nodes at level $h-p$ during the p^{th} cycle, where h is the maximum Hamming distance and $0 \leq p < h$. Hence, if the root does not have any message to send to nodes at level $h-p$, that cycle is just wasted. The brute force algorithm has performance inferior to that of the equibalancing algorithm because random selection of the next intermediate node does not generally give better performance.

As can be seen from the simulation results, when the system is lightly loaded, i.e., the percentage of senders is more and the percentage of destinations per sender is less, the one-step time-lookahead algorithm performs slightly better than the equibalancing algorithm. The former algorithm tends to avoid the next intermediate node which is expected to receive more messages from its neighbors in the next cycle. Optimum range for the threshold turns out to be between 0.7 and 1.0. Tables 5.7 and 5.8 show the communication time as a function of the threshold.

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	BR_FC
a.	3	7	90	20	149	155	215
b.	1	9	40	80	237	241	384
c.	1	5	20	90	127	127	160
d.	2	8	50	70	267	267	369
e.	5	7	50	50	287	287	421

Table 5.4: Simulation results for a 6-cube (many-to-many communication).

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	BR_FC
a.	5	10	90	20	111	113	147
b.	2	15	90	40	207	207	314
c.	1	9	30	80	100	100	135
d.	2	10	90	10	52	53	83
e.	2	15	20	90	173	173	215

Table 5.5: Simulation results for a 5-cube (many-to-many communication).

	lnom1	lnom2	%senders	%dest	com_time		
					ALG II	ALG I	BR_FC
a.	1	8	90	40	43	44	59
b.	2	8	20	90	39	39	40
c.	1	10	80	80	68	68	97
d.	3	23	50	90	196	196	234
e.	3	9	90	10	15	15	19

Table 5.6: Simulation results for a 4-cube (many-to-many communication).

threshold	com_time
0.2	153
0.4	153
0.6	151
0.8	149
1.0	149

Table 5.7: Communication time as a function of the threshold.
6-cube: lnom1 = 3 : lnom2 = 7 : %senders = 90 ; %dest = 20.

threshold	com_time
0.2	113
0.4	113
0.6	112
0.8	112
1.0	111

Table 5.8: Communication time as a function of the threshold.
5-cube: lnom1 = 5 ; lnom2 = 10 ; %senders = 90 ; %dest = 20.

Chapter 6

Conclusion

This research has proposed two routing algorithms for efficient implementation of many-to-many personalized transfers in an n-dimensional hypercube. For some very special cases of communication transfers, we presented analytical results for the overall exact communication time. These special cases included mainly one-to-many and many-to-one personalized communication. The performance of the proposed algorithms was compared with the performance of two existing algorithms; the first algorithm was suggested by Johnsson and Ho, while the other algorithm applies a Brute-Force technique. The algorithm suggested by Johnsson and Ho gives poor performance with regards to the total communication time. This is due to its Reverse-Breadth-First scheduling discipline. If a root does not have any message to send to nodes at a particular level, that cycle is just wasted. The poor performance of the Brute-Force algorithm is due to the random selection of the next interme-

diate node. The first algorithm proposed attempts to equibalance the number of messages at intermediate nodes and was shown to perform superior to both existing algorithms. The reason is that this algorithm selects as the next intermediate node the one which has the minimum number of messages to transmit. In some special cases, when the system is lightly loaded, the second proposed algorithm, which applies one-step time-lookahead equibalancing, performs slightly better than the equibalancing algorithm. This is because it avoids the next intermediate node which is expected to receive more messages from its neighbors in the next cycle.

Both of the proposed algorithms attemps to avoid the creation of a bottelneck at any node. Hence, this avoids unbalance of traffic in the system. This, in turn, results in a reduced overall communication time. This clearly shows that the careful selection of the next intermediate node reduces the total communication time. The one-step time-lookahead algorithm can be extended to multiple-step time-lookahead algorithm for even better communication time.

We also need to emphasize that the proposed algorithms are recommended for deterministic application algorithms; otherwise, the associated overhead may be too high.

Chapter 7

Bibliography

Bibliography

[1] Y. Saad and M. Schultz, "Topological Properties of Hypercubes," *IEEE Trans. Computers*, Vol. C-37, No. 7, July 1988, pp. 867-872.

[2] S. L. Johnsson and C. T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computers*, Vol. C-38, No. 9, Sept. 1989, pp. 1249-1268.

[3] B. Kuszmaul, "Fast Deterministic Routing on Hypercubes Using Small Buffers," *IEEE Trans. Computers*, Vol. C-39, No. 11, Nov. 1990, pp 1390-1393.

[4] L. Valient and G. J. Brebner, "Universal Schemes for Parallel Communication," *Proc.* 13^{th} *ACM Symposium on the Theory of Computation*, ACM, 1981, pp. 263-277.

[5] Q. F. Stout and B. Wager, "Intensive Hypercube Communication," *Journal of Parallel and Distributed Computing*, Vol. 10, pp. 167-181.

