Middle East University for Graduate Studies

Faculty of Information Technology
Computer Science Department

Embedding Hex-Cells into Tree-Hypercube Networks

By

Awad Ibrahim Qatawneh

Supervised by
Prof. Dr. Ali Meligy

A Thesis Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of
the Requirements for The Degree of

Master of Science
in Computer Science

December, 2008

In the name of Allah,
The Most Beneficent,
The Most Merciful.

Read: In the Name of Your Lord
Who Created, Created Man from a Clot

Read: And Your Lord is
The Most Bounteous.

Who Taught by the Pen.

Taught Man that which
He did not know

Qur'an , Sura 96

Committee Decision

This Thesis (Embedding Hex-Cells into Tree-Hypercube Networks) was successfully defended and approved on December 13 th 2008.

Examination Committee Signature

Prof. Dr. Ali Meligy
Professor, Department of Computer Science
(Middle East University for Graduate Studies) ----------

Dr. Emad Abuelrub
Associate Professor, Department of Computer Science
(Zarqa Private University) ----------

Dr. Jihad Al-Sadi
Associate Professor, Department of Computer Science
(Arab Open University) ----------

Dr. Nidal Shilbayeh
Associate Professor, Department of Computer Science
(Middle East University for Graduate Studies) ----------

جامعة الشرق الأوسط للدراسات العليا

نموذج تفويض

أنا عوض إبراهيم عوض القطاونة، أفوّض جامعة الشرق الأوسط للدراسات العليا بتزويد نسخ من رسالتي/ أطروحتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع :

التاريخ :

Middle East University for Graduate Studies
Authorization Form

I, Awad Ibrahim Awad Qatawneh authorize The Middle East University for Graduate Studies to supply copies of my Thesis/Dissertation to libraries or establishments or individuals on request.

Signature:

Date :

Dedication

Dedicated to my beloved mother and father, who brought me up from childhood educated me, and gave me the courage to confront the challenges of life…….

To my brothers and sisters, Mohammed, Kamal, Islam , and Aya ……..

To all my friends and colleagues ……

I dedicate this work

Awad Qatawneh

Acknowledgments

Prior to acknowledgments, I must glorify Allah the Almighty who gave me courage and patience to carry out this work successfully.

I would like to express my great thanks to my advisor, Prof. Dr. Ali Meligy. He was always receptive to the ideas that I came up with, and provided me great knowledge and continuous encouragements.

I would like to express my great thanks to my parents, brothers , and sisters .

Awad Qatawneh

Table of contents

List of Figures

List of Tables

List of Abbreviations

BRGC	Binary Reflected Gray Code.
HC	Hex-Cell Network .
TH	Tree-hypercube Network.

Abstract

Graph embedding or graph mapping is an important aspect for interconnection networks used for communication between processors in parallel systems. Some parallel algorithms use communication structures which can be represented by hex-cells. In order to run these algorithms on a tree-hypercube multiprocessor system, without changing the current topology and the running application, their communication graphs need to be embedded into tree- hypercube.

In this thesis, we have designed an algorithm for embedding hex-cells of n nodes into tree-hypercube TH(2,d) where d >= 2. The embedding has dilation one, congestion one, and expansion 1.1. In the algorithm an embedding of irregular shape of hex-cells into tree-hypercube TH(2,d) where d >= 2 is performed. We have also designed an algorithm for embedding hex-cells HC(i), where i= 1,2 into tree-hypercube TH (2, d), where d = 2i.

As a result the embedding of Hex-cells into Tree-Hypercube in algorithm 4.1 has dilation one, congestion one, and expansion 1.1, and an embedding of irregular shape of hex-cells into tree-hypercube TH(2,d) where d >= 2 is performed. In algorithm 4.2 the embedding has dilation 1, congestion 1 and expansion 1.1 when mapping HC(1); and dilation 1, congestion 1 expansion 1.3 when mapping HC(2).

الملخص

يعتبر دمج الرسومات التي تمثل شبكات الاتصال أحد المواضيع المهمة بالنسبة لشبكات الاتصال التي تستخدم لتبادل البيانات بين المعالجات في النظم المتوازية . وعلى اعتبار أن الخوارزميات المتوازية تستخدم تراكيب الاتصال والمتمثلة بشبكات الخلايا السداسية. من أجل تمكين هذه الخوارزميات من العمل على نظم متوازية تستخدم الشبكات الشجرية لفوق المكعبات كنموذج للاتصال، مع عدم تغيير تراكيب الاتصال الحالية والبرامج الفعالة عليها، لا بد من دمجها بالشبكات الشجرية للمكعبات.

في هذه الرسالة، تم تصميم خوارزمية لدمج الخلايا السداسية ل ن من المعالجات في الشبكات الشجرية لفوق المكعبات بدرجه شبكة مقدارها 2 وبعمق متغير د = > 2. بالإضافة إلى أن عوامل الدمج، عامل التمدد =1 ، عامل الازدحام =1 ، والتضخم = 1.1. حيث أن عملية الدمج المتمثلة بالخوارزمية تتم للشكل غير المنتظم في الخلايا السداسية. في هذه الرسالة أيضا، تم تصميم خوارزمية لدمج الخلايا السداسية ذات البعد الاول والثاني، ن= 1، 2 في الشبكات الشجرية لفوق المكعبات، وبدرجه شبكة مقدارها 2 وبعمق متغير د = 2 ن.

إن النتائج التي تم التوصل إليها هي أن عوامل الدمج، عامل التمدد =1 ، عامل الازدحام =1 ، والتضخم = 1.1. وأن عملية الدمج المتمثلة بالخوارزمية تتم للشكل غير المنتظم في الخلايا السداسية وذلك بسبب تركيب الشبكات الشجرية لفوق المكعبات. بالإضافة إلى أن عوامل الدمج، عامل التمدد =1 ، عامل الازدحام =1 ، والتضخم = 1.1 لدمج الخلايا السداسية ذات البعد الأول. عامل التمدد =1، عامل الازدحام=1 ، والتضخم = 1.3 لدمج الخلايا السداسية ذات البعد الثاني.

Chapter One

Introduction

Parallel computing is a form of computing in which many instructions are carried out simultaneously [12]. Parallel computing operates on the principle that large problems can almost always be divided into smaller ones, which may be carried out concurrently ("in parallel"). Parallel computing exists in several different forms: bit-level parallelism, instruction level parallelism, data parallelism, and task parallelism. It has been used for many years, mainly in high performance computing, but interest in it has become greater in recent years due to physical constraints preventing frequency scaling. Parallel computing has recently become the dominant paradigm in computer architecture, mainly in the form of multi core processors [4].

Interconnection networks provide mechanisms for data transfer between processing nodes or between processors and memory modules; they are classified as static and dynamic. Static interconnection networks consist of point–to-point communication links among processing nodes, they are also referred to as direct networks. Dynamic networks on the other hand, are built using switches and communication links. Communication links are connected to one another dynamically by the switches to establish paths among processing nodes and memory banks. Dynamic networks are also referred to as indirect networks.

One of the primary design considerations for a massive multiprocessor system is the topology of the interconnection network used for communication between processors. Some of the important interconnection networks which have been studied so far are hypercube, meshes, trees, mesh-of-trees [27], tree-hypercube [19], hex-cell [24], and pyramids. tree-hypercube is one of the popular general purpose networks due to its small degree, short diameter, symmetry, optimal fault-tolerant properties, embedding of other networks, and their ability of implementing fast algorithms [19].

One of the important properties of any general interconnection network is that it should be able to embed other interconnection networks with low overheads. This attribute makes it a good candidate for a topology underlying a general-purpose parallel machine [7]. The problem of mapping other interconnection topologies into tree-hypercube network has not received much attention from researchers.

Topology embedding is an important aspect in parallel computing for performing mapping techniques between network topologies. Such that each processing element (node or vertex) and edge in the a graph G = (V,E) that represents the guest topology is mapped into nodes and edges of graph G' = (V',E') which represents the host graph. This mapping has two advantages; the first one is that it reduces the amount of time processes spend interacting with each other, and the second is that it reduces the total amount of time some processes are idle while the others are engaged in performing some tasks. This improves the performance of communication between processing nodes, and gets over the problem of congesting communication. Another main advantage for embedding, is that if a graph G is mapped into G', then G' can simulate the behavior of G with less overhead [12] of executing tasks in parallel.

In our research, we will discuss the problem of embedding hex-cell static networks into tree-hypercube network. In this thesis, we have designed an algorithm for embedding hex-cells of n nodes into tree-hypercube TH(2,d) where d >= 2. We have also designed an algorithm for embedding hex-cells HC (i), where i = 1, 2 into tree-hypercube TH (2, d), where d = 2i, with dilation and congestion one, and expansion 1.1 for mapping HC(1), and expansion 1.3 for mapping HC(2), depending on the structure and depth of the tree-hypercube.

1.1 Terminologies:

Here, we introduce various criteria used to characterize the cost and performance of static interconnection networks [12]. We use these criteria to evaluate static networks, the embedding of such network topologies introduced in chapter two.

Degree

The degree of an interconnection network is defined as the maximum number of nodes at any node in the network. A desirable feature in an interconnection network is that the number of ports would not grow at the same rate as the number of processors. Therefore, it is useful to have a network with relatively few ports per node, since this will affect the expandability of the network [19].

Diameter

The diameter of an interconnection network is the maximum distance between any two nodes in the network, measured in terms of number of edges. The diameter is an important measure of the power of the interconnection network. It is useful to make this parameter as low as possible, since it will not only reduce the traveling time for messages, but also minimizes message density in the links of the network.

Average Distance

The average distance of an interconnection network is the average value of all distances between pairs of nodes in the network. One important measure of the power of a an interconnection network is the average distance between all nodes. It is very important to keep this parameter as low as possible, since it will reduce the traveling time for messages and provides more uniform message density in all links [19].

Bisection Width

The bisection width of a network, is defined as the minimum number of edges that must be removed to partition the network into two equal halves [19].

Bisection Bandwidth

The bisection bandwidth of a network is defined as the minimum volume of communication allowed between any two halves of a network [19].

Cost

The number of edges in the network, the smaller the better [19].

Congestion of Mapping

The maximum number of edges in E that are mapped onto any edge in E' is called the congestion of mapping [12].

Dilation of Mapping

The maximum number of edges in E' that any edge in E is mapped into is called the dilation of mapping.

Expansion of Mapping

The ratio of the number of nodes V' in the guest graph G' = (V', E') to nodes V in host graph G = (V, E).

1.2 Statement of the problem

There are some static interconnection networks which have been studied in the literature, such as hypercube, tree hypercube, tree, star, and ring. In particular, we are concerned with hex-cell interconnection network topology and tree-hypercube network. We study the problem of embedding hex-cell networks into tree-hypercube networks.

A good mapping is said to exist when adjacent processors in the guest network are mapped to reasonably close processors in the host network (i.e. small dilation) and when the paths between adjacent processors in the guest network are chosen in such a way that the congestion at each host node and across each host edge is moderately small (i.e. small congestion).

1.3 Goals

- Exploring and introducing the problem of embedding hex-cell networks into tree-hypercube networks.
- Developing a new mapping algorithm which allows the embedding hex-cell into tree-hypercube networks. We consider some embedding parameters such as congestion of mapping, dilation, and expansion.
- Developing an application to show the advantage of embedding of hex-cells into tree-hypercube networks.

1.4 Significance

Mapping interconnection topologies into tree-hypercube networks improves the performance of parallel systems by minimizing the overhead of executing tasks in parallel; we consider embedding hex-cell networks into tree-hypercube networks. The need for processor embedding or mapping comes from the need of a programmer to run an application on other topologies. In our case the programmer needs to run a hex-cell application on a tree-hypercube interconnection topology, without changing the application and the current interconnection topology (hex-cell). So we need to map the hex-cell into tree-hypercube networks, which means that the tree-hypercube can simulate hex-cell topology.

Chapter Two

Network Topologies

A wide variety of network topologies have been used in interconnection networks. These topologies try to trade off cost and scalability with performance. While pure topologies have attractive mathematical properties, in practice interconnection networks tend to be a combination or modification of the pure topologies discussed in this section.

2.1 Interconnection Networks Overview

In this subsection we introduce some interconnection networks that have been studied in literature, such as bus-based networks**,** completely-connected network, star**,** linear arrays, meshes, hypercubes, tree-hypercubes, and hex-cells.

Bus-Based Networks

A bus-based network is the simplest network consisting of a shared medium that is common to all the nodes. A bus has the desirable property that the cost of the network scales linearly as the number of nodes p. This cost is typically associated with bus interfaces. Furthermore, the distance between any two nodes in the network is constant (O (1)). Buses are also ideal for broadcasting information among nodes. Since the transmission medium is shared, there is little overhead associated with broadcast compared to point-to-point message transfer [12].

Completely-Connected Networks

In a completely-connected network, each node has a direct communication link to every other node in the network. Figure 2.1 illustrates a completely-connected network of eight nodes. This network is ideal in the sense that a node can send a message to another node in a single step, since a communication link exists between each two nodes [12].

Star- Connected Networks

In a star-connected network, one processor acts as the central processor. Every other processor has a communication link connecting it to this processor. Figure 2.2 shows a star-connected network of nine processors. Communication between any pair of processors is routed through the central processor, just as the shared bus forms the medium for all communication in a bus-based network. The central processor is the bottleneck in the star topology [12].

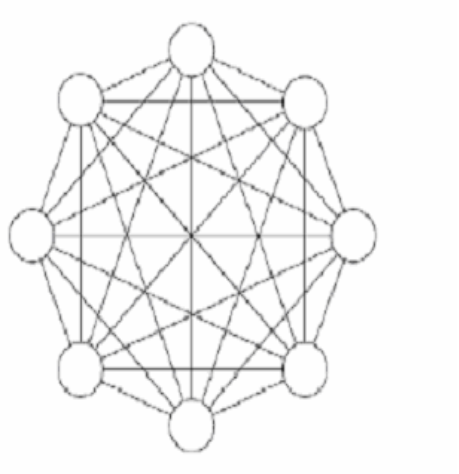

Figure 2.1: A completely connected network

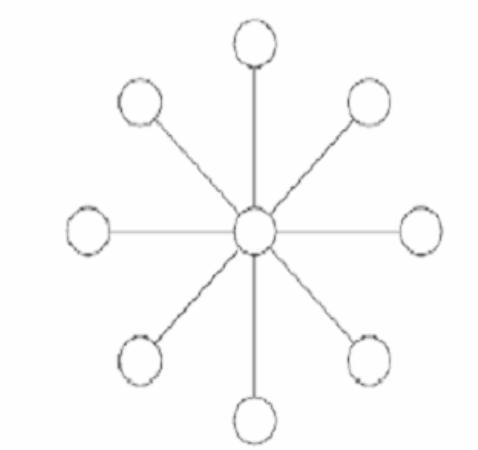

Figure 2.2: A star connected network.

Linear Arrays and Meshes

A linear array is a static network in which each node (except the two nodes at the ends) has two neighbors, one each to its left and right. A simple extension of the linear array (Figure 2.3-a) is the ring (Figure 2.3-b). The ring has a wraparound connection between the extremities of the linear array. In this case, each node has two neighbors.

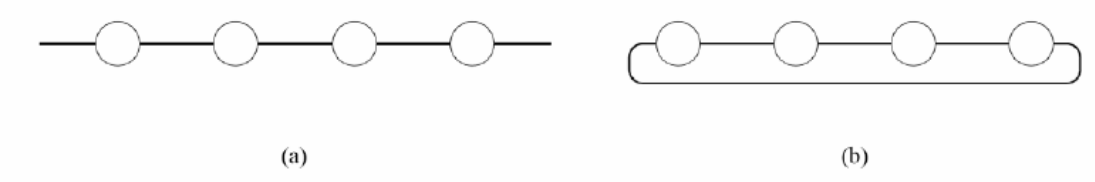

Figure 2.3: linear arrays: (a) with no wraparound links, (b) with wraparound links.

A two-dimensional mesh illustrated in Figure 2.4 is an extension of the linear array to two-dimensions. Each dimension has $\sqrt{p}$ nodes with a node identified by a two-tuple (i, j). Every node (except those on the periphery) is connected to four other nodes whose indices differ in any dimension by one. A 2-D mesh has the property that it can be laid out in 2-D space, making it attractive from a wiring standpoint. Furthermore, a variety of regularly structured computations map very naturally to a 2-D mesh. For this reason, 2-D meshes were often used as in parallel machines [12].

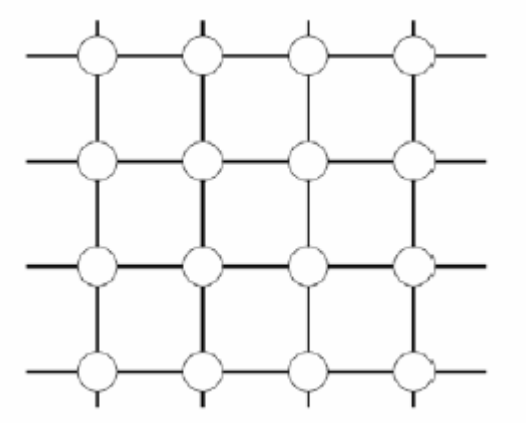

Figure 2.4: Two-dimensional mesh**.**

Hypercubes

The hypercube topology is an interconnection topology and known to be able to simulate other structures such as linear arrays and rings. The construction of a hypercube is illustrated in Figure 2.5. A zero-dimensional hypercube consist of 2^0, i.e, one node. A one-dimensional hypercube is constructed from two zero-dimensional hypercube by connecting them. A two dimensional hypercube of four nodes is constructed from two one-dimensional hypercubes by connecting corresponding nodes. In general, a *d*-dimensional hypercube is constructed recursively by connecting corresponding nodes of two (*d-1*) dimensional hypercubes, Figure 2.5 illustrates this for up to 16 nodes in a 4-D hypercube [12].

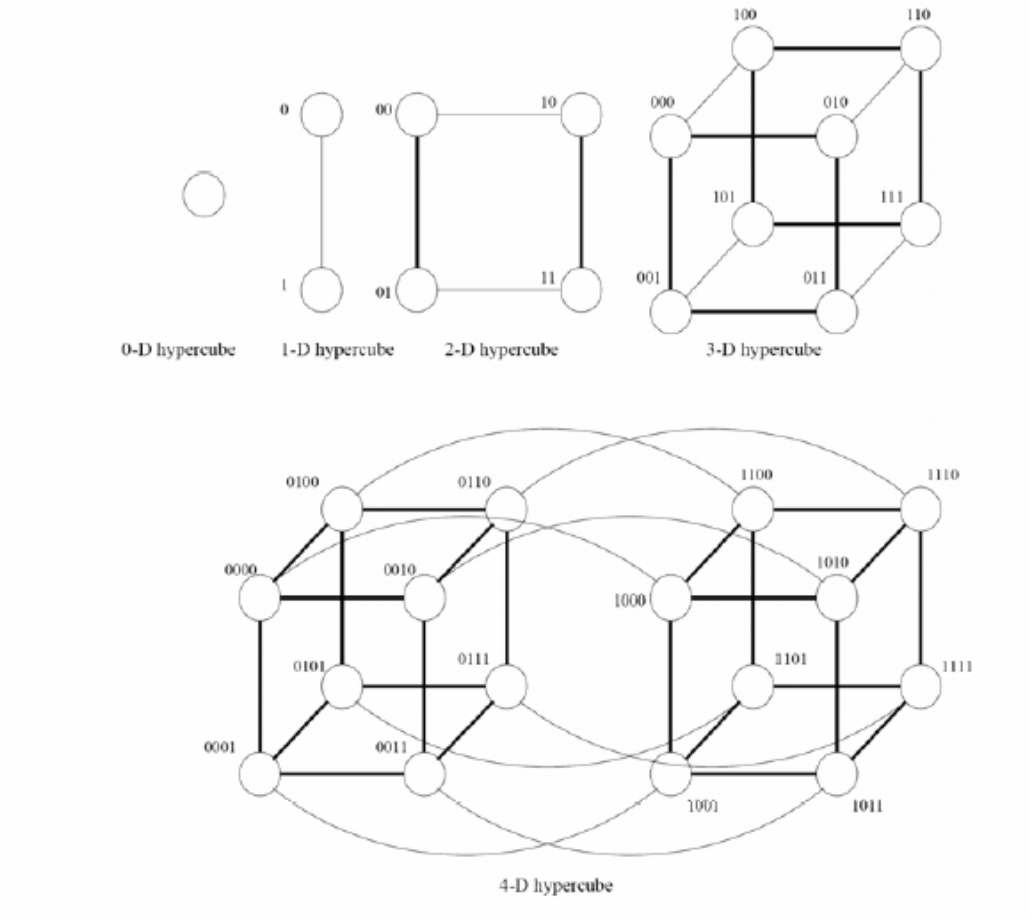

Figure 2.5: Recursive construction of hypercubes from hypercubes of lower dimensions

It is useful to derive a numbering scheme for nodes in a hypercube. A simple numbering scheme can be derived from the construction of a hypercube as illustrated in Figure 2.5. If we have a numbering of two subcubes of dimension *d-1*, we can derive a numbering scheme for the cube of dimension *d* by prefixing the labels of one of the subcubes with " 0" and the labels of the other subcubes with a "1". This numbering scheme has the useful property that the minimum distance between two nodes is given by the number of bits that are different in two labels. For example, nodes labeled 0110 and 0101 are two edges apart, since they differ at two bit positions. This property is useful for deriving a number of parallel algorithms for the hypercube architecture.

