1993

# Interconnection Networks Embeddings and Efficient Parallel Computations.

Emadeddin Mohamed Abuelrub
*Louisiana State University and Agricultural & Mechanical College*

# Interconnection networks embeddings and efficient parallel computations

Abuelrub, Emadeddin Mohamed, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1993

# INTERCONNECTION NETWORKS EMBEDDINGS AND EFFICIENT PARALLEL COMPUTATIONS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Emadeddin Abuelrub
BS in Computer Engineering, Oklahoma State University, 1984
BS in Computer Science, Oklahoma State University, 1985
MS in Computer Science, Alabama A&M University, 1987
August 1993

# Acknowledgements

I would like to express my sincere appreciations to Prof. Said Bettayeb for all the guidance and encouragement he has given me throughout my study. He has been a good friend and a very supportive advisor.

I would like to thank my committee members, Prof. Ahmad El-Amaway, Prof. Raymond Fabec, Prof. Sitharama Iyengar, Prof. Bush Jones, and Prof. Si-Qing Zheng, for all their encouragement and support. Finally, I thank my brothers Ahmad and Mohamed whose endless support served as the foundation for my achievements.

# Table of Contents

**Acknowledgements** .................... ii

**List of Figures** .................... v

**Abstract** .................... vii

**CHAPTER 1 Introduction** .................... 1
- 1.1 Flynn's Taxonomy .................... 2
- 1.2 A Taxonomy of Topologies .................... 6
- 1.3 Data Routing .................... 8
- 1.4 Overview .................... 10
- 1.5 Preliminaries and Terminology .................... 13
- 1.6 Graph Embedding .................... 18
- 1.7 Cost Functions .................... 20
- 1.8 Fault-Tolerance .................... 22
- 1.9 Outline of the Dissertation .................... 24

**CHAPTER 2 Parallel Computation on the Twisted Hypercube** .................... 26
- 2.1 Data Communication .................... 26
- 2.2 Basic Operations .................... 27
  - 2.2.1 Associative Computations .................... 28
  - 2.2.2 Parallel Prefix .................... 32
- 2.3 Summary .................... 33

**CHAPTER 3 Embedding Trees into Twisted Hypercubes** .................... 36
- 3.1 Introduction .................... 36
- 3.2 Embedding Complete Binary Trees into Twisted Hypercubes .................... 38
  - 3.2.1 The Recursive Embedding .................... 38
  - 3.2.2 The Inorder Embedding .................... 45
- 3.3 Embedding Complete Quad Trees into Twisted Hypercubes .................... 49
- 3.4 Summary .................... 57

**CHAPTER 4 Embedding Rings into Faulty Twisted Hypercubes** .................... 58
- 4.1 Introduction .................... 58
- 4.2 Fault-Free Embeddings .................... 59

4.2.1 The Embedding Sequence ........ 60
4.2.2 Divide-Conquer Embeddings ........ 63
4.3 Fault-Tolerance Embeddings ........ 66
4.3.1 Embedding in the Presence of a Single Fault ........ 66
4.3.2 Embedding in the Presence of Multiple Faults ........ 70
4.4 Summary ........ 75

**CHAPTER 5 Fault-Tolerance Embedding of Rings into Hypercubes** ........ 76
5.1 Introduction ........ 76
5.2 Fault-Free Embeddings ........ 77
5.2.1 Divide-Conquer Embeddings ........ 80
5.3 Fault-Tolerance Embeddings ........ 83
5.3.1 Embedding in the Presence of a Single Fault ........ 83
5.3.2 Embedding in the Presence of Multiple Faults ........ 88
5.4 Summary ........ 91

**CHAPTER 6 Concluding Remarks** ........ 93

**Bibliography** ........ 96

**Vita** ........ 103

# List of Figures

Figure 1.1 SIMD and MIMD Paradigms .......... 4

Figure 1.2 Some popular interconnection networks .......... 9

Figure 1.3 Hypercubes and twisted hypercubes for n = 1, 2, and 3 .......... 17

Figure 2.1 Broadcasting in a twisted hypercube .......... 27

Figure 2.2 The addition operation on a twisted hypercube of dimension 3 .......... 30

Figure 2.3 The addition operation via the broadcast tree .......... 31

Figure 2.4 The prefix operation on a twisted hypercube of dimension 3 .......... 34

Figure 3.1 Embedding CB into TQ for n = 1, 2, 3, and 4 .......... 39

Figure 3.2 Embedding CB into TQ for n = 5 .......... 40

Figure 3.3 The recursive embedding of CB into TQ .......... 42

Figure 3.4 The inorder embedding of CB into TQ for n = 1, 2, and 3 .......... 45

Figure 3.5 The inorder embedding of CB into TQ .......... 47

Figure 3.6 Partitioning CQ .......... 50

Figure 3.7 The recursive embedding of CQ into TQ .......... 51

Figure 3.8 Embedding CQ into TQ for n = 2 .......... 52

Figure 3.9 Embedding CQ into TQ for n = 3 .......... 54

Figure 4.1 The embedding sequence .................... 60

Figure 4.2 Fault-free embedding .................... 63

Figure 4.3 Blocks and cubes .................... 64

Figure 4.4 All possible locations of a faulty node .................... 68

Figure 4.5 Single fault embedding .................... 69

Figure 4.6 Multiple faults embedding .................... 72

Figure 4.7 All possible cases of two adjacent faulty cubes .................... 74

Figure 5.1 The embedding sequence .................... 78

Figure 5.2 The cube .................... 81

Figure 5.3 Fault-free embedding .................... 82

Figure 5.4 All possible locations of an upper faulty node within a cube .......... 85

Figure 5.5 All possible locations of a lower faulty node within a cube .......... 86

Figure 5.6 Single fault embedding .................... 87

Figure 5.7 Multiple faults embedding .................... 90

Figure 5.8 All possible cases of two adjacent faulty cubes .................... 92

# Abstract

To obtain a greater performance, many processors are allowed to cooperate to solve a single problem. These processors communicate via an interconnection network or a bus. Parallel machines are classified as either message passing machines where processors have their own memory or shared memory machines where several processors share the same memory. In this dissertation, we focus on the former. The most essential function of the underlying interconnection network is the efficient interchanging of messages between processes in different processors. The potential communication bottleneck has been the main drive in the design of interconnection networks. Parallel machines based on the hypercube topology have gained a great respect in parallel computation because of its many attractive properties. Many versions of the hypercube have been introduced by many researchers mainly to enhance communications. The twisted hypercube is one of the most attractive versions of the hypercube. It preserves the important features of the hypercube and reduces its diameter by a factor of two. This dissertation investigates relations and transformations between various interconnection networks and the twisted hypercube and explore its efficiency in parallel computation. The capability of the twisted hypercube to simulate complete binary trees, complete quad trees, and rings is demonstrated and compared with the hypercube. Finally, the fault-tolerance of the twisted hypercube is investigated. We present optimal algorithms to simulate rings in a faulty twisted hypercube environment and compare that with the hypercube.

CHAPTER 1

# Introduction

The need for faster computers has not ceased since the beginning of the computer era. New applications seem to push existing computers to their limit. The computer industry shows a continuous effort to increase the computational speed of computers. In the last four decades, dramatic increases in computing speed were achieved. Most of these were largely due to the use of faster electronic components by computer manufacturers. As we went from vacuum cubes to transistors and from small to very large scale integration, we witnessed the growth in size and range of the computational problems that we could solve. The state-of-the-art in VLSI technology can't satisfy the growing computational demands in many scientific and engineering applications. Without high performance computers, many of these challenges can't be solved within a reasonable time period.

In the last decade, as progress in VLSI has led to small size, low cost, and high performance processors, it has become practical to build parallel computers containing a very large number of processors. In parallel computation, a collection of processors cooperate to solve a problem by working simultaneously on different parts of the problem. The two major components of a parallel machine are the processors and the interconnection network that ties them together. A main concern in the development of such a system with this many processors is fault-tolerance. Since the probability of one or more processors or links becoming fault in such complex systems is significant, it is desirable to build some fault-tolerance features into the architecture.

Although parallel processing is not a new concept, its deviation from the traditional Von Neumann computational model has introduced many new problems. The extra complexity required for data communication among the processors can degrade system performance and make programming on a parallel processing system much harder than on a uniprocessor system. If each of the processors works autonomously, the synchronization among different processes will further increase the complexity of the system. Unless we have a clear understanding of these problems and the efficient tools to solve them, the full power of parallel processing cannot be achieved.

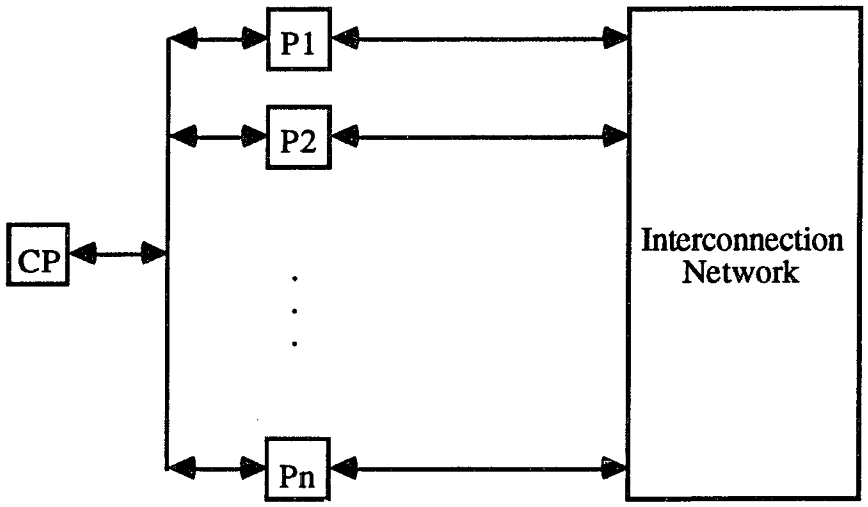
This dissertation adds to the growing body of work that addresses highly parallel computing for models of parallel machines. We specifically investigate relations and transformations between various interconnection networks and explore their efficiency in parallel computation. Both faulty and fault-free parallel architectures are considered.

## 1.1 Flynn's Taxonomy

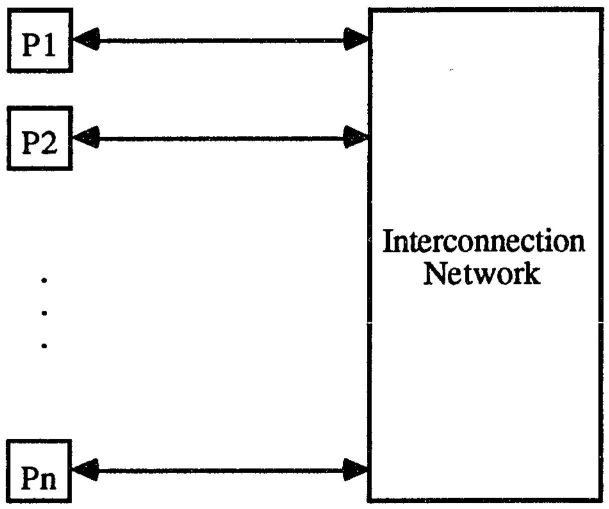
Parallel machines can be categorized by their interconnection network topologies. Also, we classify parallel machines as either shared memory or message passing machines. Within each of these categories, we further divide them into vector versus MIMD within the shared memory category and static versus dynamic within the message passing category. Message passing designs offer higher levels of parallelism through the interconnection of thousands of processors via an interconnection network. In such systems, there is no global memory or program space. The design of message passing parallel machines places great demand on communication speed, data partitioning, and routing.

The most widely accepted classification of parallel computation models is the one proposed by Flynn [F], who viewed the Von Neumann model as a single stream of instructions controlling a single stream of data (SISD). Flynn viewed parallelism as a single stream of instructions controlling a multiple stream of data (SIMD) or a multiple stream of instructions controlling a multiple stream of data (MIMD). Figure 1.1 shows SIMD and MIMD paradigms. Traleaven [T] classified MIMD machines further. The data mechanism was divided into shared-memory and message-passing approaches. The terms multicomputer and multiprocessor, respectively, are usually used to distinguish these two approaches.

In SIMD machines, all processors operate under the control of a single instruction stream issued by a central control processor. All processors do the same instruction, or nothing, each on a different datum. SIMD is the most useful paradigm for massively parallel scientific computing. Many scientific applications naturally fall into the SIMD paradigm such as image processing and particle simulation. In SIMD machines, a single instruction stream is acted upon by many processors in a lock step fashion. Only one instruction counter is used to sequence through a single copy of the program. The data that is processed by each processing element differs from one



(a) SIMD



(b) MIMD

Figure 1.1: SIMD and MIMD paradigms.

processor to another. Therefore, a single program and a single control unit simultaneously act on many different collections of data by controlling a collection of homogeneous processors. SIMD is the basic paradigm of synchronous data parallel computing. The classic example of parallel SIMD computers is the ILLIAC-IV, with 64 identical processing elements each receiving the same stream of instructions to be executed on its own data item.

In MIMD machines, processors operate under the control of their own stream of instructions which allows great flexibility. Each processor is fully programmable and capable of executing its own program. MIMD is the most general model of parallelism. Synchronization is achieved explicitly and locally rather than through a global synchronization mechanism. This provides a lot of flexibility, but it also means that the software that is needed to program the machine is more complex and much harder to implement. MIMD is useful when the problem allows multiple heterogeneous tasks to be performed at the same time. This is most likely to occur when the number of tasks to be performed is not known and the tasks perform different operations from one another.

MIMD is general enough to contain SIMD, because we can emulate SIMD behavior by restricting MIMD through careful programming. However, there may be severe performance penalties inherent in simulation of one form on a machine of different form.

## 1.2 A Taxonomy of Topologies

Interconnection networks and their combinatorial properties have been the topic of many recent research in the area of parallel processing ([AJ], [AK], [CLe], [FS], [Gou], [II], [K], [Lei], [LE], [Si], [Sn]). An efficient interconnection structure should have a low number of links per node, a small internode distance, and a large number of alternate paths between a pair of nodes for fault-tolerance. In a parallel machine, the average internode distance, message traffic density, and fault-tolerance are very much dependent on the diameter and the degree of the network. There is a tradeoff between the diameter and the degree of a network. A network with a low degree has a large diameter and a network with a low diameter usually has a large degree. A ring structure and a completely connected structure represent the two extremes. The diameter multiplied by the degree is usually a good criterion to measure the efficiency of an interconnection structure [AJ].

Most of the communication problems in parallel processing systems come from the fundamental different approaches adopted by uniprocessor systems and parallel processing systems to support interprocess communications. In a uniprocessor system, all processes reside in a single processor and all interprocess communications are supported by main memory references. As a result, any process can easily send a message to any other process with a uniform delay determined principally by the main memory clock cycle. On the other hand, in a parallel processing system, different processes usually reside in different processors. Interprocess communications are supported by an interconnection network. The delay incurred in an interconnection network is much greater than that in a uniprocessor. The delay time depends on the number of processors and the communication pattern. We call the extra interprocess communication time in a parallel processing system the *communication delay.*

The two main sources of the extra communication overhead in parallel processing are the time for the messages to go through one or more intermediate processors, in the absence of a direct link between the two processors communicating, and the contention for a single link by more than one message at the same time. These delays result from the mismatch of the communication characteristics of the parallel programs and those of the parallel processing system.

An *interconnection topology* of a set of processors is a mapping from the set of processors onto the same set of processors. The mapping describes how to connect processors to other processors, with each processor usually connected to a small number of processors in a regular pattern. For example, a ring topology is a mapping that connects a processor with label $i$ to processors with labels $i - 1$ and $i + 1$. A complete binary tree topology, is a mapping that associates processors to the nodes of a complete binary tree where the root processor is connected to two other processors, interior processors are connected to three other processors, and leave processors are connected to only one processor. Most parallel machines are distinguished by their interconnection topologies. While the speed and capacity of parallel machines may vary, the most significant difference between them is their interconnection topologies.

Interconnection networks that provide communication between the processors have ranged from the simple to the complex, representing the trade off between speed and cost. At one extreme is the ring network, in which each processor is linked to only two other processors. Messages are passed along the network from one processor to another by hopping through intermediate processors. At the other extreme in connectivity is the all-to-all network, in which each processor has its own private link to every other processor in the network. Between these two extremes, there is a number of other networks with intermediate numbers of neighbors. Figure 1.2 shows some popular interconnection networks.

Interconnection networks can be classified into dynamic and static networks. Dynamic networks create links between processors as the program executes. Static networks are fixed by design and can't be changed after the machine is built. Parallel machines based on the hypercube static interconnection structure are one of the most popular because they possess many attractive properties that are needed in parallel processing.

## 1.3 Data Routing

In a parallel processing system, if more than one message must be sent from a source to a destination at the same time, some links can be contended by more than one message. Since each link can support the communication of only one message at any instant, this contention introduces extra communication delay into the system. A good data routing algorithm should support parallel communication in the system with minimum delay.

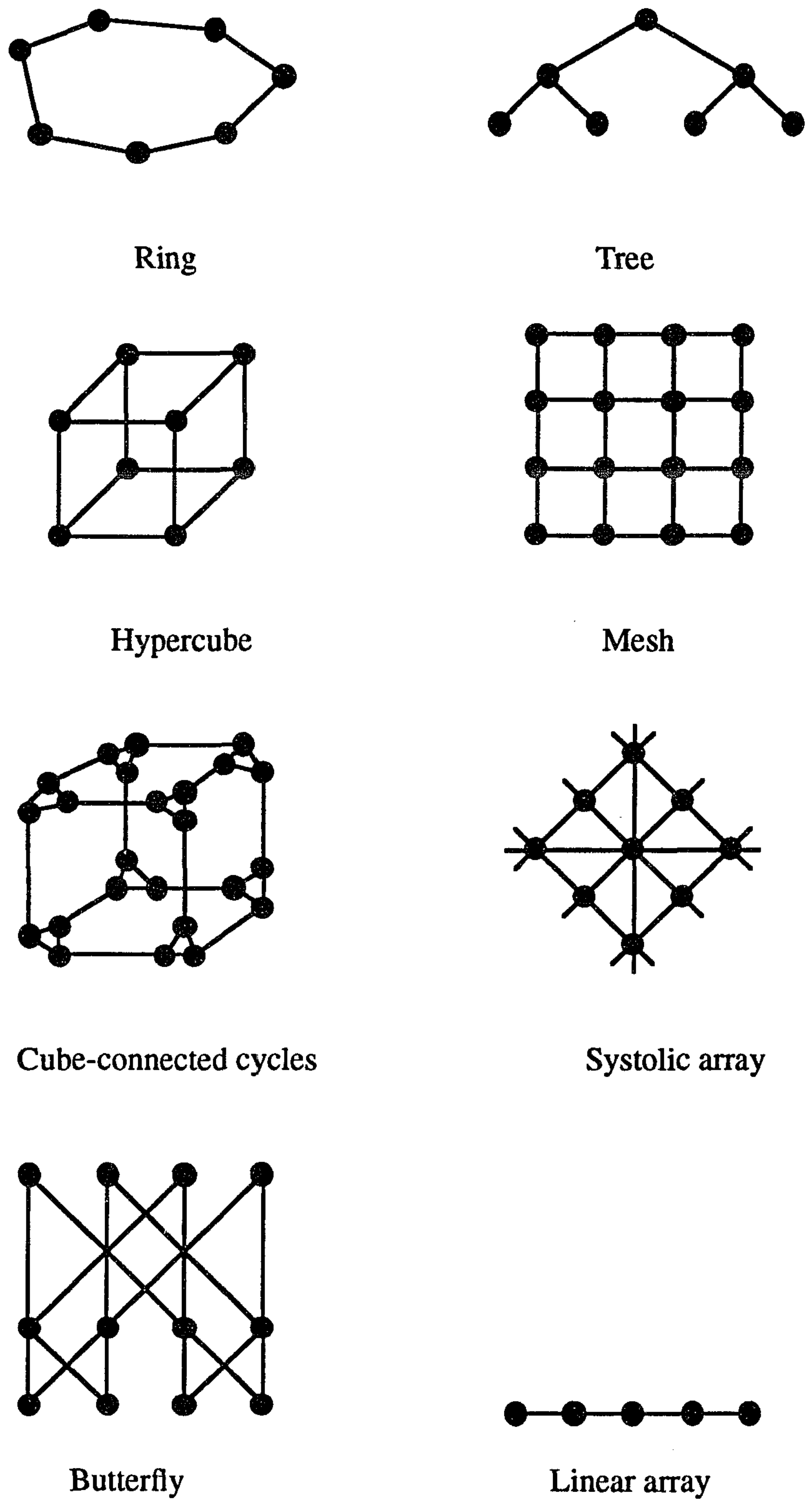


Figure 1.2: Some popular interconnection networks.

Circuit switching and packet switching are the two principal kinds of data routing mechanisms. In circuit switching, a physical path is established between the source and the destination. In packet switching, data are put in packets and routed through the interconnection network without establishing a physical connection path. Circuit switching is generally much more suitable for bulk data transmission, while packet switching is more efficient for many short messages.

In parallel processing systems for image processing, computer graphics, robot vision, and scientific computation, communications are heavy and message sizes are small. For these reasons, packet switching is usually preferred. There are two kinds of control strategies for packet switching, centralized and distributed. In centralized control, the decision to route packets is based on global information. In distributed control, each processor decides how to route the data based on its local information.

## 1.4 Overview

The Parallel Random Access Machine (PRAM) is used as a standard theoretical model for parallel computation. A PRAM is a synchronized machine with an unbounded number of identical processors and a global memory which allows simultaneous reads and writes from and into the same memory location ([AG], [Q], [U]). Algorithms will run faster on this model than on real machines. Actual machines can't be built without a significant delay in access time. The best that one can hope for is that access time is proportional to log $N$, where $N$ is the number of processors [AG]. This led many institutions to design parallel machines based on the message passing

MIMD approach. The classic example of such an architecture is the MARK-II Cosmic Cube.

Based on Kung's sorting algorithm for meshes [TK] and Batcher's merge sort for cube connected machines, Nassimi and Sahni [NS] proved that a Random Access Read (RAR) can be accomplished with complexity $O(q^2n)$ on a $q$ dimensional $n^q$ mesh machine and $O(log^2 N)$ on an $N$ cube connected or perfect shuffle machine. Also, they proved that a Read Access Write (RAW) can be accomplished with complexity $O(q^2n + dqn)$ on a $q$ dimensional mesh machine and $O(log^2 N + dlog N)$ on an $N$ cube connected or perfect shuffle machine, where $d$ is the maximum number of data items written into any processor.

Many researchers have concentrated on fiding efficient ways to simulate PRAM on other parallel machines. The first reasonable deterministic simulation of a PRAM was proposed by Upfal and Wigderson [UW]. Their simulation achieved $O(log^2 N \log \log N)$ time to simulate one step of a PRAM algorithm on an $N$ processor network. Alt *et al.* [AHMP] subsequently improved the time complexity to $O(log^2 N)$.