[6] Y. Saad and M. H. Schultz, "Data Communication in hypercubes," Res. Rep. YALEU/DCS/RR-428, Yale University, Oct. 1985 (revised Aug. 1987).

[7] D. P. Bertsekas, C. Ozveren, G. D. Stamoulis, P. Tseng, and J. N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 11, pp. 263-275, 1991.

[8] M. S. Chen and K. G. Shin, "Adaptive Fault-Tolerent Routing in Hypercube Multicomputers," *IEEE Trans. on Computers*, Vol. C-39, pp. 1406-1416, 1990.

[9] H. P. Katseff, "Incomplete Hypercube," *IEEE Trans. on Computers*, Vol. C-37, pp. 604-608, 1988.

[10] O. A. Olukotun and T. N. Mudge, "Hierarchical Gate-Array Routing on a Hypercube Multiprocessor," *Journal of Parallel and Distributed Computing*, Vol. 8, pp. 313-324, 1990.

[11] Y. Lan, A. H. Esfahnian and L. M. Ni, "Multicast in Hypercube Multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 8, pp. 30-41, 1990.

[12] C. T. Ho and S. L. Johnsson, "Spanning Balanced Trees in Boolean Cubes," *SIAM Journal Sci. Stat. Compt.*, Vol. 10, No. 4, July 1989, pp. 607-630.

[13] S. C. Patel and S. G. Ziavras, "Comparative Analysis of Techniques that Map Hierarchical Structures onto Hypercubes," *Parallel and Distributed Computing and Systems Conference*, Oct. 8-11, 1991, Washington, D.C.

[14] T. F. Chan and Y. Saad, "Multigrid algorithms on the hypercube multiprocessor," *IEEE Trans. Comput.*, C-35, No. 11 (Nov. 1986), pp. 969-977.

[15] Y. E. Ma and L. Tao, "Embeddings among toruses and meshes," *Proc. 1987 Internat. Conf. on Parallel Processing*, Aug. 1987, pp. 178-187.

[16] S. L. Johnsson and C. T. Ho, "On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two," *Proc. 1987 Internat. Conf. on Parallel Processing*, Aug. 1987, pp. 188-191.

[17] A. Y. Wu, "Embedding of tree networks into hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 2, No. 3 (Aug. 1985), 238-249.

[18] M. Y. Chan and S. J. Lee, "Distributed Fault-Tolerant Embeddings of Rings in Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 11, No. 1, pp. 63-71, 1991.

[19] M. Y. Chan, "Embedding of d-dimensional grids into optimal hypercubes," *Proc. ACM Symposium on Parallel Algorithms and Architectures*, 1989.

[20] S. N. Bhatt and I. F. Ipsen, "How to Embed Trees in Hypercubes," *Tech. Rep. YALEU/CSD/RR-443*, Dept. of Computer Science, Yale Univ., New Haven, CT. December 1985.

[21] T. H. Lai and W. White, "Embedding Pyramids in Hypercubes," *Tech. Rep.*, Dept. of Computer and Information Science, Ohio State Univ., November 1987.

[22] S. N. Bhatt, F. Chung, T. Leighton, and A. Rosenberg, "Optimal simulation of tree machines," *Proc.* 27^{th} *Annu. Symp. Foundations Comp. Sci.*, Oct. 1986, pp. 274-282.

[23] Q. F. Stout, "Sorting, merging, selecting, and filtering on tree and pyramid machines," *Proc. 1983 Internat. Conf. on Parallel Processing*, 1983, pp. 214-221.

[24] R. Miller and Q. F. Stout, "Data movement techniques for the pyramid computer," *SIAM J. Comput.*, Vol. 16, No. 1 (Feb. 1987), pp. 38-60.

[25] J. H. Chang, O. H. Ibarra, T. C. Pong, and S. M. Sohn, "Two-dimensional convolution on a pyramid computer," *Proc. 1987 Internat. Conf. on Parallel Processing*, Aug. 1987, pp. 780-782.

