5-31-1992

Processor allocation strategies for modified hypercubes

Nagasimha G. Haravu
New Jersey Institute of Technology

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

ABSTRACT

Processor Allocation Strategies for Modified Hypercubes

by

Nagasimha G. Haravu

Parallel processing has been widely accepted to be the future in high speed computing. Among the various parallel architectures proposed/implemented, the hypercube has shown a lot of promise because of its poweful properties, like regular topology, fault tolerance, low diameter, simple routing, and ability to efficiently emulate other architectures. The major drawback of the hypercube network is that it can not be expanded in practice because the number of communication ports for each processor grows as the logarithm of the total number of processors in the system. Therefore, once a hypercube supercomputer of a certain dimensionality has been built, any future expansions can be accomplished only by replacing the VLSI chips. This is an undesirable feature and a lot of work has been under progress to eliminate this stymie, thus providing a platform for easier expansion.

Modified hypercubes (MHs) have been proposed as the building blocks of hypercube-based systems supporting incremental growth techniques without introducing extra resources for individual hypercubes.

However, processor allocation on MHs proves to be a challenge due to a slight deviation in their topology from that of the standard hypercube network. This thesis addresses the issue of processor allocation on MHs and proposes various strategies which are based, partially or entirely, on *table look-up* approaches. A study of the various task allocation strategies for standard hypercubes is conducted and their suitability for MHs is evaluated. It is shown that the proposed strategies have a perfect subcube recognition ability and a superior performance. Existing

processor allocation strategies for pure hypercube networks are demonstrated to be ineffective for MHs, in the light of their inability to recognize all available subcubes. A comparative analysis that involves the buddy strategy and the new strategies is carried out using simulation results.

Processor Allocation Strategies for Modified Hypercubes

by

Nagasimha G. Haravu

A Thesis Submitted to the Faculty of
New Jersey Institute of Technology in Partial Fulfillment of
the Requirements for the Degree of
Master of Science
Department of Electrical and Computer Engineering
May 1992

APPROVAL SHEET

Processor Allocation Strategies for Modified Hypercubes

by

Nagasimha G. Haravu

5/4/95

Dr. Sotirios Ziavras, Thesis Advisor
Assistant Professor of Electrical and Computer Engineering,
New Jersey Institute of Technology

5/4/92

Dr. John Carpinelli, Committee Member
Assistant Professor of Electrical and Computer Engineering,
New Jersey Institute of Technology

5/4/92

Dr. Edwin Hou, Committee Member
Assistant Professor of Electrical and Computer Engineering,
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Nagasimha G. Haravu

Degree: Master of Science in Electrical Engineering

Date: May, 1992

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering, New Jersey Institute of Technology, Newark, NJ, 1992
- Bachelor of Engineering in Electronics Engineering, Bangalore University, Bangalore, India, 1990

Major: Electrical and Computer Engineering

This thesis is dedicated to

my parents

ACKNOWLEDGEMENT

The author would like to express his sincere gratitudes to his advisor Prof. Sotirios Ziavras for his invaluable advice and encouragement. The author would like to thank him also for the support and excellent opportunities provided by him during the entire course of research.

Special thanks to Professors John Carpinelli and Edwin Hou for serving as members of the committee.

This work has benefited from substantial outside support provided by the National Science Foundation under Grant No. CCR-9109084 and this is gratefully appreciated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Importance Of Parallel Processing Techniques

Parallel computers have proven themselves very useful in solving many engineering and scientific problems. The main advantage of using a parallel computer over the sequential one is that different parts of the user program can be executed at the same time, allowing a reduction in the execution time of the program [14]. Although earlier supercomputers like the IBM 3081/3084, CRAY-2 and Burroughs D-825 achieved their high performance by increasing the raw speed of the electronic components and logic circuits, further increase in their speed is limited by the speed of light. Parallelism at the processor level has been shown to effectively overcome the above bottleneck [30]. As a result of evolutionary advancements in computation and communication technology, supercomputers consisting of thousands of processors are already available. In addition to providing enhanced availability, recongifurability and resource sharing, these massively parallel systems can theoretically multiply the computational power of a single processor by a large factor. The key advantage of such systems is that they allow concurrent execution of tasks which can be independent programs or partitioned modules of a single program.

The extremely powerful parallel machine will still fail to perform at its full strength unless the communication overhead, which has been proving to be the

major pitfall, is largely reduced. Research in this area has produced breakthroughs in the form of optical fiber technology. High speed optically interconnected parallel machines may become commercially available in the near future.

The areas in which parallel processing can be effectively used in scientific and engineering research is very vast. They include Aerodynamics, Astrophysics, Biology, Computer Science, Chemistry, Geophysics, High Energy Physics, Material Science, Meteorology, Nuclear Physics and Plasma Physics. Industrial applications of parallel computers include the oil, the automobile and the pharmaceutical industries [15].

1.2 Networks For Parallel Computing Systems

One major way of classifying parallel computers is by their architecture or topology of the interconnection between the processing elements (PEs). Some of the possible choices are illustrated in Figure 1.1. An important class of parallel computers make use of *shared memory*. The simplest design of this class is shown in Figure 1.1(a) and uses a common bus, or communication channel, to allow the individual PEs to access the shared memory. This design is appropriate if N, the number of PEs, is small. However, since the effective bus communication time in this approach is proportional to N, the design becomes inadequate for large N.

When the two open ends of the bus are joined together, the resulting topology is called the ring topology. This shared memory network, illustrated in Figure 1.1(b), has the advantages of better compatibility with fiber optics communication channels and better maintainability.

A more sophisticated shared memory design is typified by Figure 1.1(c) and involves a dynamic switch connecting the processor units to the shared memory.

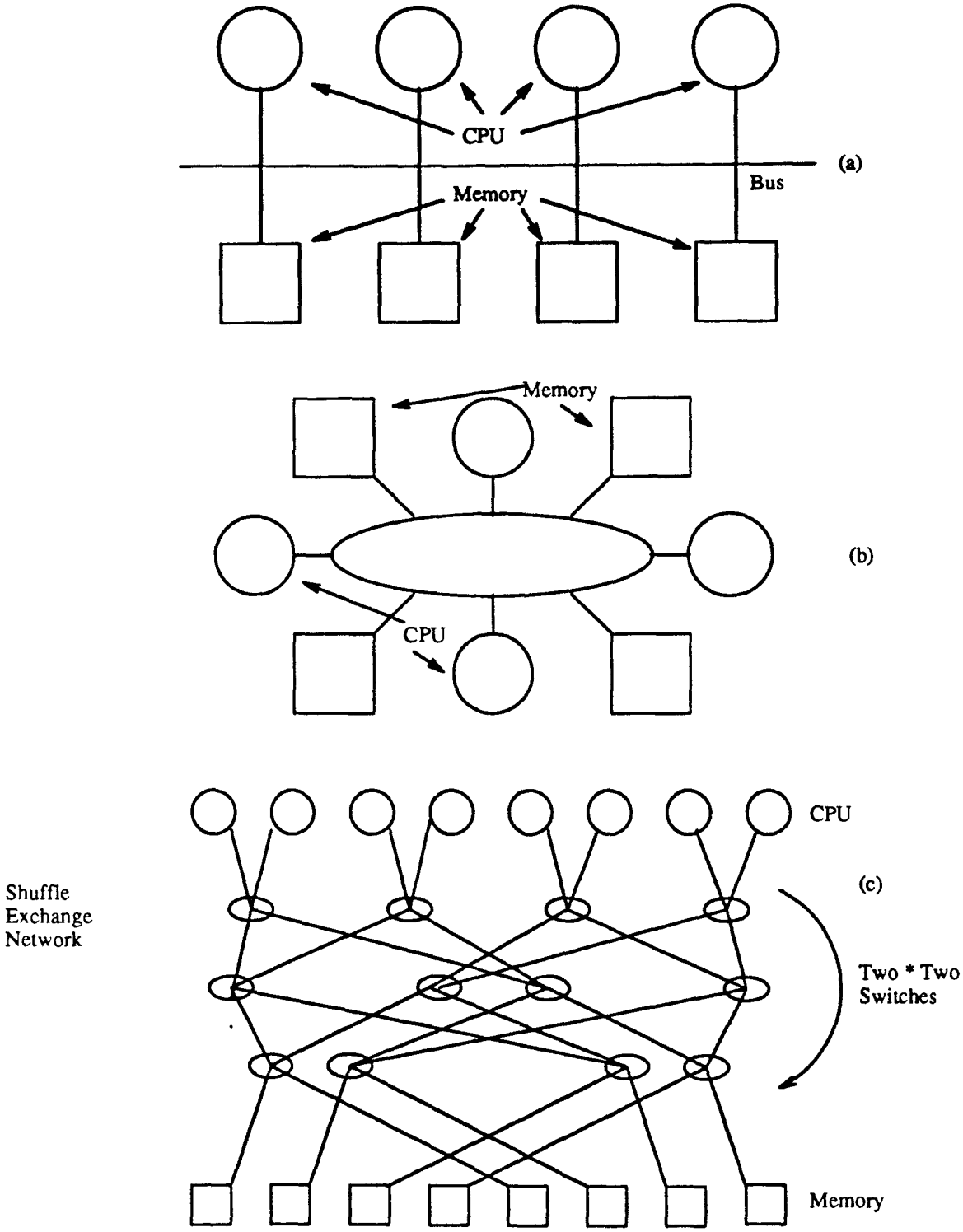
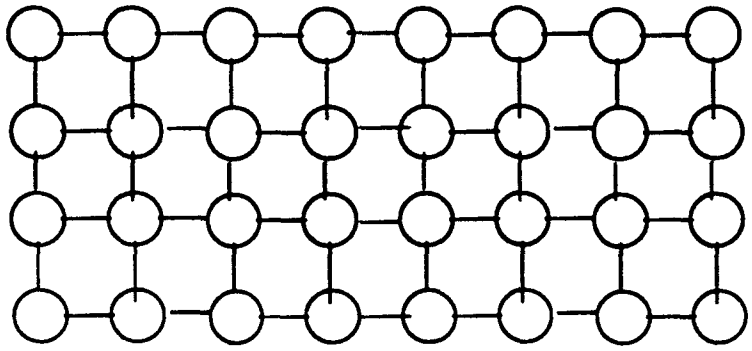
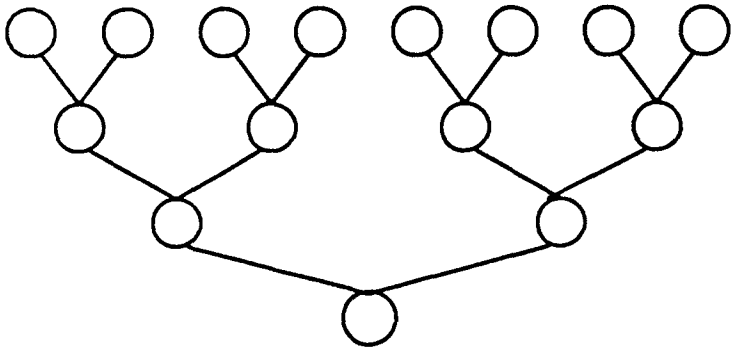


Figure 1.1: Examples for parallel computing systems topology. (a) A linear array; (b) the ring topology; (c) a dynamic switch; (d) a 2-d mesh; (e) the binary tree; (f) the hypercube.

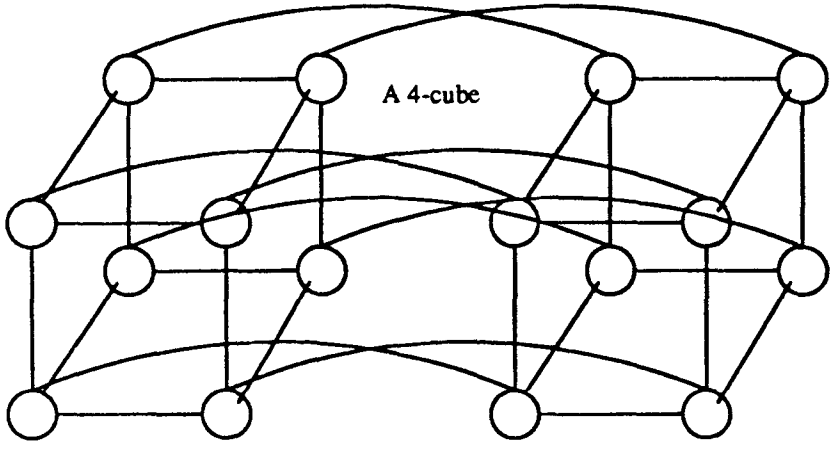


(d)



(e)

Node including Local Memory



(f)

In Figures 1.1(d)-(f), we illustrate a different class of machines characterized by a *distributed memory.*

In pure distributed memory machines, the basic PE includes local memory to the exclusion of shared or global memory. The two-dimensional mesh and the binary tree are a few examples wherein the number of channels needed per node is independent of N. The hypercube, on the other hand, requires that the number of channels per node grow logarithmically with N. The two-dimensional mesh is a popular architecture for image processing applications, whereas, the tree topology is most useful in managing huge data bases. The hypercube has proved to be the most popular among all the above architectures and will be discussed at greater depth at a later section.

1.3 Motivations and Objectives

A large number of papers have dealt with hypercubes and some commercial hypercube-based systems have successfully been introduced. Such systems are the Intel iPSC, the nCUBE, the Connection Machine, etc. The most important reason for the undisputable success of the hypercube network has been its inherent capability of supporting the interconnection of large number of resources with small resultant diameters. The *diameter* of a network is defined as the maximum of the shortest distances between pairs of nodes. The low diameter of the hypercube network, which is equal to n for the n-dimensional hypercube, is the direct result of its high degree of interconnectivity and its highly regular structure [33].

