This is to certify that the

dissertation entitled

GRAPH EMBEDDING AND DATA COMMUNICATION IN A LARGE FAMILY OF NETWORK TOPOLOGIES

presented by

Jenshiuh Liu

has been accepted towards fulfillment
of the requirements for

DOCTOR degree in PHILOSOPHY

[illegible]
Major professor

Date July 24, 1992

GRAPH EMBEDDING AND DATA COMMUNICATION IN A LARGE FAMILY OF NETWORK TOPOLOGIES

By

Jenshiuh Liu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

August, 1992

697-6553

ABSTRACT

GRAPH EMBEDDING AND DATA COMMUNICATION IN A LARGE FAMILY OF NETWORK TOPOLOGIES

By

Jenshiuh Liu

In a parallel computer or a distributed system the interconnection pattern (*network topology, interconnection topology*) of the communicating modules is one of the key factors that determine the efficiency, reliability, and applicability of the system. Most previous results in the network design area are about a single kind of interconnection topology; relatively few have considered fault-tolerance. Here we present new results about graph embedding and data communication for a large *family* of network topologies, which consists of the generalized Fibonacci cube [63, 80] and the Incomplete Hypercube [68]. This family includes the hypercube as a special case. However, unlike the hypercube, both the generalized Fibonacci cube and the Incomplete Hypercube are *irregular* graphs in general. The Incomplete Hypercube offers unrestrictive network size; however, it suffers from a low degree of fault-tolerance under certain situations. On the other hand, the generalized Fibonacci cube offers structural recurrences and also provides a guaranteed degree of fault tolerance. This family of network topologies has two major areas of application: (1) Fault Tolerance – each member network serves as an alternative structure for reconfiguring a hypercube in the presence of faults, and (2) Incremental Expansion – allowing a system to add nodes in small increments, which is an important advantage for systems that evolve with time.

To investigate their applications in parallel or distributed systems, we present a collection of provably efficient *graph embedding* and *data communication* algorithms for this large family of network topologies. The study of graph embedding is to determine whether these networks can flexibly simulate other common networks. The results can be applied to the emulation of a given network on a family of networks; analysis of these results offers insights about the structural relationships between different networks. On the other hand, the study of common communication primitives provides efficient tools for developing application algorithms for this family of networks; it also offers insights about the relations between *forms* (structural properties) and *functions* (algorithms). Specifically, we demonstrate how to embed linear array, ring, mesh, and hypercube on the generalized Fibonacci cube; we also show how to embed linear array and ring on the Incomplete Hypercube. In addition, efficient algorithms for single node broadcasting/accumulation, single node scatter/gather, and multinode broadcasting/accumulation are presented for both the generalized Fibonacci cube and the Incomplete Hypercube. All of these algorithms are carefully analyzed under both the all- and one-port models.

To my parents and my wife

ACKNOWLEDGEMENTS

I would like to thank Professor Herman D. Hughes, my thesis co-advisor, for his guidance, intellectual advice and support. Many thanks to Dr. Moon Jung Chung for his suggestions and careful reading the manuscript. I am grateful to my other guidance committee members, Professor Carl V. Page, Professor T. Y. Li and Professor Richard O. Hill, for their help, encouragement and guidance.

I am indebted to my wife Bi-Yun for her support, understanding and encouragement during my stay at MSU. Without her love and support, this thesis would not have been possible. I am grateful to my parents for their support, to my sons Sean and Andy for giving me a lot of joy and inspiration during the writing of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

High performance computers have become indispensable for many application areas such as engineering design, system modeling and simulation, medical and basic science research [4] [13] [27] [30] [35] [49] [65] [74] [104] [114] . Two fundamental approaches have been employed to improve the speed of computers. The first approach is to increase the speed of the circuitry used in building the computers. The second approach, which is commonly called *parallel processing,* is to use multiple processors in executing multiple operations concurrently. The speed of light imposes a fundamental limit on the first approach, whereas the number of processors is virtually unlimited. Hence parallel processing appears to be the more promising approach.

There are two basic models for studying the parallel computers. In the Parallel Random Access Machine (PRAM) model [45] (also referred to as the Shared-Memory model), multiple processors can randomly read or write any locations of a shared memory, and processors communicate by writing/reading through the shared memory. On the other hand, in the Distributed-Memory (DM) model, a memory module is associated with each processor, and two processors communicate by explicitly sending/receiving *messages.* With the current technology, it is sometimes criticized that the PRAM model is unrealistic because for a large system the cost of the interconnection network will become prohibitively high [2]. Therefore, the DM model is much more suitable for a large-scale parallel system. Since a DM model based on the complete graph may not be cost-effective, networks with relatively sparse interconnections are often used for the DM systems. There are many types of networks, and each one offers different performance. Therefore, it is important to investigate the relations between the structural characteristics and the performance under common functional criteria.

1.1 Network Topology

Graphs are often used to model the *network topology* (*i.e.*, the pattern of the interconnections) of the DM parallel/distributed systems, where a node represents a processor and an edge (link) between two nodes denotes the communication link between the corresponding processors. The following criteria are commonly applied to evaluate network topologies for the DM multiprocessor systems.

1. **Degree.** The *degree* of a graph is the maximum number of edges incident on a node. A network of a large degree usually implies more alternative paths between a pair of nodes and hence a larger bandwidth. However, a large degree may also result in a higher hardware cost or difficulty in implementation.

2. **Diameter.** The *distance* between a pair of nodes is the smallest number of edges that have to be traversed in order to reach one node from the other. The *diameter* of a graph G is the maximum distance between any pair of nodes in G. The diameter is often a lower bound on the time required to perform certain calculations on a network. This is because one processor i may need the information residing on another processor j, which has to be transferred from j to i. Therefore, the distance between two nodes determines a lower bound of the communication time. In general, a small diameter is desirable.

3. **Bisection width.** The bisection width of a graph G with N nodes is the minimum number of edges that have to be removed in order to dissect G into two subgraphs with $\lceil N/2 \rceil$ and $\lfloor N/2 \rfloor$ nodes respectively. The bisection width is often a critical factor in determining the speed with which a network can perform computations that involve a major exchange of information between two parts of the network. For example, to sort a list of data in a network into certain order, the data items residing in one part of the network may need to be swapped with those residing in the other part. Clearly, under this situation, this operation has to be achieved through the set of links connecting the two halves. Therefore, the bisection width of the network determines a lower bound on the time required for sorting on this network.

4. **Expansibility.** The expansibility of a network measures the ease to add or delete nodes to/from the network. This consideration is important especially when the network is still evolving or when it is necessary to construct the system incrementally.

5. **Fault-tolerance.** With a large number of processors/links in systems, the probability of having a processor or link failure increases significantly. It is desirable to have a network that provides independent paths between all pairs of nodes and hence a higher degree of fault-tolerance.

Notice that some of the criteria mentioned above are actually related or even in conflict, hence they cannot be improved without compromising the others. For example, a low degree usually translates into a large diameter, low bisection width, fewer alternative paths and hence low fault-tolerance. The designer, therefore, will have to balance between the cost constraints and the performance requirements of the network.

1.2 Motivation

A graph is *regular* if each node has the same degree [108]. Most network topologies for the DM model are based on regular graphs [2] [46] [75] [97]. Very limited knowledge has been accumulated for irregular network topologies for lack of appropriates tools. The driving force behind this work is to further our knowledge on network topologies that display less regularity. Also, instead of proposing a set of unrelated networks, we aim to provide a family of networks, where the interconnection patterns and the performance features can be characterized by a set of parameters.

The *family* of network topologies studied here consists of the *generalized Fibonacci cube* [63, 80] and the *Incomplete Hypercube* [68]. Both the Generalized Fibonacci Cube (GFC) and the Incomplete Hypercube (IHC) are irregular topologies in general. However, both of them contain the hypercube (HC) as a subset; moreover, each member of this family (*i.e.*, a generalized Fibonacci cube or an Incomplete Hypercube) is a subgraph of a hypercube. Figure 1.1 shows the constitution of this family (see Chapter 2 for details). The generalized Fibonacci cube stems from the *Fibonacci cube* (FC) proposed by Hsu [61, 62], which is also a subset of the generalized Fibonacci cube. In this thesis, we will study efficient *graph embedding* and *data communication*

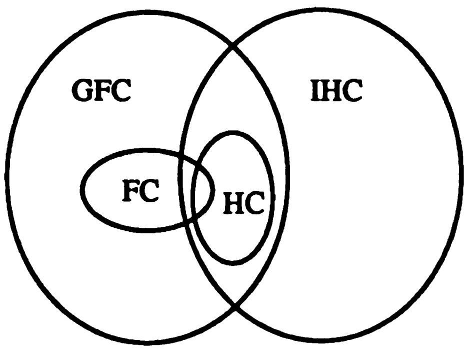

Figure 1.1. The composition of the family of network topologies.

for the members of this family.

1.2.1 Hypercube

Numerous network topologies have been proposed and studied. Among them, the trees, meshes, meshes of trees, and the hypercube have drawn the most attentions (see, for example, [2] [46] [75] [97]). Many useful computations, such as matrix operations [2] [75] [97] and image processing [32] [33] [91], can be naturally mapped into trees, meshes, and meshes of trees. However, trees, meshes, and meshes of trees suffer from a low bisection width or a large diameter, which prevents the development of logarithmic-time parallel algorithms for fundamental applications such as sorting.

The hypercube (or Boolean cube, binary n-cube, n-cube) of dimension n (denoted by B_n) consists of 2^n nodes and $n2^{n-1}$ edges. The nodes are labeled from 0 to $2^n - 1$ and each node has an n-bit binary code (called *address*) such that two nodes are connected if and only if the binary codes of their labels differ in exactly one bit. Recently, the hypercube has become a popular network topology for parallel processing [55] [103] [106], because it has many appealing properties such as the logarithmic diameter and node degree, high bisection width, and ease to embed other common structures [19] [20] [39] [44] [60] [75] [101] [112] [113]. Intel's iPSC, NCUBE, Ametek 14 and FPS-T series are a few examples of commercially available parallel computers based on the hypercube topology. Many applications have been successfully developed for the hypercube, *e.g.*, image processing [31] [98] [99], matrix algebra [18] [36] [47], Particle-in-Cell (PIC) [86], and others [46].

Although the hypercube is a powerful and popular topology for parallel computation, different variations have been proposed to improve certain characteristics, such as the diameter and node degree, of the hypercube. The approaches can be generally divided into the following three categories: (1) *twisted hypercube* [42] [56], (2) *enhanced hypercube* [40] [41] [111], and (3) bounded-degree hypercube-derivatives *e.g.*,

butterfly*, *cube-connected cycles (CCC) [96], and ***shuffle-exchange*** [105]. These networks provide either smaller diameter, or shorter average internode distance, or low node degree. However, their permissible sizes are generally restricted to powers of two. As such, they have lower expansibility than the networks which we are about to study.

1.2.2 Incomplete Hypercube and Generalized Fibonacci Cube

The *Incomplete Hypercube* proposed by Katseff [68] improves the expansibility of the hypercube. The Incomplete Hypercube with N nodes (denoted by I_N), where N may be any positive integer, is constructed in the same way as the hypercube. In other words, nodes are labeled from 0 to $N-1$, and two nodes are connected by an edge if and only if their binary representations differ in exactly one bit. It can be shown that I_N is a subgraph of B_n where $n = \lceil \log N \rceil$.* Algorithms for routing and single node broadcasting have been devised by Katseff [68]; Tzeng [109] has also analyzed some structural properties, such as diameter, internode distance and traffic density, of the Incomplete Hypercube, which are shown to be comparable with that of the hypercube. Furthermore, Tzeng et al. [110] have shown that binary trees and two-dimensional (2D) meshes can be embedded in the Incomplete Hypercube. These existing results suggest that the Incomplete Hypercube is a useful network topology. However, the Incomplete Hypercube suffers from two shortcomings. One is low degree of fault tolerance under certain situations. For example, a node may have a degree of *one* when an Incomplete Hypercube contains exactly $2^n + 1$ nodes. The other shortcoming is the lack of recurrence in its structure, which makes the designing of certain application algorithms in the Incomplete Hypercube much more difficult.

*In this thesis, all logs are to the base 2.

Hsu [61, 62] has proposed the Fibonacci cube based on the Fibonacci numbers [53, 71]. Recently, he also presented the generalized Fibonacci cube [63] based on the *generalized Fibonacci numbers* (see, *e.g.*, [72]). The generalized Fibonacci cube can be defined by two integral parameters n and k where $n \geq k \geq 2$. The k^{th} *order Fibonacci number* is defined by $F_n^{[k]} = F_{n-1}^{[k]} + F_{n-2}^{[k]} + \cdots F_{n-k}^{[k]}$ for $n \geq k$, and $F_0^{[k]} = F_1^{[k]} = \cdots = F_{k-2}^{[k]} = 0, F_{k-1}^{[k]} = 1$. Informally, the k^{th} *order Fibonacci cube of dimension* n (denoted by Γ_n^k) consists of $F_n^{[k]}$ nodes; the nodes are numbered from 0 to $F_n^{[k]}-1$ and each node has a unique address, called *generalized Fibonacci code,* such that two nodes are connected by an edge if and only if their generalized Fibonacci codes differ in exactly one bit. The generalized Fibonacci cube bears two important features: (1) Γ_n^k can be decomposed into k subcubes: $\Gamma_{n-1}^k, \Gamma_{n-2}^k, \cdots, \Gamma_{n-k}^k$ which are of sizes $F_{n-1}^{[k]}, F_{n-2}^{[k]} \cdots, F_{n-k}^{[k]}$, respectively, and (2) Every node has a logarithmic number of degree (as a function of the total number of nodes). Therefore, the generalized Fibonacci cube offers structural recurrences and an improved *fault tolerance* over the Incomplete Hypercube.

1.2.3 Graph Embedding and Data Communication

When using DM parallel computer systems, we often partition an application into many tasks and map (assign) the tasks to the processors. The interaction among tasks located on different processors are achieved through sending and receiving messages. This inter-processor communication is usually modeled by a graph which will be referred to as the *guest* graph. On the other hand, the graph that represents the network topology of a given parallel system will be referred to as the *host* graph.

Let $G = (V_G, E_G)$ denote a guest graph and $H = (V_H, E_H)$ a host graph, where V_G and V_H denote the node sets of the guest and host graphs respectively, and E_G and E_H edge sets. The embedding of G in H is a function f that maps each node in G into a node in H. *Dilation* and *expansion* are two common metrics used in evaluating graph embeddings: the dilation of the embedding is $max(dis(f(i), f(j)), \forall(i,j) \in E_G)$, and the expansion of the embedding is $\frac{|V_H|}{|V_G|}$, where $|V_G|$ denotes the cardinality of the set V_G, $|V_H|$ the cardinality of V_H, and $dis(i,j)$ the distance between two nodes i and j.

A parallel computer should allow the users to write programs based on the algorithmic structures (guest graphs) rather than the underlying structure of the system (host graph). Each application usually dictates a certain guest graph when it is mapped into a parallel computer system. Our study of graph embedding can provide useful tools for emulating various guest graphs on a given host graph. Furthermore, the study of graph embedding will provide insight about the relations among different network topologies.

Advances in the VLSI technology have enabled us to construct parallel computer systems consisting of hundreds or thousands processors. This trend has contributed to an ever finer granularity in parallelism. Consequently, the computation per communication is becoming smaller and smaller. With the current technology, it is easier to build a high speed processor than to construct a high bandwidth interconnection network. Therefore, efficient data communications are critical for high performance parallel computers. In this thesis, we will demonstrate the effectiveness of the family of network topologies by designing tools for graph embedding and data communication in it.

1.3 Overview

We now give a brief overview of this thesis.

In Chapter 2, we review the Incomplete Hypercube and some of its properties. Then we formally define the generalized Fibonacci cube, characterize its recurrence relation, and study some of its basic properties such as the number of edges and node degrees.

In Chapter 3, we present some graph embeddings in the generalized Fibonacci cube. The *Hamiltonian problem* is to determine whether a graph contains a spanning (Hamiltonian) path or cycle. In [52], it is considered as one of the fundamental problems in graph theory. The linear array and ring are two common structures which have many applications in parallel processing [2] [10] [75] [97]. We will show that Γ_n^k contains a Hamiltonian path, which implies the embedding of linear array. Then, by decomposing the generalized Fibonacci cube into subcubes and properly rearranging the Hamiltonian paths in the subcubes, we prove that each member of the generalized Fibonacci cube contains a Hamiltonian cycle if and only if it has an even number of nodes; on the other hand, for members which do not contain a Hamiltonian cycle, the longest cycles contain all but *one* node. This result implies that a ring of size $F_n^{[k]}$ can be embedded in Γ_n^k with a dilation no more than two. The mesh structure is commonly used in many applications [2] [10] [75] [97]. Therefore, we will also present the 2D and 3D mesh embeddings. In addition, we show that the hypercube can be embedded in a generalized Fibonacci cube, even though each generalized Fibonacci cube is a subgraph of a hypercube.

In Chapter 4, we study the data communication in the generalized Fibonacci cube, where we devise and analyze efficient algorithms for the following data communication primitives:

1. *Routing:* a node sends a message to another node,
2. *Single node broadcasting* (*accumulation*): a node sends a message to all the other nodes (receives messages from all the other nodes),
3. *Single node scatter* (*gather*): a node sends different messages to all the other nodes (receives one message from all the other nodes), and
4. *Multinode broadcasting* (*accumulation*): every node sends a message to all the other nodes (receives a message from all the other nodes).

These primitives are studied because broadcasting is used in many algebraic computations, see, for example [18] [48] [66]; accumulation can be applied to, for instance, processor synchronization [94]; scatter and gather are commonly used in, for example, prefix computation and tree contraction [1] [12] [14] [64] [73] [88] [89] [90]. As in [9, 67], we will analyze all of our data communication algorithms under two different models. In the *one-port* model, one assumes that only one port on every node can be used for sending and receiving data (*one-port* model) at any moment. On the other hand, in the *all-port* model, one assumes that all the communication ports on every node can be used for sending and receiving data at any time.

In Chapter 5, we consider the Hamiltonian problem and data communication in the Incomplete Hypercube. By employing the Binary Reflected Gray code (see, *e.g.*, [25, 101]), we show that, for $N \geq 4$, I_N contains a Hamiltonian cycle if and only if N is even, whereas the longest cycle contains all but *one* node if and only if N is odd. Consequently, I_N contains a Hamiltonian path if N is even. On the other hand, when N is odd, we are still able to construct a Hamiltonian path by substituting one edge in the longest cycle with another one. Therefore, we have optimal embeddings of linear array and ring in the Incomplete Hypercube. For data communication, we demonstrate that all of the communication primitives mentioned earlier (single node broadcasting and accumulation, single node scatter and gather, and multinode broadcasting and accumulation) can be also efficiently implemented in the Incomplete Hypercube under both the one- and all-port models.

In Chapter 6, we summarize our main results, highlight contributions, and discuss some directions for future work.

CHAPTER 2

PRELIMINARIES

In this chapter, we present the basic properties of the family of network topologies. As mentioned earlier, the family of network topologies of our interest consists of both the Incomplete Hypercube [68] and the generalized Fibonacci cube [63, 80]. A graph G is represented by (V, E), where V denotes the set of *nodes* and E the set of *edges*. The *Hamming distance* between two binary strings is the number of bits where these two strings differ.

2.1 Incomplete Hypercube

The Incomplete Hypercube was first proposed by Katseff [68]. An Incomplete Hypercube with N nodes (where N may be any positive integer) is constructed in the same way as the hypercube, *i.e.*, two nodes are linked if and only if the Hamming distance between their addresses (*i.e.*, binary codes) is exactly one. Formally, let $H(i,j)$ denote the Hamming distance between the binary representations of two integers i and j. The Incomplete Hypercube can be defined as follows.

Definition 2.1 [68] *Let $N \geq 1$ denote an integer. The Incomplete Hypercube with N nodes, denoted by I_N, is a graph (V, E), where $V = \{0, 1, \cdots, N-1\}$ and $E = \{(i,j) \mid H(i,j) = 1, for\ 0 \leq i,j \leq N-1\}$*

Figure 2.1 shows the structure of I_6, where nodes are labeled by their addresses. The

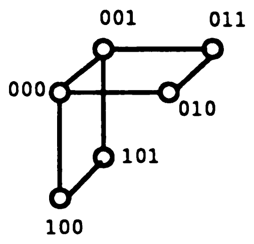

Figure 2.1. Structure of I_6.

following two lemmas are important in order to see the applications of the Incomplete Hypercube.

Lemma 2.1 *I_N is a subgraph of I_{N+1} for any $N \geq 1$.*

Lemma 2.2 [68] *I_N is a subgraph of B_n where $n = \lceil logN \rceil$.*

With Lemma 2.1, we can build the Incomplete Hypercube incrementally. In other words, for example, to have an I_4, we may construct I_2 first, then add one node and one edge to form I_3, and finally add one more node and edge. On the other hand, Lemma 2.2 suggests that the Incomplete Hypercube may have fault-tolerant applications in the hypercube. For example, when any single node fails in B_3, the remaining network can still be used as an I_7.

2.2 Generalized Fibonacci Cube

In this section, we will define the generalized Fibonacci cubes. For this purpose, we first examine the *generalized Fibonacci numbers* (see *e.g.*, [72]). The k^{th} order Fibonacci numbers, denoted by $F_n^{[k]}$, are defined by the following equation

$$F_n^{[k]} = F_{n-1}^{[k]} + F_{n-2}^{[k]} + \cdots + F_{n-k}^{[k]}, \quad \text{for} \quad n \geq k;$$

where

$$F_i^{[k]} = 0 \quad \text{for} \quad 0 \le i \le k-2, \quad \text{and} \quad F_{k-1}^{[k]} = 1.$$

In other words, each number in the sequence is the sum of the preceding k numbers. Table 2.1 lists the first 10 non-zero generalized Fibonacci numbers of orders 2 to 10. Notice that we have $F_{k-1}^{[k]} = F_k^{[k]}$ for $k \ge 2$; the usual Fibonacci numbers are obtained

Table 2.1. Example of generalized Fibonacci numbers.

i	$F_{i+2}^{[2]}$	$F_{i+3}^{[3]}$	$F_{i+4}^{[4]}$	$F_{i+5}^{[5]}$	$F_{i+6}^{[6]}$	$F_{i+7}^{[7]}$	$F_{i+8}^{[8]}$	$F_{i+9}^{[9]}$
-1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1
1	$\underline{2}$	2	2	2	2	2	2	2
2	3	$\underline{4}$	4	4	4	4	4	4
3	5	7	$\underline{8}$	8	8	8	8	8
4	8	13	15	$\underline{16}$	16	16	16	16
5	13	24	29	31	$\underline{32}$	32	32	32
6	21	44	56	61	63	$\underline{64}$	64	64
7	34	81	108	120	125	127	$\underline{128}$	128
8	55	149	208	236	248	253	255	$\underline{256}$

when $k = 2$. Furthermore, it is not difficult to see that $F_i^{[k]} = 2^{i-k}$ (*i.e.*, $F_i^{[k]}$ is a power of 2) for $2 \le k \le i \le 2k-1$. We next introduce the Fibonacci codes which enable us to define the generalized Fibonacci cubes.

Definition 2.2 *Assume that $n \ge k \ge 2$. Let i be an integer where $0 \le i < F_n^{[k]}$. The k^{th} order* **Fibonacci code** *(k-FC) of i, denoted by $FC_n^{[k]}(i)$, is a sequence of $n-k$ binary numbers $(b_{n-k-1}, \ldots, b_1, b_0)$ where*

1. $i = \sum_{j=0}^{n-k-1} b_j \cdot F_{j+k}^{[k]}$, and

2. $b_j \cdot b_{j+1} \cdot \ldots \cdot b_{j+k-1} = 0$, for $0 \le j \le (n-2k)$.

In the following, we write $FC(i)$ for $FC_n^{[k]}(i)$ if k and n can be explicitly deduced from context. We adopt the following convention to number a binary string. The rightmost (least-significant) bit of a binary string is referred to as bit 0 and the second to the rightmost as bit 1, and so on. The following two observations are immediate from the preceding definition:

1. bit j in the k^{th} order Fibonacci code (k-FC) carries the weight of $F_{j+k}^{[k]}$, and
2. No k or more consecutive 1's appear in k-FCs.

These properties distinguish the Fibonacci codes from the (base-2) binary codes. In the ***generalized Fibonacci number system*** [17, 72], every nonnegative integer has a unique representation as a sum of distinct k^{th} order Fibonacci numbers ($F_i^{[k]}$, for $i \geq k$) such that no k consecutive Fibonacci numbers are used. The k-FC of an integer I can be obtained by the following greedy approach: find the greatest $F_j^{[k]}$ that is less than or equal to I, assign 1 to bit $j - k$, then proceed recursively on $I - F_j^{[k]}$; finally, all unassigned bits are set to 0's. Table 2.2 lists 2-FCs and 3-FCs for all integers from 0 up to 12.

Table 2.2. Examples of 2-FCs and 3-FCs.

i	$FC^{[2]}(i)$	$FC^{[3]}(i)$	i	$FC^{[2]}(i)$	$FC^{[3]}(i)$
0	00000	0000	7	01010	1000
1	00001	0001	8	10000	1001
2	00010	0010	9	10001	1010
3	00100	0011	10	10010	1011
4	00101	0100	11	10100	1100
5	01000	0101	12	10101	1101
6	01001	0110			

The definition of the generalized Fibonacci cubes is based on the Fibonacci codes and the Hamming distance.

Definition 2.3 *The k^{th} order* **Fibonacci cube** *of dimension n, denoted by Γ_n^k (where $n \geq k \geq 2$), is a graph (V_n^k, E_n^k), where $V_n^k = \{0, 1, .., F_n^{[k]} - 1\}$ and $E_n^k = \{(i,j) \mid H(FC_n^{[k]}(i), FC_n^{[k]}(j)) = 1, 0 \leq i, j < F_n^{[k]}\}$. Define $\Gamma_i^k = (\phi, \phi)$ for $0 \leq i \leq k-2$, and $\Gamma_{k-1}^k = (\{0\}, \phi)$.*

Recall that $F_{k-1}^{[k]} = F_k^{[k]}$ for $k \geq 2$. Definition 2.3 implies that both Γ_{k-1}^k and Γ_k^k represent the same graph and they contain exactly one node. Figure 2.2 shows the structure of the second-order Fibonacci cubes (or Fibonacci cubes as in [61, 62]) of dimensions 2 to 6, where nodes are labeled by 2-FCs. It can be seen that Γ_n^2 can be partitioned into *two* subcubes: one subgraph (induced by the black nodes in Figure 2.2) is isomorphic to Γ_{n-1}^2 and the other (induced by the white nodes) is isomorphic to Γ_{n-2}^2. Figure 2.3 shows some of the third-order Fibonacci cubes. Similarly, Γ_n^3 can be partitioned into *three* subcubes: the first one (the black nodes in Figure 2.3) is isomorphic to Γ_{n-1}^3, the second (grey nodes) to Γ_{n-2}^3 and the third (white nodes) to Γ_{n-3}^3. The following observation is important: Γ_n^k is a subgraph of B_{n-k} for $n \geq k$. In the following section, we will prove this observation and other properties of the generalized Fibonacci cubes.

2.3 A Characterization of Generalized Fibonacci Cube

It is clear that the definition of the generalized Fibonacci cubes is analogous to that of the Boolean cubes. In this section, we study some basic properties of the generalized Fibonacci cubes and their relations with the Boolean cubes. It is necessary to examine the properties of k-FCs at this point. For convenience, we view each k-FC as a string.

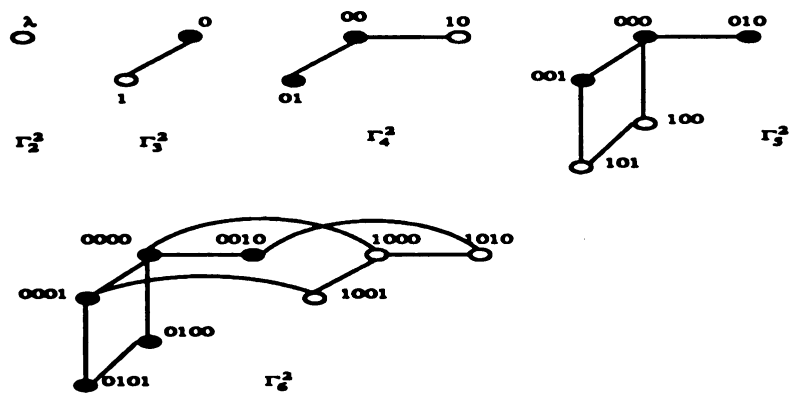

Figure 2.2. Second-order Fibonacci cube of dimensions 2 to 6.

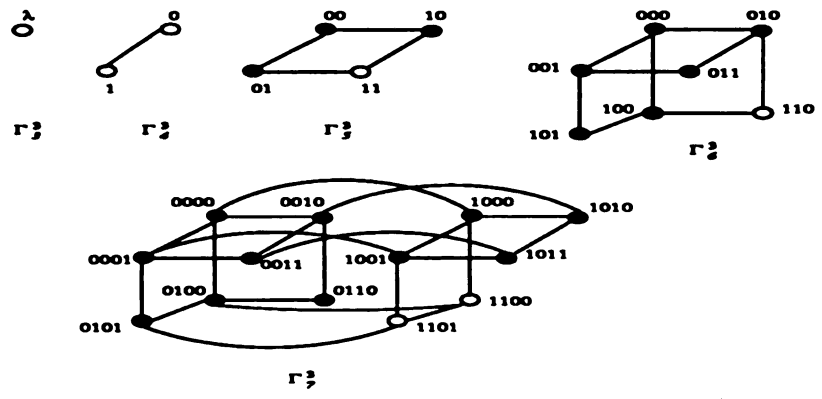

Figure 2.3. Third-order Fibonacci cube of dimensions 3 to 7.

Let α and β denote two strings, and let λ represent the null string. The following notations will be used: the *concatenation* of two strings α and β is denoted by $\alpha \cdot \beta$, or simply $\alpha\beta$. Define $\lambda \cdot \alpha = \alpha \cdot \lambda = \alpha$. We use α^i to represent the concatenation of a string α with itself for i times, where $i \geq 1$; define $\alpha^0 = \lambda$. For a set of strings S, define $\alpha \cdot S = \{\alpha \cdot \beta \mid \beta \in S\}$.