APPENDIX

Simulation of Equibalancing Algorithm

```
# include "stdio.h"

/* EQUIBALANCING ALGORITHM :
   Purpose of this program is to find communication time in a hypercube
   when all of its nodes are communicating with each other.The scheduling
   descipline is such that every node will transmit a message to the node
   having largest hamming distance,after transmitting all the messages to
   node, it will transmit messages to the node having smaller hamming distance,
   and so on..The next node will be selected such that all nodes have almost
   equal no. of messages to be transmitted ( equibalance ) i.e.,the node with
   minimum no. of messages ( to be transmitted ) will be selected as a next
   node.If more than one nodes have minimum no. of messages,one node will
   be selected randomly.

    ix is the seed for random number generator,
    n is the dimension of the hypercube,
   a[] is the array used to store addresses of next nodes on hamming path
       with given source and destination,
   src[i].dest[j] : is the destination address for node i,
   src[i].message[j] : is the no. of messages to be transmitted from node i
                       to the destination address src[i].dest[j],
   src[i].mtotal : is the total no. of messages to be transmitted from node i,
 /

ong int ix ;
nt n ,a[130] ;

truct sdm  {
              int  dest[130] ;
              int  message[130] ;
              int  mtotal ;
            } ;


* Function rannum() generates pseudo-random number uniformly distributed
  between 0-1 */

loat rannum()
 ix = ( ix*32949 ) + 8237 ;
 if ( ix < 0 ) ix = ( ix + 2147483647 ) + 1 ;
 return ( ix * 0.4656613e-9 );


* Function exp(x) calculates  2**x  */

nt exp(x)
nt x ;
 int p1 = 1,11;
 for ( 11 = 1 ; 11 <= x ; ++11 )
     p1 = p1 * 2 ;
 return (p1);


* Function hamdis ( sr,dest ) calculates hamming distance between
  source sr and destination dest  */

nt hamdis ( sr,dest )
nt sr,dest ;
  int i1,m,y,hamdist = 0 ;

  for ( i1 = n ; i1 >= 1 ; --i1 )
    { m = exp ( i1-1 ) ;
      y = m & ( sr ^ dest ) ;
      if ( y == m )
          hamdist += 1 ;
    }
```

```
   return ( hamdist ) ;

 }

/* Function search ( s,d ) finds a set of next intermediate nodes towards
   destination on hamming paths & then randomly selects one */

void search (s,d)
int s,d ;
{  int i1,j1,k1,h1,y,z ;
   k1 = 0 ;
   for ( i1 = n ; i1 >= 1 ; --i1 )
     { h1 = exp( i1-1 );
       y = h1 & ( s ^ d );
       z = h1 & s;
       if ( y == h1 )
         { k1 += 1;
           if ( h1 == z )
             a[k1-1] = ((exp(n) - 1 ) & ( s & (~h1)));
           else
             a[k1-1] = s | h1 ;
         }
     }
 }

main()

{
  int i,p,j,N,k,l,t,com_time = 0,q,na,r,v,m[130],aa[130] ;
  int temp,temp1,temp2 ;
  float rn ;
  struct sdm src[130] ;

  scanf ( "%d%d",&ix,&n ) ;

  N = exp(n) ;

  for ( l = 0 ; l < N ; ++l )
     { for ( v = 0 ; v < N-1 ; ++v )
          scanf ( "%d%d",&src[l].dest[v],&src[l].message[v] ) ;
     }


  for ( l = 0 ; l < N ; ++l )
     { src[l].mtotal = 0 ;
       for ( v = 0 ; v < N-1 ; ++v )
          src[l].mtotal = src[l].mtotal + src[l].message[v] ;
     }


b3: for ( i = 0 ; i < N ; ++i )
    {    j = n ;
      br: k = 0 ;
      br1: temp1 = src[i].dest[k] ;
           if ( j != hamdis ( i,temp1 ) )
               {       k += 1 ;
                       if ( k <= N-2 )
                            goto br1 ;
                       j -= 1 ;
                       if ( j > 0 )
                            goto br ;
                       continue ;
               }

       if ( src[i].message[k] == 0 )
```

```
        {  k += 1 ;
           if ( k <= N-2 )
              goto br1 ;
           j -= 1 ;
           if ( j > 0 )
                goto br ;
           continue ;
        }

    temp2 = src[i].dest[k] ;
    search ( i,temp2 ) ;

    for ( l = 0 ; l < j ; ++l )
        { t = a[l] ;
          m[l] = src[t].mtotal ;
        }

/* Following segment finds address of the next node having minimum
   no. of messages to be transmitted and stores that address in a[0] */

    for ( l = 1 ; l < j ; ++l )
       { if ( m[0] > m[l] )
            { temp = m[0] ;
              m[0] = m[l] ;
              m[l] = temp ;

              temp = a[0] ;
              a[0] = a[l] ;
              a[l] = temp ;
            }
        }

/* Following segment finds the set of addresses having this minimum
   no. of messages and stores in array aa[]  */

    p = 0 ;
    aa[0] = a[0] ;
    for ( l = 1 ; l < j ; ++l )
        { if ( m[0] == m[l] )
               { p += 1 ;
                 aa[p] = a[l] ;
               }
        }

/* Following segment selects the next address randomly from the set of
   next addresses ( aa[] ) and stores it in na   */


    rn = rannum() ;
    for ( q = 0 ; q <= p ; ++q )
      { if (( rn >= (float) q / (p+1)) && ( rn <= (float) (q+1) / (p+1)))
           na = aa[q] ;
      }

/* Following segment transmits  a message to the next node selected  */

    if ( j == 1 )
        { src[i].message[k] -= 1 ;
          src[i].mtotal -= 1 ;
        }
    else
        { src[i].message[k] -= 1 ;
          src[i].mtotal -= 1 ;

          for ( r = 0 ; r < N-1 ; ++r )
              { if ( src[i].dest[k] == src[na].dest[r] )
```

```
                    {  src[na].message[r] += 1 ;
                       goto br2 ;
                    }
                }
        }

    br2: src[na].mtotal += 1 ;
  }

/* Following segment checks if all the messages are transmitted or not;
   if not, it repeats the cycle and increments the communication time
   parameter  */

  for ( v = 0 ; v < N ; ++v )
     { for ( l = 0 ; l < N-1 ; ++l )
          if ( src[v].message[l] != 0 )
             { com_time += 1 ;
               goto b3 ;
             }
      }

  com_time += 1 ;

printf ( " Communication time is %d \n ",com_time ) ;
```

