Properties and Algorithms of the KCube Interconnection Networks

Keivan Noroozi

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University
St. Catharines, Ontario

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Ke Qiu, for all his help throughout this process. His guidance has helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my master's study.

I would like to thank the members of my supervisory committee, Dr. M. Winter, Dr. S. Houghten, and Dr. B. Farzad, for their time and insightful advice.

Also I would like to thank the Department of Computer Science, Brock University, for giving me the opportunity to study and work in such a friendly environment, and for providing me with financial support.

Finally, I would like to thank my family: my parents, brothers and sister for their unconditional love and support.

Co-Authorship

Some preliminary results were reported in the following paper:

1. “On the Hamiltonicity of the KCube Interconnection Network.” (with K. Qiu and L. Zhao), Second International Symposium on Computing and Networking (CANDAR) IEEE, 2014.

2. “On the Hamiltonicity, Connectivity, and Broadcasting Algorithm of the KCube.” (with K. Qiu), Third International Symposium on Computing and Networking (CANDAR) IEEE, 2015.

Abstract

The KCube interconnection network was first introduced in 2010 in order to exploit the good characteristics of two well-known interconnection networks, the hypercube and the Kautz graph. KCube links up multiple processors in a communication network with high density for a fixed degree. Since the KCube network is newly proposed, much study is required to demonstrate its potential properties and algorithms that can be designed to solve parallel computation problems.

In this thesis we introduce a new methodology to construct the KCube graph. Also, with regard to this new approach, we will prove its Hamiltonicity in the general $KC(m, k)$. Moreover, we will find its connectivity followed by an optimal broadcasting scheme in which a source node containing a message is to communicate it with all other processors.

In addition to KCube networks, we have studied a version of the routing problem in the traditional hypercube, investigating this problem: whether there exists a shortest path in a Q_n between two nodes 0^n and 1^n, when the network is experiencing failed components. We first conditionally discuss this problem when there is a constraint on the number of faulty nodes, and subsequently introduce an algorithm to tackle the problem without restrictions on the number of nodes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In computer science, problems can be solved by two major approaches, using single a processor and multiple processors. Computers that use a single processor are called sequential computers, and the ones that use multiple processors are called parallel computers. In parallel computers several processors cooperate simultaneously to accomplish a task in a shorter time than it requires for a sequential computer. Parallel computers have two main advantages over sequential computers. First, in many computational problems, the time needed to obtain a solution using a sequential computer (single processor) is unacceptably high. There is a stage that the speed of computation cannot be increased regardless of how other components of the system are improved. Second, there exist problems impossible to tackle sequentially regardless of how much time it takes. These drawbacks of sequential computers motivate computer scientists to address a variety of computational problems using multiple processors.

This field of computer science opens a new door in terms of solving computational problems and it proposes new techniques for the design and analysis of algorithms. Accordingly, it requires a completely new theoretical knowledge in addition to the sequential methods of solving those problems.

Based on how the processors communicate with each other, there are two major computational models for which parallel algorithms can be designed, namely, *shared-memory parallel machines* and *interconnection networks*.

In this chapter, we first introduce the classification of computer architectures. Then we explain two major computational models, shared-memory machines and interconnection networks, as well as introducing some well-known examples of the latter. At the end of this chapter we discuss the parameters used to measure and analyze parallel algorithms in order to understand their efficiency.

1.1 Classification of Computer Architecture

The computer machine structure can be described using the stream concept. A stream simply means a sequence of items that can be either instructions or data. Machines have been classified into four different categories based on the interaction between instructions and data streams [13]:

- **SISD - Single Instruction, Single Data Stream**

  This represents the most conventional sequential computing machines. This class of computers does not utilize any parallelism within their computations except instruction pipelining. The processor follows a single instruction stream to process a data stream.

- **SIMD - Single Instruction, Multiple Data Stream**

  Here, multiple identical processors can operate a single instruction in parallel on different pieces of data. Processors have their own local memory to store data and the single instruction stream is issued by a central control unit. All the major parallel models and algorithms in this thesis are discussed in accordance with this class of computers.

- **MISD - Multiple Instruction, Single Data Stream**

  MISD machines have multiple processors, each of which has its own control unit and together they share a common memory unit. They can execute multiple instructions on a single data sequence. In reality there is no implementation for this class of computers.

- **MIMD - Multiple Instruction, Multiple Data Stream**

  This class of parallel computers use multiple processors that have their own memory and control units. They function autonomously and execute different instructions on different data sequences. Each processor can solve a subproblem of the main problem independently. Hence, MIMD parallel machines are considered as the most powerful class among the other classes.

SIMD and MIMD machines can utilize either *shared-memory* family or *interconnection networks*.

The processors of a parallel system have to communicate with each other in order to send or receive a piece of data, and finally, to obtain a result for the main problem. There are two computational models according to the manner they communicate: Shared-memory parallel machines and Interconnection Networks.

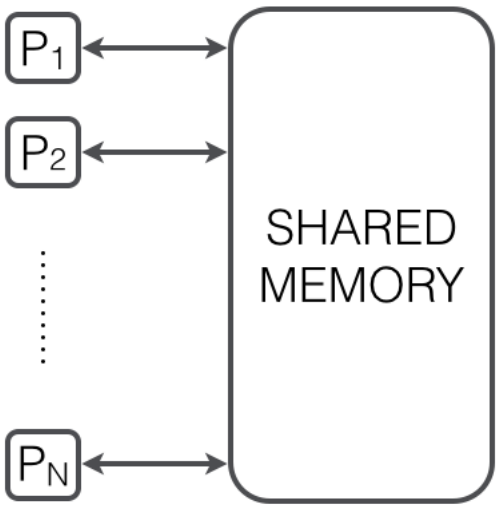

Figure 1.1: A Shared-Memory Parallel Computer

1.2 Shared-Memory Parallel Machines

Shared-memory parallel machines are one of the fundamental computational models that allow processors to communicate through a single and common Memory Access Unit (MAU). As it can be understood from the name of this model, processors do not have their own memory. The shared-memory model is also called Parallel Random Access Memory (PRAM), shown in Fig 1.1.

In the PRAM model, when two processors, say P_i and P_j, wish to communicate with each other, they have to accomplish this by using a shared memory as a bulletin board. For example, P_i has a piece of data needed by P_j. First, P_i writes it to a specific location of the memory and then it will be read by P_j.

There are different restrictions for processors to access the shared-memory to read or write a piece of data. The restrictions are classified as follows:

- **Exclusive Read (ER):** In this form of restriction, only one processor in one time unit can read from a particular memory location. Then in parallel, p processors can simultaneously read from p distinct memory locations.

- **Concurrent Read (CR):** In this form of restriction, multiple processors in one time unit can read from a particular memory location. Then in parallel, p processors can simultaneously read from p' distinct memory locations, where $p' \leq p$. Each of the p'

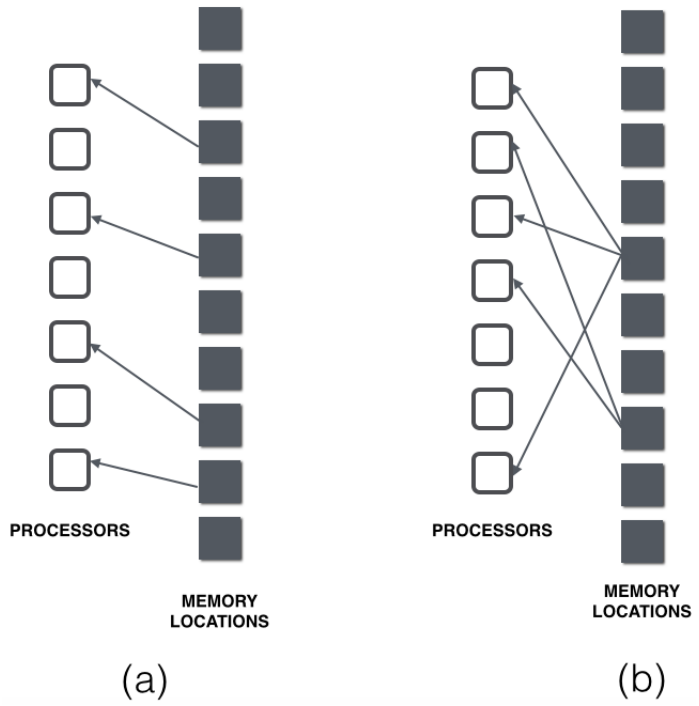

Figure 1.2: Reading Access to Memory: a. Exclusive Read ; b. Concurrent Read

memory locations can be read possibly by multiple processors. It is evident that ER is a special case of CR.

Fig 1.2 illustrates the two types of reading restriction for the processors.

- **Exclusive Write (EW):** In this form of restriction, only one processor in one time unit can write to a particular memory location. Then in parallel, p processors can simultaneously write to p distinct memory locations.

- **Concurrent Write (CW):** In this form of restriction, multiple processors in one time unit can write to a particular memory location. Then in parallel, p processors can simultaneously write to p' distinct memory locations, where $p' \leq p$. Each of the p' memory locations can be written into possibly by multiple processors. Again, EW is a special case of CW.

  Fig 1.3 illustrates the two types of writing restriction for the processors.

Combining these different restrictions for read/write, we have four types of PRAM computers regarding reading and writing the data within the shared memory which are as follows:

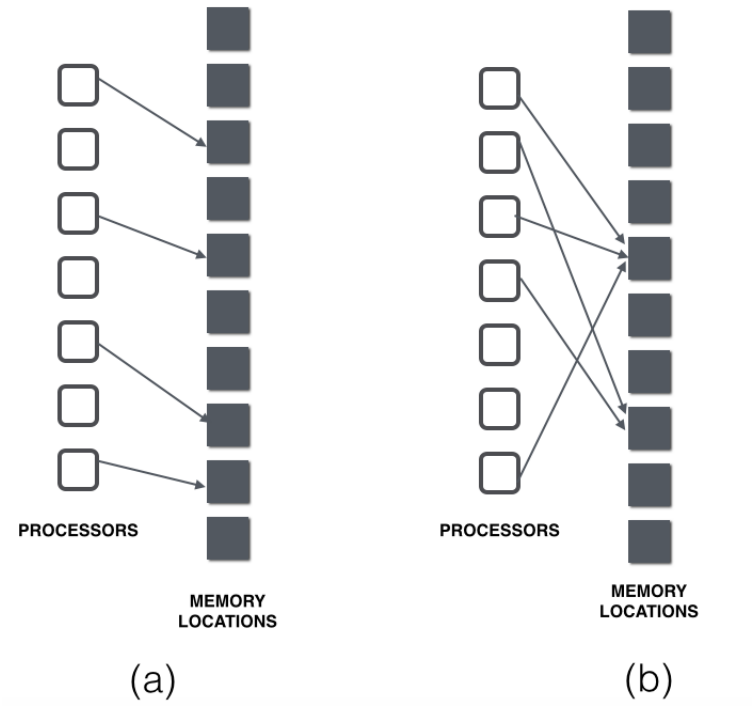

Figure 1.3: Writing Access to Memory: a. Exclusive Write ; b. Concurrent Write

1. **Exclusive Read, Exclusive Write (EREW):** Here, any memory location can be read from and written to, only by one processor at the same time. This class is the most restricted among the others and considered the weakest model.

2. **Exclusive Read, Concurrent Write (ERCW):** Here, multiple processors can write to but not read from the same memory location concurrently.

3. **Concurrent Read, Exclusive Write (CREW):** Here, multiple processors can read from but not write to the same memory location concurrently.

4. **Concurrent Read, Concurrent Write (CRCW):** Here, any memory location can be read from and written to, by multiple processors at the same time. This class is the most powerful among the others since it has the maximum flexibility in reading from and writing into the memory.

1.3 Interconnection Networks

Interconnection networks are another class of parallel processing. In contrast to the shared-memory model of parallel processing, in which processors share a common memory, each

processor in interconnection networks has its own memory. There are either two-way or one-way links between processors that make communication between them possible. Therefore, if a processor wishes to send a piece of data to other processors, it has to be done through these links for routing the data from its memory to the memory of the destination processors.

Network topology is a key factor when the performance of an interconnection network is being discussed. They are modeled by a graph $G = (V, E)$ where V is the set of nodes (each node indicates a processor in the network), and E is the set of edges that indicates the links between processors.

1.3.1 Definitions

Since interconnection networks are represented by graphs, we can use their properties and parameters to evaluate and analyze the networks. We follow the standard terminology of graph theory and interconnection networks [4, 19], and use the terms "interconnection network" and "graph" , "processor" and "node" , "edge" and "link" interchangeably. Further in this section, we will review the essential definitions and properties that are needed to analyze and compare different interconnection networks. Some well-known and important interconnection networks will be discussed afterwards.

Definition 1. *A **graph** $G(V, E)$ is a non-empty set V of nodes and a set E of edges. If an edge $e \in E = (u, v)$ where $u, v \in V$, then u and v are said to be **neighbours** and the edge e is said to be **incident** on these nodes. A **directed graph** (or **digraph**) is a graph where edges (or arcs) have a direction associated with them, i.e., the pair of nodes corresponding to each edge is ordered.*

Definition 2. *In a graph $G = (V, E)$, the **degree** of node $u \in V$ is the number of neighbours of V. The degree of a graph is said to be the maximum of all node degrees.*

Definition 3. *A graph $G = (V, E)$ is called **regular** when all the nodes in G have the same degree. We denote such a graph by n-regular when n is the degree of each node (also the degree of the graph)*

Definition 4. *In a graph G, a **path** is defined as a sequence of successive edges which connect a sequence of distinct nodes. The shortest path in number of edges between two nodes $u, v \in V$ is called **distance** between u and v. The maximum distance between any two nodes $u, v \in V$ is called the **diameter** of the graph.*

When we are evaluating an interconnection network, the diameter is considered as one of the important properties. This is because it plays an essential role in many aspects such

as routing and broadcasting a message. A smaller diameter is much more desirable for a network as it is related to the routing and broadcasting times.

Definition 5. *A **Hamiltonian cycle (Eulerian Cycle)**, also called a Hamiltonian circuit (Eulerian circuit), is a cycle that visits each node (edge) exactly once. A graph possessing a Hamiltonian cycle (Eulerian Cycle) is said to be Hamiltonian (Eulerian).*

Definition 6. *A Hamiltonian path in a graph is a path that visits each node of the graph exactly once.*

Definition 7. *A graph is **bipartite** if we can divide its nodes into two disjoint sets V_1 and V_2 such that there is no edge between the nodes of each set. Then each edge connects u to v where $u \in V_1$ and $v \in V_2$. Each set of bipartite graph is called **bipartition**.*

Definition 8. ***Bisection width** of a network refers to the minimum number of the links that need to be removed to break the network into two equal sized disconnected networks.*

Definition 9. *A graph G is n-connected if there exist n internally node-disjoint paths between any pair of nodes. The connectivity $\kappa(G)$ of G is the greatest integer n such that G is n-connected*

Definition 10. *An **Isomorphism** from graph G to graph H is a bijection $f : V(G) \to V(H)$ such that $(u, v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. G is isomorphic to H if there is an isomorphism from G to H. An **automorphism** of a graph is an isomorphism from G into G*

Definition 11. *A graph is **node-symmetric (edge-symmetric)** if for every pair of nodes $u, v \in V(G)$ ($e, f \in E(G)$), there is an automorphism that maps u to v (e to f). Simply, when a graph is node-symmetric (edge-symmetric), it means if you look at the whole graph from any node (edge) of $V(G)$ ($E(G)$) it will be exactly the same graph.*

Being symmetric is very important to a network. For example, in a node-symmetric network it is much easier to design a routing and broadcasting algorithm due to the same accessibility between the processors. Also the starting or destination point of the algorithm is less of an issue since all the nodes are the same.