Let $\mathcal{V}_n^k$ denote the set of k-FCs for integers between 0 and $F_n^{[k]} - 1$, where $n > k$. Define $\mathcal{V}_{k-1}^k = \mathcal{V}_k^k = \{\lambda\}$.* It should be clear that the nodes in Γ_n^k can be labeled

*Notice that Γ_{k-1}^k and Γ_k^k each contains exactly one node. We use λ to address the node in this case.

exactly by the codes in $\mathcal{V}_n^k$. From Table 2.2, we can observe that $\mathcal{V}_3^2 = \{0,1\}$, $\mathcal{V}_4^2 = \{00,01,10\}$, and $\mathcal{V}_5^2 = \{000,001,010,100,101\} = \{000,001,010\} \cup \{100,101\} = 0\cdot \mathcal{V}_4^2 \cup 10\cdot \mathcal{V}_3^2$. Moreover, $\mathcal{V}_3^3 = \{\lambda\}$, $\mathcal{V}_4^3 = \{0,1\}$, and $\mathcal{V}_5^3 = \{00,01,10,11\}$, and $\mathcal{V}_6^3 = \{000,001,010,011,100,101,110\} = \{000,001,010,011\} \cup \{100,101\} \cup \{110\} = 0\cdot \mathcal{V}_5^3 \cup 10\cdot \mathcal{V}_4^3 \cup 110\cdot \mathcal{V}_3^3$. This observation leads to the following lemma that describes an important recurrence of the set $\mathcal{V}_n^k$.

Lemma 2.3 *Assume that $k \geq 2$, and $n \geq 2k$. Let $\mathcal{V}_n^k$ denote the set of k-FCs for integers between 0 and $F_n^{[k]} - 1$. Then*

$$\mathcal{V}_n^k = 0 \cdot \mathcal{V}_{n-1}^k \cup 10 \cdot \mathcal{V}_{n-2}^k \cup \cdots 1^{k-2}0 \cdot \mathcal{V}_{n-k+1}^k \cup 1^{k-1}0 \cdot \mathcal{V}_{n-k}^k.$$

Proof: By induction on n. □

We next examine the recurrence of the generalized Fibonacci cubes. By definition, Γ_n^k contains $F_n^{[k]}$ nodes. Recall that $F_n^{[k]}$ is defined as the sum of the k preceding generalized Fibonacci numbers. Lemma 2.4 shows that Γ_n^k contains k disjoint subgraphs that are isomorphic to Γ_{n-1}^k , Γ_{n-2}^k, $\cdots$, and Γ_{n-k}^k respectively; moreover, each node in a subcube of dimension j is connected to one node in every subcube of dimension i, provided $j < i$.

Lemma 2.4 *Assume that $k \geq 2$ and $n \geq 2k$. Define $\mathcal{G}_n^k(i) = 1^i 0 \cdot \mathcal{V}_{n-1-i}^k$, and $G_n^k(i)$ the subgraph induced by $\mathcal{G}_n^k(i)$ in Γ_n^k, where $0 \leq i < k$. Then Γ_n^k can be partitioned into $G_n^k(0)$, $G_n^k(1)$, $\cdots$,$G_n^k(k-1)$ such that*

(1) $G_n^k(i)$ is isomorphic to Γ_{n-1-i}^k for $0 \leq i < k$, and

(2) For two nodes x and y in Γ_n^k such that x in $G_n^k(i)$, and y in $G_n^k(j)$ and $0 \leq i < j < k$, the edge (x,y) is in Γ_n^k if and only if $y - x = F_{n-1-i}^{[k]}$.

Proof: Property (1) is an immediate result of Lemma 2.3. It remains to show property (2). The *if* part follows from Definition 2.3. We need to prove the *only if* part. By definition, we have $\mathcal{G}_n^k(i) = 1^i0\ \mathcal{V}_{n-1-i}^k$ and $\mathcal{G}_n^k(j) = 1^j0\ \mathcal{V}_{n-1-j}^k$. By recursively applying Lemma 2.3 on $\mathcal{V}_{n-1-i}^k$, we can deduce that $\mathcal{G}_n^k(i)$ contains $1^i01^{j-i-1}0$ $\mathcal{V}_{n-1-j}^k$. Notice that $\mathcal{G}_n^k(j)$ can be rewritten as $1^i11^{j-i-1}0\ \mathcal{V}_{n-1-j}^k$. Since (x, y) is an edge in Γ_n^k, they must differ in the $(n-k-1-i)^{th}$ bit. Therefore, the difference between y and x is $F_{n-1-i}^{[k]}$. □

Example 2.1 Let $k = 3$ and $n = 6$. We have $\mathcal{G}_6^3(0) = \{000, 001, 010, 011\}$, $\mathcal{G}_6^3(1) = \{100, 101, \}$ and $\mathcal{G}_6^3(2) = \{110\}$. Figure 2.3 shows that Γ_6^3 can be partitioned into $G_6^3(0)$ (black nodes), $G_6^3(1)$ (gray nodes) and $G_6^3(2)$ (white node), which are, respectively, isomorphic to Γ_5^3 , Γ_4^3 and Γ_3^3. Furthermore, $G_6^3(1)$ is connected to $G_6^3(0)$ by the edges (100, 000) and (101, 001); $G_6^3(2)$ is connected to $G_6^3(0)$ by the edge (110, 010), and $G_6^3(2)$ is connected to $G_6^3(1)$ by the edge (110, 101).

The relation among different generalized Fibonacci cubes is captured by the following lemma.

Lemma 2.5 *Assume that $k \geq 2$ denotes an integer.*

(1) Γ_n^k is a subgraph of Γ_{n+1}^k for $n \geq k$, and

(2) Γ_n^k is a subgraph of Γ_n^{k+1} for $n > k$.

Proof: By Definition 2.3. Omitted. □

Lemma 2.5 has the following two implications.

1. We may build a generalized Fibonacci cube incrementally. In other words, for example, to construct Γ_8^4, we can first build Γ_6^2 then adding nodes and edges to form Γ_7^2, Γ_7^3, and finally Γ_8^4.

2. The generalized Fibonacci cube defines a class of network topologies with a large alternatives for fault-tolerance. This is because when certain nodes and/or links

fail in Γ_n^k, we may form a $\Gamma_{n'}^{k'}$ (where $2 \leq k' \leq k$ and $k < n' \leq n$) from the remaining nodes and links.

It is known that nodes in B_{n-k} can be addressed by $(n-k)$-bit binary codes where $n > k$, similarly it is seen in Section 2.2 that nodes in Γ_n^k can also be addressed by $(n-k)$-bit Fibonacci codes. Since the set of m-bit Fibonacci codes is a subset of all m-bit binary codes, we have the following relations between the generalized Fibonacci cube and the Boolean cube.

Lemma 2.6 *Assume that $k \geq 2$ denotes an integer.*

(1) Γ_n^k is isomorphic to B_{n-k} for all n, where $k \leq n < 2k$.

(2) Γ_n^k is a proper subgraph of B_{n-k} for all $n \geq 2k$.

One way to obtain Γ_n^k from B_{n-k} is to remove every node (along with their incident edges) whose binary address is not a k-FC. Lemma 2.6 also indicates that the generalized Fibonacci cube may have fault-tolerant applications for the hypercube.

We now closely re-examine Fig. 1.1 that shows the constitution of the family of network topologies under our study. Recall that $F_n^{[k]} = 2^{n-k}$ for $2 \leq k \leq n \leq 2k-1$. Therefore, by varying k and n, it can be shown that the hypercube is a special case (subset) of the generalized Fibonacci cube. Furthermore, $F_{2k}^{[k]} = 2^k - 1$ for $k \geq 2$. Hence, when $2 \leq k \leq n \leq 2k$, Γ_n^k (which contains $F_n^{[k]}$ nodes) and $I_{F_n^{[k]}}$ are isomorphic, because both are either a hypercube or a hypercube with one node missing.

2.4 Basic Properties of Generalized Fibonacci Cube

In this section we study some basic properties of the generalized Fibonacci cube. In particular, we will count the number of edges and the node degrees. The number of edges is an important metric in studying a network topology because it is a factor related to the cost of the network. On the other hand, the minimal degree of a topology usually sets a bound on the connectivities of a graph, which is a prime factor in studying fault-tolerance.

2.4.1 Number of Edges

In this section, we count the number of edges in the generalized Fibonacci cubes. Let $E_n^{[k]}$ denote the number of edges in Γ_n^k. By Lemma 2.4, Γ_n^k consists of k subgraphs $G_n^k(i)$, where $0 \leq i < k$. The recurrence equation of $E_n^{[k]}$ is

$$E_n^{[k]} = E_{n-1}^{[k]} + E_{n-2}^{[k]} + \cdots + E_{n-k}^{[k]} + F_{n-2}^{[k]} + 2F_{n-3}^{[k]} + \cdots + (k-1)F_{n-k}^{[k]} \qquad (2.1)$$

where

$$E_i^{[k]} = 0 \text{ for } 0 \leq i \leq k \text{ and } E_{k+1}^{[k]} = 1.$$

To see Eq. (2.1), we note that the first k terms account for the number of edges in subgraphs $G_n^k(i)'s$ and the remaining $k-1$ terms account for the number of edges among these subgraphs. For example, the subgraph $G_n^k(0)$ has $E_{n-1}^{[k]}$ edges, and there are $F_{n-2}^{[k]}$ edges connecting the subgraphs $G_n^k(1)$ to $G_n^k(0)$.

We now solve Eq. (2.1). For convenience, we introduce the *generating functions* (see *e.g.*, [53]) for the generalized Fibonacci numbers. Let $G^{[k]}(z)$ denote the generating function for the k^{th} order Fibonacci numbers, where $k \geq 2$. It is known (see *e.g.*, [72]) that

$$G^{[k]}(z) = \frac{z^{k-1}}{1 - z - z^2 - \cdots - z^k} \qquad (2.2)$$

We will first solve the case when $k = 2$. In this case, we have

$$E_n^{[2]} = E_{n-1}^{[2]} + E_{n-2}^{[2]} + F_{n-2}^{[2]} \tag{2.3}$$

Let $H^{[2]}(z)$ denote the generating function for $E_n^{[2]}$*'s*, and define $E_i^{[2]} = 0$ for $i < 0$. By multiplying both sides of Eq. (2.3) with z^n and taking summations, we have

$$\begin{aligned} H^{[2]}(z) &\equiv \sum_{n=0}^{\infty} E_n^{[2]} z^n \\ &= \sum_{n=0}^{\infty} E_{n-1}^{[2]} z^n + \sum_{n=0}^{\infty} E_{n-2}^{[2]} z^n + \sum_{n=0}^{\infty} F_{n-2}^{[2]} z^n \\ &= zH^{[2]}(z) + z^2 H^{[2]}(z) + z^2 G^{[2]}(z) \\ &= \frac{1}{1 - z - z^2} z^2 G^{[2]}(z) \end{aligned}$$

where

$$G^{[2]}(z) = \frac{z}{1 - z - z^2}$$

as we saw in Eq. (2.2). Further calculation leads to

$$H^{[2]}(z) = z[G^{[2]}(z)]^2 \tag{2.4}$$

It can be shown that $H^{[2]}(z)$ is a convolution of the function $G^{[2]}(z)$ with itself and followed by a shift. Therefore, we can solve Eq. (2.4) and get

$$E_n^{[2]} = \sum_{i=0}^{n-1} F_i^{[2]} F_{n-1-i}^{[2]} \quad for \quad n \geq 1 \tag{2.5}$$

We next proceed to solve the case when $k = 3$. According to Eq. (2.1), we have

$$E_n^{[3]} = E_{n-1}^{[3]} + E_{n-2}^{[3]} + E_{n-3}^{[3]} + F_{n-2}^{[3]} + 2F_{n-3}^{[3]}$$

Let $H^{[3]}(z)$ denote the generating function for $E_n^{[3]}$'s, and define $E_i^{[3]} = 0$ for $i < 0$. Follow a similar procedure, we have

$$H^{[3]}(z) = \frac{1}{1-z-z^2-z^3}(z^2G^{[3]}(z) + 2z^3G^{[3]}(z))$$

where

$$G^{[3]}(z) = \frac{x^2}{1-z-z^2-z^3}$$

Hence,

$$H^{[3]}(z) = [G^{[3]}(z)]^2 + 2z[G^{[3]}(z)]^2$$

It is not surprising that the relation between $H^{[3]}(z)$ and $G^{[3]}(z)$ is similar to what we observed in the second order case. Therefore, we have

$$E_n^{[3]} = \sum_{i=0}^{n} F_i^{[3]}F_{n-i}^{[3]} + 2\sum_{i=0}^{n-1} F_i^{[3]}F_{n-1-i}^{[3]} \quad \text{for } n \geq 1$$

For the general case, let $H^{[k]}(z)$ denote the generating function for $E_n^{[k]}$'s, and define $E_i^{[k]} = 0$ for $i < 0$. Follow a similar procedure, we have the expression for $H^{[k]}(z)$:

$$H^{[k]}(z) = \frac{1}{1-z-z^2-\cdots-z^k} \cdot X$$

where

$$X = (z^2G^{[k]}(z) + 2z^3G^{[k]}(z) + \cdots + (k-2)z^{k-1}G^{[k]}(z) + (k-1)z^kG^{[k]}(z))$$

By substituting Eq. (2.2), we have

$$H^{[k]}(z) = \frac{1}{z^{k-3}}[G^{[k]}(z)]^2 + \frac{2}{z^{k-4}}[G^{[k]}(z)]^2 + \cdots + (k-2)[G^{[k]}(z)]^2 + (k-1)z[G^{[k]}(z)]^2$$

where $G^{[k]}(z)$ is the generating function for the k^{th} order Fibonacci numbers as expressed in Eq. (2.2). Therefore, it can be shown that

$$E_n^{[k]} = \sum_{j=1}^{k-1}(k-j)\sum_{i=0}^{n-2+j} F_i^{[k]}F_{n-2+j-i}^{[k]} \text{ for } n > k \tag{2.6}$$

Let X_n^k denote the number of edges in B_{n-k} for $n \geq k \geq 2$. For sufficiently large n, it can be shown that

$$0.79 \leq \frac{E_n^{[k]}}{X_n^k} \leq 1 \tag{2.7}$$

where the lower bound is obtained [62] when $k = 2$, and the upper bound is achieved when Γ_n^k becomes a hypercube.

2.4.2 Node Degrees

We now study the node degrees in the generalized Fibonacci cube. In particular, we will determine the maximum and minimum node degrees. Recall that nodes in Γ_n^k are addressed by $(n-k)$-bit k-FCs. Clearly, node 0 has a degree of $n-k$, since every node adjacent to node 0 has exactly one bit with value 1 in it k-FC. Therefore, the maximum node degree of Γ_n^k is $n-k$. In the remainder of this section we will determine the minimum node degree.

Two observations are immediate:

1. For $k \geq 2$ and $k \leq n < 2k$, by Lemma 2.6, Γ_n^k is isomorphic to B_{n-k}. Hence all nodes have the same degree of $n-k$.

2. For $k \geq 2$ and $n = 2k$, Γ_n^k is obtained by removing the node with 1^k as its address. Therefore the minimum degree of nodes is $k-1$.

The next lemma characterizes the general case.

Lemma 2.7 *For $k \geq 2$ and $n \geq 2k+1$, let p denote the node in Γ_n^k, whose k-FC is formed by repeating the sequence $01^{k-1}0$. Then p has the minimum degree in Γ_n^k.*

Proof: We distinguish among the following three cases:

Case 1: $n = 2k+1 \quad$ (*i.e.*, $n - k = k + 1$)

Clearly, node p has a degree of $k-1$. By observation 2 in the preceding paragraph, the minimum degree of nodes in Γ_{n-1}^k is also $k-1$. Furthermore, it can be shown inductively that the minimum node degree of Γ_n^k is a non-decreasing function of n for a fixed k. Therefore node p has the minimum degree in Γ_n^k.

Case 2: $2k+1 < n < 3k+2 \quad$ (*i.e.*, $k+1 < n-k < 2(k+1)$)

Let $r = n - 2k - 1$. We divide the $(n-k)$-bit generalized Fibonacci code into two parts: Part L which includes the leftmost $k+1$ bits, and Part R which includes the rightmost r bits. We claim that the minimum node degree of Γ_n^k is $k+r-2$ if $r = k$, or $k+r-1$ if otherwise (*i.e.*, $r < k$). Then, it is not difficult to verify that p is a node which has the minimum node degree as we just specified. To see the previous claim, we further consider the following two possibilities:

(a) Part L is of the form $(01^{k-1}0)$. The minimum node degree determined by Part L is $k-1$, which has been shown in Case 1. The minimum node degree determined by Part R is $r-1$ if $r = k$ and Part R contains exactly $k-1$ 1's, or r if otherwise (*i.e.*, $r < k$). Thus the minimum node degree is exactly as we claimed.

(b) Part L is not of the form $(01^{k-1}0)$. There is at most 1 bit in both Part L and R which cannot be flipped from 0 to 1; hence the minimum node degree is $k+r-1$.

Comparing the above two, we conclude that the minimum degree is captured by (a), which is what we claimed.

Case 3: $n \geq 3k+2 \quad$ (*i.e.*, $n-k \geq 2(k+1)$)

This can be proved similarly as we did in Case 2. □

Example 2.2 Consider $k = 4$. Nodes (011001), (0110011) and (01100110) are respectively a node with the minimum node degree in Γ^4_{10} , Γ^4_{11} and Γ^4_{12} , respectively. The minimum node degree is 4 in the above three cases.

Lemma 2.7 leads to the following result:

Lemma 2.8 *Let δ^k_n denote the minimum node degree of Γ^k_n. For $n \geq k+1$, we have the following recurrence equation:*

$$\delta^k_n = \delta^k_{n-k-1} + k - 1 \qquad (2.8)$$

where $\delta^k_n = n - k$ for $k \leq n \leq 2k-1$ and $\delta^k_{2k} = k - 1$.

One way to solve Eq. (2.8) is to apply the generating function as we did in Section 2.4.1 to count the number of edges. For simplicity, we make use of some observations obtained in the proof of Lemma 2.7 to solve Eq. (2.8). Recall that the node with its k-FC formed by repeating the sequence $01^{k-1}0$ has the minimum degree. Let $m = \lfloor (n-k)/(k+1) \rfloor$. By completing Case 3 in the proof of Lemma 2.7, it can be shown that

$$\delta^k_n = \begin{cases} n - k - 2m - 1 & \text{if } n = m(k+1) + 2k \\ n - k - 2m & \text{otherwise} \end{cases} \qquad (2.9)$$

where

$$m = \lfloor (n-k)/(k+1) \rfloor .$$

2.5 Summary of Results

We have reviewed the Incomplete Hypercube and defined the generalized Fibonacci cubes. Table 2.3 compares some parameters between the hypercube and the generalized Fibonacci cube.

T

F

n =

the I

as a

appl

1.

2.

Table 2.3. A comparison of the hypercube and generalized Fibonacci cube.

	B_{n-k}	Γ_n^k
Number of nodes	2^{n-k}	$F_n^{[k]}$
Number of edges	$(n-k)2^{n-k-1}$	Eq. (2.6)
Max. Degree	$n-k$	$n-k$
Min. Degree	$n-k$	Eq. (2.9)
Diameter	$n-k$	$n-k$†

† See Chapter 4.

For each element of the family, we have shown that I_N is a subgraph of B_n where $n = \lceil \log N \rceil$, and Γ_n^k is also a subgraph of B_{n-k} for $n \geq k \geq 2$. Additionally, both the Incomplete Hypercube and the generalized Fibonacci cube include the hypercube as a special case. The family of network topologies has the following two areas of application:

1. to allow an incremental expansion of a hypercube-like parallel computer systems, and

2. to serve as a fault-tolerant structure for the hypercube systems.

CHAPTER 3

GRAPH EMBEDDING IN GENERALIZED FIBONACCI CUBE

As explained in Chapter 1, graph embedding is an important issue in parallel computer systems. In this chapter, we will study embeddings of linear array, ring, mesh, and hypercube in the generalized Fibonacci cube.

Embeddings in the hypercube have been studied extensively in the past [19] [20] [21] [26] [39] [57] [59] [60] [101] [112] [113] . In many of these studies the Gray code [50] has been an important tool. The Gray codes are binary codes representing a sequence of numbers such that the codes representing two successive numbers are different in exactly one bit. From the property of the Gray code, it can be seen that a linear array can be embedded in the hypercube [101]. Furthermore, by employing the binary reflected Gray codes Saad and Schultz [101] have shown that cycles of even lengths can be embedded in the hypercube.

By choosing m_1-bit Gray codes for the x-direction and m_2-bit Gray codes for the y-direction, it is also shown in [101] that an (m_1+m_2)-cube embeds a 2D mesh of size $2^{m_1} \times 2^{m_2}$. In general, a $(2^{m_1} \times 2^{m_2} \times \cdots \times 2^{m_d})$ d-dimensional mesh can be embedded in B_n, provided that $n = m_1 + m_2 + \cdots + m_d$ [101]. Clearly, when the side lengths of a mesh are not powers of two, the Gray code scheme may not result in a minimal-expansion embedding. Chan and Chin [20] have employed the Gray code combined with an *ad hoc* method to achieve minimal-expansion, dilation-2 embeddings of most 2D meshes. By using optimal embeddings of small meshes with certain sizes and a scheme called "*graph decomposition,*" Ho and Johnsson [59, 60] have shown that 87% of all 2D meshes and a large portion of 3D meshes can be embedded in the hypercube with minimum expansion and dilation two. Finally, Chan [19] has settled the question regarding minimal-expansion, dilation-two embeddings of *all* 2D meshes in the hypercube.

The Hamiltonian problem in the generalized Fibonacci cube has been studied in [83] where we have constructively proved the following two optimal results:

1. Γ_n^k contains a Hamiltonian path, and
2. for $F_n^{[k]} \geq 4$, Γ_n^k contains a Hamiltonian cycle if and only if $F_n^{[k]}$ is even, whereas the longest cycle in Γ_n^k contains all but *one* node if and only if $F_n^{[k]}$ is odd.

These results imply that Γ_n^k embeds a linear array of size $F_n^{[k]}$ with expansion-1 and dilation-1, and Γ_n^k embeds a ring of size $F_n^{[k]}$ with expansion-1 and dilation no greater than two. In this chapter, we extend our work to the embeddings of meshes, which stem from the embedding of linear arrays and rings. Furthermore, we will also report the embedding of the hypercube in the generalized Fibonacci cube.

The remainder of this chapter is organized as follows. In Section 3.1, we prove that each generalized Fibonacci cube contains a Hamiltonian path. By embedding linear arrays in the subcubes and properly arranging them, we show in Section 3.2 that any cycle of length l can be embedded in Γ_n^k, provided that $4 \leq l \leq F_n^{[k]}$ and l is even. With the Hamiltonian path, in Section 3.3 we study the embeddings of 2D and 3D meshes. Section 3.4 describes the embedding of the hypercube in the generalized

Fibonacci cubes. Finally, we summarize our graph embedding results in Section 3.5.

3.1 Hamiltonian Path

In this section, we will show that there is a Hamiltonian path in every Γ_n^k, which serves as the basis for our study of cycles in the generalized Fibonacci cubes. To motivate, we first study the Hamiltonian paths in the second-order generalized Fibonacci cube.

Example 3.1 The sequence $P_0 = (01, 00, 10)$ and $P_1 = (0, 1)$ are, respectively, Hamiltonian paths in Γ_4^2 and Γ_3^2 (refer to Fig. 2.2). By Lemma 2.4, the graph Γ_5^2 contains a copy of Γ_4^2 and a copy of Γ_3^2. To construct a Hamiltonian path in Γ_5^2, we attach a prefix '0' to each code in P_0 and "10" to each in P_1, respectively, then reverse both sequences. The two resulted sequences are (010,000,001) and (101,100). Since $H(001, 101) = 1$, these two sequences can be linked to form a Hamiltonian path in Γ_5^2. Repeating the same procedure, we can obtain a Hamiltonian path in Γ_6^2 by concatenating (0100,0101,0001,0000,0010) and (1010,1000,1001).

By using induction, it can be shown that the recursive procedure described in Example 3.1 always produces a Hamiltonian path in the second order generalized Fibonacci cube. The following theorem extends this observation to the generalized Fibonacci cube.

Theorem 3.1 *Γ_n^k contains a Hamiltonian path for $n \geq k \geq 2$.*

Proof: Here we will prove the case of $k = 3$ by induction (on n). Cases when $k > 3$ can be proved similarly. For convenience, nodes are referred to by their k-FCs. A path will be represented by the sequence of the nodes in it.

It is easily seen that $P_0 = (\lambda), P_1 = (0, 1)$ and $P_2 = (01, 00, 10, 11)$ are Hamiltonian paths in Γ_3^3, Γ_4^3, and Γ_5^3, respectively. We establish our induction basis for $n = 6$. To obtain a Hamiltonian path in Γ_6^3, we take the following procedure:

Step 1 Prefix a '0' (respectively, "10" and "110") to each code in P_0 (respectively, P_1 and P_2). Let the resulted sequences be denoted by $0P_0, 10P_1$, and $110P_2$, respectively.

Step 2 Reverse each code in $0P_0, 10P_1$, and $110P_2$, respectively. Let the resulted sequences be denoted by $0\overline{P_0}$, $10\overline{P_1}$ and $110\overline{P_2}$, respectively.

Step 3 Concatenate the three sequences in the following order: $0\overline{P_0}$, $10\overline{P_1}$ and $110\overline{P_2}$.

Therefore, we obtain the sequence $(011, 010, 000, 001, 101, 100, 110)$ that is a Hamiltonian path in Γ^3_6. We assume that the above procedure always results in a Hamiltonian path in Γ^3_n for all $n \leq I$, where $I \geq 6$ denote an integer.

Now consider the case when $n = I + 1$, *i.e.*, Γ^3_{I+1}. By Lemma 2.4, $\Gamma^{[3]}_{I+1}$ can be decomposed into three subgraphs, $G^3_I(0)$, $G^3_{I-1}(1)$ and $G^3_{I-2}(2)$, which are isomorphic to $\Gamma^{[3]}_I$, $\Gamma^{[3]}_{I-1}$, and $\Gamma^{[3]}_{I-2}$ respectively. By induction hypothesis, there is a Hamiltonian path in $\Gamma^{[3]}_I$, $\Gamma^{[3]}_{I-1}$ and $\Gamma^{[3]}_{I-2}$, respectively. Let P_j (where $0 \leq j \leq 2$) denote the Hamiltonian path in $\Gamma^{[3]}_{I-j}$ obtained by the previous procedure. Let $\overline{P_j}$ denote the reversed sequence of P_j. It remains to show that the three sequences (*i.e.*, $0\overline{P_0}$, $10\overline{P_1}$ and $110\overline{P_2}$) obtained in Step 2 of the previous procedure can be concatenated.

Let the first (respectively, last) node of a path be referred to as the *head* (respectively, *tail*), and let h_i (respectively, t_i) denote the head (respectively., tail) of the Hamiltonian path obtained by the previous procedure in Γ^3_i. (For example, $h_3 = t_3 = \lambda$, $h_4 = 0$, and $t_4 = 1$.) It is can be shown that the following recurrence relation is a consequence of the procedure described above: $h_n = 0 \cdot t_{n-1}$.

Given this, observe that the tail of $0\overline{P_0}$ (which was the head of $0P_0$) is now labeled by $0 \cdot h_I = 0 \cdot 0t_{I-1}$. On the other hand, the head of $10\overline{P_1}$ (which was the tail of $10P_1$) is labeled by $10 \cdot t_{I-1}$. Since $H(00t_{I-1}, 10t_{I-1}) = 1$, there is an edge between the two nodes, *i.e.*, the two (Hamiltonian) paths can be linked together. Similarly, the tail of $10\overline{P_1}$ now has the label $10 \cdot h_{I-1} = 100 \cdot t_{I-2}$, while the head of $110\overline{P_2}$ now has the label $110 \cdot t_{l-2}$. Hence there is also an edge between the two nodes. Therefore, we conclude that the previous procedure results in a Hamiltonian path in $\Gamma^{[3]}_{l+1}$.

It should be clear that the above argument can be extended to the general case when $k > 3$. □

3.2 Cycles

In this section we will study the embedding of cycles in the generalized Fibonacci cubes. The embedding of cycles is based on the embedding of Hamiltonian paths presented in the previous section and the recursive properties of the generalized Fibonacci cubes. Let L^k_n be the Hamiltonian path constructed according to the algorithm described in the previous section. To motivate, let us consider embeddings of cycles in Γ^2_8 in the following example. Recall that L^2_8 in Γ^2_8 is obtained by a concatenation of two sequences in $0\mathcal{V}^2_7$ and $10\mathcal{V}^2_6$. By Lemma 2.4, Γ^2_8 consists of two copies of Γ^2_6 (*i.e.*, $00\mathcal{V}^2_6$ and $10\mathcal{V}^2_6$) and one copy of Γ^2_5 (*i.e.*, $010\mathcal{V}^2_5$). By properly arranging three Hamiltonian paths in these three subcubes, we can obtain embeddings of cycles in Γ^2_8.

Example 3.2 In Example 3.1 we have shown that L^2_6 = (0100, 0101, 0001, 0000, 0010, 1010, 1000, 1001). Now, let $L_{0,0}$ = (000100, 000101, 000001, 000000, 000010, 001010, 001000, 001001), $L_{0,1}$ = (010100, 010101, 010001, 010000, 010010), and L_1 = (100100, 100101, 100001, 100000, 100010, 101010, 101000, 101001). It can be seen that $L_{0,0}$, $L_{0,1}$ and L_1 are Hamiltonian paths in $00\mathcal{V}^2_6$, $010\mathcal{V}^2_5$ and $10\mathcal{V}^2_6$, respectively. Figure 3.1 shows that Γ^2_8 contains a cycle of length 20. Notice that the nodes are addressed by 6-bit 2-FCs, where the least significant 4 bits are shown next to the nodes and the most significant two bits are shown on the far left-hand side. Note that the first l nodes in both $L_{0,0}$ and L_1 can be used to construct a cycle of length

$2l$, where $2 \leq l \leq 8$. Furthermore, nodes in $L_{0,1}$ are needed to construct a cycle of length 18 or 20.