Simulation of One-Step Time-Lookahead Equibalancing Algorithm

```
# include "stdio.h"

/* ONE-STEP TIME-LOOKAHEAD ALGORITHM :

   Purpose of this program is to find communication time in a hypercube
   when all of its nodes are communicating with each other.The scheduling
   descipline is such that every node will transmit a message to the node
   having largest hamming distance,after transmitting all the messages to
   node, it will transmit messages to the node having smaller hamming distance,
   and so on..The next node will be selected such that all nodes have almost
   equal no. of messages to be transmitted ( equibalance ) i.e.,the node with
   minimum no. of messages ( to be transmitted ) and is going
   to receive min. no. of messages in the next cycle ,will be
   selected as a next
   node.If more than one nodes have minimum no. of messages,
   one node will
   be selected randomly.

    ix is the seed for random number generator,
    n is the dimension of the hypercube,
   a[] is the array used to store addresses of next nodes on hamming path
       with given source and destination,
   src[i].dest[j] : is the destination address for node i,
   src[i].message[j] : is the no. of messages to be transmitted from node i
                       to the destination address src[i].dest[j],
   src[i].mtotal : is the total no. of messages to be transmitted from node i,
*/

long int ix ;
int n ,a[130] ,aaa[100];

struct sdm  {
               int  dest[130] ;
               int  message[130] ;
               int  mtotal ;
             } ;


/* Function rannum() generates pseudo-random number uniformly distributed
   between 0-1 */

float rannum()
{ ix = ( ix*32949 ) + 8237 ;
  if ( ix < 0 ) ix = ( ix + 2147483647 ) + 1 ;
  return ( ix * 0.4656613e-9 );
}

/* Function exp(x) calculates  2**x  */

int exp(x)
int x ;
{ int p1 = 1,l1;
  for ( l1 = 1 ; l1 <= x ; ++l1 )
      p1 = p1 * 2 ;
  return (p1);
}

/* Function hamdis ( sr,dest ) calculates hamming distance between
   source sr and destination dest  */

int hamdis ( sr,dest )
int sr,dest ;
{  int i1,m,y,hamdist = 0 ;

   for ( i1 = n ; i1 >= 1 ; --i1 )
     { m = exp ( i1-1 ) ;
```

```
        y = m & ( sr ^ dest ) ;
        if ( y == m )
            hamdist += 1 ;
      }

    return ( hamdist ) ;

  }

  void next (s1)
  int s1 ;
  { int z, g,h = 0 ;

    for ( z = n ; z >= 1 ; --z )
        { g = exp( z-1 ) ;
          if ( g == ( g & s1 ) )
              aaa[h] = s1 & (~g) ;
          else
             aaa[h] = s1 | g ;
          h += 1 ;
        }
  }


/* Function search ( s,d ) finds a set of next intermediate nodes towards
   destination on hamming paths  */

void search (s,d)
int s,d ;
{  int i1,j1,k1,h1,y,z ;
   k1 = 0 ;
   for ( i1 = n ; i1 >= 1 ; --i1 )
     { h1 = exp( i1-1 );
       y = h1 & ( s ^ d );
       z = h1 & s;
       if ( y == h1 )
         { k1 += 1;
           if ( h1 == z )
             a[k1-1] = ((exp(n) - 1 ) & ( s & (~h1)));
           else
             a[k1-1] = s | h1 ;
         }
     }
  }

  int ns ( )
  { float rn1 ;
    int vv  ;

    rn1 = rannum() ;
    vv =  (int) ( exp(n) * rn1 ) ;

    return (vv) ;
  }

  int nom ( x1,y1 )
  int x1, y1 ;
  { float rn2 ;
    int uu ;

    rn2 = rannum() ;
    uu = (int) ( x1 + ( y1 - x1 ) * rn2 ) ;

    return (uu) ;
  }
```

```
main()

{
  int i,p,j,N,k,l,t,com_time ,q,na,r,v,m[130],aa[130] ,ss,qq ,b[100] ;
  int temp,temp1,temp2 ,add,tt,l1,l2,d,c[100],tempp ,temp11 ,rr ;
  int lnom1 , lnom2 , nd ,l1 ,q1 ,nm , sum_diff = 0 , total_messages = 0 ;
  int nsenders  , avg_diff , avg_message , ee , con ;
  float rn , psenders , pnod ,ratio = 0. ,step ;
  struct sdm src[130],src1[130] ;
  FILE *in, *out;

  in = fopen ( "data52_out","r" ) ;
  out = fopen ( "data1","w" ) ;