These advantages of the hypercube network, along with its inherent fault-tolerance capabilities (due to its highly-interconnected structure), make it very flexible for the emulation of other frequently used networks. In fact, dozens of algorithms have been proposed for embedding several important networks into the

hypercube. For example, the problem of embedding rectangular meshes has been addressed, among others, by Chan and Saad [7], and Johnsson [20]. Algorithms for embedding trees have been proposed, among others, by Wu [32], Deshpande and Jenevin [10], Ho and Johnsson[18], and Johnsson[20]. Finally, algorithms for embedding pyramids have been designed, among others, by Chan and Saad [7], and Lai and White [24].

Nevertheless, systems that contain a pure hypercube network have two major drawbacks: (1) They always contain a number of processors which is a power of two. (2) The numbers of communication ports and links per processor increase as the logarithm of the total number of processors in the system. Although many hypercube variations (discussed in a later section) have been proposed in the literature, there does not currently exist any solution to the problem of interconnecting multiple hypercubes, of not necessarily the same number of dimensions, at the lowest possible cost (i.e., without the introduction of any extra resources). In contrast, Ziavras [33] has proposed a methodology that slightly modifies hypercubes in order to support their incremental growth without introducing or wasting any resources for individual hypercubes. Systems comprising clusters of slightly *modified hypercubes (MHs)* are called *multicube systems.* The proposed methodology allows the construction of systems that satisfy the following four vital goals. (1) The basic building blocks of multicube systems are slightly modified hypercubes. (2) The total number of processors in a multicube system is not necessarily a power of two. (3) The numbers of communication ports and links per processor do not grow logarithmically with the total number of processors in multicube systems. (4) Finally, all the available resources in multicube systems could be fully utilized.

Processor allocation is a crucial factor in determining the performance of a parallel computer. Many problems in engineering and science can be formulated in

terms of directed and undirected graphs. Various algorithms have been reported in the literature for mapping such graphs onto regular interconnection topologies such as rings, trees, meshes, pyramids, etc. Most of these topologies can easily be embedded into the hypercube [27]. Given a task graph, the processor allocation problem for the hypercube can be formulated as continuous requests for subcubes of arbitrary size and residence time. The objective then becomes the allocation of appropriate subcubes in the parallel machine so that certain performance measures are maximized or minimized. The problem of processor allocation is in contrast to the task scheduling problem wherein there is a continuous request of tasks to be scheduled on the hypercube and the scheduler decides on the number of processors to be allocated to the requesting tasks, depending on the parallelism in the problem; the scheduler may also migrate some of the tasks for efficient load balancing. Further details on scheduling algorithms for hypercubes can be found in [2] and [7]. The various processor allocation strategies proposed for standard hypercubes fail when applied to MHs because of the slight difference between the topologies of standard hypercubes and MHs.

1.3.1 Problem Statement

The chief objective of this research is to propose processor allocation techniques for the family of MHs and to determine the performance of the techniques by simulation methods.

1.3.2 Research Contributions

A summary of all the contributions of this research follows:

- Study of the existing allocation techniques on standard hypercubes and determining their suitability for MHs.

- Identification of all possible subcubes of all possible dimensions in a given MH.
- Proposing a processor allocation strategy much suited for the MH; this strategy is based on a "table look-up" approach.
- Modifying the table look-up technique to incorporate some features of the "free list" strategy, an effective allocation strategy for standard hypercubes.
- Proposing a parallel version of the table look-up technique in order to reduce the time complexity and achieve even higher resource utilization.
- Finally, simulation of the table look-up technique and the buddy strategy, another strategy proposed for the standard hypercube, and compare their performances when implemented for MHs.

CHAPTER 2

HYPERCUBE-BASED NETWORKS

2.1 Pure Hypercube Networks

The hypercube or Boolean N-cube is a network configuration of 2^N processors such that each processor has exactly N neighbors. The positive integer N is the order of the cube. a three dimensional cube, for example, is a third order Boolean cube. Each node. representing a processing element, in the latter case has three adjacent nodes and three processors that are one node away. The twelve edges of the cube represent the direct communication links between pairs of nodes. Simultaneous communication between several pairs of nodes can occur.

Higher order cubes are more difficult to visualize. Figure 2.1 shows a fourth-order cube which can be described as a cube inside a larger cube with corresponding corner nodes connected. Messages sent between non-neighboring nodes are passed from node to node until they reach their destination. The routing path can be configured easily by successively inverting one bit of the source address until it exactly matches the destination address. For example, to route data from 0101 node to 1010 node, it can be passed through nodes 0100, 0110, and 0010. For efficient routing, the Hamming distance between the routing nodes and the target node must always decrease by 1.

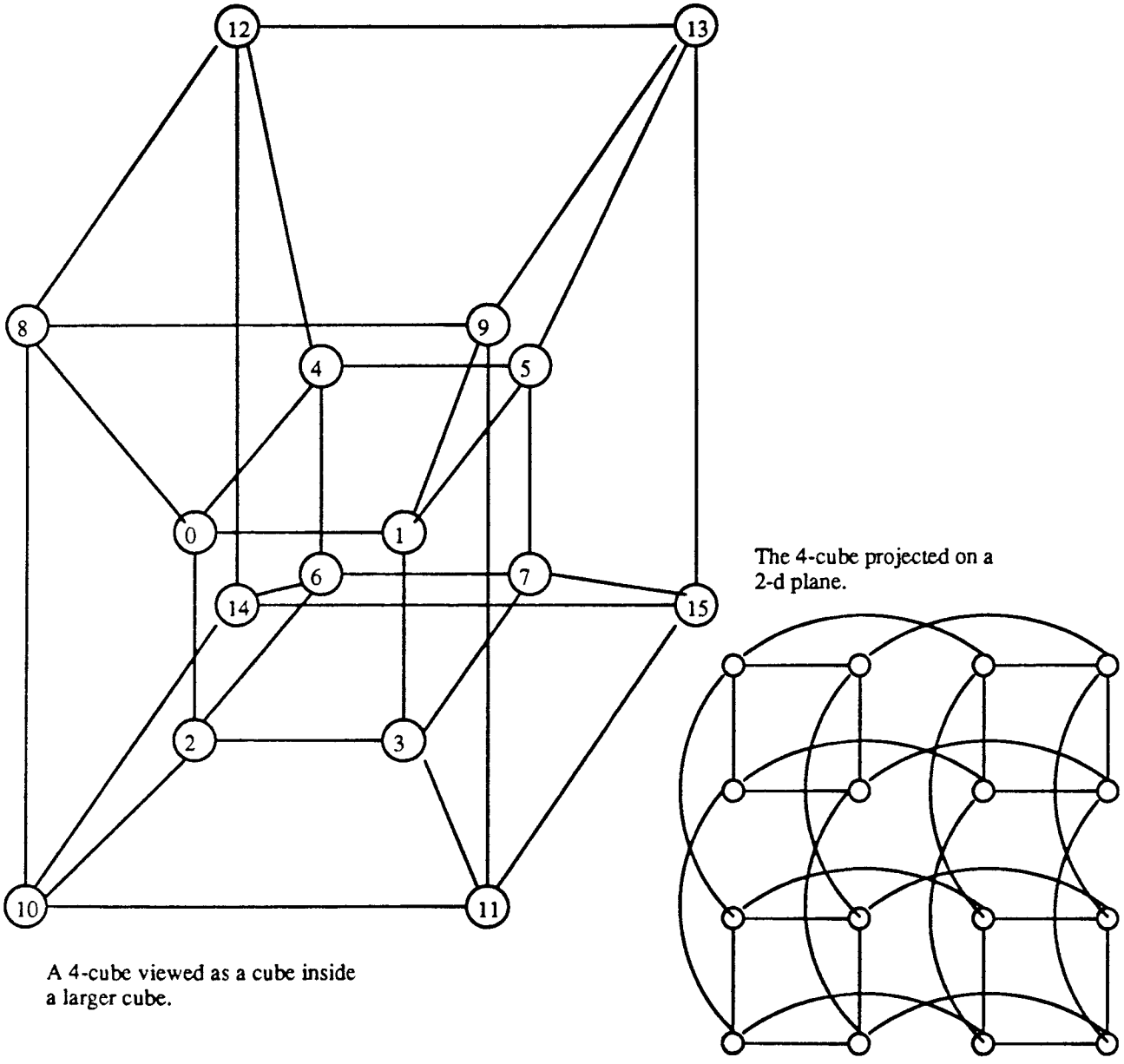


Figure 2.1: A four dimensional hypercube.

For an n-processor network, the farthest node is only $\log_2 n$ away. For every pair of nodes there are $(log_2 n)!$ possible routes; hence if a particular node is busy, the data can be rerouted using another path. Because of this, the system will never deadlock - a condition when all channels are blocked. The hypercube is also fault tolerant, i.e., if some channels or nodes are disabled, the cube can still operate in a degraded mode [30]. Other important properties of the hypercube network are (1) simple routing, (2) high degree of fault tolerance, and (3) ability to efficiently map other architectures onto the hypercube.

2.2 Hypercube-Like Networks

Several variations of the hypercube network have been proposed in the past. Although they may improve one or more topological properties of the hypercube, they do not support ease of expansion for hypercube systems. A brief discussion of the most important hypercube variants follows.

The twisted cube [1] needs the same amount of resources as the standard hypercube. It is constructed by repositioning some of the links in the hypercube so that they span two dimensions. The advantage of the twisted cube is that its diameter is only $\lceil (n+1)/2 \rceil$ for a network with 2^n nodes. The obtained asymmetric network affects the dynamic performance of the system, so that the improvement in performance over the hypercube is not nearly as much as the reduction in diameter. Another twisted cube variant repositions two links within a single 4-cycle [13]. For a network composed of 2^n nodes, the diameter is $n - 1$. Also, enhanced incomplete hypercubes [21] can be constructed from incomplete hypercubes by directly connecting pairs of PEs with extra links attached to otherwise unused ports. The extra links decrease the diameter of the network and the average distance between pairs of nodes. A folded hypercube [4] is constructed from a standard hypercube by

establishing for each node direct connection with the unique node that is farthest from it. The folded hypercube may perform better than the corresponding hypercube because of its smaller diameter, which is $\lceil n/2 \rceil$ for a network containing 2^n nodes, better average distance, and less message traffic density. However, its major drawback, when compared to the hypercube, is the increased number of communication ports and links. More specifically, a folded hypercube with 2^n nodes has 2^n extra communication ports and 2^{n-1} extra links when compared to the corresponding n-dimensional hypercube. Some other variations, for which it could safely be stated that they are farther away from the hypercube than the above variations, are: the cube-connected cycles [25], the cubical ring connected cycles [5], the block-shuffled hypercube [19], hyper-rectangulars [22], the generalized folding cube [9], the spanning bus hypercube [31], the generalized supercube [28], etc.

In another variant, the incomplete hypercube [21, 29], a large number of communication ports could be wasted and as a consequence a large portion of a system's cost should be spent for unused resources. For example, an incomplete hypercube with 1280 processors could be constructed from two complete hypercubes composed of 1024 and 256 processors respectively. However, in order to implement the interconnection between the two complete hypercubes, the number of communication ports per processor will be 11 and 9 respectively for the two constituent hypercubes (this is in contrast to 10 and 8 for the corresponding standard hypercubes). The total number of unused communication ports in this system will be equal to 768 (i.e., $1024 - 256$), assuming that all the extra links of the hypercube of size $2^{n_0} + 2^{n_1}$ are implemented as the interconnection of two complete hypercubes with n_0 and n_1 dimensions respectively, then the lower limit on the total number of unused communication ports will be equal to $|2^{n_0} - 2^{n_1}|$.

In a similar fashion, the basic building block of hierarchical cubic networks

(HCNs) [16] is the hypercube. However, HCNs use a smaller number of links per node and their diameter is smaller than that of corresponding hypercubes. An HCN that contains 2^{m+n} nodes, where $n \geq m$, has a diameter which is smaller than or equal to $m + n$; $m + n$ is the diameter of the hypercube with the same number of nodes. This HCN will employ $n + 1$ links per node. Similarly to incomplete hypercubes, HCNs can not be viewed as interconnections of hypercubes because they assume the existence of extra hardware for individual processors (more specifically, they need one extra communication port per processor when compared to the corresponding hypercubes).

Since the present work is based on the modified hypercube network, a deeper insight into it is pertinent.

2.3 Modified Hypercubes

2.3.1 The Structure of Modified Hypercubes

The constituent hypercubes of multicube systems should not generally be connected to hosts and/or peripheral devices in a way similar to the one used for intra-hypercube communications. This is also true for commercial hypercubes. For example, each processor in the Intel iPSC/1 system contains I/O hardware and implements the Ethernet protocol [26]. Since the entire system has a single host that serves as the I/O handler, the I/O bandwidth of this system is relatively low. The I/O structure of the Intel iPSC/2 system is much more effective because it uses multiple disks and data declustering techniques. In the nCUBE system, subcubes of eight processors are assigned a single I/O processor and the I/O processors are interconnected partially. In the Connection Machine CM2 system with 64K processors, as many as 2K processors can send or receive data at a time [17]. Transfers of data are performed by I/O controllers which interface through an I/O channel

to CM2 data lines. These I/O controllers operate under the control of up to four sequencers. Two controllers may be active simultaneously on each sequencer.