Definition 12. *A graph G is called **f-fault tolerant** when the removal of f or less nodes does not make the graph disconnected. The **fault tolerance** of a graph G is the largest value for f that G is f-fault tolerant.*

Fault-tolerance is the property that indicates the capability of a network to continue its operation when failed components exist in the network. Highly fault-tolerant networks are most likely able to continue to perform without any drop in efficiency when some components have failed. From the interconnection network perspective, this property is highly important since working with too many processors will increase the probability of failure among them and it is essential for the network to tolerate it as much as possible.

Many interconnection networks have been proposed in order to connect as many processors as possible while the efficiency of network remains acceptable. Here we review some of the popular and basic interconnection networks and compare their performance with each other.

1.3.2 Linear Array and Ring

A **linear array** is a simple interconnection network to implement. It is a sequence of processors that form a one-dimensional array where each processor has two neighbours, except the terminal processors (first and last). See Fig 1.4. The degree of this graph is 2 except for the first and last nodes, and the diameter is $N - 1$ when N is the number of processors. If the first and last nodes of a linear array are connected to each other the graph is called a **ring**. One of the considerable advantages of the ring over the linear array topology is its diameter $\lfloor N/2 \rfloor$, which is half of that of a linear array's diameter.

1.3.3 Complete Graph

This network has the most possible number of links between processors such that each processor P_i, $1 \leq i \leq N$ is directly connected to all other $N - 1$ processors, and it is called *complete graph* or *clique*. We denote complete graph by K_n, when n is the size of the graph. This network has the best diameter which is 1. However, the number of edges in the network of size N is equal to $\binom{N}{2}$, which is unacceptably high to be implemented in real world.

1.3.4 Mesh and Torus

Mesh topology is an $m * n$ grid that forms a two-dimensional array with m rows and n columns. Each processor P_{ij} (where i and j denote the row and column number), that does not lie in the boundary rows and columns, has a degree of four and the neighbours $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$, $P_{i,j+1}$. It can be easily shown that this network has a diameter equal to

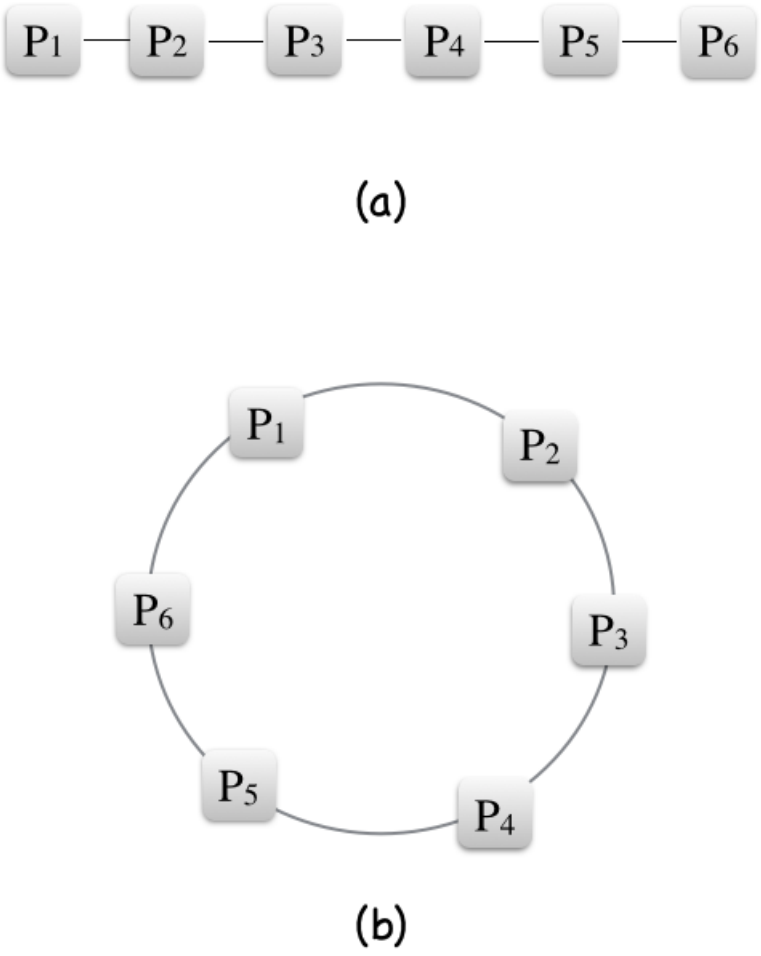

Figure 1.4: a. Linear Array of Size 6 ; b. Ring of Size 6

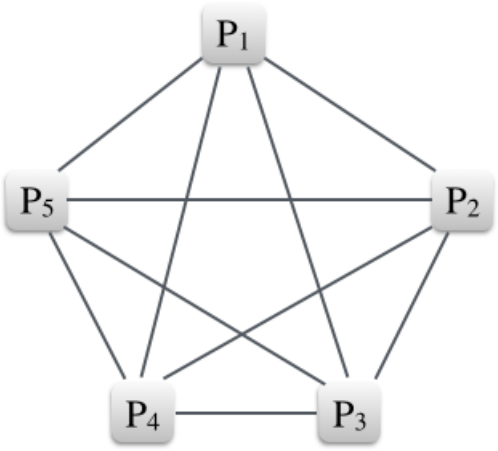

Figure 1.5: Complete Network of Size 5, K_5

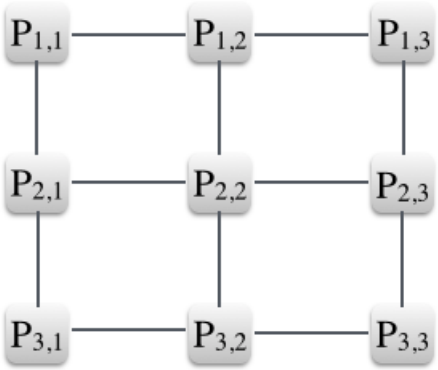

Figure 1.6: A 3×3 Mesh

$O(m+n)$. The mesh network topology is easy to layout and it has been used in many multiprocessor systems. Three or higher dimensional meshes also exist. However in practice, they become less desirable to be used.

Torus is a network topology similar to the mesh while the first and last node in each row and column are connected. See Fig 1.5. These additional edges give the nodes in the boundary of the network the same characterizations as internal nodes. For instance, mesh is not edge-symmetric while torus avoids this problem.

1.3.5 Tree

In this network topology, processors connected to each other form a complete binary tree. In a binary tree of level d there are $2^d - 1$ processors. Each processor P_i is connected to its parent by a link except for the root node. Also each processor is connected to its two children except the leaf nodes.

1.3.6 Hypercube

An n-dimensional hypercube, Q_n or n-cube, has 2^n nodes labeled from 0 to $2^n - 1$. Each node is denoted by a binary representation with length n. There is an undirected edge between (u, v) if their binary representation differ in exactly one position (one bit), i.e., $u = u_{n-1}u_{n-2}\ldots u_{i+1}u_iu_{i-1}\ldots u_1u_0$ and $v = u_{n-1}u_{n-2}\ldots u_{i+1}\bar{u_i}u_{i-1}\ldots u_1u_0, 0 \leq i \leq n-1$. Among all network topologies, the hypercube is one of the most popular networks for interconnecting many processors in a parallel computer and has been widely studied [24, 31]. This network has drawn the attention of many computer scientists during the past decades for its desirable characteristics essential to link up multiple processors such as

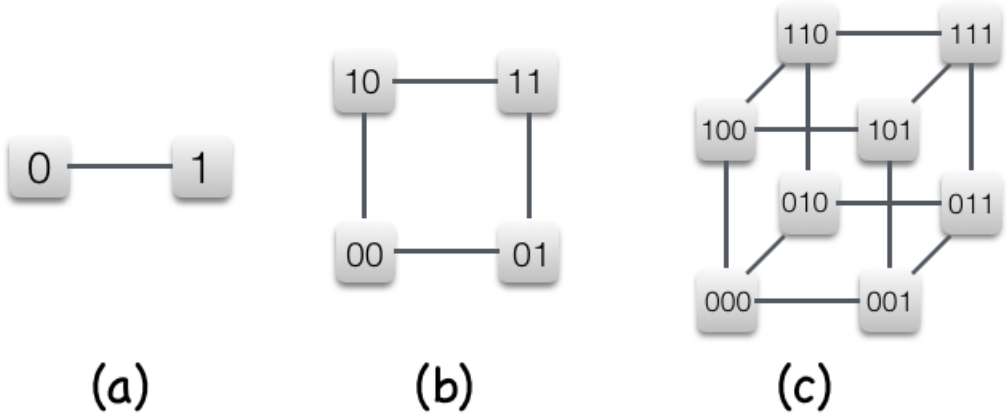

Figure 1.7: a. 1-cube; b. 2-cube; c. 3-cube

symmetric, regularity, embedding capability, and logarithmic diameter.

In Q_n, the Hamming distance of two nodes $u = u_{n-1}u_{n-2}\dots u_0$ and $v = v_{n-1}v_{n-2}\dots v_0$ is defined as $|\{i|u_i \neq v_i\}|$ denoted by $H(u, v)$. For example, $H(1110, 0111) = |\{0, 3\}| = 2$. It is straightforward to find the shortest path from u to v in Q_n by applying the following approach: Moving across the binary sequence of u from right to left, whenever a bit differs from v's bit in the same position, flip it. Therefore, we can see that the diameter of the hypercube of dimension n is $\log N = n$ when $N = 2^n$ is the number of processors.

Moreover, the broadcasting in the hypercube are optimal and can be performed efficiently which we discuss comprehensively in the following chapters.

An important property of the hypercube is that it can be decomposed into two identical hypercubes of one lower dimension. Many algorithms designed on the hypercube utilize this recursive property (called *tearing* property in some papers [31]) since the main problem can be divided into two subproblems that will be solved recursively. If an n-cube is split into two identical $(n-1)$-cubes, their nodes are in a one-to-one correspondence. There are n different ways for an n-cube to be cut into two $(n-1)$-cubes.

1.3.7 De Bruijn Graph and Kautz Graph

A de Bruijn graph $D(d, k)$ $d, k \geq 1$ is a directed graph of out-degree d with d^k nodes, and the diameter of k [6, 34]. Each node is represented as $x_k x_{k-1} \dots x_2 x_1$ where $i \in \{0, 1, \dots, d-1\}$. Each node $x_k x_{k-1} \dots x_2 x_1$ has a directed edge pointing to the node $x_{k-1} \dots x_2 x_1 \alpha$ where $\alpha \in \{0, 1, \dots, d-1\}$. See Fig 1.8.

A similar graph to the de Bruijn graph is the Kautz graph $K(d, k)$. It is also a directed graph but with more nodes, $d^k + d^{k-1}$, and the diameter of k. Each node is represented

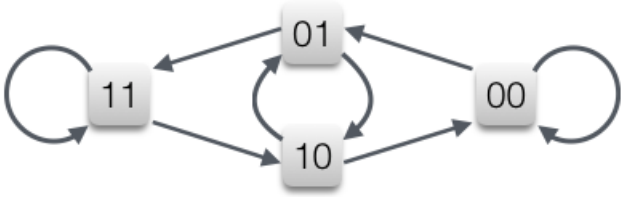

Figure 1.8: $D(2,2)$

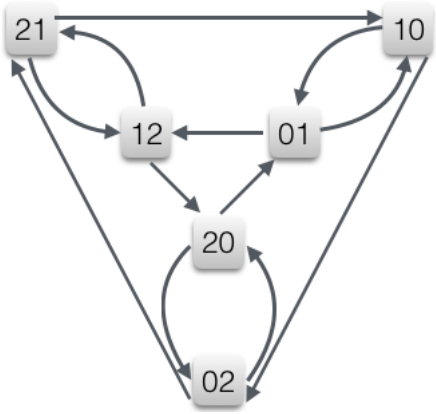

Figure 1.9: $K(2,2)$

as $x_k x_{k-1} \dots x_2 x_1$ where $x_i \in \{0, 1, \dots, d\}$ and two successive digits x_i and x_{i+1} cannot be equal ($x_i \neq x_{i+1}$). Each node $x_k x_{k-1} \dots x_2 x_1$ has a directed edge pointing to the node $x_{k-1} \dots x_2 x_1 \alpha$ where $\alpha \in \{0, 1, \dots, d\} - \{x_1\}$. See Fig 1.9.

Different properties and algorithms of this network topology are discussed in detail further in this thesis.

Table 1.1 lists different interconnection networks and some of their important properties.

1.4 Evaluating Parallel Algorithms

Before describing the ways of evaluating parallel algorithms in detail, let us review what $O(n)$ and $\Omega(n)$ indicate exactly:

- The function $f(n)$ is $\Omega(g(n))$ if there exist constants $n_0 \geq 1$ and $c > 0$ such that when $n \geq n_0$, we have $f(n) \geq cg(n)$.
- The function $f(n)$ is $O(g(n))$ if there exist constants $n_0 \geq 1$ and $c > 0$ such that when $n \geq n_0$, we have $f(n) \leq cg(n)$.

The efficiency of parallel algorithms can be assessed and evaluated by three basic and main criteria: the algorithm's running time, the number of processors, and the *Cost*. However, there are other less common tools to analyze parallel algorithms, such as the algorithm's probability of success [1]. Here, we first explain the *elementary step*, followed by a brief look into the important criteria of evaluating parallel algorithms:

Definition 13. *An elementary step is a measure of the algorithm's speed. There are two types of elementary step:*

Table 1.1: Degree, Diameter and Hamiltonicity of Interconnection Networks

Interconnection Network	Number of Nodes	Degree	Diameter	Hamiltonian
Linear Array	N	2	N	No
Ring	N	2	$\lfloor N/2 \rfloor$	Yes
Complete Graph	N	$N-1$	1	Yes
Mesh $(m * n)$	mn	4	$m+n$	No
Torus $(m * n)$	mn	4	$\lfloor (m+n)/2 \rfloor$	Yes
Hypercube	N	$\log_2 N$	$\log_2 N$	Yes
de Bruijn (d, k)	d^k	d	k	Yes
Kautz (d, k)	$d^k + d^{k-1}$	d	k	Yes

1. *Computational Step: It refers to a basic arithmetic or logical operation such as adding, comparing, and swapping two numbers that is performed by a processor on one or two data.*

2. *Routing Step: It refers to the transferring of a piece of data from one processor to another, that needs to be performed according to the algorithm's instruction. This can be accomplished via the links between processors or the shared memory.*

Algorithm running time: This is the number of time units from the beginning of the first processor operation to the end of the last processor operation to solve a computational problem. It is assumed that each computational step or any other elementary step [1], takes a constant unit of time. By counting these steps consecutively, we can measure the running time of the algorithm. Therefore, it is almost equal to the number of steps. The running time of an algorithm is referred to as the worst case scenario of that problem to be solved by the algorithm. That is, the time required for the most difficult instance of the main problem to be fully performed, and is denoted by $t(n)$.