 /* printf ( "Enter values for ix,n,lnom1,lnom2,psenders,pnods,step \n" ) ;  */
 /* scanf ( "%d%d%d%d%f%f%f",&ix,&n,&lnom1,&lnom2,&psenders,&pnod,&step ) ;  */
  scanf ( "%d%d%d%f",&ix,&n,&con,&step ) ;

  N = exp(n) ;

  for ( l = 0 ; l < N ; ++l )
     { for ( v = 0 ; v < N-1 ; ++v )
          { if ( v < l )
               src[l].dest[v] = v ;
            else
               src[l].dest[v] = v + 1 ;
          }
     }

  for ( l = 0 ; l < N ; ++l )
     { for ( v = 0  ; v < N - 1 ; ++v )
          src[l].message[v] =  con ;
     }

 /* nsenders = psenders * N ;    */

 /* for ( l = 0 ; l < nsenders ; ++l )
{      nd = nod ( lnod1,l1nod2 ) ;
nd = pnod * N ;
ee = ns() ;
    printf ( " nsenders = %d nd = %d ee = %d \n ",nsenders,nd,ee ) ;
for ( l1 = 0 ; l1 < nd ; ++l1 )
   { rn = rannum() ;
     for ( q1 = 0 ; q1 <= N-2 ; ++q1 )
      { nm = nom ( lnom1,lnom2 ) ;
        if (( rn >= ( float ) q1 / ( N - 1 ) ) &&
            ( rn <= ( float ) ( q1 + 1 ) / ( N  - 1 )))
               src[ee].message[q1] = nm ;
      }
   }
}
*/
printf ( "%d %d \n ",ix,n ) ;
/* fprintf ( out," lnos1 = %f lnos2 = %f \n ",lnos1,lnos2 ) ;
printf ( " lnod1 = %f lnod2 = %f \n ",lnod1,lnod2 ) ;
fprintf ( out," lnom1 = %d lnom2 = %d \n ",lnom1,lnom2 ) ;
*/
for ( l = 0 ; l < N ; ++l )
{ for ( v = 0 ; v < N-1 ; ++v )
   printf ( " %d %d \n ",src[l].dest[v],src[l].message[v] ) ;

}


 /* for ( l = 0 ; l < N ; ++l )
```

```
        printf ( " %d %d \n ",src[l].dest[v],src[l].message[v] ) ;
    }  */

 for ( l= 0 ; l < N ; ++l )
    { for ( v = 0 ; v < N-1 ; ++v )
        { src1[l].dest[v] = src[l].dest[v] ;
          src1[l].message[v] = src[l].message[v] ;
        }
    }

 for ( l = 0 ; l < N ; ++l )
   { src[l].mtotal = 0 ;
     for ( v = 0 ; v < N-1 ; ++v )
        src[l].mtotal = src[l].mtotal + src[l].message[v] ;
   }

/* for ( l = 0 ; l < N ; ++l )
    total_messages = total_messages + src[l].mtotal ; */

/* avg_message = (float) total_messages / nsenders ; */


 while ( ratio <= 1. )
{
   com_time = 0 ;

   for ( l = 0 ; l < N ; ++l )
      { for ( v = 0 ; v < N-1 ; ++v )
          { src[l].dest[v] = src1[l].dest[v] ;
            src[l].message[v] = src1[l].message[v] ;
          }
      }

   for ( l = 0 ; l < N ; ++l )
       { src[l].mtotal = 0 ;
         for ( v = 0 ; v < N-1 ; ++v )
            src[l].mtotal = src[l].mtotal + src[l].message[v] ;
       }

   ratio = ratio + step ;

3: for ( i = 0 ; i < N ; ++i )
   {    j = n ;
     br: k = 0 ;
     br1: temp1 = src[i].dest[k] ;
          if ( j != hamdis ( i,temp1 ) )
              {       k += 1 ;
                      if ( k <= N-2 )
                           goto br1 ;
                      j -= 1 ;
                      if ( j > 0 )
                           goto br ;
                      continue ;
              }

      if ( src[i].message[k] == 0 )
          {  k += 1 ;
             if ( k <= N-2 )
                goto br1 ;
             j -= 1 ;
             if ( j > 0 )
                  goto br ;
             continue ;
          }
```

```
      sum_diff = sum_diff + j ;
      temp2 = src[i].dest[k] ;
      search ( i,temp2 ) ;

      for ( l = 0 ; l < j ; ++l )
          c[l] = a[l] ;
/* Following segment finds all the neighbours of the next node
   and then selects rr neighbours randomly  and stores them in
   array b[]                  */
     for ( l = 0 ; l < j ; ++l )
       {  next ( c[l] ) ;

    /*      rr = (int) (ratio * n) ;
          ss = 0 ;
          while ( ss < rr )
            {  br5: rn = rannum() ;

               for ( qq = 0 ; qq <= n-1 ; ++qq )
                    { if (( rn >= qq / ( float ) n ) && ( rn <= ( qq + 1 )
                            / ( float ) n ))
                          { if ( aaa[qq] == i )