In order to construct a multicube system that uses identical technologies and protocols for the implementation of all inter-processor communications, some of the intra-hypercube communication links of individual hypercubes should be cut and used for inter-hypercube communications [33]. Thus, the existing I/O structure of the constituent hypercubes will not be modified. In fact, we should expect that the already existing I/O structure of individual hypercubes will perform very well for the resultant multicube system, since it is designed to perform very well for the constituent hypercubes. The term *I/O PEs* (processing elements, which are composed of a processor and local memory) is used in [33] to denote PEs in the system which are used for inter-hypercube communications.

The larger the number of I/O PEs and the more uniform their distribution in the system, the higher the utilization of multicube PEs in general, because of the fast implementation of inter-hypercube communications. For example, if a hypercube has two I/O PEs, these PEs should be diametrically opposite (i.e., their binary addresses should complement one another). In a hypercube with four I/O PEs, two pairs of diametrically opposite PEs should be chosen for I/O. Figure 2.2 shows the embedding of four I/O PEs in a 4-cube. PEs which are not used for I/O are adjacent to I/O PEs in this figure. Reddy and Banerjee [26] have used the term *perfect I/O embedding* for the case where every PE in the system is adjacent to exactly one I/O PE (however, they have instead studied pure I/O in standard hypercubes). They have shown that a perfect embedding exists in the n-dimensional hypercube if and only if $n = 2^l - 1$, for some integer l.

For reasons of uniformity, the total number of I/O PEs in MHs is assumed to be a power of two. Hypercubes are slightly modified in order to embed an I/O

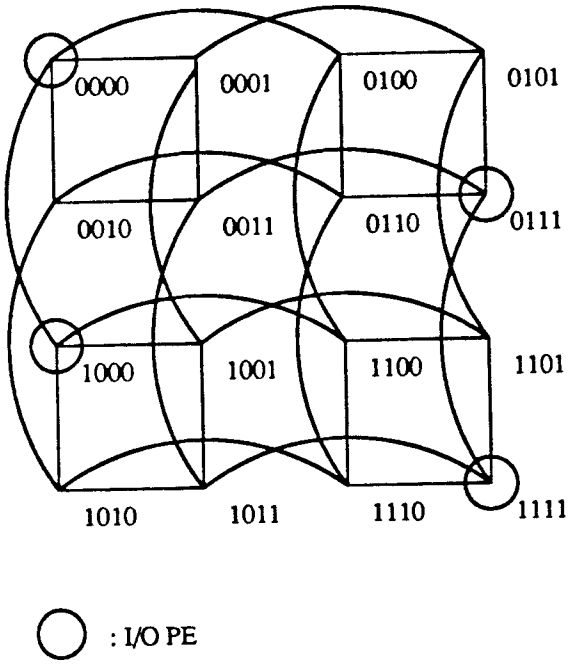


Figure 2.2: I/O PEs in a 4-cube [33].

structure (for support of inter-hypercube communications), without the introduction of any type of extra resources. In addition, the resultant MHs are capable of fully utilizing all their resources. In the following discussion, $H(n, nil)$ denotes a standard hypercube with n dimensions (nil stands for 0 I/O PEs), while $H(n, \lambda)$ denotes the modified form of $H(n, nil)$ that contains 2^λ I/O PEs. The procedure to be followed for the choice of I/O PEs is as follows. IF $\alpha = 2^{\beta+1}$, where $\beta = n - \lambda$, then the decimal addresses of the I/O PEs in the transformed system $H(n, \lambda)$ will be $\Gamma_\gamma = \gamma\alpha$ and $\Gamma_{\zeta+\gamma} = (2^n - 1) - \Gamma_\gamma$, where $0 \leq \gamma \leq (2^{\lambda-1} - 1)$ and $\zeta = 2^{\lambda-1}$. An important property of the MH is listed below.

The I/O PEs in $H(n, \lambda)$ form $2^{\lambda-1}$ pairs of diametrically opposite PEs in $H(n, nil)$. Since the structure of individual PEs can not be modified during the construction of multicube systems, one link of each I/O PE is cut and used for inter-hypercube communications. From the n candidate links to be cut for each I/O PE, the link that corresponds to the lowest dimension is broken (i.e., direct connection will not exist any more with the PE having binary address that differs from the I/O PE's address only in the LSB). However, such a technique leaves one communication link unused for any non-I/O PE which is attached to a broken link. All the broken links are then interconnected in pairs, so that all the resources (including communication ports) become available for use in multicube systems. More specifically, pairs of broken links attached to diametrically opposite non-I/O PEs are directly connected. Such pairs can be found because of the following two reasons. First, the addresses of the non-I/O PEs which are attached to previously broken links are $C(\Gamma_j)$, for $0 \leq j \leq (2^\lambda - 1)$, where $C(\Gamma_j)$ complements the LSB in the binary representation of Γ_j. Second, it can be shown that the I/O PEs are diametrically opposite in pairs [33]. Figure 2.3 shows the I/O PEs and the repositioning of previously broken links (which are shown by dashed lines) for MHs

$H(4, \lambda)$, with $1 \leq \lambda \leq 4$. Figure 2.3(d) contains a non-connected graph which is composed of two 3-cubes. A graph is connected if and only if there exists at least one path between any pair of nodes.

The other major structural properties of MHs are:

- A modified hypercube $H(n, \lambda)$ is not connected if and only if $n = \lambda$.

Therefore, the case of $n = \lambda$ is not studied any further.

- The I/O PEs in $H(n, \lambda)$ form two disjoint $(\lambda - 1)$-dimensional hypercubes.
- The non-I/O PEs which are attached to repositioned links in $H(n, \lambda)$ form a λ-dimensional hypercube.

The last two properties show that despite the modifications carried out on the original hypercube, the hypercube is the dominant topology in $H(n, \lambda)$.

Various structures which are efficiently embedded into the hypercube can still efficiently be mapped onto the modified hypercube structure because the only difference between the original structure and its modified form is present in connections corresponding to the lowest dimension. Since the maximum distance between any pair of PEs whose addresses differ in the LSB is now equal to 3, an MH $H(n, \lambda)$ can very efficiently emulate the corresponding standard hypercube $H(n, nil)$.

2.3.2 Topological Properties of Modified Hypercubes

A through investigation of the capabilities of MHs was included in [33]. This subsection summarizes those properties. The *total node connectivity* of $H(n, \lambda)$ is n. The *intra-hypercube node connectivity* for I/O nodes is $n - 1$, while it is n, for all other nodes.

The diameter of a multicube system is a measure of its suitability to yield high performance. The smaller the value of the diameter, the faster the exchange of values between distant PEs in the multicube system. Of course, a parameter that heavily influences the diameter of a multicube system is the *distance of non-I/O*

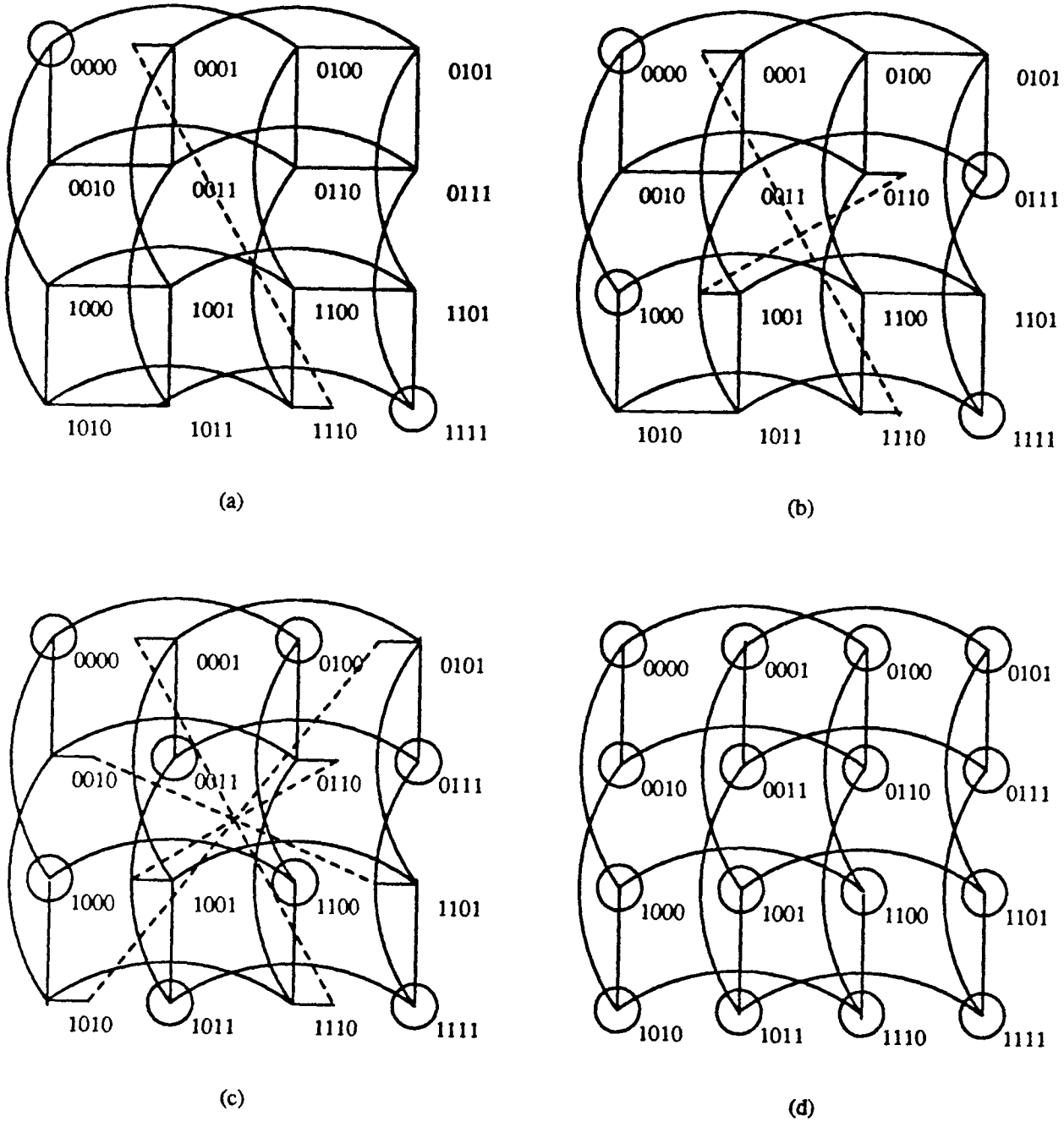


Figure 2.3: Modified Hypercubes H(4,λ). (a) $\lambda = 1$; (b) $\lambda = 2$; (c) $\lambda = 3$; (d) $\lambda = 4$ [33].

PEs from I/O PEs which serve as the gateways for data transfers between MHs. More properties follow.

• In a modified hypercube $H(n, \lambda)$, the maximum $\Delta_{I/O}$ of the shortest distances from I/O PEs is

$$\Delta_{I/O} = \begin{cases} \lceil \beta/2 \rceil & \text{if } \beta > 4 \text{ or } \beta = 1 \\ 3 & \text{if } 3 \leq \beta \leq 4 \\ 2 & \text{if } \beta = 2 \end{cases}$$

The *average shortest distance from I/O PEs* is defined here to be the ratio of the sum of the shortest distances of all PEs from I/O PEs over the total number of PEs (including the I/O PEs which may also carry out computation tasks). A small value for this measure could aid the task of uniformly distributing the workload of inter-dependent tasks among the constituent MHs of a multicube system, with the result being high utilization of PEs and high performance.

• The average shortest distance from I/O PEs in $H(n, \lambda)$ is equal to

$$2^{-\beta} \left(\sum_{\epsilon=1}^{\lceil \beta/2 \rceil} \epsilon \binom{\beta}{\epsilon} + \sum_{\epsilon=0}^{\beta - \lceil \beta/2 \rceil - 1} (\epsilon + 1) \binom{\beta}{\epsilon} + u_\beta \right), \quad \text{where} \quad u_\beta = \begin{cases} 2 & \text{if } \beta \geq 3 \\ 1 & \text{if } \beta = 2 \end{cases}$$

For $\beta = 1$, the value of this measure is equal to 0.5. $\binom{\beta}{\epsilon}$ is the number of distinct ϵ-combinations of β items.

Table 2.1 shows the average shortest distance from I/O PEs in $H(10, \lambda)$, where $1 \leq \lambda \leq 9$. To avoid the creation of communication bottlenecks, uniform utilization of the I/O PEs may be required. Therefore, sometimes PEs may communicate with PEs resident in other MHs of multicube systems via I/O PEs which are not at the shortest distance. Thus, it is also important to know the distance from all I/O PEs in MHs.

• For a PE with binary address $s_{n-1}s_{n-2}\ldots s_0$, which is not attached to any repositioned link in a modified hypercube $H(n, \lambda)$, the distances from the 2^λ I/O

Table 2.1: The average shortest distance from I/O PEs in H(10,λ), where $1 \leq \lambda \leq 9$ [33].