Number of processors: Another criterion for evaluating a parallel algorithm is the number of processors required to accomplish the algorithm. In general, it is in our interest to use fewer processors in our computer as that naturally results in a less expensive system. Therefore, when comparing two different algorithms that have the same performance

except for the number of processors, the one with the fewer number of processor is much preferred. We denote this number by either $p(n)$, when it is a function of n (size of input), or by p when the number of processors is independent of the size of input.

Cost: This is an important parameter to assess a parallel algorithm and is defined as follows: $c(n) = p(n) \times t(n)$. This number presents the product of algorithm's running time and its number of processors. In other words, cost is an upper bound for the number of steps performed by the algorithm since some of the processors might be idle during the process. Then, if all the processors are active, $c(n)$ is equal to the total number of steps.

Now, the question is when the cost of an algorithm is considered good or efficient, i.e., when it is beneficial to solve a problem in parallel while there already exists a sequential algorithm to solve it. To answer this question, there is an ***efficiency*** chart by which we can measure the goodness of the parallel algorithm's cost. First we define efficiency, when t_1 is the worst case running time of a fastest existing sequential algorithm for a given problem, and t_n is the worst case running time of the parallel algorithm for the same problem, as follows:

$$E(1, p) = \frac{t_1}{cost}$$

Then we measure the efficiency number according to the following cases:

1. If $E(1, p) < 1$, then the parallel algorithm is not cost optimal.
2. If $E(1, p) = 1$, then the parallel algorithm is cost optimal.
3. If $E(1, p) > 1$, then it can be implied that there is a faster sequential algorithm for that particular problem.

For the third case, after achieving the faster sequential solution, which can be simulated from the parallel algorithm, we have to recalculate the efficiency number with the new t_1.

1.5 Organization of the Thesis

Many interconnection networks have been proposed in order to be used in parallel computers. Many of the newly proposed interconnection networks are improved versions of the old and existing ones. They can be a compound network of two or more networks, or just a variant of an old network with one or more enhancements to its properties. For example, the hypercube network has a lot of variants such as Twisted Cube, Cube Connected Cycles, Folded Cube, Augmented Cube, Exchanged hypercube, Fibonacci Cube, and etc. An example of a compound interconnection network is the KCube network, which we discuss

in detail throughout this thesis. It employs two well-known networks for its construction, the hypercube and the Kautz graph. Since it has been proposed only recently, many of its properties, as well as the possible algorithms, have not been revealed yet. Some of the KCube's properties and algorithms will be studied in the next chapters of this thesis. More specifically, we will proceed according to the following organization:

Chapter 2: A literature review of the KCube network as well as the hypercube and the Kautz graph.

Chapter 3: A thorough discussion of the KCube new construction methodology, as well as its properties, such as the Hamiltonian cycle and connectivity.

Chapter 4: A broadcasting algorithm for KCube that combines Kautz graph and hypercube broadcasting.

Chapter 5: Investigation of a routing problem on a faulty hypercube, and a general algorithm to determine the existence of the shortest path in such a hypercube.

Chapter 6: Conclusion and future work.

Chapter 2

Literature Review of the KCube Graph

2.1 Introduction

Both hypercubes and Kautz graphs have many desirable characteristics, and hence have been studied extensively. The KCube network was first proposed by Guo et al. (2010) [9], to combine these two networks and make use of their advantages. It has good modularity, expansibility and regularity [9]. In this chapter, we first go over the KCube definition and construction that was proposed by Guo et al. [9] for the first time. Then we review its already proven properties and algorithms. However, since KCube is a newly proposed architecture compared to other networks such as the hypercube, many of its potential properties have not been found yet, and therefore has room for a great deal of investigation in the future.

2.2 KCube Network

Definition 14. *Given two regular graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, a compound graph $G_2(G_1)$ is defined such that every $v \in V_2$ in G_2 is replaced by a copy of G_1 and every $e \in E_2$ is replaced by an edge connecting corresponding two copies of G_1.*

KCube is a compound graph by replacing each node of the Kautz graph by a hypercube of appropriate dimension. A KCube of dimensions d and k, denoted by $KC(d, k)$, replaces each node of $K(d, k)$ with a hypercube (sometimes called a hypercube cluster) of dimension m where $d = 2^{m-1}$. This can be derived from the fact that $K(d, k)$ has an indegree and outdegree equal to d, and if each node of $K(d, k)$ is replaced with a hypercube Q_m, we have $2d = 2^m$ due to the number of nodes in each hypercube cluster. Then, half the nodes in the hypercube will act as input nodes and half the nodes as output nodes. Now, each

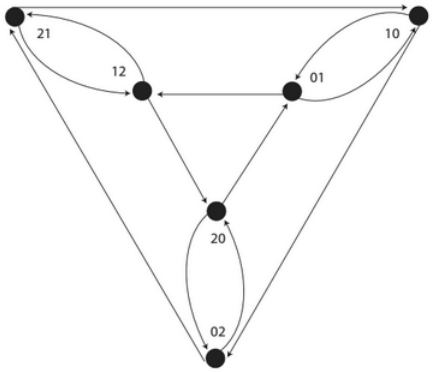

Figure 2.1: $K(2, 2)$

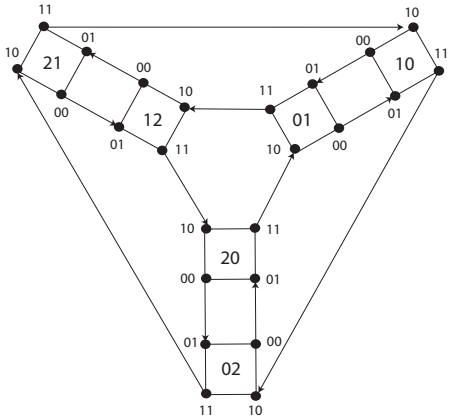

Figure 2.2: $KC(2, 2)$

edge of the $K(2^{m-1}, k)$ is replaced with an edge (remote arc) connecting two nodes from two different hypercube clusters such that the starting point of the edge is an output node of a Q_m, and the end point is an input node in another corresponding Q_m. Therefore, the edges in a KCube are divided into two groups of remote edges and hypercube edges. We know that in the $KC(d, k)$, $d = 2^{m-1}$, thus we can denote a KCube as $KC(m, k)$ where m is the hypercube clusters dimension. Fig. 2.1 and Fig. 2.2 illustrate $K(2, 2)$ and $KC(2, 2)$.

According to the definition by Guo et al. [9], as long as the two preconditions hold, the graph belongs to the KCube family. Then, there are different options that we can use to construct a KCube graph. The two preconditions that have to be satisfied in order to have a KCube are as follows [9]:

1. The outdegree and indegree of each node in $K(2^{m-1}, k)$ is 2^{m-1}, and each hypercube cluster contains 2^m nodes. These nodes need to be divided into two equal sets. One is output nodes which are starting point/source of the arcs from one hypercube cluster to other corresponding hypercube cluster. The other set is input nodes which are end point/destination of arcs from a corresponding hypercube cluster.

2. There are $d(d^k + d^{k-1})$ remote arcs in $KC(m, k)$. These arcs need to be mapped to pairs of nodes, where one end of an arc is an output node of a hypercube cluster and the other is an input node of another hypercube cluster.

An approach is proposed by Guo et al. [9] to satisfy the first precondition in which it partitions the nodes of each hypercube based on the two first bits of their binary sequences such that in Q_m the input nodes are of the form $x_{m-1} \dots x_2 10$ or $x_{m-1} \dots x_2 01$. In other words, nodes that start with "01" or "10" are considered as input nodes (white nodes). Conversely, the output nodes are of the form $x_{m-1} \dots x_2 11$ or $x_{m-1} \dots x_2 00$, that is, nodes

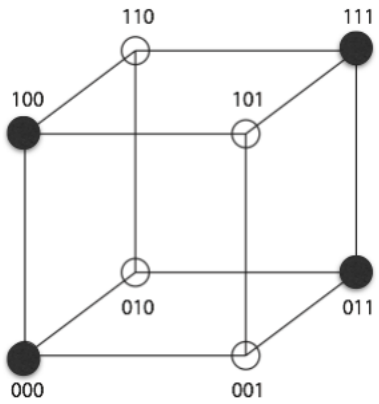

Figure 2.3: Input and Output Nodes in Q_3 by Original Approach

that end with "11" or "00" are considered as output nodes (black nodes), Fig. 2.3 shows the input and output nodes in a 3-cube. Obviously, the number of nodes in both cases are equal so that the first precondition holds.

As for the second precondition, an approach given by Guo et al. [9], maps the remote arcs with the following approach.

First, the input nodes and output nodes will be sorted in an ascending order. For example in $KC(2,2)$ where the hypercube clusters are of dimension 2, the sorted input nodes would be $01, 10$ and output nodes would be $00, 11$. Then we also sort the out-arcs (out-degree edges) and in-arcs (in-degree edges), based on their source and destination in $K(2^{m-1}, k)$ with the following approach. According to the Kautz definition, each node $x_k x_k \dots x_1$ has an arc pointing to the node $x_{k-1} \dots x_1 \alpha$, $\alpha \in \{0, 1, \dots, d\} - \{x_1\}$. We define i, $0 \leq i \leq d$, as the clockwise distance from x_k to α (if $x_k \neq x_1$) and from $x_k + 1$ to α (if $x_k = x_1$). Each arc has a unique i. Fig 2.6 shows the example of clock-wise order of i in $324 \rightarrow 241$. Note that the i^{th} out-arc of any node $x_k \dots x_2 x_1$ is also the i^{th} in-arc of a corresponding node $x_{k-1} \dots x_1 \alpha$. Therefore, out-arcs and in-arcs can be sorted in an ascending order based on i.

To complete our example, consider the same $KC(2,2)$, a Kautz node, say 02, has been replaced by a Q_2 and it has two out-arcs in the corresponding $K(2,2)$:

$o_1 = 02 \rightarrow 20$

$o_2 = 02 \rightarrow 21$

And two in-arcs:

$i_1 = 20 \rightarrow 02$

$i_2 = 10 \rightarrow 02$

Then after sorting, in-arcs ascending order is o_1, o_2 and out-arc ascending order is i_1, i_2.

Now, to constitute a KCube and satisfy the second precondition according to Guo et al. [9], the i^{th} arc of node x in $K(2^{m-1}, k)$ is replaced with a remote arc in $KC(m, k)$ from i^{th} output node in a Q_m to i^{th} input node of the other corresponding Q_m.

Each node in $KC(m, k)$ is denoted by $< x = x_k \dots x_2 x_1, y = y_m \dots y_1 >$, where the first part, x, represents the Kautz-label of that node, and the second part, y, represents the hypercube-label of that node.

2.3 Properties

Since KCube has been proposed recently, there are limited existing studies on it in the literatures. However, several properties of KCube have been proved by Guo et al. [9] and Zhao [37], which are mentioned below.

Theorem 2.3.1. *In a $KC(m, k)$, the number of hypercube clusters is $2^{k(m-1)} + 2^{(k-1)(m-1)}$ and the number of nodes is $2^{k(m-1)+m} + 2^{k(m-1)+1}$ [9].*

Proof. According to the definition of Kautz graph, there is $d^k + d^{k-1}$ nodes. In KCube, each node is replaced by a hypercube cluster of dimension m, and each has 2^m nodes. We also know in KCube, $d = 2^{m-1}$. Therefore, Theorem 2.3.1 holds. □

Lemma 1. *An upper bound on the shortest path between an output node and input node within a hypercube cluster of dimension m is $m - 1$ [9].*

Proof. In a Q_m, the longest length of the shortest path is the path between two complement nodes $x_m \dots x_2 x_1$ and $\overline{x_m} \dots \bar{x}_2 \bar{x}_1$. The definition of input node and output nodes in $KC(m, k)$ implies that the two complement nodes in Q_m, are from the same group, either input or output nodes. Then, going from an input (output) node to an output (input) node will take at most $m - 1$ steps. □

Theorem 2.3.2. *An upper bound on the diameter of $KC(m, k)$ is $m(k + 1) + 1$ [9].*

Proof. Let us assume $u =< x, y >$ and $v =< x', y' >$ are two arbitrary nodes in $KC(m, k)$. We wish to traverse from u to v through a shortest path. We divide such a path into three different parts: source Q_m where u lies, intermediate Q_m's and the destination Q_m where v lies. Considering the diameter of the Kautz graph k, such a shortest path will go through at most $k - 1$ intermediate Q_m's. In the source Q_m, the path needs to take at most m hops from u to the appropriate output node in the same hypercube cluster to get to the first intermediate Q_m. Inside each intermediate cluster, since the path enters through the input node and exits from an output node, according to the Lemma 1, it requires at most $m-1$ steps for

each intermediate hypercube cluster to route the message. For the destination cluster, the same argument as the source cluster holds and m is the maximum number of hops needed to take. Therefore, considering the number of Kautz edges which is k, the total number of steps is bounded by: $2m + (k-1)(m-1) + k = m(k+1) + 1$. □

Theorem 2.3.3. *The KCube is a bipartite graph.*

It is shown by Zhao [37] how this property holds with the bipartite sets of:
$V_1 = \{$ All the nodes with even Hamming weights$\}$
$V_2 = \{$ All the nodes with odd Hamming weights$\}$
Hamming weight is said to be the number of 1's in the binary sequence of each node of the hypercube. It can be observed that the two nodes of the same parity (either two nodes of even or two nodes of odd parity) do not share an edge since by the definition of hypercube, two nodes are directly connected to each other only when their binary sequences differ in exactly one bit. Then V_1 and V_2 are the two bipartition sets in KCube.

We discuss this property in further detail in Chapter 3 where we prove the Hamiltonicity of KCube.

Proposition 1. $KC(m,k)$ *is a regular but not node-symmetric graph [37].*

This is derived from the regularity of the Kautz graph and the hypercube. In the KCube all the nodes have the degree $m+1$, when m is for the hypercube edges and 1 is for an in-degree or an out-degree edge, so KCube is regular. However, it is not vertex-symmetric since there is not an automorphism for all the nodes. It is worth mentioning that Kautz graphs and de Bruijn graphs are also not node-symmetric [7, 23].

2.4 Routing Algorithm

One of the fundamental algorithms of interconnection networks is the routing process of that network and the capability of transferring a piece of data among the processors through the shortest path. Network topologies that are known to have an effective and fast routing algorithm are very desirable to be employed in practice. There are several different routing paradigms, e.g., the classic routing problem from a source node to a destination node, the routing from a source node to multiple destination nodes through disjoint paths, and from a set of nodes to another set of nodes. Combinations of these problems have also been studied, for example, finding the shortest and disjoint paths from a source node to multiple destination nodes [27, 29, 30].

In this chapter we first look at the hypercube routing and the problem of routing with faulty nodes (failed components). Next, we briefly go over the literature related to the routing algorithm of Kautz graphs. Finally, we present the existing algorithm for the KCube routing which utilizes the hypercube and the Kautz graph routing schemes.