                               goto br5 ;
                            b[ss] = aaa[qq] ;
                          }
                    }
               ss += 1 ;
            }          */

 /* Following segment checks if any of selected neighbours
    have messages to transmit to destinations whose hamming
    path passes through the next node; if so,the no. of messages
    which are going to be transmitted are incremented..    */

          add = 0 ;

          for ( l1 = 0 ; l1 < n ; ++l1 )
             { tt = aaa[l1] ;
               if ( tt == i )
                  continue ;
               for ( v = 0 ; v < N-1 ; ++v )
                  {
                    if ( src[tt].message[v] == 0 )
                       continue ;
                    templ1 = src[tt].dest[v] ;
                    d = hamdis ( tt,templ1 ) ;
                    if ( d == 1 )
                       continue ;

                    search ( tt,templ1 ) ;

                    for ( l2 = 0 ; l2 < d ; ++l2 )
                        { if ( a[l2] == c[l] )
                             { add += 1 ;
                               goto br7 ;
                             }
                        }
                  }
                br7: continue ;
             }

    /*      printf ( " add = %d \n ",add ) ;    */
          tempp = c[l] ;
          rr = (int) ( ratio * add ) ;
          m[l] = src[tempp].mtotal +  rr ;
       /*   printf ( "m[%d] = %d rr = %d \n ",l,m[l],rr ) ;    */
```

```
/* Following segment finds address of the next node having minimum
   no. of messages to be transmitted and stores that address in c[0] */

    for ( l = 1 ; l < j ; ++l )
       { if ( m[0] > m[l] )
            { temp = m[0] ;
              m[0] = m[l] ;
              m[l] = temp ;

              temp = c[0] ;
              c[0] = c[l] ;
              c[l] = temp ;
            }
       }

/* Following segment finds the set of addresses having this minimum
   no. of messages and stores in array aa[]  */

    p = 0 ;
    aa[0] = c[0] ;
    for ( l = 1 ; l < j ; ++l )
        { if ( m[0] == m[l] )
               { p += 1 ;
                 aa[p] = c[l] ;
               }
        }

/* Following segment selects the next address randomly from the set of
   next addresses ( aa[] ) and stores it in na   */


    rn = rannum() ;
    for ( q = 0 ; q <= p ; ++q )
      { if (( rn >= (float) q / (p+1)) && ( rn <= (float) (q+1) / (p+1)))
           na = aa[q] ;
      }

/* Following segment transmits  a message to the next node selected  */

    if ( j == 1 )
        { src[i].message[k] -= 1 ;
          src[i].mtotal -= 1 ;
        }
    else
        { src[i].message[k] -= 1 ;
          src[i].mtotal -= 1 ;

          for ( r = 0 ; r < N-1 ; ++r )
               { if ( src[i].dest[k] == src[na].dest[r] )
                    {  src[na].message[r] += 1 ;
                       goto br2 ;
                    }
               }
        }

    br2: src[na].mtotal += 1 ;
  }

/* Following segment checks if all the messages are transmitted or not;
   if not, it repeats the cycle and increments the communication time
   parameter  */

  for ( v = 0 ; v < N ; ++v )
     { for ( l = 0 ; l < N-1 ; ++l )
```

```
            if ( bit[v].message[i] != 0 )
               { com_time += 1 ;
                 goto b3 ;
               }
        }

 /*  avg_diff = ( float ) sum_diff / total_messages ;  */

   com_time += 1 ;
   printf ( " ratio = %f  com_time = %d \n ",ratio,com_time ) ;

}
 /*  printf ( " total_messages = %d \n ", total_messages ) ;
   printf ( " avg_message = %d \n ", avg_message ) ;
   printf ( " psenders = %f \n ", psenders ) ;
   printf ( " avg_diff = %d \n ", avg_diff ) ;
   printf ( " nsenders = %d \n ", nsenders ) ;  */
```