λ	No. of I/O PEs	% of I/O PEs	β	Ave. Shortest Dist. from I/O PEs
1	2	0.195	9	3.773
2	4	0.391	8	3.227
3	8	0.781	7	2.922
4	16	1.536	6	2.438
5	32	3.125	5	2.125
6	64	6.250	4	1.688
7	128	12.500	3	1.500
8	256	25.000	2	1.000
9	512	50.000	1	0.500

PEs are: $\epsilon + j_1$ for $\binom{\lambda-1}{j_1}$ of them, where ϵ is the total number of bits s_l, for $0 \leq l < \beta$, which are equal to the binary complement s'_β of s_{β_1} and $0 \leq j_1 \leq (\lambda-1)$; finally, $\beta - \epsilon + j_2 + 1$ for $\binom{\lambda-1}{j_2}$ of them, where $0 \leq j_2 \leq (\lambda-1)$. For a PE which is attached to a repositioned link, the value of 2 is added to the distances corresponding to I/O PEs with a value of s'_0 for the LSB of the binary address.

Table 2.2 contains the distances from the I/O PEs in $H(4,2)$.

- The *diameter* Δ of $H(n,\lambda)$ is $\Delta = \begin{cases} \min\{\Delta_{I/O} + \lceil n/2 \rceil, n\} & \text{if } \beta > 1 \\ n+1 & \text{if } \beta = 1 \end{cases}$

The *average distance* of a regular network is defined as the ratio of the sum of distances of all its nodes from a given node (for the sake of simplicity, this may also include a zero distance for the PE that serves as the reference point) over the total number of nodes. The value of this measure for the n-dimensional hypercube is equal to $\frac{\sum_{j=1}^{n} j\binom{n}{j}}{2^n} = \frac{n}{2}$

Since MHs do not comprise perfectly regular topologies, the average distance of MHs is defined here as the sum of distances involving all possible pairs of nodes, divided by the total number of pairs. Table 2.3 shows the average distance of

Table 2.2: The distances from the I/O PEs in H(4,2) [33].

PE address	I/O PE address			
	0	7	8	15
0	0	3	1	4
1	3	2	3	3
2	1	2	2	3
3	2	1	3	2
4	1	2	2	3
5	2	1	3	2
6	2	3	3	3
7	3	0	4	1
8	1	4	0	3
9	3	3	3	2
10	2	3	1	2
11	3	2	2	1
12	2	3	1	2
13	3	2	2	1
14	3	3	2	3
15	4	1	3	0

some networks. If $\beta > 1$, then the average distance of $H(n, \lambda)$ is smaller that the average distance of $H(n, nil)$. This is true because of the existence not only of the repositioned links but also of paths composed of original hypercube links and repositioned links. If $\beta = 1$, then the average distance of $H(n, \lambda)$ is larger than the average distance of $H(n, nil)$ because the original hypercube is divided into two disjoint hypercubes interconnected with repositioned links, so some original hypercube paths are not present any more.

The reader is referred to [33] for more information on MHs and specifically on routing in MHs, and for several multicube examples.

Table 2.3: The average distance of some networks [33].

Network	Average Distance
H(4,nil)	2.000
H(4,1)	1.938
H(4,2)	1.938
H(4,3)	2.250
H(5,nil)	2.500
H(5,1)	2.453
H(5,2)	2.437
H(5,3)	2.748

CHAPTER 3

PROCESSOR ALLOCATION TECHNIQUES

3.1 Definitions And Notations

Let Σ be a ternary symbol set $\{0, 1, x\}$, where x represents the "don't care" digit. Since an n-cube uses n address bits, every subcube of an n-cube can be uniquely represented by a sequence of ternary symbols in Σ, which is called the address of the corresponding subcube. The Hamming distance between two subcube addresses is defined as follows [23].

Definition 1 (Hamming distance): The Hamming distance, $H : \Sigma^k \times \Sigma^k \rightarrow I^+$, between two address strings $A = a_{n-1} \ldots a_1 a_0$ and $B = b_{n-1} \ldots b_1 b_0$ in Σ^n for some integer n is defined as $H(A, B) = \sum_{i=0}^{n-1} h(a_i, b_i)$, where $h(a_i, b_i) = 1$ if [$a_i = 0$ and $b_i = 1$] or [$a_i = 1$ and $b_i = 0$], and $h(a_i, b_i) = 0$ otherwise.

Similarly, the exact distance $E : \Sigma^k \times \Sigma^k \rightarrow I^+$, between A and B is defined as $E(A, B) = \sum_{i=1}^{k} e(a_i, b_i)$ where $e(a, b) = 0$ if $[a = b]$, and $e(a, b) = 1$ otherwise. □.

The definition can be extended to multiple address strings. In the extended definition, $h(a_i, b_i, c_i, \ldots) = 1$, if none of the ith bit is x and at least two ith bits differ in the bit value. Otherwise, $h(a_i, b_i, c_i, \ldots) = 0$. Similarly, $e(a_i, b_i, c_i, \ldots) = 0$ if all the ith bits have the same value, otherwise $e(a_i, b_i, c_i, \ldots) = 1$. For example,

let us take $A = 001x$ and $B = x001$. Then $H(A,B) = 1$ and $E(A,B) = 3$. Now if $C = 101x$, then $H(A,B,C) = 1$ and $E(A,B,C) = 3$.

Definition 2 (Gray Code generation): Let G_n be the GC (Gray Code) with parameters g_i, $1 \leq i \leq n$, where a sequence $\langle g_1, g_2, \ldots, g_n \rangle$ is a permutation of $Z_n = \{1, 2, \ldots, n\}$. Then, G_n is defined recursively as follows.

$G_0 = \{\}$,

$G_k = \{(G_{k-1})^{0\backslash r_k}, (G^*_{k-1})^{1\backslash r_k}\}, \qquad 1 \leq k \leq n$

where r_k is the partial ranking of g_k. □.

Let $\langle g_1, g_2, \ldots, g_n \rangle$ be a sequence of distinct integers. The *partial ranking* r_i of g_i for $1 \leq i \leq n$ is defined as the rank of g_i in the partial set $\{g_1, g_2, \ldots, g_n\}$ when the set is arranged in ascending order. Let G be a sequence of binary strings of length k denoted by $G^{b\backslash r_k}$, $b \in \{0,1\}$, that can be obtained by either inserting a bit b into the position immediately right of the r_kth bit from the right side of every string in G if $1 \leq r_k \leq k$, or by prefixing a bit b to every string in G if $r_k = k$. Also, let G^* denote a sequence of binary strings obtained from G by reversing the order of strings in G. The GC can be generated combining $(G_{k-1})^{0\backslash r_k}$ and $(G^*_{k-1})^{1\backslash r_k}$. As an example, let us consider the generation of 3-bit GC code with parameters $\langle 3, 2, 1 \rangle$. The parameters have the partial ranking, $\langle 1, 1, 2 \rangle$. Then $G_1 = \{0,1\}$, $G_2 = \{00, 10, 11, 01\}$, and $G_3 = \{000, 100, 101, 001, 011, 111, 110, 010\}$.

Definition 3 (Adjacent and Complementary Cubes): Two cubes A and B are adjacent if $H(A,B) = 1$. A complement of cube A, is defined as $A' = a_1, a_2, \ldots, a_{j-1}, b_j, a_{j+1}, \ldots, a_n$ with bit j having any position between 1 and n, $a_j, b_j, \in \{0,1\}$, and $a_j \neq b_j$. A complement cube is a special case of adjacent cube where the two cubes differ exactly in one bit position. □.

For example, a cube $\{0x1x\}$ can have a complement cube $\{1x1x\}$ or $\{0x0x\}$.

3.2 Processor Allocation Techniques For Standard Hypercubes

In this section, we describe briefly the buddy, GC, modified buddy and the free list strategies before introducing the table look-up strategy. This is because the table look-up strategy has also some similar features of those of the previous strategies.

3.2.1 The Buddy Strategy

The buddy strategy is one of the earliest strategies proposed for processor allocation on standard hypercube multiprocessors. We present this strategy here and the simulation results of the buddy strategy applied to MHs are presented in Chapter 4.

Since there exist 2^n nodes in an n-cube, 2^n allocation bits are used by this strategy in order to keep track of the availability of nodes. A 0(1) value in the allocation bit indicates the availability(unavailability) of the corresponding node [23].

Allocation:

Step 1: Set k equal to the dimension of a subcube required to accomodate the current request.

Step 2: Determine the least integer m such that all the allocation bits from $m2^k$ to $(m+1)2^k - 1$ are 0's. Set all these allocation bits to 1's.

Step 3: Allocate the nodes found in Step 2 to the current request.

Deallocation:

Step 1: For all values of p, where p is the address of a released node from the set of nodes included in the deallocated subcube, reset the pth allocation bit to 0.

3.2.2 The Gray Code Strategy

Similar to the buddy strategy, 2^n allocation bits are used to keep track of the availability of all nodes. But, the sequence of allocation bits follows the gray code pattern.

Allocation:

Step 1: Same as in Buddy.

Step 2: Determine the least integer m such that all the (i mod 2^n)th bits are 0's where $i \in \#[m2^{k-1}, (m+2)2^{k-1} - 1]$, and set all these 2^k allocation bits to 1's.

Step 3: Allocate nodes to the current request.

Deallocation:

Step 1: Same as in Buddy.

3.2.3 The Modified Buddy Strategy

Similar to the buddy strategy, 2^n allocation bits are used to keep track of the availability of all nodes. An integer A represented by m bits is regarded as free if $(A^m)^{0\backslash 1}$ and $(A^m)^{1\backslash 1}$ are free. For example, an integer three in 2 bits, i.e., 11, is free if integers six and seven in 3 bits are free. This notation implies the free subcubes of smaller dimension. Detailed description of the modified buddy strategy can be found in [3].

Allocation:

Step 1: Same as in Buddy.

Step 2: Determine the least integer $A, 0 \leq A \leq 2^{n-k+1}-1$, such that A^{n-k+1} is free, and it has a pth partner, $1 \leq p \leq (n-k+1), A_p^{n-k+1}$ which is also free. Take p as small as possible.

Step 3: Allocate nodes to the current request and set their allocation bits to 1.

Deallocation:

Step 1: Same as in Buddy.

3.2.4 The Free List Strategy

In the free list strategy, $n+1$ independent lists are maintained so that for every request of dimension k, where $0 \leq k \leq n$, there exists a respective list of available subcubes. Each element in the list is represented by a unique address, a sequence of n ternary symbols. An n-cube is represented as a sequence of n Xs initially. When there is a request for a k-cube, where $k \leq n$, the algorithm looks for a k-cube in the free list. If there is not any k-cube in the free list, one of the available nearest higher dimension subcubes is decomposed from the most significant bit side in order to identify a k-cube. The resulting k-cube is then assigned to the request.

Allocation:

Step 1: Same as in the buddy strategy.

Step 2: If there is an available subcube for the request size in the k-cube list, then allocate it and terminate the algorithm.

Step 3: Otherwise, find the nearest higher dimension cube that is available. If there is no higher dimension cube in the list, then keep the request in the waiting queue.

Step 4: Choose one of the nearest higher dimension cubes and decompose it into two subcubes. Repeat this step until the requested cube size is reached. Allocate the resulting k-cube.

Deallocation:

Step 1: Include the released k-cube in the corresponding k-cube list.

Step 2: Compare the released k-cube to all free cubes and form new cubes of higher dimension, if possible.

Step 3: Do the following for all dimensions, starting from the lowest dimension in the list:

a. Generate the new (overlapping) i-dimensional cubes, if possible, by combining the cubes of the same dimension i.

b. Generate the new (overlapping) i- or $(i+1)$-dimensional cubes, if possible, by combining the cubes of dimension i and the cubes of the nearest nonempty higher dimension(s).

c. Combine two complement subcubes of dimension i, if they exist, and form an $(i+1)$-dimensional cube.

Step 4: Make the cubes generated after Step 3 mutually disjoint to each other as follows.

a. Select a cube from the highest dimension list and decompose all other same or lower dimensional cubes that have a common node with the selected cube.

b. The cubes with a common node(s) with the selected cube are deleted from the list.

c. Repeat steps a and b for all the free cubes except the ones already selected.

3.2.5 Analysis of the Previous Strategies

Static allocation is concerned with the accommodation of the incoming requests without considering processor relinquishment. An allocation strategy is said to be statically optimal if an n-cube can accommodate any input request sequence $\{I_i\}_{i=1}^{m}$ iff $\sum_{i=1}^{m} 2^{|I_i|} \leq 2^n$, where $|I_i|$ is the dimension of a subcube required to accommodate request I_i. It has been proved that all the previous strategies are statically optimal [8, 3, 23].

In a dynamic environment, when processor relinquishment is taken into consideration, the buddy strategy is shown to be poor because it generates more external fragmentation. It also cannot detect all available subcubes in an n-cube. If the request is for a subcube of size k, the buddy strategy looks for only those cubes with Xs in the k least significant bits of their hypercube address. That is, it can recognize only $1/\binom{n}{k}$ percent of all possible subcubes. On a 6-cube this percentage for a 3-cube request becomes 5%. Nevertheless, the time complexity of the buddy strategy is $O(2^n)$. For example, let us examine the following sequence of requests and relinquishment in a 4-cube system.

Example-1: Sequencer

1) A request for a 2-cube (I_1),

2) A request for a 2-cube (I_2),

3) A request for a 2-cube (I_3),

4) A 2-cube relinquishment,

5) A request for a 3-cube.