Tree- Hypercube Networks

Here, we define a tree-hypercube [19] and describe a recursive construction mechanism. A tree-hypercube network TH (s, d) is constructed by taking a full tree of degree s, where s is a power of 2 and depth *d*. Levels of the tree are numbered 0, 1, …, d. Each level *k* has s^k nodes representing processing elements and labeled from 0 to s^k*-1* in binary code and interconnected as a hypercube. Thus, nodes at level k constitute *(k log s)*-cube. Each node in a tree-hypercube is identified by a pair (l, x), where l denotes the level number and x is the cube address. The total number of nodes in TH (s, d) is $N = (s^{d+1}-1)/(s-1)$. Figures 2.6 and 2.7 display two tree-hypercube TH (2, 2) and TH (2, 3), respectively.

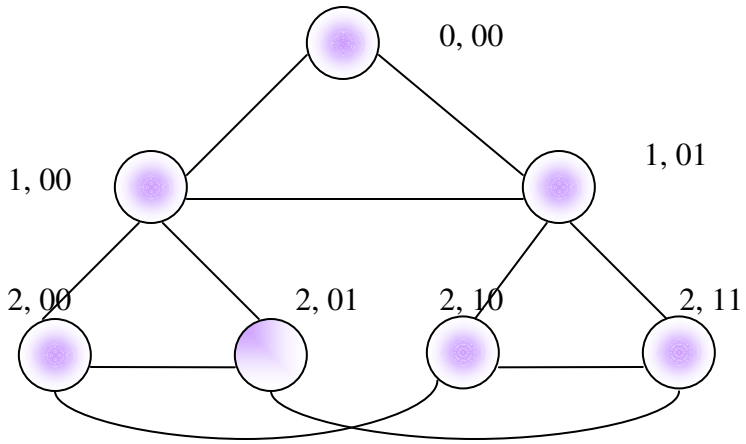

Figure 2.6: Tree-Hypercube TH (2, 2)

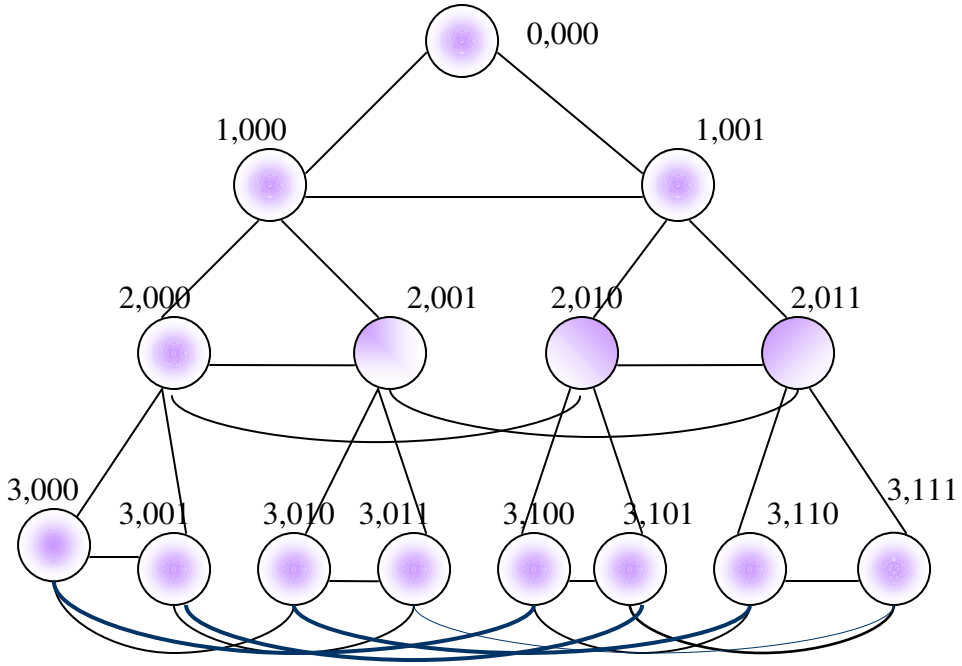

Figure 2.7: Tree-Hypercube TH (2, 3)

A Tree-Hypercube network combines the advantages of both trees and hypercubes and avoids their shortcoming. Its diameter is less than that of the hypercube. The tree-hypercube network can emulate many interconnection topologies such as linear array, ring, tree, hypercubes and meshes.

A Tree-hypercube network can be viewed as a recursive structure. TH (s,0) is the smallest tree-hypercube and consists of only one node. TH (s,d) can be constructed from s copies of TH (s, d-1) by adding a new node to be the root node of TH (s,d) and label it with (0,0) then connect it to the s TH (s,d-1)'s. Label these TH (s,d-1) networks by 0,1, ….., s-1, and transform the label of each of the s nodes (0,x) to (1,x.i) where i=0, 1, ….., s-1 in binary notation. Then transform the label of each node (l,x) in the i-th network to (l+1, i.x (l log s)-1 x(l log s)-2 …. x0). Now, any two nodes (l,x.r) and (l,x.j) become adjacent if r and j differ in only one bit position. As a result, all nodes at level L form an (l log s)–cube [19].

Another way of constructing TH (s,d) from TH (s,d-1) is by adding a new level to the bottom of TH (s, d-1). That level consists of s^d nodes labeled from 0 to $s^d - 1$ in binary and connected in (d log s)-cube structure. To establish interconnections between level d-1 and level d, we make every node (d-1, x) at level d-1 to be adjacent to s nodes in the new level d. These s nodes are (d, x.a), where a= 0,1, …, s-1 in binary notation.

This recursive structure is helpful in routing as well as in partitioning. It is also useful when extending TH (s,d) to TH (s,d+1) by adding a new level using the second method above with a fraction of the cost. As we mentioned earlier, s is a power of 2, and in practice s is preferably 2 or 4 because TH (2,d) and TH (4,d) satisfies all the conditions for practical parallel computer systems [19].

Hex-Cell Network

A hex-cell network [24] with depth d is denoted by HC (d) and can be constructed by using units of hexagon cells, each of six nodes. A hex-cell network with depth d has d levels numbered from 1 to d, where, level 1 represents the innermost level corresponding to one hexagon cell. Level 2 corresponds to the six hexagon cells surrounding the hexagon at level 1. Level 3 corresponds to the 12 hexagon cells surrounding the six hexagons at level 2, as shown in Figure 2.8 [24]. The levels of the HC(d) network are labeled from 1 to d. Each level i has N_i nodes, representing processing elements and interconnected in a ring structure. Addressing nodes in HC is shown in Figure 2.9 [24].

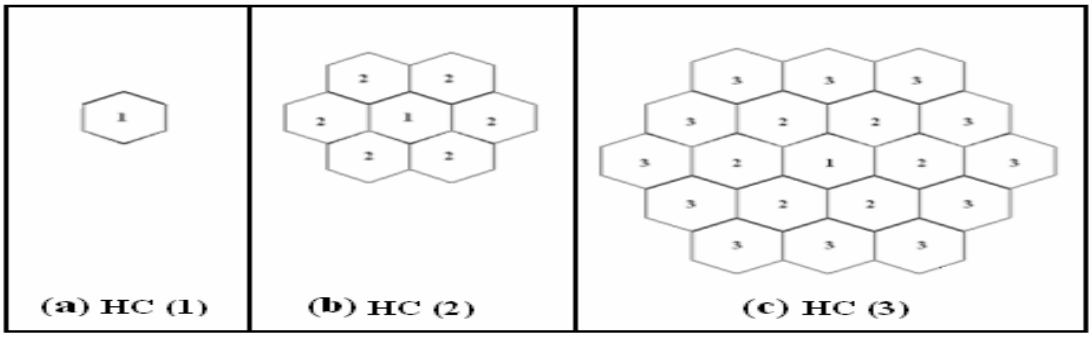

Figure 2.8**:** (a) HC (one level) (b) HC (two levels) (c) HC (three levels)

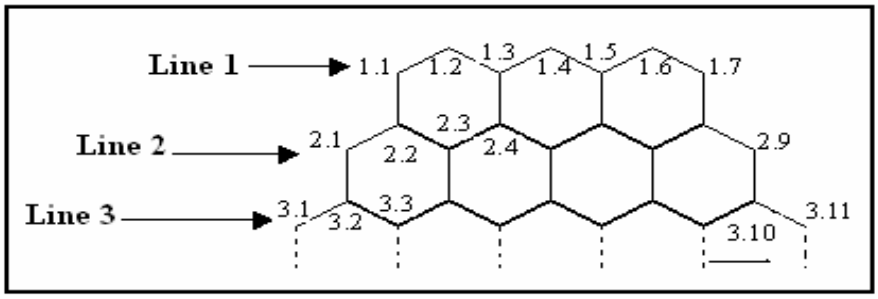

Figure 2.9**:** Addressing nodes in HC.

In a hex-cell network, the number of nodes at level I is : $N_i = 6(2i -1)$.

The total number of nodes in a hex-cell network HC(d) is : $N = 6d^2$.

The number of links in HC(d) is $L = 9d^2 - 3d$.

The bisection of HC(d) is 2d.

The proposed architecture of hex-cells has some of the interesting characteristics of hypercubes, binary trees, and linear arrays [24]. The degree and total number of links of the HC is less than those of the hypercube. In a hypercube, the node degree of each node is a logarithmic function of the total number of nodes, which is a drawback of the topology. Consequently, the hypercube topology is not a good candidate for interconnection networks for massive parallel computers due to limitations concerning integrated circuit technology and port numbers. The hex-cell topology is suitable for massive parallel computers, since it has a fixed degree which is 3. The diameter of hex-cell is less than that of the linear array and ring, and greater than that of the binary tree and hypercube [24].

Table 2.1 shows a comparison of parameters for different topologies including hex-cell [24].

Remarks on network size	Bisection Bandwidth	Number of Links	Diameter	Maximum Node Degree	Topology Name
N is the number of nodes	$2\sqrt{(N/6)}$	$3N/2 - 3\sqrt{(N/6)}$	$4\sqrt{(N/6)}-1$	3	**Hex-Cell**
N is the number of nodes	N/2	$\log_2 N$ (N/2)	$\log_2 N$	$\log_2 N$	**Hypercube**
N is the number of nodes,	1	N-1	$2(\log_2 N -1)$	3	**Binary Tree**
N nodes	1	N-1	N-1	2	**Linear Array**
N nodes	2	N	N/2	2	**Ring**
r × r torus where r = √N	N/2	2N	2(r/2)	4	**2D-Torus**
N= k × 2^k nodes with a cycle length k ≥ 3	N/(2k)	3N/2	2k-1+[k/2]	3	**Cube-Connected Cycles**
N is the number of nodes	$2.32\sqrt{N}$	$3N - 8.66\sqrt{N}$	$1.16\sqrt{N}$	6	**3D Hexagonal**

Table 2.1: Comparison of parameters for different topologies.

The degree of the HC is constant when d > 1 and less than the degree of hypercube and falls in between of linear array and 2d-torus as shown in Figure 2.10. Constant node degree facilitates modularity in building blocks for scalable systems [24]. The diameter of HC is less than that of the linear array and ring, and greater than that of the binary tree and hypercube as illustrated in Figure 2.11. The total number of links for HC is less than hypercube but greater than binary tree as shown in Figure 2.12. The proposed topology has the ability to efficiently simulate programs written for architectures such as linear arrays, rings and meshes. And can be used in wide range network applications such as mobile systems, and Ad-Hoc mobile networks [24].

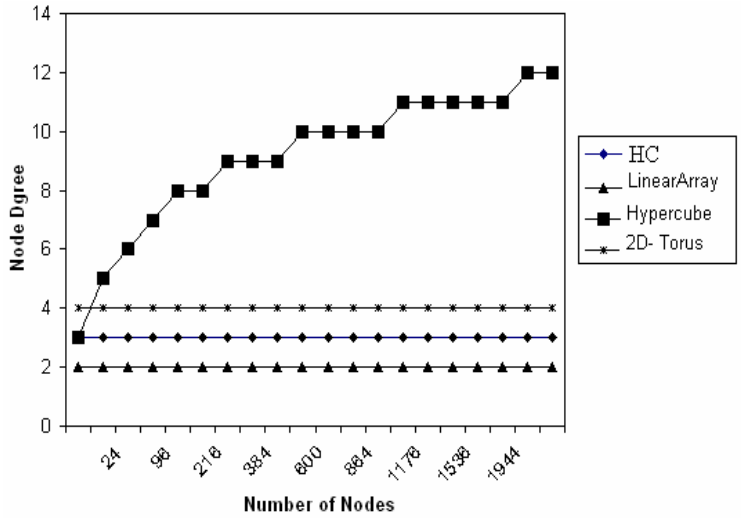

Figure 2.10**:** Degree of HC against the network size

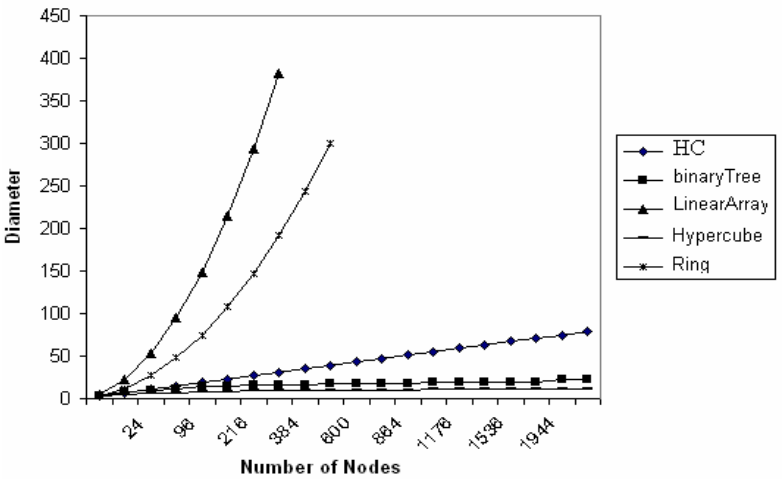

Figure 2.11: Diameter of HC against network size.

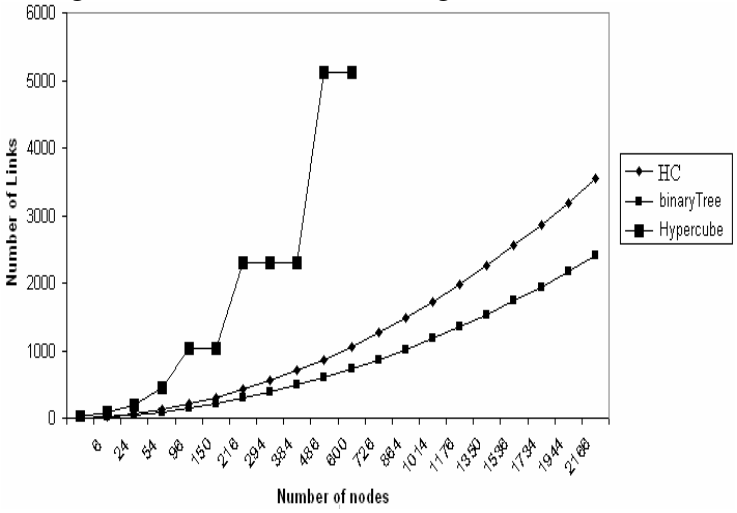

Figure 2.12: The number of links of HC against network size

2.2 Tree-Hypercube and Hypercubes Comparison

As we mentioned earlier, a tree-hypercube network combines the advantages of both trees and hypercubes and avoids their shortcoming [19]. Its diameter is less than that of the hypercube. The tree-hypercube network can emulate many interconnection topologies such as linear arrays, rings, trees, hypercubes, and meshes. In this section we present a comparison between tree-hypercube and hypercube topology.

Here, we introduce some criteria used to characterize the cost and performance of tree-hypercubes, and compare them with hypercubes; these criteria include degree, diameter, and average distance.

Degree

For a TH with s=2 the degree of the TH is *d+2* but for a hypercube with almost the same number of nodes the degree is *d+1*. Also for s=4 it is *2d+3*, but for a hypercube with almost the same number of nodes the degree is *2d+1*. The degree of a TH is larger than that of a hypercube by 1 and 2 for the cases when s=2 and s=4, respectively [19].

Diameter

The diameter of TH(s,d) is *d* log s*, the diameter of tree-hypercube is always smaller than the diameter of the hypercube [19].

Average Distance

The average distance V2 of TH (2,d) is:

$2 ((d+1) \ 4^d - d2^{d+1} -1) \ / \ (2^{d+1} -1)^2$, where $d/2 <= \ v2 < \ d+1 /2$.

And the average distance V4 of TH(4,d) is:

$d- 1/15 +8/(15(4^{d+1} -1)) - \ (d+1)/(4^{d+1} – 1)^2$, for $d > 1$, $d-(1/15) < \ V4 < d$.

When s=4, the average distance of tree-hypercube is always smaller than the average distance of the hypercube of almost the same number of node. When s=2, the average distance of tree-hypercube is smaller than that of hypercube for values $d \leq 9$, and they are equal for $d > 9$ [19].

2.3 Topology Embeddings

Embedding a guest graph G = (V, E) into a host graph G' = (V', E') is a mapping of each vertex in the set V into a vertex (or a set of vertices) in set V' and each edge in the set E into an edge (or a set of edges) in E'. When mapping graph G = (V, E) into G' = (V', E'), the following must be taken into consideration. First, one or more edge in E can be mapped into a single edge in E'. The maximum number of edges mapped into any edge in E' is called the *congestion* of mapping. Second, an edge in E may be mapped into multiple contiguous edges in E'. The maximum number of links in E' that any edge in E is mapped into is called the *dilation* of mapping. This is significant because traffic on the corresponding communication link in G must traverse more than one link in G', possibly contributing to the *congestio*n on the network. Third, the sets V and V' may contain different numbers of vertices. A node in V may correspond to more than one node in V' and vice versa. The ratio of the number of nodes in V' to that in V is called the *expansion* of mapping. If there is a good mapping of a graph G into G', then G' can simulate the behavior of G with less overhead [12].

2.3.1 Binary Reflected Gray Code (RGC)

The embedding of linear arrays into hypercubes discussed in [12], is based on mapping nodes of the linear array to the nodes of a hypercube, using the binary reflected gray code method, as presented below. A linear array (or a ring) composed of 2^d nodes (labeled 0 through $2^d - 1$) can be embedded into a d-dimensional hypercube by mapping node i of the linear array onto node G(i,d) of the hypercube. The function G(i,x) is defined as follows:

$$G(0,1) = 0$$
$$G(1,1) = 1$$
$$G(i, x+1) = \begin{cases} G(i, x), & i < 2^x \\ 2^x + G(2^{x+1} - 1 - i, x), & i >= 2^x \end{cases}$$

The function G is called the *Binary Reflected Gray Code (BRGC)*. The entry G(i,d) denotes the i th entry in the sequence of Gray codes of d bits. Gray codes of d+1 bits are derived from a table of Gray Codes of d bits by reflecting the table and prefixing the reflected entries with a 1 and the original entries with a 0. This process is illustrated in Figure 2.13. A careful look at the Gray Code table reveals that the two adjoining entries G(i, d) and G(i,d+1) differ from each other at only one bit position.