Simulation of Johnsson's Algorithm

```
include "stdio.h"

* JOHNSSON'S ALGORITHM:
  Purpose of this program is to find communication time in a hypercube
  when all of its nodes are communicating with each other.
  The scheduling discipline follows Johnsson's Reverse Breadth First
  scheduling.
/
* ix is the seed for random number generator,
   n is the dimension of the hypercube,
  a[] is the array used to store addresses of next nodes on hamming path
      with given source and destination,
  src[i].dest[j] : is the destination address for node i,
  src[i].message[j] : is the no. of messages to be transmitted from node i
                      to the destination address src[i].dest[j],
  src[i].mtotal : is the total no. of messages to be transmitted from node i,
/

ong int ix ;
nt n ,a[130] ;

truct sdm  {
             int  dest[130] ;
             int  message[130] ;
           } ;


* Function rannum() generates pseudo-random number uniformly distributed
  between 0-1 */

loat rannum()
 ix = ( ix*32949 ) + 8237 ;
 if ( ix < 0 ) ix = ( ix + 2147483647 ) + 1 ;
 return ( ix * 0.4656613e-9 );


* Function exp(x) calculates  2**x  */

nt exp(x)
nt x ;
 int p1 = 1,l1;
 for ( l1 = 1 ; l1 <= x ; ++l1 )
     p1 = p1 * 2 ;
 return (p1);


* Function hamdis ( sr,dest ) calculates hamming distance between
  source sr and destination dest  */

nt hamdis ( sr,dest )
nt sr,dest ;
  int i1,m,y,hamdist = 0 ;

  for ( i1 = n ; i1 >= 1 ; --i1 )
    { m = exp ( i1-1 ) ;
      y = m & ( sr ^ dest ) ;
      if ( y == m )
          hamdist += 1 ;
    }

  return ( hamdist ) ;

 }

/* Function search ( s,d ) finds a set of next intermediate nodes towards
   destination on hamming paths & then randomly selects one */
```

```
void search (s,d)
int s,d ;
{  int i1,j1,k1,h1,y,z ;
   k1 = 0 ;
   for ( i1 = n ; i1 >= 1 ; --i1 )
     { h1 = exp( i1-1 );
       y = h1 & ( s ^ d );
       z = h1 & s;
       if ( y == h1 )
         { k1 += 1;
           if ( h1 == z )
             a[k1-1] = ((exp(n) - 1 ) & ( s & (~h1)));
           else
             a[k1-1] = s | h1 ;
         }
     }
}

main()

{
  int i,j,N,k,l,com_time = 0,q,na,r,v ;
  int temp,temp1 ;
  float rn ;
  struct sdm src[130] ;
/*  FILE *in, *out;

  in = fopen ( "data1_in","r" ) ;
  out = fopen ( "data7_out","w" ) ;

  fscanf ( in, "%d%d",&ix,&n ) ;  */
  scanf ( "%d%d", &ix,&n ) ;

  N = exp(n) ;

  for ( l = 0 ; l < N ; ++l )
     { for ( v = 0 ; v < N-1 ; ++v )

           scanf ( "%d%d",&src[l].dest[v],&src[l].message[v] ) ;
     }

/* for ( l = 0 ; l < N ; ++l )
     { for ( v = 0 ; v < N-1 ; ++v )
          printf ( " src[%d].dest[%d] = %d src[%d].message[%d] = %d
        \n ",l,v,src[l].dest[v],l,v,src[l].message[v] ) ;

     }  */


  br: for ( j = n ; j >= 1 ; --j )
        { for ( i = 0 ; i < N ; ++i )
             { k = 0 ;
               br1: temp = src[i].dest[k] ;
               if ( j != hamdis ( i,temp ) )
                       { k += 1 ;
                         if ( k <= N-2 )
                            goto br1 ;
                         continue ;
                       }
                    if ( src[i].message[k] == 0 )
                        { k += 1 ;
                          if ( k <= N-2 )
                             goto br1 ;
                          continue ;
                        }
```

```
                    templ = src[i].dest[k] ;
        /*            printf ( " k = %d i = %d  templ = %d \n ", k,i,templ ) ; *
,

                    search ( i, templ ) ;

                    rn = rannum () ;

                    for ( q = 0 ; q <= j-1 ;  ++q )
                        { if (( rn >=  q / ( float ) j ) && ( rn <= ( q+1 ) /
                                ( float ) j ))
                            na = a[q] ;
                        }
                  /*   printf ( " na = %d \n ", na ) ;  */

                    if ( j == 1 )
                         src[i].message[k] -= 1 ;
                    else
                       { src[i].message[k] -= 1 ;

                         for ( r = 0 ; r < N-1 ; ++r )
                            { if ( src[i].dest[k] == src[na].dest[r] )
                                 { src[na].message[r] += 1 ;
                                   break ;
                                 }
                            }

                       }
              }
              com_time += 1 ;
           /*  printf ( " com_time = %d \n ", com_time ) ;  */
        }


   for ( l = 0 ; l < N ; ++l )
      { for ( v = 0 ; v < N-1 ; ++v )
           {  if ( src[l].message[v] != 0 )
                 goto br ;
           }
      }
/*   for ( l = 0 ; l < N ; ++l )
     { for ( v = 0 ; v < N-1 ; ++v )
 printf ( " src[%d].message[%d] = %d \n ", l,v,src[l].message[v] ) ;
     }  */

   printf ( " Communication time is %d \n ",com_time ) ;
```