2.4.1 Hypercube Routing

The routing algorithm on the hypercube was investigated by Saad and Schultz [31]. Let $H(u_1, u_2)$ be the Hamming distance between two arbitrary nodes in an n-cube. Then, if there is no faulty node, the length of the shortest path between two nodes u_1 and u_2 is equal to $H(u_1, u_2)$. Please note that we also use $H(u)$ to represent the Hamming weight of u, the number of 1's in u's binary representation. Furthermore, the number of parallel or disjoint paths between u_1 and u_2 is also equal to $H(u_1, u_2)$. The shortest path can be generated by correcting the differing bits one by one between u_1 and u_2. As an example of this routing scheme, consider a 3-cube where the source node and the destination node are $u_1 = 010$ and $u_2 = 101$ respectively. Here, $H(010, 101) = 3$, then there is a shortest path of length 3 that can be generated by correcting the differing bits. We can start by changing the leftmost bit and correcting it from left to the right of its binary sequence.

$$u_1 = 010 \xrightarrow{correct\ the\ leftmost\ bit} 110 \xrightarrow{correct\ the\ second\ bit} 100 \xrightarrow{correct\ the\ rightmost\ bit} 101 = u_2$$

2.4.2 Routing in a Faulty Hypercube

An interconnection network in its practical use may consist of a great number of processors. As this number grows, the likelihood of having a failed component due to any unknown reason will increase. Therefore, it is necessary for any network to incorporate fault-tolerant routing algorithms. In particular, this problem in the hypercube has been studied by many researchers such as Chiu and Wu [12], Kaneko and Ito [21], Chen and Shin [10], and Gu and Peng [16].

This problem can be looked at from different angles and has several different versions. One of them that is within the scope of this thesis is to determine whether there exists a shortest path between two complement nodes s and t, i.e., $H(s, t) = n$, in a Q_n when some of the nodes are blocked (the path cannot go through them and must ignore them). This problem will be investigated thoroughly in Chapter 5, but first we review the existing results [12, 21] here. We use the terms "*faulty nodes*" and "*blocking nodes*" interchangeably.

The *unsafe node* notion was introduced by Chiu and Wu [12] to potentially avoid the blocking nodes. A node is defined as an unsafe node if it has either two or more faulty

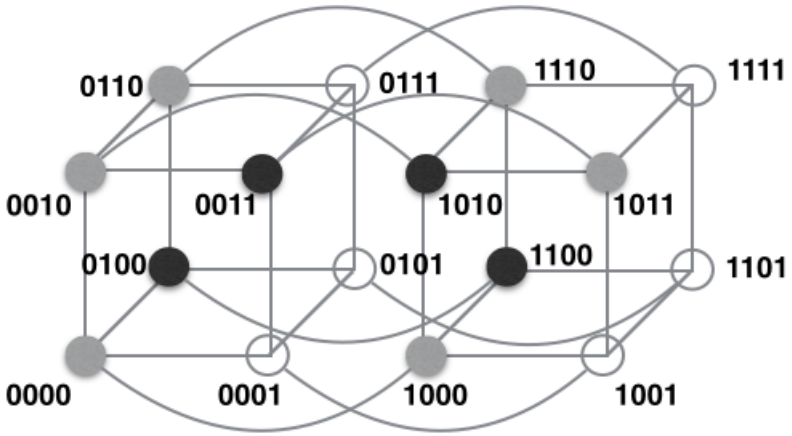

Figure 2.4: The Unsafe Nodes (Gray Nodes), and Faulty Nodes (Black Nodes) in a 4-cube.

neighbours, or three or more faulty or unsafe neighbours. If a node is neither a faulty nor an unsafe node, it is called safe node.

In Fig 2.4, the black nodes are faulty nodes, the gray nodes are unsafe nodes, and the white nodes are safe ones. An algorithm by Chiu and Wu [12] is presented to identify the status of all the nodes (safe, unsafe, or faulty) in an n-cube.

Theorem 2.4.1. *In a faulty hypercube , there is a shortest path between two nodes A and B with the length of $H(A, B)$ if node A is a safe node while B is a non-faulty node [12].*

Proof. Let $h = H(A, B)$. First assume, $h = 1$, obviously, there is a direct edge between A and B. If $h = 2$, there are two disjoint shortest paths between A and B. By definition, A is a safe node and there cannot be two faulty nodes adjacent to A. Then there exists at least one path between A and B with a Hamming distance of two. Now suppose $h \geq 3$. There are h nodes of A's neighbours that lie in different shortest paths from A to B. We know that since A is a safe node, it has at most two faulty or unsafe nearest neighbours. Hence, at least one of these h nodes exists that is guaranteed to be safe. We choose one of them for the next node. Now, $H(A, b)$ has been reduced to $h - 1$. Similarly, we repeat this method as long as the Hamming weight of the current node and the destination node (B) is greater or equal to 3. Finally when we are two steps from B, we apply the above mentioned instruction for the case $h = 2$ and reach the destination. Therefore a shortest path of length h between A and B has been generated. □

Kaneko and Ito [21] expanded this idea, and introduced the notion of full reachability.

Definition 15. *A non-faulty node u is called fully reachable relative to h if there is a path of length h between u and all the nodes in Hamming distance h from u.*

The full reachability algorithm (FR), proposed by Kaneko and Ito [21], is based on fully reachability concept and extends the algorithm by Chiu and Wu [12]. Evaluation of their algorithm shows that it can find paths consisting of no faulty nodes in less time in comparison with the algorithm by Chiu and Wu [12]. These findings are the result of great research efforts made to tackle this problem. In addition, we approach it through another way that complements their results.

2.5 Kautz Graph Routing

The Kautz graph $K(d, k)$ [22] is a directed graph with the maximum degree d and the diameter k . The Kautz graphs are known as highly dense graphs. The maximum number of nodes in a graph of fixed degree d, and diameter k is called *Moore bound* and is equal to $1+d+d^2+\ldots+d^k$ [5]. There is no such graph with the density equal to *Moore bound*, thus it is not feasible to have this number of processors in practice. The Kautz graph has $d^k + d^{k-1}$ nodes which is very close to *Moore bound*, and they are the densest graphs when the diameter is equal to two [25]. According to *Moore bound* it has been shown that the lower bound for the diameter of graph with N nodes and the degree of d is $\lceil log_d(N(d-1)+1)\rceil - 1$ [25]. Now, if we replace N with $d^k + d^{k-1}$, we obtain $\lceil log_d(d^{k+1} - d^{k-1} + 1)\rceil - 1 = k$. This indicates that Kautz graph has an optimal diameter. In fact, between any two nodes in a Kautz graph, there is a unique path [38]. There exists a simple routing algorithm for the shortest path between node x and node y [18, 38], determined by the longest common sequence that is a suffix of x and prefix of y. For example, if the length of such sequence is $k - l$, that is, $x = x_k x_{k-1} \cdots x_1 = x_k x_{k-1} \cdots x_{k-l+1} z_1 \cdots z_{k-l}$ and $y = y_k y_{k-1} \cdots y_1 = z_1 z_2 \cdots z_{k-l} y_l \cdots y_2 y_1$, then the distance between x and y is l and the routing is done as follows:

$$
\begin{aligned}
x_k x_{k-1} \cdots x_1 &= x_k x_{k-1} \cdots x_{k-l+1} z_1 \cdots z_{k-l} \\
&\rightarrow x_{k-1} \cdots x_{k-l+1} z_1 z_2 \cdots z_{k-l} y_l \\
&\rightarrow x_{k-2} \cdots x_{k-l+1} z_1 z_2 \cdots z_{k-l} y_l y_{l-1} \\
&\vdots \\
&\rightarrow x_{k-l+1} z_1 z_2 \cdots z_{k-l} y_l y_{l-1} \cdots y_2 \\
&\rightarrow z_1 z_2 \cdots z_{k-l} y_l \cdots y_2 y_1.
\end{aligned}
$$

Consequently, the diameter of $K(d,k)$ is k.

For example, the steps of routing from $u = 3021$ to $v = 2302$ in a $K(4,4)$ are as follows:

$$u = 3021 \longrightarrow 0212 \longrightarrow 2123 \longrightarrow 1230 \longrightarrow 2302 = v$$

Note that in this example, there is no common sequence as the suffix of u and the prefix of v. Then it requires $k = 4$ steps for the source node to reach the destination.

This interconnection network is fault-tolerant and it is shown that its connectivity is d and there are d node disjoint paths between any two nodes [36]. The existence of these disjoint paths can be effective in faulty Kautz network when the source node contains the information of all the faulty nodes throughout the network so that it can find the fault-free path to its destination. Although the Kautz graph has a good connectivity, having the option of different disjoint paths is effective only when the identity and location of faulty nodes have been broadcasted throughout the network. This process of informing all the nodes about the failed components takes up a lot of space and bandwidth. A distributed fault-tolerant routing is proposed by Chiang and Chen [11] that fully covers this problem.

2.6 KCube Routing

The routing algorithm in the KCube [9] uses the routing of the Kautz graph and the routing of the hypercube. Suppose we wish to route a message from $u =< x, y >$ to $v =< x', y' >$ in a $KC(m,k)$. According to the routing algorithm of Kautz graph we can determine the next hypercube cluster to send a message to, and subsequently we can find the i^{th} output node in the source hypercube cluster Q_m. Then if the source node is not the appropriate output node, we perform a hypercube routing inside the Q_m in order to transfer the message to the desirable output node. Otherwise, the message will be exited directly from the output node. At each step the next hypercube cluster is chosen based on the corresponding $K(d,k)$ routing steps, and inside the hypercube clusters the hypercube routing is used to reach from the input node to the appropriate output node. Finally, at the last hypercube cluster where v resides, if the destination is the input node which the path has entered from, the process is finished. Otherwise, it performs another hypercube routing to reach the node $v =< x', y' >$.

For example, consider the $KC(3,2)$, where the corresponding Kautz graph is $K(4,2)$. Given the source node $s =< 324, 101 >$, and the destination node $t =< 413, 001 >$, the path between them will be generated by the following steps.

First, we can determine the next hypercube cluster based on the Kautz routing which is 241. The arc $324 \rightarrow 241$ has order $i = 3$, because of the clockwise distance from 3 to 1.

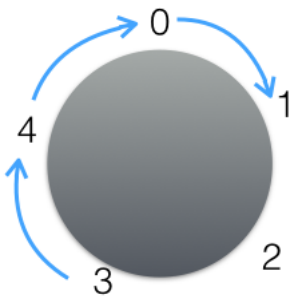

Figure 2.5: Clock-Wise Order of i. $i = 3$ in: $324 \rightarrow 241$.

Please see Fig 2.5. Thus, we need to go to the third output node in the first Q_3 which is 100:
$< 324, 101 > \rightarrow < 324, 100 >$, and then it can exit that Q_3 and get to the next Q_3, where third input node $< 241, 101 >$ is the appropriate node for entering the next cluster:
$< 324, 100 > \rightarrow < 241, 101 >$. Here, since the next Kautz arc is $241 \rightarrow 413$, the clockwise distance between 2 and 3 is one and $i = 1$. So we route $< 241, 101 >$ to the first output node inside the same hypercube cluster: $< 241, 101 > \rightarrow < 241, 100 > \rightarrow < 241, 000 >$. Then, it goes to the next and final Q_3, $< 241, 000 > \rightarrow < 413, 001 >$. Note that the entering input node is exactly the destination node, therefore, we do not need to perform any hypercube routing in the last Q_3.

Chapter 3

Topological Properties of the KCube Graph

3.1 A New Methodology for the KCube Construction

In Chapter 2 we saw that a KCube graph needs to satisfy two preconditions. This definition gives us options to construct a KCube by assigning different input and output nodes. In this chapter we introduce a new way to partition input and output nodes. We show that this new KCube allows us to prove one important property of interconnection networks, Hamiltonicity. Moreover, the new approach of KCube construction increases the flexibility of routing throughout the network which is explained in more detail.

3.1.1 The New Arrangement of the Input/Output Nodes

It is shown by Zhao [37] that the KCube is bipartite where the input and output nodes are defined as described in Guo et al. [9]. It is well-known and can be easily observed that the hypercube is bipartite. Thus, hypercube nodes can be divided into two disjoint sets V_1 and V_2 where for any edge $e = (u, v)$, $u \in V_1,\ v \in V_2$. This means there is no such edge that its nodes lie in the same bipartite set. In the hypercube the bipartite sets can be created based on Hamming weights of nodes, that is, the nodes with an odd Hamming weight form the V_1 and the nodes with an even Hamming weight form the V_2 and in this fashion, we have $|V_1| = |V_2|$. We make use of this property of the hypercube to partition the input and output nodes in a KCube. Thus all the nodes with an even Hamming weights constitute output nodes and all the nodes with odd Hamming weights constitute input nodes. Fig 3.1 illustrates such a partitioning. As a result, the first precondition will be satisfied automatically since the number of nodes with odd Hamming weights and nodes

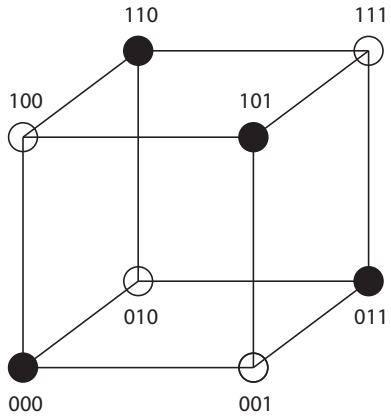

Figure 3.1: Input Nodes (Black) and Output Nodes (White) in Q_3

with even Hamming weights are both equal to 2^{m-1}.

Our approach of defining input and output nodes has several advantages that are discussed further. One that immediately will be achieved is that each input node now is connected to m output nodes that each one has a direct link to m different clusters. Similarly, each output node is connected to m input nodes which are pointed to by m clusters. While in the KCube definition by Guo et al. [9] each input (output) node is only connected to two output (input) nodes.

When $m = 2$, $d = 2^{m-1} = 2$, the KCubes of the two different versions become the same. Fig 2.2 shows $KC(2, 2)$. Clearly, the KCube defined with our approach of node partitioning is also bipartite.

In the following sections we will see how this new definition will affect different aspects of the KCube network.

3.2 The Average Distance Between Two Nodes

In Chapter 2 we discussed the routing algorithm of the original KCube according to the old definition of input and output nodes. The upper bound $m(k + 1) + 1$ for diameter of $KC(m, k)$, obtained in [9], is based on the fact that the largest length of the shortest path between an input and output node inside an intermediate hypercube cluster is $m - 1$. Intermediate clusters are referred to as the clusters other than the first and last clusters where the source node and destination node reside. Now, after our way of defining KCube, the following result will be achieved.

Lemma 2. *The average number of steps required to reach from an input node to an output node in a $KC(m, k)$ inside an intermediate hypercube cluster is $\frac{m}{2}$.*

Proof. Since the hypercube is node-symmetric, without loss of generality we assume the entering input node is 0^m that wishes to reach to the appropriate output node to route the message to the next cluster. There are 2^{m-1} different output nodes and we want to examine the distance of each from the node 0^m. Changing each 0 to 1 will increment the Hamming weight by one and as a result, after taking each step, the group of output nodes or input nodes will be changed to the other group. In other words, nodes at distance one are output nodes, nodes at distance two are input nodes, nodes at distance three are output nodes and so on. Therefore, all the output nodes are residing at odd distance from node 0^m. Since we assumed we enter the Q_m from node 0^m, we can simply count the nodes with the same Hamming weight which also represent the distance from 0^m:

$$if\ m\ is\ odd : 1 * \binom{m}{1} + 3 * \binom{m}{3} + \ldots + m * \binom{m}{m}$$

$$if\ m\ is\ even : 1 * \binom{m}{1} + 3 * \binom{m}{3} + \ldots + (m-1) * \binom{m}{m-1}$$

There is 2^{m-1} output nodes, then the average distance in both cases will be:

$$\frac{1 * \binom{m}{1} + 3 * \binom{m}{3} + \ldots + m * \binom{m}{m}}{2^{m-1}} = \frac{m}{2};\ \frac{1 * \binom{m}{1} + 3 * \binom{m}{3} + \ldots + (m-1) * \binom{m}{m-1}}{2^{m-1}} = \frac{m}{2}$$

□

Theorem 3.2.1. *The average distance between two nodes in Kautz graph, $K(d,k)$, is $k - \frac{1}{d-1}$ [33].*

It can be seen that the average distance in a Kautz graph is very close to its diameter and in fact, it can be approximated by k . Using this result together with what we have achieved in lemma 2, we are able to determine the average distance between a pair of nodes in a KCube network.