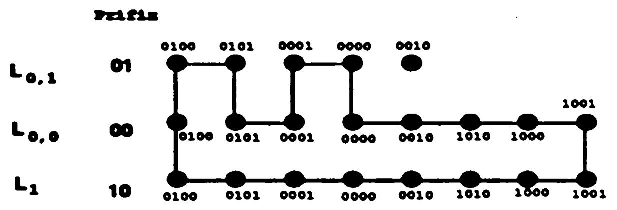

Figure 3.1. Γ_8^2 contains a cycle of length 20.

We now proceed to study the embedding of cycles in the generalized Fibonacci cubes. Lemma 3.1 will facilitate our presentation.

Lemma 3.1 *Assume that $k \geq 2$, and $n \geq 2k+1$. We have*

$$\begin{aligned}
\mathcal{V}_n^k &= 0\mathcal{V}_{n-1}^k \cup 1(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k) \\
&= 0(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k) \cup 1(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k) \cup 01^{k-1}0\mathcal{V}_{n-k-1}^k
\end{aligned}$$

Proof: By Lemma 2.3 we have

$$\begin{aligned}
\mathcal{V}_n^k &= 0\mathcal{V}_{n-1}^k \cup 10\mathcal{V}_{n-2}^k \cup 110\mathcal{V}_{n-3}^k \cup \cdots \cup 1^{k-2}0\mathcal{V}_{n-k+1}^k \cup 1^{k-1}0\mathcal{V}_{n-k}^k \\
&= 0\mathcal{V}_{n-1}^k \cup 1(0\mathcal{V}_{n-2}^k \cup 10\mathcal{V}_{n-3}^k \cup \cdots \cup 1^{k-3}0\mathcal{V}_{n-k+1}^k \cup 1^{k-2}0\mathcal{V}_{n-k}^k) \\
&= 0\mathcal{V}_{n-1}^k \cup 1(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k) \\
&= 0(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k) \cup 01^{k-1}0\mathcal{V}_{n-k-1}^k \cup 1(\mathcal{V}_{n-1}^k \setminus 1^{k-1}0\mathcal{V}_{n-k-1}^k)
\end{aligned}$$

□

I
c
3
o
L
st
pa
th
sh
em
Ca
The
add
one
node
Case

Theorem 3.2 *Assume that $k \geq 2$. For all $n \geq k+2$, any cycle of length l can be embedded in Γ_n^k provided that l is even and $4 \leq l \leq F_n^{[k]}$.*

Proof: An example for $k = 2$ has been examined in Example 3.2. We will first show the case when $k = 3$. By Lemma 3.1, we have

$$\mathcal{V}_n^3 = 0\mathcal{V}_{n-1}^3 \cup 1(\mathcal{V}_{n-1}^3 \setminus 110\mathcal{V}_{n-4}^3)$$

Define S_0 to be the subgraph induced by nodes in $0\mathcal{V}_{n-1}^3$, and S_1 the subgraph induced by nodes in $1(\mathcal{V}_{n-1}^3 \setminus 110\mathcal{V}_{n-4}^3)$, *i.e.*, nodes in $10\mathcal{V}_{n-2}^3 \cup 110\mathcal{V}_{n-3}^3$. Note that S_0 is isomorphic to Γ_{n-1}^3 and S_1 is isomorphic to a graph obtained by removing a subcube Γ_{n-4}^3 from Γ_{n-1}^3 (refer to Figure 3.2 (a)). By Theorem 3.1, there exists a Hamiltonian path in S_0; furthermore, from the construction of the Hamiltonian path in Γ_{n-1}^3, we can observe that there exists a Hamiltonian path in S_1. Let L_0 be the sequence of 3-FCs formed by attaching a prefix '0' to each 3-FC of L_{n-1}^3, and L_1 be the sequence of 3-FCs formed by attaching a prefix '1' to each of the first $(F_{n-1}^{[3]} - F_{n-4}^{[3]})$ 3-FCs of L_{n-1}^3. Note that attaching the prefix '1' to each element of the set $(\mathcal{V}_{n-1}^3 \setminus 110\mathcal{V}_{n-4}^3)$ still results in a valid 3-FC. One should be able to see that L_0 and L_1 are Hamiltonian paths in S_0 and S_1, respectively. Moreover, for all i (where $1 \leq i \leq F_{n-1}^{[3]} - F_{n-4}^{[3]}$), there is an edge connecting the i^{th} nodes of L_0 and L_1 (refer to Fig. 3.2 (b)). We next show that, by properly selecting nodes in the L_0 and L_1, cycles of even length can be embedded in Γ_n^3. We distinguish between the following two cases:

Case 1: $4 \leq l \leq 2(F_{n-1}^{[3]} - F_{n-4}^{[3]})$.

The cycle is constructed by the first $l/2$ nodes in both L_0 and L_1 together with two additional links (represented by heavy segments in Fig. 3.2(b)). The two links are: one edge connecting the heads of L_0 and L_1; the other edge connecting the two $(l/2)^{th}$ nodes in both Hamiltonian paths involved.

Case 2: $2(F_{n-1}^{[3]} - F_{n-4}^{[3]}) < l \leq F_n^{[3]}$.

Define $l' = l - 2(F^{[3]}_{n-1} - F^{[3]}_{n-4})$. We further break L_0 into two parts: $L_{0,0}$ which is a Hamiltonian path in $00\mathcal{V}^3_{n-2} \cup 010\mathcal{V}^3_{n-3}$, and $L_{0,1}$ which is a Hamiltonian path in $0110\mathcal{V}^3_{n-4}$ (refer to Fig. 3.2(c)). Notice that $00\mathcal{V}^3_{n-2}$ contains $0010\mathcal{V}^3_{n-4}$ (see Fig. 3.2 (a)), and each node in $0010\mathcal{V}^3_{n-4}$ has an edge connecting to one node in $0110\mathcal{V}^3_{n-4}$. The cycle of length l consists of *all* nodes in $L_{0,0}$ and L_1 together with the first l' nodes in $L_{0,1}$. Since the inclusion of two adjacent nodes in $L_{0,1}$ requires a pair of "up" and "down" edges (represented by heavy segments in Fig. 3.2 (c)), only cycles of even length can be embedded in Γ^3_n.

We next discuss the case for arbitrary k where $k > 3$. Following the same approach as in $k = 3$, let us define $L_{0,0}$, L_1 and $L_{0,1}$ to be the Hamiltonian paths in $0(\mathcal{V}^k_{n-1} \setminus 1^{k-1}0\mathcal{V}^k_{n-k-1})$, $0(\mathcal{V}^k_{n-1} \setminus 1^{k-1}0\mathcal{V}^k_{n-k-1})$, and $01^{k-1}0\mathcal{V}^k_{n-k-1}$, respectively. It is not difficult to see that any cycle of length $2s$ can be obtained by selecting the first s nodes from each of $L_{0,0}$ and L_1, where $2 \le s \le F^{[k]}_{n-1} - F^{[k]}_{n-k-1}$. To have a cycle of length $2s$, where $F^{[k]}_{n-1} - F^{[k]}_{n-k-1} < s \le F^{[k]}_n/2$, we just include the first $2s - 2(F^{[k]}_{n-1} - F^{[k]}_{n-k-1})$ nodes in the $L_{0,1}$. $\square$

It can be easily seen from the previous proof that Γ^k_n contains a Hamiltonian cycle if the length of $L_{0,1}$ is even. Notice that $L_{0,1}$ is a Hamiltonian path in a graph isomorphic to Γ^k_{n-k-1}, which contains exactly $F^{[k]}_{n-k-1}$ nodes. We next examine some properties of the generalized Fibonacci numbers and the generalized Fibonacci cubes, and show that Γ^k_n contains a Hamiltonian cycle if and only if $F^{[k]}_n \ge 4$ and is even.

Lemma 3.2

(1) For $k = 2$ and $i \ge 2$,

$F^{[2]}_{3i}$ *is even, and* $F^{[2]}_{3i-1}$, $F^{[2]}_{3i-2}$ *are odd.*

(2) For $k \ge 3$ and $i \ge 2$,

$F^{[k]}_{(k+1)i}$, $F^{[k]}_{(k+1)i-3}$, $F^{[k]}_{(k+1)i-4}$, $\cdots$, $F^{[k]}_{(k+1)i-k}$ *are even, and*

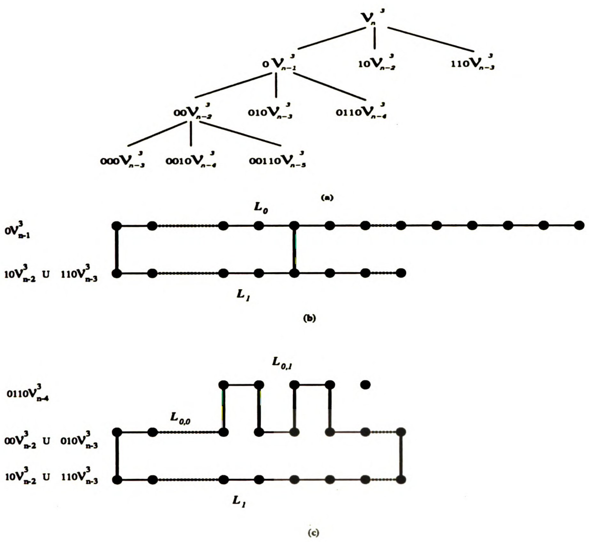

Figure 3.2. Embedding of cycles in Γ_n^3 : (a) a decomposition of Γ_n^3, (b) Case 1, and (c) Case 2.

$F^{[k]}_{(k+1)i-1}$, $F^{[k]}_{(k+1)i-2}$ *are odd.*

Proof: By induction on i. □

Note that $F^{[k]}_{n-k-1}$ is even if and only if $F^{[k]}_n$ is even, where $n \geq 2(k+1)$. Hence by Theorem 3.2, Γ_n^k is Hamiltonian if $F^{[k]}_n \geq 4$ and it is even. Furthermore, by Lemma 3.2, $k-1$ out of the $k+1$ consecutive k^{th} order Fibonacci numbers are even.

We next study the Hamiltonian problem for the generalized Fibonacci cubes whose total numbers of nodes are odd. It is known (see *e.g.*, [44]) that the binary hypercubes are bipartite. Moreover, a bipartite graph cannot have a cycle of odd length. Hence we have the following property (see *e.g.*, [101]):

Lemma 3.3 *There is no cycle of odd length in the binary hypercubes.*

Lemma 2.6 shows that Γ_n^k is a subgraph of B_{n-k}. Therefore Γ_n^k is also bipartite. Consequently, we have

Lemma 3.4 *There is no cycle of odd length in the generalized Fibonacci cubes.*

Theorem 3.2, Lemmas 3.2 and 3.4 lead to the following result:

Theorem 3.3

(1) For $k = 2$, Γ_n^2 is Hamiltonian if and only if $n = 3i$ for some integer $i \geq 2$; otherwise, the length of the longest cycle is $F_n^{[2]} - 1$.

(2) For $k \geq 3$, Γ_n^k is Hamiltonian if n can be expressed as one of the following forms: $(k+1)i$, $(k+1)i-3$, $(k+1)i-4$, $\cdots$,$(k+1)i-k$ for some integer $i \geq 2$; otherwise, the length of the longest cycle is $F_n^{[k]} - 1$.

Corollary 3.1 *For $k \geq 2$ and $n - k \geq 3$, a ring of size $F_n^{[k]}$ can be embedded in Γ_n^k with dilation one if $F_n^{[k]}$ is even, and dilation two if otherwise.*

Proof: The case for $F_n^{[k]}$ being even is proved in Theorem 3.3. If $F_n^{[k]}$ is odd, the only node that is not included in the longest cycle has at least one adjacent node which is in the cycle. Hence the embedding of ring can be achieved by paying a higher price (*i.e.*, dilation is 2). □

3.3 Mesh

In this section, we will study the embedding of mesh in the generalized Fibonacci cube. We will begin our discussion with the embedding of 2D mesh, then extend our result to mesh of higher dimensions. For convenience, all the Hamiltonian paths referred in this section are obtained by the procedure mentioned in the proof of Theorem 3.1.

3.3.1 2D Mesh

To motivate, we examine an example of embedding in Γ_9^2.

Example 3.3 In Example 3.1, we saw that (01,00,10) and (010,000,001,101,100) are Hamiltonian paths in Γ_4^2 and Γ_5^2, respectively. Figure 3.3 shows that an $F_5^{[2]} \times F_5^{[2]}$ and an $F_4^{[2]} \times F_4^{[2]}$ mesh can be embedded in Γ_9^2, where a 2-FC is divided into a left (leftmost 4-bit) and a right (rightmost 3-bit) parts. Notice that the sequence of codes over the y-direction in the $F_5^{[2]} \times F_5^{[2]}$ mesh corresponds exactly to the Hamiltonian path in Γ_5^2, whereas the sequence of codes over the x-direction is obtained by suffixing a '0' to each code of the same Hamiltonian path. On the other hand, the sequences of codes over the x- and y-direction in the $F_4^{[2]} \times F_4^{[2]}$ mesh are obtained by suffixing "01" and prefixing '0' to the Hamiltonian path in Γ_4^2.

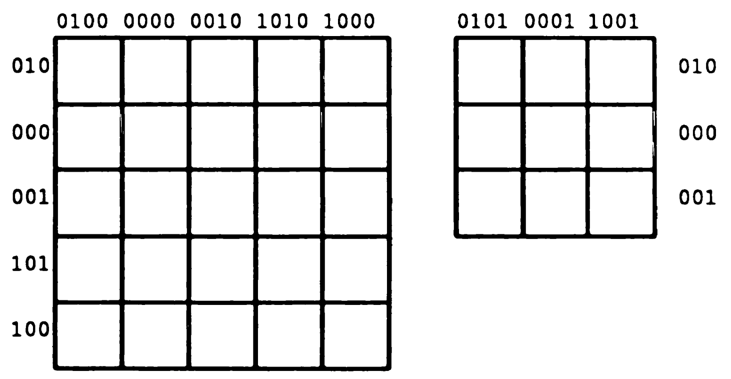

Figure 3.3. Embedding of an $F_5^{[2]} \times F_5^{[2]}$ and an $F_4^{[2]} \times F_4^{[2]}$ meshes in Γ_9^2.

Figure 3.4 explains the idea behind the embedding shown in Example 3.3. It can be seen that there are exactly two possible combinations when dividing an $(m+n)$-bit 2-FC into an m- and n-bit 2-FCs. Consider a node whose bit n is a 0 (combination A).

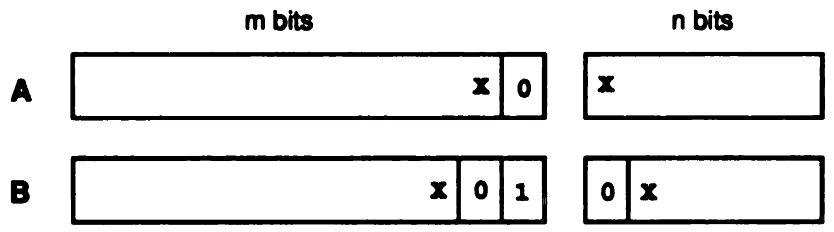

Figure 3.4. Dividing an $(m+n)$-bit 2-FC into an m- and n-bit 2-FCs.

Observe that all the n bits in the right part can vary, which results in all n-bit 2-FCs; on the other hand, all but the rightmost bit (which is fixed to 0) in the left part can vary, which results in all $(m-1)$-bit 2-FCs. Recall that any generalized Fibonacci cube contains a Hamiltonian path. We can select the Hamiltonian path in Γ^2_{n+2} as the sequence of codes for embedding over the x-direction, and the Hamiltonian path in Γ^2_{m+1} with a suffix '0' as the sequence of codes for y-direction. Therefore, the nodes captured in combination A will enable an embedding of a mesh with size $F^{[2]}_{m+1} \times F^{[2]}_{n+2}$. Similarly, the nodes captured in combination B will enable an embedding of a mesh with size $F^{[2]}_{m} \times F^{[2]}_{n+1}$. We next show that this scheme can be generalized to embed 2D meshes in the generalized Fibonacci cubes.

Theorem 3.4

(1) For $m \geq 2$ and $n \geq 1$, Γ^2_{m+n+2} embeds an $F^{[2]}_{m+1} \times F^{[2]}_{n+2}$ and an $F^{[2]}_{m} \times F^{[2]}_{n+1}$ meshes.

(2) For $m \geq 3$ and $n \geq 2$, Γ^3_{m+n+3} embeds an $F^{[3]}_{m+2} \times F^{[3]}_{n+3}$, an $F^{[3]}_{m+2} \times (F^{[3]}_{n+2} + F^{[3]}_{n+1})$ and an $F^{[3]}_{m} \times F^{[3]}_{n+2}$ meshes.

(3) For $k \geq 4$, $m \geq k$ and $n \geq k-1$, Γ^{k}_{m+n+k} embeds an $F^{[k]}_{m+k-1} \times F^{[k]}_{n+k}$, an $F^{[k]}_{m+k-2} \times (F^{[k]}_{n+k-1} + F^{[k]}_{n+k-2} + \cdots + F^{[k]}_{n+1})$, an $F^{[k]}_{m+k-3} \times (F^{[k]}_{n+k-1} + F^{[k]}_{n+k-2} + \cdots + F^{[k]}_{n+2})$, $\cdots$,and an $F^{[k]}_{m} \times F^{[k]}_{n+k-1}$ meshes.

Proof: The case when $k = 2$ has been examined in the preceding paragraph. We will prove the case when $k = 3$. Consider dividing an $(m+n)$-bit 3-FC into two parts, where the left part consists of the leftmost m bits and the right part the remaining n bits. Figure 3.5 shows the four possible combinations. It can be shown that the nodes captured in combination A (respectively, D) enable an embedding of a mesh with size $F^{[3]}_{m+2} \times F^{[3]}_{n+3}$ (respectively, $F^{[3]}_{m} \times F^{[3]}_{n+2}$). To see the embedding of the mesh with size $F^{[3]}_{m+2} \times (F^{[3]}_{n+2} + F^{[3]}_{n+1})$, observe that the nodes captured in combinations B and C have the same left part and their right parts can be obtained from $\mathcal{V}^{3}_{n+3}$ by removing all codes in the $110\mathcal{V}^{3}_{n}$. Hence by the procedure presented in the proof of Theorem 3.1, we are able to embed a linear array of size $F^{[3]}_{n+2} + F^{[3]}_{n+1}$ in them.

The case when $k \geq 4$ can be handled similarly. For example, Figure 3.6 shows the case when $k = 4$. It can be seen that there are seven possible combinations to break a 4-FC. Again, nodes captured in combination A enable an embedding of an $F^{[4]}_{m+3} \times F^{[4]}_{n+4}$ mesh, nodes captured in combination G an $F^{[4]}_{m} \times F^{[4]}_{n+3}$ mesh, nodes captured in combinations B, C and D enable an $F^{[4]}_{m+2} \times (F^{[4]}_{n+3} + F^{[4]}_{n+2} + F^{[4]}_{n+1})$ mesh. Finally, nodes captured in combinations E and F enable an $F^{[4]}_{m+1} \times (F^{[4]}_{n+3} + F^{[4]}_{n+2})$ mesh. □

The embedding of meshes in Γ^{3}_{10} is displayed in Figure 3.7. It is seen that there are three separate 2D meshes. This is similar to what we saw in Fig. 3.3 where two separate meshes are obtained when $k = 2$. One interesting question is whether these separate meshes are connected? Figure 3.8 shows that the two meshes embedded in Γ^{2}_{9} (refer to Fig. 3.3) are indeed connected and form one piece. Notice that the code sequence over the x-direction can be obtained by reversing each 2-FC of the

F

r

fo

a

Fi

2-

ca

Le

p_{ι}-

Th

Pro

one

unit

their

1. *R*

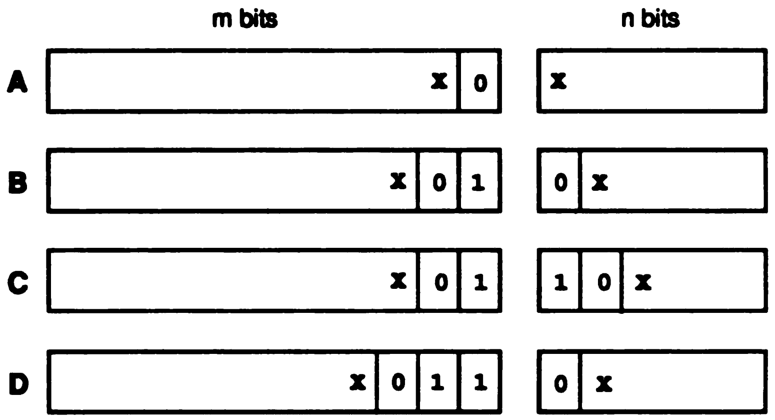

Figure 3.5. Dividing an $(m+n)$-bit 3-FC into an m- and n-bit 3-FCs.

Hamiltonian path in Γ_6^2 (refer to Example 3.1). We next demonstrate that by properly rearranging the code sequence over the x-direction, these meshes are connected and form one piece. It is necessary to define the *Reversed Fibonacci Code* (RFC). Given a k-FC $P = (p_i p_{i-1} \cdots p_1 p_0)$, the RFC of P (denoted by $RFC(P)$) is $(p_0 p_1 \cdots p_{i-1} p_i)$. Figure 3.9 shows that there are two different ways to label the nodes in Γ_6^2: one is 2-FC and the other is in *reversed* 2-FC. The following lemma shows that the RFC can be also used to label nodes in the generalized Fibonacci cube.

Lemma 3.5 *Let p $(0 \leq p < F_n^{[k]})$ be a node in Γ_n^k, and $FC(p) = p_{n-k-1}p_{n-k-2}\cdots p_1 p_0$. Let $q = p_0 F_{n-1}^{[k]} + p_1 F_{n-2}^{[k]} + \cdots + p_{n-k-2}F_{n-k+1}^{[k]} + p_{n-k-1}F_{n-k}^{[k]}$. Then q can be used to relabel the node p if all nodes in Γ_n^k are relabeled by their RFCs.*

Proof: The proof consists of two parts. We first show that $RFC(\cdot)$ is a *one-to-one* and *onto* function from $\mathcal{V}_n^k$ to $\mathcal{V}_n^k$. Then we demonstrate that two nodes with unit Hamming distance under their k-FCs will preserve unit Hamming distance under their RFCs. The first part of our proof consists of three steps:

1. $RFC(\cdot)$ is a one-to-one function. To see this, assume that $RFC(\cdot)$ is not a one-to-one function. Then there exist $0 \leq p \neq p' < F_n^{[k]}$ such that $RFC(p) = RFC(p')$.

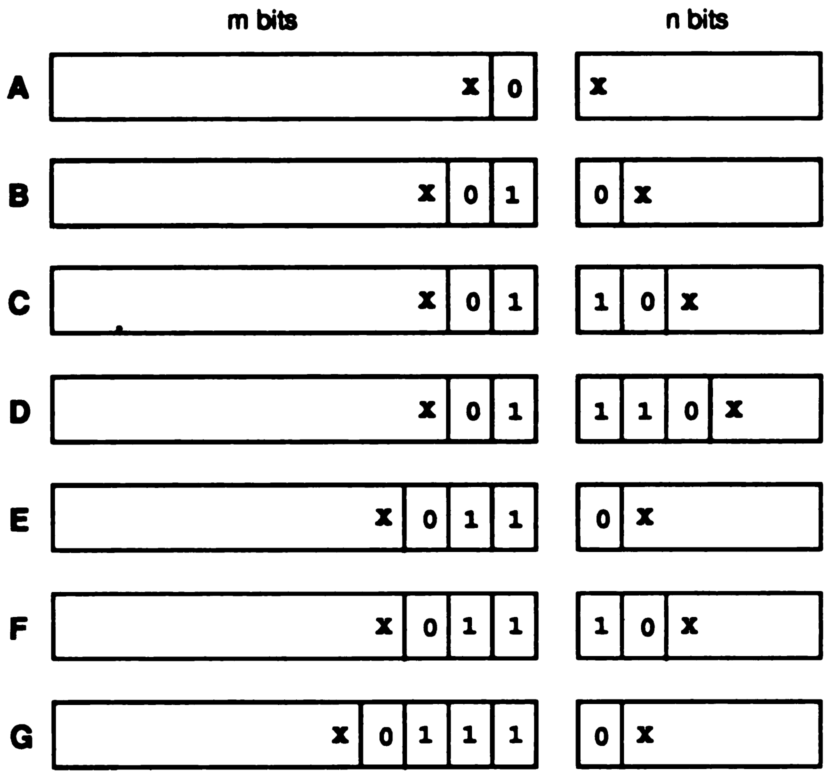

Figure 3.6. Dividing an $(m+n)$-bit 4-FC into an m- and n-bit 4-FCs.

This implies that $FC(p) = FC(p')$. Since the k-FCs for both p and p' are unique, we conclude that $p = p'$, which is a contradiction.

2. Given $\mathcal{V}_n^k$ as the domain of the function $RFC(\cdot)$, then the range will also be $\mathcal{V}_n^k$. To see this, assume that the binary code $Q = RFC(p)$ is *not* in $\mathcal{V}_n^k$, then Q must contain k consecutive 1's. This implies that $FC(p)$ contains k consecutive 1's, which is a contradiction.

3. $RFC(\cdot)$ is an onto function. This follows from 1 and 2.

Now, we prove the second part. Recall that two nodes p and p' are adjacent in Γ_n^k if and only if their k-FCs differ in exactly one bit. In other words, p and p' are adjacent if and only if $|p - p'| = F_i^{[k]}$ for some $k \leq i < n$. Given a pair of adjacent nodes p and p', let q and q' be their second labels, respectively. It can be shown that

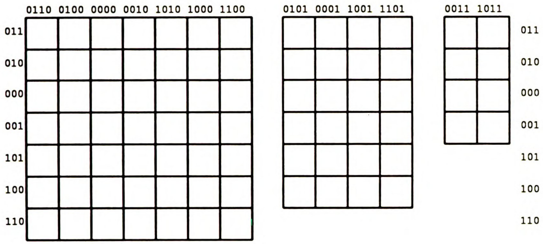

Figure 3.7. Embedding of meshes in Γ^3_{10}.

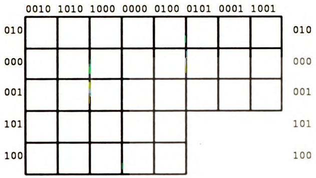

Figure 3.8. Embedding of meshes in Γ^2_9.

$|q - q'| = F^{[k]}_{n+k-1-i}$ if and only if $|p - p'| = F^{[k]}_i$ for some $k \leq i < n$. In other words, two nodes with unit Hamming distance will preserve unit Hamming distance in their second labels. □

From Fig. 3.5, it can be seen that the least significant bits(s) in the left parts are fixed to '0,' "01" and "011," respectively. Lemma 3.5 shows that nodes can be also labeled by their RFCs. The Hamiltonian path constructed by the recursive procedure presented in Section 3.1 will include nodes with leading bit(s) '0,' then nodes with

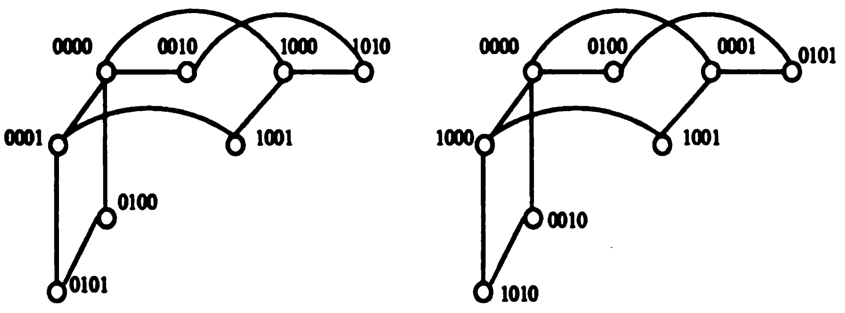

Figure 3.9. Two different labelings of Γ_6^2.

"10" and followed by nodes with "110." Figure 3.10 shows that by first obtaining the Hamiltonian path and relabeling each node with its RFC over the x-direction, we are able to connect the three separate meshes into one piece for the embedding in Γ_{10}^3. Clearly, this scheme can be applied to $k > 3$.

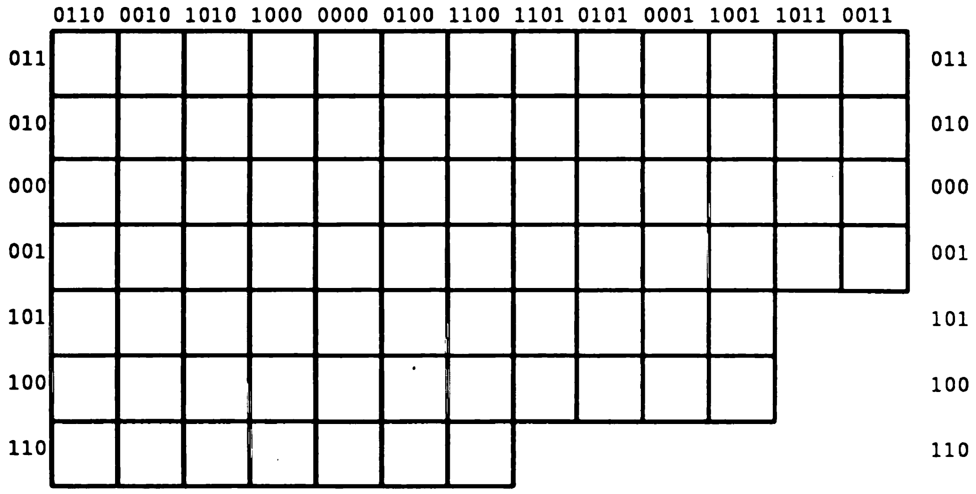

Figure 3.10. Embedding of meshes in Γ_{10}^3.