As an example on using the *Binary Reflected Gray Code,* we present the embedding of linear arrays or rings into a hypercubes. Since node i in the linear array is mapped to node G(i, D), and node i +1 is mapped to G(i+1, d), there is a direct link in the hypercube that corresponds to each direct link in the linear array. Recall that two nodes whose labels differ at only one bit position have a direct link in a hypercube. Therefore, the mapping specified by function G has a dilation of one and a congestion of one. Figure 2.14 illustrates the embedding of eight-node ring into a three-dimensional hypercube.

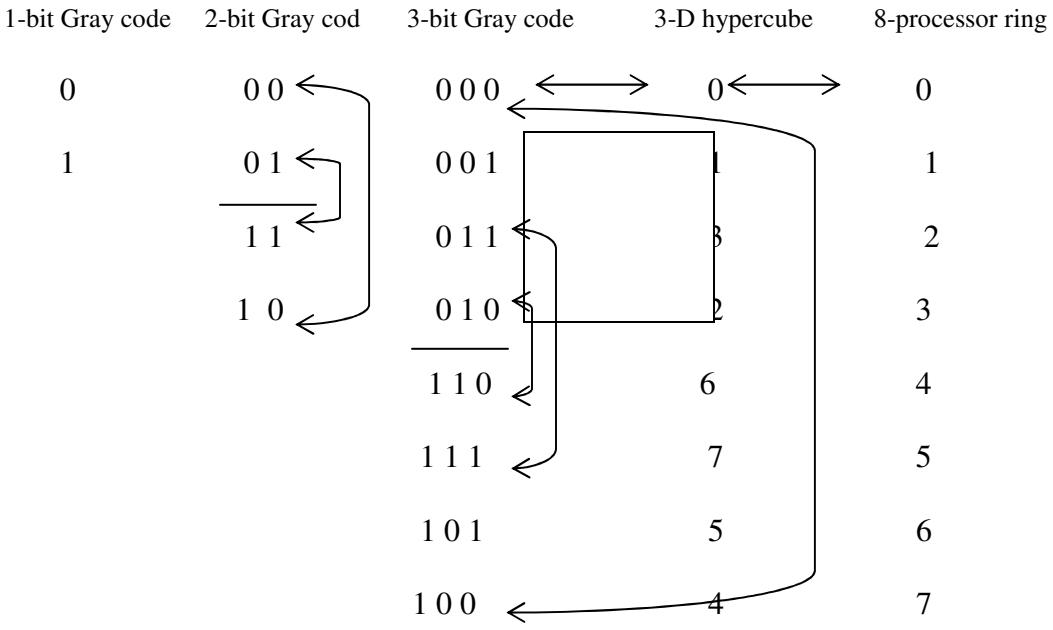

Figure 2.13: A three-bit reflected Gray Code ring.

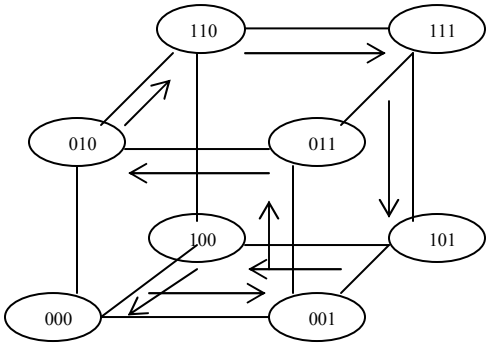

Figure 2.14: The embedding of eight-node ring into a three-dimensional hypercube

Chapter Three

Related Work

In this chapter, we discuss some of the work that are related to our research, the embedding of a hex-cell into a tree-hypercube. These related work include embedding of a ring into a tree-hypercube, embedding of a linear array into a tree-hypercube, and other embeddings.

3.1 Embedding Topologies into Hypercubes

In this section, we present some embeddings of different topologies into the hypercube networks, these different topologies include arbitrary binary trees, balanced binary trees, ring, and mapping of meshes on star graphs.

In [17] embeddings in hypercubes, stated that one important aspect of efficient use of a hypercube computer to solve a given problem is the assignment of subtasks to processors in such a way that the communication overhead is low. The subtasks and their inter-communication requirements can be modeled by a graph, and the assignment of subtasks to processors viewed as an embedding of the task graph into the graph of the hypercube network. We survey the known results concerning such embeddings, including expansion/dilation tradeoffs for general graphs, embeddings of meshes and trees, packings of multiple copies of a graph, the complexity of finding good embeddings, and critical graphs which are minimal with respect to some property.

In [14] the embedding of hierarchical hypercube networks into the hypercube was presented. And states that the embedding of one interconnection network into another is a very important issue in the design and analysis of parallel algorithms. Through such embeddings, the algorithms originally developed for one architecture can be directly mapped to another architecture. This paper describes a new embedding method, based on matrix transformations, for optimally embedding hierarchical hypercube networks (HHNs) into the hypercube (binary n-cube). Thus, this embedding method has practical importance in enhancing the capabilities and extending the

usefulness of the hypercube, since hierarchical hypercube networks have proven to be very cost-effective for a wide range of applications**.**

On mapping *n*-dimensional meshes on a star graph of degree *n* an algorithm developed in [22], the mapping has expansion 1 and dilation 3. This shows that an *n*-degree star graph can efficiently simulate an *n*-dimensional mesh. The mapping in this research is based on performing a sequence of exchanges along each dimension i of the mesh. On the other hand, embedding binary trees into hypercubes [9], states that hypercubes are known to be able to simulate other structures such as grids and binary trees. In this paper, an embedding of arbitrary binary trees into a hypercube with expansion 1 and dilation no more than 2 is proposed, and embeddings with expansion 2 and dilation 1.

On embedding star network into hypercubes [6]. The star interconnection network has recently been suggested as an alternative to the hypercube. As hypercubes are often viewed as universal and capable of simulating other architectures efficiently, we investigate embeddings of star network into hypercubes. Our embeddings exhibit a marked trade off between dilation and expansion. We also show that the embedding of S_n into its optimum hypercube requires dilation $\Omega (log_2 n)$.

The embedding of arbitrary binary tree into its optimal hypercube have been studied in [15]. A d-dimensional hypercube is a very popular model of parallel computing, and the execution of many algorithms can be represented by binary trees, making desirable fast simulations of binary trees on hypercubes. This paper presents a simple embedding of an arbitrary binary tree into its optimal hypercube with dilation 8 and constant congestion. The novelty of the method used here is based on the use of an intermediate quadtree data structure, which also permits the embedding to be efficiently computed on the hypercube itself. The embedding of *r*-ary *m* cubes into hypercubes is presented in [13], using matrix transformations as an alternate to reflected gray code, which achieves the same result. This new method has a nice property that makes it suitable to be used in divide-and-conquer algorithms. Thus, it constitutes a useful tool for the design of parallel algorithms for the hypercube. The method of matrix transformation

is based on, folding the hypercube recursively, so that the final layout contains the *r*-ary *m*-cube.

Embedding incomplete binary trees into incomplete hypercubes [16]. It has been proved that an incomplete binary tree cannot be embedded into an incomplete hypercube with dilation 1 and expansion 1. By applying some properties of in order traversal, the authors present an embedding scheme with dilation 2, edge-congestion 2 and expansion ratio $(N + 1)/N$, where N is the number of nodes in an incomplete binary tree. With this embedding scheme, a method is developed that can be used to simulate a binary tree on an incomplete hypercube effectively. Under the distributed environment, the mapping addresses of neighboring nodes in an incomplete binary tree can be identified in constant time without repeating the mapping work. Furthermore, experimental results show that this scheme is much better than the corresponding best known dilation 1 embedding scheme in terms of hardware costs and implementation. Even in total time costs (addressing time, computation time and transmission time), this approach is quite competitive.

On embedding balanced binary trees into hypercubes [2] states that in the context of parallel computing, the problem of embedding binary trees that represent communication structures arises. Whereas much research has focused on arbitrary trees, here the researcher concentrates on the subclass of balanced binary trees. The motivation for embedding into hypercube is that hypercube multiprocessor systems are very prominent type of parallel machines, because of their recursive structure and the fact that they contain structures like rings, 2-D-meshes, and higher dimensional meshes make them suitable for a large variety of problems. Many parallel algorithms use communication structures which can be represented by binary trees. In order to run these algorithms on a hypercube multiprocessor system, their communication graphs need to be embedded in the corresponding hypercube.

On the embedding of rings into faulty twisted hypercubes [1], stated that the hypercube is emerging as one of the most effective and popular network architectures for large scale parallel machines. Hypercube based machines are becoming more popular due to many of their attractive features in parallel computing. An attractive version of the hypercube is the twisted hypercube. It preserves many properties of the hypercube and most importantly reduces the diameter by a factor of two. In this paper an optimal embeddings of rings into faulty twisted hypercubes with up to 2^{n-3} faulty processes was presented.

In distributed fault-tolerant embedding of several topologies into hypercubes, [28] proposed a distributed fault tolerant embedding based on the faulty link model, for embedding several topologies into hypercubes, with faulty links and /or faulty nodes. Since embedding has great importance in the applications of parallel computing, every parallel application has its intrinsic communication pattern. The communication pattern graph is mapped into a topology of multiprocessor structure so that the application can be executed. To increase the reliability of parallel applications, fault-tolerant embedding is necessary. In this research, embedding of rings and toruses is studied and proposed. An optimal embedding of a honeycomb network (honeycomb mesh and honeycomb torus) of size n into a hypercube is presented in [5], with expansion ratio of 4/3 = 1.33 when n is a power of two. When n is not a power of two, the expansion is 16/3 = 5.33, which we conjecture to be near optimal. For a honeycomb mesh, the dilation of the embedding is 1. A honeycomb network, built recursively using hexagon tessellation, is a multiprocessor interconnection network that is better than the planar mesh with the same number of nodes in terms of degree, diameter, number of links, and bisection width.

In [26] Cycles embedding in hypercubes with node failures. The hypercube has been widely used as the interconnection network in parallel computers. The n-dimensional hypercube Q_n is a graph having 2^n vertices each labeled with a distinct n-bit binary strings. Two vertices are linked by an edge if and only if their addresses differ exactly in the one bit position. Let f_v denote the number of faulty vertices in Q_n. For $n \geq 3$, this paper, proofs that every fault-free edge and fault-free vertex of Q_n lies on a fault

free cycle of every even length from 4 to 2^n-2f_v inclusive even if $f_v \leq n-2$. These results are optimal.

In [11] the embedding of cycles in twisted cubes was studied. It has been proven in the literature that, for any integer l, $4 \leq l \leq 2^n$, a cycle of length l can be embedded with dilation 1 in an n-dimensional twisted cube, $n \geq 3$. In this paper the researcher obtained a stronger result of embedding of cycles with edge-pancyclic. And proven that, for any integer l, $4 \leq l \leq 2^n$, and a given edge (x,y) in an n-dimensional twisted cube, $n \geq 3$, a cycle C of length l can be embedded with dilation 1 in the n-dimensional twisted cube such that (x,y) is in C in the twisted cube. Based on the proof of the edge-pancyclicity of twisted cubes, further provided an $O(l \log l + n^2 + nl)$ algorithm to find a cycle C of length l that contains (u,v) in TQ_n for any $(u,v) \in E(TQ_n)$ and any integer l with $4 \leq l \leq 2^n$.

3.2 Embedding Topologies into Tree-Hypercube Networks

Here, we introduce the mapping of Linear Array network into tree-hypercube [21], and its extension which is the mapping of a ring into a tree-hypercube [3]. The embedding of Linear Array network into tree-hypercube is presented in [21]. This research shows that graph embedding exists when adjacent processors in the guest network are mapped to reasonably close processors in the host network (i.e. small dilation), and when the paths between adjacent processors in the guest network are chosen in such way that the congestion of each host node and across each host edge is moderately small (i.e. small congestion). An approach for embedding linear array into tree-hypercube is proposed in that paper. Arrays are one of the most natural (data or process) structures for many applications. Linear array of $2^{d+1}-1$ processors (where d=0,1,2,,3,…,n) can be embedded into a tree-hypercube TH (s,d) by mapping processor G onto processor G'(L,X) of the tree-hypercube by two steps: In the first step, the mapping is performed as a hypercube (at each level of the tree-hypercube, because each level i in the tree-hypercube has s^i nodes representing processing elements as a hypercube) nodes that are represented in BRGC order. The BRGC embedding has the property that any two adjacent numbers are mapped to neighboring nodes that can communicate directly through a link. Therefore, it uses only one link to route a message

from any node to its destination node. An embedding of a linear array into each level of tree-hypercube is a mapping of its linear array elements to hypercube nodes such that each linear array element is mapped to a distinct node. In the second step of mapping, the mapping is performed as tree nodes [21].

An algorithm for mapping a ring into tree-hypercube has been proposed in [3]. In that paper, the authors revealed that TH (s,d) has the ability to implement algorithms that use the communication pattern of linear array ring. They also showed that the tree-hypercube which combines the advantages of both trees and hypercubes, can emulate the topology of a ring, and also other topologies such as linear arrays, trees, and meshes. The algorithm for mapping Ring topology into tree-hypercube network includes: first, mapping a linear array at each level of tree-hypercube networks TH(s, d), in order to obtain the decimal address for each node at each level k in TH(s, d), using BRGC, then finds the start node in the last level. An example that explains the procedure of mapping a ring with 2^{d+1}-1 nodes into a TH (2, 3) is shown in Figure 3.1 [3].

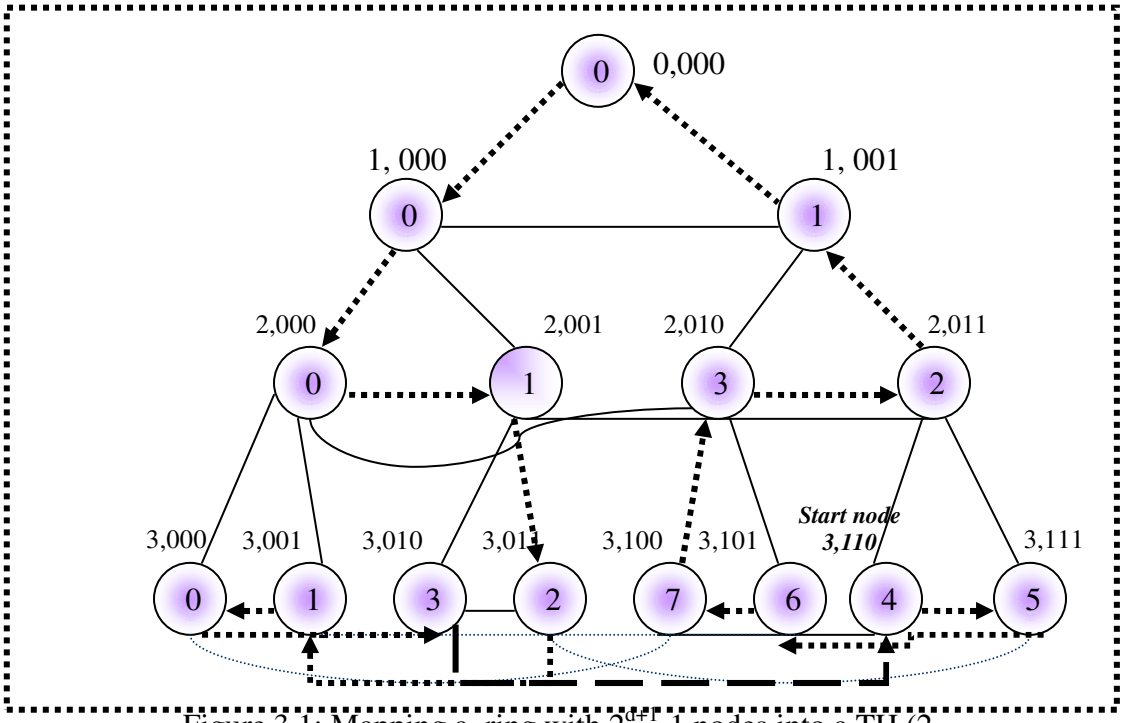

Figure 3.1: Mapping a ring with 2^{d+1}-1 nodes into a TH (2,

The following related work includes, the embedding of binary trees into hypercubes, embedding of combinational circuits in hypercubes, and the embedding of

fault-free cycles in crossed cubes with conditional link faults. In [8], embedding of binary trees into hypercubes is described, the method is based on an iterative embedding of binary trees into their graphs, and presents a class of binary trees, that allow a dilation two embedding into their optimal hypercubes. Then, concludes that there exist an embedding for an arbitrary binary tree.

In [29] the embedding torus in hexagonal honeycomb torus, states that a number of parallel algorithms admit a static torus-structured task graph. Hexagonal honeycomb torus (HHT) networks are regarded as promising candidates for interconnection networks. In order to efficiently execute a torus-structured parallel algorithm on an HHT, it is essential to map the tasks to processors so that the communication overhead is minimized. The study proves that a (3n, In) torus can be embedded into an nth-order HHT with dilation 3, congestion 4, expansion 1 and load factor 1. Consequently, a parallel algorithm with a (3n, 1n) torus task graph can be executed on an nth-order HHT efficiently.

An embedding of combinational circuits in hypercubes was considered in [23], such that the nodes of the scheme go into vertices of the hypercube and bundles of arcs go into similar bundles or the so-called transition trees of the hypercube having no common internal vertices. This embedding was considered since that in the recent years, it has become popular to realize Boolean functions by combinational circuits. In many cases, the further use of the scheme constructed requires its geometric realization, i.e., a certain embedding in one or another specific geometric structure. The role of such structure is often played by the unit n-dimensional cube [23]. In [25], the embedding of fault-free cycles in crossed cubes with conditional link faults, the researcher states that the crossed cube, which is a variation of the hypercube, possesses some properties that are superior to those of the hypercube. It is shown that with the assumption of each node incident with at least two fault-free links, an n-dimensional crossed cube with up to $2n$-5 link faults can embed, with dilation one, fault-free cycles of lengths ranging from 4 to $2n$. The assumption is meaningful, for its occurrence probability is very close to 1, and the result is optimal with respect to the number of link faults tolerated.

3.3 Tree-Hypercube Partitioning and the Embedding of Rings and Linear Arrays

Network partitionability is an important feature of networks underlying parallel machines, that support multitasking to serve several users or to run multiple tasks of the same algorithm simultaneously. To run multiple tasks in parallel, where each task can itself be parallel, the machine is partitioned into independent parts. To support the communication of each part, the network has to be partitioned into independent subnetworks. One important property of partionability is to have the partitions have the same topology as the original network [18].

To be able to partition tree-hypercube networks, extending the definition of TH is needed by allowing any h levels to be a tree-hypercube. The notation of TH's was modified to indicate the top and bottom level s of the h levels. The tree-hypercube is denoted by TH (t, d, s), the graph consisting of the levels t, t+1,... d of the TH(s, d) [18]. The structure of tree-hypercubes allows for vertical partitioning, and the flexibility of having any successive levels to be a TH allows for horizontal partitioning. Horizontal partitioning consists of splitting the TH (t, d, s) at level i and getting a TH (t, i, s) and a TH (i+1, d, s). Figure 3.2 shows an example of horizontal partitioning.