Valiant [V] reported a probabilistic routing algorithm that can perform any permutation on a hypercube machine of size $N$ in $O(log N)$ steps. The algorithm consists of two consecutive phases. In the first phase, it sends each packet $p$ to a randomly chosen node $v$. For each packet $p$, every node has the same probability of being chosen , which is $\frac{1}{N}$. The choices for the different packets are independent of each other.

In the second phase, it routes each packet $p$ from the intermediate node $v$ to its destination node. At each instant, there is exactly one copy of each packet. A packet might be transmitted along an edge, waiting in a queue associated with an edge, or stored as a loose packet in an intermediate node. For simplicity, the algorithm is described in a synchronized fashion. It alternates between a transmitting mode and a bookkeeping mode. In the transmitting mode, the packet at the head of each queue is transmitted along the edge associated with it and stored as a loose packet at the recipient node. In the bookkeeping mode, each loose packet is assigned to the queue of one of the outgoing edges according to some random choice, unless it has nowhere further to go in the current phase.

Valiant proved that this distributed randomized algorithm can route packets to their destination in a hypercube machine without two packets passing through the same communication link at the same time in $O(log\ N)$ with high probability. Each packet carries with it $O(log\ N)$ bits of information and no other communication among the nodes is needed. This result implies that a hypercube machine can simulate a PRAM with an increase in the execution time for each step. Each PRAM step can be simulated in approximately $O(log\ N)$ steps on a network of size $N$. Therefore, we can develop algorithms for the PRAM since we know how to translate them into algorithms for actual machines.

## 1.5 Preliminaries and Terminology

Several Structures have been proposed in the literature for interconnecting a large network of processors. Many parallel machines that are based on these structures are now commercially available. The Cosmic Cube [Se] is the first completed experimental parallel machine based on the hypercube structure. It becomes the archetype of early operative parallel machines. Since the Cosmic Cube, many machines based on the hypercube structure have been built and made commercially available such as Amet S/14, NCUBE/10, Intel IPSC, and the Connection Machine [H].

Parallel machines based on the hypercube topology have gained a great interest in parallel computing because of their flexibility and suitability for general purpose applications. Many of the properties of the hypercube that make it a desirable parallel machine are a direct consequence of the graph theoretic properties of the hypercube topology. The hypercube offers a rich interconnection topology with large bandwidth, logarithmic diameter, simple routing and broadcasting of data, recursive structure that is suited to divide and conquer applications, homogeneous and symmetric structure, and the ability to simulate other interconnection networks with minimum overhead. Also, it has a high fault-tolerance structure. Fault-tolerance and related issues are becoming an important topic in the design and analysis of parallel machines.

The hypercube has been the topic of many recent research. Various researchers have done extensive work in showing the parallel computational power of the hypercube machine in many directions. In one direction, many researchers have shown the capability of the hypercube machine to simulate other networks such as rings, trees, grids, and other interconnection networks with minimum overhead ([BCGS], [BCLR], [BMS], [BSu], [MS], [SS], [Lei]). In another direction, researchers have shown the power of the hypercube in solving many computational problems in parallel such as sorting, merging, matrix multiplication, and parallel prefix ([A], [HB], [LE], [Lei], [P], [Q], [QD], [St]). In a third direction, researchers have shown the robustness and fault-tolerance of the hypercube, focusing on the hypercube's ability to simulate, compute, route, and reconfigure itself in the presence of faults ([AGr], [BS], [HLNa], [HLNb], [WCM], [CL]).

Finally, many researchers have proposed modifications on the hypercube structure to improve its computational power ([BH], [EBSS], [ENS], [EL], [PV], [YN]). Bhuyan and Agrewal [BA] proposed a generalized hypercube structure. Preparata and Vuillemin [PV] introduced the cube-connected cycles in which the degree of the diameter was reduced to 3. Latifi and El-Amaway [EL] proposed the folded hypercube to reduce the diameter and the traffic congestion with little hardware overhead. Youssef and Narahari [YN] proposed the Banyan-hypercube network which combines the advantageous features and properties of Banyans and hypercubes and thus reduce the communication overhead.

A *hypercube* of dimension $n$, denoted by $Q_n$, is an undirected graph consisting of $2^n$ vertices, each vertex corresponds to an $n$-bit binary string, labeled from 0 to $2^n - 1$ and such that there is an edge between any two vertices if and only if the binary representation of their labels differ in exactly one bit position. Each vertex is incident to $n$ other vertices, one for each bit position. The edges of the hypercube can be naturally partitioned according to the dimensions that they traverse. An edge is called a *dimension i edge* if it links two vertices that differ in the $i^{th}$ bit position.

Another version of the hypercube, called the twisted hypercube, was introduced by Efe *et. al.* [EBSS]. Twisted hypercubes proved to contain the attractive properties of the hypercube and a better communication capabilities. In parallel machines, the communication cost dominates the computation cost. The overall performance of the parallel machine depends heavily on the underlying interconnection network. In twisted hypercubes, the diameter is reduced by a factor of two over that of the hypercube. Many of the hypercube's attractive features such sa partitioning, routing, and embedding are incorporated into the twisted hypercube and new gains are achieved in diameter, average distance, and embedding efficiency ([ABc], [E], [Z]).

Two binary strings $x = x_1x_0$ and $y = y_1y_0$, each of length two, are *pair-related* if and only if $(x, y) \in \{(00,00), (10,10), (01,11), (11,01)\}$. Let $G$ be any undirected labeled graph, then $G^b$ is obtained from $G$ by prefixing every vertex label with $b$. We define a twisted hypercube as follows.

A *twisted hypercube* of dimension $n$, denoted $TQ_n$, is an undirected graph consisting of $2^n$ vertices labeled from 0 to $2^n - 1$ and defined recursively as follows [EBSS].

(i) $TQ_1$ is the complete graph of two vertices with labels 0 and 1.

(ii) For $n > 1$, $TQ_n$ consists of two copies of $TQ_{n-1}$ one prefixed by 0, $TQ^0{}_{n-1}$, and the other by 1, $TQ^1{}_{n-1}$. Two vertices $u = 0u_{n-2}...u_0 \in TQ^0{}_{n-1}$ and $v = 1v_{n-2}...v_0 \in TQ^1{}_{n-1}$ are adjacent if and only if

1. $u_{n-2} = v_{n-2}$, if $n$ is even, and
2. for $0 \le i \le \lfloor (n-1)/2 \rfloor$, $u_{2i+1}u_{2i}$ and $v_{2i+1}v_{2i}$ are pair-related.

Such an edge $(u, v)$ is referred to as a *dimension n edge*, for all $n > 1$.

There exist a dilation two and expansion one embedding of the twisted hypercube into the hypercube and vice virsa [E]. Figure 1.3 shows hypercubes and twisted hypercubes for $n = 1, 2,$ and 3. It is more convenient to view both the hypercube and the twisted hypercube in this way, where the upper part consists of all nodes with even labels and the lower part consists of all nodes with odd labels. An *upper node* is a node that lies in the upper part of the structure, *i.e.*, its least significant bit is a 0. A *lower node* is a node that lies in the lower part of the structure, *i.e.*, its least significant bit is a 1. An *upper link* is a link that connects two upper nodes and a *lower link* is a link that connects two lower nodes.

Trees are special kind of graphs which have a wide variety of applications in the field of computer science. A *k-ary tree* of height $n-1$ is an undirected graph that has $\frac{(k^n - 1)}{(k-1)}$ vertices and consists of a root of degree $k$ with no parent and $k$ children, interior nodes of degree $k+1$ with one parent and $k$ children, and leaves of degree one with one parent and no children. Spanning trees are very important in the context of efficient communications and in the determination of distances between nodes in a network. Binary trees are important tools in the evaluation of formulas and in the study of branching of processes.

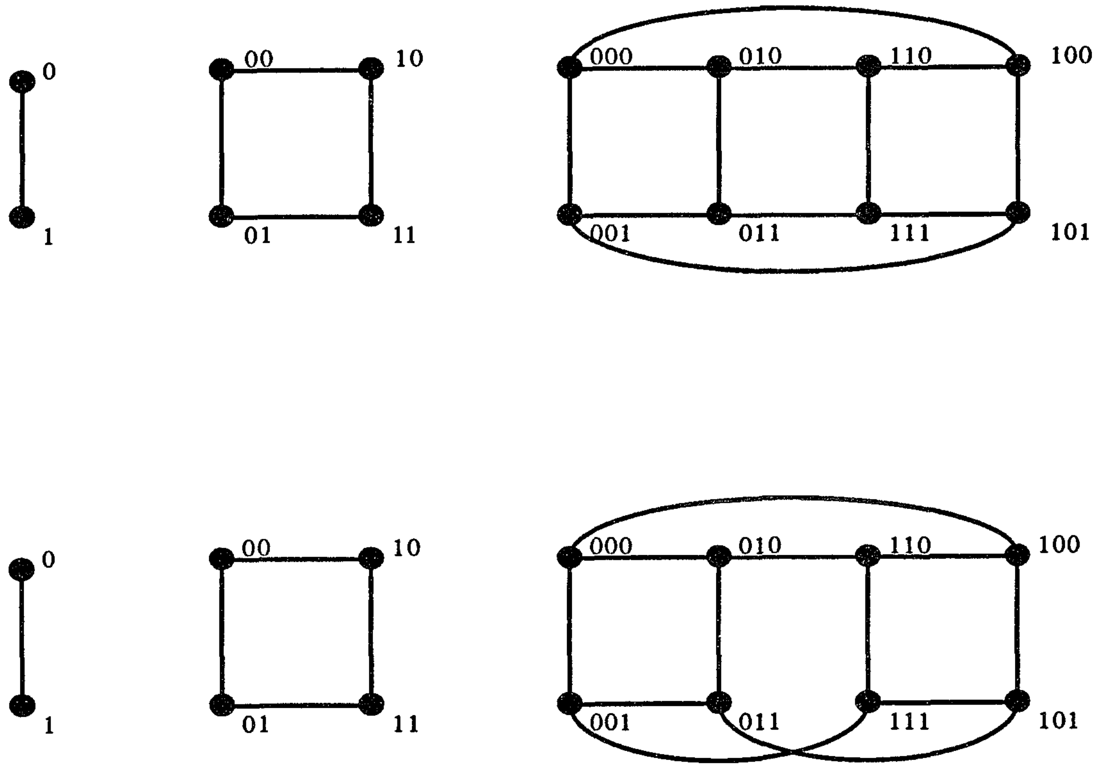


Figure 1.3: Hypercubes and twisted hypercubes for $n = 1, 2$, and 3.

The importance of complete binary trees comes from the fact that this class of structures is useful in the solution of banded and sparse systems by direct elimination and capture the essence of divide and conquer algorithms ([BI], [Gor], [HS]). A *complete binary tree* of height $n - 1$, denoted by $CB_n$, is an undirected graph consisting of $2^n - 1$ vertices, such that every vertex of depth less than $n - 1$ has exactly two children and every vertex of depth $n - 1$ is a leaf.

Quad trees are becoming an important representation technique in the domains of image processing, computer graphics, and robotics [Sa]. This representation is based on the principle of recursive decomposition. A *complete quad tree* of height $n-1$, denoted $CQ_n$, is an undirected graph consisting of $\frac{(4^n-1)}{3}$ vertices, such that every vertex of depth less than $n-1$ has exactly four children and every vertex of depth $n-1$ is a leaf.

Rings are another special kind of graphs that has many real world applications and are used in the solution of many computer science problems such as the passing token problem and the Hamiltonian circuit problem [I]. A *ring* of size $n$, denoted $R_n$, is an undirected graph consisting of $n$ vertices labeled from $v_1$ to $v_n$ such that node $v_i$ is a neighbor to node $v_{(i+1) mod\, n}$, $1 \leq i \leq n$.

## 1.6 Graph Embedding

In this dissertation, we use undirected graphs to model interconnection networks. Each vertex represents a processor and each edge a communication link between processors. The *embedding* of a guest graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$ is an injective mapping $f$ from $V_G$ to $V_H$, where $V_G$, $E_G$ and $V_H$, $E_H$ are the vertex and edge sets of $G$ and $H$, respectively.

Many computational problems in parallel processing can be formulated as graph embedding problems. Embedding one interconnection network into another is very useful in the area of parallel computing for portability of algorithms across various architectures, layout of circuits in VLSI, and mapping logical data structures into computer memories ([BMS], [Len]). Also, the problem of organizing computations on a network of processors can be formulated as a graph embedding problem [KS]. When a process can be naturally decomposed into a collection of subprocesses that can be executed simultaneously with occasional communication between them, a task graph can be constructed by denoting each subprocess by a node and each communication between two subprocesses during the computation by an edge.

The problem of simulating one interconnection network by another is a natural graph embedding problem. Usually, it is assumed that the host network can grow arbitrarily large. This assumption is not realistic and does not correspond to actual parallel machines. In the real world, a parallel machine has a fixed number of processors. Thus, the problem of efficiently simulating a large network is an important issue. This type of embedding is called *many-to-one*, where more than one node in the guest graph are mapped to a single node in the host graph. If the embedding maps a single node in the guest graph to more than one node in the host graph, then the embedding is *one-to-many*. In this dissertation the word embedding refers to *one-to-one embedding*, where a single node in the guest graph is mapped to exactly one single node in the host graph. Many variations of embeddings in interconnection networks have been studied in the literature ([AR], [BCGS], [BCLR], [BI], [BLD], [BMS], [BSu], [DS], [JLD], [Lei], [LEI], [MS]). These variations differ principally in the optimization measures used in the embeddings.

## 1.7 Cost Functions

The quality of an embedding is often guided by some constraints which may differ from one application to another. The most common measures are dilation, expansion, edge congestion, and load factor [HMR]. If $u$ and $v$ are two adjacent nodes in $G$, denoted $u - v$, then the distance from $u$ to $v$, $d = (u, v)$, is the length of the shortest path from $u$ to $v$. The *dilation* $D$ is the maximum distance in $H$ between the images of adjacent vertices of $G$

$$D = \max \{d(f(u), f(v)), \text{ where } u - v \in E_G\}$$

The *expansion* $E$ is the ratio of the cardinality of the host vertex set to the cardinality of the guest vertex set

$$E = \frac{|V_H|}{|V_G|}$$

Minimizing each of these measurements has a direct implication on the quality of the simulation of the guest network by the the corresponding host network. The dilation of an embedding measures how far apart neighboring guest processors are placed in the host network. Clearly if adjacent guest processors are placed far apart in the host network, then there will be a significant degradation in simulation due to the long length of the communication path between them. The expansion of an embedding measures how much larger is the host network than the guest network during the simulation. We want to minimize expansion, as we want to use the smallest possible host network that has at least as many processors as in the guest network.

In reality, we usually have a fixed size host network and we may have to consider many-to-one embedding for larger guest networks. When the size of the guest network is not equal to the size of the host network in terms of the number of processors, then we try to find the smallest host network that has at least as many processors as the guest network. Such a host network is referred to as the *optimal host network.* There is a trade off between dilation, which measures the communication delay, and expansion, which measures processor utilization, such that one can achieve lower expansion at a cost of greater dilation and vice versa.

Another cost measure is the congestion which is the maximum number of edges of the guest graph routed through a single edge of the host graph. Edge congestion is a measurement of possible degradation due to communication delay. If a particular link in the the host network is needed for several different communication messages, then the messages will suffer some delay time since the link can't pass more than one message at a time. This will add extra time to the communication cost between processors.

In embeddings that are many-to-one maps, an important measure is load factor which is the maximum number of guest processors to be simulated by a single processor in the host interconnection network. This has been considered by many researchers, for instance ([BL], [DS]). Clearly, it is very important to minimize the load factor in the simulation of one network by another, as the distinct processors in the guest network assigned to the same processor in the host network will be running sequentially. An unbalanced processor load will degrade the simulation time as lightly used processors must wait for heavily used processors to finish their tasks. Thus, the amount of time needed to simulate one step of the guest network is proportional to the maximum number of processors assigned to the same host network.

## 1.8 Fault-Tolerance

One of the most important issues related to parallel machines is fault-tolerance. As the number of processors in parallel machines becomes larger, models without faults are becoming increasingly unrealistic. A *fault* is a processor or a link that fails. We use a strong fault model where a faulty node can neither compute nor communicate with its neighbors. A node fault will completely distroy the node and all links incident to it. We model a faulty link by making one of the nodes incident to the link faulty. An interconnection network containing faulty components is called a *faulty network* and a one without faulty components is called a *fault-free network.*

Fault-tolerant network architectures have emerged as an important area of study in parallel processing ([AGr], [BS], [CL], [HLNa], [HLNb], [LBT], [PM], [WCM]). The *fault-tolerance of a network* is the capability of the network to compute, route, simulate other networks, and reconfigure itself in the presence of faults. Clearly, if all the immediate neighbors of a nonfaulty node become faulty, the network will become disconnected. Many researchers studied the implementation of algorithms that are designed for fault-free machines on faulty machines. The efficiency of the implementation is usually measured by its slowdown. The *slowdown S* is the ratio between the algorithm's time requirements on the faulty machine and the algorithm's time requirements on the fault-free machine

$$S = \frac{Time\ of\ algorithm\ on\ a\ faulty\ machine}{Time\ of\ algoritm\ on\ a\ fault-free\ machine}$$

A significant difference between multiprocessor machines and other parallel machines is that these machines use message passing instead of shared memory for communication between processors. Each processor has a private local memory. This type of architecture can be scaled up to a very large number of processors compared to multicomputer designs based on globally shared memory. This model has some desirable characteristics with respect to fault-tolerance and error confinement as well. A faulty processor can be prevented from corrupting data in other processors if the faults are detected quickly. Contrast this to a shared memory multiprocessor where a faulty processor can potentially write into any location in memory and thereby corrupt an entire system within a very short time.

One issue that is usually addressed in the design of fault-tolerance is the mechanism for detecting faulty processors. Many researchers suggested the use of off-line testing of each processor, assuming there is a set of functional tests that can be run by one processor on another. But it is very difficult to validate the completeness of the functional testing strategies. Also, off-line testing can only detect permanent faults. Intermediate and partial faults occur more frequently than permanent faults in parallel machines. In order to detect these faults, it is necessary to have some kind of concurrent fault detection features [AG]. This dissertation is not addressing the detection of faults. Therefore, the faults are assumed to be known in advance.

Massively parallel message passing machines are receiving increasing attention to meet the demand for high speed reliable computing. Hypercube interconnection networks have emerged as one of the most effective and popular network architectures for fault-free and faulty environments. The hypercube structure is highly fault-tolerant and can handle a reasonable amount of interprocessor message traffic. When one or more processors fail, the relatively large number of links often enables the nonfaulty processors to continue communicating with one another. The ability of hypercube machines to simulate, route, and reconfigure themselves in the presence of faults has been addressed by many researchers ([BS], [CL], [HLNa], [PM], [WCM]).

## 1.9 Outline of the Dissertation

The capabilities of the twisted hypercube as a parallel machine is demonstrated in Chapter 2. We show the capabilities of the twisted hypercube to provide efficient broadcasting and routing and to perform basic parallel computations. The communication time of several computations is reduced by a factor of two over that of the hypercube. These include sorting, matrix multiplication, and associative computations. Finally, an implementation of the parallel prefix operation on the twisted hypercube is presented.

In Chapter 3, we present dilation two and expansion one embeddings of complete binary trees and complete quad trees into twisted hypercubes. We introduce two different schemes to embed a complete binary tree into a twisted hypercube of approximately the same size. The first scheme uses a recursive technique to embed the complete binary tree $CB_n$ into the twisted hypercube $TQ_n$ based on the embedding of $CB_{n-1}$ into $TQ_{n-1}$. The second scheme uses the inorder labeling of the complete binary tree to embed it into the twisted hypercube. Finally, a recursive scheme to embed a complete quad tree into its optimal twisted hypercube is presented.

In Chapter 4, we present optimal algorithms for embedding a ring into a twisted hypercube with fault-free nodes, single faulty node, and multiple faults. We show that a twisted hypercube $TQ_n$ with $2^n$ nodes can simulate a ring $R_{2^n-f}$ with $2^n - f$ nodes in the presence of $f$ twisted hypercube faults. We use divide and conquer techniques and a new data structure called a cube to achieve our results.

Chapter 5 presents new techniques to embed a ring of size $2^n - 2f$ into a hypercube of dimension $n$ despite the presence of $f \leq 2^{n-3}$ faults. The basic idea behind our technique is to partition the whole structure into cubes, avoid the faults within the cubes by using unused links, and construct the whole ring by connecting adjacent cubes. Finally, we conclude with discussion and open problems in Chapter 6.

CHAPTER 2

# Parallel Computation on the Twisted Hypercube

This chapter addresses data communication and basic parallel computations on the twisted hypercube. This chapter is organized as follows. Section 1 reviews some of the work that has been done to show the capability of the twisted hypercube to perform efficient broadcasting and routing of data. Section 2 addresses some of the basic parallel operations on the twisted hypercube. Section 3 concludes the chapter.

## 2.1 Data Communication

One of the most important components of an interconnection network is its communication mechanism. In a parallel machine, communications become a bottleneck due to a great amount of time that is spent in interchanging information between different processors. It is very important to get the right data to the right place within a reasonable time.

Broadcasting is the most essential communication operation in an interconnection network. The height of the broadcast tree of a network is at most its diameter. Since the twisted hypercube reduces the diameter by a factor of two, the height of its broadcast tree is also reduced by a factor of two. The broadcast tree of any network can be easily found by running a breadth first algorithm. The breadth first spanning tree constructed by the breadth first algorithm represents the broadcast tree of the network [Lei]. Efe [E] and Zheng [Z] independently introduced broadcasting and routing algorithms for the twisted hypercube. Figure 2.1 shows the broadcast tree of a twisted hypercube for $n = 3$.

## 2.2 Basic Operations

This section demonstrates the ability of the twisted hypercube to perform many of the basic operations that are needed in designing parallel algorithms. These operations usually appear as subproblems in solving other major problems. Sorting is the most common subtask activity performed on parallel computers. It is the heart of many other computations. Many problems involve a sort so that later access of

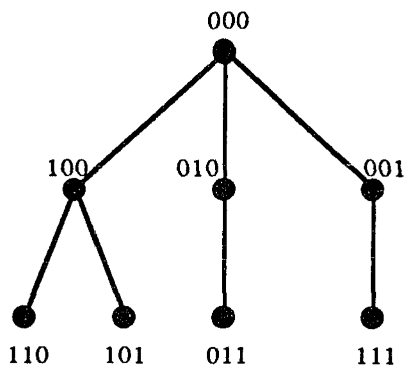


Figure 2.1: Broadcasting in a twisted hypercube.

information can be done efficiently. In [E], the author shows that the twisted hypercube can reduce the communication steps for the rank sort by a factor of two. The rank sort is considered to be the fastest sorting algorithm implemented on the hypercube machine. Like sorting, matrix multiplication is a fundamental operation that appears in many numerical computations. Efe [E] shows that the twisted hypercube can reduce the communication time of the matrix multiplication algorithm by a factor of two.