3.3.2 3D Mesh

The embedding of 3D meshes is a further extension of the results obtained in the previous section. Recall that the approach taken before is to divide the k-FC into two parts. To have an embedding of 3D mesh, we divide the k-FC into three parts: left, middle and right, which are used for embedding over the x, y and z directions. Figure 3.11 shows that there are four possible combinations when dividing an $(l + m + n)$-bit 2-FC into an l-, m- and n-bit 2-FCs. Notice that the difference between

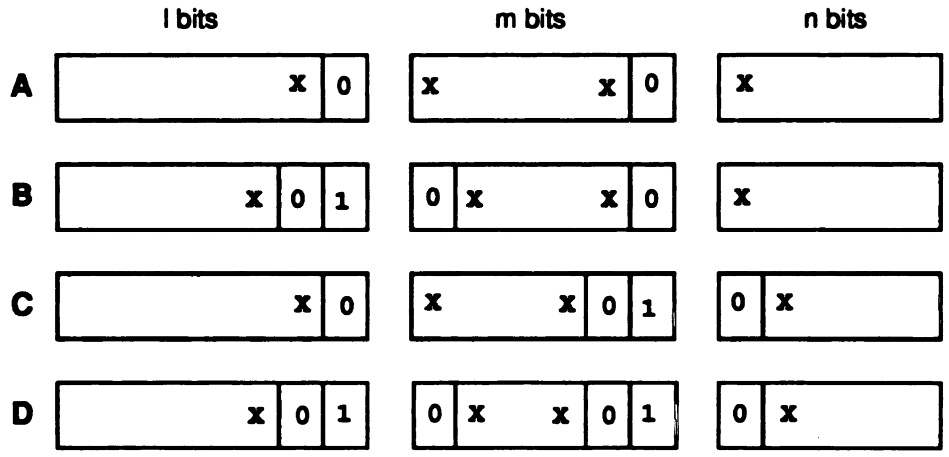

Figure 3.11. Dividing an $(l + m + n)$-bit 2-FC into an l-, m- and n-bit 2-FCs.

the left and middle part in combinations A and B is similar to what we saw in the 2D case (refer to Fig. 3.4), except that the rightmost bit in both middle parts are 0's. By analogy, it can be shown that nodes captured in combination A and B embed the following two 3D meshes of sizes: (A) $F_{l+1}^{[2]} \times F_{m+1}^{[2]} \times F_{n+2}^{[2]}$, and (B) $F_{l}^{[2]} \times F_{m}^{[2]} \times F_{n+2}^{[2]}$, respectively. Similarly, the nodes captured in combinations C and D embed the following two 3D meshes of sizes: (C) $F_{l+1}^{[2]} \times F_{m}^{[2]} \times F_{n+1}^{[2]}$, and (D) $F_{l}^{[2]} \times F_{m-1}^{[2]} \times F_{n+1}^{[2]}$, respectively.

The embeddings in $\Gamma_{l+m+n+3}^{3}$ is pictured in Figure 3.12. There are a total of 16 possible combinations. By analogy, the sizes of the meshes are:

A. $F^{[3]}_{l+2} \times F^{[3]}_{m+2} \times F^{[3]}_{n+3}$,

B. $F^{[3]}_{l+1} \times F^{[3]}_{m+1} \times F^{[3]}_{n+3}$,

C. $F^{[3]}_{l+1} \times F^{[3]}_{m} \times F^{[3]}_{n+3}$,

D. $F^{[3]}_{l} \times F^{[3]}_{m+1} \times F^{[3]}_{n+3}$,

E. $F^{[3]}_{l+2} \times F^{[3]}_{m+1} \times F^{[3]}_{n+2}$,

F. $F^{[3]}_{l+1} \times F^{[3]}_{m} \times F^{[3]}_{n+2}$,

G. $F^{[3]}_{l+1} \times F^{[3]}_{m-1} \times F^{[3]}_{n+2}$,

H. $F^{[3]}_{l} \times F^{[3]}_{m} \times F^{[3]}_{n+2}$,

I. $F^{[3]}_{l+2} \times F^{[3]}_{m+1} \times F^{[3]}_{n+1}$,

J. $F^{[3]}_{l+1} \times F^{[3]}_{m} \times F^{[3]}_{n+1}$,

K. $F^{[3]}_{l+1} \times F^{[3]}_{m-1} \times F^{[3]}_{n+1}$,

L. $F^{[3]}_{l} \times F^{[3]}_{m} \times F^{[3]}_{n+1}$,

M. $F^{[3]}_{l+2} \times F^{[3]}_{m} \times F^{[3]}_{n+2}$,

N. $F^{[3]}_{l+1} \times F^{[3]}_{m-1} \times F^{[3]}_{n+2}$,

O. $F^{[3]}_{l+1} \times F^{[3]}_{m-2} \times F^{[3]}_{n+2}$, and

P. $F^{[3]}_{l} \times F^{[3]}_{m-1} \times F^{[3]}_{n+2}$, respectively.

3.4 Hypercube

We saw that each generalized Fibonacci cube is a subgraph of a hypercube. In Theorem 3.5, we will show that a generalized Fibonacci cube also embeds a hypercube.

Theorem 3.5 *For $n \geq k \geq 2$, Γ^k_n embeds B_d where $d = n - k - m$ and $m = \lfloor (n-k)/k \rfloor$.*

Proof: Recall that nodes in Γ_n^k are addressed by $(n-k)$-bit k-FCs. Table 3.1 shows some embeddings of hypercubes in Γ_n^k for some $2 \leq k \leq 5$ and $2 \leq n \leq 10$. Notice that a '$-$' means that such Γ_n^k is undefined; a 'λ' stands for the graph containing a single node, which trivially embeds B_0; a 'x' means "don't care." Finally, the total number of 'x' in an entry determines the dimension of the hypercube that can be embedded in the Γ_n^k. Since no k consecutive 1's is allowed in the k-FCs, it can be shown that every $k-1$ x's should be followed by a 0. Hence we have $d = n - k - m$.

□

Table 3.1. Embedding of Hypercubes.

$n \setminus k$	2	3	4	5
2	λ	λ	$-$	$-$
3	x	λ	λ	$-$
4	x0	x	λ	λ
5	x0x	xx	x	λ
6	x0x0	xx0	xx	x
7	x0x0x	xx0x	xxx	xx
8	x0x0x0	xx0xx	xxx0	xxx
9	x0x0x0x	xx0xx0	xxx0x	xxxx
10	x0x0x0x0	xx0xx0x	xxx0xx	xxxx0

3.5 Summary of Results

We have considered the Hamiltonian problem, mesh and hypercube embeddings in the generalized Fibonacci cube. For the Hamiltonian problem the following optimal results have been obtained:

1. Γ_n^k contains a Hamiltonian path. Hence a linear array of size $F_n^{[k]}$ can be embedded in Γ_n^k with unit dilation.

2. The longest cycle in Γ_n^k (where $n - 3 \geq k \geq 2$) contains $F_n^{[k]}$ nodes if $F_n^{[k]}$ is even, and $F_n^{[k]} - 1$ nodes if otherwise. Consequently, a ring of size $F_n^{[k]}$ can be embedded with dilation-1 if $F_n^{[k]}$ is even, with dilation-2 if otherwise.

Using the result of Hamiltonian path, we have shown that certain 2D and 3D meshes can be embedded in the generalized Fibonacci cube with expansion-1 and dilation-1. Moreover, we have also shown how to embed a hypercube in the generalized Fibonacci cube.

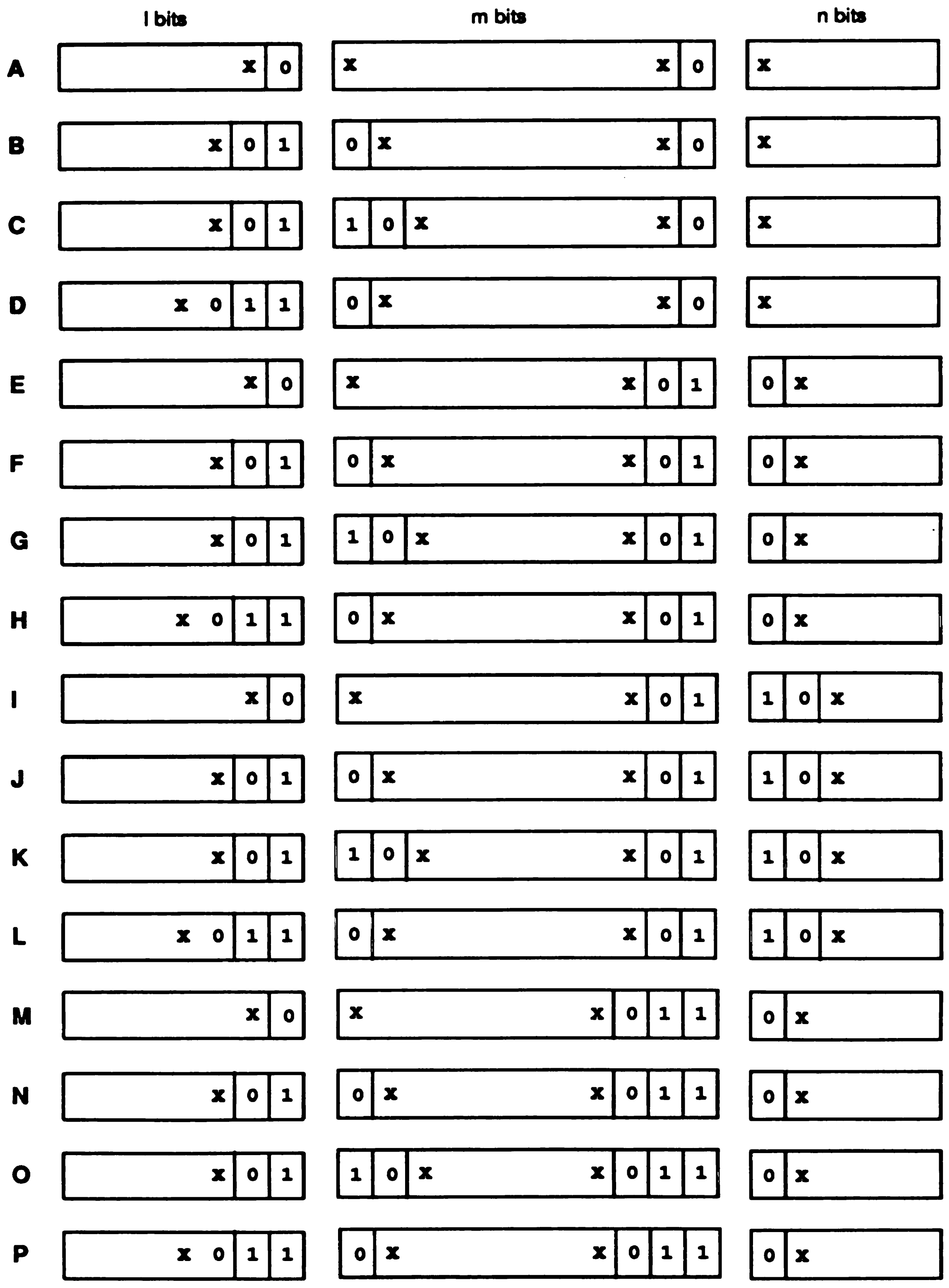
l bits
m bits
n bits
A x 0 | x x 0 | x
B x 0 1 | 0 x x 0 | x
C x 0 1 | 1 0 x x 0 | x
D x 0 1 1 | 0 x x 0 | x
E x 0 | x x 0 1 | 0 x
F x 0 1 | 0 x x 0 1 | 0 x
G x 0 1 | 1 0 x x 0 1 | 0 x
H x 0 1 1 | 0 x x 0 1 | 0 x
I x 0 | x x 0 1 | 1 0 x
J x 0 1 | 0 x x 0 1 | 1 0 x
K x 0 1 | 1 0 x x 0 1 | 1 0 x
L x 0 1 1 | 0 x x 0 1 | 1 0 x
M x 0 | x x 0 1 1 | 0 x
N x 0 1 | 0 x x 0 1 1 | 0 x
O x 0 1 | 1 0 x x 0 1 1 | 0 x
P x 0 1 1 | 0 x x 0 1 1 | 0 x

Figure 3.12. Dividing an $(l + m + n)$-bit 3-FC into an l-, m- and n-bit 3-FCs.

CHAPTER 4

DATA COMMUNICATION IN GENERALIZED FIBONACCI CUBE

Efficient data communications are critical to the performance of the parallel or distributed systems. Many previously known results have focused on systems that are based on regular interconnection topologies such as the hypercubes and meshes [58] [67] [92] [93] [100] [106]. In this chapter, we design and analyze algorithms for data communications in the generalized Fibonacci cubes.

Algorithms for routing and single node broadcasting messages in hypercubes have been extensively studied in the past: Deadlock-free and shortest-path routing algorithm is first presented by Sullivan and Bashkow [106]. By employing an integer *weight* [106] for bookkeeping, they have also presented an optimal single node broadcasting algorithm. In [100], the communication latency was considered by Saad and Schultz in designing algorithms for single node broadcasting and other data communication primitives. By splitting packets and using multiple paths, several optimal and near optimal broadcasting algorithms have been reported by Ho and Johnsson [58, 67]. Unfortunately, none of these routing and single node broadcasting algorithms applies to the generalized Fibonacci cube, even though they are subgraphs of the hypercubes.

Message routing under the faulty hypercube has also drawn attention from researchers. Chen and Shin [23, 24] have proposed a *depth-first search* scheme for routing in an injured hypercube; Gordon and Stout [51] have studied *random* and *sidetracking* routing for the hypercube in the presence of faults. However, these approaches do not guarantee that the paths produced are the shortest possible and deadlock-free.

Routing and single node broadcasting algorithms for the Incomplete Hypercube are studied by Katseff [68]. We showed in [81] that the algorithms designed for the Incomplete Hypercubes can be applied to the (second-order) Fibonacci cubes as well, which is a little surprising because the latter appear to be less regular than the former. The extension of the single node broadcasting algorithm to the generalized Fibonacci cubes has been reported in [80]. In this chapter we further extend our work and report results on the following communication primitives:

1. Single node broadcasting,
2. Single node accumulation,
3. Single node gather,
4. Single node scatter,
5. Multinode broadcasting, and
6. Multinode accumulation.

Communication in the system is achieved by *message* passing on links. A message originated from one processor (node) and destined for another may be routed through some intermediate processors. A *path* is represented by an ordered list of nodes (processors) in which every two consecutive nodes are connected by a link (edge). The *length* of a path is the number of links in the path. A link has a *link number i* if and only if it connects two nodes whose addresses differ in exactly bit i.

The *relative address* of two nodes is the bitwise Exclusive-OR, denoted by $\oplus$, of their addresses (which are binary codes for nodes in the hypercube or Fibonacci codes for nodes in the generalized Fibonacci cube). For convenience, we use $\mathcal{E}_n^j$ to denote an n-bit binary string with bit j being 1 and all other bits being 0, *i.e.*, $\mathcal{E}_n^j = e_{n-1}e_{n-2}\cdots e_1e_0$ where $e_j = 1$ and $e_i = 0\ \forall i \neq j$.

The organization of this chapter is as follows. We present distributed routing algorithm in Section 4.1, and show that the routing algorithm carries two most desirable properties, *i.e.*, all the paths determined are the shortest possible and deadlock-free. The distributed shortest-path, deadlock-free single node broadcasting algorithm is presented in Section 4.2, which is based on a spanning tree (called STF) on the generalized Fibonacci cube. In Section 4.3, we analyze the single node broadcasting in two models: We show that our proposed *all-port* broadcasting algorithm is optimal in terms of minimizing time steps. By using the *most-significant-link-first* scheduling discipline, our *one-port* broadcasting algorithm is shown to be optimal for many cases. Results for the single node accumulation are reported in Section 4.4. Algorithm for the single node scatter/gather is presented and analyzed in Section 4.5. Section 4.6 reports the results for the multinode broadcasting and accumulation. Finally, we summarize our results in Section 4.7.

4.1 Routing

The routing problem we studied in this section is to send a message from one node to another node. Katseff [68] has devised a routing algorithm for the Incomplete Hypercube. The key idea is to identify an existing link and forward the message via it until the message reaches the destination node. We discover that a similar approach can be applied to the generalized Fibonacci cubes.

The routing algorithm is a procedure that is executed by the source (originating) node and by every intermediate node on the path to the destination. It is initiated by the source node which computes its relative address with respect to the destination node, then it sends the message to the next node on the path. This procedure is repeated until the message reaches its destination. Let (*reladd*, *msg*) denote the information to be sent on links, where *reladd* is the relative address between the destination node and an intermediate node while *msg* denotes the message to be sent. Notice that *reladd* is updated on each intermediate node when the message is being routed toward the destination. Formally, the routing algorithm is described as follows.

Algorithm 1 To send a message from node *source* to node *destination*:

```
if  node_id = source then
      $reladd \leftarrow source \oplus destination$
else /* this is an intermediate node */
      receive ($reladd$, $msg$).
if  $reladd \neq 0$ then
begin /* message has reached an intermediate node */
      Let $i$ be the bit number of the most significant 1 bit
      in the $reladd$, where link $i$ from this node exists.
      Send ($reladd \oplus \mathcal{E}^i_{n-k}$, $msg$ ) on link $i$.
end
```

4.1.1 Shortest path

In [68], Katseff proved that the length of a path determined by this algorithm, when applied to the Incomplete Hypercube, is equal to the Hamming distance between any source and destination nodes. We next show that Algorithm 1 can be applied to the generalized Fibonacci cubes; furthermore, we will show that all routing paths determined are the shortest possible. It is necessary to introduce a few notations at this point. Let $B = (b_{n-1}b_{n-2}\cdots b_1b_0)$ be an n-bit string, then B[i:j] denotes the substring $(b_ib_{i-1}\cdots b_j)$ of B, where i and j are two integers and $n-1 \geq i \geq j \geq 0$. For example, suppose that $B = (b_4b_3b_2b_1b_0)$, then $B[4:2] = (b_4b_3b_2)$. The minimum (respectively, maximum) of two integers i and j is denoted by *min(i,j)* (respectively, *max(i,j)*) Lemma 4.1 characterizes the interconnection rules for nodes in the generalized Fibonacci cubes.

Lemma 4.1 Interconnection rules : *Let p denote an arbitrary node in Γ_n^k. Let $FC(p) = p_{n-k-1}p_{n-k-2}\ldots p_1p_0$ be the k-FC of node p, and Q denote $FC^{[k]}(p) \oplus \mathcal{E}_{n-k}^j$. Then, link j from node p exists if and only if one of the following two conditions is satisfied:*

R1 $p_j = 1$, *or*

R2 $p_j = 0$ *and no k consecutive 1's appear in Q[min(n-k-1,j+k-1):max(j-k+1,0)].*

Proof: The flipping of bit j in $FC(p)$ from 1 to 0 always results in a valid k-FC, because this never creates k consecutive 1's. Hence we have the rule R1. Similarly, to obtain a new k-FC by flipping bit j in $FC(p)$ from 0 to 1 is possible if and only if the new code does not contain k or more consecutive 1's. Now, the flipping of bit j in $FC(p)$ from 0 to 1 can possibly create k or more consecutive 1's only in *Q[min(n-k-1,j+k-1):max(j-k+1,0)].* This is captured by rule R2. □

We now present an important property of Algorithm 1.

Lemma 4.2 *Let* s *and* d *be two distinct nodes in Γ_n^k, and let H(s,d) be the Hamming distance between these two nodes. Algorithm 1 always finds a path of length H(s,d) from* s *to* d.

Proof: First we prove that Algorithm 1 always succeeds in finding a link at each intermediate node. Let $FC(s) = s_{n-k-1}s_{n-k-2}\cdots s_1s_0$ and $FC(d) = d_{n-k-1}d_{n-k-2}\cdots d_1d_0$ be the k-FCs of nodes s and d respectively. If $H(FC(s), FC(d)) = 1$, then by Definition 2.3 s and d are connected. Without loss of generality, assume that $H(FC(s), FC(d)) \geq 2$. Let $R = r_{n-k-1}r_{n-k-2}\cdots r_1r_0$ be the relative address of nodes s and d. We say that R *specifies* bit i if $r_i = 1$. Assume that j is the most significant and l is the second most-significant bit specified by R. We show that either link j or link l exist(s) from node s. By repeating the same procedure, we conclude that Algorithm 1 always succeeds in finding a link at each intermediate node. To see the existence of link j or l, we distinguish between the following two cases.

Case 1: $l \geq j - k + 1$.

We further consider the following two subcases: (a) $s_j = 1$ $(d_j = 0)$. By rule R1 of Lemma 4.1 link j exists from s. (b) $s_j = 0$ $(d_j = 1)$. If $s_l = 1$ $(d_l = 0)$, then by rule R1, link l exists from s. On the other hand, when $s_l = 0$ $(d_l = 1)$ it should be clear that there is no k consecutive 1's in $FC(d)[n-k-1 : max(j-k+1, 0)]$; hence by rule R2, link j exists from s.

Case 2: $l < j - k + 1$.

If $s_j = 1$ $(d_j = 0)$, then by rule R1, link j exists from s. On the other hand, if $s_j = 0$ $(d_j = 1)$, then link j also exists from s. To see this, observe that there is no k consecutive 1's in $FC(d)[n-k-1 : max(j-k+1, 0)]$, additionally, j is the only bit where $FC(s)[n-k-1 : max(j-k+1, 0)]$ and $FC(d)[n-k-1 : max(j-k+1, 0)]$ differ. Hence by rule R2, link j exists from s.

It is clear that, every time the message is forwarded from any intermediate node, the distance to the destination is also reduced by 1. Hence the length of each routing path determined by Algorithm 1 is exactly the Hamming distance between the source and destination nodes, *i.e.*, $H(s,d)$. □

Example 4.1 Consider the routing of a message from the source node 01010 to the destination node 10101 in Γ_7^2. The Hamming distance between these two nodes is 5. Figure 4.1 shows the (directed) routing path between the source and the destination nodes given by Algorithm 1, which has a length of 5. The dotted edges represent the remaining links in Γ_7^2.

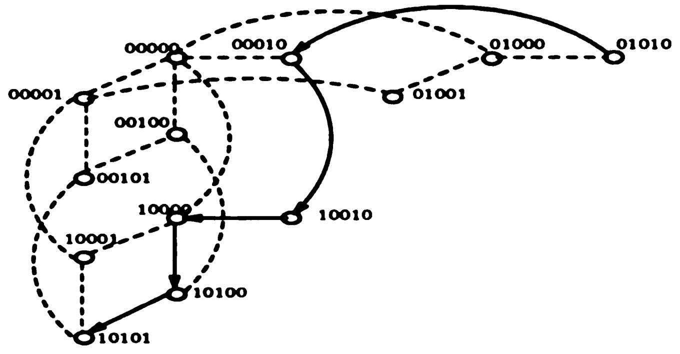

Figure 4.1. Routing path from 01010 to 10101 in Γ_7^2.

It is known that given two nodes s and d in a hypercube, it is impossible to have a path between s and t with a length less than $H(s,d)$. Since the generalized Fibonacci cube is a subgraph of the hypercube, the same bound applies. Therefore we have proved our main result in this section.

Theorem 4.1 *Algorithm 1 can be applied to any generalized Fibonacci cube to route messages between two arbitrary nodes; moreover, the path determined is alway the shortest possible.*

Corollary 4.1 *The graph Γ_n^k is connected and its diameter is $n - k$.*

Proof: Given any two nodes in Γ_n^k, Algorithm 1 always succeeds in finding a path between them; hence the graph is connected. To see the diameter, we note that the

Hamming distance between the two nodes $(1010\cdots)$ and $(0101\cdots)$ is exactly $n-k$, which is clearly the maximum between any two nodes. □

4.1.2 Deadlock

An important consideration of any message routing algorithm is to avoid deadlock when multiple sources and destinations are involved in routing concurrently. We now show that Algorithm 1 is deadlock-free for the generalized Fibonacci cubes. We assume that all links are bidirectional and there is a buffer of fixed size at either end of each link. Lemma 4.3 is taken from [68], which says that if there is a deadlock, there must be a cycle among the deadlocked nodes.

Lemma 4.3 *If a network with a finite number of nodes is deadlocked, then there exists a cycle of nodes $a_0, a_1, \cdots, a_{m-1}$ (where a node may appear more than once) in which each node a_i is blocked sending a message that has arrived from the link $(a_{(i-1)\ mod\ m}, a_i)$ and is to be sent on the link $(a_i, a_{(i+1)\ mod\ m})$.*

Theorem 4.2 shows that the condition described in Lemma 4.3 cannot occur when applying Algorithm 1 to the generalized Fibonacci cubes.

Theorem 4.2 *Algorithm 1 is deadlock-free when it is applied to the generalized Fibonacci cubes.*

Proof: Observe that each link number must appear for an even number of times in the cycle. To see this, each node in the cycle differs from the previous one in exactly 1 bit, and a bit must be flipped an even number of times so as to return to its original value. Assume that there is a deadlock in Γ_n^k. Then there is a cycle of nodes $a_0, a_1, \cdots, a_{m-1}$ as described in Lemma 4.3. Let ln_i be the link number through which a_i is intended to send a message to $a_{(i+1)\ mod\ m}$. Let $z = max\{ln_i | 0 \leq i < m\}$.

a

a

n

C

In

an

to

4.

In t

the

span

cubes

at noc

for B_n

1. T

Then there exists a node a_j at which $ln_j = z$ and ln_j represents a 1 to 0 transition in Fibonacci codes.

We now consider the message that is blocked at transmission from node a_j. Clearly, by our assumption this message came from $a_{(j-1)\,mod\,m}$ and is destined for $a_{(j+1)\,mod\,m}$ or beyond. It should be easy to see that $FC(a_{(j-1)\,mod\,m})$ and $FC(a_{(j+1)\,mod\,m})$ differ in two bits: z and z', where $z \geq z'$ and link z' connects $a_{(j-1)\,mod\,m}$ to a_j and link z connects a_j to $a_{(j+1)\,mod\,m}$. We further distinguish between the following two cases.

Case 1: $z' < z$.

In this case, node $(a_{(j-1)\,mod\,m} - F^{[k]}_{z+k})$ is in Γ^k_n, because its bit z is 0 while nodes a_j and $a_{(j-1)\,mod\,m}$ have bit z being 1. By rule R1 of Lemma 4.1, there is a link between $a_{(j-1)\,mod\,m}$ and $(a_{(j-1)\,mod\,m} - F^{[k]}_{z+k})$. Therefore, by Algorithm 1, a_{j-1} would send the message along link z (instead of link z'), which is a contradiction.

Case 2: $z' = z$.

In this case, there are exactly two nodes in the cycle, *i.e.*, $a_{(j-1)\,mod\,m} (= a_{(j+1)\,mod\,m})$ and a_j. Therefore z' must represent a transition from 0 to 1, since z is defined as a 1 to 0 transition. This is also a contradiction, because all links are bidirectional. □

4.2 Single Node Broadcasting

In this section, we study the problem of sending a message from one node to all the other nodes, *i.e.*, the *single node* [9] (or *one-to-all* [67]) broadcast. The familiar *spanning binomial trees*(SBTs) [67] have been used for broadcasting in the Boolean cubes [67, 100, 106]. Figure 4.2 displays an example of a (directed) SBT in B_4 rooted at node 0000, where links are labeled by their link numbers. Broadcasting algorithms for B_n based on the SBT can be schematically described as follows (see *e.g.*, [100]).

1. The source node sends the broadcast message along its link $n-1$.

2. For $i = 2, 3, \cdots n$ do:
   Each node (including the source node) that has received the broadcast message in a previous step sends the message via its link $n - i$.

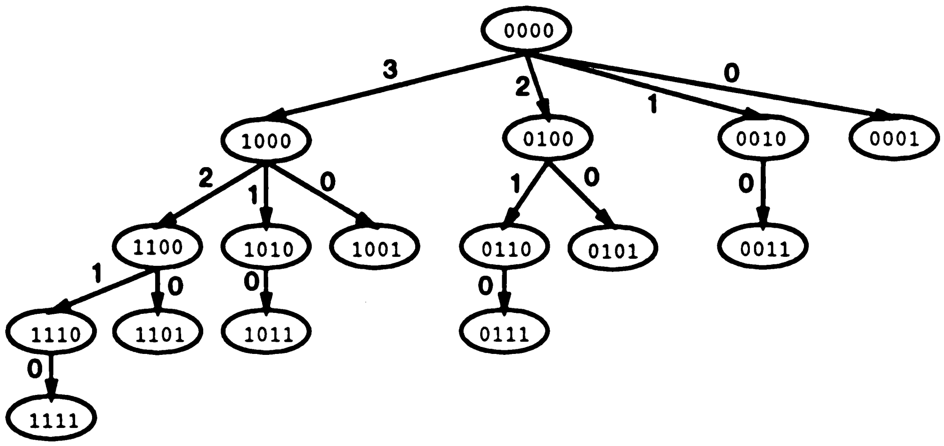

Figure 4.2. An SBT rooted at node 0000.

The above description also demonstrates that a message broadcast from a node to all the other nodes in B_n can be finished in n steps based on properties of the SBT. In the following, we view the SBTs as *directed* trees, where the direction of a link is defined as the direction in which the message flows. Given a node p, the link from which p receives its message(s) is called an *in-edge* of p, similarly, the link through which p sends its message(s) is called an *out-edge* of p. It is clear that the root of an SBT has out-edge(s) only, leaf nodes have in-edges, whereas all the other nodes have one in-edge and at least one out-edge. We note that different SBTs may be obtained by using different construction rules. The SBTs we will be referring to in the remainder of this presentation are obtained according to the following rules: (1) The root node selects *all* of its incident links as out-edges, and (2) Let i be a non-root node and l_i the link number of its in-edge. Node i selects all of its incident link(s) whose link number is less than l_i as out-edge(s) in constructing the SBT (see g

a

a

e

tr

fr

ca

Ex

lin

elli

the

init

I

in ro

sendi

Fig. 4.2).

In [106], an integer *weight* is used to implement the SBT-based broadcasting in Boolean cubes. The key idea is to sent a weight along with the message to indicate which link(s) each receiving node should use to forward the message. Katseff [68] modified this idea for broadcasting messages in Incomplete Hypercubes. His algorithm uses an array (called *travel*) of size $\lceil \log N \rceil$ to keep track of all link information during the broadcast process; it records links that have not been traversed due to their absence in the previous node(s) so that they can be compensated for in a later stage once these links exist.