Simulation of Brute-Force Algorithm

```
include "stdio.h"

* BRUTE-FORCE ALGORITHM :
  Purpose of this program is to find communication time in a hypercube
  when all of its nodes are communicating with each other.The scheduling
  descipline is such that every node will transmit a message to the node
  having largest hamming distance,after transmitting all the messages to
  node, it will transmit messages to the node having smaller hamming distance,
  and so on..The next node will be selected at random.

   ix is the seed for random number generator,
   n is the dimension of the hypercube,
  a[] is the array used to store addresses of next nodes on hamming path
      with given source and destination,
  src[i].dest[j] : is the destination address for node i,
  src[i].message[j] : is the no. of messages to be transmitted from node i
                      to the destination address src[i].dest[j],
  src[i].mtotal : is the total no. of messages to be transmitted from node i,
/

ong int ix ;
nt n ,a[130] ;

truct sdm  {
              int  dest[130] ;
              int  message[130] ;
              int  mtotal ;
            } ;


* Function rannum() generates pseudo-random number uniformly distributed
  between 0-1 */

loat rannum()
 ix = ( ix*32949 ) + 8237 ;
 if ( ix < 0 ) ix = ( ix + 2147483647 ) + 1 ;
 return ( ix * 0.4656613e-9 );


* Function exp(x) calculates  2**x  */

nt exp(x)
nt x ;
 int p1 = 1,11;
 for ( 11 = 1 ; 11 <= x ; ++11 )
     p1 = p1 * 2 ;
 return (p1);


* Function hamdis ( sr,dest ) calculates hamming distance between
  source sr and destination dest  */

nt hamdis ( sr,dest )
nt sr,dest ;
  int i1,m,y,hamdist = 0 ;

  for ( i1 = n ; i1 >= 1 ; --i1 )
    { m = exp ( i1-1 ) ;
      y = m & ( sr ^ dest ) ;
      if ( y == m )
          hamdist += 1 ;
    }

  return ( hamdist ) ;

}
```

```
* Function search ( s,d ) finds a set of next intermediate nodes towards
  destination on hamming paths & then randomly selects one */

oid search (s,d)
nt s,d ;
  int i1,j1,k1,h1,y,z ;
  k1 = 0 ;
  for ( i1 = n ; i1 >= 1 ; --i1 )
    { h1 = exp( i1-1 );
      y = h1 & ( s ^ d );
      z = h1 & s;
      if ( y == h1 )
        { k1 += 1;
          if ( h1 == z )
            a[k1-1] = ((exp(n) - 1 ) & ( s & (~h1)));
          else
            a[k1-1] = s | h1 ;
        }
    }
}

ain()


 int i,p,j,N,k,l,t,com_time = 0,q,na,r,v ;
 int temp,temp1,temp2 ;
 float rn ;
 struct sdm src[130] ;

 scanf ( "%d%d",&ix,&n ) ;

 N = exp(n) ;

 for ( l = 0 ; l < N ; ++l )
    { for ( v = 0 ; v < N-1 ; ++v )
          scanf ( "%d%d",&src[l].dest[v],&src[l].message[v] ) ;
    }


3: for ( i = 0 ; i < N ; ++i )
   {    j = n ;
     br: k = 0 ;
     br1: temp1 = src[i].dest[k] ;
          if ( j != hamdis ( i,temp1 ) )
              {       k += 1 ;
                      if ( k <= N-2 )
                           goto br1 ;
                      j -= 1 ;
                      if ( j > 0 )
                           goto br ;
                      continue ;
              }

      if ( src[i].message[k] == 0 )
          {  k += 1 ;
             if ( k <= N-2 )
                goto br1 ;
             j -= 1 ;
             if ( j > 0 )
                  goto br ;
             continue ;
          }

      temp2 = src[i].dest[k] ;
      search ( i,temp2 ) ;
```

```
'* Following segment selects next intermediate at random */

        rn = rannum() ;
        for ( q = 0 ; q < j ; ++q )
          { if (( rn >= (float) q / j ) && ( rn <= (float) (q+1) / j))
                na = a[q] ;
          }

   /* Following segment transmits  a message to the next node selected  */

        if ( j == 1 )
             src[i].message[k] -= 1 ;
        else
            { src[i].message[k] -= 1 ;

              for ( r = 0 ; r < N-1 ; ++r )
                  { if ( src[i].dest[k] == src[na].dest[r] )
                       {  src[na].message[r] += 1 ;
                       }
                  }
            }

     }

   /* Following segment checks if all the messages are transmitted or not;
      if not, it repeats the cycle and increments the communication time
      parameter  */

     for ( v = 0 ; v < N ; ++v )
        { for ( l = 0 ; l < N-1 ; ++l )
             if ( src[v].message[l] != 0 )
                { com_time += 1 ;
                  goto b3 ;
                }
         }

     com_time += 1 ;

    printf ( " Communication time is %d \n ",com_time ) ;

}
```