Corollary 3.2.2. *The average distance between two nodes, residing in two different hypercube clusters in a $KC(m,k)$, is $\frac{m(k+1)}{2} + k$.*

Proof. There are $k+1$ hypercube clusters to traverse as well as k remote arcs (Kautz edge):

$$t(n) = \underbrace{(k+1) * \frac{m}{2}}_{hypercube\ steps} + \underbrace{k}_{Kautz\ steps} = \frac{m(k+1)}{2} + k$$

□

This number is almost half of the upper bound achieved in Guo et al. [9]. As we expect intuitively, the new arrangement of input and output nodes has made the routing procedure among the processors of the network more flexible.

In the next section we study the Hamiltonicity property of our proposed KCube network.

3.3 Hamiltonicity of KCube

The Hamiltonicity property for an interconnection network is one of many important properties relevant to parallel computing and has been studied for many interconnection networks, e.g. [19, 38]. For example, if a network is Hamiltonian, then we can embed a linear array into the network so that all algorithms designed for linear arrays can be readily executed on the network as well. As the embedding capability increases, the network can be a host to more guest graphs. Hamiltonicity is also used to create independent spanning tree and results in designing fault-tolerant protocols [15].

In this section we first review some properties of the hypercube and Kautz graph needed to prove the existence of a Hamiltonian cycle in KCube. A valuable attempt has been made by Zhao [37] to show that $KC(1,k)$ and $KC(2,k)$ are Hamiltonian, while we prove there is a Hamiltonian cycle in the general $KC(m,k)$ using our definition of input and output nodes.

Definition 16. *A connected bipartite graph with bipartitions V_1 and V_2 is called Hamiltonian-laceable if it has $u \rightsquigarrow v$ Hamiltonian path for $\forall u, v$ where $u \in V_1$ and $v \in V_2$.*

Graphs possessing this property allow us to start the path from one arbitrary node which lies in one set of bipartition, traverse a Hamiltonian path, and terminate it at any node lying in the other bipartition set. Fig 3.2 shows a Hamiltonian path in a bipartite graph that starts from one set of bipartition and ends at the other set of bipartition.

Theorem 3.3.1. *All hypercube graphs are Hamiltonian-laceable [8].*

This property of hypercubes can be proved by induction on the dimension of the hypercube Q_m. For the case $m = 1$ it is trivial that the statement holds. Suppose Q_m is Hamiltonian-laceable. If we join the two identical copies of Q_m to form a Q_{m+1}, the Hamiltonian paths of Q_m's can also be joined to form a Hamiltonian path in Q_{m+1}.

Theorem 3.3.2. *Kautz digraphs are Eulerian [2].*

This is because a digraph G has an Eulerian cycle if and only if for each node v of G, its in-degree is equal to its out-degree [19] and this condition is satisfied by the Kautz graph.

Now we are ready to show the Hamiltonicity of KCube graph.

Theorem 3.3.3. *$KC(m,k)$ is Hamiltonian.*

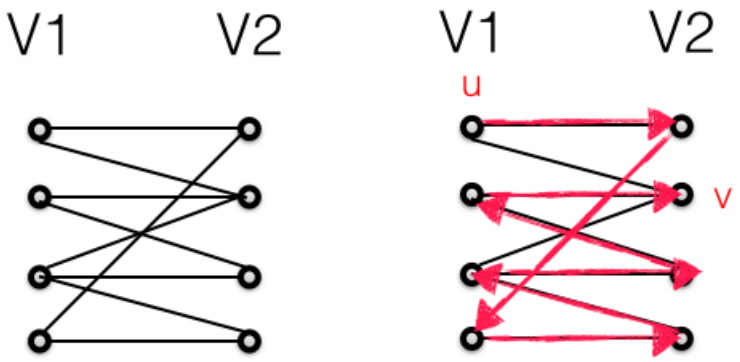

Figure 3.2: Hamiltonian Path $u \rightsquigarrow v$ in a Laceable Bipartite Graph

Proof. Recall that in a KCube $KC(m,k)$ corresponding to the Kautz graph $K(d,k)$, we have $d = 2^{m-1}$.

For $m < 3$:

If $m = 1$, then $d = 2^{1-1} = 1$ and the corresponding Kautz graph $K(1,k)$ has exactly two nodes that form a loop, for any $k \geq 1$. In this case, the hypercube cluster used is Q_1 with two nodes and $KC(1,k)$ is simply a 4-cycle, thus Hamiltonian. See Fig 3.3.

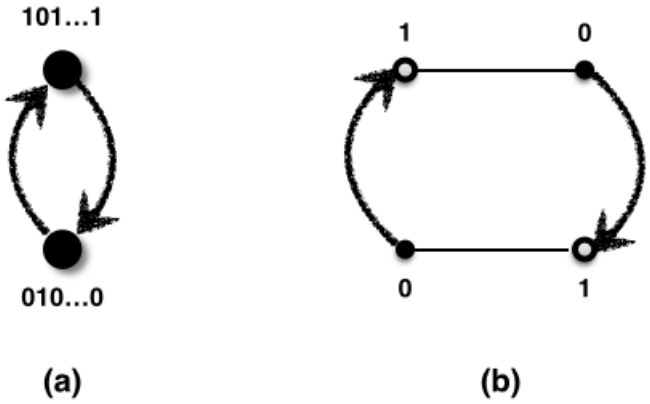

Figure 3.3: a. $K(1,k)$; b. $KC(1,k)$

If $m = 2$, we show that $KC(2,k)$ is Hamiltonian for any $k \geq 1$. Since $K(2,k)$ is Eulerian, we take any such a cycle. The in-degree and out-degree of $K(2,k)$ are 2, which means that each node of it is entered twice and exited from twice. Now consider

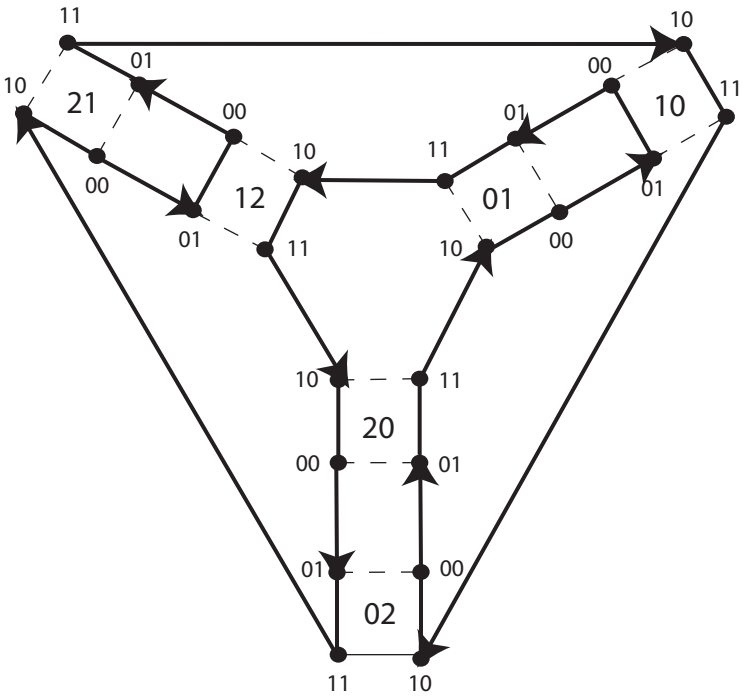

Figure 3.4: A Hamiltonian Cycle in $KC(2,2)$.

the $KC(2,k)$ associated with this $K(2,k)$. When the Eulerian cycle enters a hypercube cluster, it does so through an input node. Now it needs to go to another cluster which can be done by going to an output node which is directly connected to this input node. In fact, since the hypercube cluster is a 2-cube, this input node is directly connected to both output nodes. So no matter which Eulerian cycle is used, we can combine the Eulerian cycle with the fact that each hypercube cluster is a 2-cube to obtain a Hamiltonian cycle. For example, if we pick the Eulerian cycle

$$02 \to 21 \to 12 \to 21 \to 10 \to 02 \to$$
$$20 \to 01 \to 10 \to 01 \to 12 \to 20 \to 02$$

in $K(2,2)$, then the Hamiltonian cycle associated with it in $KC(2,2)$ by our construction is given in Fig 3.4.

For $m \geq 3$:

It is well-known that all Kautz graphs are Hamiltonian [18]; we know that $K(d,k)$ has a Hamiltonian cycle. This Hamiltonian cycle can then be used to construct a Hamiltonian cycle in $KC(m,k)$ that is associated with $K(d,k)$ where $d = 2^{m-1}$. To see this, we use the result from Theorem 3.3.1 that a hypercube is Hamiltonian-laceable. Let the Hamiltonian cycle of the $K(d,k)$ be $u_1 \to u_2 \to \ldots \to u_n \to u_1$. We start from an output node v in the

hypercube cluster corresponding to u_1. Following the Hamiltonian cycle, we go to an input node of the hypercube cluster corresponding to u_2. In general, When the Hamiltonian cycle enters a node in $K(d,k)$, it now enters an input node in the corresponding $KC(m,k)$, which then traverses through all the nodes in the Q_m through a Hamiltonian path (guaranteed to exist by Theorem 3.3.1) to an output node, which then leaves the node to go to an input node from another cluster which is the next node in the Hamiltonian cycle of the $K(d,k)$. Please see Fig 3.5. Finally, we will go back to an input node in the cluster corresponding to u_1 which can then go to the original output node in the same cluster, forming a Hamiltonian cycle in the $KC(m,k)$.

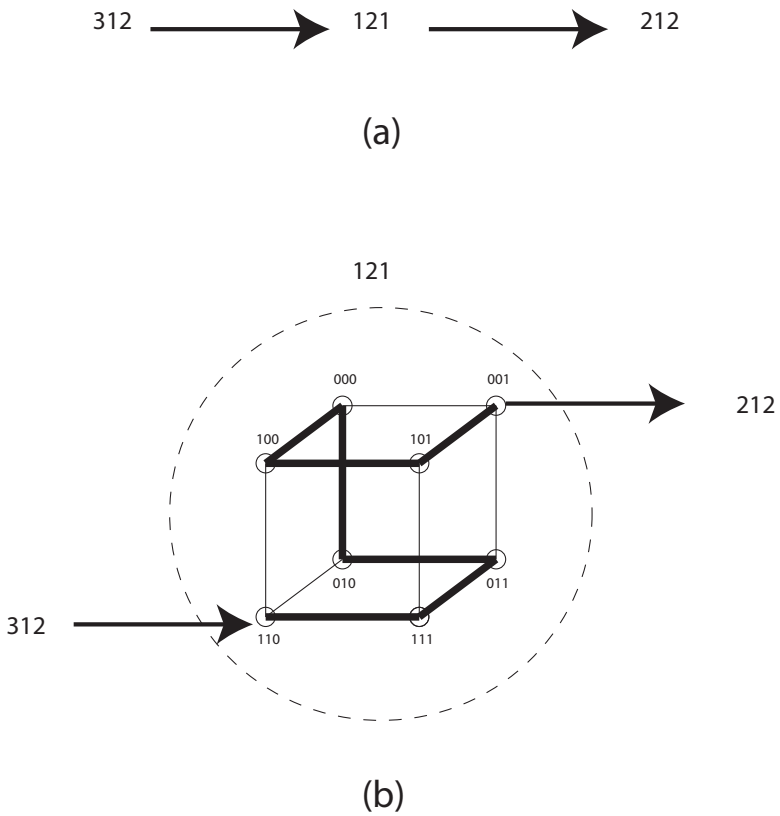

Figure 3.5: a. Part of Hamiltonian Cycle in $K(4,3)$; b. Part of Hamiltonian Cycle in $KC(3,3)$

□

3.4 Connectivity of KCube Networks

A graph G is n-connected if there exist n internally node-disjoint paths between any pair of nodes. The connectivity $\kappa(G)$ of G is the greatest integer n such that G is n-connected [19].

The connectivity of a network is directly related to the network's fault tolerance property.

In $KC(m,k)$ where $d = 2^{m-1}$, $m+1$ is an upper bound on its connectivity because there can be at most $m+1$ disjoint paths between two nodes with a degree of $m+1$ where the source is an output node and the destination is an input node.

We have the following result on the connectivity of the KCube whose proof is straightforward:

Lemma 3. *The connectivity $\kappa(KC(m,k))$ of the KCube is m.*

Proof. It is shown that the $\kappa(K(d,k)) = d$ [18] and $\kappa(Q_m) = m$ [19]. Also, that each node u in the KCube can be uniquely written as $u =< u_1, u_2 >$ where u_1 is the Kautz label of u and u_2 is the hypercube label. Consider two arbitrary nodes $u =< u_1, u_2 >$ and $v =< v_1, v_2 >$. We divide the proof into two cases:

Case 1: If u and v are in the same hypercube cluster, namely, $u_1 = v_1$, then clearly there are m disjoint paths between u and v since Q_m is m-connected. There is one more disjoint path that can possibly exist if u is an output node and v is an input node. In that case, the path traverses from u using Kautz edges and ends the path at v. Note that no edges of the hypercube clusters that u and v reside are used. However, in other three possibilities for u and v's group, namely, input/input, output/output, and input/output, there will be m disjoint paths from u to v, since the path from u has to use inside hypercube cluster edges to reach v .

Case 2: If u and v are in two different hypercube clusters. We show that in this case that there are also m node-disjoint paths between u and v. In the hypercube with Kautz label u_1, we can first travel from node u to m different output nodes (note that each hypercube cluster has 2^m nodes that half of them are output nodes, the other half are input nodes, $d = 2^{m-1}$, and m such paths are guaranteed to exist). Each of these output nodes is a starting point of a unique path to an input node of the hypercube cluster of v_1, and there are m such paths (guaranteed to exist by the connectivity of the $\kappa(K(d,k)) = d$ where $m \leq d$, and the fact that each input node of a cluster is connected to one unique output node from another cluster). Once inside the hypercube cluster v_1, there exist m disjoint paths from these m input nodes to the node $v = (v_1, v_2)$ since Q_m is m-connected. Similar to case1, there can be $m+1$ disjoint paths only when u is an output node and v is an input node. In that case, in addition to the paths explained before, there is one more path that does not use hypercube edges of clusters which u and v reside. In other three possibilities of u and v's group, namely, input/input, output/output and input/output, there cannot be a path that avoids hypercube edges of source and destination clusters. □

It is worth noting that if we treat the KCube as an undirected graph, then the connectiv-

ity would be $m + 1$. This can be obtained from the fact that each node in the KCube has a degree of $m + 1$. Therefore, removing $m + 1$ neighbours of a node will make this node an isolated one and the graph will become disconnected while after removal of m nodes the graph stays connected.

