Analysis of Benes Network and implementation in C++

Submitted by

Birju Shaw
20CS4125
M.Tech(2020-2022)

Under the guidance of

DR.Jaydeep Howlader
Assistant Professor
Dept. of Computer Science and Engineering
NIT Durgapur, INDIA-713209

Thesis is submitted for the degree of
Master of Technology
in Computer Science and Engineering

MAY 2022

Certificate of Approval

The forgoing thesis is hereby approved as a creditable study of Technological subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite with degree for which it has been submitted. It is to be understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for the purpose for which it has been submitted.

Board of Thesis Examiners

-
-
-
-

Declaration

I Declare that

- The work contained in this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any University or elsewhere for the award of a diploma or degree. Except where explicitly stated otherwise by reference or acknowledgement, the work presented is entirely my own.
- Whenever I have used materials (data, theory, analysis, figures, and text) from other sources, I have given credit to them by citing them in the text of the thesis and giving their details in the references.
- I have confirmed to the norms and guidelines given in the Ethical code of Conduct of the Institute.

I, Birju Shaw, hereby declare that this thesis entitled " Analysis of Benes Network and implementation in C++" is submitted to the Department of Computer Science & Engineering, National Institute of Technology, Durgapur, West Bengal, India for acceptance to award the degree of master of technology in Computer Science & Engineering is prepared by me and the same has not been is not being submitted to any other institution.

(Signature of Student)
Birju Shaw
20CS4125

Certificate

This is to certify that the Dissertation Report entitled, "Analysis of Benes Network and implementation in C++" submitted by Mr. Birju Shaw (20CS4125) to Department of Computer Science & Engineering, National Institute of Technology, Durgapur, India, is a record of bonafide Project work carried out by her under my supervision and guidance and is worthy of consideration for the award of the degree of Master of Technology in Computer Science and Engineering of the Institute.

(Signature of Project Guide)
Dr. Jaydeep Howlader
Assistant Professor
Department of Computer Science and Engineering

(Signature of HOD)
Dr. Subrata Nandi
Head of the Department
Department of Computer Science and Engineering

Acknowledgements

I hereby wish to express my sincere gratitude and respect to Assistant Professor Dr. Jaydeep Howlader, Dept. of Computer Science and Engineering, NIT, Durgapur, under whom I had proud privilege to work. His valuable guidance and encouragement have really led to the path of completion of this project. Any amount of thanks would not be enough for the valuable guidance of my supervisor. It was thus that working under his expertise and receiving a part of his tremendous knowledge became rewarding experience.

I would also like to thank all the faculty members of Computer Science and Engineering Department (specially Dr. Sajal Mukhopadhyay and Mr. Nanda Dulal Jana) for their devoted help. I also cordially thank all PhD students of Computer Science and Engineering Department (Mr. Arjun Kumar, Mr. Soumya Sarkar, Mr. Dhaneshwar Mardi).

Finally, I would like to pen down my gratitude towards my parent and my friend Sumit Kumar Gupta, Ashish Barui for their continuous support and encouragement. It would have not been possible to complete my work without their support.

(Signature of Student)
Birju Shaw
20CS4125

Abstract

We used C++ programming to create an 8 x 8 input-output Benes Network. We used 8 inputs [1,2,3,4,5,6,7,8] and a configuration matrix 4 x 5 (which has a total permutation set of 2^{20} (0/1) bits). Then we calculated all the permutations of the eight inputs and discovered that the permutation is repeating. We calculated the number of times each permutation occurs and plotted the distribution graph. In the following step, we used a heuristic to reduce redundancy (the number of times the same permutation occurs).

Similarly, I have implemented Benes Network for $16 * 16$ input-output.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Interconnecting Network

In a parallel machine, an interconnection network sends data from any source node to any desired destination node. With as little latency as feasible, this task should be accomplished. It should be possible to do a large number of such transfers at the same time. In addition, it should be affordable in comparison to the rest of the equipment.[16] Multistage interconnection networks (MINs) connect input and output devices via a series of switch stages, with each switch acting as a crossbar network.

1.1.1 Organizational Structure

The network is made up of links and switches that help to send data from the source node to the destination node. The topology, routing algorithm, switching strategy, and flow control mechanism of a network define it.

Interconnection networks are made up of the three basic components listed below.

- Links - A link is a cable made up of one or more optical fibres or electrical wires that connects to a switch or network interface port at each end. The original digital information stream is obtained by transmitting an analogue signal from one end and receiving it from the other.

- Switches - A switch is made up of input and output ports, an internal "cross-bar" that connects all input to all output, internal buffering, and control logic that affects the input-output connection at any given time. The number of input ports is usually the same as the number of output ports.

- Network Interfaces - The network interface differs from switch nodes in that it can be connected via specific links. The packets are formatted and routing and control information is constructed by the network interface. In comparison to a switch, it may feature input and output buffering. End-to-end error checking and flow control are possible. As a result, the processing complexity, storage capacity, and number of ports all influence the price.

1.1.2 Classification of Interconnecting network

The general categorisation of interconnection networks is depicted in the graphic below. Arranged, static networks, also known as assigned networks, are usually used to connect weakly connected processors to construct parallel processing machines, whereas an irregular static network is any broad packet switching network. By linking processing elements with buses, shared path static networks are created. Because the delay through the network is dependent on the distance between the communication nodes, conventional static networks have been confined to packet switched interlock of weakly linked processors in general.Furthermore, the routing algorithm's processing postponement may render the use of small packets wasteful. It was challenging to expand to vast networks while keeping the regularity of the structure with better, regular static structures. The adoption of a regular mesh topology in the Manhattan Street Network, which has been proposed as a MAN, is a key exclusion.