We show that the same strategy can be applied to perform broadcasting on the generalized Fibonacci cubes. Specifically, let *travel*[i] denote the i^{th} element of the array. The source node originates the broadcast by setting all elements in the travel array to TRUE, and it sends the message plus an updated travel array via each link existing from the source node. Suppose that a message received at a node t with *travel*[i] set to TRUE, then the message will be sent along link i provided it exists from node t.

Let (*travel*, *msg*) denote the information to be forwarded. The detailed broadcasting algorithm is shown as follows.

Example 4.2 Consider a broadcast from node 01010 in Γ_7^2. Figure 4.3 shows the links determined by Algorithm 2 in the broadcast. Each node (represented by an ellipse) is labeled by its 2-FC. The links shown are those determined by Algorithm 2; the *travel* array received at each node is shown above it. Note that the source node initializes the array to all 1's (TRUE).

If we further review Figures 4.1 and 4.3, it is not difficult to see that the path used in routing messages from node 01010 to node 10101 is exactly the same one used in sending a broadcasting from node 01010 to node 10101. This is no coincidence. We *n*

th

pr

no

a b

fact

fron

with

Theo

by Al

Algori

```
Algorithm 2 Single node broadcasting in $\Gamma_n^k$:
if node_id = source then
    set all bits in travel to TRUE
else /* This is not the source node. */
    receive (travel, msg ).
if none of the links specified by the travel array exists
    OR all elements in the travel array are FALSE,
then stop.
for each link $l$ that exists from this node:
begin
    if travel[$l$] is TRUE then
        send (newtravel, msg) on link $l$ with newtravel computed
          as follow:
        for $i \in \{j | 1 \leq j \leq n - k\}$ :
            newtravel[$i$] $\leftarrow$ TRUE iff
                (travel[$i$] is TRUE)   AND
                ($i < l$ OR link $i$ from this node does not exist)
end
```

now show that the broadcasting algorithm (Algorithm 2) routes broadcast messages through *exactly* the same path given by the routing algorithm (Algorithm 1). More precisely, let p be the path determined by Algorithm 1 to route a message from a node s to another node t, and let p' be the path determined by Algorithm 2 to route a broadcast message originated from s to t. Then we show that $p = p'$. Given this fact, it should be easy to see that Algorithm 2 routes a broadcast message originated from any node to every other node exactly once and it does so along a shortest path with length no greater than $n - k$.

Theorem 4.3 *Let* s *and* t *be two distinct nodes in* Γ_n^k. *Let* p *be the path determined by Algorithm 1 to route a message from* s *to* t. *If a message is broadcast from* s *by Algorithm 2, then it will reach* t *along a unique path that coincides with* p.

(

n

P

or

by

p_i

no

ele

and

sign

p_{i-1}

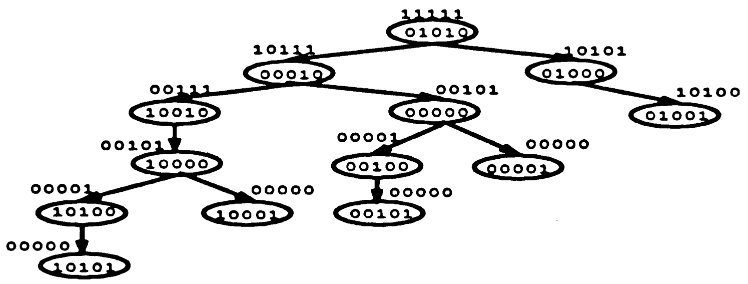

Figure 4.3. Broadcast from node 01010 in Γ_7^2.

Proof: Let p' be a path determined by Algorithm 2 to route the broadcasting message from s to t. We prove the equivalence of the two paths p and p' in two steps: We first show that (Claim 1) the broadcasting algorithm routes the message via the path determined by the routing algorithm. Then we prove that (Claim 2) the broadcast messages are not routed along any other path. □

Claim 1: *When applying Algorithm 2, the path p is used to route the broadcasting messages from s to t.*

Proof: Let d be the Hamming distance between s and t. Define p_i to be the i^{th} node on path p. In other words, $p = (p_0, p_1, \cdots, p_{d-1}, p_d)$ where $p_0 = s$ and $p_d = t$. We prove by induction on i that the message broadcast from s based on Algorithm 2 reaches p_i via the path $(p_0, p_1, \cdots, p_{i-1})$, where $0 \leq i \leq d$; in addition, *travel*$[j]$ received at node p_{i-1} is also set to TRUE for every bit j in which $FC(p_{i-1})$ and $FC(t)$ differ.

For $i = 0$, it should be clear that $p_0 = s$ and Algorithm 2 starts out by setting all elements in the travel array to TRUE.

Assume that the hypothesis is true for $i - 1$, where i is any arbitrary integer and $0 < i \leq d$. We will prove that the lemma is true for i. Let z be the most significant bit specified by the relative address of p_{i-1} and t such that link z from p_{i-1} exists. (From the proof of Lemma 4.2, we deduce the existence of z.) By the definition of path p, Algorithm 1 routes the message from p_{i-1} to p_i along link z. By induction hypothesis, *travel*$[z]$ received at p_{i-1} is TRUE. Hence Algorithm 2 also sends the broadcast message from p_{i-1} to p_i via link z. It remains to show that *travel*$[j]$ received at p_i is set to TRUE for every bit j in which $FC^{[k]}(p_i)$ and $FC^{[k]}(t)$ differ. The following three observations complete our proof:

1. $FC(p_{i-1}) \oplus FC(t)$ and $FC(p_i) \oplus FC(t)$ differ in exactly bit z.

2. For $0 \leq j < z$, the *travel*$[j]$ received at p_i is the same as that received at p_{i-1}, since Algorithm 2 never changes these bits (which, by induction hypothesis, are set appropriately when they reach p_{i-1}).

3. For $z < j < n - k - 1$, the *travel*$[j]$ received at p_{i-1} is TRUE if and only if link j does not exist from any node in the path $(p_0, p_1, \cdots p_{i-2})$. Obviously, link j does not exist from p_{i-1} either. Otherwise, Algorithm 1 would select link j to route the message (instead of link z). Therefore, for $z < j < n - k - 1$, the *travel*$[j]$ received at p_i is also set to TRUE, wherever $FC(p_i)$ and $FC(t)$ differ.

$\square$

Claim 2: *When applying Algorithm 2, p is the only path to route the broadcasting messages from s to t.*

Proof: Let p' denote a second path determined by Algorithm 2 to route a message from s to t such that $p' \neq p$. Define p_i (respectively, p'_i) to be the i^{th} node of path p (respectively, p'). Let j be the smallest integer where $p_j \neq p'_j$, *i.e.*, $p_i = p'_i$ for $0 \leq i < j$ and $p_j \neq p'_j$. Let z be the bit in which $FC(p_{j-1})$ and $FC(p_j)$ differ, and let z' the bit in which $FC(p'_{j-1})$ $(=FC(p_{j-1}))$ and $FC')$ differ. That is, $FC(p_{j-1})$ differs from $FC(t)$ in both bits z and z'. By assumption $p_j \neq p'_j$, it should be clear that $z \neq z'$ and both links z and z' from p_{j-1} exist. Consider the following two cases:

Case 1: $z > z'$.

(
t

F
A
w
dc

re
ho
of
tha
sho

span
diffe
cube
span

According to Algorithm 2, *travel*$[z]$ received at p'_j would be set to FALSE. Therefore the broadcast message would never reach node t, which is a contradiction to Claim 1.

Case 2: $z < z'$.

Algorithm 1 would send the message along link z' (instead of link z), which is also a contradiction.

Hence we conclude that $p = p'$. □

An important result follows from Theorem 4.3.

Corollary 4.2 *The nodes and links traversed by executing Algorithm 2 for broadcasting from a given node in Γ_n^k form a spanning tree.*

Proof: By Corollary 4.1, Γ_n^k is connected with diameter $n-k$. Furthermore, initially Algorithm 2 sets all bits in the *travel* array to TRUE. It can be shown that Algorithm 2 will traverse all nodes in Γ_n^k. It remains to show that all edges traversed by Algorithm 2 do not form any cycle.

Let s be the source node to originate the broadcast. By Lemma 4.2 and Theorem 4.3, the broadcast message will reach a node, say p_j, in exactly $H(FC(s), FC(p_j))$ hops. Hence the only possible situation for any cycle involving p_j is to have two copies of the broadcast message arrive at p_j from two distinct nodes, say p_{j-1} and p'_{j-1}, such that both nodes have the same distance $H(FC(s), FC(p_j)) - 1$ from the source s. It should be easy to see that this is a contraction to Theorem 4.3. □

The spanning tree generated by executing Algorithm 2 will be referred to as the *spanning tree for the generalized Fibonacci cube* (STF). We note that two STFs with different roots are generally *not* isomorphic. In contrast, all SBTs of a Boolean cube are isomorphic. In the next section, we will closely examine these two kinds of spanning trees and show how to transform an SBT to an STF in order to analyze the

4

W

th

B

of

op

ste

Th

the

can

time required by the single node broadcasting algorithm.

4.3 Analysis of Single Node Broadcasting

In this section, we will analyze the time complexity of Algorithm 2. Two communication models are considered as in [9, 67]. In the *one-port* [67] (or *single link availability* [9]) model, a processor can only send and receive a message on one of its communication ports at any given time. On the other hand, in the *all-port* [67] (or *multiple link availability* [9]) model a processor can communicate (send and receive) on *all* of its ports concurrently. The one-port model is more suitable for networks with a relative high degree of interconnection from the implementation point of view.

For convenience, we will follow the model used in [9]. In other words, we will use the number of *time steps* to measure the time requirement; all messages are assumed to be of unit length and it takes one time step to send (forward) a message from a node to its adjacent node(s).

4.3.1 All-port Model

We first consider the all-port communication model in the Boolean cube. Notice that the farthest node from any given node in B_n is at a distance of n (the diameter of B_n). Hence n is a lower bound of any broadcasting algorithm for B_n. Since the height of any SBT in B_n is exactly n, broadcasting algorithms based upon the SBTs are optimal. Similarly, by Theorems 4.1 and 4.3, Algorithm 2 requires only minimal time steps to complete any single node broadcasting in the generalized Fibonacci cubes.

Theorem 4.4 *Let d_s be the maximal distance between an (arbitrary) node s and all the other nodes in Γ_n^k. Then, under the all-port communication model Algorithm 2 can optimally complete a broadcast from s in exactly d_s time steps.*

c
c
j
ru
th
fol
sp
def

bec
be s
the
Fig 4
exact
requi
is nec
point.

Notice that for Γ_n^k we have $\lfloor\frac{(n-k)(k-1)}{k}\rfloor \leq d_s \leq n-k$, and the lower bound $\lfloor\frac{(n-k)(k-1)}{k}\rfloor$ is achieved when broadcasting from node 0.

4.3.2 One-port Model

In the one-port model, only one port is available for transmitting message at any moment. Hence the *scheduling scheme* applied to nodes which need to send/forward a broadcast message to multiple nodes will make a critical difference in the overall broadcasting time. With the Boolean cubes, it is known [67] that SBTs combined with the "Tallest Remaining Subtree First" (TRSF) scheduling discipline results in an optimal one-port broadcasting algorithm. We now present a scheduling discipline called *Most-Significant-Link-First* (MSLF) for the generalized Fibonacci cubes. Recall that under Algorithm 2 a node p should forward the message through every link j provided that the corresponding *travel*[j] received at p is TRUE. With the MSLF rule, each node first sends the message via the specified most-significant link, then via the specified second most-significant link, and so on. In other words, a node should follow the left-to-right scanning order in sending messages to multiple links that are *specified* in the received *travel* array. Our *one-port broadcasting algorithm* is thus defined as Algorithm 2 combined with the MSLF scheduling.

Notice that with the Boolean cubes the MSLF reduces to the TRSF scheme, because there are no missing links. Let l_i be the link number of a link in an SBT; it can be shown that $n-l_i$ is exactly the time step at which the MSLF scheme used to forward the broadcast messages when the one-port broadcasting algorithm is applied (cf. Fig 4.2). Consequently, the time required by our one-port broadcasting algorithm is exactly n for B_n. We next show that our one-port single node broadcasting algorithm requires no greater than $n-k$ time steps for any single node broadcasting in Γ_n^k. It is necessary to examine closely the relation between the STF and the SBT at this point.

t

li

u

Th

cor

and

link

This

numb

W

rithm

the se

SBT ar

Lemma

connecti

sume tha

1. Nod

2. With

To simplify our presentation, in the remainder of this section a node in the Boolean cube or the generalized Fibonacci cube will be referred to by its *binary address* (which is either a binary code or a k-FC). For convenience, we introduce a few new notations. Let $SBT_n(s)$ denote the SBT rooted at a node s in B_n. Similarly, let $STF_n^k(s)$ denote the STF rooted at a node s in Γ_n^k.

A path between node s and node t can be also represented as an ordered list of link numbers which comprises a sequence of link numbers leading from s to t. In this case, we will refer to *link sequence* to emphasize that the path is in terms of link numbers instead of node labels. The relation between a path p and its *corresponding link sequence* q is as follows. Let $p = (u_0, u_1, \cdots u_{i-1}, u_i)$ denote a path from node u_0 to node u_i and $q = (l_1, l_2, \cdots l_{i-1}, l_i)$ the link sequence corresponding to the path. Then $u_j = u_0 \oplus l_1 \oplus l_2 \cdots \oplus l_j$ for $1 \leq j \leq i$. As an example, in Fig. 4.2 the path connecting nodes 0000 and 1111 in the $SBT_4(0000)$ is (0000, 1000, 1100, 1110, 1111) and the corresponding link sequence is (3,2,1,0). Let $(l_1, l_2, \cdots, l_i)$ be an arbitrary link sequence in an SBT. It is important to note that $l_1 > l_2 > \cdots > l_i$ is always true. This is because, as defined in Section 4.2, for any internal node of an SBT the link number(s) of the out-edge(s) is less than that of the in-edge.

We now proceed to analyze the time required by our one-port broadcasting algorithm when it is applied to the generalized Fibonacci cubes. We write $\mathcal{V}_n^k$ to represent the set of k-FCs for all nodes in Γ_n^k. Lemma 4.4 establishes a relation between an SBT and an STF, which will enable us to transform an SBT to an STF.

Lemma 4.4 *Let u_0 be a node in $\mathcal{V}_n^k$. Let (u_i, u_{i+1}, u_{i+2}) be the path in $SBT_{n-k}(u_0)$ connecting node u_i to node u_{i+2}, and (l_{i+1}, l_{i+2}) the corresponding link sequence. Assume that u_{i+2} is in $\mathcal{V}_n^k$ and u_{i+1} is not in $\mathcal{V}_n^k$. Then*

1. *Node u_i must be in $\mathcal{V}_n^k$,*

2. *Within B_{n-k}, link l_{i+2} from u_i leads to a node r that must be in $\mathcal{V}_n^k$, and*

b

n

fl

o

(1

u_i

(2

by

(3)

con

in $\mathcal{V}$

With

node

3. Within B_{n-k}, link l_{i+1} from r leads to u_{i+2}.

Proof: First we prove that node u_i is in $\mathcal{V}_n^k$ by contradiction. Assume that node u_i is not in $\mathcal{V}_n^k$, we distinguish between the following two cases. (Notice that u_i and u_{i+1} differ in exactly bit l_{i+1}.)

Case 1: There are k 1's in both $u_i[n-k-1 : l_{i+1}+1]$ and $u_{i+1}[n-k-1 : l_{i+1}+1]$. This implies that there are k 1's in $u_{i+2}[n-k : l_{i+1}+1]$, because u_{i+2} inherits these bits from u_{i+1}. Therefore node u_{i+2} would not be in $\mathcal{V}_n^k$, which contradicts to our assumption.

Case 2: There are k 1's in both $u_i[l_{i+1}-1 : 0]$ and $u_{i+1}[l_{i+1}-1 : 0]$.
This implies that there are k 1's in $u_0[l_{i+1}-1 : 0]$, because both u_i and u_{i+1} inherit these bits from u_0. Therefore node u_0 would not be in $\mathcal{V}_n^k$, which is also a contradiction.

We now prove that r is in $\mathcal{V}_n^k$. Let $(b_{n-k-1}, b_{n-k-2}, \cdots, b_{l_{i+1}}, b_{l_{i+1}-1}, \cdots, b_{l_{i+2}}, \cdots, b_0)$ be the binary code of u_i. We claim that $b_{l_{i+1}} = 0$ and $b_{l_{i+2}} = 1$. Consequently, node r must be in $\mathcal{V}_n^k$, because r is obtained from u_i (which has been proven to be in $\mathcal{V}_n^k$) by flipping bit l_{i+2} from 1 to 0. To verify the previous claim, we examine all the three other possibilities:

(1) $b_{l_{i+1}} = 0$ and $b_{l_{i+2}} = 0$. Node u_{i+2} would not be in $\mathcal{V}_n^k$, since u_{i+2} is obtained from u_{i+1} (which is not in $\mathcal{V}_n^k$) by flipping bit l_{j+2} from 0 to 1. This is a contradiction.

(2) $b_{l_{i+1}} = 1$ and $b_{l_{i+2}} = 0$. Node u_{i+1} would be in $\mathcal{V}_n^k$, since u_{i+1} is obtained from u_i by flipping bit l_{i+1} from 1 to 0. This is a contradiction.

(3) $b_{l_{i+1}} = 1$ and $b_{l_{i+2}} = 1$. Similar to (2), node u_{i+1} would be in $\mathcal{V}_n^k$. This is also a contradiction.

Finally, the edge (r, u_{i+2}) is in B_{n-k} (more precisely, Γ_n^k), because both nodes are in $\mathcal{V}_n^k$ and they differ in exactly bit l_{i+1} as we just observed. □

With Lemma 4.4, we can devise a scheme to transform an SBT to an STF. Let u_0 be a node in $\mathcal{V}_n^k$. A link (p, q) in $SBT_{n-k}(u_0)$ can be classified into the following four groups:

E

S.

ST

an

bro

inst

are

link

We n

(1) both p and q are in $\mathcal{V}_n^k$, (2) p is in $\mathcal{V}_n^k$, whereas q is not, (3) p is not in $\mathcal{V}_n^k$, whereas q is, or (4) neither p nor q is in $\mathcal{V}_n^k$. The scheme is as follows. We retain all group 1 links, remove all group 2 links together with the corresponding $q's$, remove all group 3 links together with the corresponding $p's$, and remover links in group 4 together with the corresponding $p's$, $q's$. Each node q in group 3 needs special attention, since the corresponding p had been removed. Fortunately, Lemma 4.4 guarantees that there exists a node in $\mathcal{V}_n^k$, which has a link to q and, hence, can become the new parent of q. Algorithm 3 describes a method to transform an SBT to an STF.

```
Algorithm 3 To transform an SBT to an STF:
/* Assume that $u_0$ is an arbitrary node in $\mathcal{V}_n^k$ */
Step 1 Remove every node (along with all incident edges) that is
    not in $\mathcal{V}_n^k$.
Step 2 For each node $u$ in $\mathcal{V}_n^k$, whose parent has been removed in Step
    1, find its new parent by applying Lemma 4.4 and add a new
    link between them.
```

Example 4.3 Figure 4.4 illustrates a construction of $STF_8^3(01100)$ from $SBT_5(01100)$. For comparison, nodes not in $\mathcal{V}_n^k$ are signified by dotted ellipses in $STF_8^3(01100)$. There are two labels for each link in the STF: one is its link number and the other (parenthesized) signifies the routing step determined by the one-port broadcasting algorithm to broadcast a message from the root. To form the STF, for instance, node 11100 is deleted from the SBT in Step 1, and nodes 10100 and 11000 are respectively connected to nodes 00100 and 01000. Notice that (4,3) and (3,4) are link sequences from node 01100 to node 10100 in the SBT and the STF respectively.

We now prove the correctness of the above algorithm.

L

S

P

It

the

tha

now

whi

conn

We

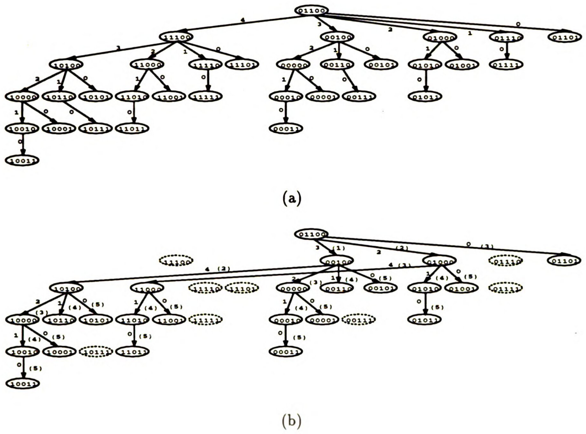

Figure 4.4. Construction of (b) $STF_8^3(01100)$ from (a) $SBT_5(01100)$.

Lemma 4.5 *Given an $SBT_{n-k}(u_0)$ as its input, Algorithm 3 correctly produces $STF_n^k(u_0)$, provided that u_0 is in $\mathcal{V}_n^k$.*

Proof: It should be clear that the graph generated by Algorithm 3 is still a tree. It remains to show that this tree is $STF_n^k(u_0)$. Consider three nodes u_i, u_{i+1} u_{i+2} and the corresponding link sequence (l_{i+1}, l_{i+2}) as specified in Lemma 4.4. It can be shown that there exists a path from u_i to u_{i+2} in $STF_n^k(u_0)$ and link l_{i+1} is in that path. We now prove by contradiction that (l_{i+2}, l_{i+1}) is exactly the link sequence in $STF_n^k(u_0)$, which connects u_i to u_{i+2}. Assume that (l'_{i+2}, l_{i+1}) is the link sequence in $STF_n^k(u_0)$ connecting u_i to u_{i+2}, where $l'_{i+2} \neq l_{i+2}$. There are two possible cases: (1) $l'_{i+2} > l_{i+2}$. We claim that $l_{i+1} > l'_{i+2}$. Because if the claim were false, then the link sequence a

ty

Le

exa

Pro

STF

and *u*

for *th*

have *i*

u are 0

Case 1

Without

shows a

(l'_{i+2}, l_{i+1}) from u_i would lead to a node other than u_{i+2}. Consequently, by definition of SBT, (l_{i+1}, l'_{i+2}) is a link sequence from u_i in $SBT_{n-k}(u_0)$, which also leads to a node other than u_{i+2}. This is a contradiction. (2) $l'_{i+2} < l_{i+2}$. In this case, it is clear that (l_{i+1}, l'_{i+2}) is a link sequence from u_i in $SBT_{n-k}(u_0)$, and this link sequence would lead to a node other than u_{i+2}. This is also a contradiction. □

Observe that links in STFs can be classified into two sets: set A includes the links added in Step 2 of Algorithm 3, and set B the remaining links (*i.e.*, the links that also exist in the corresponding SBT). For example, in $STF_8^3(01100)$ the links (00100,10100) and (01000,11000) are in set A, and all other links are in set B (refer to Fig 4.4). Links in set A are resulted from the "missing" nodes which have k 1's in their binary addresses. Each link l in set A connects two special nodes: the one which defines l as an out-edge is called type-α node and the other (which defines l as an in-edge) type-β node. We next show that each type-α node has exactly one type-β node as its child.

Lemma 4.6 *Let s be a node in Γ_n^k and u a type-α node in $STF_n^k(s)$. Then u has exactly one type-β node as its child in $STF_n^k(s)$.*

Proof: Let p be the parent of u and m the link number of the link (p, u) in the STF. We prove the lemma by contradiction. Assume that u has *two* type-β nodes v and w as its children in the STF, additionally, i and l (where $i > l$) are link numbers for the links (u, v) and (u, w), respectively. Since both links i and l are in set A, we have $i > l > m$. From the proof of Lemma 4.4, we deduce that all bits i, l and m of u are 0's. We further distinguish among the following three cases:

Case 1: $i \geq l + 2$ and there exists j such that $i > j > l$ and bit j of u is 1.

Without loss of generality, assume that $i = j+1, j = l+1$ and $l = m+1$. Figure 4.5 (a) shows a portion of the SFT, where nodes are specified by bits i, j, l and m. Since v

As
the

The

is in the STF, there is no k consecutive 1's in $v[n-k-1:j]$. Similarly, there is no k consecutive 1's in $p[l:0]$. Hence, there exists a node u' in the STF, whose k-FC is $v[n-k-1:j] \cdot p[l:0]$. Consequently, at node p Algorithm 2 would forward the message to u', and u' would become the parent of v. This contradicts our assumption that u is the parent of v.

Case 2: $i \geq l+2$ and all bits in $u[i-1, l+1]$ are 0's.

Again, we may assume that $i = j+1, j = l+1$ and $l = m+1$. Figure 4.5 (b) illustrates the situation. Similarly, node u' (specified by 1001) is in the STF, and u' would be the parent of v in stead of u, which is also a contradiction to our assumption that u is the parent of v.

Case 3: $i = l+1$.

This is a degenerated situation of Case 2. □

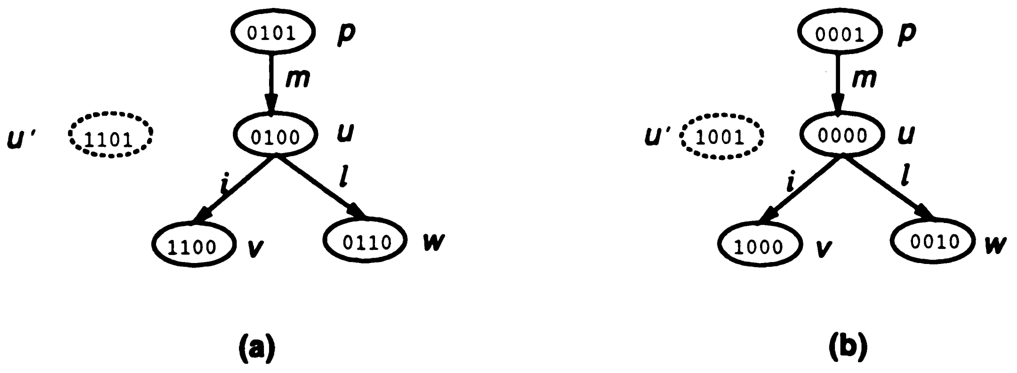

Figure 4.5. Type-α node and its type-β child.

As a consequence of Lemmas 4.5 and 4.6, the next theorem gives an upper bound on the time required by our one-port single node broadcasting algorithm.

Theorem 4.5 *The one-port broadcasting algorithm requires no greater than n-k time*

I

R

m

the

dec

affe

steps for any single node broadcasting in Γ_n^k.

Proof: Let u_0 be the source node to initiate the broadcast. We consider the scheduling for links in set A as defined before. Let u_i, u_{i+1}, u_{i+2} and r be four nodes as specified in Lemma 4.4. Thus links (u_i, u_{i+1}) and (u_{i+1}, u_{i+2}) are not in $STF_n^k(u_0)$; moreover, link (r, u_{i+2}) is added in Step 2 of Algorithm 3. It can be shown inductively that the forwarding of the broadcast message on link (r, u_{i+2}) can be scheduled at a time no later than that of (u_{i+1}, u_{i+2}) in the one-port broadcast in B_{n-k} by using $SBT_{n-k}(u_0)$. To see this, by Lemma 4.6, (r, u_{i+2}) is the only set A link added to node r; furthermore, it is the first link to be scheduled at r according to the MSLF scheme. With the removal of (u_i, u_{i+1}) the scheduling of (u_i, r) can be advanced by at least 1 step. Consequently, the scheduling of links in set B will not be delayed by the added links in set A. □

Remark: Let $(p_0, p_1, \cdots, p_i)$ be a path in $STF_n^k(u_0)$. Figure 4.6 shows that there may be two type-α nodes in such a path, *i.e.*, nodes 001011 and 101001. (In general, there may be more than two such nodes in a path.) However, one should be able to deduce that the the scheduling in the second type-α node, *i.e.*, 101001, will not be affected (delayed) by a type-α ancestor.

to

no

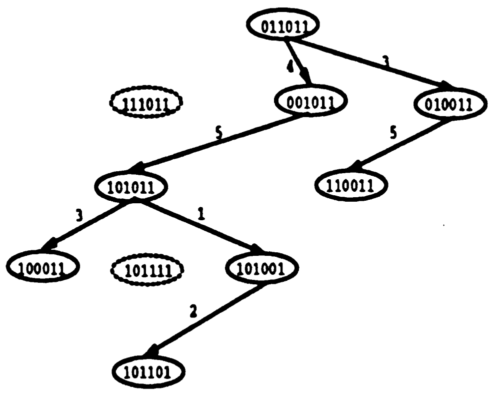

Figure 4.6. A portion of $STF_9^3(011011)$.

4.3.3 Optimality of One-port Algorithm

We now examine the optimality of our one-port broadcasting algorithm. Figure 4.7 illustrates an example of broadcasting in Γ_7^2 from source node 00000, where links are labeled by the time steps determined by the one-port algorithm. It is seen that a

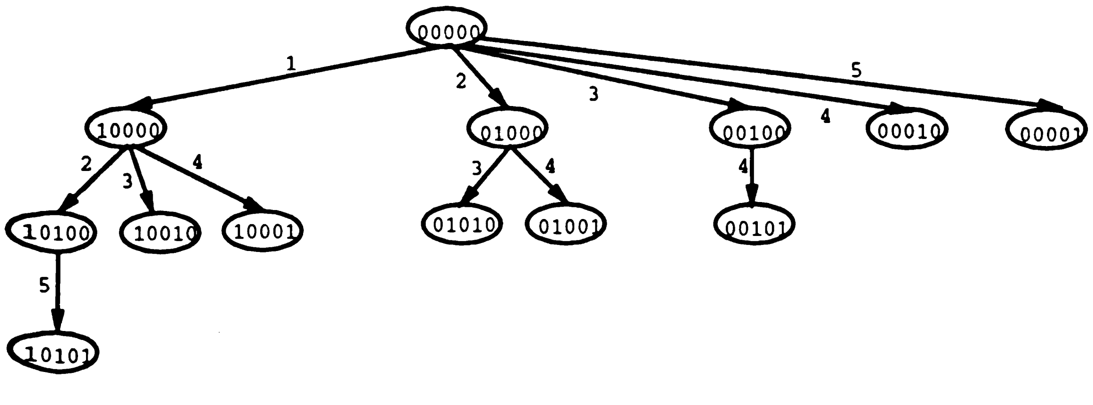

Figure 4.7. One-port broadcasting in Γ_7^2 from node 00000.

total of 5 time steps is needed to complete the broadcasting. However, by redefining node 00001 as the child of 10001 and properly rearranging the scheduling, Figure 4.8

r

re

ta

ing

$\lceil$log

broa

the b

our or

Theor

optimal

1. br

shows that the broadcasting can be completed in 4 time steps, which can be shown to be optimal.