Table 3.1 illustrates a comparison among some properties of the hypercube, the Kautz graph and the new version of KCube.

Table 3.1: Properties of Interest of the Hypercube, Kautz and the New Version of KCube Network

Graph	Network Size	Degree	Hamiltonian	Connectivity
$Q(m)$	$N_H = 2^m$	m	Yes	m
$K(2^{m-1}, k)$	$N_K = 2^{(m-1)}K + 2^{(m-1)(k-1)}$	2^m	Yes	2^m
$KC(m, k)$	$N_{KC} = N_H \times N_K$	$m + 1$	Yes	m

Chapter 4

Broadcasting on KCube Networks

4.1 Introduction

There exist a host of communication problems in any interconnection network, such as the problem of broadcasting, gossiping, and total exchange, etc. In this chapter, we consider the broadcasting algorithm, one of the most important communication primitives on an interconnection network, where one processor (source) wishes to send a piece of data to all other processors in the network. This communication problem has been widely studied for many network topologies including the de Bruijn graph and Kautz graph [3, 28], and naturally, the hypercube [20, 32]. In an interconnection network where processors are connected according to a certain topology, communications among processors are accomplished by sending messages along the interconnection links. Two possible communication models exist for each node of the network: *single-port* and *all-port*. In a single-port model, in one time unit, a processor can send (receive) at most one datum of fixed length to (from) one and only one of its neighbours. On the other hand, in an all-port model, in one time unit, a processor can send (receive) one datum of fixed length to (from) all its neighbours. For our discussion, we assume the single-port model. For any interconnection network with a total of N processors, a trivial lower bound is its diameter. Another trivial lower bound is $\Omega(\log N)$. This is because after each broadcasting step, the number of processors with the message can at most double. Therefore, for $KC(m, k)$, the lower bound for broadcasting is $\Omega(\log((d^k + d^{k-1})2^m))$ = $\Omega(k \log d)$ = $\Omega(km)$ where $d = 2^{m-1}$. For the n-dimensional hypercube Q_n with 2^n nodes, the broadcasting can be done easily in optimal time $O(\log 2^n)$ = $O(n)$.

4.2 Kautz Digraph Broadcasting and Spanning Tree

As the first step, we review some definitions needed for this chapter and then we dicuss Kautz digraph broadcasting and spanning tree [14] of this network.

Definition 17. *A spanning tree T of graph G is a subgraph of G in which $V(T) = V(G)$ where T is a tree (a graph that contains no cycle).*

Definition 18. *A d-ary tree is defined to be a tree whose nodes have no more than d children.*

Definition 19. *For any digraph G and node $u \in V(G)$, a broadcast spanning tree of G with root u is defined to be a spanning tree T_u of G such that for any node v, if nodes receive the message directly from v, then there are arcs between those nodes and v [28].*

Fig 4.1 illustrates a de Bruijn graph $B(2, 3)$ and its corresponding broadcast spanning tree.

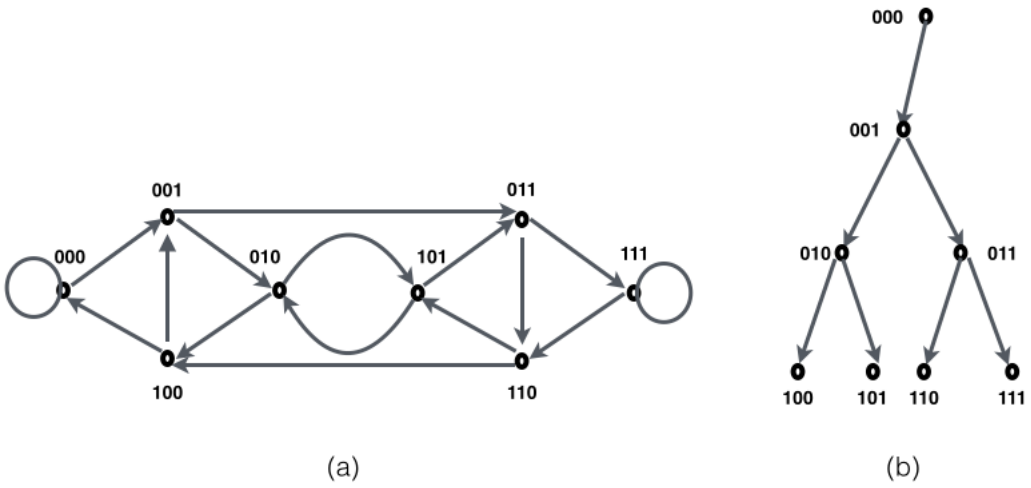

Figure 4.1: a. $B(2, 3)$; b. Broadcast Spanning Tree of $B(2, 3)$ with Root 000

Since the diameter of $K(d, k)$ is k, from any node, all other nodes can be reached in at most k steps. Additionally, the following lemma from [28] implies that we are able to generate the broadcast tree in $K(d, k)$:

Lemma 4. *For any Kautz digraph $K(d, k)$ with $d \geq 2$ and $k \geq 2$, and any vertex u, there exists a spanning d-ary tree of $K(d, k)$ of depth k with root u, and this depth k is the smallest possible.*

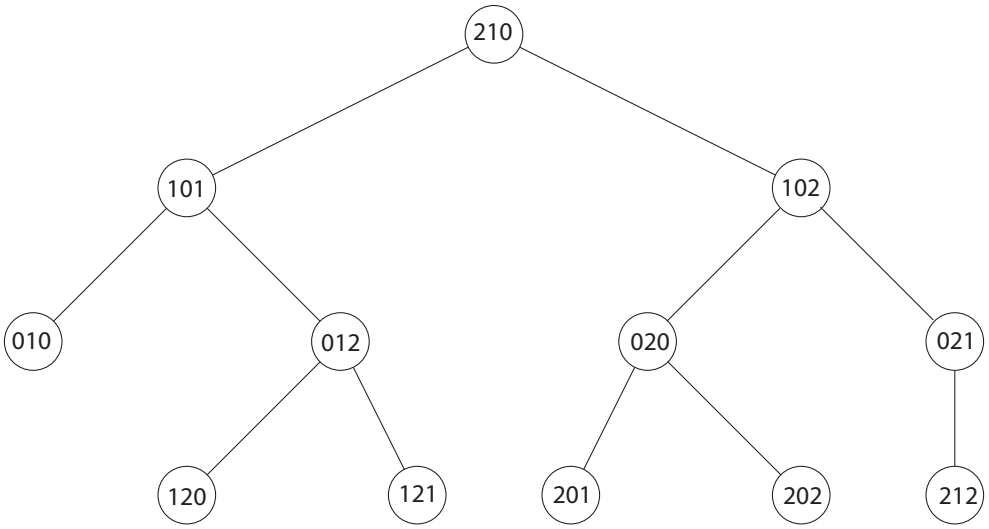

Figure 4.2: Spanning Tree in $K(2,3)$ Starting in Node 210

It can be observed that since the diameter is k, the depth of the spanning tree cannot exceed k. Each node $x_k x_{k-1}...x_1$ of the tree has children from left to right $x_{k-1}...x_1\alpha, \alpha \in \{0,1,...,d\}-\{x_1\}$ except if they already appear at a previous level of the tree, in which case the corresponding branch stops. Fig 4.2 is an example of generating a d-ary broadcasting spanning tree for $K(2,3)$.

However, the spanning tree generated by this approach is not guaranteed to be generic. This property is important in terms of hardware implementation efficiency [35]. A generic spanning tree for a graph with diameter k and fixed out-degree d has two conditions:

1. All leaf nodes are at distance k from the root

2. All non-leaf nodes have d outgoing links.

By applying an approach from [35], we can make the broadcast spanning tree for $K(d,k)$ almost generic. It uses specific nodes for source nodes called *twin* nodes. Twin nodes are pairs of nodes with double links or parallel links, i.e., they both have out-degree arcs to each other. Such nodes in $K(2,3)$ are node pairs $\{010,101\}$ or $\{202,020\}$.

In [35], it has been shown that if the source node is among the twin nodes, we can obtain an almost generic spanning tree. That is, all leaf nodes are at distance k from the root and it is a d-ary tree with the exception of the twin brother node of the root. This twin brother node of the root has $d-1$ outgoing edges.

In this process, since the broadcasting spanning tree is d-ary, in each unit of time, or level of the tree, each processor can send the message to d other processors through its out-degree links according to Kautz digraph definition, except for the twin brother node of the root which has $d-1$ outgoing links.

Fig 4.3 shows the almost generic spanning tree of $K(2,3)$.

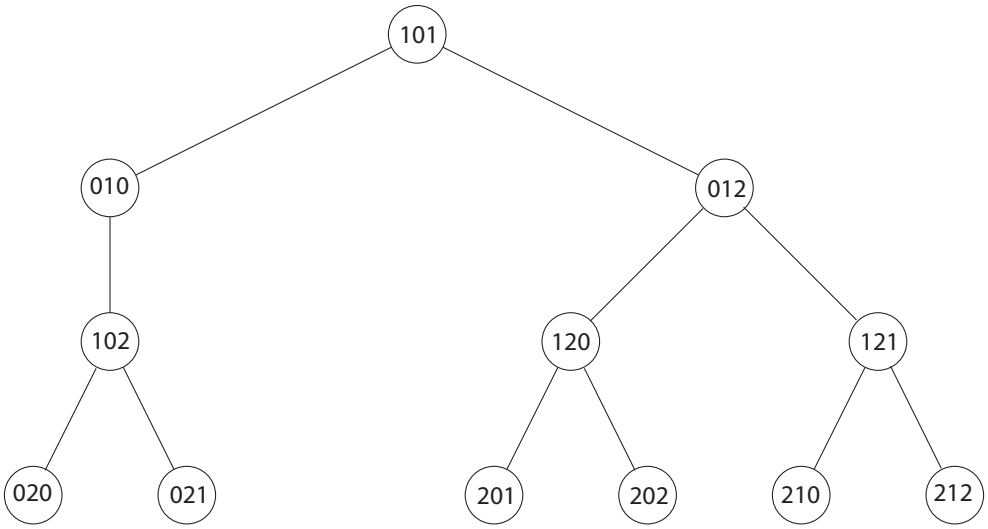

Figure 4.3: Almost Generic Spanning Tree in $K(2,3)$ Starting in Twin Node 101

Hypercube Broadcasting

According to the hypercube construction, in an n-cube, a source node can easily broadcast a message to all other processors in a single-port model with the following scheme [31]:

Each node is denoted by $x_n x_{n-1} \dots x_1,\ x_i \in \{0,1\}$. Since the hypercube is node-symmetric, without loss of generality, we assume the source node containing the initial message is 0^n. At the first step, it will communicate the message with node $0^{n-1}1$ through the direct edge by changing the first (rightmost) bit. Now, the number of informed nodes is doubled. At the second step, these nodes will communicate the message through the edge corresponding to their second bit. Generally, the communications at step i occur by changing the i^{th} position of the binary sequence of informed nodes. In other words the message is sent through the edge corresponding to the i^{th} bit of the nodes. This process will be completed after $\log_2 N$ (N is the number of processors) steps, because at each step the number of nodes that receive the message is doubled. Fig 4.4 shows the steps of broadcasting in a 3-cube. Note that the black nodes are those who have received the message by the time of that step.

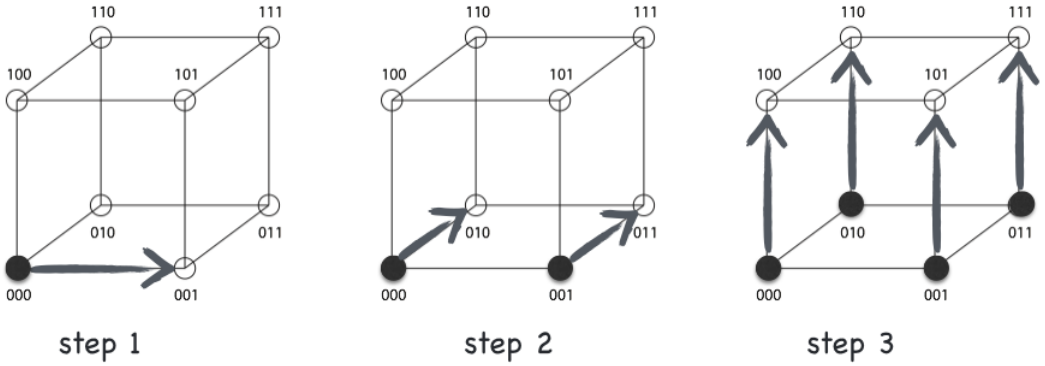

Figure 4.4: Broadcasting Steps in 3-cube

This process can be performed starting from any node of the n-cube. Furthermore, the nodes that are engaged in step i will form a hypercube of dimension i if the missing edge between the nodes are added.

4.3 Broadcasting on the KCube Network

We know that $KC(m, k)$, where m is the dimension of hypercube clusters and k is the corresponding Kautz graph's diameter, is a graph obtained by combining a Kautz digraph and hypercubes such that each node of the Kautz digraph is replaced by a hypercube of appropriate dimension. Our broadcasting scheme in the KCube takes advantage of this composition. Hypercube broadcasting also plays an essential role. The idea for our broadcasting algorithm is simple: we first perform the hypercube broadcasting in the hypercube cluster C where the source node resides. This cluster C, when viewed as a single node in the almost generic spanning tree, is connected to d other clusters (also viewed as nodes). This hypercube cluster is connected to d other clusters via its output nodes and the input nodes of the other clusters. The step when C sends its message to its d children in one step (all-port model) in the spanning tree can now be accomplished in constant time in a single-port model on the corresponding KCube. These clusters that just get the message can perform a hypercube broadcasting, all in parallel, and the process continues to the next level of clusters in the spanning tree and so on until all leaf clusters all have the message. This broadcasting algorithm is for the source node whose Kautz label is a twin node. If the source node is not a twin node, it can route the message to a twin node first in $O(m)$ time. Fig 4.5 illustrates broadcasting in $KC(2, 3)$. In this example the node $< 101, 00 >$ wishes to broadcast a piece of data to all other nodes. First, by performing the hypercube

broadcasting all the nodes will be informed in the first and starting hypercube cluster. Afterwards, through the output nodes of that cluster (black nodes), the message are sent to the hypercube clusters directly connected to C. By repeating this, the broadcast spanning tree for $KC(2,3)$ will be generated.