The allocation for the buddy strategy is shown as a K-map in Figure 3.1(a). The last request for the 3-cube allocation can be accommodated immediately if the relinquished 2-cube is exactly the third 2-cube (I_3). Even though there is a 3-cube available when the second allocation (I_2) is relinquished, it cannot be detected using the buddy scheme. This is because the buddy strategy tries to allocate a 3-cube as shown in Figure 3.1(b). Allocation of 2-cubes using the GC strategy is shown in Figure 3.1(c). The request for a 3-cube (sequence 5) can be accommodated using the GC scheme if the relinquished 2-cube is either the first or the third one. Since the GC strategy has more subcube recognition ability, as shown in Figure 3.1(d), it can detect a 3-cube when the first 2-cube (I_1) is also relinquished. Allocation of 2-cubes with the modified buddy strategy is the same as in Figure 3.1(a), and the allocation strategy for the 3-cube request is the same as in Figure 3.1(d). It can detect a 3-cube when the second request (I_2) or the third request (I_3) is relinquished.

The external fragmentation caused by the GC strategy is even less than that of the buddy strategy, since there is a tendency that the nodes allocated first are also released first for uniformly distributed service time. For example, there is an external fragmentation if the first 2-cube (I_2) is relinquished in the buddy and the modified buddy strategies. Release of the second 2-cube (I_2) results in external fragmentation in the GC scheme. Since it is likely that I_1 is released earlier than I_2, the buddy strategy is more susceptible to external fragmentation.

The free list strategy gives a better system utilization and less delay compared to the other bit-map schemes. The multiple GC strategy is the only other scheme that gives close results to that of the free list strategy. However, both the serial and parallel versions of the multiple GC strategy are more complex than the corresponding free list versions. A slightly modified version of the free list strategy

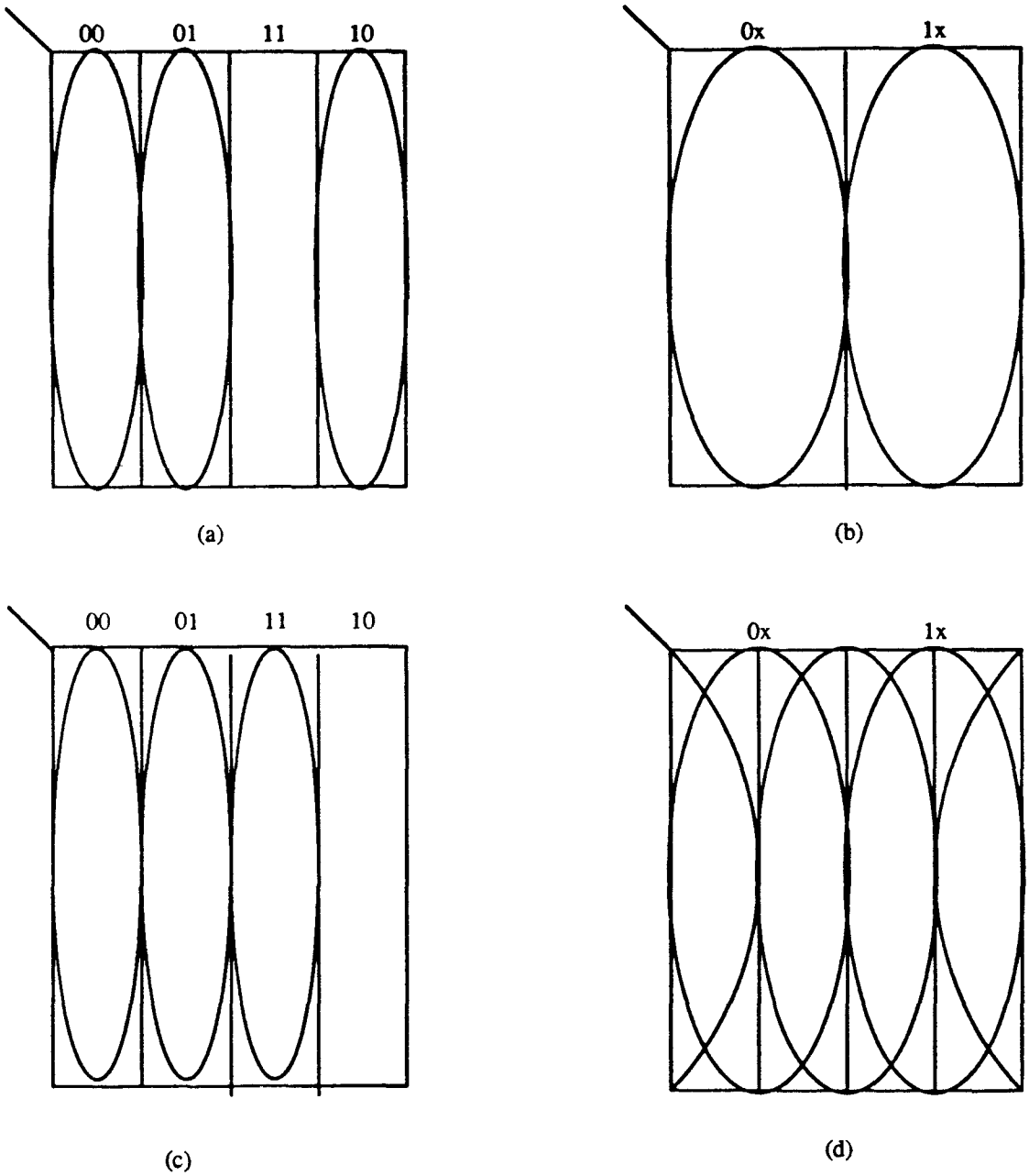


Figure 3.1: (a) An allocated 2-cube under Buddy scheme. (b) Buddy strategy location rule for 3-cube. (c) An allocated 2-cube under GC scheme. (d) GC strategy allocation rule for 3-cube.

gives the same performance at reduced complexity. Another advantage of the free list scheme is its fast allocation ability. It takes $O(n)$ time to allocate a k-cube. The allocation time complexity of all other schemes is $O(2^n)$. Hence, if a subcube is already available, the free list can allocate it very fast. But, the free list strategy has a deallocation time complexity of $O(n^3)$. It can easily be parallelized by running the algorithm on a parallel machine with $n + 1$ processors, where the processors operate on mutually disjoint lists of available subcubes.

Dutt and Hayes have also used the free list concept for hypercube allocation [10]. The difference between the two strategies in the allocation lies in the complexity and optimality. If there is a free cube in the list for the requested dimension k, both strategies assign the k-cube immediately. The two strategies differ when there is no free cube of exact dimension k, but there are free cube in the higher dimension list. The free list strategy selects the first free cube from the nearest higher dimension list for decomposition. In the other strategy, each free cube in the nearest higher dimension list is combined with free cubes of all lower dimensions to generate all possible k-cubes. Among the free cubes in the nearest higher dimension list, the one that generates the maximum number of k-dimension free cubes is selected for decomposition. Whereas the former technique is simple and quick the latter one is optimal and time consuming.

3.3 Processor Allocation Strategies For MHS

This section addresses the processor allocation problem for MHs. Since the problem of identifying subcubes of certain dimensions in MHs is more difficult than the corresponding problem in standard hypercubes, Subsection 3.3.1 addresses the former problem. Subsection 3.3.2 proposes a "table look-up" allocation/deallocation strategy which is based on the free list strategy for standard hypercubes. Subsec-

tion 3.3.3 presents a slightly modified version of this strategy which is more space efficient. Subsection 3.3.4 proposes a space efficient strategy which is based on the strategy of Subsection 3.3.2 and the buddy strategy. Finally, subsection 3.3.5 discusses a parallel version of the first strategy.

3.3.1 Identification of subcubes in MHs

In order to find the total number of k-cubes in an n-dimensional standard hypercube, one has to simply evaluate $\binom{n}{k} 2^{n-k}$. Further, the addresses of these subcubes can easily be obtained by generating all possible sequences of n-ternary symbols containing k X bits.

The ease with which the above procedure can be accomplished is severely affected when one or more of the hypercube links are removed, and more so when some links are repositioned as in MHs. As seen earlier, the I/O and repositioned links in MHs were used to implement interconnections in the lowest dimension of the original standard hypercubes. Therefore, removing these links affects the number of 1-cubes whose addresses have an X in the least significant bit. Moreover, the number of 1-cubes is increased due to the repositioning of links. To conclude, a number of subcubes of various dimensions which were present in the original hypercube cease to exist in the MH due to the removed links, and a number of new subcubes are added to the original hypercube due to the repositioning of half of the removed links.

We may distinguish between two categories of subcubes in MHs. All the subcubes that are also part of the original hypercube fall into the first category, whereas subcubes that were not present in the original hypercube belong to the second category. The following lemmas and theorems are pertinent.

Lemma 1: In any n-cube, with $n > 0$, there are only two nodes which do not have

at least a zero(0) *and* a one(1) in their binary addresses.

Proof: Since there can be only one node whose binary address consists of all zeroes and, similarly, only one node with binary address consisting of all ones, the number of nodes in an n-cube with their respective addresses consisting of at least a zero *and* a one is $2^n - 2$. □

Lemma 2: The number of subcubes of dimension k in an n-cube is $\binom{n}{k} 2^{n-k}$.

Proof: The number of distinct ways in which k bits can be selected among n bits is $\binom{n}{k}$. For each of these combinations, the set of the remaining $n-k$ bits can take 2^{n-k} distinct values. Hence, the result. □

Theorem 1: In an n-cube the number $\Phi(n,k,\lambda)$ of subcubes of dimension k that do not contain any of the $2^{\lambda-1}$ nodes whose binary addresses have in their lower $n-\lambda$ bits either all zeroes or all ones is

$$\Phi(n,k,\lambda) = \begin{cases} \sum_{i=0}^{min\{\lambda,n-k-2\}} \binom{\lambda}{i} 2^i (2^{n-k-i}-2) \binom{n-\lambda}{k-(\lambda-i)} \\ \text{if } k > \lambda \text{ and } k \leq (n-2) \\ \\ \sum_{i=0}^{min\{k,n-\lambda-2\}} \binom{\lambda}{k-i} 2^{\lambda-(k-i)} (2^{n-\lambda-i}-2) \binom{n-k}{i} \\ \text{if } k \leq \lambda \text{ and } k \leq (n-2) \end{cases}$$

Proof: **Case (a): $k > \lambda$ and $k \leq (n-2)$.**

The objective here is to find all possible subcubes of dimension k so that the lower $n-\lambda$ bits of the subcube addresses contain at least a zero and a one. With the usual notation for the subcubes, let the upper λ bits of the address contain all Xs. Then, the remaining $k-\lambda$ Xs can be selected among the available $n-\lambda$ lower address bits in $\binom{n-\lambda}{k-\lambda}$ distinct ways. For each of these combinations, the remaining $n-k$ bits can occupy any combination such that there is at least a zero and a one in these bits. From *Lemma 1*, this equals $2^{n-k}-2$. Now, let the upper λ bits

contain $\lambda - 1$ Xs. From *Lemma 2*, this can be selected in $\binom{\lambda}{\lambda-1} 2^1$ distinct ways. The remaining $k - (\lambda - 1)$ Xs can be accommodated in the lower $n - \lambda$ bits in $\binom{n-\lambda}{k-(\lambda-1)}$ distinct ways. Again, for every combination the remaining bits can be selected in $2^{n-(k+1)} - 2$ distinct ways, from *Lemma 1*. The above procedure continues until there are no Xs in the upper λ bits or until there are no more than $n - \lambda - 2$ Xs in the lower $n - \lambda$ bits. $\Phi(n, k, \lambda)$, the number of subcubes of dimension k, is equal to the sum of all the above combinations and is given by

$\Phi(n, k, \lambda) = \sum_{i=0}^{min\{\lambda, n-k-2\}} \binom{\lambda}{i} 2^i (2^{n-k-i} - 2) \binom{n-\lambda}{k-(\lambda-i)}$

if $k > \lambda$ and $k \leq (n-2)$

Case (b): $k \leq \lambda$ **and** $k \leq (n-2)$.

Let all the Xs be in the upper λ bits of the address. From *Lemma 2*, this can be selected in $\binom{\lambda}{k} 2^{\lambda-k}$ distinct ways and the remaining $n - \lambda$ bits can be selected in $2^{n-\lambda} - 2$ distinct ways. All other valid combinations for the existence of the subcube can be obtained by transferring the Xs from the upper λ bits to the lower $n - \lambda$ bits, one-by-one, until there are no more Xs in the upper λ bits or until there are at least two bits in the lower $n - \lambda$ bit field which are not Xs. Then, $\Phi(n, k, \lambda)$ is given by

$\Phi(n, k, \lambda) = \sum_{i=0}^{min\{k, n-\lambda-2\}} \binom{\lambda}{k-i} 2^{\lambda-(k-i)} (2^{n-\lambda-i} - 2) \binom{n-k}{i}$

if $k \leq \lambda$ and $k \leq (n-2)$.□

Theorem 2: The number $N(n, k, \lambda)$ of subcubes of dimension k in an MH $H(n, \lambda)$ is equal to

$2\binom{n-1}{k} 2^{n-1-k} + \Phi(n-1, k-1, \lambda) + 2^{\lambda-k} \left[\binom{\lambda}{k} - \binom{\lambda-1}{k} \right]$

where $\Phi(n', k', \lambda')$ is given by Theorem 1.

Proof: From the definition of modified hypercubes, $H(n, \lambda)$ has 2^λ I/O nodes

which implies that 2^{λ} of the links in the original hypercube $H(n, nil)$ have been removed. The removed links have "addresses" similar to those of the I/O PEs except that their LSBs are don't cares (i.e., Xs). The I/O PEs have addresses such that, for every possible combination in the upper $\lambda - 1$ bits there are two such nodes with one of them containing all zeroes in the lower $n - \lambda + 1$ bits and the other one containing all ones in the lower $n - \lambda + 1$ bits.