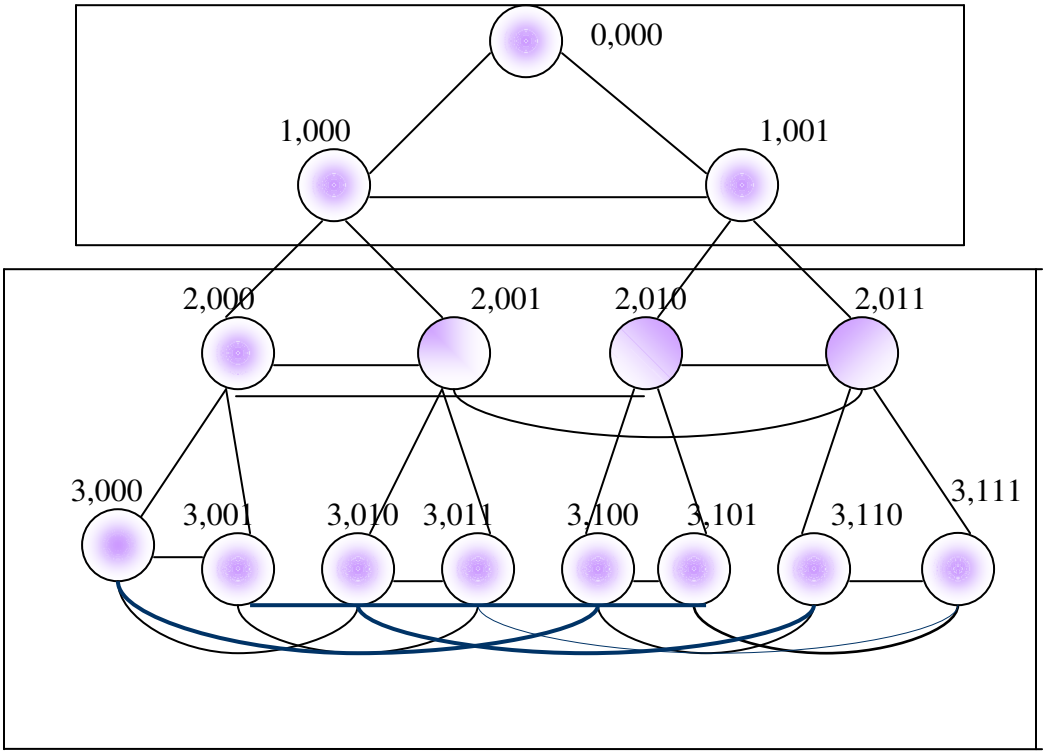

Figure 3.2: Horizontal Partitioning TH (0, 3, 2) into TH(0,1, 2) and TH(2, 3, 2).

While studying the embedding of linear array and ring into Tree-Hypercube, we have found that we can map the linear array, and ring just at each level of the tree-hypercube; using the horizontal partitioning presented previously. This gives the tree-hypercube a good advantage so that it can simulate other topologies not just by mapping nodes of the guest topology on the tree-hypercube from the root node to the leaves.

But mapping can be performed at each level of the Tree-Hypercube, by performing another horizontal partitioning on the partitioned tree-hypercube. Figure 3.3 illustrates examples of this embedding. The embedding in Figure 3.4 is based on the binary reflected gray code (BRGC), in which each node is connected with the other only if it differs with it at 1 bit position (binary address of the node). The same corresponds for embedding the ring into Tree-Hypercube, by connecting the last node with the start node. As shown in Figure 3.4.

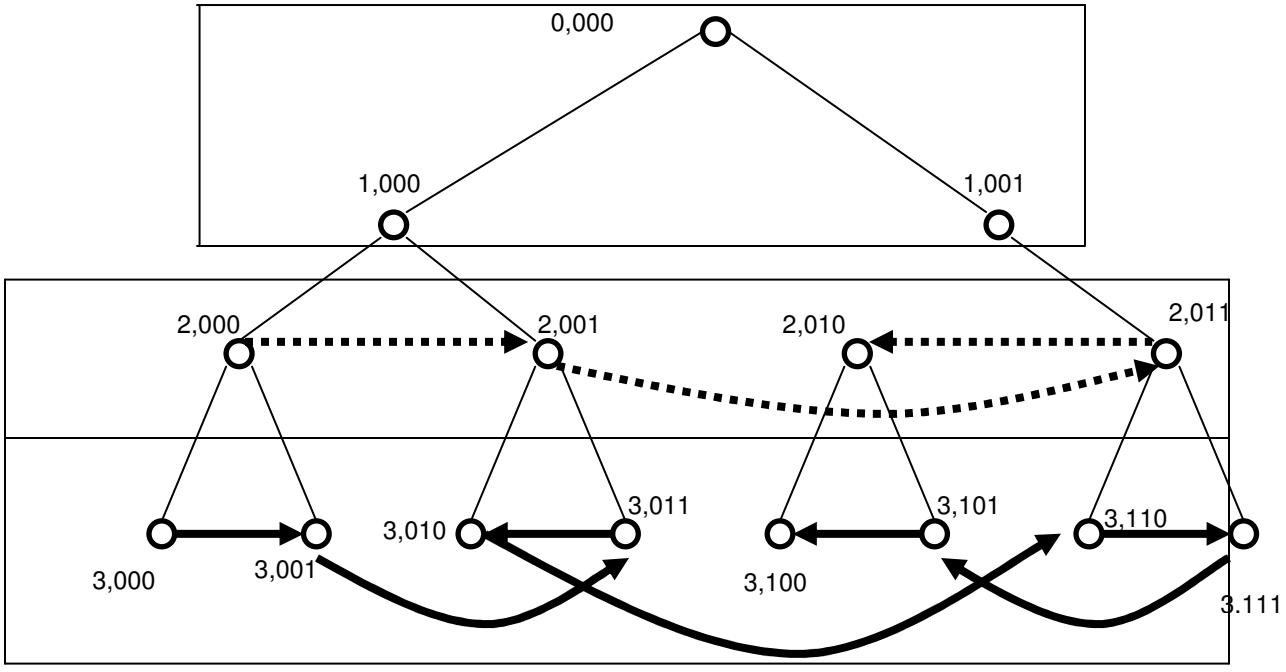

Figure 3.3: Embedding of both, four nodes and eight nodes Linear Array into TH(0, 1, 2), and TH(2, 3, 2).

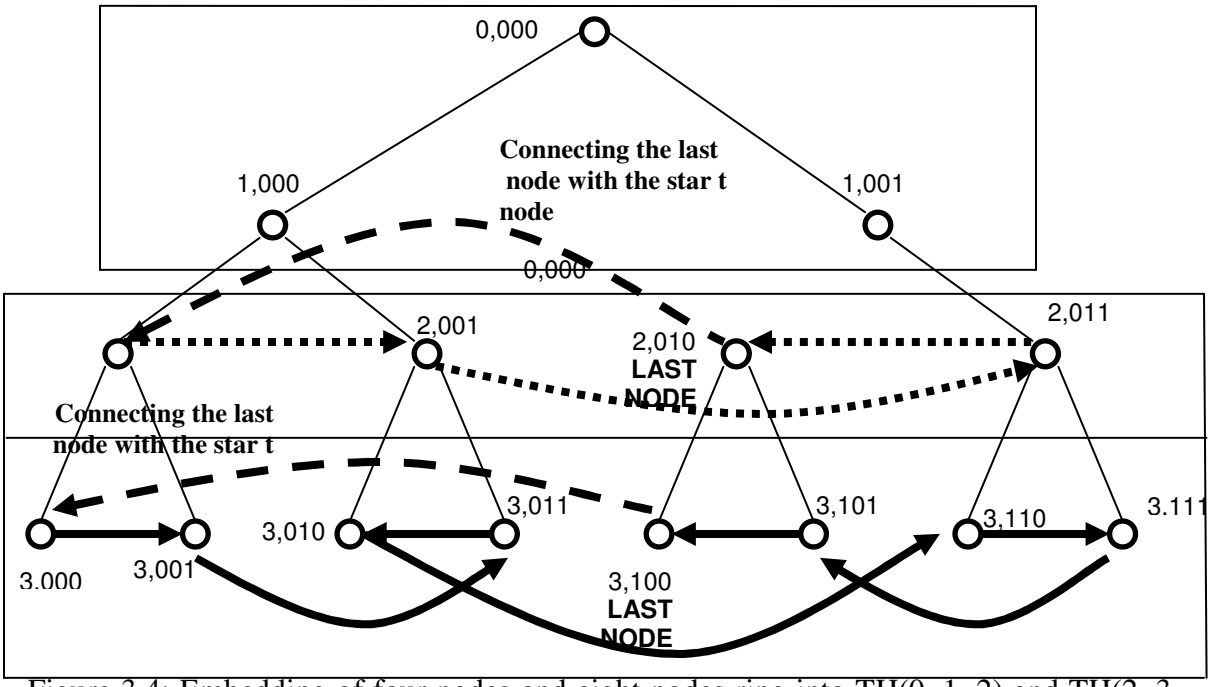

Figure 3.4: Embedding of four nodes and eight nodes ring into TH(0, 1, 2) and TH(2, 3, 2).

We can also use a whole part of the partition to map a linear array or ring on both tree edges and hypercube edges of the tree-hypercube, as illustrated in the following Figure.

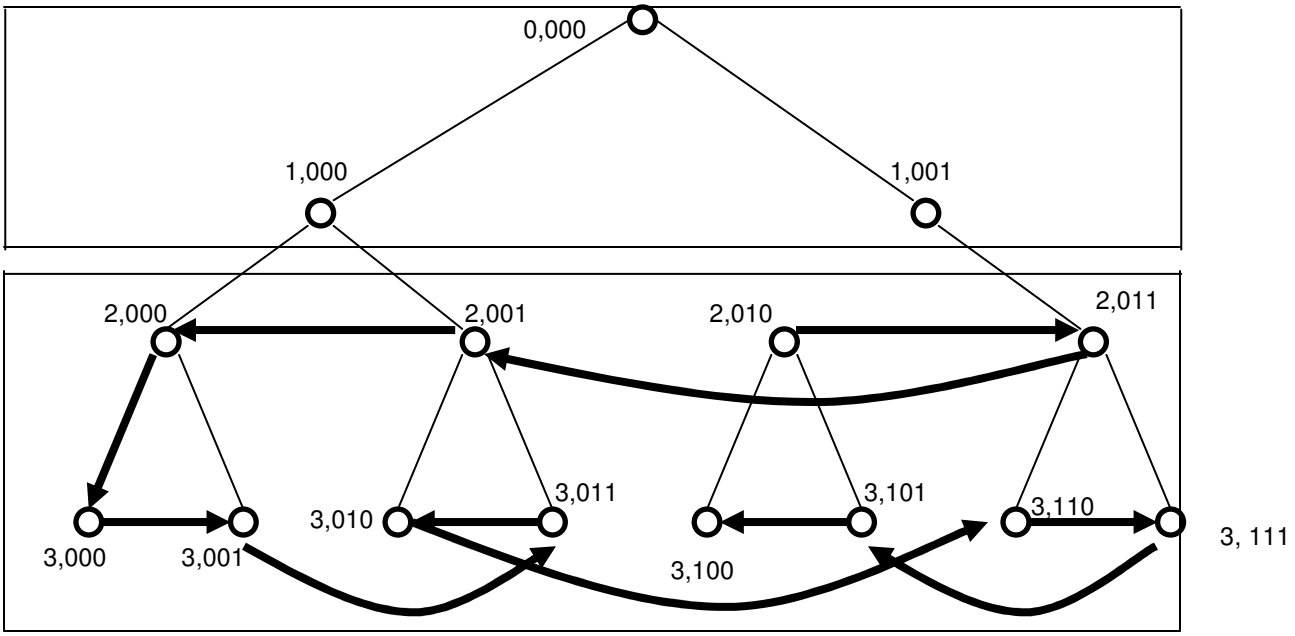

Figure 3.5: Mapping of 12 nodes linear array onto partitioned tree-hypercube TH(2,3,2)

Chapter Four

Embedding Hex-Cells into Tree-Hypercubes Networks

Hex-Cell is a new interconnection topology, which is suitable for large parallel computers, and can connect massive number of nodes with 3 links per node [24]. It has the ability to efficiently simulate programs written for architectures such as linear arrays, rings, and meshes. The hex-cell topology can easily embed these structures into it.

Hex-Cell can be embedded into tree-hypercube by mapping nodes of the hex-cell into nodes of the tree-hypercube, and edges of the hex-cell onto edges of the tree-hypercube; without having additional number of edges (i.e. dilation of one) and with lower expansion and congestion as possible. Since the smaller dilation of mapping, the shorter communication delay that the host graph (tree-hypercube) simulates the guest graph (hex-cell), and the smaller the expansion of mapping, the more efficient the processor utilization that the host graph (Tree-Hypercube) simulates the guest graph (hex-cell) [10]. In the next section we present the algorithm for mapping hex-cells into tree-hypercube TH (2,d).

4.1 The Proposed Algorithm for Embedding Hex-Cells into Tree-Hypercube TH(2,d).

We have designed an algorithm for mapping hex-cells of n nodes into tree-hypercube TH(2,d); such that every hex-cell with six nodes and six edges is mapped onto nodes and edges of the tree-hypercube TH(2,d) where d >= 2. The mapping has dilation one, congestion one, and expansion 1.1.

Algorithm 4.1 for embedding hex-cells into tree-hypercube TH(2,d), where d >= 2.

```
int   d,    //  depth of Tree-Hypercube
if  ( d >= 2 )
{
    For   i = 1   to   (d - 1)          // each level of the tree-hypercube
          For   j = 1   to   2^i   step 2       // each node at that level of the tree-hypercube.

              Node (i ,   j-1) in TH = Node 1 in HC.
              Node (i ,   j) in TH = Node 2 in HC.
              Node (i+1, 2*(j-1) + 3) in TH = Node 3 in HC
              Node (i+1, 2*(j-1) +2) in TH= Node 4 in HC.
              Node (i+1, 2*(j-1)) in TH =  Node 5 in HC.
              Node (i+1, 2*(j-1) + 1) in TH = Node 6 in HC.
              Connect node 1 with node 6 through a link.
         Next j
  Next i
 }
```

4.2 Examples on The Embedding Algorithm 4.1, and Discussion.

Example 4.1; embedding 1 Hex-Cell (Figure 4.2) into Tree-Hypercube TH(2,2); since d=2, with respect to the algorithm nodes of the Hex-Cell will be mapped as the following, Figure 4.1 illustrates the example:

node (1,0) in TH = Node 1 in HC
node (1,1) in TH = Node 2 in HC
node (2,3) in TH = Node 3 in HC
node (2,2) in TH = Node 4 in HC
node (2,0) in TH = Node 5 in HC
node (2,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

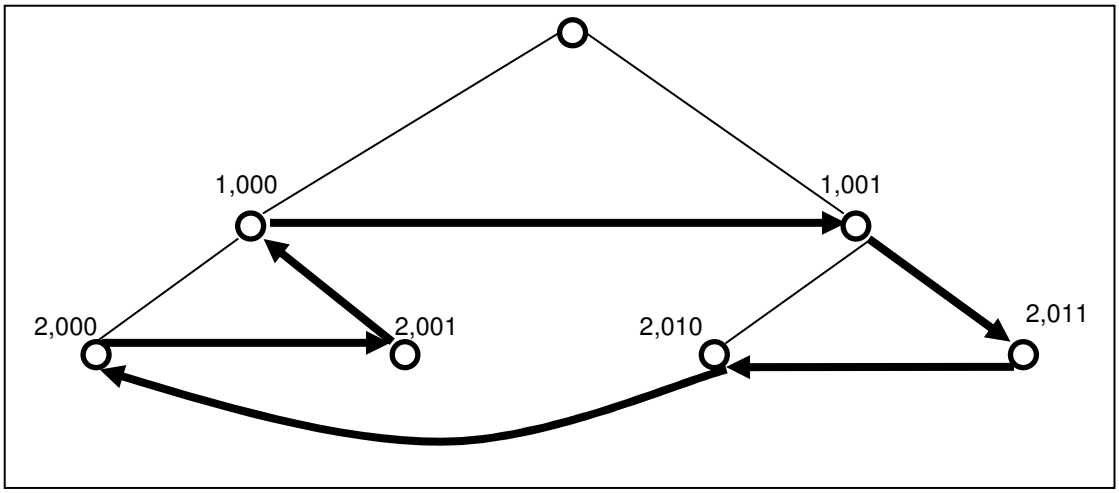

Figure 4.1: Embedding One Hex-Cell into Tree-Hypercube TH(2, 2).

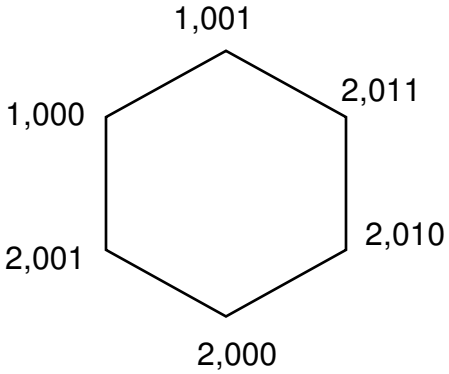

Figure 4.2: 1 Hex-Cell.

Example 4.2; embedding 3 Hex-Cells (Figure 4.4) into Tree-Hypercube TH(2,3); since d=3, with respect to the algorithm nodes of the Hex-Cell will be mapped as the following, Figure 4.3 illustrates the example:

At First Iteration (i = 1):
Node (1,0) in TH = Node 1 in HC
Node (1,1) in TH = Node 2 in HC
Node (2,3) in TH = Node 3 in HC
Node (2,2) in TH = Node 4 in HC
Node (2,0) in TH = Node 5 in HC
Node (2,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.
At Second Iteration (i = 2):
Node (2,0) in TH = Node 1 in HC
Node (2,1) in TH = Node 2 in HC
Node (3,3) in TH = Node 3 in HC
Node (3,2) in TH = Node 4 in HC

Node (3,0) in TH = Node 5 in HC
Node (3,1) in HC = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (2,2) in TH = Node 1 in HC
Node (2,3) in TH = Node 2 in HC
Node (3,7) in TH = Node 3 in HC
Node (3,6) in TH = Node 4 in HC
Node (3,4) in TH = Node 5 in HC
Node (3,5) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

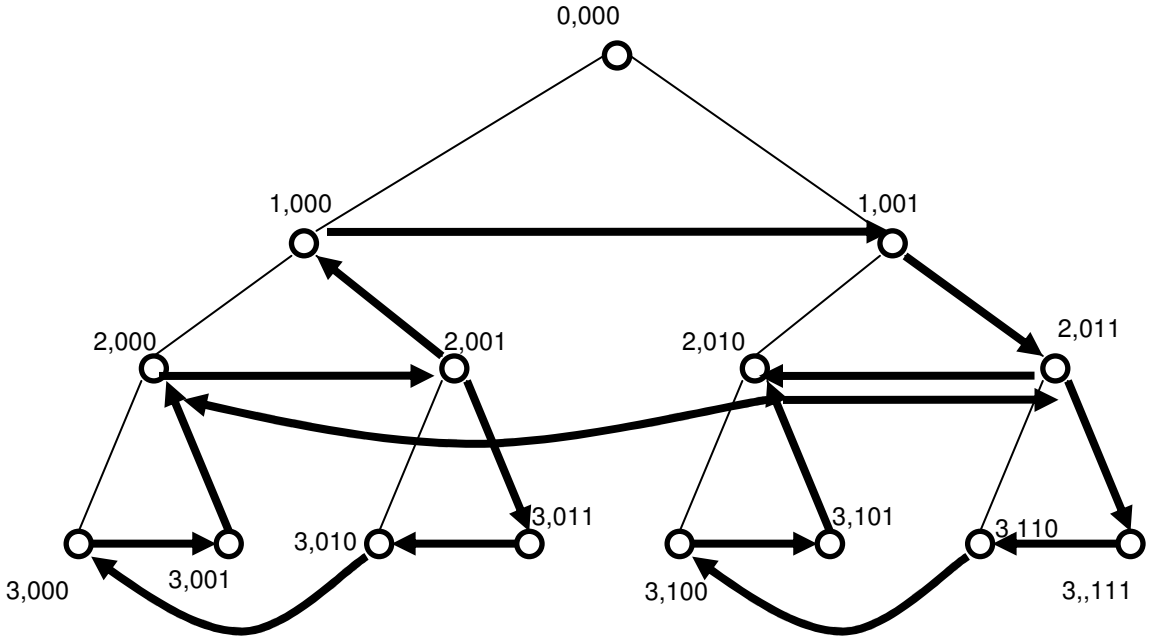

Figure 4.3 : Embedding Three Hex-Cells into Tree-Hypercube TH (2,3).