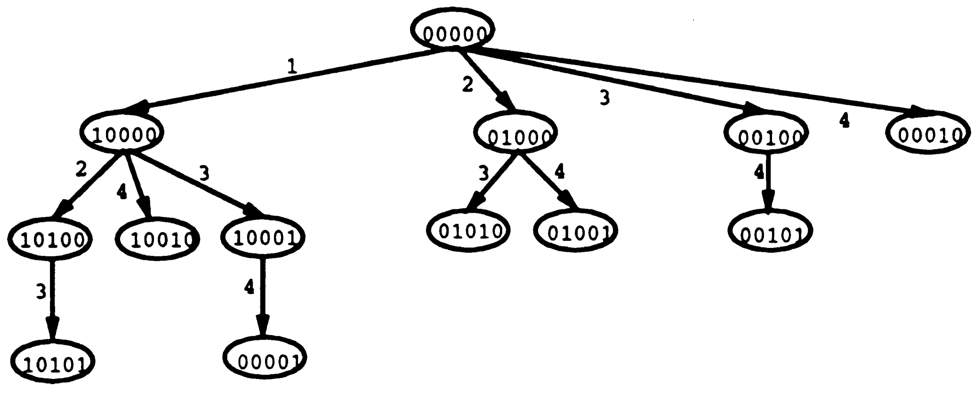

Figure 4.8. An optimal one-port broadcasting in Γ_7^2 from node 00000.

To study the optimality, we need to determine the lower bound of one-port broadcasting. Under the one-port model, each node can send messages to exactly one other node at any given time. Clearly, under the one-port model, the number of nodes that receive the broadcast message is at most doubled after each time step. Therefore, it takes at least $\lceil \log N \rceil$ time steps to complete a broadcasting for any network consisting of N nodes. Consequently, a lower bound for any one-port broadcasting in Γ_n^k is $\lceil \log F_n^{[k]} \rceil$.

Recall that Theorem 4.5 shows that $n - k$ is an upper bound for our one-port broadcasting in Γ_n^k. Let s be a node in Γ_n^k, and $\bar{s}$ the node whose binary address is the binary complement of $FC(s)$. The following theorem identifies situations where our one-port algorithm achieves optimality.

Theorem 4.6 *For $n > k \geq 2$, the one-port single node broadcasting algorithm is optimal under the following two conditions:*

1. *broadcast from any node in Γ_n^k, where $F_n^{[k]} > 2^{n-k-1}$, or*

c

ti

a

4.

A d

(or

othe

with

messa

when

additio

mulatio

into a d

Therefo

generali

2. broadcast from a node s in Γ_n^k such that the node $\overline{s}$ is also in Γ_n^k.

Proof: Condition 1 is true because the lower bound $\lceil \log F_n^{[k]} \rceil$ is equal to $n-k$. Condition 2 follows from the observation that the minimum amount of time to send a message from node s to $\overline{s}$ is $H(s,\overline{s}) = n-k$. □

For a fixed k, there exists an integer n_0 such that $F_n^{[k]} \leq 2^{n-k-1}$ for $n > n_0$; hence condition 1 may not always hold. However, since the diameter of Γ_n^k is $n-k$, the one-port algorithm may already be optimal for single node broadcasting in the generalized Fibonacci cube.

We now further show that other common communication primitives can be efficiently implemented in the generalized Fibonacci cubes. We follow the same assumptions that all messages are of unit length and it takes one time step to send (forward) a message.

4.4 Single Node Accumulation

A dual operation for single node broadcasting is the *single node accumulation* [9, 10] (or *reduction* [67]) in which a particular node is to receive messages (data) from all the other nodes. In this operation, the messages received at a node can be "combined" with the local message before transmission. Here we assume that the "combined" message still takes one time steps for transmission. This problem arises, for example, when we want to compute the sum consisting of one term from each node, where addition can be viewed as "combining" messages. The broadcasting and the accumulation are no different in concept: any broadcasting algorithm can be transformed into a dual algorithm for accumulation by simply *reversing* the data paths [9, 10, 100]. Therefore, the times required by the all- and one-port single node accumulation in the generalized Fibonacci cubes are also captured by Theorems 4.4 and 4.5, respectively.

senc

in *B*

In

ing) a

rithm

algorit

conveni

4.5 Single Node Gather and Scatter

Many algorithms require a particular node to collect one ***personalized*** message from every other node; this data transfer operation is referred to as the ***single node gather*** [9, 10, 100] (or ***all-to-one personalized communication*** [67]). Notice that in the gathering operation messages may not be "combined" (cf. accumulation). A dual operation for the single node gather is the ***single node scatter*** [9, 10, 100] (or ***one-to-all personalized communication*** [67]) in which a particular node sends different personalized messages to all the other nodes. Again, gathering and scattering are dual problems: any algorithm for gathering can be translated into a dual algorithm for scattering, and vice versa [9, 10, 100]. Algorithm 4 is a simple elegant gathering algorithm for the hypercube, which was proposed in [100].

```
Algorithm 4 Single Node Gathering in $B_n$ :
for $i = 0$ to $n-1$ do:
    All nodes addressed with $*^{n-i-1}10^i$ send messages accumulated from
    the previous steps to nodes $*^{n-i-1}0^{i+1}$.
```

It can be seen that Algorithm 4 consists of n steps: at the i^{th} step nodes $*^{n-i-1}10^i$ send 2^i messages to nodes $*^{n-i-1}0^{i+1}$. Figure 4.9 illustrates an example for gathering in B_4, where node 0 collects messages.

Indeed, the gather (scatter) algorithm is similar to the accumulation (broadcasting) algorithm. Not surprisingly, the communication graph determined by Algorithm 4 (refer to Fig. 4.9) is exactly the SBT. To design and analyze the gathering algorithm for the generalized Fibonacci cubes, we further examine the STFs. For convenience, we introduce some new terms regarding the SBTs. The *children* of a node

link

from

flippin

node u

and r =

defined

node u

flipping

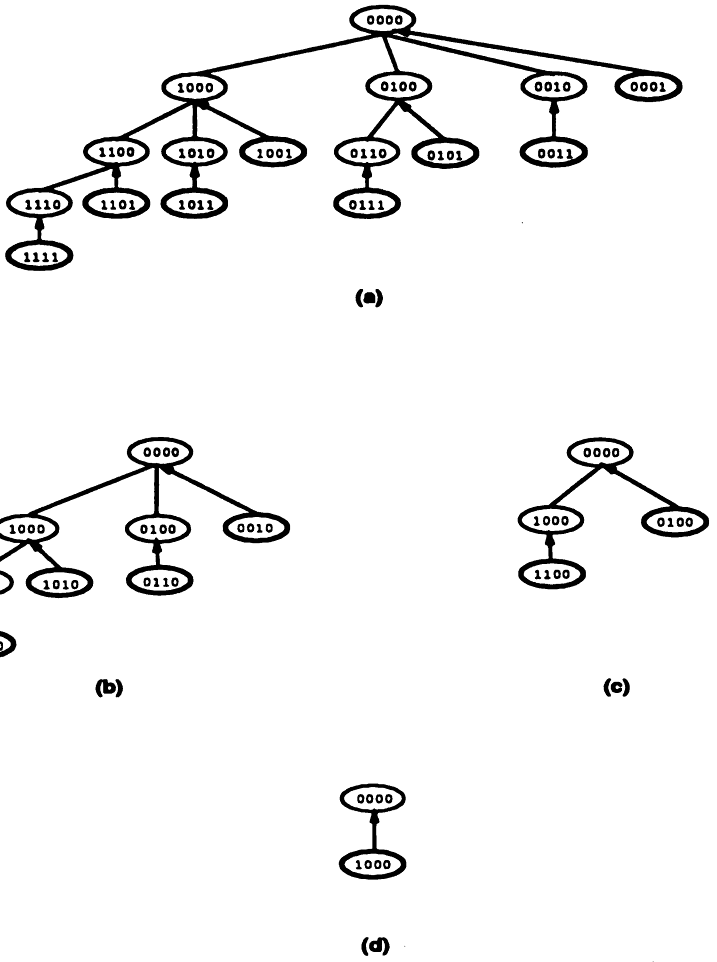

Figure 4.9. Single node gathering in B_4.

node p in $SBT_n(s)$ is represented by $C_n(p,s)$. Let u be a child of node p and j the link number of the link (p,u). The i^{th} *child* of u is obtained by flipping bit $j-i$ from u, where $1 \leq i \leq j$; the i^{th} child of the root is defined as the node obtained by flipping bit $n-i$ from the root, where $1 \leq i \leq n$. Node r is the i^{th} *right sibling* of node u in $SBT_n(s)$ if both u and r are in $C_n(p,s)$ for a node p such that $u = p \oplus \mathcal{E}_n^j$ and $r = p \oplus \mathcal{E}_n^{j-i}$ for $0 \leq j \leq n-1$ and $1 \leq i \leq j$; the 0^{th} right sibling of node u is defined to be u itself. In other words, node r is referred to as the i^{th} right sibling of node u if they have the same parent p in $SBT_n(s)$ such that u and r are obtained by flipping respectively bits j and $j-i$ from p. It is worth mentioning that for any node

I

tc

S

th

Γ_{n}^{k}

ass

to t

be m

□

Exam

and th

in the S

in SBT_5

mapped

of $SBT_5($

in $STF_8^3($

in a given SBT the number of its children equals to the number of its right siblings. The root of an SBT defines *level* 0, the children of the root define level 1, and so on.

In what follows, we will first show that $STF_n^k(s)$ is isomorphic to a *subgraph* of $SBT_{n-k}(s)$. With this important observation, we will be able to demonstrate that Algorithm 4 can be adopted to the generalized Fibonacci cubes.

Lemma 4.7 *Let s be a node in Γ_n^k. The tree $STF_n^k(s)$ is isomorphic to a subgraph of $SBT_{n-k}(s)$ for $n \geq k \geq 2$.*

Proof: We show how to map type-α nodes in $STF_n^k(s)$ to $SBT_{n-k}(s)$. Recall that Γ_n^k is a subgraph of B_{n-k}. Let us compare $SBT_{n-k}(s)$ with $STF_n^k(s)$ from level 0 to $n-k$. Clearly, level 0 of both trees are equivalent. Let q be a node in level 1 of $SBT_{n-k}(s)$, which is to be removed in Step 1 of Algorithm 3. Assume that node v is the i^{th} child of q and v is in Γ_n^k. By Lemma 4.4, the i^{th} right sibling of q, say u, is in Γ_n^k, and u is the parent of v in $STF_n^k(s)$. By Lemma 4.6, v is the only type-β node associated with u (a type-α node). Therefore, node u in $STF_n^k(s)$ should be mapped to the $i-1^{th}$ right sibling of q in $SBT_{n-k}(s)$. Consequently, descendants of u should be mapped accordingly. The nodes in level 2 through $n-k$ can be argued similarly. □

Example 4.4 We redraw $STF_8^3(01100)$ in Figure 4.10 (a), where the type-β nodes and their descendants are highlighted by dotted rectangles. Since node 11100 is not in the STF, node 00100 (the first right sibling of 11100) should be mapped to 11100 in $SBT_5(01100)$. Similarly, node 01000 (the second right sibling of 11100) should be mapped to 00100 in $SBT_5(01100)$. Hence $STF_8^3(01100)$ is isomorphic to the subgraph of $SBT_5(01100)$ shown in Figure 4.10 (b). Notice that all descendants of node 00100 in $STF_8^3(01100)$ are mapped to different nodes in $SBT_5(01100)$ except the subtree

To

lem

Len

perfo

steps

opera

Proo

show t

rooted at 10100. On the other hand, all descendants of node 01000 in $STF_8^3(01100)$ are mapped to different nodes in $SBT_5(01100)$.

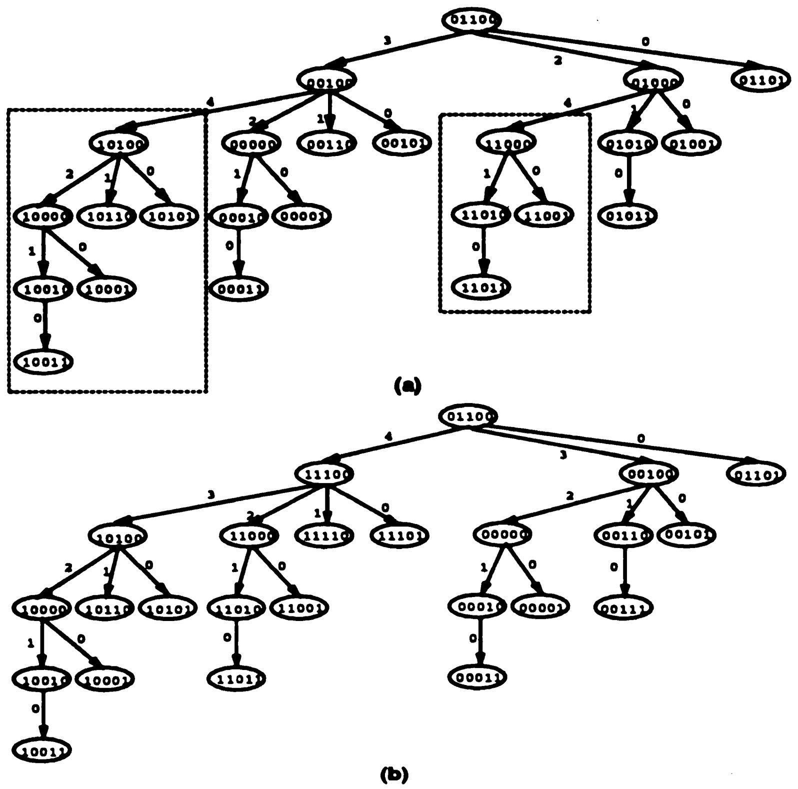

Figure 4.10. (a) $STF_8^3(01100)$ and (b) a subgraph of $SBT_5(01100)$.

To adopt Algorithm 4 to the generalized Fibonacci cubes, we need the following lemma.

Lemma 4.8 *Let s be a node in Γ_n^k. Assume that algorithm $\mathcal{A}$ can be applied to perform the gather (respectively, scatter) operation based on $SBT_{n-k}(s)$ in X time steps. Then algorithm $\mathcal{A}$ can be applied to perform the gather (respectively, scatter) operation based on $STF_n^k(s)$ in no more than X time steps.*

Proof: By Lemma 4.7, $STF_n^k(s)$ is isomorphic to a subgraph of $SBT_{n-k}(s)$. To show that algorithm $\mathcal{A}$ can be applied to the STF, we take the following approach:

T

ci

or

4.

Re

bro

ing

ing,

in *A*

broa

hype

Lem

single

Proof

the ob

time st

- Map the $STF_n^k(s)$ to the subgraph of $SBT_{n-k}(s)$ that is isomorphic to $STF_n^k(s)$, and
- Based on the SBT, apply algorithm $\mathcal{A}$ to the subgraph and take "no-operation" whenever the node(s)/link(s) that should be involved is not in the subgraph.

Clearly, it takes no more than X time steps for algorithm $\mathcal{A}$ to complete the gather (scatter) in the STF. □

Therefore Algorithm 4 can be adopted for gathering in the generalized Fibonacci cubes. We next analyze the time required by the algorithm. Again, we consider the one- and all-port models separately.

4.5.1 One-port Model

Recall that the scheduling is a critical issue for minimizing time steps in the one-port broadcasting. This is also true for the one-port gathering. Notice that scheduling is implicitly defined in Algorithm 4, since at each step an active node is sending/receiving message(s) to/from exactly one other node. Indeed, the scheduling in Algorithm 4 is the reverse of the MSLF (refer to Fig. 4.9) for the single node broadcasting. Therefore Algorithm 4 is exactly a one-port gather algorithm for the hypercube, and its complexity is as follows.

Lemma 4.9 *Under the one-port model, Algorithm 4 takes $2^n - 1$ time steps for any single node gather in B_n for $n \geq 1$, which is optimal.*

Proof: The proof of time complexity is given in [100]. The optimality follows from the observation that the root needs to collect $2^n - 1$ data, which takes at least $2^n - 1$ time steps under the one-port model. □

r

(

B

F,

lev

wil

req

we

The

in Γ

4.5.

Impro

We ex

Lemm

single r

By Lemmas 4.7, 4.8 and 4.9, one should be able to see that under the one-port model, any single node gather in Γ_n^k can be achieved in no more than $2^{n-k}-1$ time steps. On the other hand, since the node that initiates the gather operation needs to receive $F_n^{[k]}-1$ messages, a lower bound for any one-port single node gather in Γ_n^k is $F_n^{[k]}-1$. We now present a scheme for single node scatter, which attains this lower bound. The scheduling rule is:

> *The source node sends the message to the farthest node first (break ties arbitrarily) along the links in the STF.*

The above scheme will be referred to as the STF Farthest Node First (**STF-FNF**) rule. To analyze the STF-FNF scheme, let N_i denote the number of nodes with (Hamming) distance i from the root (*i.e.*, the number of nodes in level i of the STF). By the STF-FNF rule, the nodes in level 1 will be scheduled between $F_n^{[k]}-N_1$ and $F_n^{[k]}-1$, and the messages will reach them in one time step. Similarly, the nodes in level 2 will be scheduled between $F_n^{[k]}-N_1-N_2$ and $F_n^{[k]}-N_1-1$, and the messages will reach them in two time steps. In general, it can be shown that the STF-FNF requires $F_n^{[k]}-1$ time steps to complete any scatter in Γ_n^k as long as $N_i \geq 1$. Therefore we have proved the following theorem.

Theorem 4.7 *Under the one-port model, a single node scatter (respectively, gather) in Γ_n^k can be achieved in exactly $F_n^{[k]}-1$ time steps for $n > k \geq 2$, which is optimal.*

4.5.2 All-port Model

Improvements can be obtained by taking advantage of the all-port communications. We examine the case in the hypercube first.

Lemma 4.10 *Under the all-port model, Algorithm 4 takes 2^{n-1} time steps for any single node gather in B_n for $n \geq 1$.*

T

i:

In

is

fro

me

utili

Proof: Since all SBTs rooted at different nodes are isomorphic, we need only examine the case for $SBT_n(0)$. Observe that $SBT_n(0)$ can be divided into two SBTs: T_1 which consists of the node 10^{n-1} together with all its descendants, and T_0 the remainder of $SBT_n(0)$. We prove the lemma by induction on n. Clearly, Algorithm 4 takes 1 time step when $n = 1$. Assume that the lemma is true for all $n < I$, where I denotes an (arbitrary) positive integer. Let us consider the case when $n = I$. By induction hypothesis, Algorithm 4 takes 2^{I-2} time steps in both T_0 and T_1 and they can be done in parallel. A $(2^{I-2} + 2^{I-1})$ time scheduling is easy to obtain, because node 10^{n-1} can gather all 2^{I-1} messages located in T_1 in 2^{I-2} time steps then passes all of them to node 0. The key to achieve a 2^{I-1} time scheduling is that node 10^{n-1} should performs the gathering and passing messages to node 0 concurrently. Indeed it can be shown that the following scheduling rule will result in 2^{I-1} time:

> *Given a node u with all messages gathered in it, u sends the message originated from the node with the largest label first.*

□

Again, by Lemmas 4.7, 4.8 and 4.10, we have the following result.

Theorem 4.8 *Under the all-port model, any single node gather (respectively, scatter) in Γ_n^k can be achieved in no more than 2^{n-k-1} time steps for $n > k \geq 2$.*

Indeed, it can be shown that a tighter upper bound for the single node gather/scatter is $|T_s|$, where $|T_s|$ is the maximum size (number of nodes) of the subtrees branched from the root. This is because the root can follow the STF-FNF in sending the messages on all its incident links concurrently. By Lemma 4.7, $|T_s| \leq 2^{n-k-1}$ for Γ_n^k.

Remark: A way to improve the all-port single node gathering in B_n is to fully utilize all the n links incident from the root by partitioning all other nodes into

r

b

A

th

dia

a r

for

It is

to no

'Th

equivalent class based on their weights.* Based on this idea, the "perfectly balanced spanning trees" defined in [10, 9] have been proposed for all-port single node gathering in B_n. They have shown that the gathering can be achieved in $\lceil \frac{2^n-1}{\log n} \rceil$ time steps, which is optimal, because the root needs to collect $2^n - 1$ messages through its $\log n$ links. However, it can be shown that their approach do not apply to the generalized Fibonacci cubes directly, due to the "missing" nodes and links.

4.6 Multinode Broadcasting and Accumulation

Another useful data communication primitive is the *multinode broadcasting* [10, 9] (or *all-to-all broadcasting* [67]) in which every node sends a data (message) to all the other nodes. Similarly, the *multinode accumulation* is a dual operation of the multinode broadcasting. A naive approach is to broadcast the message from each node in turn by using the single node broadcasting algorithm. Trivially, the time required by this approach is $F_n^{[k]} \cdot T_b$, where T_b is the time required for the single node broadcasting algorithm.

A simple idea from [100] can make much improvement. Let (p, q) be a link in B_n. At a step i node p should send all received messages originating from s to q such that the Hamming distance between s and q is i. Since B_n is a connected graph with diameter n, it can be shown that after n steps (iterations) a message originating from a node will reach all the other nodes. Formally, a multinode broadcasting algorithm for B_n is described as follows [100].

Under the all-port model, (1) and (2) in Algorithm 5 can be performed in parallel. It is shown in [100] that at the i^{th} step, node p will send exactly $\begin{pmatrix} n-1 \\ i-1 \end{pmatrix}$ messages to node q. Therefore by summing i from 1 to n the total time required by Algorithm 5

*The weight of a binary code is defined as the number of 1's in it.

i

g

l
z

bo

to

```
Algorithm 5 All-port Multinode Broadcasting in $B_n$ :
for $i = 1$ to $n$ do:
    for every link $(p,q)$ do:
        (1)Send from $p$ to $q$ all received messages which originated
           from $s$ such that $H(s,q) = i$.
        (2)Send from $q$ to $p$ all received messages which originated
           from $s$ such that $H(s,p) = i$.
```

is 2^{n-1} time steps. By Corollary 4.1, Γ_n^k is a connected graph with diameter $n-k$. Hence it can be shown that Algorithm 5 also applies to the generalized Fibonacci cubes, and consequently the time required is as follows.

Theorem 4.9 *Under the all-port model, the multinode broadcasting (respectively, accumulation) in Γ_n^k can be achieved in no more than 2^{n-k-1} time steps for $n > k \geq 2$.*

We now focus on the one-port model. Let us consider the multinode broadcasting in a ring topology with size N, denoted by R_N; it is known [10, 67] that Algorithm 6 gives an optimal result.

```
Algorithm 6 One-port Multinode Broadcasting in $R_N$ :
Node $j$ sends its broadcasting message to node $(j+1) \bmod N$.
for $i = 2$ to $N-1$ do:
    Node $j$ sends the broadcasting message received in step $i-1$
    to node $(j+1)\bmod N$.
```

Clearly, the time required by Algorithm 6 is exactly $N-1$ time steps. A lower bound for any one-port multinode broadcasting is also $N-1$, since each node needs to receive $N-1$ messages. Therefore, we have proved the following lemma.

r

I

s

in

q)

se

no

wh

tak

tim

4.7

We h

scatte

Lemma 4.11 *Under the one-port model, the multinode broadcasting (respectively, accumulation) in a ring of size N can be achieved in $N-1$ time steps, which is optimal.*

By Theorem 3.3, the length of the longest cycle in Γ_n^k is $F_n^{[k]}$ if $F_n^{[k]}$ is even and $F_n^{[k]}-1$ if otherwise. Therefore, by embedding a longest cycle in Γ_n^k, we have the following result.

Theorem 4.10 *Under the one-port model, the multinode broadcasting (respectively, accumulation) in Γ_n^k (where $n-3 \geq k \geq 2$) can be achieved in $F_n^{[k]}$ time steps if $F_n^{[k]}$ is even and in $2F_n^{[k]}-3$ time steps if otherwise.*

Proof: The case when $F_n^{[k]}$ is even follows from Theorem 3.3 and Lemma 4.11. It remains to show the case when $F_n^{[k]}$ is odd. In this case, Γ_n^k contains a cycle of length $F_n^{[k]}-1$. Let q denote the node that is not in the longest cycle; Figure 4.11 shows a schematic diagram of the longest cycle. The multinode broadcasting can be divided into two phases. In the first phase, all nodes in the longest cycle (*i.e.*, all but node q) execute Algorithm 6 to accomplish a multinode broadcasting among them. In the second phase, (refer to Fig. 4.11) node q receives all the other $F_n^{[k]}-1$ messages from node p and sends its message to s which forwards the message to r then to t from which the message travels clockwise along the cycle until it reaches p. The first phase takes $F_n^{[k]}-2$ time steps and the second phase takes $F_n^{[k]}-1$. Therefore, the total time required is $2F_n^{[k]}-3$ steps. □

4.7 Summary and Remarks

We have studied the routing, single node broadcasting/accumulation, single node scatter/gather, and multinode broadcasting/accumulation data communication prim-

f
t
a
ir
te
is

stu

erat
diffe
or m
to al

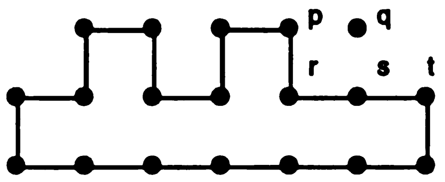

Figure 4.11. One-port multinode broadcasting through the longest cycle.

itives in the generalized Fibonacci cube. The SBTs have been widely used to implement data communication in the hypercube. Indeed, the broadcasting/accumulation and scatter/gather primitives discussed in this chapter can be efficiently achieved by employing the SBTs combined with some appropriate scheduling rules [67] [100]. It is known that the hypercube is a regular graph; all SBTs with different roots in B_n are isomorphic. This fact simplifies the analysis of the SBT-based data communication primitives, since only $SBT_n(0)$ need to be considered. Here, for the generalized Fibonacci cube, we have defined the STFs and presented the STF-based algorithms for single node broadcasting/accumulation and single node scatter/gather. Although the STFs with different roots in Γ_n^k are not isomorphic in general, we have presented an algorithm which allows us to transform an SBT to an STF. Moreover, by showing that an STF is isomorphic to a subgraph of an SBT, we have presented a new technique for studying data communication in the generalized Fibonacci cube which is irregular in general.

Table 4.1 summarizes the communication primitives and their time complexities studied in this chapter.

One communication primitive we have not yet addressed is the *total exchange* operation [9] (or *all-to-all personalized communication* [67]) in which every node sends different messages to all the other nodes. This is equivalent to the multinode scatter or multinode gather, since every node needs to receive and send different messages to all the other nodes. Clearly, the total exchange can be implemented by executing

Table 4.1. Time complexities for communication primitives in Γ_n^k.

Communication Primitive	Model	Time
single node broadcasting (accumulation)	all-port	d_s^*
	one-port	$n-k$
single node scatter (gather)	all-port	2^{n-k-1}
	one-port	$F_n^{[k]}-1$
multinode broadcasting (accumulation)	all-port	2^{n-k-1}
	one-port	$F_n^{[k]}-1$† $2F_n^{[k]}-3$‡

*The distance between source node s and all the other nodes.

† If $F_n^{[k]}$ is even.

‡ If $F_n^{[k]}$ is odd.

single node scatter (gather) by all nodes, taking turns. This results in $F_n^{[k]} \cdot T_s$ complexity, where T_s is the time required by the single node scatter (gather). Obviously improvements can be made. One approach is to invoke single node scatter (gather) concurrently on all nodes. However, this leads to a queueing delay. The analysis of this approach remains open.

a

t

d

ha

so

[67

the

dist

2^k n

[107]

CHAPTER 5

GRAPH EMBEDDING AND DATA COMMUNICATION IN INCOMPLETE HYPERCUBE

In this chapter, we will study the Hamiltonian problem and data communications in the Incomplete Hypercube. Much research has been undertaken to study the Incomplete Hypercube. Many useful properties of the Incomplete Hypercube, such as diameter and traffic density, have been shown [109] to be comparable with that of the hypercube. In [110], Tzeng et al. have shown that the binary trees and the two-dimensional meshes can also be embedded in the Incomplete Hypercube. Katseff [68] has presented algorithms for routing and single node broadcasting. Recently, more sophisticated single node broadcasting algorithms based on results for the hypercube [67] have been devised and analyzed by Tien et. al. [107]. These results suggest that the Incomplete Hypercube is an attractive candidate for interconnecting parallel and distributed systems.

Note that results reported in [107] [109] [110] are for Incomplete Hypercubes of $2^n + 2^k$ nodes, where $0 \leq k < n$. It can be seen that the "incomplete hypercubes" studied in [107] [109] [110] represent only a small subset of the general case. For example, there

i

t

o

In

Inc

con

Hyp

sing

for t

ter/g

and a

are only 10 different such incomplete hypercubes within the size between 2^{10} and 2^{11} (exclusive); on the other hand, there are a total of 2^{10} different Incomplete Hypercubes in such a range. In this chapter, we present embedding and data communication primitives for I_N where N can be *any* positive integer.

As seen in Chapter 3, embeddings of linear array and ring are essential to the embeddings of other structures such as mesh. Here we will show that the longest cycle in I_N contains N nodes if N is even, and $N-1$ nodes if otherwise. Nevertheless, I_N always contains a Hamiltonian path. Consequently, our results on embeddings of linear array and ring in the Incomplete Hypercube are both optimal in terms of minimal dilation.