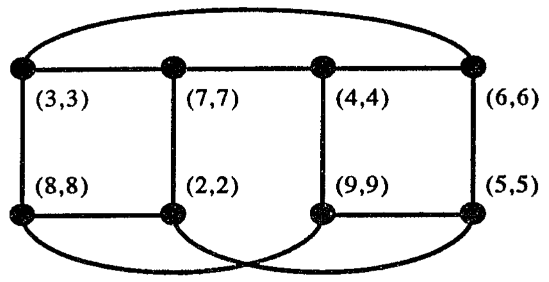
### 2.2.1 Associative Computations

Associative operations are used frequently and appear as subproblems in solving other problems. They include addition, multiplication, finding the smallest, finding the largest, and others. Let + be the addition operation on some domain $X$. For a given tuple $\{x_0, x_1, ..., x_{k-1}\} \in X$, the addition operation is to compute the summation $y_0 = x_0 + x_1 + ... + x_{k-1}$.

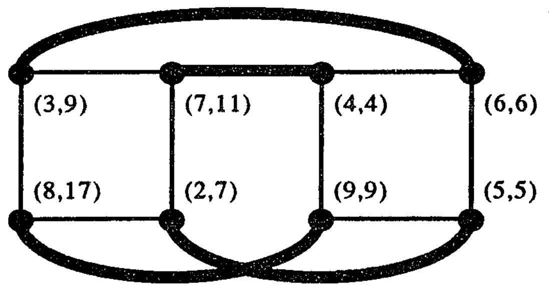
We assume that each processor $P_i$, $0 \le i \le 2^n - 1$, contains the value $x_i$. The computation is considered to be complete when the final summation $y_0$ is at processor 0. The symbol $\Leftarrow^j$ denotes a data transfer from a processor to an adjacent processor by a link through dimension $j$. The function BIT($j$) returns the $j^{th}$ bit of the node's label. The addition operation is performed by the following algorithm.

```
ADDITION ($X$)
begin
    for all $P_i$, $0 \le i \le 2^n - 1$, do
        $y_i \leftarrow x_i$
    for j $\leftarrow n$ to 1 do
        for all $P_i$, $0 \le i \le 2^j - 1$, do
            if BIT($j$) = 1
            then $temp_k \Leftarrow^j y_i$, where $P_k$ is a neighbor through dimension $j$.
            if BIT($j$) = 0
            then $y_i \leftarrow y_i + temp_i$
        end for
    end for
end
```

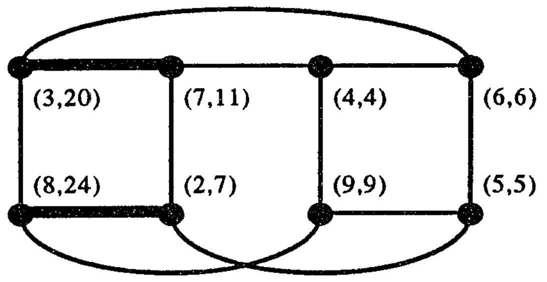
Figure 2.2 shows the addition operation on a twisted hypercube of dimension 3. The initial value $x_i$ and the current sum $y_i$ of each node are given for each phase. Algorithm ADDITION takes $n$ communication steps which is the same time that takes to run the same procedure in a hypercube machine. But since the height of the broadcast tree of the twisted hypercube is reduced by a factor of two, then the number of communication steps of the addition operation is also reduced by a factor of two ([E], [Z]). Since some of the nodes in the broadcast tree might have up to $n$ children, the binary adders must be replaced by $(n+1)$-adders. Figure 2.3 shows the implementation of the addition operation via the paths of the broadcast tree. The number of steps is reduced from 3 to 2 communication steps.



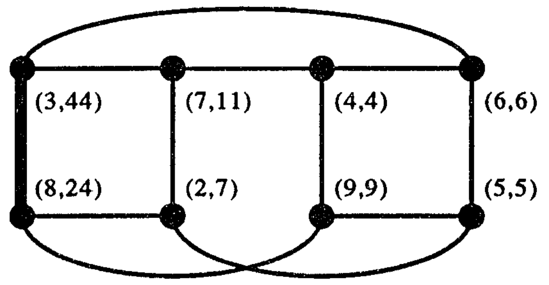
(a) Initial values



(b) After step 1

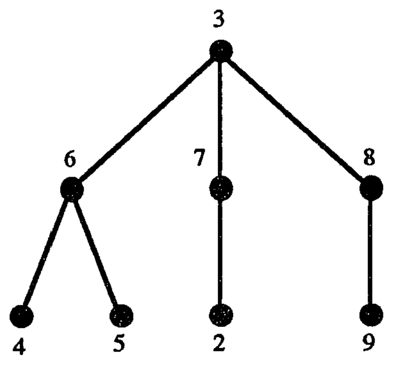


(c) After step 2

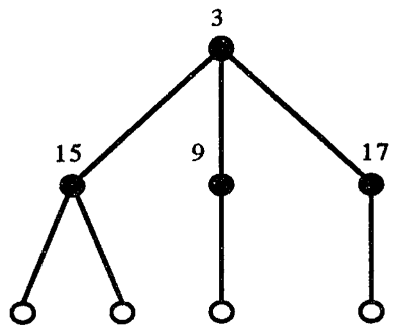


(d) After step 3

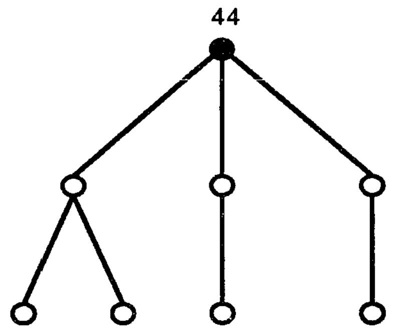
Figure 2.2: The addition operation on a twisted hypercube of dimension 3.



(a) Initial values



(b) After step 1



(c) After step 2

Figure 2.3: The addition operation via the broadcast tree.

## 2.2.2 Parallel Prefix

In this section, we implement the parallel prefix operation on a twisted hypercube. The prefix operation is a very important operation that appears frequently in designing parallel algorithms. It was first introduced by Ladner and Fischer [LF] to solve the carry look-ahead problem for binary addition. The prefix operation was used by many researchers to solve a variety of problems in the field of computer science. In [Lei], the prefix operation was used to solve recurrence equations, to find convex hulls of images, to route packets in interconnection networks, and to solve the problem of computing carries. In [A], the prefix sum was used to solve the job sequencing problem with deadlines and the knapsack problem. Plaxton [P] used the prefix operation to implement a fast sorting algorithm called smooth sort, which was designed to run on the hypercube.

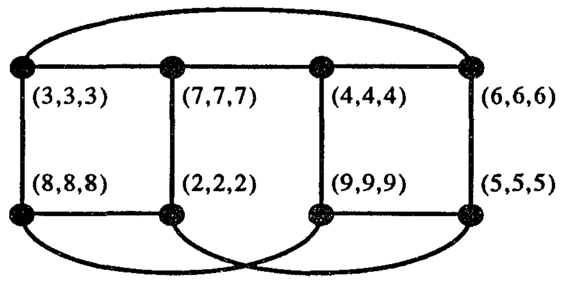
Let $\oplus$ be a binary associative operation on some domain $X$. For a given tuple $\{x_0, x_1, ..., x_{k-1}\} \in X$, the prefix problem is to compute each of the partial sums, assuming $\oplus$ is addition, $y_i = x_0 \oplus x_1 \oplus ... \oplus x_i$, $0 \le i \le k-1$. We assume that each processor $P_i$, $0 \le i \le 2^n - 1$, contains the value $x_i$. The computation is considered to be complete when the partial sum $y_i = x_0 \oplus x_1 \oplus ... \oplus x_i$ has been completed at processor $i$, $0 \le i \le 2^n - 1$. The local variables $y_i$ and $t_i$ accumulate the partial and total sums, respectively. The symbol $\overset{j}{\Leftarrow}$ denotes a data transfer from a processor to an adjacent processor by a link through dimension $j$. The function BIT($j$) returns the $j^{th}$ bit of the node's label.

```
PREFIX ($X$)
begin
    for all $P_i$, $0 \le i \le 2^n - 1$, do
        $y_i \leftarrow x_i$
        $t_i \leftarrow x_i$
    end for
    for j $\leftarrow$ 1 to $n$ do
        for all $P_i$, $0 \le i \le 2^n - 1$, do
            $temp_k \Leftarrow^j t_i$, where $P_k$ is a neighbor through dimension $j$.
            $t_i \leftarrow t_i \oplus temp_i$
            if BIT($j$) = 1
            then $y_i \leftarrow y_i \oplus temp_i$
        end for
    end for
end
```

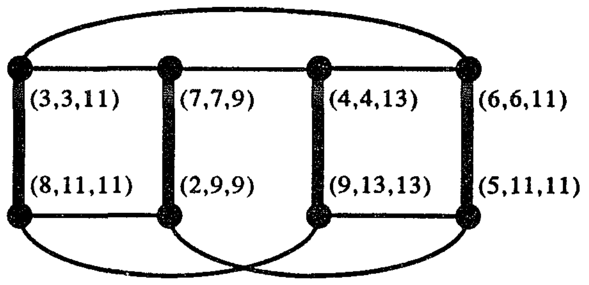
It is obvious that the algorithm runs in $n$ time steps, where $n$ is the dimension of the twisted hypercube. During the $j^{th}$ step, each node sends its current total sum to its adjacent node through dimension $j$. The partial and total sums of each node are updated based on the value of the $j^{th}$ bit of its label. Figure 2.4 shows the prefix computation on a twisted hypercube of dimension 3. The initial value $x_i$, the current partial sum $y_i$, and the current total sum $t_i$ of each node are given for each phase.

## 2.3 Summary

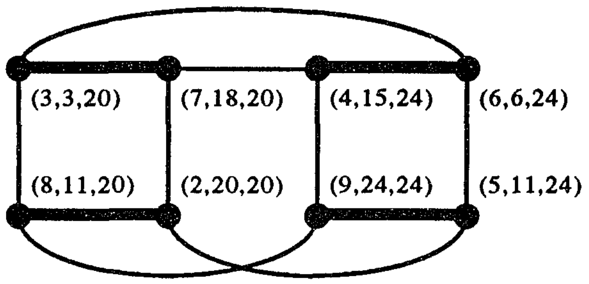
This chapter demonstrated the capabilities of the twisted hypercube as a parallel machine to provide efficient broadcasting and routing and to perform basic parallel computations. The communication time of several computations is reduced by a factor of two over that of the hypercube. These include sorting, matrix multiplication, and associative computations. Finally, an implementation of the parallel prefix



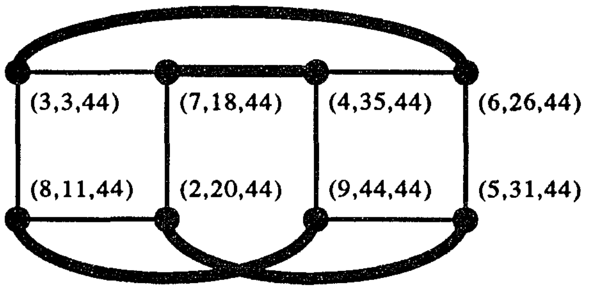
(a) Initial values



(b) After step 1



(c) After step 2



(d) After step 3

Figure 2.4: The prefix operation on a twisted hypercube of dimension 3.

operation on the twisted hypercube is presented. At the end of the computation, each processor will have its initial value, the partial sum, and the total sum.

CHAPTER 3

# Embedding Trees into Twisted Hypercubes

## 3.1 Introduction

Embedding trees into other interconnection networks attracted the attention of many researchers: in [BCLR], [BI], and [MS] embeddings trees into hypercubes were considered; in [BLD] the authors have considered embedding complete binary trees into hypercubes; in [LE1] the authors have considered embedding binary trees into 3-D mesh arrays; [DS] considered simulation of binary trees and X-trees on pyramid networks; [HJ] addressed embedding quad trees into hypercubes; and [KHI] considered a reconfigurable embedding of a complete quad tree into a faulty hypercube environment.

It is well known that the complete binary tree $CB_n$ with $2^n - 1$ nodes is not a subgraph of the hypercube $Q_n$ with $2^n$ nodes. This means that a unit dilation and unit expansion embedding from $CB_n$ into $Q_n$ is not possible. The proof is straightforward by the use of bipartite graphs. Both complete binary trees and hypercubes are bipartite graphs, their nodes can be assigned two colors so that adjacent nodes are not assigned the same color. Coloring $Q_n$ produces equal number of nodes in each color class, where coloring of $CB_n$ gives unequal number of nodes in each color class. Therefore,

$CB_n$ can't be a subgraph of $Q_n$ since it has more nodes in one color class than the number of nodes of $Q_n$ in the same color class, *i.e.*, $2^{n-1} + 2^{n-3} + \cdots > 2^{n-1}$.

The complete binary tree $CB_n$ can be embedded into $Q_n$ such that exactly one of its edges is assigned to a path of length two in the hypercube and all other edges are assigned to paths of length one in the hypercube. So, $CB_n$ can be embedded into $Q_n$ with dilation two and expansion one ([BCLR], [BI], [Lei], [W]). Bhatt and Ispen [BI], Barasch *et. al.* [BLD], and Wu [W] gave recursive dilation two and expansion one embeddings of complete binary trees into hypercubes based on a structure called two-rooted complete binary tree.

It is an open problem whether all binary trees can be embedded into their optimal hypercube with dilation two or into their next to optimal hypercube with dilation one. Bhatt *et. al.* [BCLR] showed that arbitrary binary trees can be embedded into hypercubes with constant expansion and dilation 10. The constants were subsequently reduced by Monien and Sudborough [MS], giving a dilation 5 and expansion one embedding and a dilation 3 and constant expansion embedding.

In this chapter we introduce different schemes to embed complete binary trees and complete quad trees into twisted hypercubes ([ABa], [ABe]). The remainder of this chapter is organized as follows. Section 2 describes two different schemes to embed a complete binary tree into a twisted hypercube of the same size. Section 3 introduces a recursive technique to embed a quad tree into its optimal twisted hypercube. Section 4 concludes the chapter.

## 3.2 Embedding Complete Binary Trees into Twisted Hypercubes

This section describes our schemes to embed a complete binary tree $CB_n$ into a twisted hypercube $TQ_n$ with dilation two and unit expansion. In the first scheme, we use a recursive algorithm to embed $CB_n$ into $TQ_n$ based on the embedding of $CB_{n-1}$ into $TQ_{n-1}$. In the second scheme, we use the inorder labeling to embed $CB_n$ into $TQ_n$.

### 3.2.1 The Recursive Embedding

The complete binary trees $CB_1$, $CB_2$, $CB_3$, and $CB_4$ can be embedded with dilation one into $TQ_1$, $TQ_2$, $TQ_3$, and $TQ_4$, respectively, as shown in Figure 3.1. The complete binary tree $CB_5$ can be embedded with dilation two into $TQ_5$ as shown in Figure 3.2. For $n > 5$, we use a recursive algorithm to embed $CB_n$ into $TQ_n$ based on the embedding of $CB_{n-1}$ into $TQ_{n-1}$. The base of the recursive algorithm is $CB_5$.

We proceed in four steps. In the first step, $CB_n$ is partitioned to a left complete binary subtree $LCB_{n-1}$ with root $lr$, a right complete binary subtree $RCB_{n-1}$ with root $rr$, and a root $r$. In the second step, $TQ_n$ is partitioned to two subcubes, $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. In the third step, $LCB_{n-1}$ is embedded into $TQ^0{}_{n-1}$, $RCB_{n-1}$ is embedded into $TQ^1{}_{n-1}$, and $r$ is embedded into the extra unused node in $TQ^1{}_{n-1}$. In the fourth step, we construct $CB_n$ by joining $LCB_{n-1}$, r, and $RCB_{n-1}$. This is done by finding the paths $r{\sim}lr$ and $r{\sim}rr$, each of length two. Our embedding is such that all edges in the lowest four levels in the complete binary tree $CB_n$ are mapped to paths of length one in the

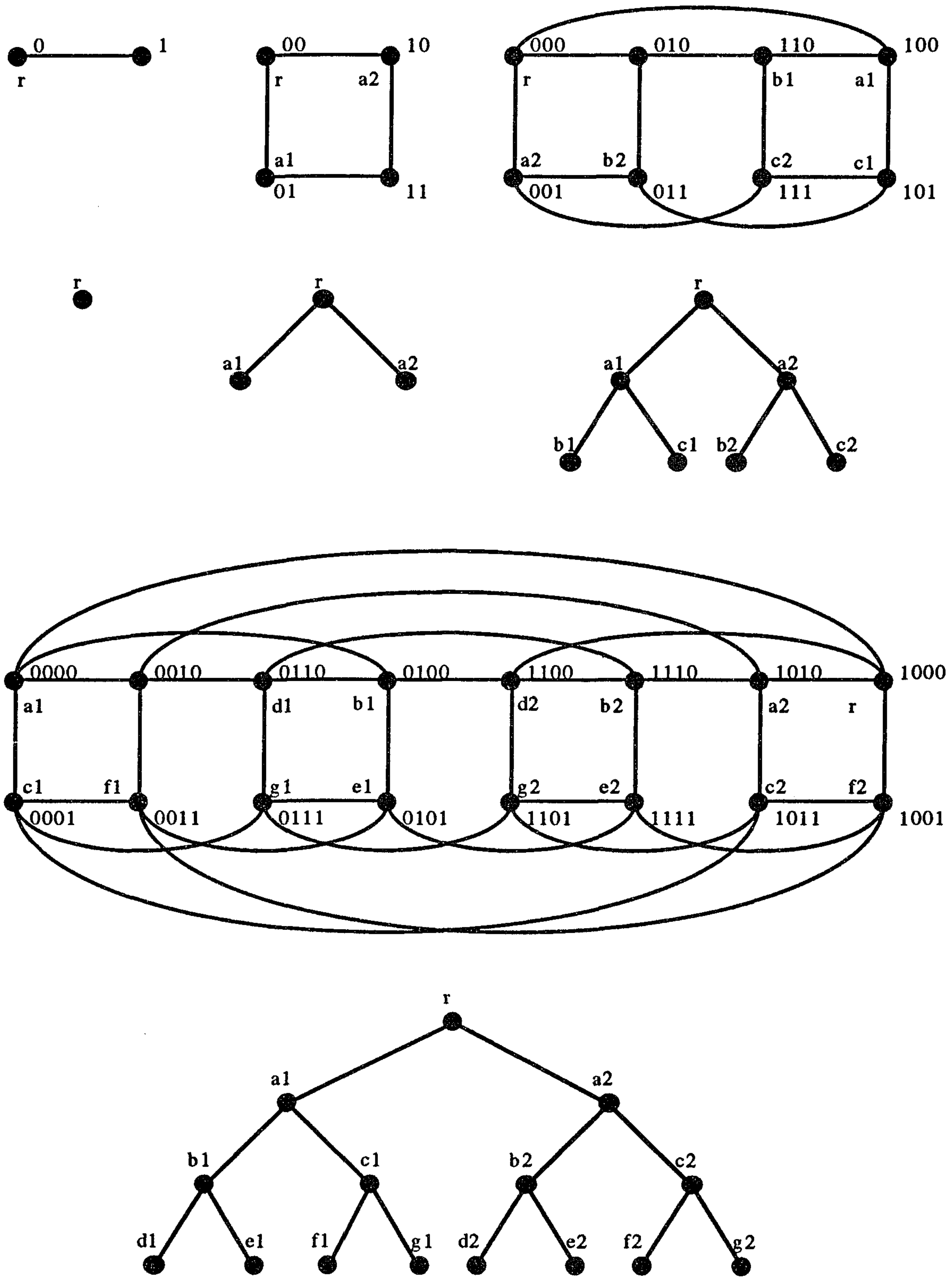


Figure 3.1: Embedding *CB* into *TQ* for $n$ = 1, 2, 3, and 4.

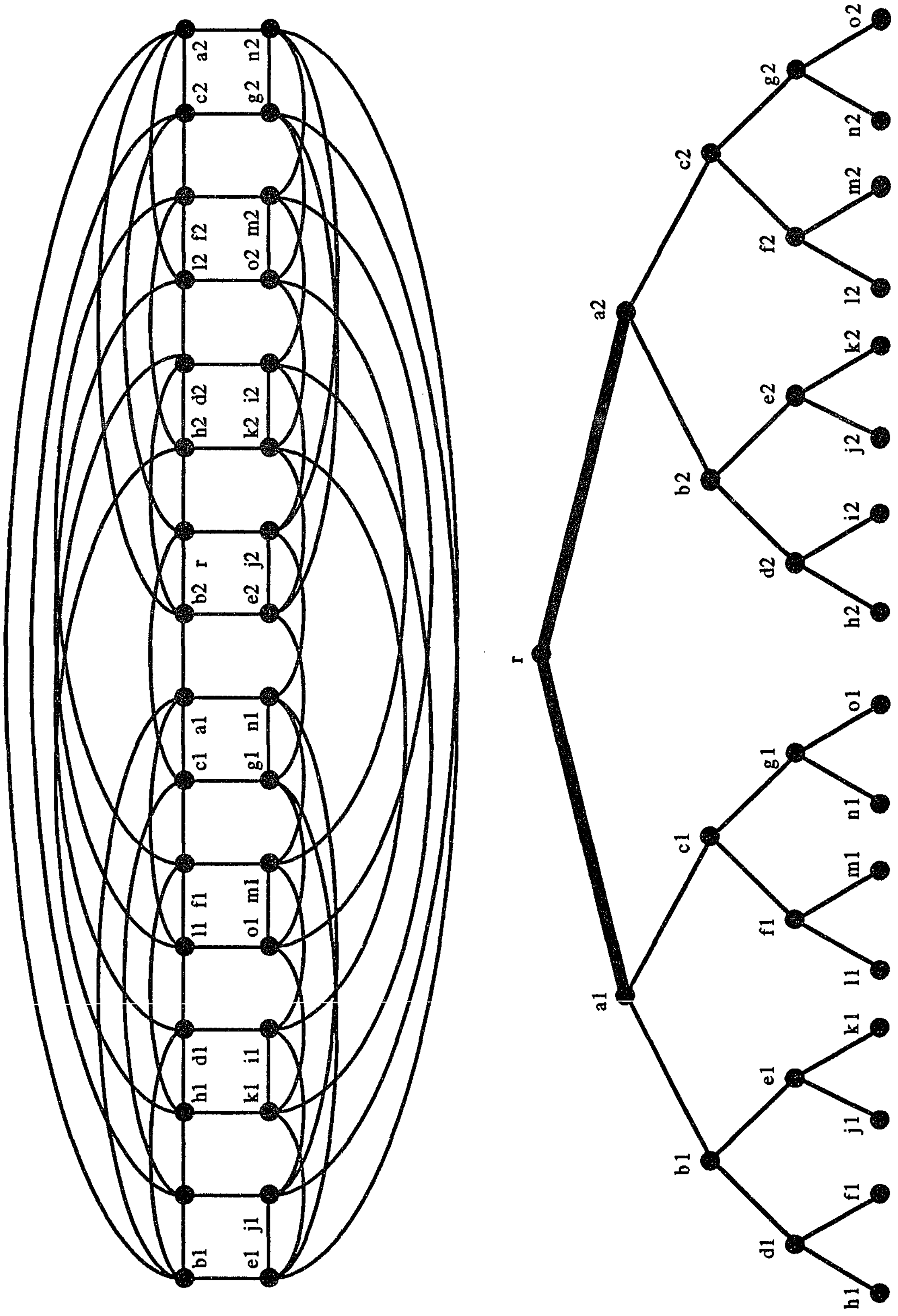


Figure 3.2: Embedding *CB* into *TQ* for $n = 5$.

twisted hypercube $TQ_n$ and all other edges in higher levels are mapped to paths of length two as shown in Figure 3.3. Now we present a formal description of the recursive algorithm described above.