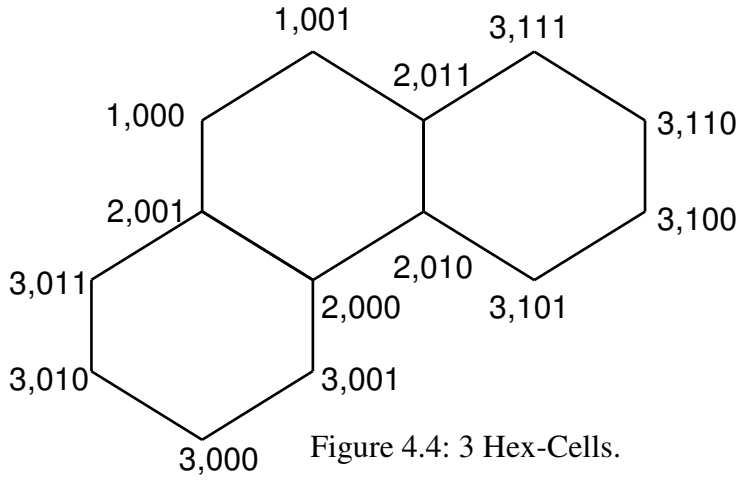

Figure 4.4: 3 Hex-Cells.

Example 4.3 ; embedding of seven hex-cells (Figure 4.6) into Tree-Hypercube TH(2,4); since d=4, with respect to the algorithm nodes of the Hex-Cell will be mapped as the following, Figure 4.5 illustrates the example:

At First Iteration (i = 1):
Node (1,0) in TH = Node 1 in HC
Node (1,1) in TH = Node 2 in HC
Node (2,3) in TH = Node 3 in HC
Node (2,2) in TH = Node 4 in HC
Node (2,0) in TH = Node 5 in HC
Node (2,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

At Second Iteration (i = 2):
Node (2,0) in TH = Node 1 in HC
Node (2,1) in TH = Node 2 in HC
Node (3,3) in TH = Node 3 in HC
Node (3,2) in TH = Node 4 in HC
Node (3,0) in TH = Node 5 in HC
Node (3,1) in HC = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (2,2) in TH = Node 1 in HC
Node (2,3) in TH = Node 2 in HC
Node (3,7) in TH = Node 3 in HC
Node (3,6) in TH = Node 4 in HC
Node (3,4) in TH = Node 5 in HC
Node (3,5) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

At Third Iteration (i = 3)
Node (3,0) in TH = Node 1 in HC
Node (3,1) in TH = Node 2 in HC
Node (4,3) in TH = Node 3 in HC
Node (4,2) in TH = Node 4 in HC
Node (4,0) in TH = Node 5 in HC
Node (4,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (3,2) in TH = Node 1 in HC
Node (3,3) in TH = Node 2 in HC
Node (4,7) in TH = Node 3 in HC
Node (4,6) in TH = Node 4 in HC
Node (4,4) in TH = Node 5 in HC
Node (4,5) in TH = Node 6 in HC

Connect Node 1 with node 6 through a link.

Node (3,4) in TH = Node 1 in HC
Node (3,5) in TH = Node 2 in HC
Node (4,11) in TH = Node 3 in HC
Node (4,10) in TH = Node 4 in HC
Node (4,8) in TH = Node 5 in HC
Node (4,9) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (3,6) in TH = Node 1 in HC
Node (3,7) in TH = Node 2 in HC
Node (4,15) in TH = Node 3 in HC
Node (4,14) in TH = Node 4 in HC
Node (4,12) in TH = Node 5 in HC
Node (4,11) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

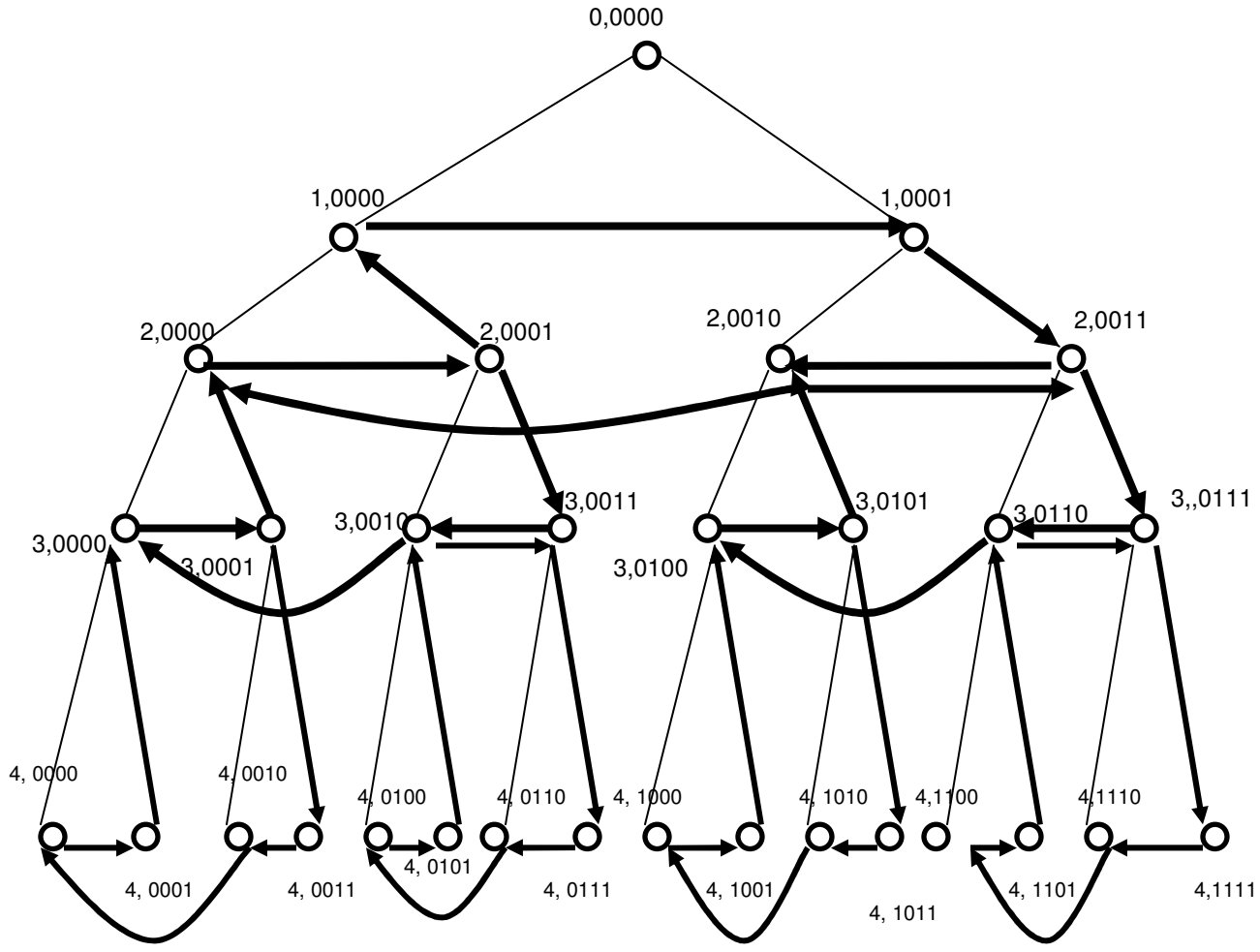

Figure 4.5: Embedding of seven Hex-Cells into Tree-Hypercube TH(2,4).

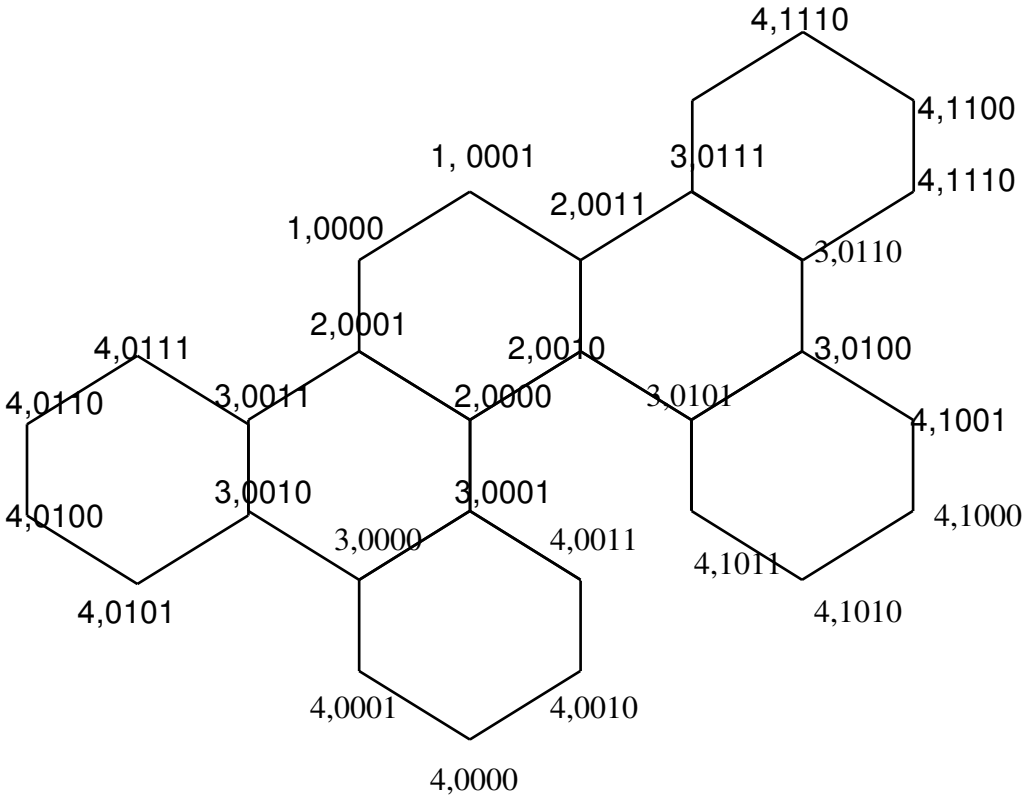

Figure 4.6: Seven Hex-Cells

Example 4.4 ; embedding of fifteen hex-cells (Figure 4.8) into Tree-Hypercube TH(2,5); since d=5, with respect to the algorithm nodes of the Hex-Cell will be mapped as the following, Figure 4.7 illustrates the example:

At First Iteration (i = 1):
Node (1,0) in TH = Node 1 in HC
Node (1,1) in TH = Node 2 in HC
Node (2,3) in TH = Node 3 in HC
Node (2,2) in TH = Node 4 in HC
Node (2,0) in TH = Node 5 in HC
Node (2,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

At Second Iteration (i = 2):
Node (2,0) in TH = Node 1 in HC
Node (2,1) in TH = Node 2 in HC
Node (3,3) in TH = Node 3 in HC
Node (3,2) in TH = Node 4 in HC
Node (3,0) in TH = Node 5 in HC

Node (3,1) in HC = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (2,2) in TH = Node 1 in HC
Node (2,3) in TH = Node 2 in HC
Node (3,7) in TH = Node 3 in HC
Node (3,6) in TH = Node 4 in HC
Node (3,4) in TH = Node 5 in HC
Node (3,5) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

At Third Iteration (i = 3)
Node (3,0) in TH = Node 1 in HC
Node (3,1) in TH = Node 2 in HC
Node (4,3) in TH = Node 3 in HC
Node (4,2) in TH = Node 4 in HC
Node (4,0) in TH = Node 5 in HC
Node (4,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (3,2) in TH = Node 1 in HC
Node (3,3) in TH = Node 2 in HC
Node (4,7) in TH = Node 3 in HC
Node (4,6) in TH = Node 4 in HC
Node (4,4) in TH = Node 5 in HC
Node (4,5) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (3,4) in TH = Node 1 in HC
Node (3,5) in TH = Node 2 in HC
Node (4,11) in TH = Node 3 in HC
Node (4,10) in TH = Node 4 in HC
Node (4,8) in TH = Node 5 in HC
Node (4,9) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (3,6) in TH = Node 1 in HC
Node (3,7) in TH = Node 2 in HC
Node (4,15) in TH = Node 3 in HC
Node (4,14) in TH = Node 4 in HC
Node (4,12) in TH = Node 5 in HC
Node (4,11) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

At Fourth Iteration (i = 4)
Node (4,0) in TH = Node 1 in HC
Node (4,1) in TH = Node 2 in HC
Node (5,3) in TH = Node 3 in HC
Node (5,2) in TH = Node 4 in HC
Node (5,0) in TH = Node 5 in HC
Node (5,1) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,2) in TH = Node 1 in HC
Node (4,3) in TH = Node 2 in HC
Node (5,7) in TH = Node 3 in HC
Node (5,6) in TH = Node 4 in HC
Node (5,4) in TH = Node 5 in HC
Node (5,5) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,4) in TH = Node 1 in HC
Node (4,5) in TH = Node 2 in HC
Node (5,11) in TH = Node 3 in HC
Node (5,10) in TH = Node 4 in HC
Node (5,8) in TH = Node 5 in HC
Node (5,9) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,6) in TH = Node 1 in HC
Node (4,7) in TH = Node 2 in HC
Node (5,15) in TH = Node 3 in HC
Node (5,14) in TH = Node 4 in HC
Node (5,12) in TH = Node 5 in HC
Node (5,13) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,8) in TH = Node 1 in HC
Node (4,9) in TH = Node 2 in HC
Node (5,19) in TH = Node 3 in HC
Node (5,18) in TH = Node 4 in HC
Node (5,16) in TH = Node 5 in HC
Node (5,17) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,10) in TH = Node 1 in HC
Node (4,11) in TH = Node 2 in HC
Node (5,23) in TH = Node 3 in HC
Node (5,22) in TH = Node 4 in HC
Node (5,20) in TH = Node 5 in HC
Node (5,21) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,12) in TH = Node 1 in HC
Node (4,13) in TH = Node 2 in HC
Node (5,27) in TH = Node 3 in HC
Node (5,26) in TH = Node 4 in HC
Node (5,24) in TH = Node 5 in HC
Node (5,25) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

Node (4,14) in TH = Node 1 in HC
Node (4,15) in TH = Node 2 in HC
Node (5,31) in TH = Node 3 in HC
Node (5,30) in TH = Node 4 in HC
Node (5,28) in TH = Node 5 in HC
Node (5,29) in TH = Node 6 in HC
Connect Node 1 with node 6 through a link.

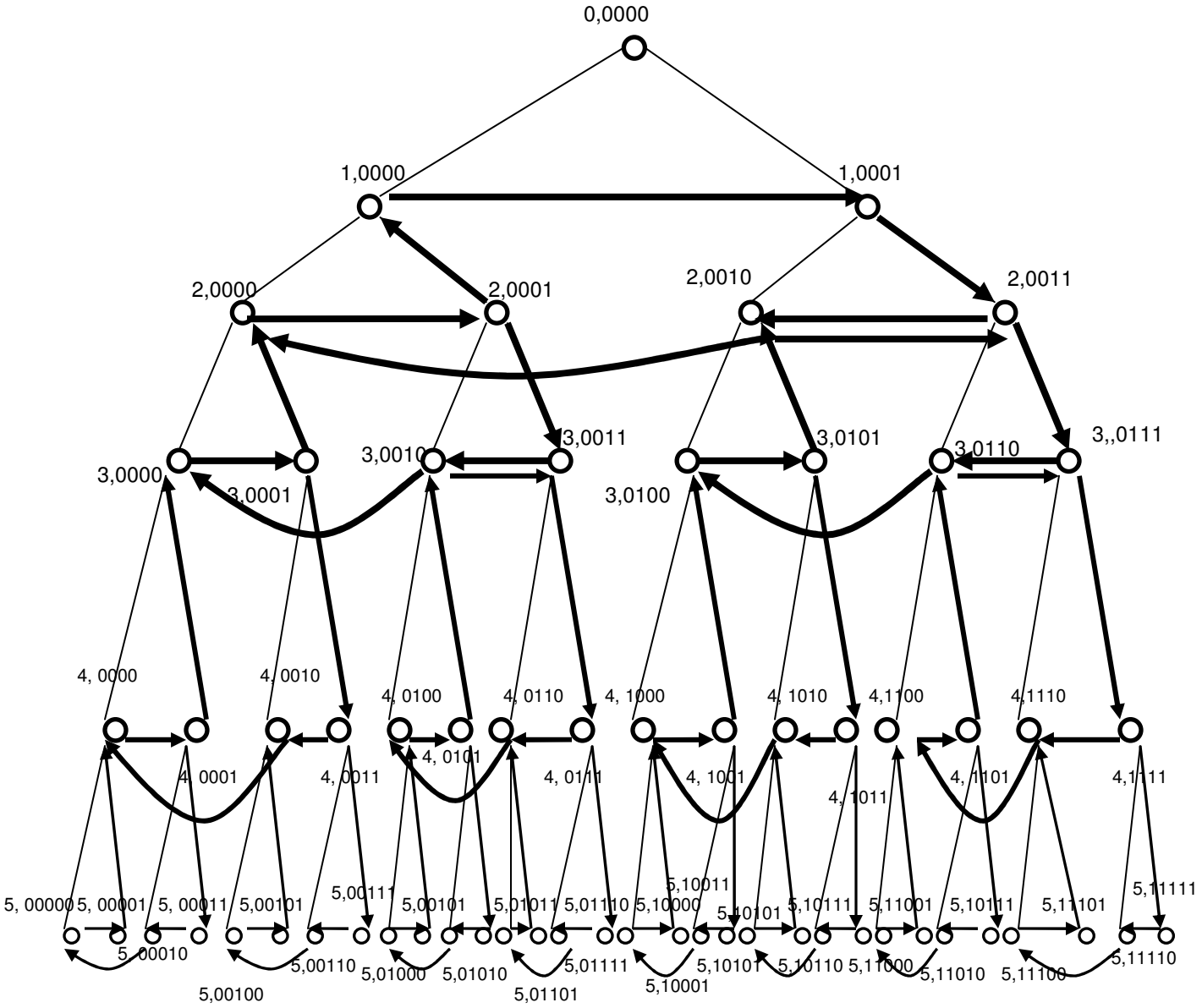

Figure 4.7: Embedding of fifteen Hex-Cells into Tree-Hypercube TH(2,5)

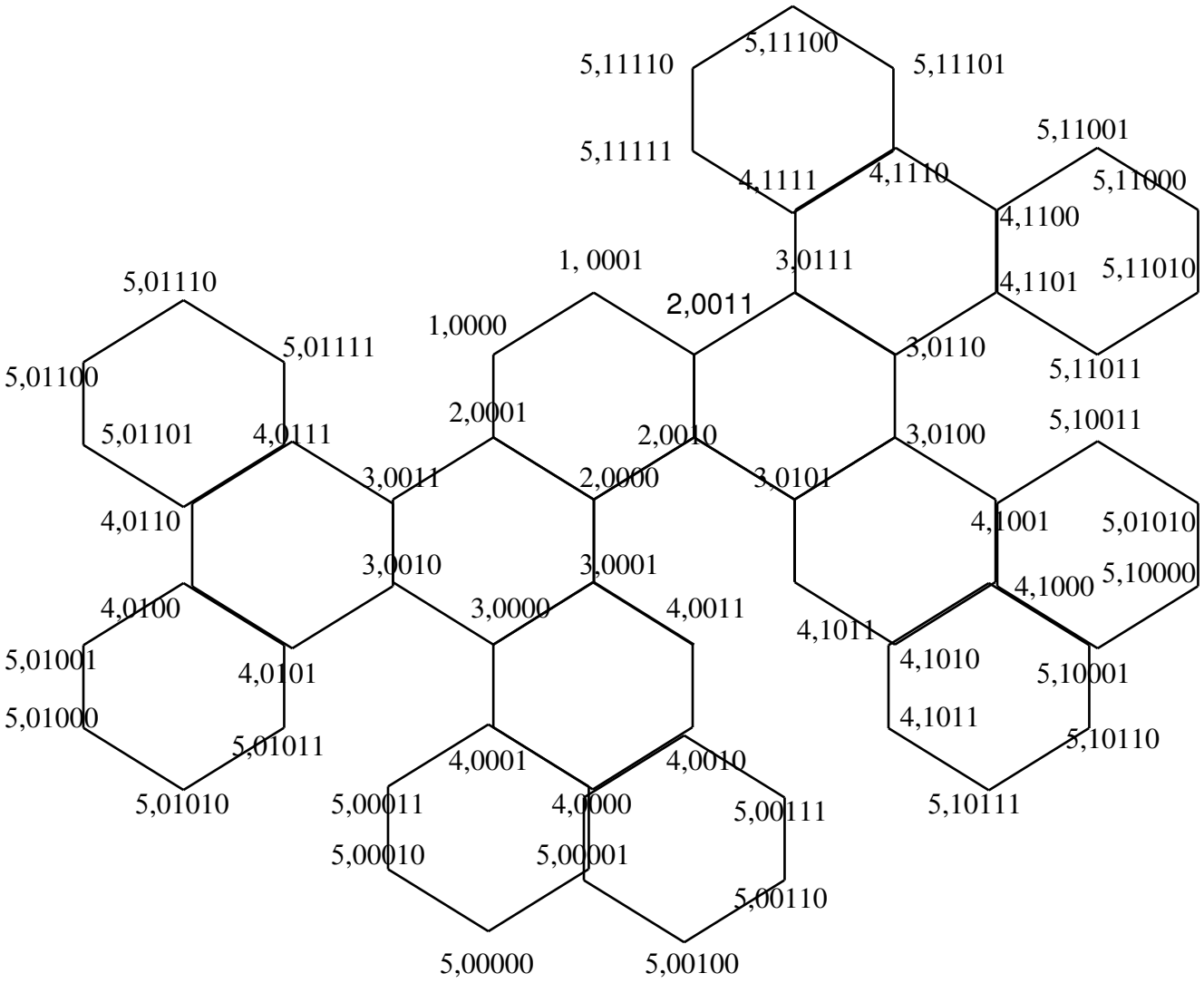

Figure 4.8: Embedding of fifteen Hex-Cells into Tree-Hypercube TH(2,5)

Note:

In algorithm 4.1 an embedding of irregular shape of hex-cells into tree-hypercube TH(2,d) where d >= 2 is performed. Embedding of regular shape of hex-cells into tree-hypercube can't be performed, requiring the structure of the tree-hypercube. And so as not to have additional number of links and not to increase the cost.