A single node broadcasting algorithm for the Incomplete Hypercube has been presented in [68], which is essentially the same as that for the generalized Fibonacci cube. In [79, 82], we have presented optimal algorithms for the single node broadcasting under both the all- and one-port communication models. The all-port broadcasting algorithm is a result from [68]; the MSLF scheduling scheme was applied again to attain an optimal one-port broadcasting. Here, we further devise and analyze other communication primitives, such as accumulation, gather and scatter, for the Incomplete Hypercubes.

In this chapter, we write N (≥ 1) to denote the total number of nodes in the Incomplete Hypercube, and $n = \lceil \log N \rceil$.

The remainder of this chapter is organized as follows. Section 5.1 shows that I_N contains a Hamiltonian path and rings can be optimally embedded in the Incomplete Hypercubes. Routing and single node broadcasting are reviewed in Section 5.2. The single node broadcasting algorithms are analyzed in Section 5.3. Optimal results for the single node accumulation are reported in Section 5.4. The single node scatter/gather is studied in Section 5.5. Section 5.6 considers the multinode broadcasting and accumulation. Finally, we summarize our results in Section 5.7.

5.1 Hamiltonian Problem

In this section we study the Hamiltonian problem in the Incomplete Hypercube. We will show that I_N ($N \geq 1$) contains a Hamiltonian path. Additionally, I_N ($N \geq 4$) contains a Hamiltonian cycle if and only if N is even, whereas the longest cycle in I_N contains all but *one* node if N is odd.

The embedding of cycles in the Incomplete Hypercubes is based on Gray codes. There are many different ways to generate Gray codes. Here we adopt the known Binary Reflected Gray Code (BRGC), see *e.g.*,[25, 101], to achieve our embedding. Let G_m denote the sequence of m-bit BRGCs. It is easily seen that $G_1 = (0, 1)$. To build the 2-bit BRGCs, prefix a '0' to each code in G_1, then reverse G_1 and prefix a '1' to each code. In other words, we get $G_2 = (00, 01, 11, 10)$. More generally, let $\overline{G}_i$ denote the sequence of codes obtained from G_i by reversing its order, and $0G_i$ (respectively, $1G_i$) the sequence of codes obtained from G_i by prefixing a '0' (respectively, '1') to each element of the sequence G_i. The m-bit BRGCs can be generated by the recursion [101]:

$$G_m = (0G_{m-1}, 1\overline{G}_{m-1})$$

Table 5.1 shows the 5-bit BRGCs, where $G_5(i)$ is the i^{th} ($0 \leq i \leq 31$) BRGC in the sequence G_5. Let $C_5(i)$ denote the (regular) 5-bit binary code for integer i. Table 5.1 also lists the integer j such that $C_5(j) = G_5(i)$ for each i and j.
The following observation is important:

Lemma 5.1 *The binary codes $C_m(i)$ and $C_m(i+1)$ differ in bit 0 if and only if i is even for $0 \leq i \leq 2^m - 1$.*

Consequently, $C_m(i)$ and $C_m(i+1)$ are two consecutive BRGCs, although $C_m(i)$ may come before or after $C_m(i+1)$ in the sequence G_m.

Table 5.1. 5-bit Binary Reflected Gray Codes.

i	$G_5(i)$	j	i	$G_5(i)$	j	i	$G_5(i)$	j	i	$G_5(i)$	j
0	00000	0	8	01100	12	16	11000	24	24	10100	20
1	00001	1	9	01101	13	17	11001	25	25	10101	21
2	00011	3	10	01111	15	18	11011	27	26	10111	23
3	00010	2	11	01110	14	19	11010	26	27	10110	22
4	00110	6	12	01010	10	20	11110	30	28	10010	18
5	00111	7	13	01011	11	21	11111	31	29	10011	19
6	00101	5	14	01001	9	22	11101	29	30	10001	17
7	00100	4	15	01000	8	23	11100	28	31	10000	16

5.1.1 Cycles

Our embedding of ring is based on the longest cycle in the Incomplete Hypercube. We first consider cycles in the Incomplete Hypercube.

Theorem 5.1 *A cycle of length l can be embedded in I_N, where l is even and $4 \leq l \leq N$.*

Proof: Since the result for the (complete) hypercubes is known [101], we assume that N is not a power of 2. Notice that I_N contains a hypercube B_{n-1}. Hence according to [101] cycles of length l can be embedded in B_{n-1} for $4 \leq l \leq 2^{n-1}$. It remains to show that the theorem is true for $2^{n-1} < l \leq N$. We distinguish between the following two cases:

Case 1: $2^{n-1} + 1 \leq N \leq 3 \cdot 2^{n-2}$.

Define $M = 3 \cdot 2^{n-2} - N$. Let $L_0 = 00G_{n-2}$, $L_1 = 01G_{n-2}$ and $L_2 = 10(G_{n-2} \setminus M)$, where $(G_{n-2} \setminus M)$ denotes the sequence of codes resulted from the removal of the M $C_{n-2}(j)'s$ from G_{n-2} for $2^{n-2} - M \leq j < 2^{n-2}$. Figure 5.1(a) shows a schema, where by selecting nodes alternatively between L_0 and L_2, a cycle of length 22 can be embedded in I_{22}. Indeed, define two nodes j and $j+1$ to be a pair, where j is an even number and $2^{n-1} < j < N$. By Lemma 5.1, $C_n(j)$ and $C_n(j+1)$ are two consecutive

BRGCs. Hence a pair of "up" and "down" edges can be used to include the pair j and $j+1$ in the L_2 to form a cycle. Consequently, every node in L_2 can be included to form a cycle if it is paired with another node.

Case 2: $3 \cdot 2^{n-2} + 1 \leq N \leq 2^n - 1$.

Define $M = 2^n - N$. Let $L_0 = 00G_{n-2}$, $L_1 = 01G_{n-2}$, $L_2 = 10G_{n-2}$ and $L_3 = 11(G_{n-2} \setminus M)$, where $(G_{n-2} \setminus M)$ is defined the same way as in case 1. Figure 5.1(b) shows that by selecting nodes alternatively between L_0 and L_2, and nodes alternatively between L_1 and L_3, a cycle of length 28 can be embedded in I_{28}. Again, only an even number of nodes in L_3 can be included in a cycle as we observed in case 1. □

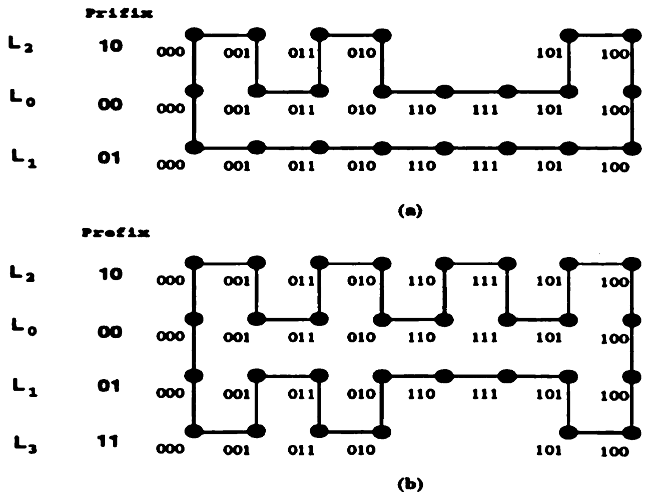

Figure 5.1. Embedding of longest cycles in (a) I_{22}, and in (b) I_{30}.

We next study cycles of odd lengths. By applying the same schema described in the previous proof, Figure 5.2 shows an embedding of a cycle with length 22 in I_{23}. Note that node 10110 (22) is excluded from the cycle, since node 23 is not in I_{23}. Recall the fact that there is no cycle of odd length in the hypercube (Lemma 3.3).

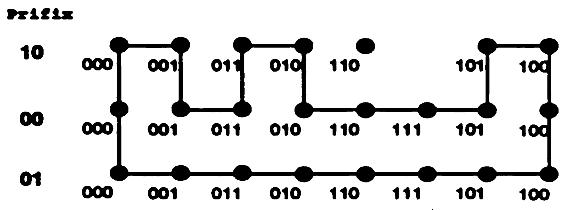

Figure 5.2. Embedding of a cycle with length 22 in I_{23}.

Furthermore, Lemma 2.2 shows that I_N is a subgraph of B_n. Hence we have

Lemma 5.2 *There is no cycle of odd length in the Incomplete Hypercube.*

Consequently, we have proved the following results:

Theorem 5.2 *For $N \geq 4$, I_N contains a Hamiltonian cycle if and only if N is even, whereas the length of the longest cycle in I_N is $N-1$ if and only if N is odd.*

Corollary 5.1 *A ring of size N can be embedded in I_N with dilation one if N is even, with dilation two if otherwise.*

Proof: It is clear when N is even. For N being odd, the only node that cannot be included in the longest cycle has at least one adjacent node in the cycle. Hence the embedding can be achieved by paying a higher price (*i.e.*, dilation is 2). □

5.1.2 Hamiltonian Path

We now focus on the embedding of linear array. By Theorem 5.1, the graph I_N contains a Hamiltonian path when N is *even*. Figure 5.3 shows that by replacing edge (00110,00111) with edge (10110,00110), I_{23} contains a Hamiltonian path. More generally, we will prove the following result:

Theorem 5.3 *I_N contains a Hamiltonian path.*

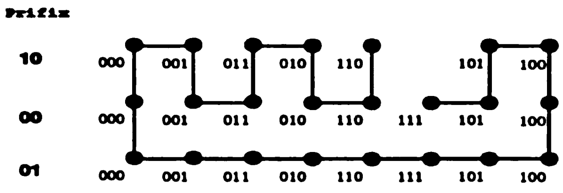

Figure 5.3. Embedding of a Hamiltonian path in I_{23}.

Proof: It is easy to verify for $N \leq 4$. For $N \geq 5$, we distinguish between the following two cases:

Case 1: N is even.

In this case, I_N contains a Hamiltonian cycle according to Theorem 5.2. By removing exactly *one* edge from the cycle, we have a Hamiltonian path.

Case 2: N is odd.

Recall that nodes are labeled between 0 and $N-1$. Based on the schema presented in the proof of Theorem 5.1, we are able to embed a cycle, LC, in I_N, which contains all but node $N-1$. Now, consider the two nodes $N-1-2^{n-1}$ and $N-2^{n-1}$. Observe that the number $N-1-2^{n-1}$ is even and hence $N-2^{n-1}$ is odd. As a consequence of Lemma 5.1, the edge $(N-1-2^{n-1}, N-2^{n-1})$ is in LC. Furthermore, the edge $(N-1, N-1-2^{n-1})$ is also in I_N. The path formed by the removal of the edge $(N-1-2^{n-1}, N-2^{n-1})$ and the addition of the edge $(N-1, N-1-2^{n-1})$ to LC is exactly a Hamiltonian path. □

5.2 Routing and Single Node Broadcasting

In this section, we review the routing and the single node broadcasting algorithms proposed for the Incomplete Hypercubes. The routing problem in the Incomplete Hypercube was first studied by Katseff [68]. Indeed, Algorithm 1 for routing in the generalized Fibonacci cubes is essentially the same as that proposed in [68]. Figure 5.4 shows the structure of I_6 and the routing path (with arrow in each link) from node 011 to 100, which is determined by Algorithm 1. Lemma 5.3 is a direct result from

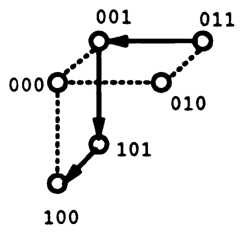

Figure 5.4. Routing path from node 011 to 100 in I_6.

[68], which says that Algorithm 1 always finds a shortest path between any two nodes.

Lemma 5.3 *Let* s *and* d *be two distinct nodes in I_N, and H(s,d) the Hamming distance between these two nodes. Algorithm 1 always finds a shortest path of length H(s,d) from* s *to* d.

It is also shown in [68] that the routing paths determined by Algorithm 1 are deadlock-free. Given any two nodes in I_N, by Lemma 5.3 there is a path between them. Hence I_N is a connected graph. Furthermore, the Hamming distance between the two nodes 2^{n-1} (with address $100\cdots0$) and $2^{n-1}-1$ (with address $011\cdots1$) in I_N is exactly n. Therefore we have proved the following result.

Lemma 5.4 *I_N is a connected graph and its diameter is n.*

Katseff [68] has also proposed an algorithm for single node broadcasting in the Incomplete Hypercube, which is essentially the same as Algorithm 2 presented in Section 4.2. Figure 5.5 illustrates an example of a single node broadcasting from node 0101 in I_{11}, when Algorithm 2 is applied. Notice that each node is represented by an ellipse labeled with its address; only the links determined by Algorithm 2 are shown in the figure. The travel array received at each node is shown above it, whereas the source node initializes the array to all 1's (TRUE).

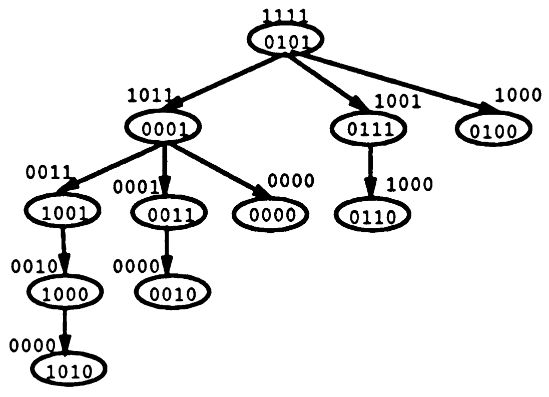

Figure 5.5. A broadcasting from node 0101 in I_{11}.

It is known [68] that the set of nodes and links traversed by executing Algorithm 2 for any single node broadcasting in I_N form a spanning tree, and the spanning tree will be referred to as STI. We write $STI_N(r)$ to denote the STI in I_N rooted with node r. Figure 5.5 also shows the $STI_{11}(0101)$. Observe that two STIs rooted at different nodes are generally not isomorphic, which is similar to what we saw in the case of the STFs. The next lemma is a result from [68], which shows that the single node broadcasting algorithm sends the message through the same path as determined by the routing algorithm.

Lemma 5.5 *Let* s *and* t *be two distinct nodes in I_N. Let* p *be the path determined by Algorithm 1 to route a message from* s *to* t. *If a message is broadcast from* s *by Algorithm 2, then it will reach* t *along a unique path that coincides with* p.

5.3 Analysis of Single Node Broadcasting

In this section, we analyze the single node broadcasting. We will follow the assumptions made in Chapter 4, *i.e.*, all messages are of unit length and each takes a unit time for transmission between two connected nodes.

5.3.1 All-port Model

Theorem 5.4 shows that Algorithm 1 is optimal under the all-port model.

Theorem 5.4 *Let d_s be the maximal distance between an (arbitrary) node s and all the other nodes in I_N. Then, under the all-port communication model, Algorithm 2 can optimally complete a single node broadcast from node s in exactly d_s time steps.*

Proof: It follows from Lemmas 5.3 and 5.5. □

Notice that d_s is n if both nodes s and $2^n - s - 1$ (*i.e.*, $\overline{s}$) exit in I_N, and $n - 1$ if otherwise.

5.3.2 One-port Model

As we saw in the case of the generalized Fibonacci cubes, the scheduling scheme makes a critical difference in the overall broadcasting time. Recall that the MSLF scheme has been proposed for one-port single node. In what follows, we will show that the *same* one-port single node broadcasting algorithm presented in Section 4.3.2 is optimal when applied to the Incomplete Hypercube.

To motivate, we revisit the broadcast from node 0101 in I_{11}. Figure 5.6 shows that the one-port broadcasting requires 4 time steps, where each link is labeled by the time step at which the link is involved in forwarding the message. Notice that the distance between the nodes 0101 and 1010 is exactly 4, which is a lower bound for such a broadcasting. Hence the one-port algorithm is optimal in this case. We

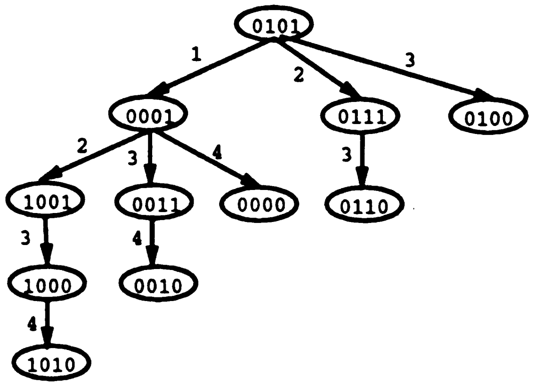

Figure 5.6. One-port broadcasting from node 0101 in I_{11}.

next show that the one-port broadcasting algorithm takes exactly n time steps for any single node broadcasting in I_N. It is necessary to examine closely the relation between the STIs and the SBTs at this point.

First we try to apply Algorithm 3 to construct STIs. Figure 5.7 illustrates a

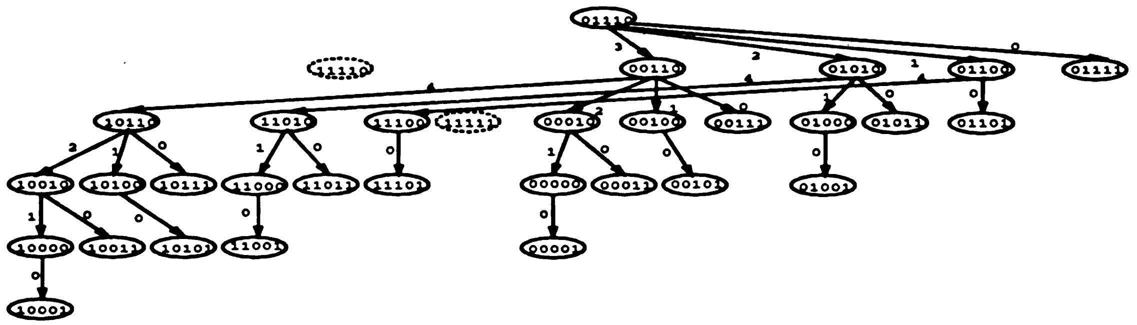

Figure 5.7. Construction of $STI_{30}(01110)$ from $SBT_5(01110)$.

construction of $STI_{30}(01110)$ from $SBT_5(01110)$. It is seen that the parents of nodes 10110, 11010 and 11100 are switched respectively to nodes 00110, 01010 and 01100, since node 11110 is not in I_{30}. However, Algorithm 3 fails to give a construction of $STI_{22}(01110)$, because both nodes 11110 and 10110 are not in I_{22} whereas node 10010 (a child of 10110) is in I_{22} (refer to Fig. 5.7). Further effort leads to a construction of the $STI_{22}(01110)$ from the $SBT_5(01110)$ as illustrated in Figure 5.8. Notice that the

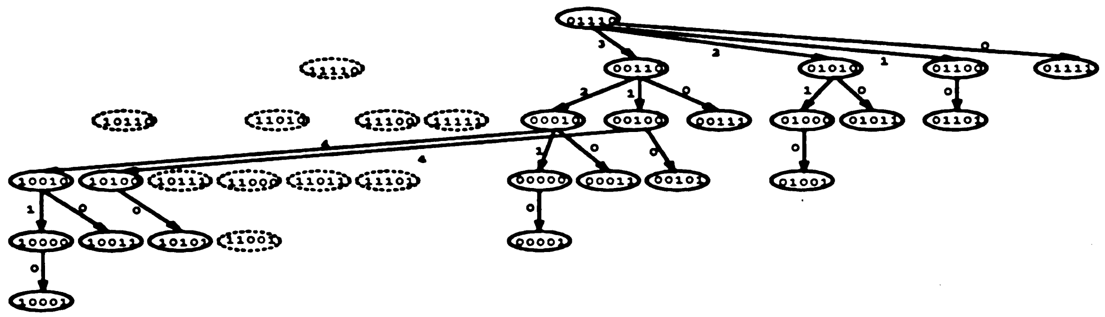

Figure 5.8. Construction of $STI_{22}(01110)$ from $SBT_5(01110)$.

link sequences (4,3,2) and (3,2,4) connect node 01110 to node 10010 in $SBT_5(01110)$ and $STI_{22}(01110)$, respectively. Similarly, the link sequences (4,3,1) and (3,1,4) connect node 01110 to node 10100 in the SBT and the STI, respectively. This observation leads to the following lemma which extends Lemma 4.4.

Lemma 5.6 *Let u_0 be a node in I_N. Let $(u_0, u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{i+j-1}, u_{i+j})$ be the path in $SBT_n(u_0)$ connecting node u_0 to u_{i+j}, and $(l_1, l_2, \cdots, l_i, l_{i+1}, l_{i+2}, \cdots, l_{i+j})$ the corresponding link sequence. Assume that u_{i+j} is in I_N and u_{i+j-1} is not in I_N. Then*

1. *There exists a node u_i that is in I_N whereas all $u_{i+1}, u_{i+2}, \cdots, u_{i+j-1}$ are not in I_N,*

2. *Within B_n, the link sequence $(l_{i+2}, l_{i+3}, \cdots, l_{i+j})$ from u_i leads to a node u that is in I_N, and*

3. *Within B_n, link l_{i+1} from u leads to u_{i+j}.*

Proof: By back tracking from node u_{i+j} in $SBT_n(u_0)$ to the root u_0, one should be able to see that there exists a node u_i that is in I_N. This is true because at least u_0 is in I_N. (Note that u_i may be the same node as u_0.)

Next we prove that u is in I_N. The following two observations are crucial:

1. u_{i+j} and u differ in exactly bit l_{i+1}, and

2. (Lemma 5.8) bit l_{i+1} of u is 0 and bit l_{i+1} of u_{i+j} is 1

Since u_{i+j} is in I_N, node u must be in I_N. This is because, by observation 2, the (integer) label of u is less than that of u_{i+j}.

Finally, the edge (u, u_{i+j}) in B_n (or, more precisely, I_N) is because both nodes are in I_N and they differ in exactly bit l_{i+1} as we just observed. □

As in the case of the generalized Fibonacci cubes, links in $STI_N(s)$ can also be classified into two sets: set A includes links which are not in $SBT_n(s)$, and set B the remaining links. For example, links (00010,10010) and (00110,00010) in $STI_{22}(01110)$ are in set A and B respectively (refer to Fig. 5.8). Links in set A are resulted from some "missing" nodes (relative to B_n). Similarly, a link l in set A connects two special nodes: the one defining l as an out-edge is called type-α node and the other is type-β. The following lemma shows that each type-α node has exactly one type-β node as its child.

Lemma 5.7 *Let u be a type-α node in $STI_N(s)$. Then u has exactly one type-β node as its child in $STI_N(s)$.*

We need the following crucial observation before proving Lemma 5.7.

Lemma 5.8 *Let (q, r) be a set A link in $STI_N(s)$ such that i is its link number and q is a type-α node. Then bit i of q is 0.*

Proof: Note that q must be an internal node in an STI, because no links in set A could incident from a root node. Assume that p is the parent of q in $STI_n(s)$ and the link number of (p, q) is j. By definition $i > j$. We prove by contradiction that bit i of q is 0. Suppose that bit i of q is 1, then bit i of p is also 1. Notice that p and r differ in exactly bits i and j. Define $t = p - 2^i$. Clearly, node t is in I_N. According to the MSLF scheme, at node p the one-port algorithm would forward the message through link i to node t which, however, would forward the message to r. This is a contradiction. □

Now, we proceed to prove Lemma 5.7.

Proof: Observe that u must be an internal node in $STI_N(s)$. Let p be the parent of u and k be the link number of link (p, u). We prove the lemma by contradiction. Suppose u has two children v and w and the link numbers for (u, v) and (u, w) are i and j, respectively. Without loss of generality, assume that $i = 2, j = 1$ and $k = 0$. By Lemma 5.8, both bits 2 and 1 of u (and hence p) are 0. Consider the following two cases: (a) bit 0 of u is 1, and (b) bit 0 of u is 0 as displayed in Figure 5.9. Observe that N must be at least 6 and 5 in case a and case b, respectively. By the one-port algorithm, the broadcasting path from p to v would be (000,100,101) in case a. Similarly, the broadcasting path from p to w would be (001,011,010) in case b. Both are contradictions. Hence we have proved the lemma. □

By Lemmas 5.6 and 5.7, one should be able to design an algorithm similar to Algorithm 3, which allows us to transform an SBT to an STI. Hence we have the time required by the one-port broadcasting algorithm.

Theorem 5.5 *The one-port single node broadcasting algorithm requires at most n time steps to broadcast one message from any node in I_N.*

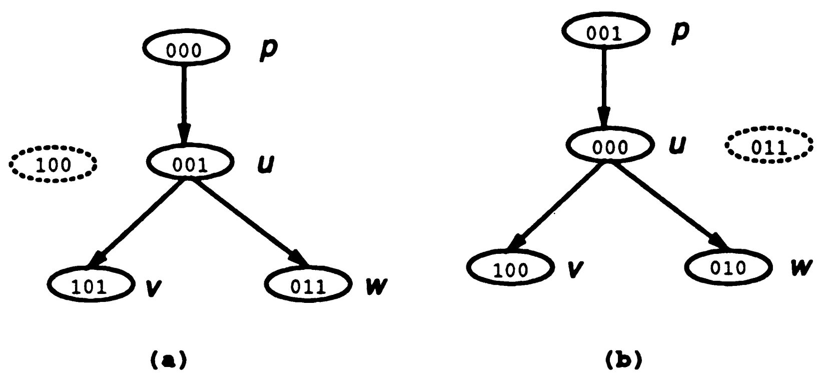

Figure 5.9. Type-α node and its type-β child.

Proof: (Sketch) Consider the scheduling in type-α nodes. Let $(u_0, u_1, \cdots, u_{i-1}, u_i, u_{i+1}, \cdots, u_{i+j-1}, u_{i+j})$ be a path and u be the node as specified in Lemma 5.6, *i.e.*, link (u, u_{i+j}) is in set A, and u (respectively, u_{i+j}) is a type-α (respectively, type-β) node. Within B_n, the broadcasting message will reach u_{i+j} at step $n - l_{i+j}$. On the other hand, it can be shown that the broadcast message will reach node u in no more than $n - l_{i+j} - 1$ time steps. By Lemma 5.7, there is exactly one set A link originated from u. Under the MSLF scheme, u will forward the message through the set A link first, then through set B link(s). Therefore, the message will reach u_{i+j} (a type-β node) no later than $n - l_{i+j}$ steps, *i.e.*, at a time no later than that resulted from the one-port broadcasting in B_n. For nodes that are not type-α, it can be shown that the scheduling on them will not be delayed by the type-α nodes. □

Remark: Let $(u_0, u_1, \cdots, u_i)$ be a path in $STI_N(u_0)$. Figure 5.10 shows that there may be two type-α nodes in such a path, *i.e.*, nodes 0001 and 1000. (In general, there may be more than two type-α nodes in a path.) Since it takes only 1 (which is 1 time step earlier compared to the scheduling in $SBT_4(1010)$) time step for the broadcasting message to reach node 0001, one should be able to see the scheduling in node 1000 will not be affected (delayed) by a type-α ancestor. This is similar to what we saw in the case of the generalized Fibonacci cubes.

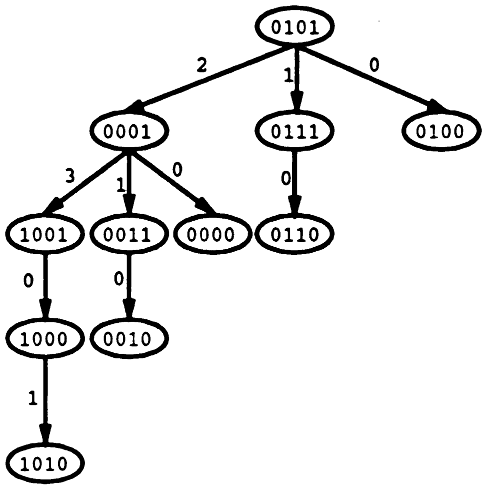

Figure 5.10. $STI_{11}(1010)$.

Corollary 5.2 *The one-port broadcasting algorithm is optimal in terms of minimal time steps.*

Proof: It suffices to show that n is a lower bound for any one-port broadcasting algorithm. To see this, under the one-port model the total number of nodes receiving broadcast message is at most doubled for every consecutive time step. Hence it takes at least n steps for all N nodes to receive the broadcast message. $\square$

We now show that accumulation, gather and scatter operations can also be efficiently implemented in the Incomplete Hypercube. The approaches taken here are similar to what we did in Chapter 4.

5.4 Single Node Accumulation

As we saw in Section 4.4, the accumulation is a dual operation of the broadcasting. Therefore, the times required for the all- and one-port single node accumulation in the Incomplete Hypercube are captured by Theorems 5.4 and 5.5, respectively.

5.5 Single Node Gather and Scatter

Recall that the STF-FNF scheme for the single node scatter in the generalized Fibonacci cubes. Observe that all paths in the STIs are shortest-path, which is a key factor to the efficiency of the STF-FNF. Along the same line, we propose the following scheduling rule:

> *The source sends the message to the farthest node first (break ties arbitrarily) along the links in the STI.*

for scatter operation in the Incomplete Hypercube. The above scheme will be referred to as STI Farthest Node First (STI-FNF), and the time required by the STI-FNF is:

Theorem 5.6 *Under the one-port model, any single node scatter (respectively, gather) in I_N (where $N \geq 2$) can be achieved in exactly $N-1$ time steps, which is optimal.*

Proof: The proof for time requirement is similar to the case of the generalized Fibonacci cubes. To see the optimality, observe that the root needs to send $N-1$ messages. □

The following two lemmas will facilitate our analysis of the all-port single node gather/scatter.

Lemma 5.9 *Let s be a node in I_N. The tree $STI_N(s)$ is isomorphic to a subgraph of $SBT_n(s)$.*

Proof: Recall that in the case of the generalized Fibonacci cubes, Lemmas 4.4 and 4.6 are essential to the proof of Lemma 4.7. Lemmas 5.6 and 5.7 show that the Incomplete Hypercube possesses similar properties. We omit the detailed proof. □

Figure 5.11 shows that $STI_{22}(01110)$ is isomorphic to a subgraph of $SBT_5(01110)$. It can be seen that node 00110 in the STI should be mapped to 11110 in the corresponding SBT. Consequently, node 00010 (respectively, 00100 and 00111) should be mapped to 10110 (respectively, 11010 and 11100) etc.