**Algorithm 3.1**

Let $\delta_i$ be the binary string of length $n$ with a 1 in position $i$ and 0 in all other positions, $\theta_k$ be the binary string of length $k$ with 0 in all positions, and $\oplus$ be the XOR operator.

For $n = 1, 2, 3$, and 4, a dilation one embedding is shown in Figure 3.1. For $n = 5$, a dilation two embedding is shown in Figure 3.2. For $n > 5$, the algorithm is as follows.

**Step 1:** Partition $CB_n$ to $LCB_{n-1}$, $RCB_{n-1}$, and $r$.

**Step 2:** Partition $TQ_n$ to $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$.

**Step 3:** (i) Embed $LCB_{n-1}$ and $RCB_{n-1}$ into $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$, respectively. $lr$ and $rr$ will appear at addresses $011\theta_{n-5}10$ and $111\theta_{n-5}10$, respectively.

(ii) Translate the embedding in $TQ^1{}_{n-1}$ by complementing the $(n-1)^{th}$ bit of each node. Formally, if a tree node was embedded at address $x$ then after the translation it will appear at address $x\oplus\delta_{n-1}$. The root $rr$ will appear at address $rr\oplus\delta_{n-1}$, *i.e.*, $rr$ will appear at address $101\theta_{n-5}10$. The extra unused node $u$ will appear at address $u\oplus\delta_{n-1}$,*i.e.*, $u$ will appear at address $110\theta_{n-5}10$.

(iii) Embed the root $r$ into the unused node $110\theta_{n-5}10$ in $TQ^1{}_{n-1}$.

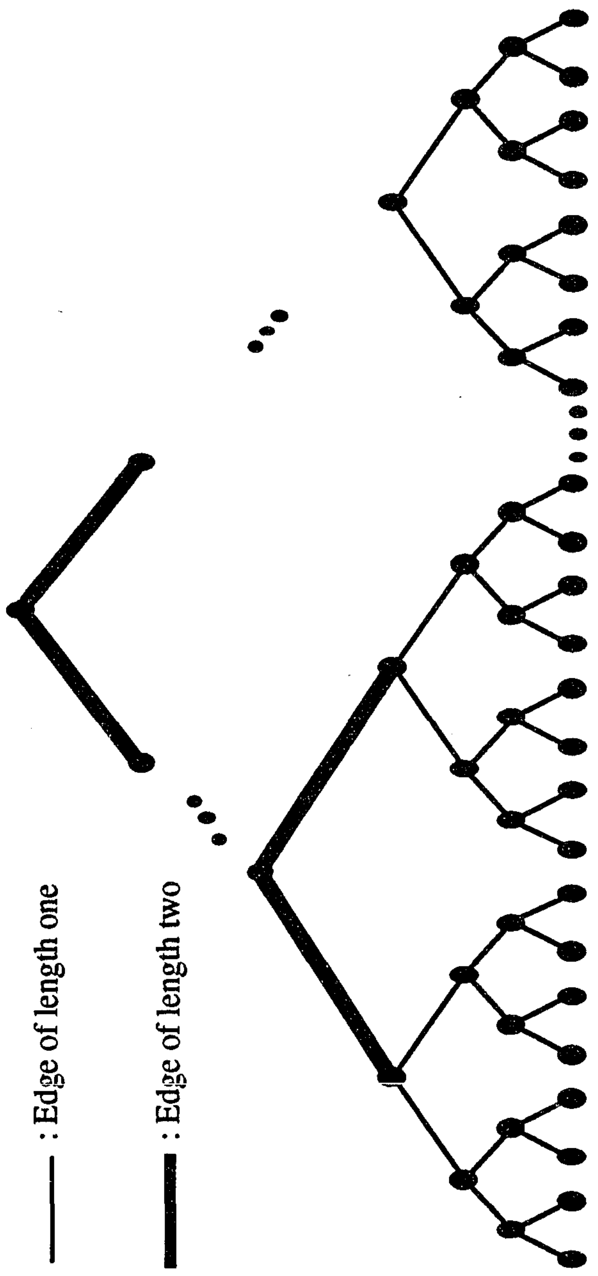


Figure 3.3: The recursive embedding of *CB* into *TQ*.

**Step 4:** Construct $CB_n$ from $LCB_{n-1}$, $r$, and $RCB_{n-1}$ by finding the shortest two paths $r \sim lr$ and $r \sim rr$, each of length two. Let $x$ and $y$ be the extra nodes that $r \sim lr$ and $r \sim rr$ go through, respectively. $x$ will appear at address $r \oplus \delta_n$ and $y$ will appear at address $r \oplus \delta_{n-2}$. The shortest paths from $r$ to $lr$ and from $r$ to $rr$ are $110\theta_{n-5}10 - 010\theta_{n-5}10 - 011\theta_{n-5}10$ and $110\theta_{n-5}10 - 111\theta_{n-5}10 - 101\theta_{n-5}10$, respectively.

**Theorem 3.1.** For all $n$, Algorithm 3.1 embeds the complete binary tree $CB_n$ within the twisted hypercube $TQ_n$ with dilation two.

*Proof:* For $n \leq 4$, the existence of an embedding with dilation one is shown in Figure 3.1. For $n = 5$, the existence of an embedding with dilation two is shown in Figure 3.2. For $n > 5$, we prove this by induction on the height of the binary tree. Our induction basis is $CB_5$. Assume the theorem is true for an embedding of $CB_{n-1}$ in $TQ_{n-1}$. We now prove that the theorem is true for the embedding of $CB_n$ into $TQ_n$. In $TQ_n$, consider the two subcubes $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. By induction hypothesis, there exist a dilation two embedding of $CB_{n-1}$ into $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. We assume that the two embeddings are isomorphic, one is obtained from the other by complementing the $(n-1)^{th}$ bit. Since the number of nodes in $CB_{n-1}$ is less than the number of nodes in $TQ_{n-1}$ by one, then $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$ contain two extra unused nodes located at addresses $000\theta_{n-5}10$ and $110\theta_{n-5}10$, respectively. Now we can use the extra unused node in $TQ^1{}_{n-1}$, the $CB_{n-1}$ of $TQ^0{}_{n-1}$, and the $CB_{n-1}$ of $TQ^1{}_{n-1}$ to construct the complete binary tree $CB_n$.

Next we prove that the dilation of this embedding is two. We use the routing algorithm of [EBSS] to show that the length of the shortest path from the root of $CB_n$ to any of its children is of length two. Let $r{\sim}lr$ be the shortest path from the root $r$ of $CB_n$ to the root $lr$ of the left complete binary subtree $LCB_{n-1}$ and $r{\sim}rr$ be the shortest path from the root $r$ of $CB_n$ to the root $rr$ of the right complete binary subtree $RCB_{n-1}$. $r$ will appear at address $110\theta_{n-5}10$, $lr$ at address $011\theta_{n-5}10$, and $rr$ at address $101\theta_{n-5}10$. Notice that if we group the addresses of $r$, $lr$, and $rr$ into pairs of bits, from right to left, then they are pair-related except for the left most three bits. By using the routing algorithm of [EBSS], the shortest paths from $110\theta_{n-5}10$ to $011\theta_{n-5}10$ and from $110\theta_{n-5}10$ to $101\theta_{n-5}10$ are $110\theta_{n-5}10 - 010\theta_{n-5}10 - 011\theta_{n-5}10$ and $110\theta_{n-5}10 - 111\theta_{n-5}10 - 101\theta_{n-5}10$, respectively. So, the dilation of this embedding is two. □

It can be proved easily that the edge congestion of this embedding is two. It is obvious that the edge congestion of the lowest four levels of the complete binary tree is one, since the dilation of the embedding is one. In the next higher level, only two hypercube edges are used twice as shown in Figure 3.2. In all higher levels, each edge from a parent to any of its children is mapped to a path of length two. Consider the shortest paths $r{\sim}lr$ and $r{\sim}rr$. The shortest path from the root $110\theta_{n-5}10$ to the left root $011\theta_{n-5}10$ is $110\theta_{n-5}10 - 010\theta_{n-5}10 - 011\theta_{n-5}10$ and from the root $110\theta_{n-5}10$ to the right root $101\theta_{n-5}10$ is $110\theta_{n-5}10 - 111\theta_{n-5}10 - 101\theta_{n-5}10$. Notice that the path $r{\sim}lr$ uses an edge through dimension $n$ from the root $r$ to an intermediate node $x$ and an edge through dimension *(n-3)* from $x$ to the left root $r$, while the path $r{\sim}rr$ uses an edge through dimension *(n-3)* from the root *r* to an intermediate node *y* and an edge through dimension *(n-1)* from *y* to the right root *rr*. Therefore, the maximum number of times a hypercube edge is used is two. So, the edge congestion of this embedding is two.

## 3.2.2 The Inorder Embedding

Another way to embed the complete binary tree $CB_n$ into the twisted hypercube $TQ_n$ is the inorder labeling of the complete binary tree as shown in Figure 3.4. The nodes of the complete binary tree are numbered inorder, each node of the complete

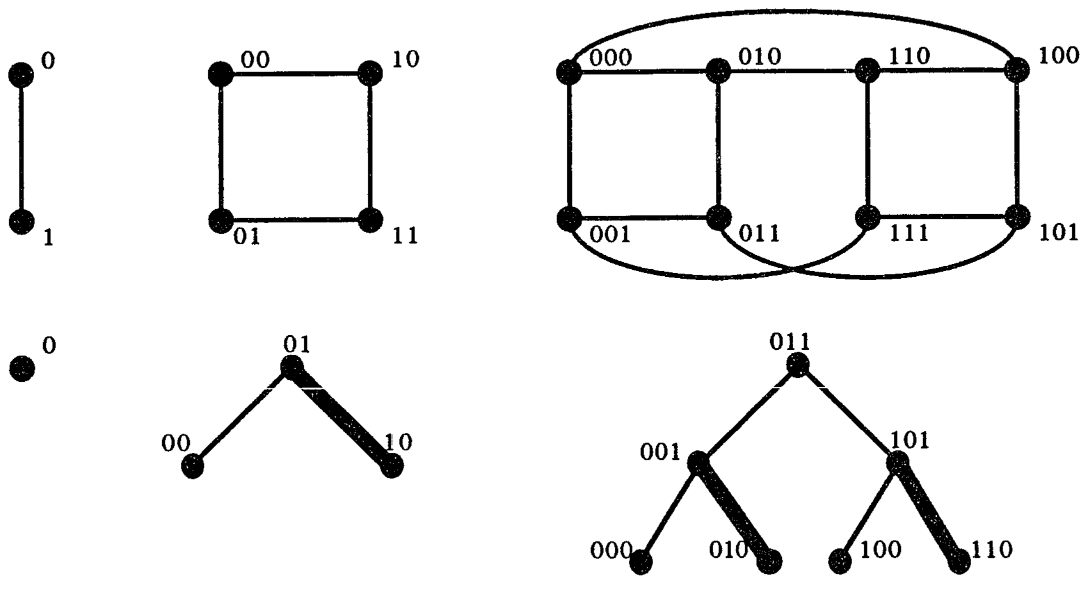


Figure 3.4: The inorder embedding of *CB* into *TQ* for $n$ = 1, 2, and 3.

binary tree is mapped to the node in the twisted hypercube with the corresponding address.

As illustrated in Figure 3.5, in the lowest level, each edge from a left child to its parent is mapped to the corresponding twisted hypercube edge between the images of the two nodes, while the edge between a right child to its parent is mapped to a path of length two, from the right child to the left child and from the left child to the parent. In the next level, each edge from a left child, or a right child, to its parent is mapped to the corresponding twisted hypercube edge between the images of the two nodes. In all higher levels, each edge from a left child, or a right child, to its parent is mapped to a path of length two. Notice that the inorder embedding is simpler, but it is less efficient in terms of the number of edges in the complete binary tree that are mapped to paths of length two in the twisted hypercube.

**Theorem 3.2.** For all $n$, the inorder labeling of the complete binary tree $CB_n$ embeds $CB_n$ within the twisted hypercube $TQ_n$ with dilation two.

*Proof:* Let $\beta_k$ be the binary string of length $k$ with 1 in all positions. For $n \leq 3$, the inorder embedding is shown in Figure 3.4. For $n > 3$, we prove the theorem by induction on the height of the binary tree. Our induction basis is $CB_3$, a dilation two embedding of $CB_3$ into $TQ_3$ is shown in Figure 3.4. Assume the theorem is true for an embedding of $CB_{n-1}$ in $TQ_{n-1}$. We now prove that the theorem is true for the embedding of $CB_n$ in $TQ_n$. In $TQ_n$, consider the two subcubes $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. By induction hypothesis, we can embed $CB_{n-1}$ into $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$, with dilation two.

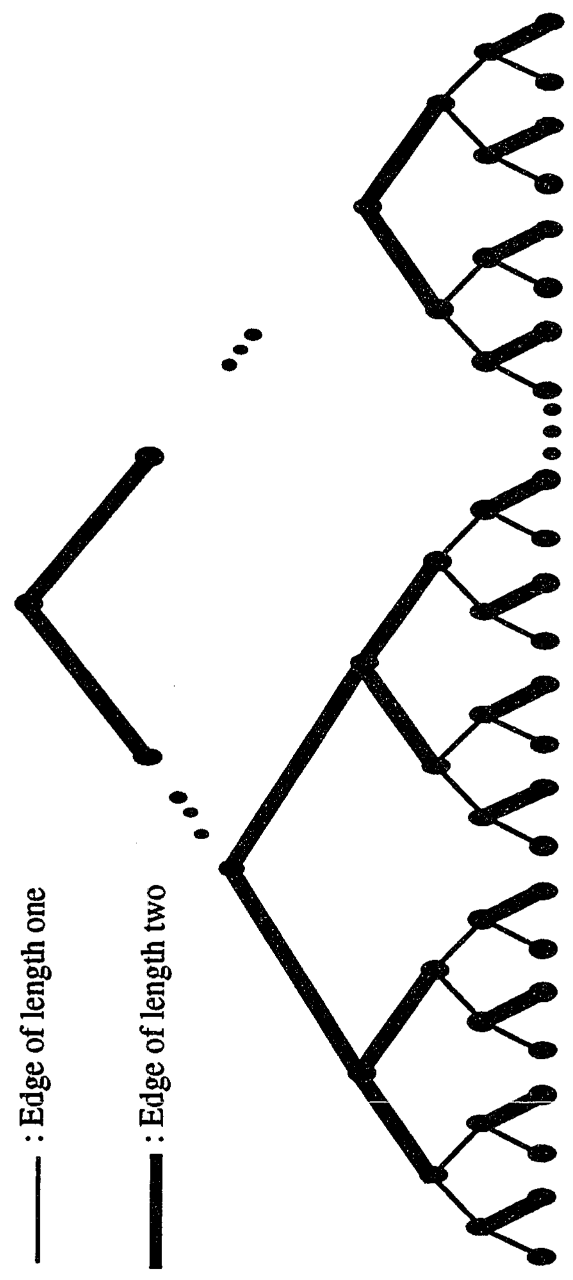


Figure 3.5: The inorder embedding of *CB* into *TQ*.

Since the number of nodes in $CB_{n-1}$ is less than the number of nodes in $TQ_{n-1}$ by one, then $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$ contain two extra unused nodes located at addresses $01\beta_{n-2}$ and $11\beta_{n-2}$, respectively. Now we can use the extra unused node in $TQ^0{}_{n-1}$, the $CB_{n-1}$ of $TQ^0{}_{n-1}$, and the $CB_{n-1}$ of $TQ^1{}_{n-1}$ to construct the complete binary tree $CB_n$ with $2^n - 1$ nodes.

Next we prove that the dilation of this embedding is two. We again use the routing algorithm of [EBSS] to show that the length of the shortest path from the root of $CB_n$ to any of its children is of length two. Let $r\sim lr$ be the shortest path from the root $r$ of $CB_n$ to the root $lr$ of the left complete binary subtree $LCB_{n-1}$ and $r\sim rr$ be the shortest path from the root $r$ of $CB_n$ to the root $rr$ of the right complete binary subtree $RCB_{n-1}$. $r$ will appear at address $01\beta_{n-2}$, $lr$ at address $00\beta_{n-2}$, and $rr$ at address $10\beta_{n-2}$. Notice that $r$, $lr$, and $rr$ are identical except for the left most two bits. By using the routing algorithm of [EBSS], the shortest paths from $01\beta_{n-2}$ to $00\beta_{n-2}$ and from $01\beta_{n-2}$ to $10\beta_{n-2}$ are of length two. So, the dilation of this embedding is two. □

It is obvious that the edge congestion of this embedding is two. In the lowest level, each edge from a parent to its left child is mapped to the corresponding twisted hypercube edge between the images of the two nodes, while the edge from a parent to its right child is mapped to a path of length two, from the parent to the left child and from the left child to the right child. This means that the only edge in this level that is used twice is the edge from a parent to its left child. In the next higher level, each edge from a child to its parent is mapped to the corresponding twisted hypercube edge between the images of the nodes, *i.e.*, each edge is used exactly once. In all higher levels, each edge from a child to its parent is mapped to a path of length two. Consider the shortest paths $r \sim lr$ and $r \sim rr$. Without loss of generality, consider the case when $n$ is even. The shortest path $r \sim lr$ from the root $01\beta_{n-2}$ to the left root $00\beta_{n-2}$ is $01\beta_{n-2}$-$00(01)^{\frac{n-2}{2}}$-$00\beta_{n-2}$ and the shortest path $r \sim rr$ from the root $01\beta_{n-2}$ to the right child $10\beta_{n-2}$ is $01\beta_{n-2}$-$11(01)^{\frac{n-2}{2}}$-$10\beta_{n-2}$, where $(01)^{\frac{n-2}{2}}$ means the repetition of the 01 pair $\frac{n-2}{2}$ times. Notice that the path $r \sim lr$ uses an edge through dimension *(n-1)* from the root $r$ to an intermediate node $x$ and an edge through dimension *(n-1)* from $x$ to the left root $r$, while the path $r \sim rr$ uses an edge through dimension $n$ from the root $r$ to an intermediate node $y$ and an edge through dimension *(n-1)* from $y$ to the right root $rr$. Therefore, the maximum number of times a hypercube edge is used is two. So, the edge congestion of this embedding is two.

## 3.3 Embedding Complete Quad Trees into Twisted Hypercubes

This section describes our scheme to embed a complete quad tree $CQ_n$ into its optimal twisted hypercube $TQ_{2n-1}$ with dilation two and expansion one. We proceed in four steps. In the first step, $CQ_n$ is partitioned into a left left complete quad tree $LLCQ_{n-1}$ with root *llr*, a left complete quad tree $LCQ_{n-1}$ with root *lr*, a right complete quad tree $RCQ_{n-1}$ with root *rr*, a right right complete quad tree $RRCQ_n$ with root *rrr*, and a root $r$ as shown in Figure 3.6. In the second step, $TQ_{2n-1}$ is partitioned into four subcubes $TQ^{00}_{2n-3}$, $TQ^{01}_{2n-3}$, $TQ^{11}_{2n-3}$, and $TQ^{10}_{2n-3}$. In the third step, $LLCQ_{n-1}$ is embedded into $TQ^{00}_{2n-3}$, $LCQ_{n-1}$ is embedded into $TQ^{01}_{2n-3}$, $RCQ_{n-1}$ is embedded

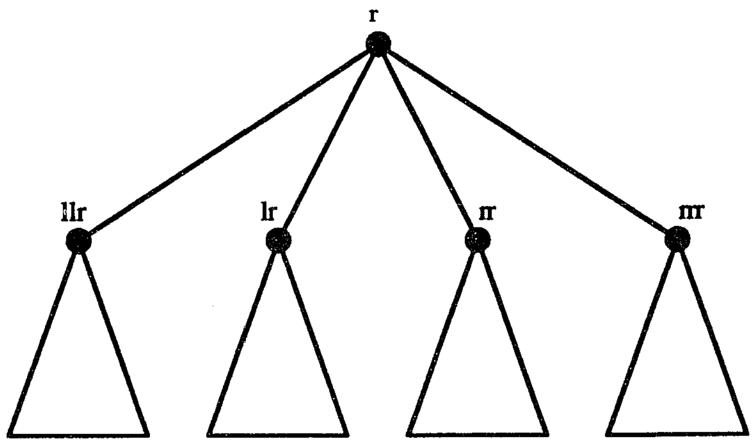


Figure 3.6: Partitioning *CQ*.

into $TQ^{11}{}_{2n-3}$, $RRCQ_{n-1}$ is embedded into $TQ^{10}{}_{2n-3}$, and the root $r$ is embedded into one of the unused nodes in $TQ^{00}{}_{2n-3}$. In the fourth step, we construct $CQ_n$ by finding the paths $r{\sim}llr$, $r{\sim}lr$, $r{\sim}rr$, and $r{\sim}rrr$, each of at most length two. The resulting embedding is such that only 37.5% of the edges in the lowest level of the complete quad tree and 50% of the edges in higher levels are mapped to paths of length two in the twisted hypercube. The rest of the edges of the complete quad tree are mapped to paths of length one in the twisted hypercube as shown in Figure 3.7. Now we present a formal description of the recursive algorithm described above.