4.3 The Proposed Algorithm for Embedding Hex-cells HC(i)), where i = 1, 2 into Tree-Hypercube Network TH(2,d), where d = 2i.

We have also designed an algorithm for embedding hex-cells HC(i), where i = 1, 2 into tree-hypercube TH(2,d), where d = 2i ; without having additional number of edges when mapping edges of the hex-cells (i.e. dilation one and congestion one), with expansion 1.1 when mapping nodes of the hex-cell HC (1) into distinct nodes of the tree-hypercube TH(2,2). Also expansion 1.3 when mapping HC(2) into TH(2,4). The first part of the proposed algorithm performs mapping of HC(1) into TH(2,2). The second part performs mapping of HC(2) into TH(2,4). Mapping nodes of hex-cell into nodes of tree-hypercube is done by making a mathematical relation between the addresses of hex-cell nodes and the addresses of tree-hypercube nodes (i.e. node (i, j-1) in TH = node (i, 3*j mod 5) in HC).

Algorithm 4.2 for embedding hex-cells HC(i)), where i = 1, 2 into tree-hypercube network TH(2,d), where d = 2i:

```
int  d,   //  depth of Tree-Hypercube

If ( d= 2)

{      //  Mapping HC (1) into TH (2,2)

 node (1,0) in TH = node (1,1) in HC
 node (1,1) in TH = node (1,2) in HC
 node (2,3) in TH = node (1,3) in HC
 node (2,2) in TH = node ( 2,3) in HC
 node (2,0) in TH = node (2,2) in HC
 node (2,1)  in TH = node (2,1) in HC
Connect Node (1,0) with node (2,1) through a link.

}     // end of  the if  statement


If ( d=4)

{

     Map (d);     //  Mapping Function

 node (4,3) in TH = node (3,4) in HC
 node (4,11) in TH = node (3,5) in HC
 node (4,10) in TH = node (3,6) in HC
 node (4,14) in TH = node (4,5) in HC
 node (4,6) in TH = node (4,4) in HC
 node (4,2) in TH = node (4,3) in HC
```

Connect Node (4,3) with node (4,2) through a link.

} *// end of the if statement*

Map (d) *// Mapping Function*

{ *// Mapping HC(2) into TH (2,4)*

int i,j,

node (1,0) in TH = node (1,1) in HC
node (1,1) in TH = node (1,2) in HC
node (2,3) in TH = node (1,3) in HC
node (2,2) in TH = node (2,4) in HC
node (2,0) in TH = node (2,3) in HC
node (2,1) in TH = node (2,2) in HC
Connect Node (1,0) with node (2,1) through a link.

For i = 2 to (d - 2) *// each level of the tree-hypercube.*

For j = 1 to 2^i step 2 *// each node at that level of the tree-hypercube.*

node (i, j-1) in TH = node (i, 3*j mod 5) in HC

node (i, j) in TH = node (2*j mod 5, $2*j^2$ mod 15) in HC

node (i+1, 2*(j-1) + 3) in TH = node (2*j mod 5, j^2 mod 5) in HC

node (i+1, 2*(j-1) +2) in TH = node (2*i – j, j^3 mod 22) in HC

node (i+1, 2*(j-1)) in TH = node (3*j mod 7, (i*j) mod 9) in HC

node (i+1, 2*(j-1) + 1) in TH = node (3*j mod 7, j +2) in HC

Connect node (i+1, 2*(j-1) + 1) with node (i, j-1) through a link.

Next j

Next i

For i = 3 to (d - 1) *// each level of the tree-hypercube.*

For j = 1 to 2^{i-1} + 1 step 4 *// each node at that level of the tree-hypercube.*

node (i, j-1) in TH = node (((3*j +i) mod 7) /2, 2*j mod 4) in HC

node (i, j) in TH = node (((3*j +i) mod 7) /2, 3*j mod 10) in HC

node (i+1, 2*(j-1) + 3) in TH = node (i, ((i-1)*(j+1)) mod 7) in HC

node (i+1, 2*(j-1) +2) in TH = node ((i+ j)mod 5, (i* j) mod 9) in HC

node (i+1, 2*(j-1)) in TH = node ((i+ j)mod 5, $2*j^2$ mod 43) in HC

node (i+1, 2*(j-1) + 1) in TH = node ((i+ j)mod 6, j^2 mod 18) in HC

Connect node (i, j-1) with node (i+1, 2*(j-1) + 1) through a link.

Next j

Next i

Return ;

} *// end of Mapping Function.*

4.4 Examples on the Embedding Algorithm 4.2, and Discussion

Example 4.5; mapping hex-cell HC(1) (Figure 4.9) into tree-hypercube TH(2,2); since d=2, with respect to the algorithm nodes of the hex-cell will be mapped as the following, Figure 4.10 illustrates the example:

node (1,0) in TH = node (1,1) in HC
node (1,1) in TH = node (1,2) in HC
node (2,3) in TH = node (1,3) in HC
node (2,2) in TH = node (2,3) in HC
node (2,0) in TH = node (2,2) in HC
node (2,1) in TH = node (2,1) in HC
Connect Node (1,0) with node (2,1) through a link.

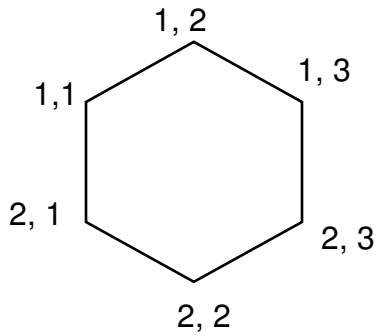

Figure 4.9: HC (1).

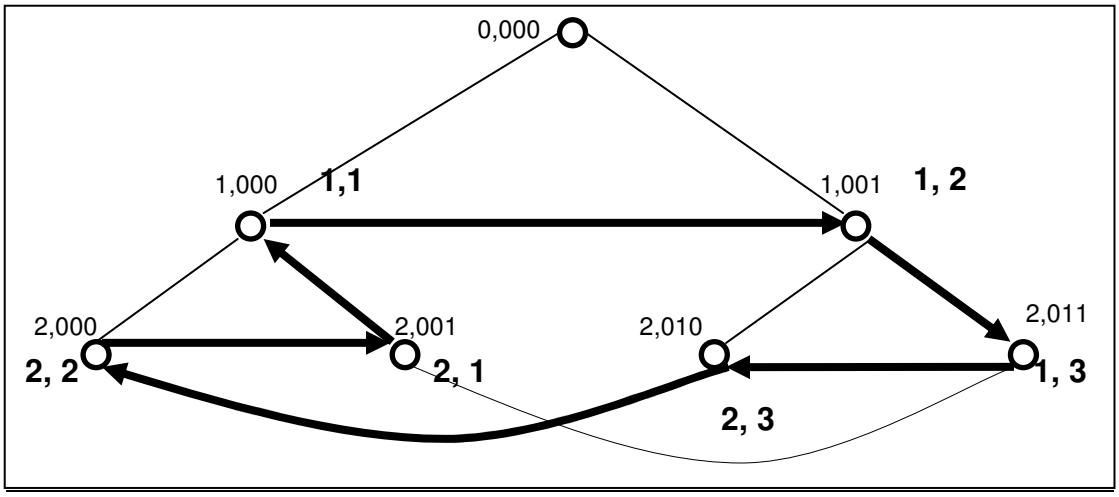

Figure 4.10: Mapping of HC (1) into TH(2, 2).

Example 4.6; mapping of hex-cell HC(2) (Figure 4.11) into tree-hypercube TH(2,4); then d=4, with respect to the algorithm nodes of the hex-cell will be mapped as the following, Figure 4.12 illustrates the example.

node (1,0) in TH = node (1,1) in HC
node (1,1) in TH = node (1,2) in HC
node (2,3) in TH = node (1,3) in HC
node (2,2) in TH = node (2,4) in HC
node (2,0) in TH = node (2,3) in HC
node (2,1) in TH = node (2,2) in HC
Connect Node (1,0) with node (2,1) through a link.

i = 2 ; j = 1
node (2,0) in TH = node (2,3) in HC
node (2,1) in TH = node (2,2) in HC
node (3,3) in TH = node (2,1) in HC
node (3,2) in TH = node (3,1) in HC
node (3,0) in TH = node (3,2) in HC
node (3,1) in TH = node (3,3) in HC
Connect Node (2,0) with node (3,1) through a link.

i = 2 ; j = 3
node (2,2) in TH = node (2,4) in HC
node (2,3) in TH = node (1,3) in HC
node (3,7) in TH = node (1,4) in HC
node (3,6) in TH = node (1,5) in HC
node (3,4) in TH = node (2,6) in HC
node (3,5) in TH = node (2,5) in HC
Connect Node (2,2) with node (3,5) through a link.

i=3; j=1;
node (3,0) in TH = node (3,2) in HC
node (3,1) in TH = node (3,3) in HC
node (4,3) in TH = node (3,4) in HC
node (4,2) in TH = node (4,3) in HC
node (4,0) in TH = node (4,2) in HC
node (4,1) in TH = node (4,1) in HC
Connect Node (3,0) with node (4,1) through a link.

i=3; j=5;
node (3,4) in TH = node (2,6) in HC
node (3,5) in TH = node (2,5) in HC
node (4,11) in TH = node (3,5) in HC
node (4,10) in TH = node (3,6) in HC
node (4,8) in TH = node (3,7) in HC
node (4,9) in TH = node (2,7) in HC
Connect Node (3,4) with node (4,9) through a link.
node (4,3) in TH = node (3,4) in HC
node (4,11) in TH = node (3,5) in HC
node (4,10) in TH = node (3,6) in HC
node (4,14) in TH = node (4,5) in HC
node (4,6) in TH = node (4,4) in HC
node (4,2) in TH = node (4,3) in HC
Connect Node (4,3) with node (4,2) through a link.

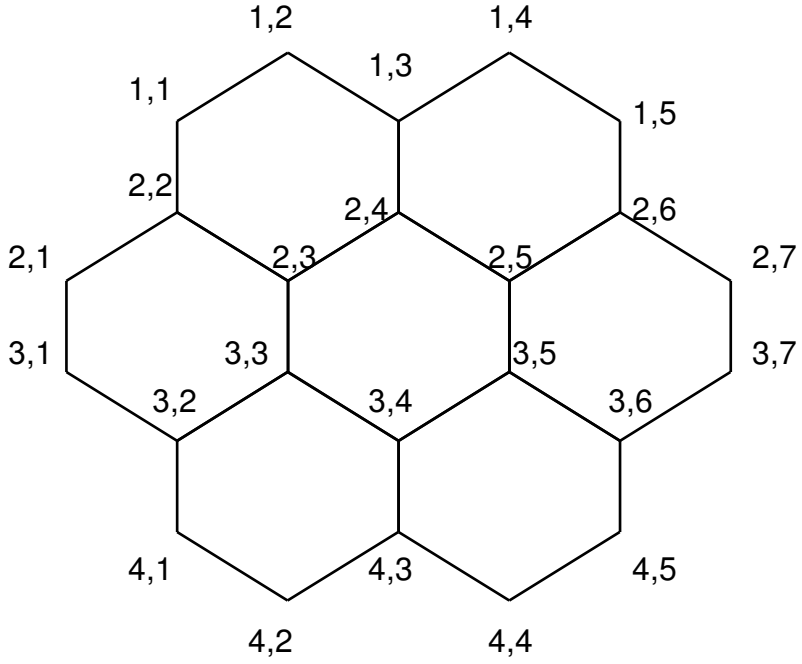

Figure 4.11: HC (2).

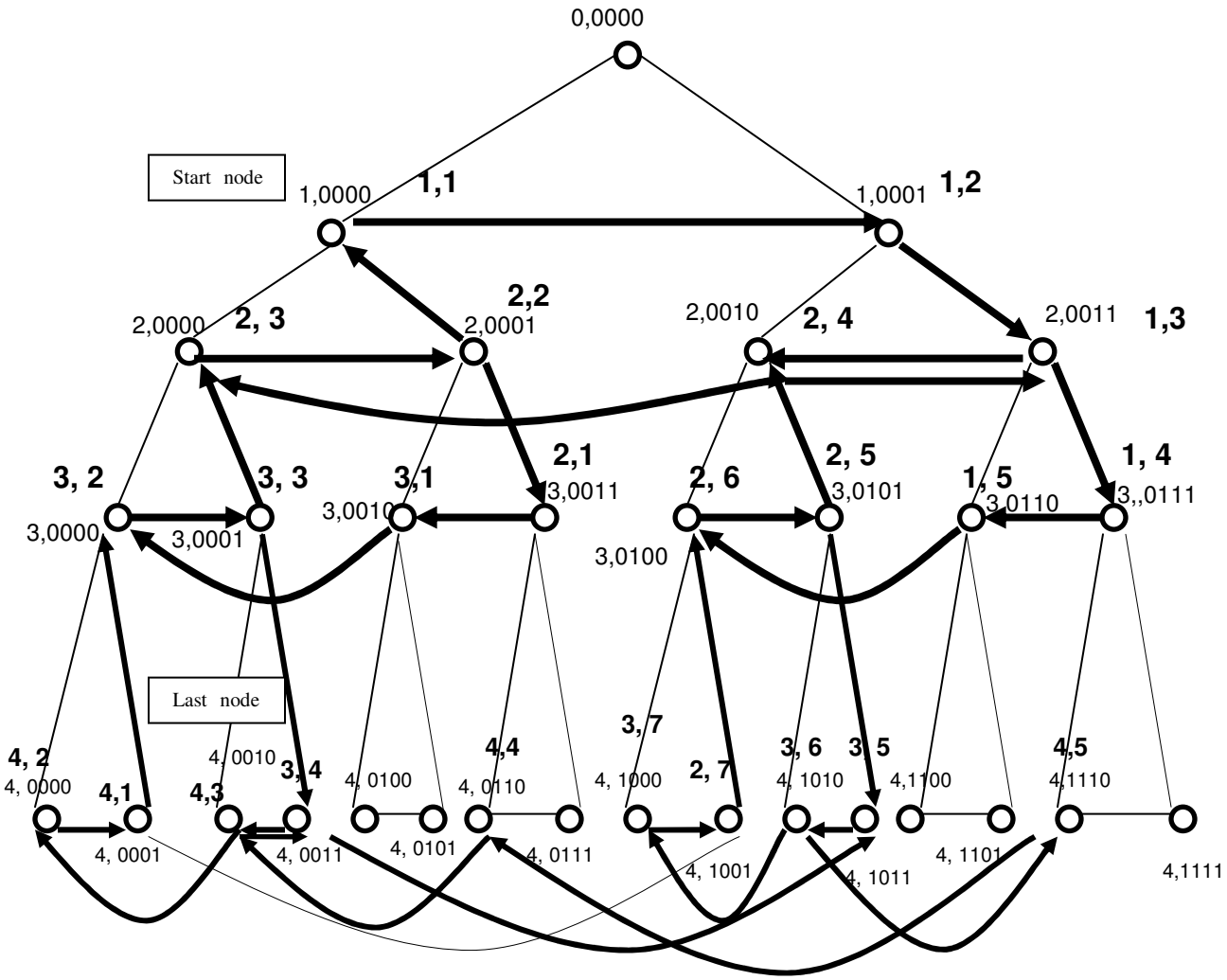

Figure 4.12: Mapping of Hex-Cells HC (2) into Tree-Hypercube TH(2,4)

In Example 4.5, a mapping of six nodes of HC(1) into TH(2,2) containing 7 nodes was performed. Here, the mapping is said to have a dilation one, since each edge in the hex-cell is mapped into only a single edge in the tree-hypercube, congestion one, since there is no more than one edge in the guest graph (HC (1)) mapped into any edge in the host graph (TH (2,2)). Also the expansion is 1.1, since the ratio of the maximum number of nodes in both graphs (hex-cell and tree-hypercube) is 7/6 = 1.1.

In Example 4.6, a mapping of HC (2) containing of 24 nodes into TH (2,4) consisting of 31 nodes, was performed. Here, the mapping is said to have a dilation one, since each edge in the hex-cell is mapped into only a single edge in the tree-hypercube, and congestion one, since there is no more than one edge in the guest graph (HC(2)) mapped into any edge in the host graph (TH (2,4)). Also the expansion is 1.3 since the ratio of the maximum number of nodes in both graphs (hex-cell and tree-hypercube) is 31/24 = 1.3. In both examples, when mapping distinct nodes of the hex-cells into distinct nodes in the tree-hypercube, a mapping of edges connecting these distinct nodes is performed.

4.5 A note to the cases HC(i), where i > 2

In the following table we list the total number of nodes in HC(i), TH(2,2i), and the number of unused nodes in the embedding of hex-cell into tree-hypercube.

i	Total number of nodes in HC(i)	Total number of nodes in TH(2,2i)	Number of unused nodes in embedding
1	6	7	1
2	24	31	7
3	54	127	73
4	96	511	415
5	150	2047	1897

Table 4.1: Number of Unused nodes in embedding hex-cells into tree-hypercube.

From the above table we notice that the number of unused nodes increases massively, this leads to the idea of using other architectures such as (Client Servers), or embedding of hex-cells into other topologies such as meshes, and stars.

4.6 Implementation and Application

4.6.1 Routing Mechanism

The most fundamental function of interconnection network is communication, or data routing. Routing is the process of sending messages from source nodes to other nodes in the network [20].

Mapping hex-cells into tree-hypercube has a benefit in communication or data routing between nodes of hex-cells. Since the maximum shortest path *(diameter*) between any two processing nodes in a hex-cell is $4\sqrt{(N/6)} - 1$, where N is the total number of nodes in the hex-cell, we can compute the total number of nodes N in a hex-cell by the equation $N= 6*d^2$ [24]. So, HC (1) has 6 nodes, and the maximum shortest path between any two nodes in HC(1) = 3. On the other hand in the tree-hypercube, the diameter is computed by the equation $d*log\ s$, where d and s are the depth and degree of the tree-hypercube, respectively. The total number of nodes in TH (s, d) is computed by the equation $N = (s^{d+1} -1) / (s-1)$ [19]. The TH (2, 2) which has 7 nodes and its diameter = 2. So we can say that the diameter after embedding minimized one edge. HC(2) has 24 nodes, and the maximum shortest path between any two nodes = 7, but in TH (2, 4) which has 31 nodes and its diameter = 4. So here the diameter after embedding minimized 3 edges.

According to the above calculations of diameter for both hex-cell and tree-hypercube, we can see that the diameter of the tree-hypercube is less than that of the hex-cell. So by mapping nodes and edges of hex-cell onto nodes and edges of tree-hypercube, nodes of the hex-cell can communicate with other nodes with lower diameter. This increases the performance of the system. As an example, if node (1,1) in HC(1) sends a message to node (2,3) in the same HC(1), it needs to traverse along a paths which consist of three edges (1,1) - (2,1) – (2,2) – (2,3) as shown in Figure 4.13. But after mapping HC(1) into TH(2,2), node 1,1 which is mapped to node (1, 000) in TH (2,2) can send a message to node (2,3) which is mapped to node (2, 010)

in TH(2,2) by traversing a path that consists of only two edges (1,000) - (2,000) - (2,010) which is (1,1) - (2.2) - (2,3). Figure 4.14 illustrates this example.