The problem of determining the number of subcubes of dimension k in $H(n, \lambda)$ can be broken down into two cases:

(a) determining the number of subcubes in $H(n, nil)$ with λ links removed as mentioned above, and

(b) determining the number of subcubes in the hypercube of dimension λ formed by the non-I/O PEs which are attached to repositioned links.

Case (a): This can be further divided into two subcases:

Subcase (i): The subcube address is selected in such a way that there is no X in the least significant bit of the hypercube address. Since k Xs can be selected among $n-1$ bits in $\binom{n-1}{k} 2^{n-k-1}$ distinct ways (*Lemma 2*) and the least significant bit can be either a '0' or a '1', the number of subcubes is equal to $2\binom{n-1}{k} 2^{n-k-1}$.

Subcase (ii): The subcube address is selected in such a way that there is an X in the least significant bit of the hypercube address. This is a special case in Theorem 1, with $n' = n-1, \lambda' = \lambda$ and $k' = k-1$. Hence, the number of subcubes is given by $\Phi(n-1, k-1, \lambda)$.

Case (b): One of the properties of MHs is that the non-I/O PEs which are attached to repositioned links form a hypercube of dimension λ.

Now this hypercube is formed by two hypercubes of dimension $\lambda - 1$ which belong to the original hypercube. All the links that connect these two hypercubes to form the hypercube of dimension λ are due to the repositioned links. The number of subcubes of dimension k due to the original hypercubes of dimension $\lambda - 1$ is $2\binom{\lambda-1}{k}2^{\lambda-1-k}$ *(Lemma 2)*, and the number of subcubes of dimension k due to the new hypercube of dimension λ is $\binom{\lambda}{k}2^{\lambda-k}$. Since we are interested in only those subcubes of dimension k that include the repositioned links (to make the subcubes selected in this fashion to be mutually exclusive from the earlier selected subcubes), we subtract the former number from the latter to get $\binom{\lambda}{k}2^{\lambda-k} - 2\binom{\lambda-1}{k}2^{\lambda-k-1}$ which is the same as $2^{\lambda-k}\left[\binom{\lambda}{k} - \binom{\lambda-1}{k}\right]$.□

3.3.2 The Table Look-Up Strategy

Since MHs are obtained by slightly modifying standard hypercubes, direct application of the earlier mentioned processor allocation strategies of Section 3.2 to MHs would lead to inefficient results. In this section, a processor allocation strategy, which is based on a *table look-up* approach, is proposed for MHs. Our strategy can recognize all possible subcubes in $H(n, \lambda)$ by maintaining $n - 1$ independent lists, one list per possible subcube dimension i, where $1 \leq i \leq (n - 1)$. The addresses of all possible subcubes in the MH are determined and stored in the form of linked lists, with each list representing subcubes of the same dimension. In fact, the structure of elements in the linked lists consists of the following fields.

1. A unique address consisting of n ternary symbols.

2. A bit to indicate the availability of the subcube.

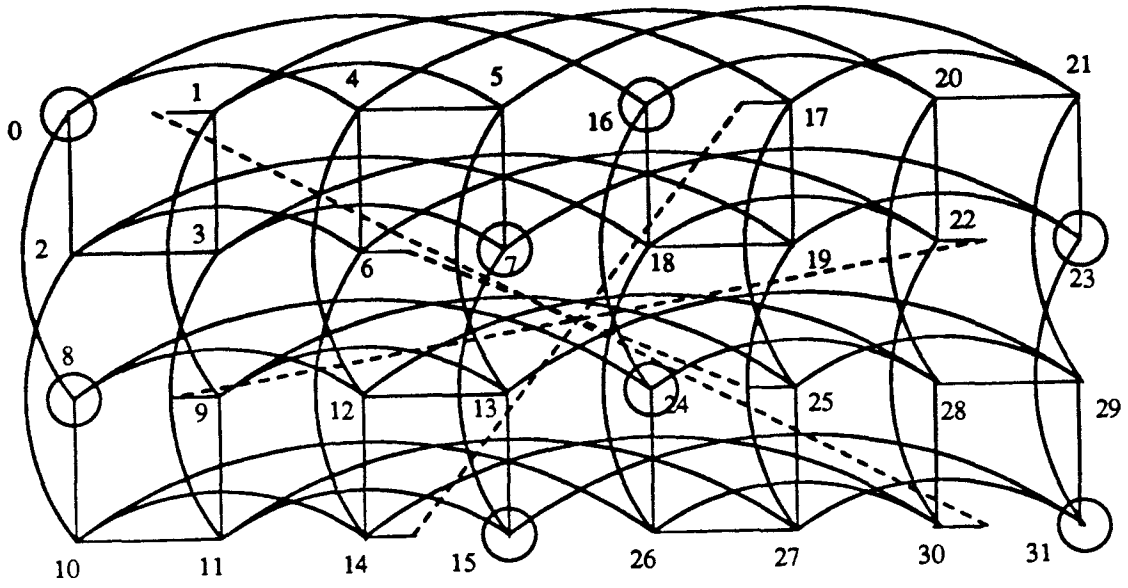
3. A bit to indicate whether the subcube contains any repositioned links.

4. An integer to represent the number of busy processors in the subcube.

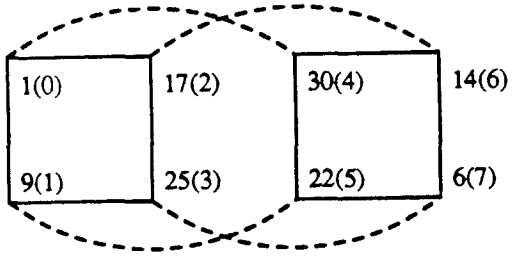
Initially, the $n-1$ independent lists are formed so that the ith list consists of $N(n,i,\lambda)$ entries. The address of the subcube in each entry is intially determined and stored in the address field of the entry whereas the subcube allocation bit and the variable representing the number of busy processors are set to zero.

Whereas the subcubes due to the original hypercube can be represented in the usual manner with n ternary symbols, the addresses of the subcubes due to the repositioned links are not obtained in a straightforward manner. The following procedure is undertaken to represent these subcube addresses.

The λ-cube formed by non-I/O PEs attached to repositioned links is mapped onto a regular hypercube j of dimension λ; i.e., the lower λ bits in the subcube address field (which is n bits long) in the lists due to the λ-cube contain the addresses of the corresponding subcubes in the regular hypercube j and the bit which signifies that the subcube is due to repositioned links is set to one. Note that the λ-cube is made of two $(\lambda-1)$-cubes which belong to the original hypercube. Also, one of the $(\lambda-1)$-cubes has Xs in the $\lambda-1$ most significant bits of its address, followed by all zeroes in the lower field, except the LSB which is a one. Similarly, the other $(\lambda-1)$-cube has Xs in the $\lambda-1$ most significant bits of its address, followed by all ones in the lower field, except the LSB which is a zero. The repositioned links connect these two $(\lambda-1)$-cubes in such a way that any two end nodes' addresses complement one another. Figure 3.2(a) shows a 5-cube with $\lambda=3$ (i.e., an $H(5,3)$). Figure 3.2(b) shows the λ-cube with the repositioned links connecting the two $(\lambda-1)$-cubes and the mapping of this λ-cube onto a regular λ-cube; the addresses of nodes between the parentheses are the addressses in the regular λ-cube.



(a)



(b)

Figure 3.2: Subcubes due to repositioned links. (a) An MH $H(5, \lambda)$; (b) the λ-cube with repositioned links mapped onto a regular λ-cube.

To obtain the actual address of any node in the λ-cube, the corresponding address a_x in the regular hypercube j is first determined. If the MSB of a_x is zero, then the actual node address is the value obtained when a_x is shifted left $n - \lambda - 1$ times with a one added to its LSB. If the MSB of a_x is one, then a_x is again shifted left $n - \lambda - 1$ times and all the bits except the LSB of the new value are complemented to obtain the actual node address.

When the subcube lists are made, approximately the first 2^{n-k} addresses in the kth list are selected in such a way that the same addresses would have been selected if the buddy strategy were to be implemented.

Allocation:

Step 1: Set k equal to the dimension of a subcube required to accommodate the current request.

Step 2: If there is an available subcube for the request size in the k-cube list (i.e., there exists an available k-cube with busy-processor count equal to zero), allocate it. For each of the nodes in the allocated subcube, determine the other subcubes among all the lists that contain the node under consideration, reset their available bit, and increment their busy-processor count.

Step 3: If a subcube of the requested size is not found, then keep the request in the waiting queue (until a subcube of the required size becomes available).

Deallocation:

Step 1: After a task is completed, the corresponding subcube is deallocated by setting its available bit.

Step 2: For each of the nodes in the subcube, determine other subcubes among all the lists that contain this node and decrement their busy-processor count.

Step 3: Set the deallocated subcube's busy-processor count to zero.

This processor allocation strategy can also be used for standard hypercubes. Of course, there will not exist any subcubes containing repositioned links. The following theorem is pertinent.

Theorem 3: The table look-up strategy is statically optimal for allocation on standard hypercubes.

Proof: For standard hypercubes, the look-up table is made up of $n + 1$ lists, one for every possible subcube dimension k, where $0 \leq k \leq n$. These lists are arranged in such a way that the first 2^{n-k} elements have the same subcube addresses with the ones the buddy strategy would look for, if the buddy strategy were to be implemented. Only if these subcubes are unavailable, will the table look-up strategy look for other possible combinations by searching the rest of the list. It has been proven in [7] that the buddy strategy is statically optimal. Hence, it can be concluded that the table look-up strategy is also statically optimal for allocation on standard hypercubes. □

The table look-up strategy has a high degree of fault tolerance. If any of the PEs are faulty, then the available bit of all the subcube entries in the lists that contain faulty PEs will be "permanently" reset.

3.3.3 The Modified Table Look-Up Strategy

This strategy also can recognize all available subcubes in an MH. Similarly to the earlier strategy, $n - 1$ independent lists are maintained, one list for each possible

subcube dimension k, where $1 \leq k \leq (n-1)$. This strategy does not keep track of the number of busy processors in each of the subcubes. Hence, the elements of the $n-1$ lists contain of the following fields:

1. An address consisting of n ternary symbols.
2. A bit to indicate the availability of the subcube.
3. A bit to indicate whether the subcube contains any repositioned links.

In addition, a global bit is associated with every PE address indicating the availability of the respective PE. Given a subcube address, the addresses of the PEs forming the subcube are determined in the same manner as explained in the previous subsection.

Allocation:

Step 1: Same as in the table look-up strategy.

Step 2: Find the first subcube in the k-cube list which is available, for which all individual contained PEs are also available. Allocate the subcube to the request and reset the available bit of the respective PEs. Reset the available bit of this subcube.

Step 3: Same as in the table look-up strategy.

Deallocation:

Step 1: Set the available bit of the released subcube as well as those of the individual PEs contained in this subcube.

Assuming that the addresses of subcubes stored in the lists by the modified table look-up strategy are arranged in the same fashion as for the table look-up

strategy, then the performance measures of these two strategies will be identical. For this reason, only the table look-up strategy was simulated to evaluate their performance. The difference between the two strategies lies in their execution times and memory requirements. More specifically, the modified strategy requires much less memory space due to the fact that it uses a global bit per PE instead of a counter per possible subcube. However, checking the availability bit of each PE in a subcube is a more time consuming process than checking the value of a single counter.

3.3.4 An Improved Processor Allocation Strategy for MHs

Although all of the proposed strategies are characterized by a perfect subcube recognition ability, their space complexity is relatively high due to the tables created at static time. This section presents a strategy which is based on the free list strategy but incorporates parts of the buddy and table look-up strategies in order to reduce the space complexity while at the same time retaining the perfect subcube recognition ability.

Similarly to the buddy strategy, 2^n bits are maintained to keep track of the availability of the 2^n nodes in the modified n-cube. A list of the "addresses" of the removed links is also formed. Also, $n-1$ free lists are maintained as in the free list strategy. Finally, lists similar to the ones used in the table look-up strategy are created, except that in the present case the lists contain only the addresses of subcubes containing repositioned links.

Allocation:

Step 1: Determine the first available subcube from the free lists in such a way that none of the subcube's link "addresses" is found in the list of "addresses" for removed links. Allocate this subcube according to the

free list strategy and reset the available bits of all the PEs that form the subcube. Terminate the algorithm.

Step 2: If a subcube was not found in the free lists, then find the first subcube from the list of subcubes that contain repositioned links so that all the PEs in the subcube are available. Allocate this subcube in accordance to the table look-up strategy and reset the allocation bits of all the PEs forming the subcube.

Step 3: If no subcube of the requested dimension is available, then keep the request in the waiting queue.

Deallocation:

Step 1: If the subcube to be relinquished contains repositioned links, then apply the deallocation procedure of the table look-up strategy, else follow the deallocation steps of the free list strategy.

Step 2: Set the individual available bits of all the PEs contained in the deallocated subcube.

3.3.5 A Parallel Table Look-Up Strategy

The inherent parallelism in the table look-up strategy can be exploited by a parallel system employing $n-1$ PEs in order to dramatically reduce the allocation time. More specifically, the $n-1$ different lists will be kept in $n-1$ different PEs.

Parallel Allocation:

Step 1: Same as in the table look-up strategy.

Step 2: Forward the k-cube request to the PE containing the k-cube list. If there is an available subcube for the request size in the k-cube list,

allocate it. Broadcast the address of the allocated subcube to all of the $n-1$ PEs. For every PE represented by the broadcasted address, check all of the $n-1$ lists in *parallel* to see if any subcube addresses represent this PE. If yes, increment the corresponding busy-processor counts. Reset the available bit of the allocated subcube.