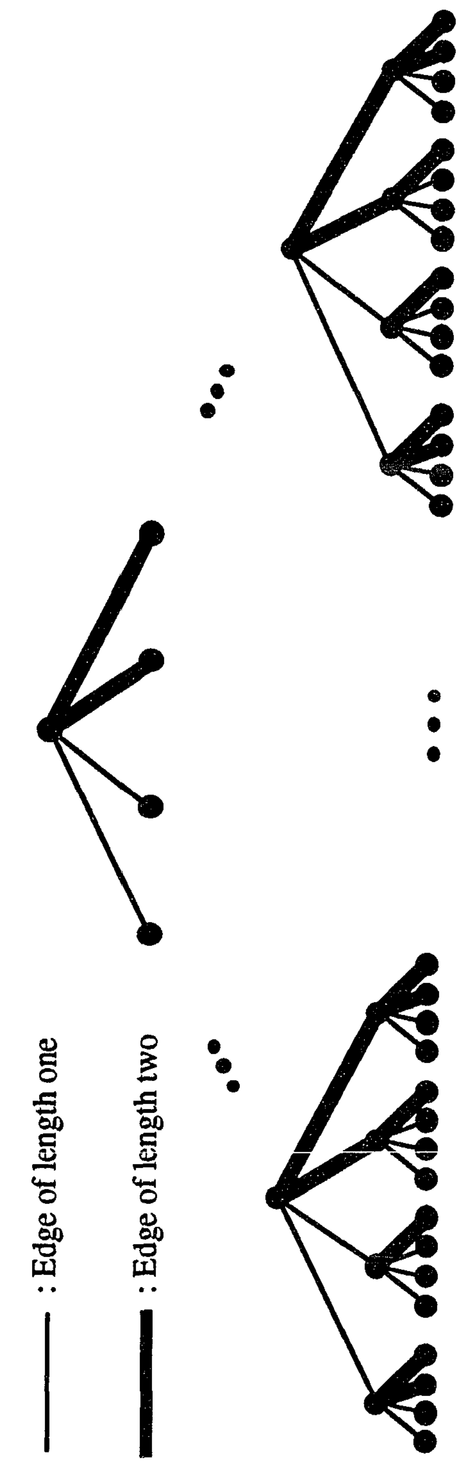
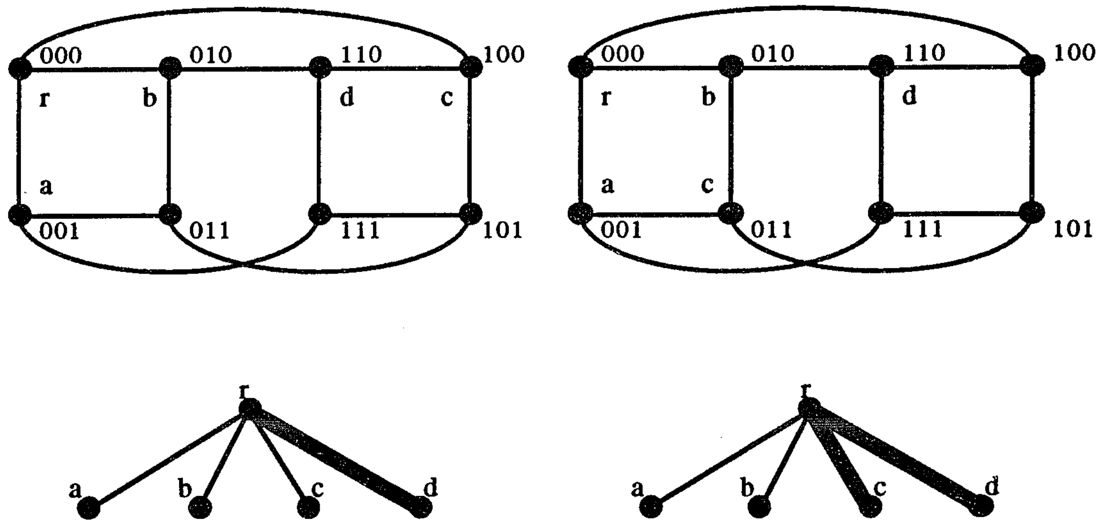


Figure 3.7: The recursive embedding of *CQ* into *TQ*.



(a) Standard (b) Alternate

Figure 3.8: Embedding $CQ$ into $TQ$ for $n = 2$.

**Algorithm 3.2**

Let $\delta_i$ be the binary string of length $n$ with a 1 in position $i$ and 0 in all other positions, $\theta_k$ be the binary string of length $k$ with 0 in all positions, and $\oplus$ be the XOR operator.

For $n = 1$, $CQ_1$ consists of exactly one node and can be embedded into $TQ_1$ with two nodes. For $n = 2$, a dilation two embedding is shown in Figure 3.8. For $n > 2$, the algorithm is as follows.

**Step 1:** Partition $CQ_n$ to $LLCQ_{n-1}$, $LCQ_{n-1}$, $RCQ_{n-1}$, $RRCQ_{n-1}$, and $r$.

**Step 2:** Partition $TQ_{2n-1}$ to $TQ^{00}{}_{2n-3}$, $TQ^{01}{}_{2n-3}$, $TQ^{11}{}_{2n-3}$, and $TQ^{10}{}_{2n-3}$.

**Step 3:** (i) Embed $LLCQ_{n-1}$ into $TQ^{00}{}_{2n-3}$, $LCQ_{n-1}$ into $TQ^{01}{}_{2n-3}$, $RCQ_{n-1}$ into $TQ^{11}{}_{2n-3}$, and $RRCQ_{n-1}$ into $TQ^{10}{}_{2n-3}$. $llr$, $lr$, $rr$, and $rrr$ will appear at addresses $000\theta_{2n-4}$, $010\theta_{2n-4}$, $110\theta_{2n-4}$, and $100\theta_{2n-4}$, respectively.

(ii) Translate the embeddings in $TQ^{00}{}_{2n-3}$ and $TQ^{10}{}_{2n-3}$ by complementing the $(2n-3)^{th}$ bit of each node. Formally if a tree node was embedded at address $x$ then after the translation it will appear at address $x \oplus \delta_{2n-3}$. After the translation the left left root $llr$ and the right right root $rrr$ will appear at addresses $001\theta_{2n-3}$ and $101\theta_{2n-4}$, respectively. Therefore, the final position of $llr$, $lr$, $rr$, and $rrr$ are $001\theta_{2n-4}$, $010\theta_{2n-4}$, $110\theta_{2n-4}$, and $101\theta_{2n-4}$, respectively.

(iii) Embed the root $r$ into the node with label 0 in $TQ^{00}{}_{2n-3}$.

**Step 4:** Construct $CQ_n$ from $LLCQ_{n-1}$, $LCQ_{n-1}$, $RCQ_{n-1}$, $RRCQ_{n-1}$, and $r$ by finding the four paths $r\sim llr$, $r\sim lr$, $r\sim rr$, and $r\sim rrr$. The edges $r-llr$ and $r-lr$ of $CQ_n$ are mapped to paths of length one in $TQ_{2n-1}$, while the edges $r-rr$ and $r-rrr$ are mapped to paths of length two. The shortest paths from $r$ to $rr$ and from $r$ to $rrr$ are $000\theta_{2n-4}-010\theta_{2n-4}-110\theta_{2n-4}$ and $000\theta_{2n-4}-100\theta_{2n-4}-101\theta_{2n-4}$, respectively.

**Theorem 3.3:** For all $n$, Algorithm 3.2 embeds the complete quad tree $CQ_n$ within the twisted hypercube $TQ_{2n-1}$ with dilation two.

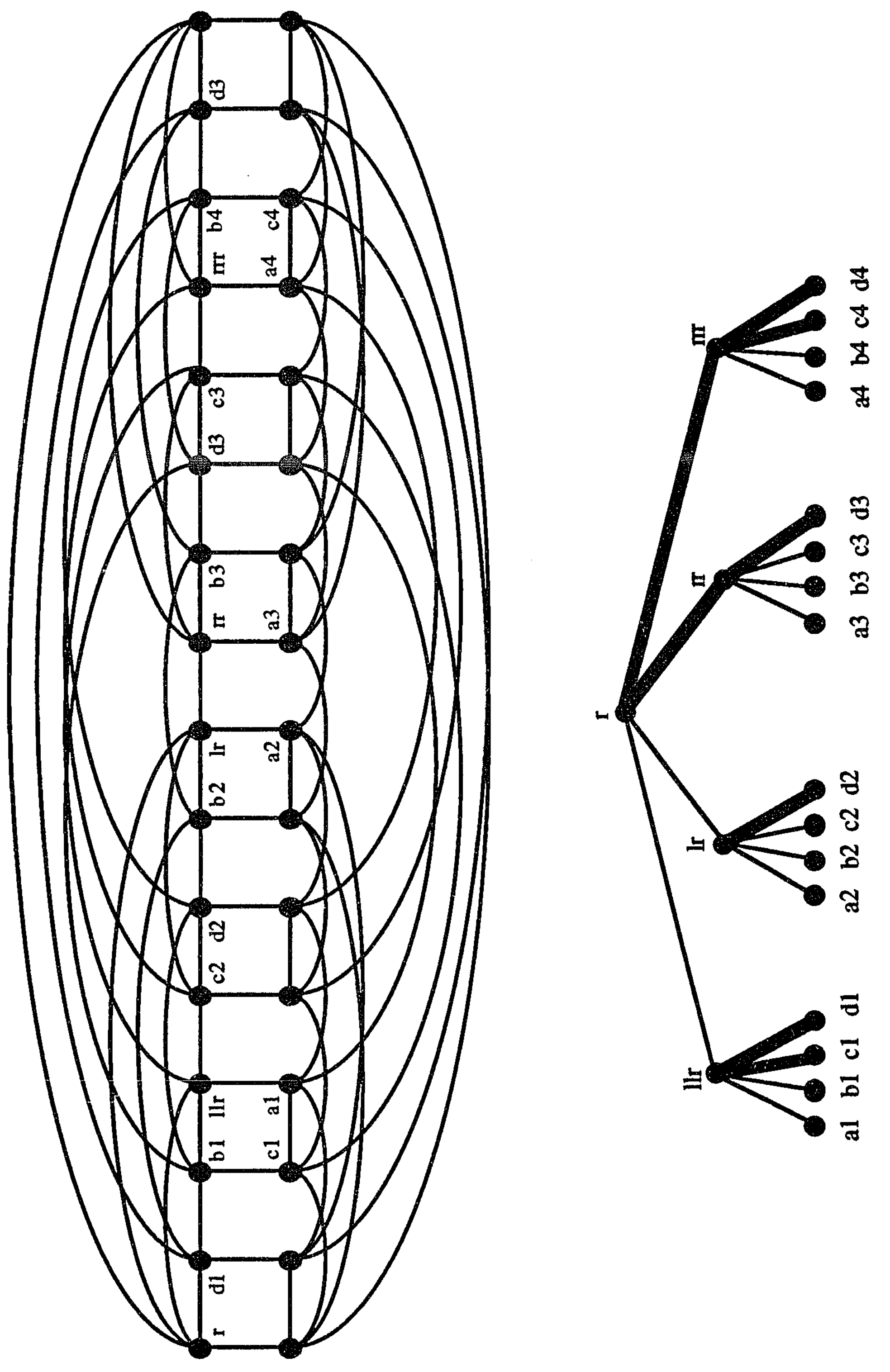


Figure 3.9: Embedding *CQ* into *TQ* for $n = 3$.

*Proof:* For $n = 1$, $CQ_1$ can be easily embedded into $TQ_1$. For $n = 2$, the existence of an embedding with dilation two is shown in Figure 3.8. For $n > 2$, we prove this by induction on the height of the complete quad tree $CQ_n$. Our induction basis is $CQ_3$, a dilation two embedding of $CQ_3$ into $TQ_5$ is shown in Figure 3.9. Assume the theorem is true for an embedding of $CQ_{n-1}$ in $TQ_{2n-3}$. We now prove that the theorem is true for the embedding of $CQ_n$ in $TQ_{2n-1}$. In $TQ_{2n-1}$, consider the four subcubes $TQ^{00}{}_{2n-3}$, $TQ^{01}{}_{2n-3}$, $TQ^{11}{}_{2n-3}$, and $TQ^{10}{}_{2n-3}$. By induction hypothesis, there exist a dilation two embedding from $CQ_{n-1}$ to $TQ^{00}{}_{2n-3}$, $TQ^{01}{}_{2n-3}$, $TQ^{11}{}_{2n-3}$, and $TQ^{10}{}_{2n-3}$. Since the number of nodes in $CQ_{n-1}$ is less than the number of nodes in $TQ_{2n-3}$, then $TQ^{00}{}_{2n-3}$, $TQ^{01}{}_{2n-3}$, $TQ^{11}{}_{2n-3}$, and $TQ^{10}{}_{2n-3}$ contain extra unused nodes. Now we can use the unused node with label 0 in $TQ^{00}{}_{2n-3}$, the $CQ_{n-1}$ of $TQ^{00}{}_{2n-3}$, the $CQ_{n-1}$ of $TQ^{01}{}_{2n-3}$, the $CQ_{n-1}$ of $TQ^{11}{}_{2n-3}$, and the $CQ_{n-1}$ of $TQ^{10}{}_{2n-3}$ to construct the complete quad tree $CQ_n$.

Next we prove that the dilation of this embedding is two. Thus, we need to show that the length of the shortest path from the root $r$ to any of its four children is at most two. Clearly, the length of the paths $r \sim llr$ and $r \sim lr$ is one since they are mapped to edges in the twisted hypercube. Let $r \sim rr$ be the shortest path from the root $r$ of $CQ_n$ to the root $rrr$ of the right complete quad subtree $RCQ_{n-1}$ and $r \sim rrr$ be the shortest path from the root $r$ of $CQ_n$ to the root $rrr$ of the right right complete quad subtree $RRCQ_{n-1}$. $r$ will appear at address $000\theta_{2n-4}$, $rr$ will appear at address $110\theta_{2n-4}$, and $rrr$ will appear at address $101\theta_{2n-4}$. Notice that if we group the addresses of $r$, $rr$, and *rrr* into pairs of bits, from right to left, then they are pair-related except for the left most three bits. By using the routing algorithm of [EBSS], the shortest path from $000\theta_{2n-4}$ to $110\theta_{2n-4}$ is $000\theta_{2n-4} - 010\theta_{2n-4} - 110\theta_{2n-4}$ and from $000\theta_{2n-4}$ to $101\theta_{2n-4}$ is $000\theta_{2n-4} - 100\theta_{2n-4} - 101\theta_{2n-4}$. So, the length of the paths *r~rr* and *r~rrr* are two. Therefore, the dilation of this embedding is two. □

It can be proved easily that the edge congestion of this embedding is two. It is obvious that the edge congestion of the lowest level of the complete quad tree is two, since one of the hypercube edges has to be used twice as shown in Figure 3.8. In all higher levels, each edge from a parent to any of its left children is mapped to a path of length one, while an edge from a parent to any of its right children is mapped to a path of length two. Clearly, the edge congestion of the paths *r~llr* and *r~lr* is one since their dilation is one. Now, consider the paths *r~rr* and *r~rrr*. The shortest path from the root $000\theta_{2n-4}$ to the right root $110\theta_{2n-4}$ is $000\theta_{2n-4} - 010\theta_{2n-4} - 110\theta_{2n-4}$ and from the root $000\theta_{2n-4}$ to the right right root $101\theta_{2n-4}$ is $000\theta_{2n-4} - 100\theta_{2n-4} - 101\theta_{2n-4}$. Notice that the path *r~rr* uses an edge through dimension *(n-1)* from the root *r* to an intermediate node *x* and an edge through dimension *n* from *x* to the right root *rr*, while the path *r~rrr* uses an edge through dimension *n* from the root *r* to an intermediate node *y* and an edge through dimension *(n-3)* from *y* to the right right root *rrr*. Therefore, the maximum number of times a hypercube edge is used is two. So, the edge congestion of this embedding is two.

## 3.4 Summary

In this chapter, two different schemes were used to embed the complete binary tree $CB_n$ into the twisted hypercube $TQ_n$. In the first scheme, we used a recursive algorithm to embed $CB_n$ into $TQ_n$ based on the embedding of $CB_{n-1}$ into $TQ_{n-1}$. The resulting embedding is such that all edges in the lowest four levels of the complete binary tree are mapped to paths of length one in the twisted hypercube and all other edges in higher levels of the complete binary tree are mapped to paths of length two in the twisted hypercube. In the second scheme, we used the inorder binary labeling of the complete binary tree $CB_n$ to embed $CB_n$ into the twisted hypercube $TQ_n$. The inorder embedding is simpler and more natural than the recursive embedding, but it is less efficient in terms of the number of edges that are mapped to paths of length two.

For complete quad trees, we used a recursive algorithm that embeds $CQ_n$ into $TQ_n$ based on the embedding of $CQ_{n-1}$ into $TQ_{n-1}$. The resulting embedding is such that 37.5% of the edges in the lowest level and 50% of the edges in higher levels of the complete quad tree are mapped to paths of length two in the twisted hypercube and the rest of edges are mapped to paths of length one.

CHAPTER 4

# Embedding Rings into Faulty Twisted Hypercubes

## 4.1 Introduction

The ability of a network to simulate, compute, route, and reconfigure itself despite the presence of faults is an important issue in parallel processing. The twisted hypercube was proposed as an alternative to the hypercube. One of the important features of the hypercube is its ability to simulate other networks in the presence of faults. If the twisted hypercube is considered as an alternative, it is necessary to show that its performance in the presence of faults is at least as good as that of the hypercube.

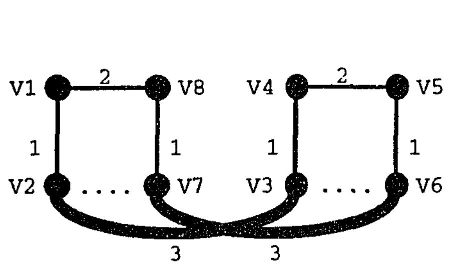
Rosenberg and Snyder [RS] showed that given any ring and any connected graph of the same size, the ring can be embedded into the graph with dilation cost $\leq 3$. They also proved that this bound is optimal. It is well known that rings can be embedded into hypercubes with dilation one using cyclic Gray Codes. Saad and Schultz [SS] used Gray Codes to embed a ring of size $l$ into a hypercube of size $2^n$ with dilation one when $l$ is even and $4 \leq l \leq 2^n$. Latifi and Zheng [LZ] generalized the cyclic Gray Code method to embed rings into twisted hypercubes. They identified $n!$ distinct Hamiltonian paths and $\frac{n!}{2} + (n-2)!$ distinct Hamiltonian circuits in a twisted hypercube.

Embedding rings into hypercubes in the presence of faults have been addressed by many researchers. Provost and Melhem [PM] have given distributed algorithms despite single, double, and multiple faults wasting up to 50% of the processors in the worst case. Chan and Lee [CL] improved the previous result by wasting only one nonfaulty processor for every faulty processor with some restriction on the number of faults. In this chapter, we consider the problem of embedding rings into twisted hypercubes in the presence of single and multiple faulty processors [ABb].

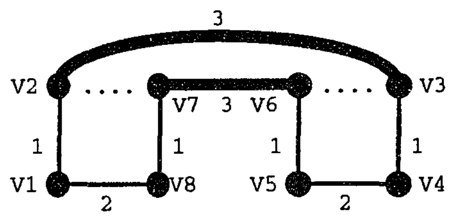
The remainder of this chapter is organized as follows. In section 2, we describe our schemes to embed a ring of size $2^n$ into a fault-free twisted hypercube of the same size. Section 3 addresses embeddings in the presence of faulty nodes. Our emphasis will be on the multiple fault case. Section 4 concludes the chapter.

## 4.2 Fault-Free Embeddings

Given a ring $R_{2^n}$ with $2^n$ nodes, consider the problem of assigning the ring nodes to the nodes of the twisted hypercube such that adjacency is preserved. That is, given any two adjacent nodes in the ring, their images by this embedding should be neighbors in the twisted hypercube through some dimension $i$, where $1 \leq i \leq n$. We can view such an embedding as a sequence of dimensions crossed by adjacent nodes. Let us call such a sequence the *embedding sequence*, denoted by ES = $(d_1, d_2, ..., d_{2^n})$, where $d_i \in \{1, ..., n\}$ for all $1 \leq i \leq 2^n$.



(a) Type A



(b) Type B

Figure 4.1: The embedding sequence.

### 4.2.1 The Embedding Sequence

Figure 4.1 shows an embedding of the ring $R_{2^3}$ into the twisted hypercube $TQ_3$. It is more convenient to view the embedded ring as well as the twisted hypercube in the way shown in Figure 4.1. All twisted hypercube nodes with even labels are in the upper level and all nodes with odd labels are in the lower level. The embedding sequence of $R_{2^3}$ is ES = (1, 3, 1, 2, 1, 3, 1, 2). For example, in Figure 4.1.a, notice that nodes $v_1$ and $v_2$ are connected by a link through dimension 1, $v_2$ and $v_3$ are connected by a link through dimension 3, $v_3$ and $v_4$ are connected by a link through dimension 1, $v_4$ and $v_5$ are connected by a link through dimension 2, and so on. The embedding sequence ES can be generated using the following algorithm.

**Algorithm 4.1**

Let $n$ be the dimension of the twisted hypercube and let the vertical bar be the concatenation operator.

```
Step 1:  ES ← 1
Step 2:  For i ← 3 to n do
             ES ← ES|i|ES
Step 3:  ES ← ES|2|ES|2
```

The embedding sequence is generated by applying Algorithm 4.1 on $n$, where $n$ is the dimension of the twisted hypercube. The number of nodes in the twisted hypercube is equal to the number of nodes in the embedded ring which is $2^n$ nodes. Thus, the embedding sequence of the ring $R_{2^4}$ is ES = (1, 3, 1, 4, 1, 3, 1, 2, 1, 3, 1, 4, 1, 3, 1, 2).

**Theorem 4.1:** For every $n$, Algorithm 4.1 will generate the embedding sequence to construct a ring of size $2^n$ in a fault-free twisted hypercube of dimension $n$.

*Proof:* We prove this by induction on the dimension of the twisted hypercube. Our induction basis is $TQ_2$, a ring of size 4 can be easily constructed in $TQ_2$ using the embedding sequence ES = (1, 2, 1, 2). Assume the theorem is true for the construction of a ring of size $2^{n-1}$ in a twisted hypercube of dimension $n$-1. We now prove that the theorem is true for the construction of $R_{2^n}$ in $TQ_n$. Consider the two twisted subcubes $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. By induction hypothesis, we can construct a ring of size $2^{n-1}$ in both $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. Let their embedding sequence be

$$\text{ES} = S_{n-1} \,|\, 2 \,|\, S_{n-1} \,|\, 2$$

where $S_{n-1}$ is a sequence of dimensions recursively defined as following

$$S_2 = 1$$

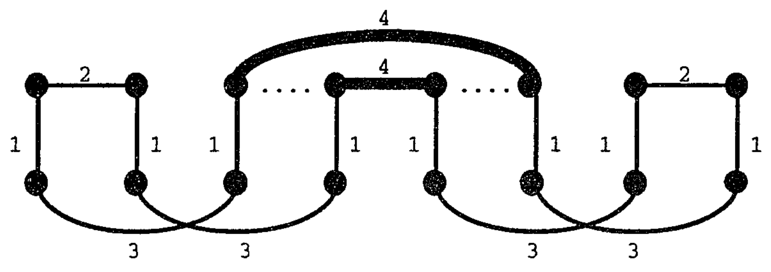
$$S_{n-1} = S_{n-2} \,|\, n \,|\, S_{n-2}$$

Now we combine two rings, each of size $2^{n-1}$, to come up with a ring of size $2^n$. This is done by replacing the first link that goes through dimension 2 of the first ring and the second link that goes through dimension 2 of the second ring by two links that go through dimension $n$. The embedding sequence of the new ring $R_{2^n}$ is

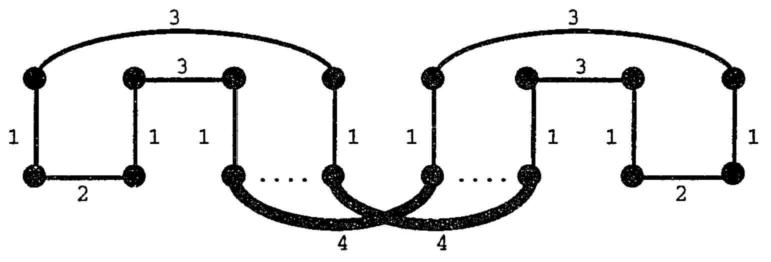
$$\begin{aligned} \text{ES} &= S_{n-1} \,|\, n \,|\, S_{n-1} \,|\, 2 \,|\, S_{n-1} \,|\, n \,|\, S_{n-1} \,|\, 2 \\ &= S_n \,|\, 2 \,|\, S_n \,|\, 2 \end{aligned}$$

which is the same embedding sequence generated by Algorithm 4.1. □

Notice that the same embedding sequence may result in different embeddings of $R_{2^n}$ into $TQ_n$ depending on the twisted hypercube node that initiates the ring construction. Among all different embeddings, we are interested in two kinds. The first embedding is when the node that initiates the ring construction in the twisted hypercube is the upper left most node, node with label 0. The second embedding is when the node that initiates the ring construction in the twisted hypercube is the lower left most node, node with label 1. Let us call the first embedding *type A embedding* and the second embedding *type B embedding.* Figure 4.2 shows both type A and type B embeddings for the ring $R_{2^4}$ into the twisted hypercube $TQ_4$.