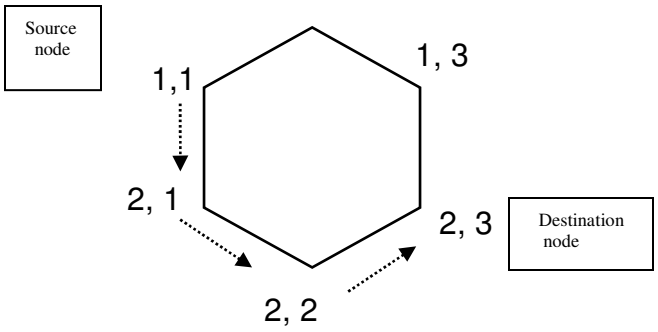

Figure 4.13: The path from node (1,1) to node (2,3) before mapping.

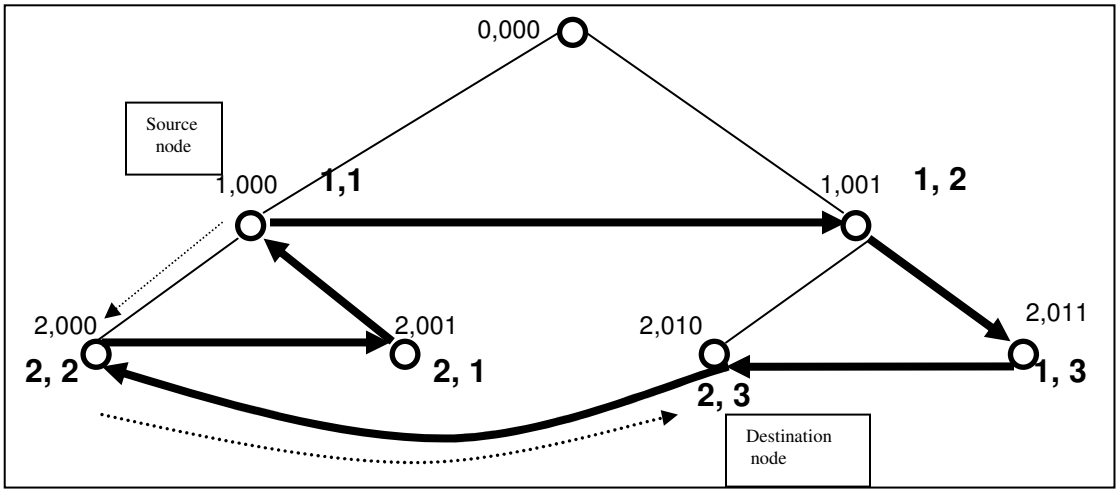

Figure 4.14: The path from node (1,1) to node (2,3) after mapping.

Another example, if node 1,2 in HC(2) sends a message to node 4,2 in the same HC(2), the shortest path that it would traverse consists of seven edges (1,2) -(1,3) - (2,4) - (2,3) - (3,3) - (3,2) - (4,1) - (4,2), as shown in Figure 4.15. But after mapping HC(2) into TH(2,4), node (1,2) which is mapped to node (1,0001) in TH(2,4) can send a message to node (4,2) which is mapped to node (4,0000) by traversing a path that consists of only four edges (1,0001) - (2,0010) - (3,0100) - (4,1000)- (4,0000) which is (1.2) - (2,4) - (2,6) -(3,7) - (4,2), as shown in Figure 4.16.

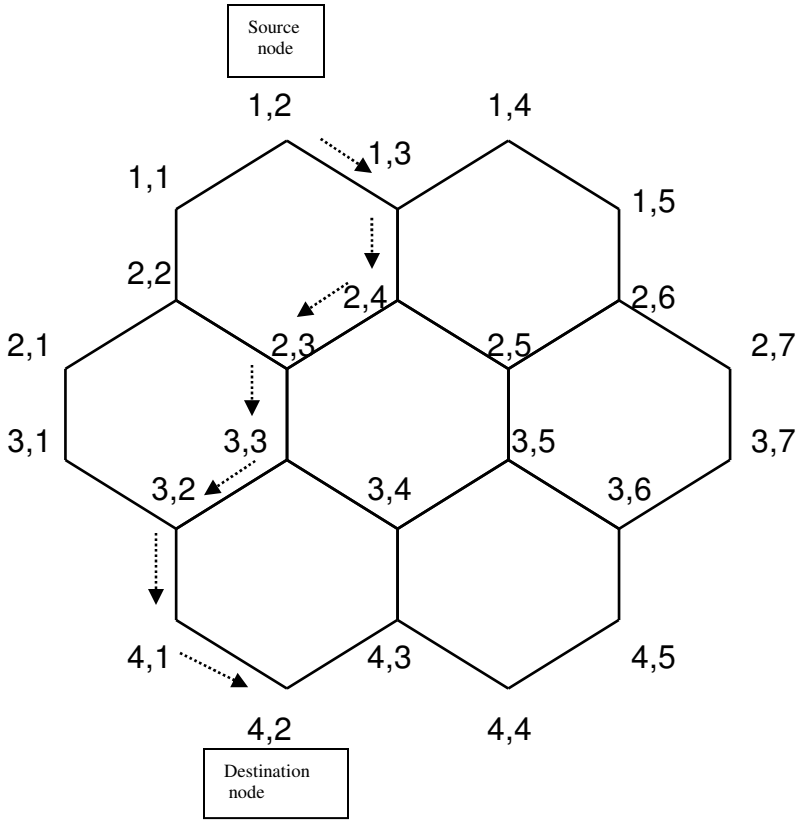

Figure 4.15: The path from node (1,2) to node (4,2) before mapping.

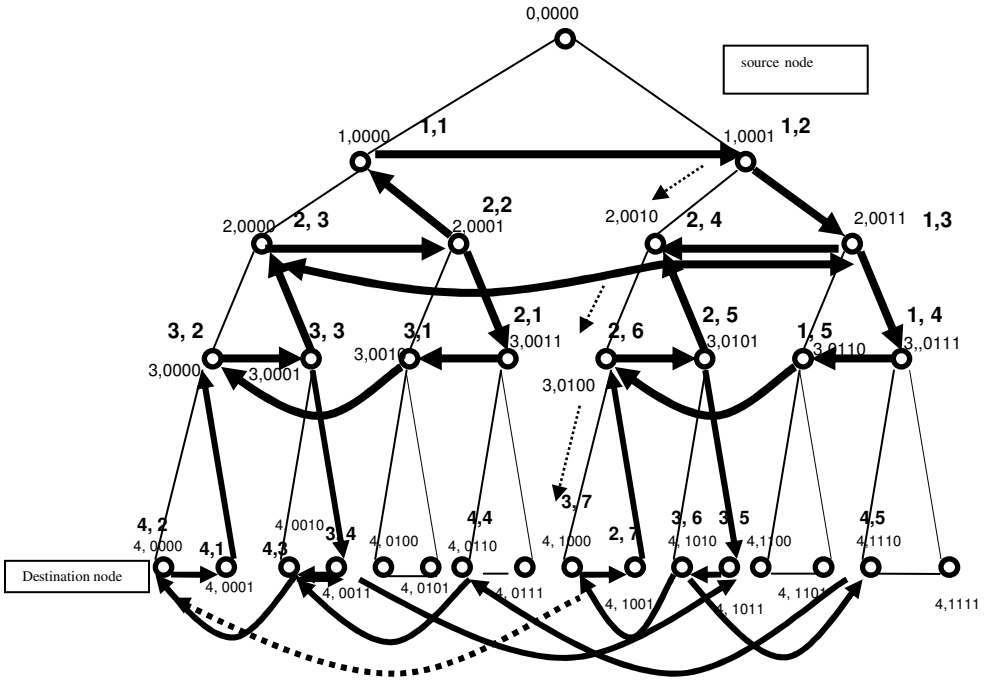

Figure 4.16: The path from node (1,2) to node 4,2, after mapping.

4.6.2 Application Program

We have developed an application program based on the routing algorithm of the hex-cell [24], and the routing algorithm of the tree-hypercube [19]. The programming language that we have used is C# programming language. The program shows a comparison of the shortest path between any source and destination nodes in hex-cell and tree-hypercube graphs, before and after embedding. The shortest path between any two nodes in a hex-cell before embedding consists of up to 7 edges, but after embedding a hex-cell into a tree-hypercube, the shortest path between any two nodes consists of up to 4 edges, this means that the shortest path have been minimized about 3 edges, this increases the performance of the parallel system. An interface of the application program is shown in Figures 4.17, 4.18, and 4.19.

In Figure 4.17 a message shows the path ((1,1)- (2,1)- (2,2)) between node (1,1) and node (2,2) in hex-cell before embedding, this path contains two edges. On the other hand Figure 4.18 shows a message of the path between same nodes ((1,1) and (2,2)) but after embedding, here the path is ((1,1)- (2,2)) which contain only one edge.

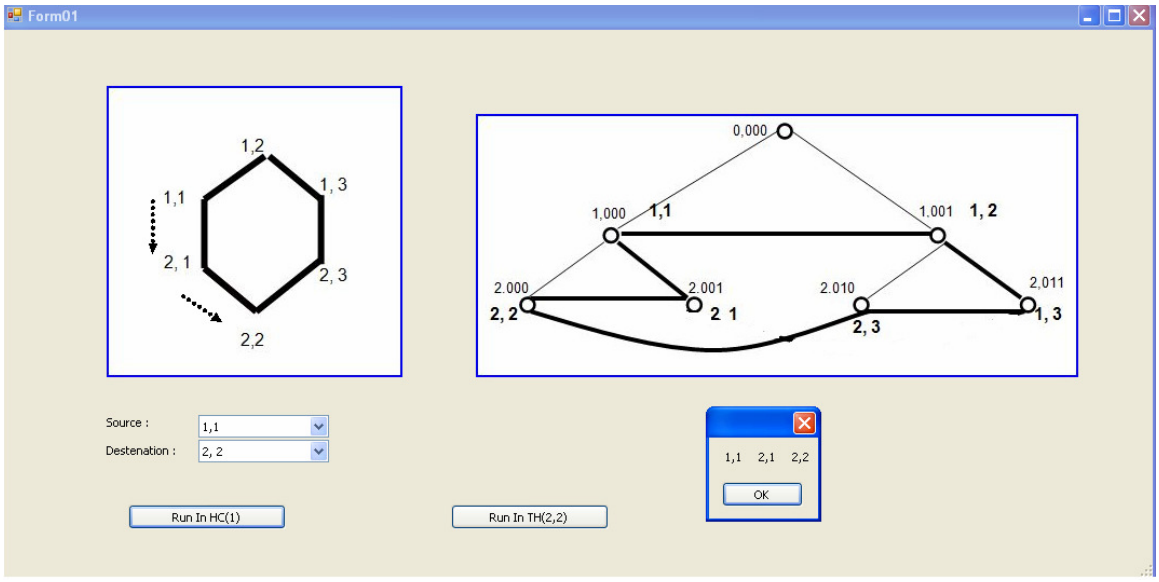

Figure 4.17: A message shows the shortest path between node (1.1) and node (2.2) in HC(1) before embedding.

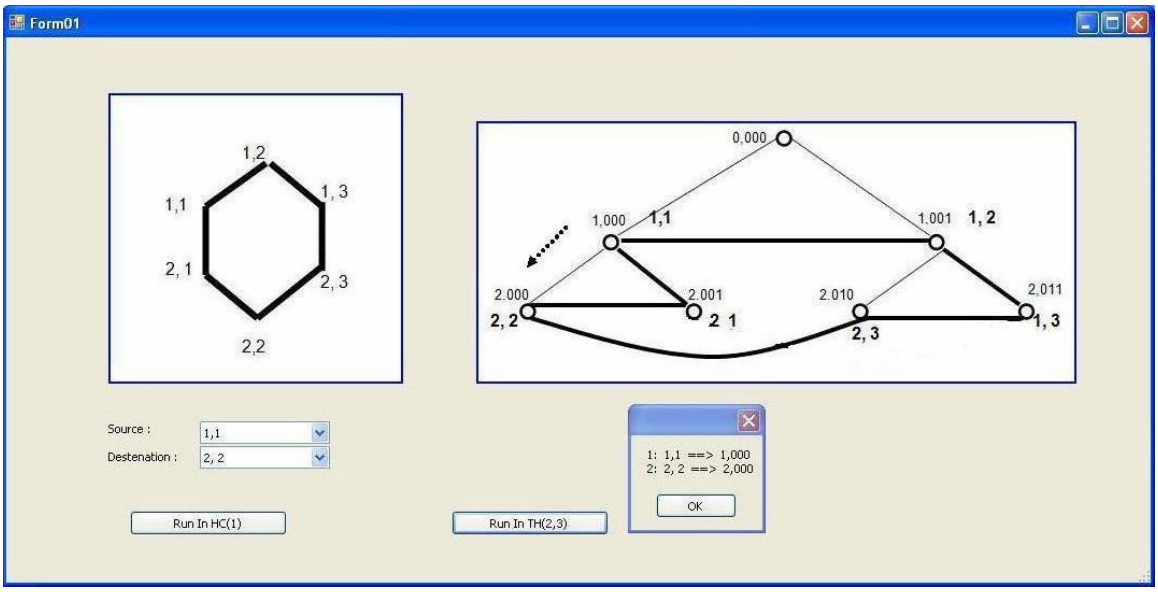

Figure 4.18: A message shows the shortest path between node (1.1) and node (2.2), after embedding HC(1) into TH(2,2).

In Figure 4.19 the interface shows a message of the path between node (1,3) and node (4,4) before embedding ((1,3)-(2,4)-(2,5)-(3,5)-(3,4)-(4,3)-(4,4)) and after embedding ((1,3)-(1,4)-(4,5)-(4,4)). We can see here that the path after embedding minimized three edges.

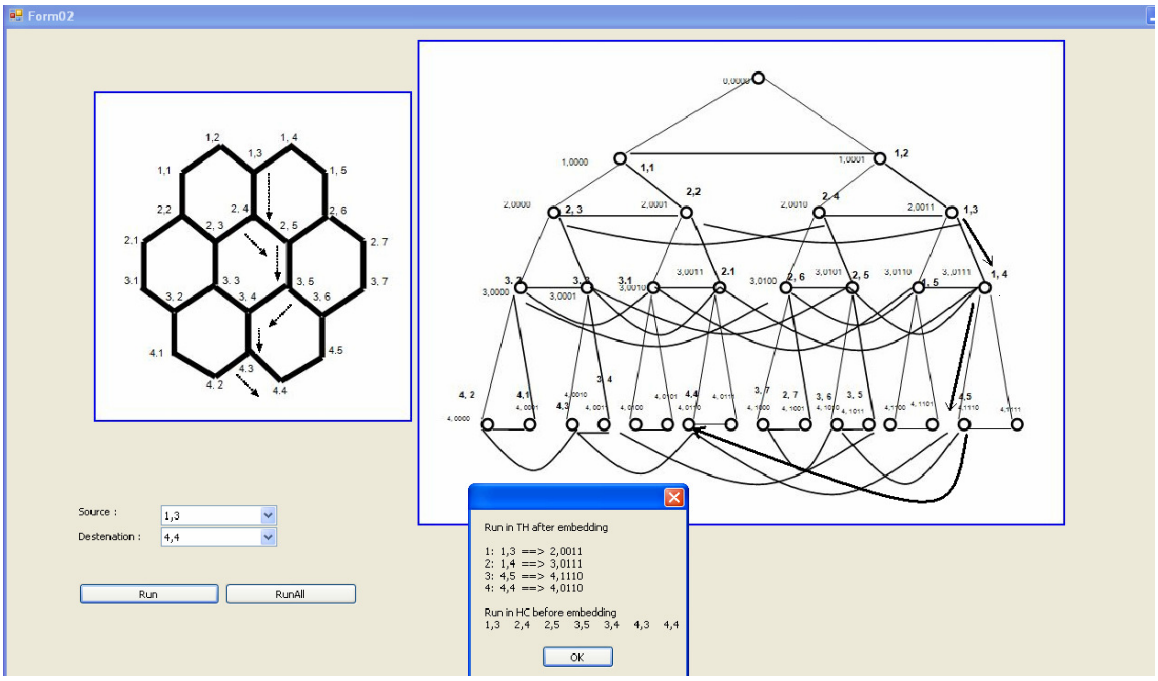

Figure 4.19: A message shows the shortest path between node (1.3) and node (4.4), after embedding HC(2) into TH(4,4), and before embedding.

Chapter Five

Contribution and Conclusion

5.1 Contribution

The problem of mapping interconnection topologies into tree-hypercube network has not received much attention from researchers. The embedding of linear array and ring into tree-hypercube have been well studied in literature, and also other embeddings into hypercube such as binary trees and star graphs. However, there has been no work reported so far on hex-cells embedding. In our thesis, We have designed an algorithm for embedding hex-cells of n nodes into tree-hypercube TH(2,d) where d >= 2. The mapping has dilation one, congestion one, and expansion 1.1. In the algorithm an embedding of irregular shape of hex-cells into tree-hypercube TH(2,d) where d >= 2 is performed. Embedding of regular shape of hex-cells into tree-hypercube can't be performed, requiring the structure of the tree-hypercube. And so as not to have additional number of links and not to increase the cost. We have also designed an algorithm for mapping hex-cells HC(i), where i = 1, 2 into tree-hypercube TH(2,d), where d = 2i.

We have developed an application program based on the routing algorithm of the hex-cell, and the routing algorithm of the Tree-Hypercube, the program shows a comparison of the shortest path between any source and destination nodes in hex-cell and tree-hypercube graphs, before and after embedding. The shortest path between any two nodes in hex-cell before embedding consists of up to 7 edges, but after embedding hex-cell into Tree-Hypercube, the shortest path between any two nodes consists of up to 4 edges, this means that the shortest path have minimized about 3 edges, this increases the performance of the parallel system.

We have also suggested a new scheme for embedding linear arrays and rings, that have been studied in literature onto tree-hypercube. The scheme is based on the partitionability property of the tree-hypercube, in order to map linear arrays and rings at each level of the

tree-hypercube. Also, we can use a whole partition of the tree-hypercube to map linear arrays and rings on both tree edges, and hypercube edges of the tree-hypercube.

5.2 Conclusion and Future Work

As a result the embedding algorithm 4.1 performs an embedding of irregular shape of hex-cells into tree-hypercube TH(2,d) where d >= 2 is performed. Embedding of regular shape of hex-cells into tree-hypercube can't be performed, requiring the structure of the tree-hypercube. And so as not to have additional number of links and not to increase the cost. The embedding of hex-cells into tree-hypercube has dilation 1, congestion 1, and expansion 1.1. In algorithm 4.2, the embedding of hex-cells into tree-hypercube has dilation 1, congestion 1, and expansion 1.1 when mapping HC(1); and dilation 1, congestion 1, and expansion 1.3 when mapping HC(2). The expansion increased because the number of nodes in the host graph (tree-hypercube) increased, and it is larger than the number of nodes in the guest graph (hex-cell).

For future work, we suggest embedding of hex-cells into other topologies, such as meshes and binary trees, and with lower dilation and expansion as possible. Furthermore, we will consider the embedding of hex-cell with respect to their depth, not depending on the structure of the host topology. In future work, in order to embed the regular shape of hex-cells, we suggest the embedding of hex-cells into hierarchal tree-hypercube, and also by using the required additional number of links.

Communication is unavoidable in parallel computing applications, and the communication patterns are intrinsically associated with the applications themselves. Therefore, the embedding of communication pattern graphs into the topologies of multiprocessor structure is of great importance. In some cases, a 100-percent fault tolerant embedding is possible. That is, there are no faulty nodes or links in the mapping of communication pattern graph on the host multiprocessor topology. For future work, we suggest to design an algorithm for a fault-tolerant embedding of hex-cell topology into tree-hypercube, and this algorithm should be based on the existence of faulty links and/or faulty nodes.