Step 3: If a subcube of the requested size is not found, then keep the request in the waiting queue.

Parallel Deallocation:

Step 1: Broadcast the released subcube address to all of the $n-1$ PEs that contain subcube allocation lists. For every PE represented by the broadcasted address, check all of the $n-1$ lists in *parallel* to see if any subcube addresses represent this PE. If yes, decrement the corresponding busy processor counts.

Step 2: Set the available bit of the released subcube and reset its busy-processor count to zero.

CHAPTER 4

SIMULATION RESULTS

The simulation procedure is similar to that used in [8] and [23]. The results obtained for the table look-up strategy are compared against those obtained for the buddy strategy. The following assumptions were made for the simulation. Incoming requests arrive in every time unit, while this process continues for a total of T time units. The residence times of the subcube requests are assumed to be uniformly distributed. Since the direct application of the buddy strategy on MHs would result in subcubes of higher dimensions not being recognized, depending on the values of λ and n, the requested subcubes are restricted to relatively low dimensions. The dimensions of the subcubes required by the incoming requests are assumed to have either uniform or biased normal distribution. A queue is maintained for every subcube dimension. Every time a new job arrives, it is pushed to the bottom of the respective queue. At every instant of time, if jobs are completed, the corresponding subcubes are released. Further, all of the queues are checked for any waiting requests and if the corresponding subcubes are available, they are allocated. Servicing the subcube requests can be done either from the highest dimension queue down to the smallest dimension queue, in which case the priority is *highest dimension first*, or in the reverse order according to the *smallest dimension first* priority. In either case, the priority within a given queue is FCFS (First-Come, First-Served).

The following data were collected by averaging the results over 100 independent runs of the simulation program.

C: Number of requests that were assigned in time T.

D: Total delay involving all requests until time T.

U: Total utilization of the MH in time T. $U = \sum_{i=1}^{C} 2^{|I_i|t_i}$, where $|I_i|$ is the ith request size and t_i is the residence time until T of the request I_i.

The three performance measures that were obtained from the above data are the average waiting delay per request, the number of jobs completed in time T, and the efficiency of the strategy. The average waiting delay is given by D/T; the efficiency is given by $U/(T2^n)$ and the job completion is directly given by C.

The simulation was performed for $n = 6$ and $\lambda = 3$. The requesting subcubes were assumed to be of size ranging from 1-3. Table 4.1 shows the simulation results when the arriving requests are assumed to have a biased normal distribution. More specifically, the arriving requests have the following probabilities for subcube dimension: $p_1 = 0.5762$, $p_2 = 0.3142$ and $p_3 = 0.1096$, where p_k is the probability that the request is for a subcube of dimension k. The priority for subcube allocation is smallest dimension first. In the next set of runs, this priority is changed to largestdimension first and the results are shown in Table 4.2. Tables 4.3 and 4.4 show similar results when the arriving requests follow a uniform distribution (i.e, $p_1 = p_2 = p_3 = 1/3$). Tables 4.1 through 4.4 present results for T equal to 100.

It can easily be seen that the table look-up strategy outperforms the buddy strategy. Plots of efficiency and job completion against the average residence time are shown in Fig. 4.1. The table look-up strategy proves to have a good performance especially when the residence time is small. Note that the buddy strategy has a better task completion count when the arriving tasks follow the biased normal

Table 4.1: Biased Normal Distribution - Smallest Dimension First

Res. Time	T	Avg. Wait. Delay		Job Completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	5.16	94.90	84.83	24.65	18.21
	200	0.00	10.59	195.00	174.04	25.11	18.55
	300	0.00	16.03	294.98	263.10	25.33	18.67
10	100	0.00	5.17	90.02	80.42	48.05	35.51
	200	0.00	10.60	190.04	169.56	49.63	36.62
	300	0.00	16.03	290.00	258.65	50.12	36.98
15	100	0.25	5.50	84.73	75.69	69.53	51.43
	200	0.29	11.06	184.27	164.12	73.20	53.86
	300	0.30	16.58	284.71	253.39	74.45	54.74
20	100	2.25	7.57	76.32	67.53	80.48	61.58
	200	4.86	16.19	171.12	149.99	84.99	64.01
	300	7.47	25.06	265.57	231.66	86.48	64.79
25	100	4.89	11.08	67.05	57.24	85.35	64.64
	200	10.34	23.90	156.70	131.68	90.00	67.27
	300	15.80	37.06	245.52	204.83	91.61	67.87

distribution, whereas the table look-up strategy has a very similar performance for both distributions. Thus, the buddy strategy has a poor performance when the requests are for subcubes of relatively high dimensions.

From the simulation results in [23], it can be observed that for standard hypercubes the performance of other processor allocation strategies, like the Gray code, the modified buddy, and the free list strategies varies 5-10% from that of the buddy strategy. Therefore, based on our simulation results, we may conclude that none of the other existing processor allocation strategies for standard hypercubes could outperform the proposed strategy for MHs. It should be noted that the strategies proposed in this thesis can also be applied to standard hypercubes with small modifications.

Table 4.2: Biased Normal Distribution - Largest Dimension First

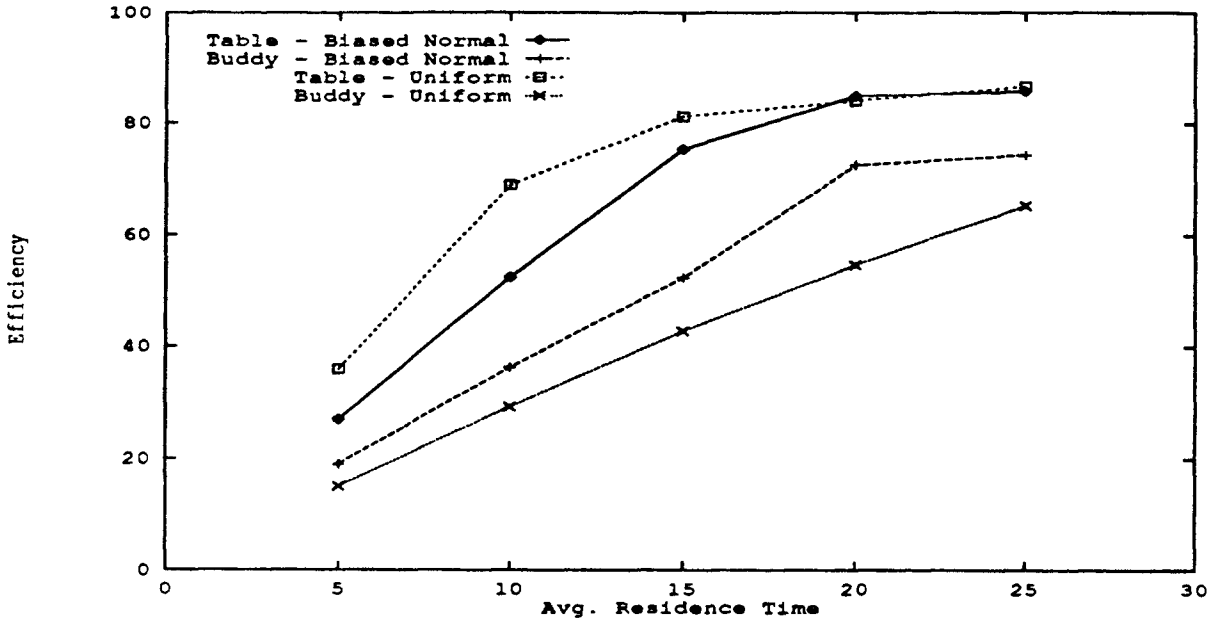
Res. Time	T	Avg. Wait. Delay		Job Completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	5.16	94.90	84.83	24.65	18.21
	200	0.00	10.59	195.00	174.04	25.11	18.55
	300	0.00	16.03	294.98	263.10	25.33	18.67
10	100	0.00	5.17	90.02	80.42	48.05	35.31
	200	0.00	10.60	190.04	169.57	49.63	36.62
	300	0.00	16.03	290.00	258.65	50.21	36.98
15	100	0.18	5.38	84.80	75.85	69.79	51.58
	200	0.20	10.89	184.40	164.45	73.29	54.00
	300	0.20	16.36	284.78	253.78	74.54	54.85
20	100	2.16	7.16	76.25	68.09	84.37	63.69
	200	4.65	14.24	171.89	154.59	90.79	68.07
	300	6.75	21.08	267.89	241.10	93.27	69.73
25	100	6.72	12.18	63.36	54.91	88.40	66.64
	200	15.88	26.53	144.48	126.20	94.02	70.74
	300	25.97	41.14	222.72	196.87	95.98	72.15

Table 4.3: Uniform Distribution - Smallest Dimension First

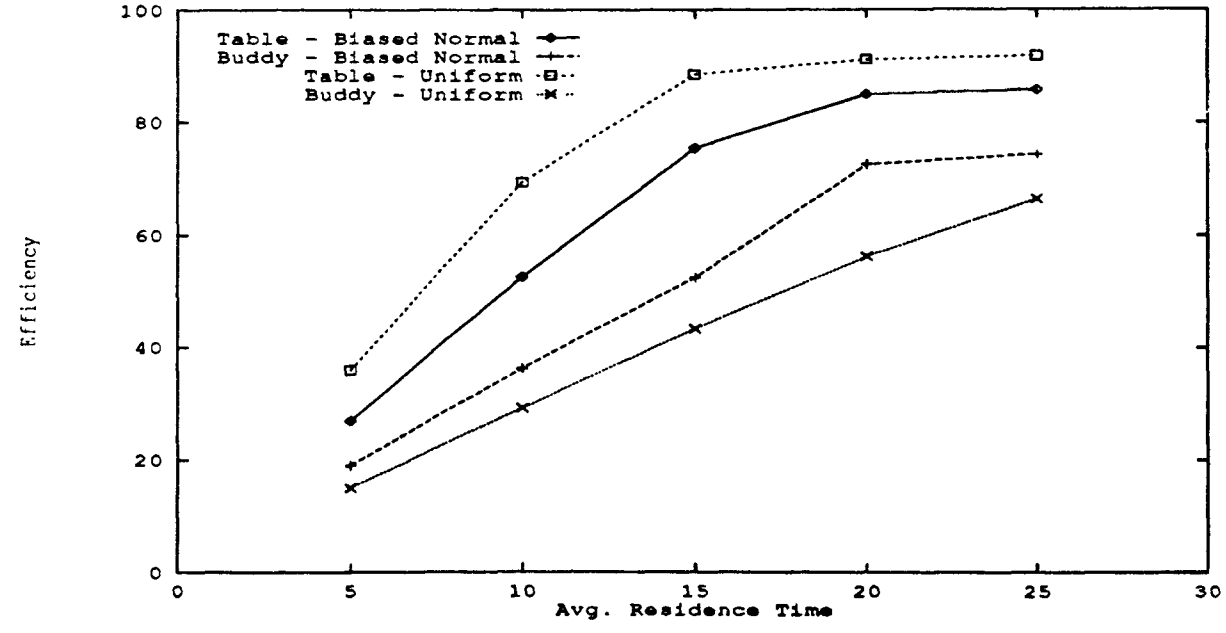
Res. Time	T	Avg. Wait. Delay		Job Completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	16.67	94.90	63.01	35.95	15.03
	200	0.00	33.56	195.00	129.08	36.58	15.27
	300	0.00	50.55	294.98	194.78	36.90	15.28
10	100	0.30	16.68	89.58	59.77	69.10	29.33
	200	0.45	33.56	189.36	125.78	71.55	30.13
	300	0.53	50.55	289.28	191.52	72.65	30.32
15	100	5.39	16.84	75.37	56.39	81.27	42.72
	200	11.51	33.78	162.83	122.00	84.17	44.51
	300	17.63	50.76	250.77	188.00	85.22	45.04
20	100	9.95	17.97	63.31	51.07	84.03	54.61
	200	21.43	36.01	140.01	115.02	86.04	56.06
	300	33.09	53.85	216.45	179.76	86.89	57.43
25	100	13.05	19.79	53.78	44.45	86.60	58.09
	200	27.55	40.79	125.48	102.46	88.76	61.16
	300	42.30	61.93	195.61	160.01	89.26	62.09

Table 4.4: Uniform Distribution - Largest Dimension First

Res. Time	T	Avg. Wait. Delay		Job Completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	16.67	94.90	63.01	35.95	15.03
	200	0.00	33.56	195.00	129.08	36.58	15.27
	300	0.00	50.55	294.98	194.78	36.90	15.28
10	100	0.21	16.68	89.69	59.77	69.51	29.33
	200	0.23	33.56	189.76	125.78	72.05	30.13
	300	0.25	50.55	289.70	191.52	73.01	30.32
15	100	5.06	16.82	76.15	56.43	88.45	42.77
	200	11.03	33.73	163.31	122.05	93.63	44.52
	300	17.61	50.72	249.09	188.03	95.55	45.05
20	100	12.06	17.71	58.19	51.47	91.06	53.41
	200	27.23	35.08	127.88	117.18	95.38	57.45
	300	43.36	52.27	193.46	183.01	96.89	58.81
25	100	17.67	19.32	45.32	45.26	91.72	60.64
	200	39.06	38.75	102.23	106.92	95.77	66.01
	300	61.37	57.80	156.44	169.38	97.18	68.21