(a) Type A embedding

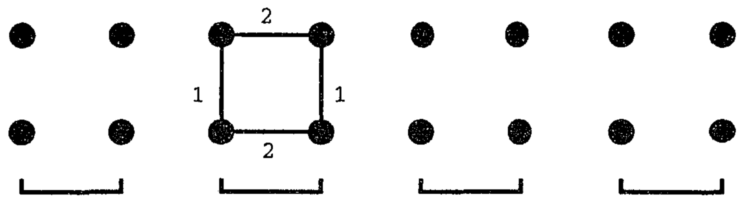


(b) Type B embedding

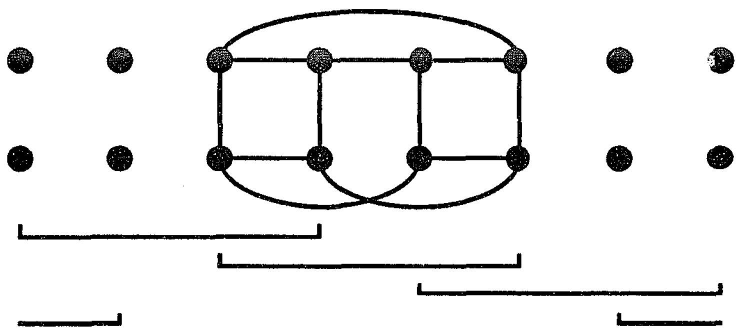
Figure 4.2: Fault-free embedding.

## 4.2.2 Divide-Conquer Embeddings

This section introduces a data structure, that is fundamental to the embeddings given in this chapter called a cube. A *cube* is a twisted subcube of dimension 3 that consists of two adjacent blocks as shown in Figure 4.3.b. A *block* is a set of four nodes in a twisted hypercube that form a ring of size 4 that has the embedding sequence ES = (1, 2, 1, 2) as shown in Figure 4.3.a. Notice that cubes overlap while blocks do not and a twisted hypercube of dimension $n$, $TQ_n$, contains $2^{n-2}$ cubes and



(a) The block



(b) The cube

Figure 4.3: Blocks and cubes.

$2^{n-2}$ blocks. A ring of size 8, $R_{2^3}$, can be embedded into a cube. In a cube, if we use the twisted lower links that go through dimension 3 to connect the two blocks, after removing the lower two links that go through dimension 2, then the embedding is of type A and if we use the upper links that go through dimension 3 to connect the two blocks, after removing the upper two links that go through dimension 2, then the embedding is of type B as shown in Figure 4.1. The cube is used in this section to introduce new techniques to embed a ring into a twisted hypercube. In the next section, this technique is generalized to embed a ring into a faulty twisted hypercube.

Now, given a ring $R_{2^n}$, we can embed it into the twisted hypercube $TQ_n$ by the following algorithm.

**Algorithm 4.2**

**Step 1:** Partition $TQ_n$ into $2^{n-3}$ node disjoint cubes.

**Step 2:** Embed the ring $R_{2^3}$ into each cube using type A, or type B, embedding.

**Step 3:** Connect the $2^{n-3}$ rings, each of size 8, through the upper links, or the twisted lower links, to come up with type A, or type B, embedding.

**Theorem 4.2:** For every $n$, Algorithm 4.2 will embed a ring of size $2^n$ in a fault-free twisted hypercube of dimension $n$.

*Proof:* We consider only type A embedding. Type B embedding can be proved in a similar way. We prove this by induction on the dimension of the twisted hypercube. Our induction basis is $TQ_3$, the embedding of a ring of size $2^3$ into a twisted hypercube of dimension 3 is shown in Figure 4.1.a. Assume the theorem is true for the construction of a ring of size $2^{n-1}$ in a twisted hypercube of dimension $n$-1. We now prove that the theorem is true for the construction of $R_{2^n}$ in $TQ_n$. Consider the two twisted subcubes $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. By assumption we can construct a ring of size $2^{n-1}$ in both $TQ^0{}_{n-1}$ and $TQ^1{}_{n-1}$. Now we combine two rings, each of size $2^{n-1}$, to come up with a ring of size $2^n$. This is done by replacing the first link that goes through dimension 2 of the first ring and the second link that goes through dimension 2 of the second ring by two upper links that go through dimension $n$. □

In the next section, we will use the same concept with minor variations to embed a ring to a faulty twisted hypercube without wasting any nonfaulty nodes.

## 4.3 Fault-Tolerant Embeddings

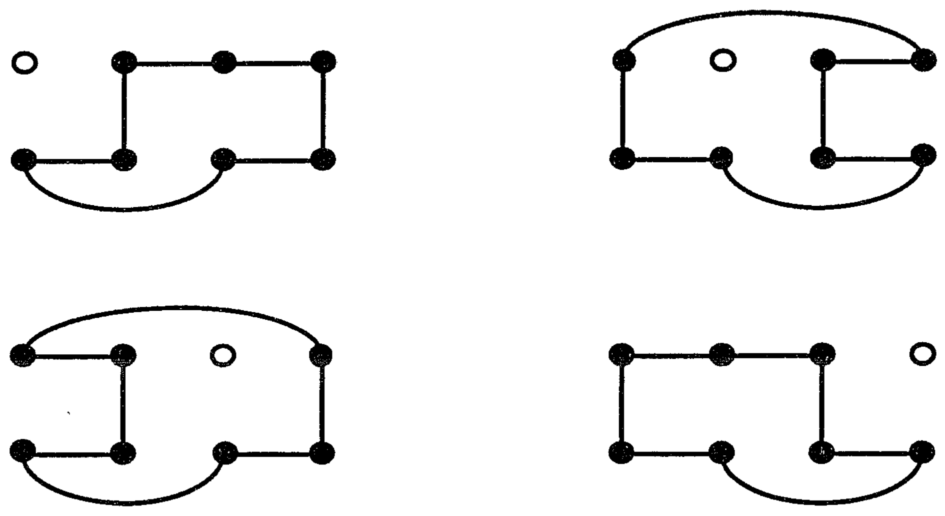
One of the special significant features of the hypercube is its capability to simulate other interconnection networks in the presence of faults. Accordingly, if the twisted hypercube is to be considered as an alternative, it is necessary to show that it is at least as good as the hypercube regarding fault-tolerance. In this section, we are interested in answering the following question. Given that some faults are present, does the twisted hypercube have the ability to simulate rings efficiently? Like the hypercube, the twisted hypercube is maximally fault-tolerant. While even one faulty processor in the twisted hypercube will degrade its overall performance, it is still capable of simulating rings without wasting any nonfaulty nodes. In the hypercube, you have to waste a nonfaulty node for every faulty node [CL]. In the next section, we extend Algorithm 4.2 to handle a single faulty node.

### 4.3.1 Embedding in the Presence of a Single Fault

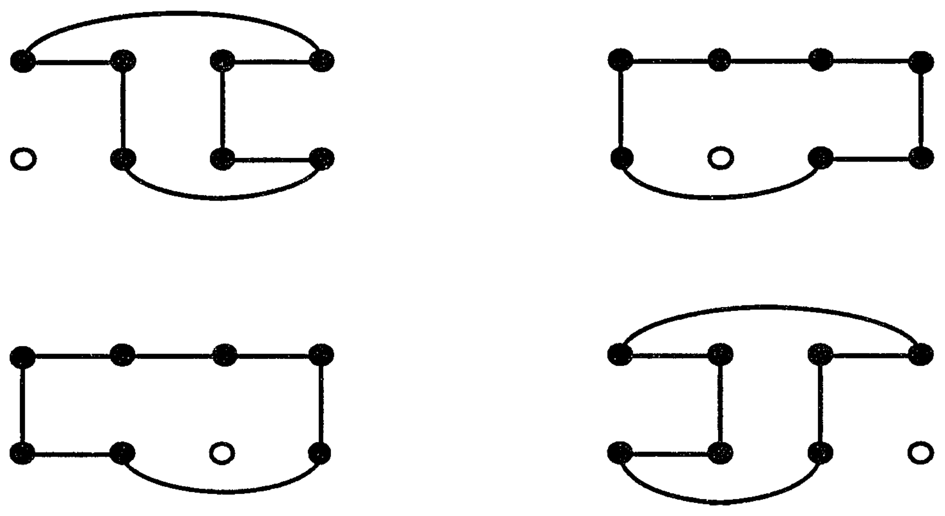
The idea behind our technique to embed a ring into a faulty twisted hypercube is to use some of the unused links to skip a faulty node. As mentioned in the previous section, Figure 4.1 shows two kinds of embeddings of a ring $R_{2^3}$ into a twisted hypercube $TQ_3$. Notice that some of the links are not part of the embedding. As an illustration, in Figure 4.1.a, the links between nodes $v_1$ and $v_5$ through dimension 3, $v_2$ and $v_7$ through dimension 2, $v_3$ and $v_6$ through dimension 2, and $v_4$ and $v_8$ through dimension 3 are unused links. We can use these unused links to avoid a faulty node.

Therefore, if node $v_i$ in a twisted hypercube $TQ_n$ is faulty, a ring $R_{2^n-1}$ can be constructed by using some of the unused links to skip the faulty node without disturbing the construction of the rest of the ring. A faulty node is either an upper node or a lower node.

The basic idea behind our technique is to identify the faulty node and the cube that contains it, then avoid the fault by using the unused links. Figure 4.4 shows all possible locations of a faulty node within a cube and the links that need to be used to avoid it in the process of constructing the ring. Part (a) shows how to handle an upper faulty node, while part (b) shows how to handle a lower faulty node. Notice that part (a) simulates type B embedding within a cube since it does not disturb the construction of the rest of the ring, the twisted lower links can be used to connect it with adjacent rings when type B embedding is used. On the other hand, part (b) simulates type A embedding within a cube since it does not disturb the construction of rest of the the ring, the upper links can be used to connect it with adjacent rings when type A embedding is used. Figure 4.5.a shows how to handle an upper faulty node, while Figure 4.5.b shows how to handle a lower faulty node. Notice that the upper faulty node is in the second cube, while the lower faulty node is in the first cube. The location of the cube that contains the faulty node might be the first, the last, or some where in between. Our technique works for all three cases by using the appropriate links. The



(a) Upper faulty node



(b) Lower faulty node

Figure 4.4: All possible locations of a faulty node.

following algorithm embeds a ring $R_{2^n-1}$ into a twisted hypercube $TQ_n$ in the presence of a faulty node.

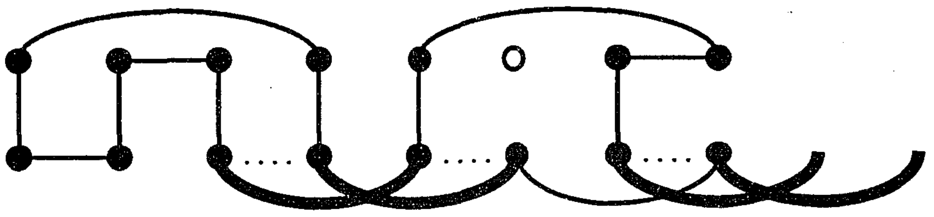
**Algorithm 4.3**

**Step 1:** Partition $TQ_n$ into $2^{n-3}$ node disjoint cubes.

**Step 2:** Locate the cube that contains the faulty node and identify whether it is an upper or a lower node.

**Step 3:** (i) If it is an upper node then

a. Choose the appropriate embedding from Figure 4.4.a.


(a) Upper faulty node

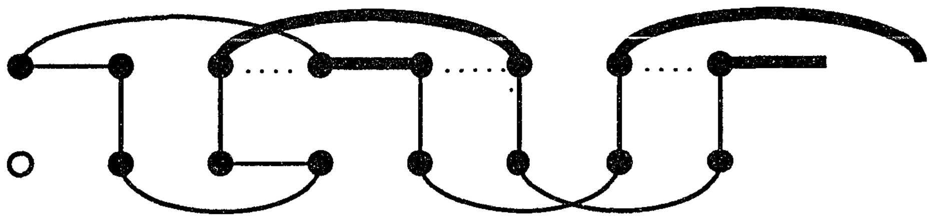

(b) Lower faulty node

Figure 4.5: Single fault embedding.

b. Embed the ring $R_{2^3}$ into each of the fault-free cubes using type B embedding.

c. Connect all the rings, one of size 7 and the rest of size 8, using the twisted lower links to come up with the ring $R_{2^n-1}$.

(ii) If it is a lower node then

a. Choose the appropriate embedding from Figure 4.4.b.

b. Embed the ring $R_{2^3}$ into each of the fault-free cubes using type A embedding.

c. Connect all the rings, one of size 7 and the rest of size 8, using the upper links to come up with the ring $R_{2^n-1}$.

**Theorem 4.3:** For every $n$, Algorithm 4.3 will embed a ring of size $2^{n-1}$ into a twisted hypercube of dimension $n$ in the presence of a faulty node.

The theorem can be proved easily by extending the proof of theorem 4.2. In the next section, we will use the same concept with minor variations to embed a ring into a faulty twisted hypercube with multiple faults.

## 4.3.2 Embedding in The Presence of Multiple Faults

In this section, we describe our scheme to embed a ring $R_{2^n-f}$, where $f$ is the number of faults, into a twisted hypercube $TQ_n$ in the presence of $f$ faults such that each cube has at most one faulty node. A cube might be an overlap cube as shown in

Figure 4.3.b. The maximum number of faults that can be handled by our technique is $f = 2^{n-3}$. The idea is to generalize Algorithm 4.3 to handle multiple faults. The following algorithm embeds a ring $R_{2^n-f}$ into a twisted hypercube $TQ_n$ in the presence of $f$ faults.

**Algorithm 4.4**

**Step 1:** Partition $TQ_n$ into $2^{n-2}$ blocks.

**Step 2:** Identify the blocks with faulty nodes.

**Step 3:** Group each faulty block with the adjacent unfaulty block to its left to form a faulty cube.

**Step 4:** Embed a ring of size 7 into each of the faulty cubes by choosing an appropriate embedding from Figure 4.4 and embed a ring of size 4 into each of the blocks.

**Step 5:** Construct a ring of size $2^n - f$ by connecting the rings, either $R_7$ or $R_4$, using the appropriate links, either upper links or twisted lower links as shown in Figure 4.6.

**Theorem 4.4:** For every $n$, Algorithm 4.4 will embed a ring of size $2^n - f$ into a twisted hypercube of dimension $n$ in the presence of $f$ faulty nodes such that each cube has at most one faulty node.

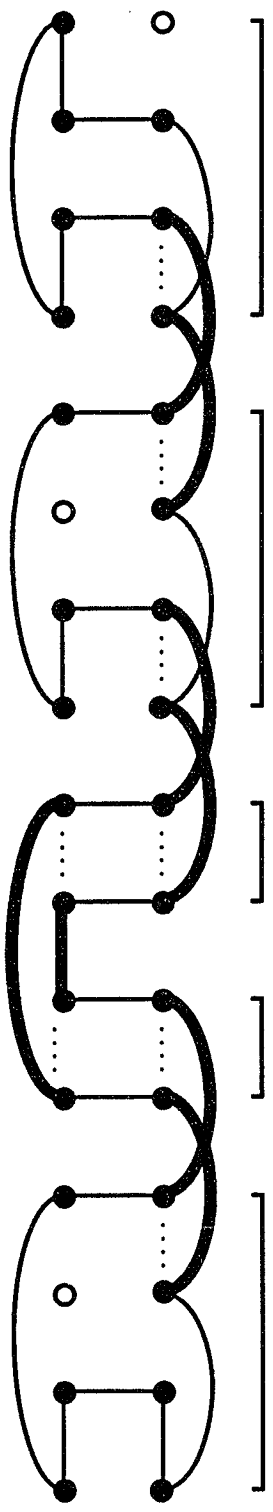


Figure 4.6: Multiple faults embedding.

*Proof:* Without loss of generality, we assume that the left most block has no faulty node. The existence of an adjacent unfaulty block to the left of any faulty block follows directly from our assumption that each cube has at most one faulty node. In the process of constructing the ring $R_{2^n-f}$, any two adjacent cubes with a fault are one of the following cases

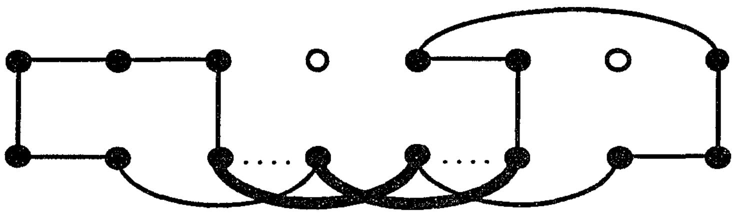
Case 1: A cube with upper fault followed by a cube with upper fault.

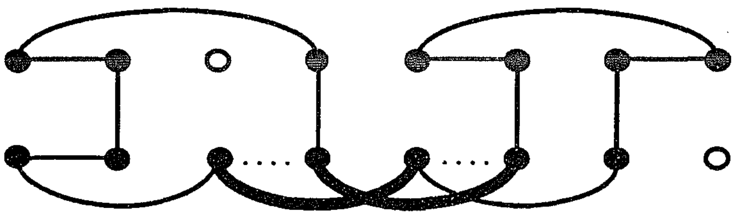
Case 2: A cube with upper fault followed by a cube with lower fault.

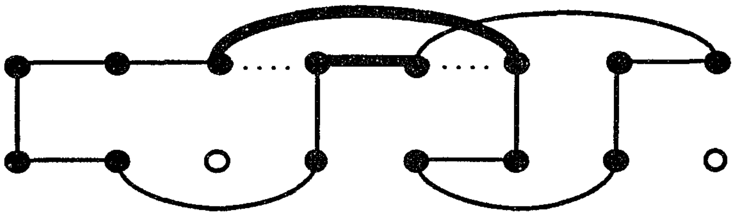
Case 3: A cube with lower fault followed by a cube with lower fault.

Case 4: A cube with lower fault followed by a cube with upper fault.

Figure 4.7 shows all four cases in the process of constructing the ring. We use the twisted lower links with an upper faulty cube followed by either an upper or a lower faulty cube and the upper links with a lower faulty cube followed by either a lower or an upper faulty cube. The way we grouped the faulty blocks with unfaulty blocks to form cubes always guarantees the existence of such links. The other cases are an upper or a lower faulty cube followed by a block and a block followed by a block or a faulty cube. We use the twisted lower links with an upper faulty cube followed by a block and the upper links with a lower faulty cube followed by a block. For the case of a block followed by a block or a faulty cube, we use the appropriate links, either upper or twisted lower links, since both are available. □


(a) Upper followed by upper


(b) Upper followed by lower


(c) Lower followed by lower

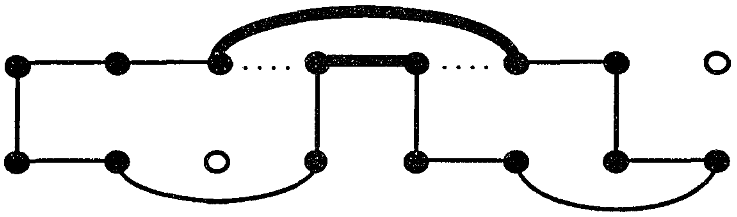

(d) Lower followed by upper

Figure 4.7: All possible cases of two adjacent faulty cubes.

## 4.4 Summary

In this chapter, we presented optimal algorithms for embedding a ring into a twisted hypercube with fault-free nodes, single faulty node, and multiple faults. We showed the capability of the twisted hypercube to simulate rings efficiently in the presence of faults. While even one faulty processor will degrade its over all performance, like any other network, but it is still capable of constructing a Hamiltonian circuit within the nonfaulty processors.

A twisted hypercube $TQ_n$ with $2^n$ nodes can simulate a ring $R_{2^n-f}$ with $2^n - f$ nodes in the presence of $f$ twisted hypercube faulty nodes with some restrictions on the location of the faults. In the hypercube, the simulation of rings achieved by wasting a nonfaulty processor for every faulty processor. The simulation of rings by twisted hypercube is more efficient since it is achieved without wasting any nonfaulty processors.

CHAPTER 5

# Fault-Tolerance Embedding of Rings into Hypercubes

## 5.1 Introduction

The hypercube has been the focus of many recent research activities. Extensive work has been done to show that the hypercube is a powerful architecture capable of simulating other interconnection networks such as rings, meshes, trees, stars, and others with minimum overhead ([BCGS], [BCLR], [BMS], [BSu], [MS], [SS], [Lei]). It has also been shown that the hypercube machine is robust and fault-tolerant and has the ability to simulate, route, and reconfigure itself despite the presence of either faulty links or nodes ([BS], [CL], [HLNa], [HLNb], [PM], [WCM]).

The problem of embedding rings into other interconnection networks has been addressed by many researchers. Rosenberg and Snyder [RS] addressed the problem of embedding rings into general graphs. They showed a dilation 3 embedding of a ring into a general graph of the same size. In [JLD] and [NSK], the authors considered embedding cycles, rings, and Hamiltonians into star networks. Saad and Schultz [SS] used Gray Codes to show the existence of a Hamiltonian circuit in a hypercube structure. Chan and Shin [CS] used Gray Codes to identify $n!$ distinct Hamiltonian paths in a hypercube network.