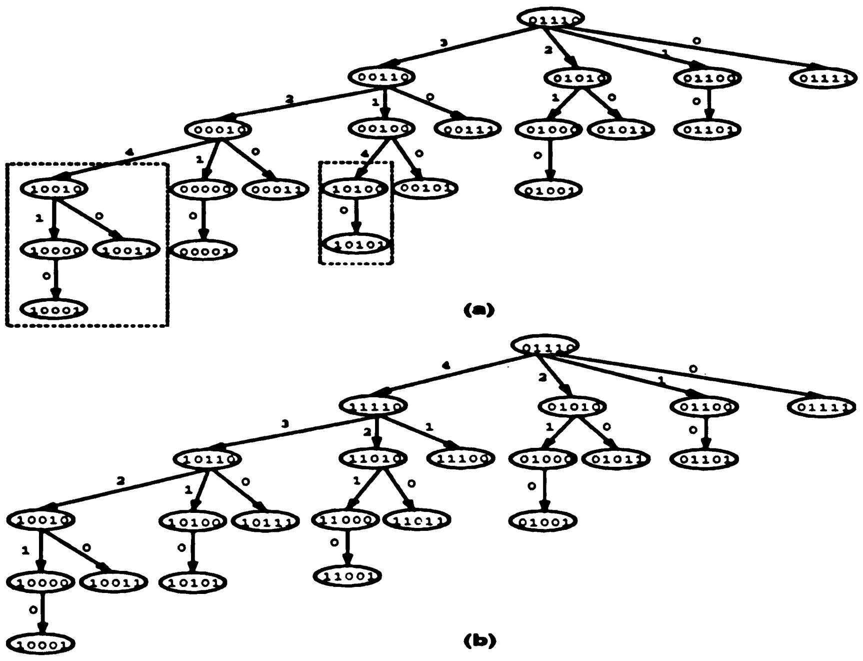

Figure 5.11. (a) $STI_{22}(01110)$ and (b) a subgraph of $SBT_5(01110)$.

Lemma 5.10 *Let s be a node in I_N. Assume that algorithm $\mathcal{A}$ can be applied to perform gather (respectively, scatter) operation based on $SBT_n(s)$ in X time steps. Then algorithm $\mathcal{A}$ can be applied to perform gather (respectively, scatter) operation based on $STI_N(s)$ in no more than X time steps.*

Proof: Similar to the proof of Lemma 4.8. Omitted. □

Therefore, by Lemmas 4.10, 5.9 and 5.10, we have the following result for the all-port model.

Theorem 5.7 *Under the all-port model, any single node gather (respectively, scatter) in I_N can be achieved in no more than 2^{n-1} time steps for $N \geq 2$.*

Again, it can be shown that a tighter upper bound is $|T_s|$, where $|T_s|$ ($\leq 2^{n-1}$) is the maximum size of the subtrees branched from the root.

5.6 Multinode Broadcasting and Accumulation

Recall that Algorithm 5 has been proposed for the multinode broadcasting/accumulation in the generalized Fibonacci cubes. By Lemma 5.4, I_N is a connected graph with diameter n. Hence it can be shown that Algorithm 5 also applies to the Incomplete Hypercube. Consequently, the time required is:

Theorem 5.8 *Under the all-port model, the multinode broadcasting (respectively, accumulation) in I_N can be achieved in no more than 2^{n-1} time steps for $N \geq 2$.*

By Theorem 5.1, the length of the longest cycle in I_N is N if N is even and $N-1$ if otherwise. Again, by embedding a longest cycle in I_N, we have the following result.

Theorem 5.9 *Under the one-port model, the multinode broadcasting (respectively, accumulation) in I_N (where $N \geq 4$) can be achieved in $N-1$ time steps if N is even and in $2N-3$ time steps if otherwise.*

Proof: Similar to the proof of Theorem 4.10. Omitted. □

5.7 Summary of Results

We have studied the Hamiltonian problem and some data communication primitives in the Incomplete Hypercube. For the Hamiltonian problem the following optimal results have been obtained:

1. I_N contains a Hamiltonian path. Hence a linear array of size N can be embedded with dilation-1, and

2. The longest cycle in I_N (where $N \geq 4$) contains N nodes if N is even, and $N-1$ nodes if otherwise. Consequently, a ring of size N can be embedded with dilation-1 if N is even, with dilation-2 if otherwise.

For data communication, we have studied the single node broadcasting/accumulation, single node scatter/gather, and multinode broadcasting/accumulation primitives in the Incomplete Hypercube. We have defined the STIs and presented the STI-based algorithms for single node broadcasting/accumulation and single node scatter/gather in the Incomplete Hypercube. The STIs with different roots in I_N are not isomorphic in general. Again, by showing that an STI is isomorphic to a subgraph of an SBT, we have presented a new technique for studying data communication in the Incomplete Hypercube.

Table 5.2 summarizes the communication primitives and their time complexities studied in this chapter.

Table 5.2. Time complexities for communication primitives in I_N.

Communication Primitive	Model	Time
single node broadcasting (accumulation)	all-port	d_s^*
	one-port	n
single node scatter (gather)	all-port	2^{n-1}
	one-port	$N-1$
multinode broadcasting (accumulation)	all-port	2^{n-1}
	one-port	$N-1$[†] $2N-3$[‡]

* The distance between the source node s and all the other nodes.

† If N is even.

‡ If N is odd.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this chapter, we summarize our main results, highlight our contributions, and discuss some directions for future work.

6.1 Main Results and Contributions

We have studied a large family of irregular network topologies that consists of the generalized Fibonacci cube and the Incomplete Hypercube. Each member of the family is shown to be a subgraph of a hypercube. Additionally, both the generalized Fibonacci cube and the Incomplete Hypercube include the hypercube as a special case. The Incomplete Hypercube offers unlimited sizes; however, it suffers from a low degree of fault-tolerance under certain situations. On the other hand, the generalized Fibonacci cube offers a guaranteed degree of fault-tolerance.

To investigate their applications in parallel or distributed systems, we have presented a collection of efficient graph embedding and data communication algorithms for this large family of network topologies. In particular, we have obtained the following graph embedding results for the generalized Fibonacci cube:

1. Γ_n^k (where $n \geq k \geq 2$) contains a Hamiltonian path, hence a linear array.

2. If $F_n^{[k]}$ is even, the longest cycle in Γ_n^k(where $n-3 \geq k \geq 2$) contains all nodes, which implies that Γ_n^k embeds a ring of size $F_n^{[k]}$ with dilation-1. On the other hand, if $F_n^{[k]}$ is odd, the longest cycle in Γ_n^k contains all but one node, which implies that Γ_n^k embeds a ring of size $F_n^{[k]}$ with dilation-2.

3. We have shown that some 2D and 3D meshes of certain sizes can be embedded in the generalized Fibonacci cube with expansion-1 and dilation-1.

4. We have shown that a hypercube can be embedded in Γ_n^k.

The Hamiltonian problem for the Incomplete Hypercube has also been studied. We have shown that I_N contains a Hamiltonian path, and the length of the longest cycle in I_N (where $N \geq 4$) is N if N is even, and $N-1$ if otherwise.

For data communication in both the generalized Fibonacci cube and the Incomplete Hypercube, we have devised efficient algorithms for the following primitives:

1. Single node broadcasting/accumulation,

2. Single node scatter/gather, and

3. Multinode broadcasting/accumulation.

All of our algorithms have been carefully analyzed under both the one- and all-port models.

The family of network topologies studied in this dissertation has the following two areas of application.

1. *It provides a spectrum of alternative structures for fault-tolerant computing in the hypercube systems.* This is because each member of the family (a generalized Fibonacci cube or Incomplete Hypercube) is a subgraph of a hypercube.

2. ***It makes small-sized incremental expansion possible.*** **This is a very desirable property when constructing parallel or distributed systems that evolve with time.**

We consider the following as our contributions.

1. **Most existing results of graph embedding and data communication are only applicable to a single kind of regular networks. Here we have presented new results that can be applied to a large family of irregular network topologies.**

2. **By using a few parameters, (*e.g.*, *n* and *k* for the generalized Fibonacci cube, and *N* for the Incomplete Hypercube) to characterize members of the family, we have been able to described the relations between the structural properties and the algorithmic efficiency of a network. Our approach has provided a new framework for studying networks.**

6.2 Future Work

Obviously, there are many important issues that have not been addressed. The following is a list of possible directions for future work.

1. **Graph embedding:** We have identified the following outstanding problems:

   - *Embedding in hypercube:* As we mentioned, this family of network topologies studied can serve as a fault-tolerant structure for the hypercube. However, we have not addressed the issue of how to efficiently embed a generalized Fibonacci cube or an Incomplete Hypercube in a faulty hypercube.

   - *Subcube allocation:* It has been widely anticipated that future parallel systems will consist of thousands or even millions nodes. Since a typical user may not require all the processors, Hayes et al. [55] and Peterson et al. [95]

have suggested that the hypercube systems should support multiprogramming to promote system utilization. The problem of subcube allocation in the hypercube systems has been studied extensively [3] [22] [25] [28] [38] [70]. Similarly, the problem of submesh allocation in the 2D mesh systems has also drawn much attention [76] [77] [85]. Since the generalized Fibonacci cube can also be decomposed into subcubes, we expect that the subcube allocation in the generalized Fibonacci cube can be tackled in a flexible fashion.

- *Optimal embedding of meshes:* We have shown that some 2D and 3D meshes can be embedded in the generalized Fibonacci cube with expansion-1 and dilation-1. Given a 2D or 3D mesh of certain size, how can we achieve a minimal expansion and/or minimal dilation embedding?

2. **Structural properties:** Our work is just a beginning. To have more complete knowledge of the family of networks, we need to investigate other topological properties such as bisection width, node/edge connectivities, independent paths, and the relation between the generalized Fibonacci cube and Incomplete Hypercube.

3. **Application algorithms:** A network topology is never useful if it does not support efficient algorithms. We have presented many tools for efficient graph embedding and data communication for the family of network topologies. Recently, we have shown that prefix computation can be done efficiently in the generalized Fibonacci cube [64]. We would like to investigate along the same line.

4. **Optimization**: We have studied a spectrum of new topologies, which supports different performance/structural features. Given a specification of desirable features, which may consist of a list of requirements (*e.g.*, number of nodes,

edges, degree, and performance features), how can we select member(s) from this family of networks that will satisfy the requirements with a minimal cost?

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] ABRAHAMSON, K., DADOUN, N., KIRKPATRICK, D. G., AND PRZYTYCKA, T. A simple parallel tree contraction algorithms. *Journal of Algorithms 10* (1989), 287–302.

[2] AKL, S. G. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, 1989.

[3] AL-DHELAAN, A., AND BOSE, B. A new strategy for processors allocation in an n-cube multiprocessor. In *Proc. IEEE the* 8^{th} *Annual Int'l Phoenix Conf. on Comput. Comm.* (1989), pp. 114–118.

[4] ANDERSON, E., AND ET. AL. LAPACK: a portable linear algebra library for high-performance computers. In *Proc. of Supercomputing Conference* (1990), pp. 2–11.

[5] BANERJEE, P. Strategies for reconfiguring hypercubes under faults. In *IEEE* 20^{th} *Intn'l Conf. on Fault-Tolerant Computing* (1990), pp. 210–217.

[6] BARNES, G. H., BROWN, R. M., KATO, M., KUCK, D. J., SLOTNICK, D. L., AND STOKES, R. A. The Illiac IV computers. *IEEE Trans. Computers C-27* (1968), 84–87.

[7] BATCHER, K. E. Sorting networks and their applications. In *Proc. of the Spring Joint Computer Conference* (1968), AFIPS Press, pp. 307–314.

[8] BECKER, B., AND SIMON, H.-U. How robust is the n-cube. *Information and Computation 77* (1988), 162–178.

[9] BERTSEKAS, D. P., OZVEREN, C., STAMOULIS, G. D., TSENG, P., AND TSITSIKLIS, J. N. Optimal communication algorithm for hypercubes. *Journal of Parallel and Distributed Computing 11* (1991), 263–275.

[10] BERTSEKAS, D. P., AND TSITSIKLIS, J. N. *Parallel and Distributed Computation.* Prentice Hall, 1989.

[11] BHUYAN, L. N., AND AGRAWAL, D. P. Generalized hypercube and hyperbus structures for a computer network. *IEEE Trans. Computers C-33*, 4 (Apr. 1984), 323–333.

[12] BLELLOCH, G. E. Scan as primitive parallel operations. *IEEE Trans. Computers 38*, 11 (Nov. 1989), 1526–1538.

[13] BRAATEN, M. E. Solution of viscous fluid flows on a distributed memory concurrent computer. In *Proc. Intn'l Conf. on Parallel Processing* (1988), pp. 243–250.

[14] BRENT, R. P. The parallel evaluation of general arithmetic expressions. *Journal of the ACM 21*, 2 (Apr. 1974), 201–206.

[15] BROWN, JR., J. L. Zeckendorf's theorem and some applications. *Fibonacci Quarterly 2*, 3 (1964), 163–168.

[16] BRUCK, J., CYPHER, R., AND SOROKER, D. Running algorithms efficiently on faulty hypercubes. In *ACM 2^{nd} Symposium on Parallel Algorithms and Architectures* (1990), ACM, pp. 37–44.

[17] BRUCKMAN, P. S. The generalized Zeckendorf theorems. *Fibonacci Quarterly 27*, 4 (1989), 338–347.

[18] CAPPELLO, P. R. Gaussian elimination on a hypercube automaton. *Journal of Parallel and Distributed Computing 4* (1987), 288–308.

[19] CHAN, M. Y. Dilation-2 embeddings of grid into hypercubes. In *Proc. Intn'l Conf. on Parallel Processing* (1988), pp. 295–298.

[20] CHAN, M. Y., AND CHIN, F. Y. On embedding rectangular grids in hypercubes. *IEEE Trans. Computers C-37*, 10 (Oct. 1988), 1285–1288.

[21] CHAN, M. Y., AND LEE, S.-J. Distributed fault-tolerant embeddings of rings in hypercubes. *J. of Parallel and Distributed Computing 11* (1991), 63–71.

[22] CHEN, G.-I., AND LAI, T.-H. Scheduling independent jobs on partionable hypercubes. *Journal of Parallel and Distributed Computing 12* (1991), 74–78.

[23] CHEN, M.-S., AND SHIN, K. G. Message routing in an injured hypercube. In *Proc. 3^{rd} Conf. Hypercube Concurrent Computers and Applications* (1988), SIAM, pp. 312–317.

[24] Chen, M.-S., and Shin, K. G. Depth-first search approach for fault-tolerant routing in hypercube multicomputers. *IEEE Trans. Parallel and Distributed Systems 1*, 2 (1990), 152–159.

[25] Chen, M.-S., and Shin, K. S. Processor allocation in an n-cube multiprocessor using Gray codes. *IEEE Trans. Computers C-36*, 12 (Dec. 1987), 1396–1407.

[26] Chen, S.-K., Liang, C.-T., and Tsai, W.-T. Loops and multi-dimensional grids on hypercubes: Mapping and reconfiguration algorithms. In *Proc. Intn'l Conf. on Parallel Processing* (1988), pp. 315–322.

[27] Christon, M. A vectorized 3-D finite element model for transient simulation of two-phase heat transport with phase transformation and a moving interface. In *Proc. of Supercomputing Conference* (1990), pp. 436–445.

[28] Chuang, P.-J., and Tzeng, N.-F. Dynamic processor allocation in hypercube computers. In *Proc. The 17th Annual Int'l Sympo. on Computer Architecture* (1990), pp. 40–49.

[29] Cvetanovic, Z. The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Trans. Computers C-36*, 4 (Apr. 1987), 421–432.

[30] Cvetanovic, Z., Freedman, E. G., and Nofsinger, C. Efficient decomposition and performance of parallel PDE, FFT, Monte Carlo simulation, simplex, and sparse solvers. In *Proc. of Supercomputing Conference* (1990), pp. 465–474.

[31] Cypher, R., Sanz, J., and Snyder, L. Hypercube and shuffle-exchange algorithms for image component labeling. *Journal of Algorithms 10*, 1 (1989), 140–150.

[32] Cypher, R., Sanz, J., and Snyder, L. Algorithms for image component labeling on simd mesh-connected computers. *IEEE Trans. Computers C-39*, 2 (1990), 276–281.

[33] Cypher, R., Sanz, J., and Snyder, L. The hough transform has O(n) complexity on n *times* n mesh-connected computers. *SIAM J. of Comput. 19*, 5 (1990), 805–820.

[34] DALLY, W. J., AND SEITZ, C. L. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Computers C-36*, 5 (May 1987), 547–553.

[35] DEIWERT, G. S. Computational aerothermodynamics. In *Proc. of Supercomputing Conference* (1989), pp. 51–57.

[36] DEKEL, E., NASSIMI, D., AND SAHNI, S. Parallel matrix and graph algorithms. *SIAM J. of Comput. 10*, 4 (1981), 657–675.

[37] DUTT, S., AND HAYES, J. P. Designing fault-tolerant systems using automorphisms. *Journal of Parallel and Distributed Computing 12* (1991), 249–268.

[38] DUTT, S., AND HAYES, J. P. Subcube allocation in hypercube computers. *IEEE Trans. Computers C-40*, 3 (Mar. 1991), 341–352.

[39] EFE, K. Embedding mesh of trees in the hypercube. *Journal of Parallel and Distributed Computing 11*, 3 (Mar. 1991), 222–230.

[40] EL-AMAWY, A., AND LATIFI, S. Bridged hypercube network. *Journal of Parallel and Distributed Computing 10* (1990), 90–95.

[41] EL-AMAWY, A., AND LATIFI, S. Properties and performance of folded hypercubes. *IEEE Trans. Parallel and Distributed Systems 2*, 1 (Jan. 1991), 31–42.

[42] ESFAHANIAN, A.-H., NI, L. M., AND SAGAN, B. The twisted n-cube with application to multiprocessing. *IEEE Trans. Computers C-40*, 1 (Jan. 1991), 88–93.

[43] FISHBURN, J. P., AND FINKEL, R. A. Quotient networks. *IEEE Trans. Computers C-31*, 4 (1982), 288–295.

[44] FOLDES, S. A characterization of hypercubes. *J. Discrete Math. 17* (1977), 155–159.

[45] FORTUNE, S., AND WYLLIE, J. Parallelism in random access machines. In *Proc.* 10th *Annual ACM Symposium on Theory of Computing* (1978), pp. 114–118.

[46] FOX, G., JOHNSON, M., LYZENGA, G., OTTO, S., SALMON, J., AND WALKER, D. *Solving Problems on Concurrent Processors.* Prentice Hall, 1988.

[47] FOX, G. C., OTTO, S. W., AND HEY, A. J. G. Matrix algorithms on a hypercube I: Matrix multiplication. *Parallel Computing 4* (1987), 17–31.

[48] Gannon, D., and Van Rosendale, J. On the impact of communication complexity in the design of parallel numerical algorithms. *IEEE Trans. Computers 33*, 12 (Dec. 1984), 1180–1194.

[49] Ghosh, K., and Fujimoto, R. M. Parallel discrete event simulation using space-time memory. In *Proc. Intn'l Conf. on Parallel Processing* (1991), pp. III-201–III-208.

[50] Gilbert, E. N. Gray codes and paths on the n-cube. *Bell System Technical Journal 37* (1958), 815–826.

[51] Gordon, J. M., and Stout, Q. F. Hypercube message routing in the presence of faults. In *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications* (1988), SIAM, pp. 318–327.

[52] Gould, R. J. Updating the hamiltonian problem – A survey. *J. of Graph Theory 15*, 2 (June 1991), 121–157.

[53] Graham, R. L., Knuth, D. E., and Patashnik, O. *Concrete Mathematics.* Addison Wisley, 1989.

[54] Hastad, J., Leighton, T., and Newman, M. Reconfiguring a hypercube in the presence of faults. In *Proc. 19th Annu. ACM Symp. Theory of Computer Science* (1987), ACM, pp. 274–284.

[55] Hayes, J. P., Mudge, T. N., and Stout, Q. F. Architecture of a hypercube supercomputer. In *Proc. Intn'l Conf. on Parallel Processing* (1986), pp. 653–660.

[56] Hilbers, P. A. J., Koopman, M. R. J., and van de Snepscheut, J. L. A. *The Twisted Cube.* Spring-Verlag, 1987, pp. 152–159. Lecture Notes in Computer Science.

[57] Ho, C.-T. *Optimal Communication Primitives and Graph Embeddings on Hypercubes.* PhD thesis, Yale University, May 1990.

[58] Ho, C.-T., and Johnsson, S. L. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proc. Intn'l Conf. on Parallel Processing* (1986), pp. 640–648.

[59] Ho, C.-T., and Johnsson, S. L. On the embedding of arbitrary meshes in boolean cubes with expansion two dilation two. In *Proc. Intn'l Conf. on Parallel Processing* (1987), pp. 188–191.

[60] HO, C.-T., AND JOHNSSON, S. L. Embedding meshes in boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing 8* (1990), 325–339.

[61] HSU, W.-J. Fibonacci cubes - properties of an asymmetric interconnection topology. In *Proc. Intn'l Conf. on Parallel Processing* (1991), pp. I722–723.

[62] HSU, W.-J. Fibonacci cubes-A new interconnection topology. *IEEE Trans. Parallel and Distributed Systems* (1992). To appear.

[63] HSU, W.-J., PAGE, C. V., AND LIU, J. Embedding meshes in a large family of graphs. *Parallel Processing Letters* (1992). Accepted for publication.

[64] HSU, W. J., PAGE, C. V., AND LIU, J. Parallel construction of Fibonacci trees and prefix computations on a family of interconnection topologies. In *Proc. CAAP/ESOP* (1992).

[65] JAKOB, R., AND HACK, J. J. Parallel MIMD programming for global models of atmospheric flow. In *Proc. of Supercomputing Conference* (1989), pp. 106–112.

[66] JOHNSSON, S. L. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing 4*, 2 (1987), 133–172.

[67] JOHNSSON, S. L., AND HO, C.-T. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers C-38*, 9 (Sept. 1989), 1249–1268.

[68] KATSEFF, H. Incomplete hypercube. *IEEE Trans. Computers C-37*, 5 (May 1988), 604–608.

[69] KERMANI, P., AND KLEINROCK, L. Virtual cut-through: A new computer communication switching technique. *Computer Networks 3*, 4 (1979), 267–286.

[70] KIM, J., DAS, C. R., AND LIN, W. A processor allocation scheme for hypercube computers. In *Proc. Intn'l Conf. on Parallel Processing* (1989), pp. II231–II238.

[71] KNUTH, D. E. *The Art of Computer Programming – Fundamental Algorithms*, 2 ed., vol. 1. Addison-Wesley, 1973.

[72] KNUTH, D. E. *The Art of computer Programming – Sorting and Searching*, 2 ed., vol. 3. Addison-Wesley, 1973.

[73] LADNER, R. E., AND FISCHER, M. J. Parallel prefix computation. *Journal of the ACM 27*, 4 (1980), 831–838.

[74] LANDER, E., AND MESIROV, J. P. Protein sequence comparison on a data parallel computer. In *Proc. Intn'l Conf. on Parallel Processing* (1988), pp. 257–263.

[75] LEIGHTON, T. *Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes.* Morgan Kaufmann, 1992.

[76] LI, K., AND CHENG, K. H. Job scheduling in a partitionable mesh using a two-dimensional buddy system partitioning scheme. *IEEE Trans. Parallel and Distributed Systems 2*, 4 (Oct. 1991), 413–422.

[77] LI, K., AND CHENG, K. H. A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. *Journal of Parallel and Distributed Computing 12* (1991), 79–83.

[78] LIU, J., CHIANG, C.-M., AND HUGHES, H. D. Performance analysis of load-sharing for hypercube multiprocessor systems. *Computer Systems Science and Engineering* (1992). Accepted for Publication.

[79] LIU, J., AND HSU, W.-J. Routing and broadcasting algorithms for a large family of interconnection topologies. Tech. Rep. CPS-91-05, Dept. of Computer Science, Michigan State University, Oct. 1991. Submitted for Publication.

[80] LIU, J., AND HSU, W.-J. Distributed algorithms for shortest-path, deadlock-free routing and broadcasting in a class of interconnection topologies. In *Proc. International Parallel Processing Symposium* (1992), pp. 589–596.

[81] LIU, J., AND HSU, W.-J. Distributed algorithms for shortest-path, deadlock-free routing and broadcast in Fibonacci cubes. In *Proc. 11th Phoenix Conf. on Computers and Communications* (1992), pp. 23–30.

[82] LIU, J., AND HSU, W.-J. Optimal broadcasting and embedding in incomplete hypercube. Submitted for publication, May 1992.

[83] LIU, J., HSU, W.-J., AND CHUNG, M. J. Generalized Fibonacci cubes are mostly hamiltonian. Submitted for publication, May 1992.

[84] LIU, J., AND HUGHES, H. D. Performance studies of hypercube multiprocessor systems. In *Proc. Summer Computer Simulation* (1990), Society for Computer Simulation.

[85] LIU, J., AND HUGHES, H. D. Dynamic job scheduling in partitionable mesh connected multicomputer systems. In *Proc. Summer Computer Simulation* (1992), Society for Computer Simulation. To appear.

[86] LUBECH, O. M., AND FABER, V. Modeling the performance of hypercubes: A case study using the particle-in-cell application. *Parallel Computing 9* (1988), 37–52.

[87] MA, Y.-W. E., AND TAO, L. Embeddings among toruses and meshes. In *Proc. Intn'l Conf. on Parallel Processing* (1987), pp. 178–187.

[88] MILLER, G. L., AND REIF, J. H. Parallel tree contraction and its application. In 26^{th} *Symposium on Foundations of Computer Science* (1985), pp. 478–489.

[89] MILLER, G. L., AND REIF, J. H. Parallel tree contraction Part I: fundamentals. In *Advances in Computing Research*, S. Micali, Ed., vol. 5. JAI Press Inc., 1989, pp. 47–72.

[90] MILLER, G. L., AND REIF, J. H. Parallel tree contraction Part II: further applications. *SIAM J. of Comput. 20*, 6 (Dec. 1991), 1128–1147.

[91] MILLER, R., AND STOUT, Q. Geometric algorithms for digitized pictures on a mesh-connected computer. *IEEE Trans. on PAMI 7*, 2 (1985), 216–228.

[92] NASSIMI, D., AND SAHNI, S. An optimal routing algorithm for mesh-connected parallel computers. *Journal of the ACM 27*, 1 (Jan. 1980), 6–29.

[93] NASSIMI, D., AND SAHNI, S. Data broadcasting in SIMD computers. *IEEE Trans. Computers 30*, 2 (Feb. 1981), 101–107.

[94] PELEG, D., AND ULLMAN, J. D. An optimal synchronizer for the hypercube. *SIAM J. of Comput. 18*, 4 (Aug. 1989), 740–747.

[95] PETERSON, J. C., TUAZON, J. O., LIEBERMAN, D., AND PNIEL, M. The Mark III hypercube-ensemble concurrent computer. In *Proc. Intn'l Conf. on Parallel Processing* (1985), pp. 71–75.

[96] PREPARATA, F. P., AND VUILLEMIN, J. The cube-connected cycles: A versatile network for parallel computation. *Communications of the ACM 24*, 5 (May 1981), 300–309.

[97] QUINN, M. J. *Designing Efficient Algorithms for Parallel Computers.* McGraw Hill, 1987.

[98] RANKA, S., AND SAHNI, S. Image template matching on MIMD hypercube multicomputers. In *Proc. Intn'l Conf. on Parallel Processing* (1988), pp. 92–99.

[99] RANKA, S., AND SAHNI, S. *Hypercube algorithms : with Applications to Image Processing and Pattern Recognition.* Springer-Verlag, 1990.

[100] SAAD, Y., AND SCHULTZ, M. H. Data communication in hypercube. Tech. Rep. YALEU/DCS/RR-428, Dept. of Computer Science, Yale Univ., 1985.

[101] SAAD, Y., AND SCHULTZ, M. H. Topological properties of the hypercubes. *IEEE Trans. Computers C-37*, 7 (July 1988), 867–872.

[102] SCHWABE, E. J. On the computational equivalence of hypercube-derived networks. In *ACM* 2^{nd} *Symposium on Parallel Algorithms and Architectures* (1990), ACM, pp. 388–397.

[103] SEITZ, C. L. The cosmic cube. *Communications of the ACM 28*, 1 (1985), 22–33.

[104] SHAPIRO, S. L., AND TEUKOLSKY, S. A. Building black holes, gravitational waves, and relativistic fluid flow: Supercomputer cinema. In *Proc. of Supercomputing Conference* (1990), pp. 805–814.

[105] STONE, H. S. Parallel processing with the perfect shuffle. *IEEE Trans. Computers C-20*, 2 (Feb. 1971), 153–161.

[106] SULLIVAN, H., AND BASHKOW, T. R. A large scale homogeneous, full distributed parallel machine, I. In *Proc. 4th Symp. Comput. Architecture* (1977), pp. 105–117.

[107] TIEN, J.-Y., HO, C.-T., AND YANG, W.-P. Broadcasting on incomplete hypercubes. Tech. Rep. RJ 8130, IBM Corp., 1991.

[108] TUTTE, W. T. *Graph Theory.* Addison-Wesley Publishing Co., 1984.

[109] TZENG, N.-F. Structural properties of incomplete hypercube computers. In *Proc. Intn'l Conf. on Distributed Computing Systems* (1990), pp. 262–269.

[110] TZENG, N.-F., CHEN, H.-L., AND CHUANG, P.-J. Embeddings in incomplete hypercubes. In *Proc. Intn'l Conf. on Parallel Processing* (1990), pp. III–335–III–339.

[111] TZENG, N.-F., AND WEI, S. Enhanced hypercubes. *IEEE Trans. Computers C-40*, 3 (Mar. 1991), 284–294.

[112] WAGNER, A. Embedding arbitrary binary trees in a hypercube. *Journal of Parallel and Distributed Computing 7* (1989), 503–520.

[113] WU, A. Y. Embedding of tree networks into hypercubes. *Journal of Parallel and Distributed Computing 2* (1985), 238–249.

[114] YANG, G.-C. Parallelizing spice2 on shared-memory multiprocessors. In *Proc. Intn'l Conf. on Parallel Processing* (1991), pp. III–151–III–158.