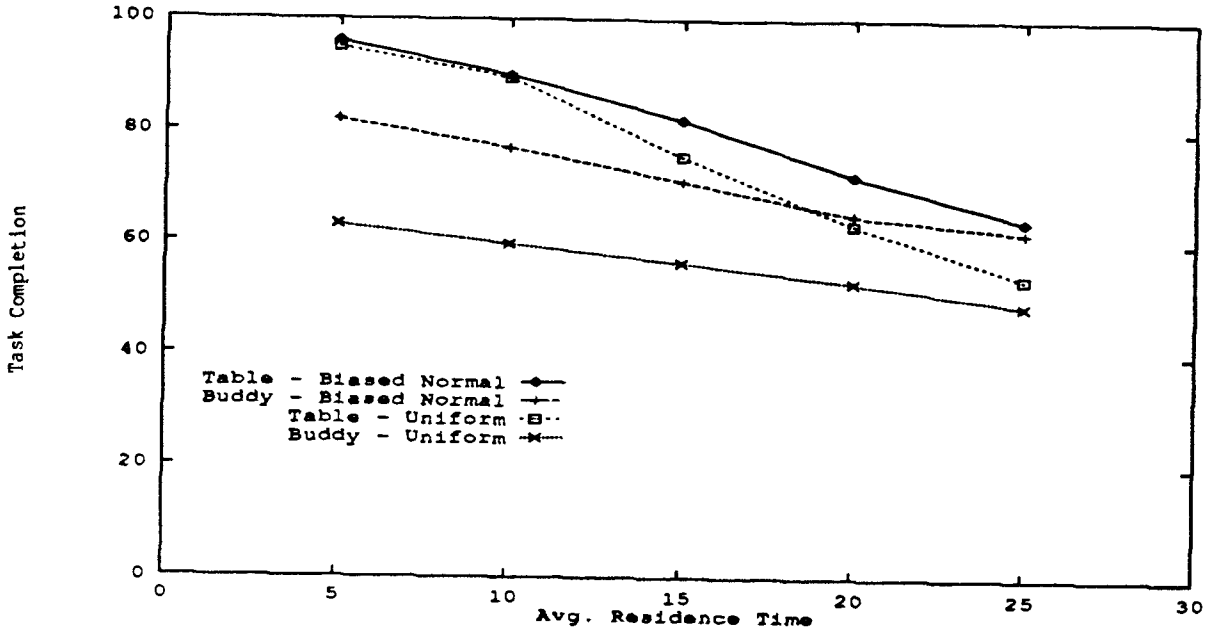


(a)

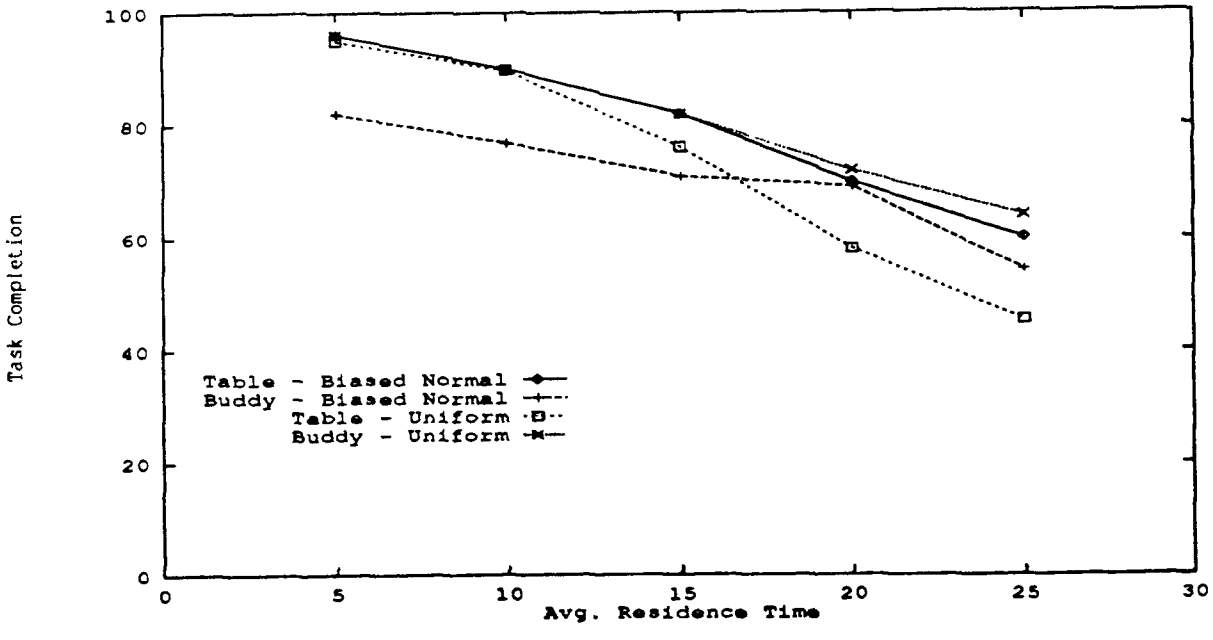


(b)

Figure 4.1: Performance of the table look-up strategy. (a) Efficiency curves - Smallest Dimension First Priority; (b) Efficiency curves - Largest Dimension First Priority; (c) Task Completion curves - Smallest Dimension First Priority; (d) Task Completion curves - Largest Dimension First Priority.


Task Completion
100
80
60
40
20
0
0
5
10
15
20
25
30
Avg. Residence Time
Table - Biased Normal
Buddy - Biased Normal
Table - Uniform
Buddy - Uniform

(c)


Task Completion
100
80
60
40
20
0
0
5
10
15
20
25
30
Avg. Residence Time
Table - Biased Normal
Buddy - Biased Normal
Table - Uniform
Buddy - Uniform

(d)

CHAPTER 5

CONCLUSIONS

Although the hypercube network is characterized by a set of very powerful features, its major drawback is that it can not be expanded in practice. In contrast, modified hypercubes (MHs) have proven to be promising alternative hypercube-like networks prone to incremental growth techniques. Processor allocation strategies reported in the literature for standard hypercube networks work poorly on MHs. This has been shown in this thesis by simulating the buddy strategy on MHs. A processor allocation strategy for MHs which is based on a "table look-up" approach was introduced. In fact, several serial versions, as well as a parallel version, of this strategy were proposed. The proposed strategies have a perfect subcube recognition ability and simulation results have shown that they are very efficient. Apart from the simulation results obtained, the following conclusions can be drawn from the study.

Although the MH is a slightly irregular structure, it provides a powerful building block for large multicube systems. Some interesting multicube architectures like the hypertorus have already been proposed [33]. The scope for introducing more such architectures is enormous. An important problem that needs to be addressed in the future is that of introducing subcube allocation techniques for multicube systems. Other related problems that need attention are those of embed-

ding frequently used topologies into multicubes and the development of techniques for mapping application algorithms onto multicubes.

BIBLIOGRAPHY

[1] Abraham, S., and Padmanabhan, K., "An Analysis of the Twisted Cube Topology," *in Proceedings of International Conference on Parallel Processing.* St. Charles, IL, Aug. 1989, pp. 116–120 (Vol. I).

[2] Ahmad, I., and Ghafoor, A., " A Semi Distributed Task Allocation Strategy for Large Hypercube Supercomputers," *in Proceedings of Supercomputing '90, IEEE Computer Society Press.* Lo Almitos, California, Nov. 1990, pp. 898–907.

[3] Al-Dhelaan, A., and Bose, B., "A New Strategy for Processors Allocation in an N-Cube Multiprocessor," *in Proceedings of International Conference on Computer Communication* Phoenix, AR, Mar. 1989, pp. 114–118.

[4] Amawy, A.E., and Latifi, S., "Properties and Performance of Folded Hypercubes," *IEEE Transactions on Parallel and Distributed Systems.* Vol.2, No. 1, Jan. 1991, pp. 31–42.

[5] Banerjee, P., "The Cubical Ring Connected Cycles: A Fault-Tolerant Parallel Computation Network," *IEEE Transactions on Computers* Vol. 37, No. 5, May 1988, pp. 632-636.

[6] Casavant, T.L., and Kuhl, J.G., "Analysis of Three Dynamic Load-Balancing Strategies with Varying Global Information Requirements," *in Proceedings of 7th International Conference on Distributed Computing.* West Germany, April 1987, pp. 185–192.

[7] Chan, T.F., and Saad, Y., "Multigrid Algorithms in the Hypercube Multiprocessors," *IEEE Transactions on Computers* Vol. C-35, No. 11, Aug. 1988, pp. 969-977.

[8] Chen, M.S., and Shin, K.G., "Processor Allocation in an N-Cube Multiprocessor using Gray Codes," *IEEE Transactions on Computers* Vol. C-36, Dec. 1987, pp. 1396–1407.

[9] Choi, S.B., and Somani, A.K., "The Generalized Folding-Cube," *in Proceedings of International Conference on Parallel Processing.* St. Charles, IL, Aug. 1990, pp. 372–375 (Vol. I).

[10] Deshpande, S.R., and Jeneven, R.M., "Scalability of a Binary Tree on a Hypercube," *in Proceedings of International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, Aug. 1986, pp. 661-668.

[11] Dutt, S., and Hayes, J.P., "On Allocating Subcubes in a Hypercube Multiprocessor," *in Proceedings of 3rd Conference on Hypercube.* Jan. 1988, pp. 801–810.

[12] Dutt, S., and Hayes, J.P., "Subcube Allocation in Hypercube Computers," *IEEE Transactions on Computers* Vol. C-40, No. 3, Mar. 1991, pp. 341–352.

[13] Esfahanian, A.-H., Ni, C.M., and Sagan, B.E., "The Twisted N-Cube with Application to Multiprocessing," *IEEE Transactions on Computers* Vol. C-40, No. 1, Jan. 1991, pp. 83-93.

[14] Fox, G.C., Kolawa, A., and Williams, R., "The Implementation of a Dynamic Load Balancer," *in Proceedings of 2nd Conference on Hypercube.* 1987, pp. 114-121.

[15] Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., and Walker, D.W. *Solving Problems on Concurrent Processors. Vol I.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[16] Ghose, K., and Desai, K.R., "The HCN: A Versatile Interconnection Network Based on Cubes," *in Proceedings of Supercomputing '89, IEEE Computer Society and ACM SIGARCH.* Reno, Nevada, Nov. 1989, pp. 426–435.

[17] Hillis, W.D. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[18] Ho, L.-T., and Johnsson, S.L., "Spanning Balanced Trees in Boolean Cubes," *SIAM Journal of Scientific Statistical Computing* 10, 4 (July 1989), pp. 607-630.

[19] Hsu, W.T.-Y., Yew, P.-C., and Zhu, C.-Q., "An Enhanced Scheme for Hypercube Interconnection Networks," *in Proceedings of International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, Aug. 1987, pp. 820-823.

[20] Johnsson, S.L., "Communication Efficient Basic Linear Algebra Computation on Hypercube Architectures," *Journal of Parallel and Distributed Computing.* 4,2 (April 1987), pp. 133-172.

[21] Katseff, H.P., "Incomplete Hypercubes," *IEEE Transactions on Computers* Vol. C-37, No. 5, May 1988, pp. 604–608.

[22] Kerola, T., and Hartmann, A., "Operational Analysis on Hyper-Rectangulars," *in Proceedings of International Conference on Parallel Processing.* St. Charles, IL, Aug. 1988, pp. 78–82 (Vol. I).

[23] Kim, J., Das, C.R., and Lin, W., "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Transactions on Parallel and Distributed Systems.* Vol. 2, No. 1, Jan. 1991, pp. 20–30.

[24] Lai, T.-H., and White, W., "Mapping Pyramic Algorithms into Hypercubes," *Journal of Parallel and Distributed Computing.* 9, 1990, pp. 42-54.

[25] Preparata, F.P., and Vuillemin, J., "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, Vol. 24, May 1981, pp. 300–309.

[26] Reddy, A.L.N., and Banerjee, P., "Design, Analysis, and Simulation of I/O Architectures for Hypercube Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems.* 1,2 (April 1990), pp. 140-151.

[27] Saad, Y., and Schultz, M.H., "Topological properties of hypercube," *IEEE Transactions on Computers* Vol. 37, Jul. 1988, pp. 867–872.

[28] Sen, A., "Supercube: An Optimally Fault-Tolerant Network Architecture," *Acta Informatica.* Vol. 26, 1989, pp. 741–748.

[29] Tzeng, N.-F., Chen, H.-L., and Chuang, P.-J., "Embeddings in Incomplete Hypercubes," *in Proceedings of International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, Aug. 1990, pp.335-339 (Vol. I).

[30] Tuazon, J., Paterson, J., Pnicl, M., and Liberman, D., "Caltech/JPL Mark II Hypercube Concurrent Processor," *in Proceedings of International Conference on Parallel Processing.* 1985, pp. 666-673.

[31] Wittie, C.D., "Communication Structures for Large Networks of Microcomputers," *IEEE Transactions on Computers.* Vol. C-30, No. 4, Apr. 1981.

[32] Wu, A.Y., "Embedding of Tree Networks into Hypercubes," *Journal of Parallel and Distributed Computing.* 2, 3 (Aug 1985), pp. 238-249.

[33] Ziavras, S.G., "On the Problem of Expanding Hypercube-Based Systems," *Journal of Parallel and Distributed Computing*, May 1992.

[34] Haravu, N.G., and Ziavras, S.G., "Processor Allocation Strategies for Modified Hypercubes," Submitted for publication.