Embedding rings into hypercubes despite the presence of faults have been addressed by many researchers. Provost and Melhem [PV] have given distributed algorithms in the presence of single, double, and multiple faults wasting up to 50% of the processors in the worst case. Chan and Lee [CL] improved the result by wasting only one nonfaulty processor for every faulty processor and allowing up to $\lfloor \frac{n+1}{2} \rfloor$ faults. This chapter uses a new technique to embed a ring of size $2^n - 2f$ into a hypercube of dimension $n$ despite the presence of $f$ faults. It wastes only one nonfaulty processor for every faulty processor and allows up to $2^{n-3}$ faults with some restriction on the location of the faults [ABd].

The remainder of this chapter is organized as follows. In section 2, we describe our scheme to embed a ring of size $2^n$ into a fault-free hypercube of the same size. Section 3 addresses embedding in the presence of faulty nodes. Our emphasis will be on the multiple fault case. Section 4 concludes the chapter.

## 5.2 Fault-Free Embeddings

Given a ring $R_{2^n}$ with $2^n$ nodes. Consider the problem of assigning the ring nodes to the nodes of the hypercube such that adjacency is preserved. In the hypercube, two nodes are adjacent if the binary representation of their labels differ in exactly one bit position, say in position $i$. We call the link that connects the two adjacent nodes a *link through dimension i*. The least significant bit in the binary representation of a label is referred to as position 1 and the most significant bit as position $n$.

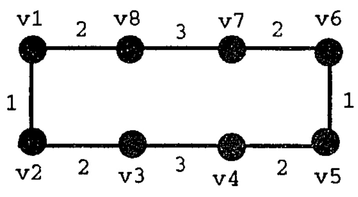


Figure 5.1: The embedding sequence.

Now given any two adjacent nodes in the ring, their images by this embedding should be neighbors in the hypercube through some dimension $i$, where $1 \leq i \leq n$. We can view such an embedding as a sequence of dimensions crossed by adjacent nodes. We call such a sequence the ***embedding sequence***, denoted by ES = $(d_1, d_2, ..., d_{2^n})$, where $d_i \in \{1, ..., n\}$ for all $1 \leq i \leq 2^n$.

Figure 5.1 shows an embedding of the ring $R_{2^3}$ into the hypercube $Q_3$. It is more convenient to view the embedded ring as will as the hypercube in the way shown in Figure 5.1. We view the hypercube as two levels where all nodes with even labels are in the upper level and all nodes with odd labels are in the lower level. The embedding sequence of $R_{2^3}$ is ES = (1, 2, 3, 2, 1, 2, 3, 2). For example, in Figure 5.1, notice that nodes $v_1$ and $v_2$ are connected by a link through dimension 1, $v_2$ and $v_3$ are connected by a link through dimension 2, $v_3$ and $v_4$ are connected by a link through dimension 3, $v_4$ and $v_5$ are connected by a link through dimension 2, and so on. The embedding sequence ES can be generated using the following algorithm.

**Algorithm 5.1**

Let $n$ be the dimension of the hypercube and let the vertical bar be the concatenation operator.

```
Step 1: ES ← 2
Step 2: For i ← 3 to $n$ do
            ES ← ES|i|ES
Step 3: ES ← 1|ES|1|ES
```

The embedding sequence is generated by applying Algorithm 5.1 on $n$, where $n$ is the dimension of the hypercube. The number of nodes in the hypercube is equal to the number of nodes in the embedded ring which is $2^n$ nodes. Thus, the embedding sequence of the ring $R_{2^4}$ is ES = (1, 2, 3, 2, 4, 2, 3, 2, 1, 2, 3, 2, 4, 2, 3, 2) and the embedding sequence of the ring $R_{2^5}$ is ES = (1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3, 2, 1, 2, 3, 2, 4, 2, 3, 2, 5, 2, 3, 2, 4, 2, 3, 2). Notice that the same embedding sequence may result in different embeddings of $R_{2^n}$ into $Q_n$ depending on the hypercube node that initiates the ring construction. Among all different embeddings, we are interested in the embedding where the node that initiates the ring construction in the hypercube is node with label 0.

**Theorem 5.1:** For every $n$, Algorithm 5.1 will generate the embedding sequence to construct a ring of size $2^n$ in a fault-free hypercube of dimension $n$.

*Proof:* We prove this by induction on the dimension of the hypercube. Our induction basis is $Q_2$, a ring of size 4 can be easily constructed in $Q_2$ using the embedding sequence ES = (1, 2, 1, 2). Assume the theorem is true for the construction of a ring of size $2^{n-1}$ in a hypercube of dimension $n$-1. We now prove that the theorem is true for the construction of $R_{2^n}$ in $Q_n$. Consider the two subcubes $Q^0{}_{n-1}$ and $Q^1{}_{n-1}$. By induction hypothesis, we can construct a ring of size $2^{n-1}$ in both $Q^0{}_{n-1}$ and $Q^1{}_{n-1}$. Let their embedding sequence be

$$\text{ES} = 1 \,|\, S_{n-1} \,|\, 1 \,|\, S_{n-1}$$

where $S_n$ is a sequence of dimensions recursively defined as follows:

$$S_2 = 2$$

$$S_{n-1} = S_{n-2} \,|\, n \,|\, S_{n-2}$$

Now we combine two rings, each of size $2^{n-1}$, to come up with a ring of size $2^n$. This is done by replacing the second link that goes through dimension 1 of the first ring and the first link that goes through dimension 1 of the second ring by two links that go through dimension $n$. The embedding sequence of the new ring $R_{2^n}$ is

$$\begin{aligned}\text{ES} &= 1 \,|\, S_{n-1} \,|\, n \,|\, S_{n-1} \,|\, 1 \,|\, S_{n-1} \,|\, \text{n} \,|\, S_{n-1} \\ &= 1 \,|\, S_n \,|\, 1 \,|\, S_n\end{aligned}$$

which is the same embedding sequence generated by Algorithm 5.1. □

## 5.2.1 Divide-Conquer Embeddings

This section introduces a data structure, that is fundamental to the embeddings given in this chapter, called a cube. A *cube* is a subcube of dimension 3 that consists

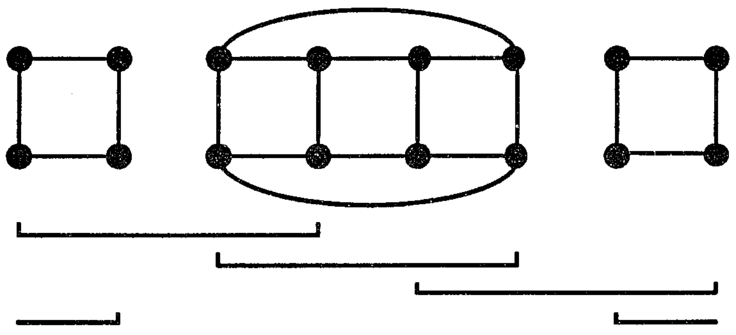


Figure 5.2: The cube.

of two adjacent blocks as shown in Figure 5.2. A *block* is a set of four nodes in a hypercube that form a ring of size 4 that has the embedding sequence ES = (1, 2, 1, 2). Notice that cubes overlap and a hypercube of dimension $n$, $Q_n$, contains $2^{n-2}$ cubes. A ring of size 8, $R_{2^3}$, can be embedded into a cube by the embedding sequence ES = (1, 2, 3, 2, 1, 2, 3, 2). The cube is used to introduce new techniques to embed a ring into a twisted hypercube. These new techniques are generalized in later sections to embed a ring into a faulty twisted hypercube. The next algorithm uses a divide-conquer technique to embed a ring $R_{2^n}$ into a hypercube $Q_n$.

**Algorithm 5.2**

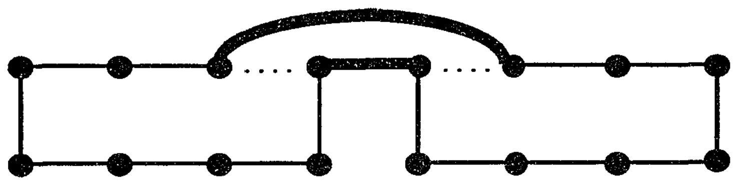
**Step 1:** Partition $Q_n$ into $2^{n-3}$ node disjoint cubes.

**Step 2:** Embed the ring $R_{2^3}$ into each cube using the embedding sequence ES = (1, 2, 3, 2, 1, 2, 3, 2).

**Step 3:** Connect the $2^{n-3}$ rings, each of size 8, through the upper, or lower, links to come up with a ring of size $R_{2^n}$.

**Theorem 5.2:** For every $n$, Algorithm 5.2 will embed a ring of size $2^n$ in a fault-free hypercube of dimension $n$.

*Proof:* We prove this by induction on the dimension of the hypercube. Our induction basis is $Q_3$, the embedding of a ring of size $2^3$ into a hypercube of dimension 3 is shown in Figure 5.1. Assume the theorem is true for the construction of a ring of size $2^{n-1}$ in a hypercube of dimension $n$-1. We now prove that the theorem is true for the


(a) Using upper links

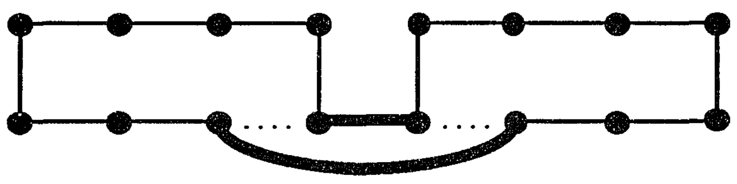

(b) Using lower links

Figure 5.3: Fault-free embedding.

construction of $R_{2^n}$ in $Q_n$. Consider the two subcubes $Q^0{}_{n-1}$ and $Q^1{}_{n-1}$. By assumption we can construct a ring of size $2^{n-1}$ in both $Q^0{}_{n-1}$ and $Q^1{}_{n-1}$. Now we combine two rings, each of size $2^{n-1}$, to come up with a ring of size $2^n$. This is done by replacing the first link that goes through dimension 2 in the upper part of the first ring and the last link that goes through dimension 2 in the upper part of the second ring by two upper links that go through dimension $n$, or by replacing the last link that goes through dimension 2 in the lower part of the first ring and the first link that goes through dimension 2 in the lower part of the second ring by two lower links that go through dimension $n$, as shown in Figure 5.3. □

## 5.3 Fault-Tolerance Embeddings

One of the special significant features of the hypercube is its ability to simulate other interconnection networks in the presence of faults. In this section, we are interested in answering the following question. Given that some nodes of the hypercube are faulty, does the hypercube have the ability to simulate rings efficiently? The hypercube is maximally fault-tolerant. While even one faulty processor will degrade its overall performance, it is still capable of simulating rings by wasting only one non-faulty processor for every faulty processor.

### 5.3.1 Embedding in the Presence of a Single Fault

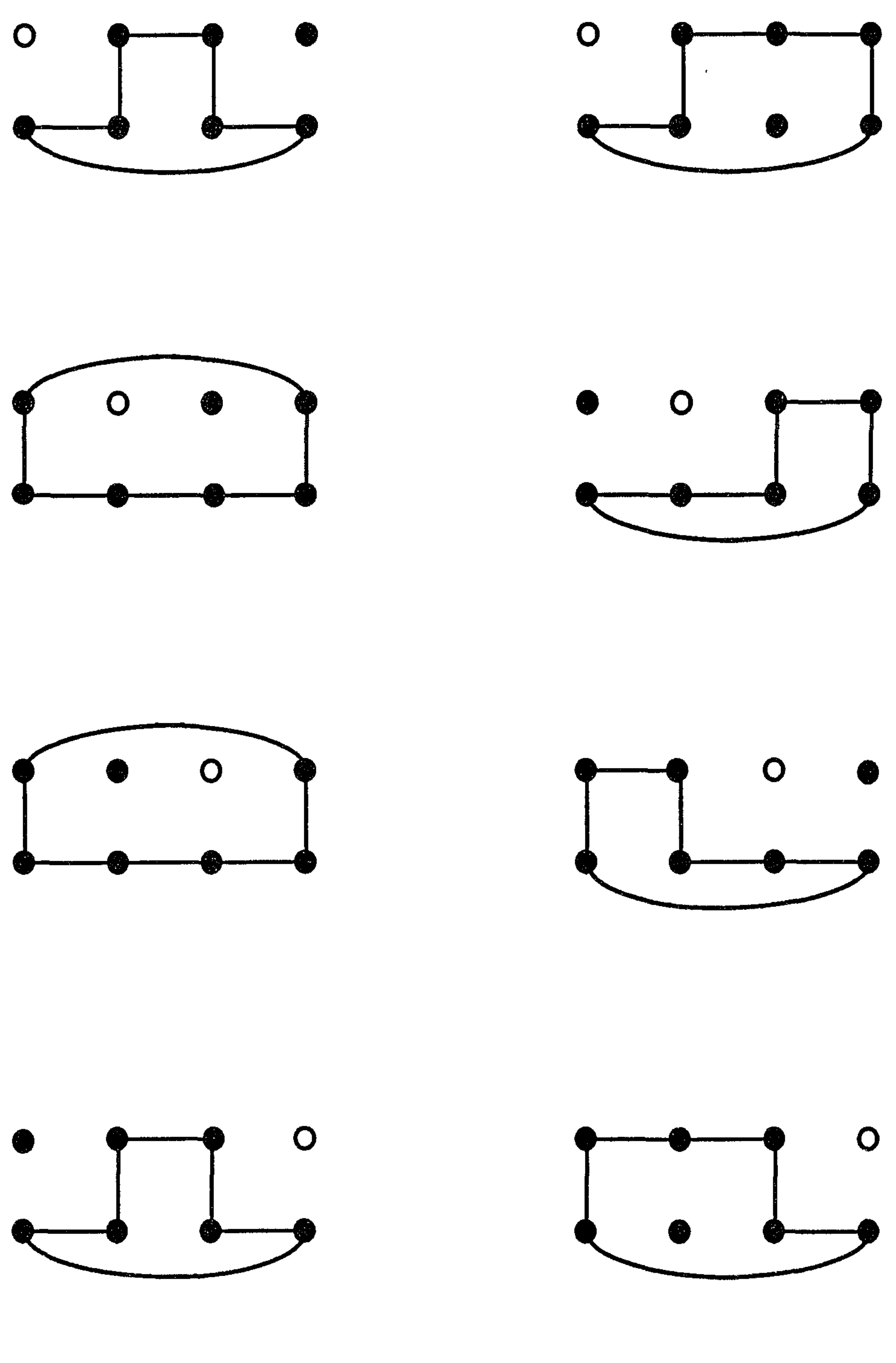
The idea behind our technique to embed a ring into a faulty hypercube is to use some of the unused links to skip a faulty node. As an illustration, in Figure 5.1, the links between nodes $v_1$ and $v_6$ through dimension 3, $v_2$ and $v_5$ through dimension 3, $v_3$ and $v_8$ through dimension 1, and $v_4$ and $v_7$ through dimension 1 are unused links. We can use these unused links to avoid a faulty node. But since the hypercube does not contain odd cycles, we have to waste a nonfaulty processor for every faulty processor. Therefore, if node $v_i$ in a hypercube $Q_n$ is faulty, a ring $R_{2^n-2}$ can be constructed by using some of the unused links to skip the faulty node without disturbing the construction of the rest of the ring.

The basic idea behind our technique is to identify the faulty node and the cube that contains it, then avoid the fault by using the unused links. Figure 5.4 shows all possible locations of an upper faulty node within a cube and the links that need to be used to avoid it in the process of constructing the ring, while Figure 5.5 shows the case of a lower faulty node. Figure 5.6.a shows how to handle an upper faulty node, while Figure 5.6.b shows how to handle a lower faulty node. The location of the cube that contains the faulty node might be the first, the last, or some where in the middle. Our technique works for all three cases by using the appropriate links. The following algorithm embed a ring $R_{2^n-2}$ into a hypercube $Q_n$ in the presence of a faulty node.

**Algorithm 5.3**

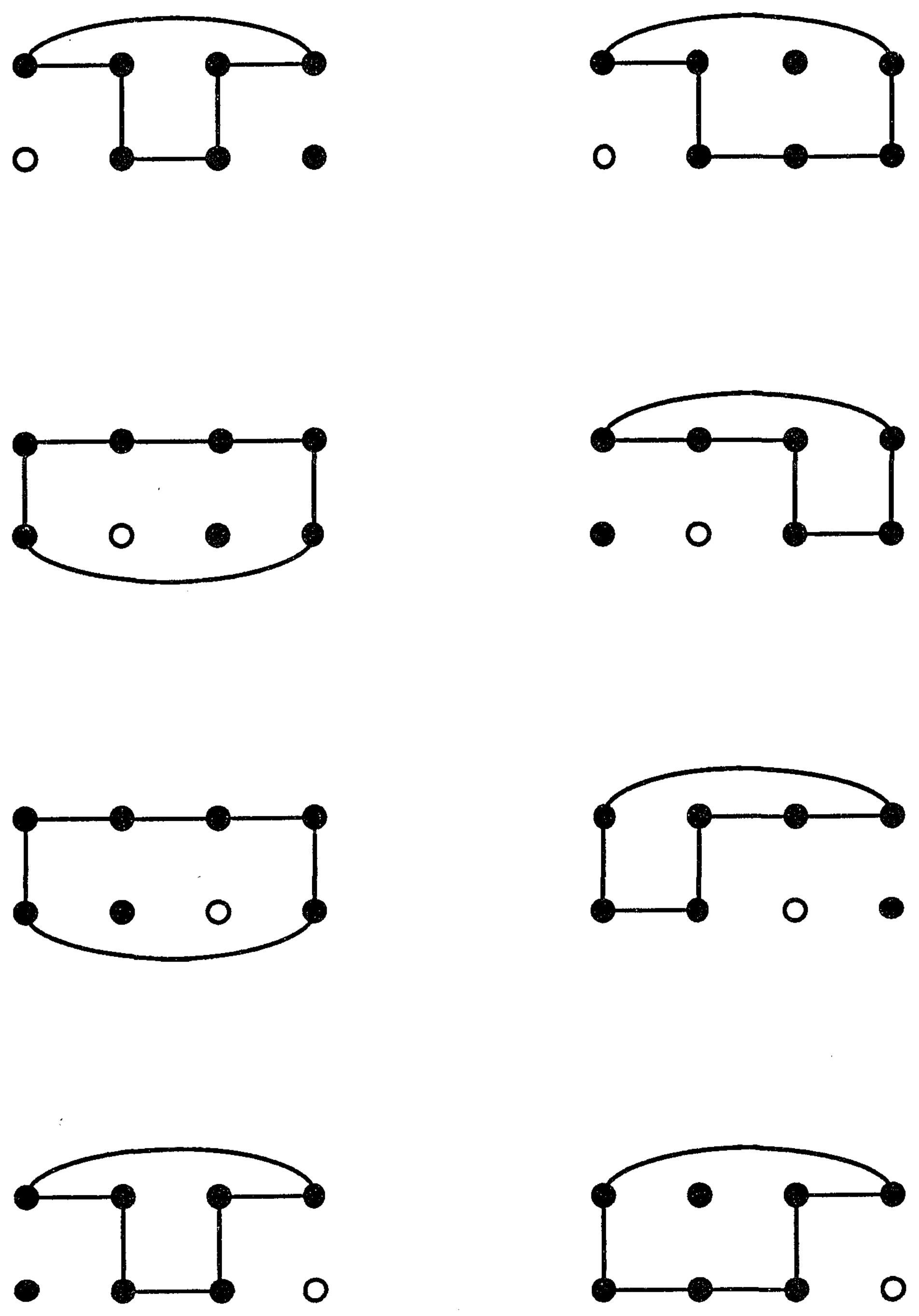
**Step 1:** Partition $Q_n$ into $2^{n-3}$ node disjoint cubes.

**Step 2:** Locate the cube that contains the faulty node and identify whether it is an upper or a lower fault.



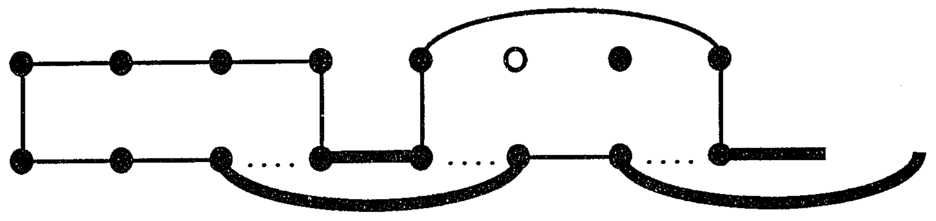
(a) Standard (b) Alternate

Figure 5.4: All possible locations of an upper faulty node within a cube.



(a) Standard (b) Alternate

Figure 5.5: All possible locations of a lower faulty node within a cube.


(a) Upper faulty node

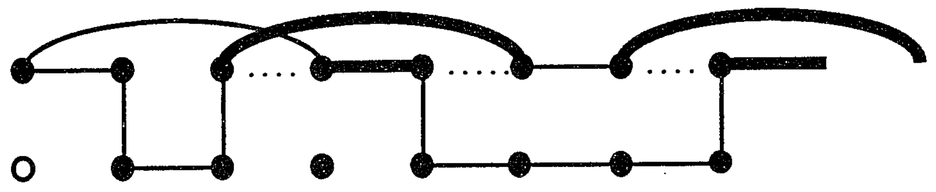

(b) Lower faulty node

Figure 5.6: Single fault embedding.

**Step 3:** (i) If it is an upper fault then

a. Choose the appropriate embedding from Figure 5.4.a.

b. Embed the ring $R_{2^3}$ into each of the fault-free cubes using the embedding sequence ES = (1, 2, 3, 2, 1, 2, 3, 2).

c. Connect all the rings, one of size 6 and the rest of size 8, using the lower links to come up with the ring $R_{2^n-2}$.

(ii) If it is a lower fault then

a. Choose the appropriate embedding from Figure 5.5.a.

b. Embed the ring $R_{2^3}$ into each of the fault-free cubes using the embedding sequence ES = (1, 2, 3, 2, 1, 2, 3, 2).

c. Connect all the rings, one of size 6 and the rest of size 8, using the upper links to come up with the ring $R_{2^n-1}$.

**Theorem 5.3:** For every $n$, Algorithm 5.3 will embed a ring of size $2^n - 2$ into a hypercube of dimension $n$ in the presence of a faulty node.

The theorem can be proven easily by induction by extending the proof of theorem 5.2. In the next section, we will use the same concept with minor variations to embed a ring into a faulty hypercube with multiple faults.

### 5.3.2 Embedding in The Presence of Multiple Faults

In this section, we describe our scheme to embed a ring $R_{2^n-2f}$, where $f$ is the number of faults, into a hypercube $Q_n$ in the presence of $f$ faults such that each cube has at most one faulty node. The maximum number of faults that can be handled by our technique is $f = 2^{n-3}$. The idea is to generalize Algorithm 5.3 to handle multiple faults. The following algorithm embeds a ring $R_{2^n-2f}$ into a hypercube $Q_n$ in the presence of $f$ faults.

**Algorithm 5.4**

**Step 1:** Partition $Q_n$ into $2^{n-3}$ node disjoint cubes.

**Step 2:** Identify the cubes with faulty nodes.

**Step 3:** Embed a ring of size 6 into each of the faulty cubes by choosing an appropriate embedding from Figures 5.4 and 5.5 and a ring of size 8 into each of the unfaulty cubes.