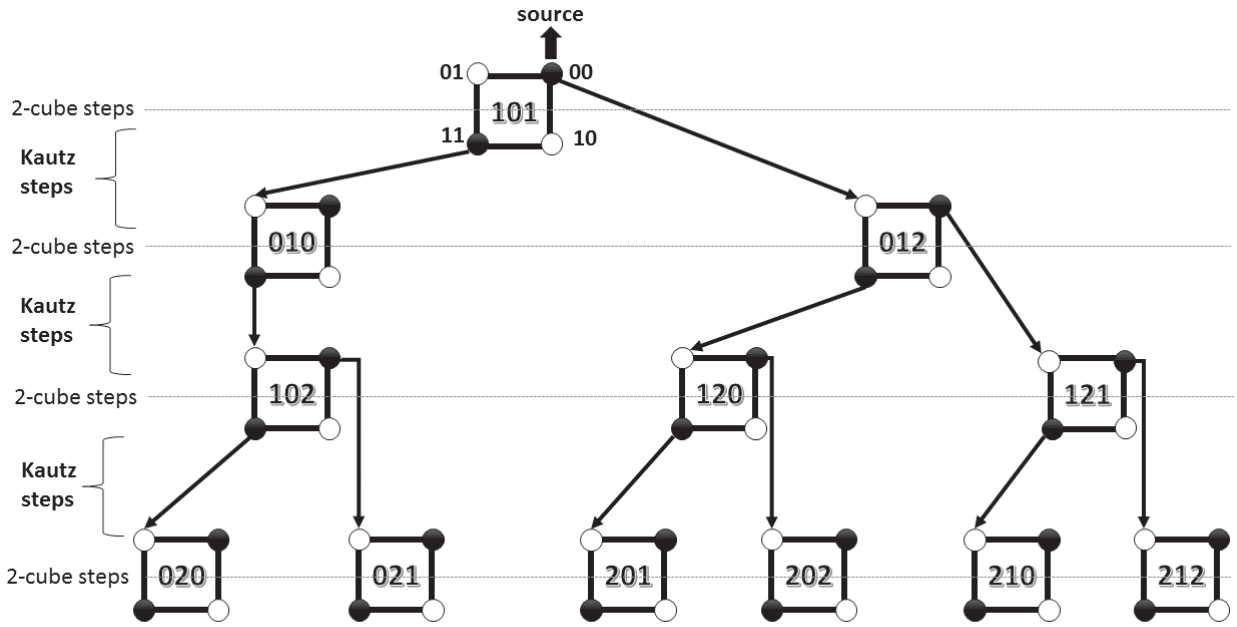

Figure 4.5: Broadcast Spanning Tree in $KC(2,3)$

As for the time of our broadcasting algorithm, since the broadcasting in Q_m takes $O(m)$ time and the spanning tree has height k, the total time is $O(mk)$ which is optimal in view of the lower bound derived earlier.

Chapter 5

Blocking Node Problem

5.1 Introduction

Routing is one of the most fundamental problems in an interconnection network. There are various routing problems. For example, given a source node s and a target node t, we may want to find a routing path between s and t. We may also ask such a path be the shortest. Similarly, we may want to find multiple disjoint paths between the two nodes. And we can also impose the condition that these disjoint paths be the shortest ones. Several other well-known disjoint path paradigms exist [26]. For example, we may want to find (1) disjoint paths between two nodes; (2) disjoint paths from one fixed node to a set of nodes; and (3) disjoint paths from a set of nodes to another set of nodes of the same cardinality. Routing algorithms are useful in designing efficient and fault-tolerant routings for the corresponding networks.

We study the problem of finding a shortest path from a source node to a target node in the hypercube with the presence of a number of blocking or faulty nodes. Fault-tolerant routing for the hypercube has been studied extensively and a vast amount of literature exists. In Chapter 2, we discussed the work by Chiu and Wu [12] as well as Kaneko and Ito [21]. Our work examines the problem from a different angle by giving some simple conditions for these paths to exist. These conditions imply a new routing algorithm which complements the previous routing and is more efficient in certain situations.

In this chapter, we first present two conditions for the shortest path to exist. Next, a routing algorithm is developed that determines the existence of such a path with no restriction on the number of blocking nodes.

5.2 Conditions for Shortest Path Routing with Faulty Nodes on the Hypercube

When some nodes in the hypercube become faulty, the shortest path routing is to find a shortest path without using any of the faulty nodes. By symmetry, without loss of generality, we can assume that one of the nodes is 0^n and the other 1^n.Thus our problem can be stated as follows.

Given nodes $s = 1^n$ and $t = 0^n$ in an n-cube (i.e., $t = 0$ and $s = 2^n - 1$), and k other nodes b_1, b_2, ..., b_k (we can call them blocking nodes or faulty nodes), $b_i \neq s$, $b_i \neq t$, $1 \leq i \leq k$, does there exist a shortest path from s to t that does not intersect any blocking node b_i?

In Chapter 2, we reviewed the work by Chiu and Wu [12] toward fault-tolerant routing in the hypercube. As defined earlier, a non-faulty node is unsafe if it is adjacent to at least two faulty nodes or more than two faulty or unsafe nodes. A non-faulty node is safe if it is not unsafe [21]. We saw the proof in Chapter 2 by Chiu and Wu [12] that if either the source or the target node is safe, then there is a shortest path from s to t involving no faulty nodes. Clearly, it is possible that even if both s and t are unsafe, a shortest path could still exist. Therefore, the condition that either the source node or the target node is safe is simply a sufficient condition for a shortest path to exist between the source and target nodes.

An improved version of the above routing algorithm is given by Kaneko and Ito [21] where the notion of full reachability is introduced. A non-faulty node u is fully reachable with respect to Hamming distance h if every non-faulty node which is distance h away from u is reachable (involving no faulty nodes) from u by a path of length h [21]. Simulation results show that their algorithm performs better than the one from [12] in that larger percentage of paths found are shortest paths.

In [17], it was assumed that the number of faulty nodes is no more than $n - 1$, node failures occur dynamically, and each node knows the faulty status of its neighbours. It is worth pointing out that in their routings, each node knows the states of its nearest neighbours only, while we know all the blocking nodes. Thus their problems are not the same as ours.

In view of the above, our work studies the routing problem from a different angle and in a sense, complements the previous work.

We will first consider the case when the number of blocking nodes is less than n. This condition was also pointed out in [12].

Lemma 5. *In Q_n, $n \geq 2$, if the number of blocking nodes is less than n, then there exists a shortest path from s to t that does not intersect any of the blocking nodes.*

Proof. If $n = 2$, with at most one blocking node, it is clear that there is a shortest path from 11 to 00 avoiding the blocking node. We now assume $n \geq 3$.

If there exists an $(n-1)$-cube containing node 0 with exactly $n-1$ blocking nodes, without loss of generality, let these blocking nodes be

$$\begin{array}{cccccc} 0 & b_{1,n-2} & b_{1,n-3} & \cdots & b_{1,1} & b_{1,0} \\ 0 & b_{2,n-2} & b_{2,n-3} & \cdots & b_{2,1} & b_{2,0} \\ \vdots & & & & & \\ 0 & b_{n-1,n-2} & b_{n-1,n-3} & \cdots & b_{n-1,1} & b_{n-1,0} \end{array}$$

In this case, since 10^{n-1} is not a blocking node, we can route 1^n to 0^n as follows: $1^n \xrightarrow{*} 10^{n-1} \rightarrow 0^n$.

If there is no such an $(n-1)$-cube containing 0 with $n-1$ blocking nodes, then each column of the blocking nodes has at least one 1. In this case, since there are n 1's in s while we only have $n-1$ blocking nodes, there must be one dimension i such that if we set the 1 of s in that dimension to 0, the resulting node is different from all the blocking nodes. Therefore, we also have an $(n-1)$-cube containing 0 that has at most $n-2$ blocking nodes. As another example, let $n = 3$, and $b_1 = 100$ and $b_2 = 011$. We want to flip one bit of $s = 111$ so that it is not equal to any of the blocking nodes. In this case, we can make it to 101 or 110. If we make it to $s' = 101$, we can throw out 011, as it is no longer in the 2-cube containing 000, 100, and 101 since 011 has a 1 in column 1. Thus, what we do now is to look for a path from 101 to 000, with at most one blocking node, in a 2-cube which can be done by the inductive assumption. In fact, the path is $111 \rightarrow 101 \rightarrow 001 \rightarrow 000$. The situation is similar if we change 111 to 110.

In both cases, the proof is then completed by induction. □

We now look at the case when the number of blocking nodes is n.

Lemma 6. *In Q_n, $n > 2$, with n blocking nodes b_1, b_2, ..., b_n, there exists a shortest path from $s = 1^n$ to $t = 0^n$ that does not intersect any of the blocking nodes if and only if*
(a) $\{b_1, b_2, \cdots, b_n\} \neq \{011...11, 101...11, \cdots, 111...10\}$ and
(b) $\{b_1, b_2, \cdots, b_n\} \neq \{00...001, 00...010, \cdots, 10...000\}$,
i.e., at least one neighbour of s is not a blocking node and at least one neighbour of t is not a blocking node.

Proof. The "only if" part is trivial because if such a path exists, it has to go through one of the neighbours of t and one of the neighbours of s.

If for some i, all the n blocking nodes have 0 in dimension i, i.e., the ith column of the blocking nodes is 0, then we can easily route s to $00\cdots010\cdots0$, where 1 is in dimension i (note that $00\cdots010\cdots0$ is not a blocking node), by correcting bits in j, $j \neq i$ and $0 \leq j \leq n-1$ in any order, and then to 0.

If there is no such a 0 column, since the only case when each column has exactly one 1 is when the blocking nodes are 000...001, 000...010, ..., 010...000, and 100...000, we claim that when the condition as specified holds, then there exists a neighbour s' of s that is not one of the blocking nodes such that the Q_{n-1} containing t and s' contains at most $n-2$ original blocking nodes, that is, there exists a column i of the blocking nodes with at least two 1's such that when we set the bit of s in dimension i to 0 to obtain s', there are at most $n-2$ blocking nodes in the Q_{n-1} containing 0 and $s' = 111\cdot101\cdots1$ and s' is different from any of the blocking nodes. To see this, since we can permute the columns of the blocking nodes (and we can certainly order the blocking nodes), the blocking nodes are as shown in Fig. 5.1 where $k \geq 1$ and each of the first k columns has at least two 1's. We consider the following two cases. If $k = n$, then the only way that changing a 1 to 0 in $s = 1^n$ in any dimension results in a blocking node is when the blocking nodes are $\{b_1, b_2, \cdots, b_n\} = \{011...111, 101...111, \cdots, 111...110\}$. If $1 \leq k \leq n-1$, changing a 1 in any of the first k dimensions of 1^n to 0 will not result in a blocking node (see Fig. 5.1). Therefore, our claim is true. This claim combined with Lemma 5 implies that there exists a shortest path from s to t not intersecting any of the blocking nodes. □

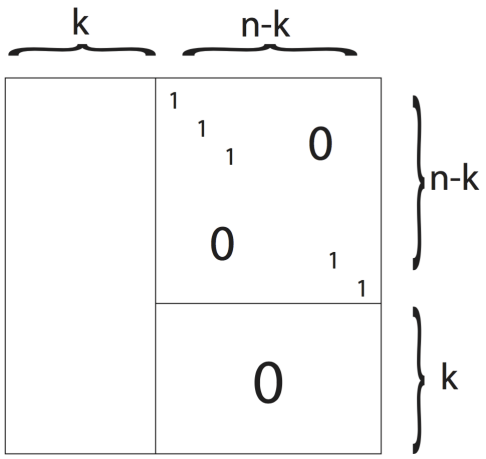

Figure 5.1: Blocking Nodes.

Intuitively, the smallest number of blocking nodes required such that both Conditions

(a) and (b) in Lemma 6 hold yet there does not exist a shortest path from 1^n to 0^n is $2(n-1)$. This is obtained by making $n-1$ neighbours of 1^n as blocking nodes and $n-1$ neighbours of the non-blocking node that is a neighbour of 1^n as blocking nodes. For example, if $n = 4$, these blocking nodes could be 1110, 1011, 1101, 0110, 0101, and 0011. Note that the last three blocking nodes are neighbours of 0111, the nonblocking neighbour of 1111. In fact, we can prove the following result easily:

Proposition 2. *In Q_n, if the number of blocking nodes is less than $2(n-1)$ and Conditions (a) and (b) from Lemma 6 hold, then there exists a shortest path from 1^n to 0^n.*

Proof. We apply induction on n.

For $n = 2$, if we have one blocking node, then we have a shortest path from 11 to 00.

Assume that the proposition holds for n. Then for Q_{n+1}, we have at most $2((n+1)-1)-1 = 2n-1$ blocking nodes. we consider the following three cases for the at most $2n-1$ blocking nodes.

Case 1. There is one column of 0's. Let us assume it's column 1 (dimension n). In this case, we can route 1^{n+1} to 0^{n+1} as follows without intersecting any blocking node: $1^{n+1} \stackrel{*}{\rightarrow} 10^n \rightarrow 0^{n+1}$. This is because 10^n is not a blocking node.

Case 2. There is (at least) one column, say column 1, with exactly one 1. Let us assume the blocking node with 1 in position 1 is u. If 10^n is not a blocking node, i.e., $u \neq 10^n$, then we can first route 1^{n+1} to 10^n without intersecting any of the blocking nodes easily (since u has at least one 1 in another position, say position 2, i.e., $u = 11\cdots$, a possible path is $1^{n+1} \rightarrow 1011\cdots1 \stackrel{*}{\rightarrow} 10^n \rightarrow 0^{n+1}$. Note that u is the only blocking node with position 1 being 1. Thus, node $1011\cdots1$ is non-blocking). If 10^n is a blocking node, i.e., $u = 10^n$, then one of the node 0^{n+1}'s other n neighbours has to be free. Let this neighbour be $v = 010^{n-1}$. Then the routing is $1^{n+1} \stackrel{*}{\rightarrow} 110^{n-1} \rightarrow 010^{n-1} \rightarrow 0^{n+1}$. Note that since there is only one blocking node with column 1 being 1. u is this node. Thus, node 110^{n-1} is not a blocking node.

Case 3. All columns have at least two 1's. Since Condition (a) holds, there must be one dimension such that if we change the 1 in that dimension to 0 in 1^{n+1}, it will not result in a blocking node. Thus we now have at most $(2n-1)-2 = 2n-3$ blocking nodes in a Q_n and by induction hypothesis, there exists a shortest path from 1^{n+1} to 0^{n+1}. □

Clearly, there are cases where such shortest paths exist from s to t in Q_n even when the number of blocking nodes is greater than $2n-3$. Thus the above result is simply a sufficient condition. It would be interesting to characterize the necessary and sufficient conditions for which such paths exist for any given set of blocking nodes.

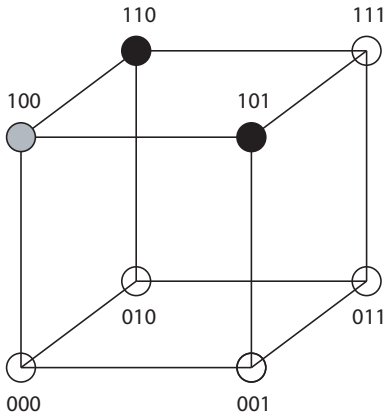

Figure 5.2: Dead End Node (Gray Node) and Blocking Nodes (Black Nodes) in Q_3.

5.3 The Routing Algorithm

In this section, we will present an algorithm which determines the existence of a shortest path in an n-dimensional hypercube from $s = 1^n$ to $t = 0^n$ with m blocking nodes where $1 \leq m \leq 2^n - (n + 1)$.

In a 2-cube, if one of the nodes 01 and 10 is not a blocking node, we know that there is a shortest path between 00 and 11. On the other hand, in a 3-cube, if nodes 110 and 101 are blocking nodes, then we know we cannot go from 111 to 100 (this is also implied by Lemma 6 in the 2-cube of nodes 100, 101, 110, and 111) and thus 100 is not reachable from 111. See Fig 5.2. Node 100 in this case becomes a de facto blocking nodes in the 3-cube being considered. This discussion inspires the notion of dead end nodes and the subsequent algorithm.