References

[1] Abuelrub E., and Bettayeb S., "Embedding Rings into Faulty Twisted Hypercubes", Computing and Informatics Journal, Vol. 16, No. 4, 1997.

[2] Aderhold M. and Slack J., "Embedding Balanced Binary Trees into the Hypercube", Journal of Information Science and Engineering, Vol. 14, No. 5, pp. 740-752, 1997.

[3] Almobaideen W., Qatawneh M., Sliet A, Salah I, and Al-Sharaeh S., "Efficient Mapping Scheme of Ring Topology onto Tree- Hypercubes". Journal of Applied Sciences, Vol. 7, No. 18, pp. 2666-26770, 2007.

[4] Asanovic K., "The Landscape of Parallel Computing: A Research View", University of California, Berkeley, Technical Report No. UCB/EECS-183, 2006.

[5] Bein D., Bein W., and Latifti S., "An Optimal Embedding of Honeycomb Networks into Hypercubes", Parallel Processing Letters, Vol. 14, No. 3-4, pp 367 375, 2004.

[6] Bettayeb S., Girou B., and Sudborough M., "Embedding star networks into Hypercubes Computers", IEEE Transactions, Vol. 45, No. 2, pp. 186-194, 1996.

[7] Chang H. Y. and Chen R.J., "Embedding cycles in IEH graphs", Information Processing Letters, Vol. 64, No. 4, pp. 23-27, 1997.

[8] Chen W. K., and Stallmann M. F., "On Embedding Binary Trees into Hypercubes", Journal of Parallel Distributed Computing, Vol. 24, No. 3, pp. 132-138, 1995.

[9] Dvorak T., "Dense Sets and Embedding Binary Trees into Hypercubes", Discrete Applied Mathematics Journal, Vol. 155, No. 4, pp. 506-514, 2007.

[10] Fan J. and Jia X., "Embedding Meshes into Crossed Cubes", Information Sciences Journal, Vol. 177, No. 15, pp. 3151-3160, 2007.

[11] Fan J., Jia X., and Xiaola L. "Embedding of Cycles in Twisted Cubes with Edge-Pancyclic", Algorithmica Journal, Vol. 51, No. 3, pp. 264-282, 2007.

[12] Grama A., Gupta A., Karypis G., and Kumar V., "Introduction to Parallel Computing", Addison Wesley, pp. 12-72, 2003.

[13] Hamdi H. and Song S. W., “Matrix Transformations: An Efficient Method for Embedding Networks into the Hypercube”, IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 6, pp. 897-902, 1997.

[14] Hamdi M. and Song S. W., "Embedding Hierarchical Hypercube Networks into the Hypercube", IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 9, pp. 876-883, 1997.

[15] Heun V. and Mayr E. W., "Eembedding of Arbitrary Binary Tree into its Optimal Hypercube ", Journal of Algorithms, Vol. 20, No. 4, pp. 431-445, 1996.

[16] Huang C. H., HSIAO J. Y. and LEE R C. T., "Embedding Incomplete Binary Trees into Incomplete Hypercubes", IEE proceedings on Computers and digital techniques, Vol. 145, No.6, pp. 377-384, 1998.

[17] Marilynn L. and Stout F., "Embeddings in Hypercubes", Mathematical and Computational Modelling Journal, Vol. 11, No. 3, pp. 222-227, 1988.

[18] Omari M., "Tree-Hypercube: Hierarchal, Partitionable Multiprocessor Interconnection Network", PHD Dissertation, Illinois Institute of Technology, USA, 1992.

[19] Omari, M. and Abu-Salem H., "Tree-Hypercube: A Multiprocessor Interconnection Topology", Abhath Al-Yarmouk, Vol. 6, No. 2, pp. 9-24, 1998.

[20] Peterson L. and Dave B., "Computer Networks, A System Approach", Morgan Kaufmann, Third Edition, 2003.

[21] Qatawneh M. “Embedding Linear Array onto Tree-Hypercube Network”, European Journal of Scientific Research, Vol. 10, No. 2, pp. 72-77, 2005.

[22] Ranka Sanjay, Jhychun Wang and Nangkang Yeh. "Embedding Meshes on the Star Graph", IEEE Proceeding Conference on Super Computers, Vol. 12, No.3 , pp. 476- 485, 1990.

[23] Sedelev O., "Realization of Boolean functions by Combinational Circuits Embedd in the Hypercube", Moscow University Computational Mathematics and Cybernet Journal, Vol. 32, No. 1, pp. 44-50, 2008.

[24] Sharieh A., Qatawneh M., Almobaideen W., and Sleit A., "Hex- Cell: Modeling, Topological Properties and Routing Algorithm", European Journal of Scientific Research, Vol.22, No. 3, pp. 457-468, 2008.

[25] Sheng J., Hung H., and Huey G., "Embedding Fault-Free Cycles in Crossed Cubes with Conditional Link Faults", Journal of Supercomputing, Vol. 46, No. 2, pp 143-152, 2008.

[26] Tsai C., "Cycles Embedding in Hypercubes with Node Failures", Information Processing Letters, Vol. 102, No. 6, pp. 242-246, 2007.

[27] Wang K. H and Briggs F. A., “Computer Architecture and Parallel Processing”, Mc-Graw Hill, 1994.

[28] Wang S. C., Leu Y. R. and Keu S. Y., "Distributed Fault Tolerant Embedding of Several Topologies into Hypercubes", Journal of Information Science and Engineering, Vol. 20, No.4, pp. 707-732, 2004.

[29] Yang X., Tang Y., and Cao J**.,** " Embedding Torus in Hexagonal Honeycomb Torus", Computers and Digital Techniques Journal, Vol. 2, No. 2, pp. 86-93, 2008.

Appendix A
Code of the Routing Application

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimNetworkApp
{
public partial class frmMain : Form
{
public frmMain()
{
InitializeComponent();
}

private void btnExit_Click(object sender, EventArgs e)
{
Application.Exit();
}

private void btnHecx_Click(object sender, EventArgs e)
{
Form01 frm = new Form01();
frm.ShowDialog();
}

private void btnSmallTree_Click(object sender, EventArgs e)
{
Form02 frm = new Form02();
frm.ShowDialog();

}

private void label1_Click(object sender, EventArgs e)
{

}

           private void label2_Click(object sender, EventArgs e)
{

}

private void frmMain_Load(object sender, EventArgs e)
{

}
}
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimNetworkApp
{
public partial class Form01 : Form
{
public Form01()
{
InitializeComponent();
}

private void Form01_Load(object sender, EventArgs e)
{
dataSet2.Paths.AddPathsRow("1,1", "1,000");
dataSet2.Paths.AddPathsRow("1, 2", "1,001");
dataSet2.Paths.AddPathsRow("2, 2", "2,000");
dataSet2.Paths.AddPathsRow("2, 1", "2,001");
dataSet2.Paths.AddPathsRow("2, 3", "2,010");
dataSet2.Paths.AddPathsRow("1, 3", "2,011");

}

private void btnRun1_Click(object sender, EventArgs e)
{

if (comboBox1.SelectedIndex == comboBox2.SelectedIndex)
{
MessageBox.Show("you can't send message to the same node !!!");
return;
}
tree cls = new tree();

cls.Ls                                                                    =
Convert.ToInt32(comboBox1.SelectedValue.ToString().Substring(0, 1));
cls.Ld                                                                    =
Convert.ToInt32(comboBox2.SelectedValue.ToString().Substring(0, 1));

cls.S = 2;


string     x     =     comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2);
string     y     =     comboBox2.SelectedValue.ToString().Substring(2,
comboBox2.SelectedValue.ToString().Length - 2);

cls.sor        =        comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2);

string     aa     =     comboBox2.SelectedValue.ToString().Substring(2,
comboBox2.SelectedValue.ToString().Length - 2).ToString();
```

```
string[] a1 = new string[3];

a1[2] = aa.Substring(0, 1);
a1[1] = aa.Substring(1, 1);
a1[0] = aa.Substring(2, 1);

for (int i = 0; i < a1.Length; i++)
{
cls.des += a1[i]; //001 textBox2.Text.Substring(2, textBox2.Text.Length
- 2);
}
string[] msg = cls.Route_th(x, y).Split('|');

string            m            ="1:                          "            +
dataSet2.Paths.Rows[comboBox1.SelectedIndex]["path1"].ToString()  +    "
==>  " + comboBox1.SelectedValue.ToString() ;

// MessageBox.Show(m);
for (int i = 0; i < msg.Length; i++)
{
for (int j = 0; j < dataSet2.Paths.Rows.Count ; j++)
{
if (msg[i].Equals(dataSet2.Paths.Rows[j]["path2"].ToString()))
{
m    +=    "\n"    +    Convert.ToInt32(i    +    1)    +    ":        "    +
dataSet2.Paths.Rows[j]["path1"].ToString()    +    "       ==>       "    +
dataSet2.Paths.Rows[j]["path2"].ToString();
break;
}
}

}



MessageBox.Show(m);
}

private void btnRun_Click(object sender, EventArgs e)
{

if (comboBox1.SelectedIndex == comboBox2.SelectedIndex)
{
MessageBox.Show("you can't send message to the same node !!!");
return;
}

clsCells cls = new clsCells();

string m = null ;

int index= comboBox1.SelectedIndex;
string[]                                val1                                =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

index= comboBox2.SelectedIndex;
```

```
string[]                                val2                                =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

int Xs = Convert.ToInt32( val1[0]);
int Ys = Convert.ToInt32(val1[1]);


int Xd = Convert.ToInt32( val2[0]);
int Yd = Convert.ToInt32( val2[1]);


m += Xs + "," + Ys;
if (Xs < Xd)
{
m += cls.moveDown(Xs, Ys,Xd, Yd, 1);
}
else if (Xs > Xd)
{
m += cls.moveUp(Xs, Ys, Xd, Yd, 1);
}
else
{
m += cls.moveHorizontal(Xs, Ys, Xd, Yd, 1);
}


MessageBox.Show(m);
}
}
}



using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SimNetworkApp
{
public partial class Form02 : Form
{
public Form02()
{
InitializeComponent();
}

private void Form02_Load(object sender, EventArgs e)
{
dataSet2.Paths.AddPathsRow("1,1", "1,0000");
dataSet2.Paths.AddPathsRow("1,2", "1,0001");
dataSet2.Paths.AddPathsRow("2,3", "2,0000");
dataSet2.Paths.AddPathsRow("2,2", "2,0001");
```

```
dataSet2.Paths.AddPathsRow("2,4", "2,0010");
dataSet2.Paths.AddPathsRow("1,3", "2,0011");
dataSet2.Paths.AddPathsRow("3,2", "3,0000");

dataSet2.Paths.AddPathsRow("3,3", "3,0001");
dataSet2.Paths.AddPathsRow("3,1", "3,0010");
dataSet2.Paths.AddPathsRow("2,1", "3,0011");
dataSet2.Paths.AddPathsRow("2,6", "3,0100");
dataSet2.Paths.AddPathsRow("2,5", "3,0101");
dataSet2.Paths.AddPathsRow("1,5", "3,0110");


dataSet2.Paths.AddPathsRow("1,4", "3,0111");
dataSet2.Paths.AddPathsRow("4,2", "4,0000");
dataSet2.Paths.AddPathsRow("4,1", "4,0001");
dataSet2.Paths.AddPathsRow("4,3", "4,0010");
dataSet2.Paths.AddPathsRow("3,4", "4,0011");
dataSet2.Paths.AddPathsRow("4,4", "4,0110");


dataSet2.Paths.AddPathsRow("3,7", "4,1000");
dataSet2.Paths.AddPathsRow("2,7", "4,1001");
dataSet2.Paths.AddPathsRow("3,6", "4,1010");
dataSet2.Paths.AddPathsRow("3,5", "4,1011");
dataSet2.Paths.AddPathsRow("4,5", "4,1110");


}

private void btnRun_Click(object sender, EventArgs e)
{

if (comboBox1.SelectedIndex == comboBox2.SelectedIndex)
{
MessageBox.Show("you can't send message to the same node !!!");
return;
}
clsCells cls = new clsCells();

string m = null;

int index = comboBox1.SelectedIndex;
string[]                              val1                                    =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

index = comboBox2.SelectedIndex;
string[]                              val2                                    =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

int Xs = Convert.ToInt32(val1[0]);
int Ys = Convert.ToInt32(val1[1]);


int Xd = Convert.ToInt32(val2[0]);
int Yd = Convert.ToInt32(val2[1]);
```

```
m += Xs + "," + Ys;
if (Xs < Xd)
{
m += cls.moveDown(Xs, Ys, Xd, Yd, 2);
}
else if (Xs > Xd)
{
m += cls.moveUp(Xs, Ys, Xd, Yd, 2);
}
else if (Xs == Xd)
{
m += cls.moveHorizontal(Xs, Ys, Xd, Yd, 2);
}


MessageBox.Show(m);
}

private void btnRun1_Click(object sender, EventArgs e)
{

if (comboBox1.SelectedIndex == comboBox2.SelectedIndex)
{
MessageBox.Show("you can't send message to the same node !!!");
return;
}
tree cls = new tree();

cls.Ls                                                                    =
Convert.ToInt32(comboBox1.SelectedValue.ToString().Substring(0, 1));
cls.Ld                                                                    =
Convert.ToInt32(comboBox2.SelectedValue.ToString().Substring(0, 1));

cls.S = 2;


string     x      =      comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2);
string     y      =      comboBox2.SelectedValue.ToString().Substring(2,
comboBox2.SelectedValue.ToString().Length - 2);

//cls.sor        =        comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2);

string     aa     =     comboBox2.SelectedValue.ToString().Substring(2,
comboBox2.SelectedValue.ToString().Length - 2).ToString();
string[] a1 = new string[4];

a1[3] = aa.Substring(0, 1);
a1[2] = aa.Substring(1, 1);
a1[1] = aa.Substring(2, 1);
a1[0] = aa.Substring(3, 1);

for (int i = 0; i < a1.Length; i++)
{
cls.des += a1[i]; //001 textBox2.Text.Substring(2, textBox2.Text.Length
- 2);
```

```
}

//================================================
string     aa2     =     comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2).ToString();
string[] a2 = new string[4];

a2[3] = aa2.Substring(0, 1);
a2[2] = aa2.Substring(1, 1);
a2[1] = aa2.Substring(2, 1);
a2[0] = aa2.Substring(3, 1);

for (int i = 0; i < a1.Length; i++)
{
cls.sor += a2[i]; //001 textBox2.Text.Substring(2, textBox2.Text.Length
- 2);
}
//============================================================


string[] msg = cls.Route_th(x, y).Split('|');

string   m   =   "Run   in   TH   after   embedding\n\n1:       "   +
dataSet2.Paths.Rows[comboBox1.SelectedIndex]["path1"].ToString()   +   "
==>  " + comboBox1.SelectedValue.ToString();

// MessageBox.Show(m);
for (int i = 0; i < msg.Length; i++)
{
for (int j = 0; j < dataSet2.Paths.Rows.Count; j++)
{
if (msg[i].Equals(dataSet2.Paths.Rows[j]["path2"].ToString()))
{
m   +=   "\n"   +   Convert.ToInt32(i   +   1)   +   ":       "   +
dataSet2.Paths.Rows[j]["path1"].ToString()   +   "       ==>       "   +
dataSet2.Paths.Rows[j]["path2"].ToString();
break;
}
}

}
//======================================================
clsCells cls1 = new clsCells();
m += "\n\nRun in HC before embedding\n";

int index = comboBox1.SelectedIndex;
string[]                                val1                                =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

index = comboBox2.SelectedIndex;
string[]                                val2                                =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

int Xs = Convert.ToInt32(val1[0]);
int Ys = Convert.ToInt32(val1[1]);
```

```
int Xd = Convert.ToInt32(val2[0]);
int Yd = Convert.ToInt32(val2[1]);


m += Xs + "," + Ys;
if (Xs < Xd)
{
m += cls1.moveDown(Xs, Ys, Xd, Yd, 2);
}
else if (Xs > Xd)
{
m += cls1.moveUp(Xs, Ys, Xd, Yd, 2);
}
else if (Xs == Xd)
{
m += cls1.moveHorizontal(Xs, Ys, Xd, Yd, 2);
}



MessageBox.Show(m);
}

private void btnRunAll_Click(object sender, EventArgs e)
{
ShowMessage msgbox = new ShowMessage();
string m = null;
for (int mm = 0; mm < comboBox1.Items.Count; mm++)
{
for (int n = 0; n < comboBox2.Items.Count; n++)
{
if (mm == n)
{
// MessageBox.Show("you can't send message to the same node !!!");

}
else
{



tree cls = new tree();

cls.Ls                                                                    =
Convert.ToInt32(dataSet2.Paths.Rows[mm]["path2"].ToString().Substring(0
, 1));
cls.Ld                                                                    =
Convert.ToInt32(dataSet2.Paths.Rows[n]["path2"].ToString().Substring(0,
1));

cls.S = 2;


string   x   =   dataSet2.Paths.Rows[mm]["path2"].ToString().Substring(2,
dataSet2.Paths.Rows[mm]["path2"].ToString().Length - 2);
string   y   =   dataSet2.Paths.Rows[n]["path2"].ToString().Substring(2,
dataSet2.Paths.Rows[n]["path2"].ToString().Length - 2);
```

```
//cls.sor       =       comboBox1.SelectedValue.ToString().Substring(2,
comboBox1.SelectedValue.ToString().Length - 2);

string   aa   =   dataSet2.Paths.Rows[n]["path2"].ToString().Substring(2,
dataSet2.Paths.Rows[n]["path2"].ToString().Length - 2).ToString();
string[] a1 = new string[4];

a1[3] = aa.Substring(0, 1);
a1[2] = aa.Substring(1, 1);
a1[1] = aa.Substring(2, 1);
a1[0] = aa.Substring(3, 1);

for (int i = 0; i < a1.Length; i++)
{
cls.des += a1[i]; //001 textBox2.Text.Substring(2, textBox2.Text.Length
- 2);
}

//================================================
string  aa2  =  dataSet2.Paths.Rows[mm]["path2"].ToString().Substring(2,
dataSet2.Paths.Rows[mm]["path2"].ToString().Length - 2).ToString();
string[] a2 = new string[4];

a2[3] = aa2.Substring(0, 1);
a2[2] = aa2.Substring(1, 1);
a2[1] = aa2.Substring(2, 1);
a2[0] = aa2.Substring(3, 1);

for (int i = 0; i < a1.Length; i++)
{
cls.sor += a2[i]; //001 textBox2.Text.Substring(2, textBox2.Text.Length
- 2);
}
//============================================================


string[] msg = cls.Route_th(x, y).Split('|');

m += "1:  " + dataSet2.Paths.Rows[mm]["path1"].ToString() + "  ==>  " +
dataSet2.Paths.Rows[mm]["path2"].ToString();

// MessageBox.Show(m);
for (int i = 0; i < msg.Length; i++)
{
for (int j = 0; j < dataSet2.Paths.Rows.Count; j++)
{
if (msg[i].Equals(dataSet2.Paths.Rows[j]["path2"].ToString()))
{
m   +=   "\n"   +   Convert.ToInt32(i   +   1)   +   ":       "   +
dataSet2.Paths.Rows[j]["path1"].ToString()   +   "       ==>       "   +
dataSet2.Paths.Rows[j]["path2"].ToString();
break;
}
}

}
```

```
m += "\n\n\n";
msgbox.richTextBox1.Text += m;


//=====================================================

clsCells cls1 = new clsCells();


int index = mm;
string[]                                  val1                                  =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

index = n;
string[]                                  val2                                  =
dataSet2.Paths.Rows[index]["path1"].ToString().Split(',');

int Xs = Convert.ToInt32(val1[0]);
int Ys = Convert.ToInt32(val1[1]);


int Xd = Convert.ToInt32(val2[0]);
int Yd = Convert.ToInt32(val2[1]);


m += Xs + "," + Ys;
if (Xs < Xd)
{
m += cls1.moveDown(Xs, Ys, Xd, Yd, 2);
}
else if (Xs > Xd)
{
m += cls1.moveUp(Xs, Ys, Xd, Yd, 2);
}
else if (Xs == Xd)
{
m += cls1.moveHorizontal(Xs, Ys, Xd, Yd, 2);
}
m += "\n=================================================\n";


}
}


}
msgbox.richTextBox1.Text = m;
msgbox.ShowDialog();
}
}
}
```