**Step 4:** Construct a ring of size $2^n - 2f$ by connecting the rings, either $R_6$ or $R_8$, using the appropriate links, either upper or lower links as shown in Figure 5.7.

**Theorem 5.4:** For every $n$, Algorithm 5.4 will embed a ring of size $2^n - 2f$ into a hypercube of dimension $n$ in the presence of $f$ faulty nodes such that each cube has at most one faulty node.

*Proof:* In the process of constructing the ring $R_{2^n-2f}$, any two adjacent cubes with a fault are one of the following cases

Case 1: A cube with upper fault followed by cube with upper fault.

Case 2: A cube with upper fault followed by cube with lower fault.

Case 3: A cube with lower fault followed by cube with lower fault.

Case 4: a cube with lower fault followed by cube with upper fault.

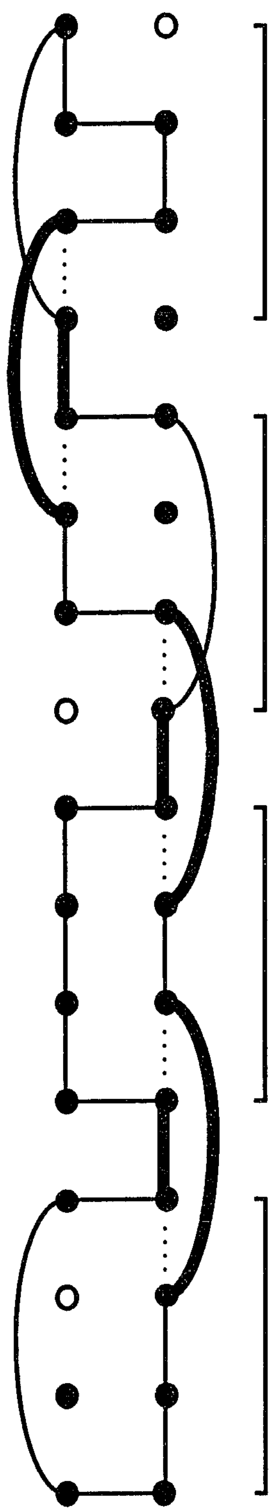
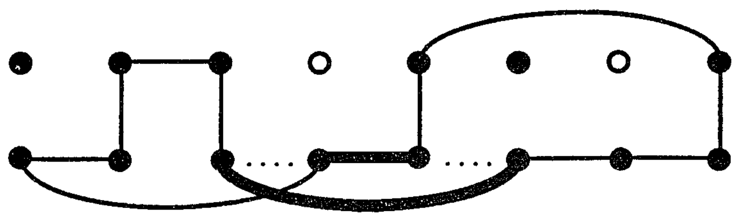


Figure 5.7: Multiple faults embedding.

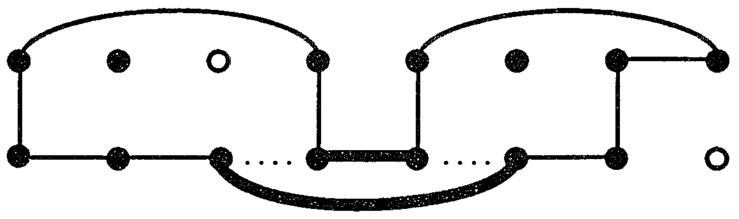
Figure 5.8 shows all four cases in the process of constructing the ring. Notice that the decision of whether to use a standard or alternate ring depends about the position of the faulty node within a cube, whether it is in the left or right block and whether it is an upper or a lower fault. Also, the position of the faults in adjacent cubes affect the type of ring to be used. We use the lower links with an upper followed by an upper, the upper links with a lower followed by a lower, and in the case of an upper followed by a lower or a lower followed by an upper we might use the upper or the lower links depending on the location of the faults. Since we are wasting one good processor for every faulty processor, the size of the embedded ring is $2^n - 2f$. □

## 5.4 Summary

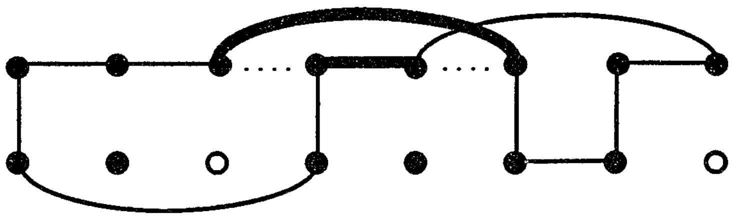
This chapter has presented new techniques to embed a ring of size $2^n - 2f$ in a hypercube of dimension $n$ despite the presence of $f \leq 2^{n-3}$ faults. The new divide-conquer technique uses a new data structure called cube. The basic idea behind the technique is to identify faulty nodes and the cubes that contains them, avoid the faults within the cube by using the unused links, and construct the ring connecting adjacent cubes. Our technique has some restrictions on the distribution of the faults. It allows up to $2^{n-3}$ faults such that each cube has at most one fault.



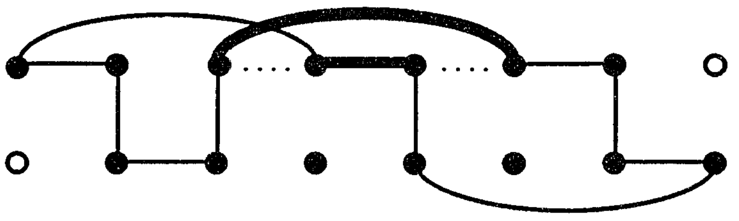
(a) Upper followed by upper



(b) Upper followed by lower



(c) Lower followed by lower



(d) Lower followed by upper

Figure 5.8: All possible cases of two adjacent faulty cubes.

CHAPTER 6

# Concluding Remarks

One of the most important factors that govern the performance of a parallel machine is the underlying interconnection network. Many interconnection networks have been introduced in the literature. The most important features of these interconnection networks are the diameter and the node degree. Another important feature of a network is fault-tolerance. Hypercubes have gained wide spread acceptance due to their many attractive properties. The twisted hypercube preserves many of the properties of the hypercube and reduces the diameter by a factor of two. This dissertation explored the efficiency and the fault-tolerance of the twisted hypercube in parallel computation and investigated relations and transformations between the twisted hypercube and various interconnection networks. These include complete binary trees, complete quad trees, fault-free rings, faulty rings, and hypercubes.

We have presented different schemes to embed complete binary trees and complete quad trees into the twisted hypercube. For complete binary trees, we have presented two different schemes to embed a complete binary tree $CB_n$ into a twisted hypercube $TQ_n$. In the first scheme, we used a recursive algorithm to embed $CB_n$ into $TQ_n$ based on the embedding of $CB_{n-1}$ into $TQ_{n-1}$. The resulting embedding is such that all edges in the lowest four levels of the complete binary tree are mapped to paths of length one in the twisted hypercube and all other edges in higher levels of the complete binary tree are mapped to paths of length two in the twisted hypercube. In the second scheme, we used the inorder binary labeling of the complete binary tree $CB_n$ to embed $CB_n$ into the twisted hypercube $TQ_n$. The inorder embedding is simpler and more natural than the recursive embedding, but it is less efficient in terms of the number of edges that are mapped to paths of length two. For complete quad trees, we have presented a recursive algorithm that embeds $CQ_n$ into $TQ_n$ based on the embedding of $CQ_{n-1}$ into $TQ_{n-1}$. The resulting embedding is such that 37.5% of the edges in the lowest level and 50% of the edges in higher levels of the complete quad tree are mapped to paths of length two in the twisted hypercube and the rest of edges are mapped to paths of length one.

Interesting results have been presented on the fault-tolerance of the twisted hypercube. We have presented optimal algorithms for embedding a ring into a twisted hypercube with fault-free nodes, single faulty node, and multiple faults. We have shown the capability of the twisted hypercube to simulate rings efficiently in the presence of faults. While even one faulty processor will degrade its over all performance, like any other network, but a Hamiltonian circuit can be constructed on the nonfaulty processors. We have shown that a twisted hypercube $TQ_n$ with $2^n$ nodes can simulate a ring $R_{2^n-f}$ with $2^n - f$ nodes in the presence of $f$ twisted hypercube faults. In the hypercube, the simulation of rings achieved by wasting a nonfaulty processor for every faulty processor. The simulation of rings by twisted hypercube is more efficient since it is achieved without wasting any nonfaulty processors.

We have presented new techniques to embed a ring of size $2^n - 2f$ in a hypercube of dimension $n$ despite the presence of $f \leq 2^{n-3}$ faults. The new divide-conquer technique uses a new data structure called cube. Our algorithm for multiple faults allows up to $2^{n-3}$ faults such that each cube has at most one fault.

In future work, we intend to study the embedding of other parallel architectures into the twisted hypercube. It may also be possible to improve on some of our results such as embedding complete binary tress into twisted hypercubes. It has been conjectured that the complete binary tree $CB_n$ is a subgraph of the twisted hypercube $TQ_n$. An interesting obvious problem left open is whether the number of faults that can be tolerated by the twisted hypercube can be improved further. Another interesting problem will be to adapt our techniques of embedding rings into faulty twisted hypercubes on other parallel architectures.

# Bibliography

[A] S. Akl, The Design and Analysis of Parallel Algorithms, *Prentice-Hall*, 1989.

[ABa] E. Abuelrub and S. Bettayeb, "Embedding Complete Binary Trees into Twisted Hypercubes," *Proc. ISCA International Conference on Computer Applications in Design, Simulation and Analysis*, pp. 1-4, 1993.

[ABb] E. Abuelrub and S. Bettayeb, "Embedding Rings into Faulty Twisted Hypercubes," *Proc. 31st ACM Southeastern Regional Conference*, pp. 48-55, 1993.

[ABc] E. Abuelrub and S. Bettayeb, "Embeddings in the Twisted Hypercube," *Technical Report*, Department of Computer Science, Louisiana State University, 1993.

[ABd] E. Abuelrub and S. Bettayeb, "Fault-Tolerance Embedding of Rings into Hypercubes," *submitted for publication.*

[ABe] E. Abuelrub and S. Bettayeb, "Embedding Complete Quad Trees into Twisted Hypercubes," *submitted for publication.*

[AG] G. Almasi and A. Gottlieb, Highly Parallel Computing, *Benjamin/Cummings*, 1989.

[AGr] J. Armstrong and F. Gray, "Fault Diagnosis in a Boolean n-Cube of Microprocessor," *IEEE Transactions on Computers*, vol. C-30, no. 8, pp. 587-590, August 1981.

[AHMP] H. Alt, T. Hagerup, K. Mehlhorn, and F. Preparata, "Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones," *SIAM Journal on Computing*, pp. 8089-835, October 1987.

[AJ] G. Anderson and E. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *Computing Surveys*, vol. 7, pp. 197-213, December 1975.

[AK] S. Akers and B. Krishnamurthy, "Group Graphs as Interconnection Networks," *IEEE Transactions on Computers*, vol. 38, pp. 555-565, 1989.

[AR] R. Aleliunas and A. Rosenberg, "On embedding Rectangular Grids in Square Grids," *IEEE Transactions on Computers*, vol. C-31, pp. 907-913, September 1982.

[BA] L. Bhuyan and D. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 323-333, April 1984.

[BCGS] S. Bettayeb, B. Cong, M. Girou, and I. Sudborough, "Embedding Permutation Networks into Hypercubes," *LATIN 92*, 1992.

[BCLR] S. Bhatt, F. Chung, F. Leighton, and A. Rosenberg, "Optimal Simulations of Tree Machines," *Proc. 27th Annual IEEE Foundations of Computer Science Conference*, pp. 274-282, 1986.

[BH] R. Beivide and E. Herrada, "Optimal Distance Networks of Low Degree for Parallel Computing," *IEEE Transactions on Computers*, vol. C-40, no. 10, pp. 1109-1123, October 1991.

[BI] S. Bhatt and I. Ispen, "How to Embed Trees in Hypercubes," *Technical Report*, Department of Computer Science, Yale University, 1985.

[BL] H. Bodlaender and J. Leeuwen, "Simulation of Large Networks on Smaller Networks," *Information and Control*, vol. 71, pp. 143-180, 1986.

[BLD] L. Barasch, S. Lakshmivarahan, and S. Dhall, "Embedding Arbitrary Meshes and Complete Binary Trees in Generalized Hypercubes," *Proc. 1st IEEE Symposium on Parallel and Distributed Processing*, 1989.

[BMS] S. Bettayeb, Z. Miller, and I. Sudborough, "Embedding Grids into Hypercubes," *Journal of Computer and System Sciences*, vol. 45, no. 3, pp. 340-366, December 1992.

[BS] B. Becker and H. Simon, "How Robust is the n-Cube," *Information and Computation*, no. 2, pp. 162-178, May 1988.

[BSu] S. Bettayeb and I. Sudborough, "Grid Embedding into Ternary Hypercubes," *Proc. 1989 ACM South Central Regional Conference*, pp. 62-64, 1989.

[CL] M. Chan and S. Lee, "Distributed Fault-Tolerance Embeddings of Rings into Hypercubes," *Journal of Parallel and Distributed Computing*, no. 11, pp. 63-71, 1991.

[CLe] G. Chartrand and L. Lesniak, Graphs and Digraphs, *Wadsworth & Brooks*, 1986.

[CS] M. Chen and K. Shin, "Processor Allocation in an n-Cube Multiprocessor Using Gray Codes," *IEEE Transactions on Computers*, vol. C-36, no. 12, pp. 396-407, December 1987.

[DS] A. Dingle and I Sudborough, "Simulating Binary Trees and X-Trees on Pyramid Networks," *Proc. 1st IEEE Symposium on Parallel and Distributed Processing*, pp. 210-219, 1989.

[E] K. Efe, "The Crossed Cube Architecture for Parallel Computation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 5, pp. 513-524, September 1992.

[EBSS] K. Efe, P. Blackwell, T. Shiau, and W. Slough, "A Reduced Diameter Interconnection Network," *Technical Report*, Department of Computer Science, University of Missouri, 1988.

[EL] A. El-Amawy and S. Latifi, "Properties and Performance of Folded Hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 1, pp. 31-42, January 1991.

[ENS] A. Esfahanian, L. Ni, and B. Sagan, "On Enhancing Hypercube Multiprocessors," *Proc. 1988 International Conference on Parallel Processing*, pp. 86-89, 1988.

[F] M. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. C-21, no. 9, September 1972.

[FS] R. Finkel and M. Solomon, "Processor Interconnection Strategy," *IEEE Transactions on Computers*, vol. C-33, pp. 1180-1194, December 1984.

[Gor] D. Gordon, "Efficient Embeddings of Binary Trees in VLSI Arrays," *IEEE Transactions on Computers*, vol. C-36, no. 9, pp. 1009-1018, September 1987.

[Gou] R. Gould, Graph Theory, *Benjamin/Cummings*, 1988.

[GW] A. Gupta and H. Wang, "Optimal Embeddings of Ternary Trees into Boolean Hypercubes," *Proc. 4th IEEE Symposium on Parallel and Distributed Processing*, pp. 230-235, 1992.

[H] W. Hills, The Connection Machine, *MIT Press*, 1985.

[HB] K. Hwang and F. Briggs, Computer Architecture and Parallel Processing, *McGraw-Hill*, 1984.

[HJ] C. Ho and S. Johnson, "Dilation d Embedding of a Hyper-Pyramid into a Hypercube," *Proc. Supercomputing 89*, pp. 294-303, 1989.

[HLNa] J. Hastad, F. Leighton, and M Newman, "Reconfiguring a Hypercube in the Presence of Faults," *Proc. 19th Annual ACM STOC*, pp. 274-284, 1987.

[HLNb] J. Hastad, F. Leighton, and M Newman, "Fast Computation Using Faulty Hypercubes," *Proc. 19th Annual ACM STOC*, pp. 251-263, 1987.

[HMR] J. Hong, K. Mehlhorn, and A. Rosenberg, "Cost Trade-Offs in Graph Embeddings, with Applications," *Journal of the Association Computer Machinary*, vol. 30, no. 4, pp. 709-728, October 1983.

[HS] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, *Computer Science Press*, 1989.

[I] O. Ibe, "Reliability Comparison of Token-Ring Network Schemes," *IEEE Transactions on Reliability*, vol. 41, no. 2, pp. 288-283, June 1992.

[II] M. Imase and M. Itoh, "Design to Minimize Diameter on Building Block Network," *IEEE Transactions on Computers*, vol. C-30, no. 6, pp. 439-448, 1981.

[JLD] J Jwo, S. Lakshmivarahan, and S. Dhall, "Embedding of Cycles and Grids in Star Graphs," *Proc. 2nd IEEE Symposium on Parallel and Distributed Processing*, pp. 540-547, 1990.

[K] K. Kwon, "Parallel Computation on the Hypercube-Like Machine," *PhD Thesis*, Department of Computer Science, Louisiana State University, 1991.

[KHI] N. Krishnakumar, V. Hegde, and S. Iyengar, "Fault Tolerant Based Embeddings of Quadtrees into Hypercubes," *Proc. International Conference of Parallel Processing*, 1991.

[KS] H. Kung and D. Stevenson, "A Software Technique for Reducing the Routing Time on a Parallel Computer with a Fixed Interconnection Network," High Speed Computer and Algorithm Optimization, *Academic Press*, pp. 423-433, 1987.

[LBT] C. Liang, S. Battachanya, and W. Tsai, "Distributed Fault-Tolerant Routing on Hypercubes: Algorithms and Performance Study," *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, pp. 474-481, 1991.

[LE] T. Lewis and H. El-Rewini, Introduction to Parallel Computing, *Prentice-Hall*, 1992.

[LE1] S. Latifi and A. El-Amawy, "Efficient Approach to Embed Binary Trees in 3-D Rectangular Arrays," *IEEE Proceedings*, vol. 137, no. 2, pp. 159-163, March 1990.

[Lei] F. Leighton, Introduction to Parallel Algorithms and Architecture: Arrays, Trees, Hypercubes, *Morgan Kaufmann*, 1992.

[Len] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, *John Wiley & Sons*, 1990.

[LF] R. Lander and M. Fischer, "Parallel Prefix Computation," *Journal of the ACM*, vol. 27, pp. 831-838, 1980.

[LZ] S. Latifi and S. Zheng, "Optimal Simulation of Linear Array and Ring Architectures on Multiply-Twisted Hypercubes," *Proc. 11th International Conference on Computers and Communications*, 1992.

[MS] B. Monien and I. Sudborough, "Simulating Binary Trees on Hypercubes," *Proc. 3rd Aegean Workshop on Computing, Lecture Notes in Computer Science*, pp. 170-180, 1988.

[NS] D. Nassimi and S. Sahni, "Data Broadcasting in SIMD Computers," *IEEE Transactions on Computers*, vol. C-30, no. 2, pp. 101-106, February 1981.

[NSK] M. Nigam, S. Sahni, and B. Krishnamurthy, "Embedding Hamiltonian and Hypercubes in Star Interconnection Graphs," *Proc. International Conference on Parallel Processing*, pp. 340-343, 1990.

[P] C. Plaxton, "Efficient Computation on Sparse Interconnection Networks," *PhD Thesis*, Department of Computer Science, Stanford University, 1989.

[PM] F. Provost and R. Melhem, "Distributed Fault-Tolerant Embedding of Binary Trees and Rings in Hypercubes," *Proc. International Workshop on Defect and Fault-Tolerance in VLSI Systems*, 1989.

[PV] F. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of the ACM*, vol. 24, no. 5, pp. 300-309, May 1981.

[Q] M. Quinn, Designing Efficient Algorithms for Parallel Computers, *McGraw-Hill*, 1987.

[QD] M. Quinn and N. Deo, "Parallel Graph Algorithms," *ACM Computing Surveys*, vol. 16, no. 3, pp. 319-348, September 1984.

[U] J. Ullman, Computational Aspects of VLSI, *Computer Science Press*, 1984.

[UW] E. Upfal and A. Wigderson, "How to Share Memory in a Distributed System," *Journal of the ACM*, pp. 116-127, January 1987.

[RS] A. Rosenberg and L. Snyder, "Bounds on the Costs of Data Encodings," *Math. Systems Theory*, vol. 12, pp. 9-39, 1978.

[Sa] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *Computing Surveys*, vol. 16, no. 2, pp. 187-260, June 1984.

[Se] C. Seitz, "The Cosmic Cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22-33, January 1985.

[Si] H. Siegel, Interconnection Networks for Large-Scale Parallel Processing, *Lexington Books*, 1985.

[Sn] L. Snyder, "Introduction to the Configurable Highly Parallel Computer," *Computer*, pp. 47-56, January 1982.

[St] H. Stone, High-Performance Computer Architecture, *Addison-Wesely*, 1987.

[SS] Y. Saad and M. Schultz, "Topological Properties of the Hypercube," *IEEE Transactions on Computers*, vol. C-37, no. 7, pp. 867-872, July 1988.

[T] P. Treleaven, "Control-Driven, Data-Driven, and Demand-Driven Computer Architecture," *Parallel Computing*, no. 2, 1985.

[V] L Valiant, "A Scheme for Fast Parallel Communications," *SIAM J. Computing*, vol. 11, no. 2, pp. 350-361, 1982.

[TK] C. Thompson and H. Kung, "Sorting in a Mesh-Connected Parallel Computer," *Communications of the ACM*, vol. 20, no. 4, pp. 263-271, April 1977.

[W] A. Wu, "Embedding of Tree Networks into Hypercubes," *Journal of Parallel and Distributed Computing*, vol. 2, no. 3, pp. 238-249, August 1985.

[WCM] A. Wang, R. Cypher, and E. Mayr, "Embedding Complete Binary Trees in Faulty Hypercubes," *Proc. 3rd IEEE Symposium on Parallel and Distributed Processing*, pp. 112-119, 1991.

[YN] A. Youssef and B. Narahari, "The Banyan-Hypercube Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 160-169, April 1990.

[Z] S. Zheng, "SIMD Data Communication Algorithms for Multiply-Twisted Hypercubes," *Proc. 5th International Parallel Processing Symposium*, pp. 120-125, 1991.

# Vita

Emadeddin Abuelrub received his BS degrees in Computer Engineering and Computer Science from Oklahoma State University in 1984 and 1985, respectively. He received his MS degree in Computer Science from Alabama A&M University in 1987. He joined the PhD program in the Department of Computer Science at Louisiana State University in 1989, where he worked on parallel algorithms and mappings on parallel machines. His other research interests include the design and analysis of algorithms, graph algorithms, and parallel and VLSI computations.

**DOCTORAL EXAMINATION AND DISSERTATION REPORT**

**Candidate:** Emadeddin Abuelrub

**Major Field:** Computer Science

**Title of Dissertation:** Interconnection Networks Embeddings and Efficient Parallel Computations

**Approved:**

[illegible]
Major Professor and Chairman

[illegible]
Dean of the Graduate School

**EXAMINING COMMITTEE:**

A. Elmaghraby

[illegible]

[illegible]

[illegible]

[illegible]

**Date of Examination:**

05/14/93