5.3.1 IDENTIFICATION Algorithm

We define a dead end node as the node which does not originally belong to the set of blocking nodes and is not reachable from node 1^n in an n-cube. Clearly, any shortest path between 0^n and 1^n, if it exists, does not go through any dead end node, that is, a dead end node acts just like a blocking node.

The IDENTIFICATION algorithm, explained further, identifies dead end nodes in the n-dimension hypercube with the set of blocking nodes. It is based on the fact that when a group of blocking nodes of the same Hamming weight, say h, exist such that they block a non-blocking node, then the blocked node, a dead end node, has a Hamming weight of $h - 1$. The correctness of the algorithm is implied by Lemma 6.

We now describe the algorithm. First, blocking nodes will be divided based on their

Hamming weights 1, 2, ..., $n-1$ (note that 1^n and 0^n cannot be blocking nodes). The algorithm performs an AND operation as follows:

For the blocking nodes with Hamming weight $n-1$, it performs a bit-wise AND operation for every possible pairs of nodes (there could be $O(\binom{h(n-1)}{2})$ such pairs, where $h(n-1)$ is the number of blocking nodes with Hamming weight of $n-1$). If the operation results in a node with Hamming weight $n-2$, the new node is a dead end node and is added to the set of blocking nodes with Hamming weight of $n-2$ as a new blocking node, If the result is not a node with Hamming weight of $n-2$ or if it is already a blocking node with Hamming weight $n-2$, we leave it and do nothing further. Essentially, this step finds all the nodes that cannot be reached from 1^n in one step (inside a 2-cube containing node 1^n). Then, the algorithm proceeds to the next step with the blocking nodes with Hamming weight $n-2$ and performs the same operations for every three of them (there could be $O(\binom{h(n-2)}{3})$ such triples). Similarly, this step finds all the nodes that cannot be reached in two steps from 1^n (inside a 3-cube containing the node 1^n). The process continues until it finishes performing for every group of $(n-1)$ blocking nodes among the blocking nodes with Hamming weight of 2.

To demonstrate this procedure, consider a Q_4 with the following set of blocking nodes: $\{1110, 1011, 1101, 0011\}$. We first group the blocking nodes into two groups with Hamming weights 3 and 2.

$$\begin{aligned} Hamming\ weight &= 3 : \{1110, 1011, 1101\} \\ Hamming\ weight &= 2 : \{0011\} \end{aligned}$$

We begin with blocking nodes with Hamming weight of $n-1=3$, which are $1110, 1011, 1101$, and we perform the following operations:

$1110 \wedge 1011 = 1010$

$1110 \wedge 1101 = 1100$

$1011 \wedge 1101 = 1001$

Since all the resulting nodes have Hamming weight 2, they are dead end nodes and will be added to the set of blocking nodes with Hamming weight of 2. Hence, the new set of blocking nodes with Hamming weight of 2 has been updated to: $\{0011, 1010, 1100, 1001\}$.

For the next set of operations, we have:

$0011 \wedge 1010 \wedge 1100 = 0000$

$0011 \wedge 1010 \wedge 1001 = 0000$

$0011 \wedge 1100 \wedge 1001 = 0000$

$1010 \wedge 1100 \wedge 1001 = 1000$

After this step, we get one dead end node 1000 which is added to the set of blocking nodes with Hamming weight 1 and the algorithm stops.

Now we are ready to present a necessary and sufficient condition to determine if there exists a shortest path from $s = 1^n$ to $t = 0^n$ in a hypercube of dimension n by utilizing the IDENTIFICATION algorithm. Obviously, the number of blocking nodes cannot exceed $2^n - (n + 1)$ because a shortest path from s to t in an n-cube consists of $n + 1$ nodes.

Theorem 5.3.1. *In Q_n with m blocking nodes, $1 \leq m \leq 2^n - (n + 1)$, there exists a shortest path from $s = 1^n$ to $t = 0^n$ that does not intersect any of the blocking nodes if and only if after performing the IDENTIFICATION algorithm, at least one of the 0^n neighbours is a non-blocking node (neither original blocking node nor dead end node).*

Proof. Necessary: If such a shortest path exists, it has to go through one of the 0^n neighbours, say u. Then u cannot be one of the blocking nodes or dead end nodes.
Sufficient: We show that if after running the algorithm, at least one of the neighbours of 0^n is a non-blocking node, then we can actually construct a shortest path from 0^n to 1^n.

Without loss of generality, assume that this non-blocking neighbour node is $j = 10^{n-1}$. We can go from 0^n to j in one step. Now we need to go to one of the neighbours of j whose Hamming weight is one more than j's. If all the neighbours of j with weight $H(j)+1 = 2$, namely, nodes $1100 \cdots 00$, $1010 \cdots 00$, ..., $1000 \cdots 01$, are blocking nodes, then j would have been generated as a blocking node during the procedure, since

$$1100 \cdots 00 \wedge \ 1010 \cdots 00 \wedge \ldots \wedge 1000 \cdots 01 = 1000 \cdots 00,$$

a contradiction. Therefore, at least one of j's neighbours with weight= $H(j)$+1 is not a blocking node. Now we add the edge from j to this non-blocking node to the shortest path and continue in the same fashion. □

As an example, consider a Q_3 with blocking nodes: $\{101, 011, 100, 010\}$. If we run the IDENTIFICATION algorithm, it will generate 001 as a dead end node. Therefore, there is no such node among 0^n neighbours which is non-blocking node, because 100 and 010 are blocking nodes.

Of course, if at any time during the algorithm, the number of blocking nodes (the original blocking nodes plus newly created dead end nodes) with Hamming weight h is equal to $\binom{n}{h}$, the total number of nodes in an n-cube with Hamming weight h, the algorithm should be terminated because we know that there does not exist a shortest path between 1^n and 0^n.

In [12], Chiu and Wu presented the notions of safe nodes and unsafe nodes that imply the likelihood for having a shortest path without going through blocking nodes. Their routing guarantees the existence of a shortest path between nodes A and B depending on whether A or B is safe. However, the step of identifying the status of each node can cause inefficiency during the procedure. Moreover, if the source and destination nodes are not safe, the algorithm is not able to find the shortest path while for our algorithm there is no constraint on the source and destination nodes to obtain the result. It is worth pointing out that using the safe node concept is effective when it is applied in low-dimensional hypercubes, while the efficiency of our approach does not rely on the dimension of the hypercube. It is also worth pointing out that assumptions made are different between our work and the others reviewed in this thesis where we know beforehand all the blocking nodes. Similarly, our algorithm always finds shortest paths if they exist.

Chapter 6

Conclusion

In this thesis we have studied the KCube interconnection network that combines the well-known Kautz graph and the hypercube. KCube is a novel architecture for connecting many processors in an interconnection network or communication network that was first proposed in 2010. We proved some interesting properties of the KCube and designed a communication algorithm as well. In addition, we investigated a well-known paradigm of the shortest path routing problem in the classic binary hypercube and achieved results pertaining to the faulty hypercube network.

In particular, we:

- Proposed a new methodology for constructing KCube with more flexibility in terms of connection between input and output nodes;
- Found the average number of steps required for routing between two nodes in a $KC(m, k)$;
- Showed the existence of a Hamiltonian cycle for the general $KC(m, k)$;
- Found the connectivity of KCube, which is equal to m when the KCube is treated as a directed graph;
- Designed an optimal broadcasting algorithm in the single-port model for $KC(m, k)$ regardless of what arrangements for input and output nodes is used. This algorithm meets the lower bound broadcasting time defined for KCube network.
- Studied a well-known paradigm of the shortest path routing problem. A few sufficient conditions for different cases depending on the number of faulty nodes were presented.

- Introduced an algorithm, which also implies a necessary and sufficient condition for the shortest path to exist with presence of any number of faulty nodes.

Since the KCube is a newly proposed topology, further research is required to unravel its potential properties and algorithms. One of our future aims is to study the routing problem and to find the diameter of the KCube. Also, it would be interesting to design application algorithms such as sorting for the KCube.

As for the blocking node problem in the hypercube, we have only concentrated on finding/characterizing necessary and sufficient conditions for the shortest paths to exist so far. Our next step would be to evaluate the time complexity of the algorithm requires. Clearly, our algorithm could be inefficient in certain cases. In an initial evaluation of our method, we find that of how blocking nodes are distributed in the hypercube affects the time to obtain the result. For example, the more blocking nodes are close to either 0^n or 1^n, the less time is needed for the IDENTIFICATION algorithm to be completed. This is due to the steps of any Hamming weight that the algorithm will skip if there is no any blocking node of that specific Hamming weight and before that.

We are working toward analyzing its average time and cases when it performs efficiently.

Bibliography

[1] S.G. Akl. *Parallel Computation: Models and Methods*. Prentice-Hall, Inc., 1997.

[2] J. Araujo, J-C. Bermond and G. Ducoffe. Eulerian and Hamiltonian Dicycles in Directed Hypergraphs. *Discrete Mathematics, Algorithms and Applications*, 6(01):pages 1450012, 2014.

[3] J-C. Bermond and S. Perennes. Efficient Broadcasting Protocols on the De Bruijn and Similar Networks. In *Proc. 2nd Colloquium on Structural Information and Communication Complexity*. Citeseer, 1995.

[4] J.A. Bondy and U.S.R. Murty. USR Murty Graph Theory with Applications, 1976.

[5] W.G. Bridges and S. Toueg. On the Impossibility of Directed Moore Graphs. *Journal of Combinatorial theory, series B*, 29(3):pages 339–341, 1980.

[6] D.J. Bruijn. A Combinatorial Problem. *Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen. Series A*, 49(7):pages 758, 1946.

[7] J.M. Brunat. Explicit Cayley Vovers of Kautz Digraphs. *the electronic journal of combinatorics*, 18(1):pages 105, 2011.

[8] C.C. Chen and N.F. Quimpo. On Strongly Hamiltonian Abelian Group Graphs. In *Combinatorial mathematics VIII*, pages 23–34. Springer, 1981.

[9] D. Guo, H. Chen, Y. He, H. Jin, C. Chen, H. Chen Z. Shu, and G. Huang. KCube: A Novel Architecture for Interconnection Networks. *Information Processing Letters*, 110(18):pages 821–825, 2010.

[10] M.S. Chen and K.G. Shin. Adaptive Fault-Tolerant Routing in Hypercube Multicomputers. *IEEE Transactions on Computers*, 39(12):pages 1406–1416, 1990.

[11] W.K. Chiang and R.J. Chen. Distributed Fault-Tolerant Routing in Kautz Networks. In *Distributed Computing Systems, 1992., Proceedings of the Third Workshop on Future Trends of*, pages 297–303. IEEE, 1992.

[12] G.M. Chiu and S.P. Wu. A Fault-Tolerant Routing Strategy in Hypercube Multicomputers. *IEEE Transactions on Computers*, 45(2):pages 143–155, 1996.

[13] M. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 100(9):pages 948–960, 1972.

[14] Z. Ge and S. L. Hakimi. Disjoint Rooted Spanning Trees with Small Depths in De Bruijn and Kautz Graphs. *SIAM Journal on Computing*, 26(1):pages 79–92, 1997.

[15] E. Ghosh. Hamiltonicity and Longest Path Problem on Special Classes of Graphs. *Thesis Department of Computer Science and Engineering Indian Institute of Technology*, 2011.

[16] Q.P. Gu and S. Peng. Optimal Algorithms for Node-to-Node Fault Tolerant Routing in Hypercubes. *The Computer Journal*, 39(7):pages 626–629, 1996.

[17] K. Day, S. Harous and A-E. Al-Ayyoub. A Fault Tolerant Routing Scheme for Hypercubes. *Telecommunication Systems*, 13(1):pages 29–44, 2000.

[18] J-C. Bermond, N. Homobono and C. Peyrat. Connectivity of Kautz Networks. *Discrete Mathematics*, 114(1):pages 51–62, 1993.

[19] L.H Hsu and C.K. Lin. *Graph Theory and Interconnection Networks*. CRC press, 2008.

[20] S.L. Johnsson and C.T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, 38(9):pages 1249–1268, 1989.

[21] K. Kaneko and H. Ito. Fault-Tolerant Routing Algorithms for Hypercube Interconnection Networks. *IEICE TRANSACTIONS on Information and Systems*, 84(1):pages 121–128, 2001.

[22] W.H. Kautz. Bounds on Directed (d, k) Graphs. Theory of Cellular Logic Networks and Machines. *AFCRL-68-0668 Final report*, pages 20–28, 1968.

[23] C. Lavault. Interconnection Networks: Graph and Group Theoretic Modelling. In *12th International Conference on Control Systems and Computer Science (CSCS12)*,

volume 2, page 207. Romanian Society of Control Engineering and Technical Informatics and Faculty of Control and Computers, 1999.

[24] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays· Trees· Hypercubes*. Elsevier, 2014.

[25] D. Li, X. Lu and J. Su. Graph-Theoretic Analysis of Kautz Topology and DHT Schemes. In *Network and Parallel Computing*, pages 308–315. Springer, 2004.

[26] M. Dietzfelbinger, S. Madhavapeddy and I.H. Sudborough. Three Disjoint Path Paradigms in Star Networks. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 400–406. IEEE, 1991.

[27] S. Gao, B. Novick and K. Qiu. From Hall's Matching Theorem to Optimal Routing on Hypercubes. *Journal of Combinatorial Theory, Series B*, 74(2):pages 291–301, 1998.

[28] M.C. Heydemann, J. Opatrny and D. Sotteau. Broadcasting and Spanning Trees in De Bruijn and Kautz Networks. *Discrete applied mathematics*, 37:pages 297–317, 1992.

[29] E Cheng, K. Qiu and Z.Z. Shen. On Disjoint Shortest Paths Routing in Interconnection Networks: A Case Study in the Star Graph. *Congressus Numerantium*, pages 157–180, 2013.

[30] K. Qiu and B. Novick. Disjoint Paths in Hypercubes. *Congressus Numerantium*, pages 105–112, 1996.

[31] Y. Saad and M.H. Schultz. Topological Properties of Hypercube. *IEEE Transactions on Computers*, 37(7):pages 867–872, 1988.

[32] Y. Saad and M.H. Schultz. Data Communication in Hypercubes. *Journal of parallel and distributed computing*, 6(1):pages 115–135, 1989.

[33] P. Salinger and P. Tvrdik. All-to-All Scatter in Kautz Networks. In *Euro-Par98 Parallel Processing*, pages 1057–1061. Springer, 1998.

[34] M.R. Samatham and D.K. Pradhan. The De Bruijn Multiprocessor Network: a Versatile Parallel Processing and Sorting Network for VLSI. *IEEE Transactions on Computers*, 38(4):pages 567–581, 1989.

[35] G. Smit and P. Havinga. Multicast and Broadcast in the Rattlesnake ATM Switch. In *MMNET*, pages 218–226. Citeseer, 1995.

[36] M. Imase, T. Soneoka and K. Okada. Fault-Tolerant Processor Interconnection Networks. *Systems and Computers in Japan*, 17(8):pages 21–30, 1986.

[37] L. Zhao. Properties and Algorithms of the KCube Graphs. *Thesis Department of Computer Science Brock University*, 2014.

[38] S. Zhou and H. Xu. A Unified Formulation of Kautz Network and Generalized Hypercube. *Computers & Mathematics with Applications*, 49(9):pages 1403–1411, 2005.