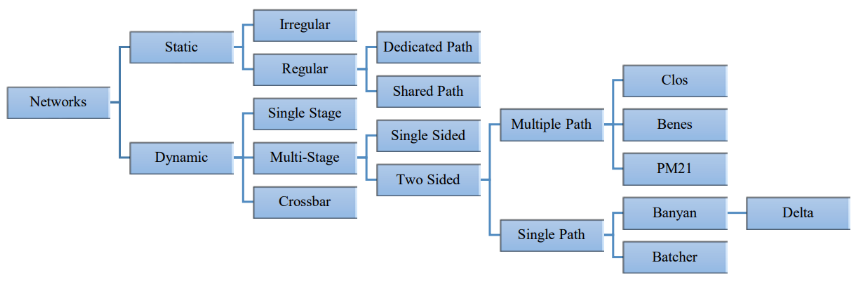

Figure 1.1: classification of interconnecting networks

Dynamic interconnection networks are so named because the network customers are connected via an array of simple switching devices. A centralised processor or a distributed algorithm can quickly change the arrangement of links between clients.

As a result, multistage interconnection networks are one of the fastest communication routes available, and they are ideal for developing parallel computers. In fact, numerous switching layers create the connection between the source and destination nodes in these networks. Different types of networks result from the way the layers are connected to one another. The multistage interconnection networks are divided into two groups: static and dynamic in one grouping. The dynamic ones rely on switching that occurs in real time to match the route between source and destination nodes. Dynamic multi-stage interconnection separated networks are divided into four groups: Non-blocking, rearrangeable non-blocking, and strictly non-blocking networks are all available.

1.2 Purpose of Interconnecting Network

Interconnection Networks are used for two primary purposes:

- **Connect the processors to the shared memory system.**
- **Connect the processors together.**

Connect the processors together. The type of interconnection network utilised in a distributed system has a significant impact on the algorithms that will be employed to execute standard and general activities in the system. The fundamental reason for this is that the data routing is determined by the interconnection pattern. This is frequently the determining element in the complexity of several algorithms in this paradigm. There are four important criteria that characterise an interconnection network from this perspective:

1. The diameter
2. The bisection width
3. The edges per node
4. The edge length

Each of these variables requires more explanation.

1.2.1 DIAMETER:

It is defined as the greatest distance between two network switch nodes. It is advantageous if a network has a small diameter since the worst-case time complexity of data routing from one node to another will be minimal (it is a function of the diameter). The overhead of data routing is automatically increased when the diameter is big.

As a result, the width frequently lowers the complexity of parallel algorithms requiring communication between two arbitrary pairs of nodes.

1.2.2 BISECTION WIDTH:

It's the smallest number of edges between switch nodes that must be eliminated to divide the network into two nearly equal halves.

It is a measurement of a system's ability to manage massive amounts of data transfer (or bottleneck). A system with a broad bisection width is beneficial because it allows enormous volumes of data to be distributed among a larger number of connections.

The complexity of algorithms that need large quantities of data movement is limited by the size of the data set divided by the bisection width.

1.2.3 EDGES PER NODE:

The number of edges per node should be a constant that is independent of the network, as this allows for greater scalability. Furthermore, routing methods can be more generalised and not limited by the number of nodes in the network.

1.2.4 EDGE LENGTH:

The maximum edge length should be consistent regardless of the network for scalability reasons. This makes it possible to plan and lay out the network in an organised manner.

1.3 Other connectivity network characteristics:

1.3.1 Rearrangeable:

A network can be rearranged if and only if routes for any pre-assigned sourcedestination pairs can be established, as long as no two sources have the same destination.

1.3.2 Non-blocking:

In the presence of other known source-destination pairings, a network is non-blocking if and only if it is possible to route from any source to any destination, provided no two sources have the same destination.

Thus, all non-blocking networks are rearrange able, but not the other way round.

Multistage connectivity networks are required:

A single stage interconnect network's limiting case is the crossbar. It has several advantages (Diameter = 1, non-blocking), but it is highly expensive ($cost = O(n2)$).

The slides include a comprehensive mathematical example that demonstrates how multistage interconnection networks can reduce costs while increasing diameter. There could be other trade-offs.

The remainder of this paper examines several interconnection networks and evaluates them using the criteria listed above (namely diameter, bisection width, edge length, edges per node, rearrange-ability, non-blocking character).

We discuss three such networks:

- CLOS
- BUTTERFLY
- Benes Network

We describe an application of an interconnect network for sorting in the final part. The Batcher's sorting network is a great example of how we might reduce sorting time complexity by taking advantage of the problem's inherent parallelism (namely sorting).

Chapter 2

Clos Network

2.1 clos network: an introduction

Clos is a switching network with multiple stages. A three-stage clos network is depicted in Figure 1. The benefit of such a network is that it may connect a large number of input and output ports with only a few tiny switches. By configuring the switches at all stages, a bipartite match between the ports can be achieved. In figure 2.1 ,The number of sources that flow into each of the m entrance stage crossbar switches is represented by n. Each ingress stage switch and each intermediate stage switch have exactly one connection, as can be observed. Each egress stage switch is connected to each middle stage switch exactly once.

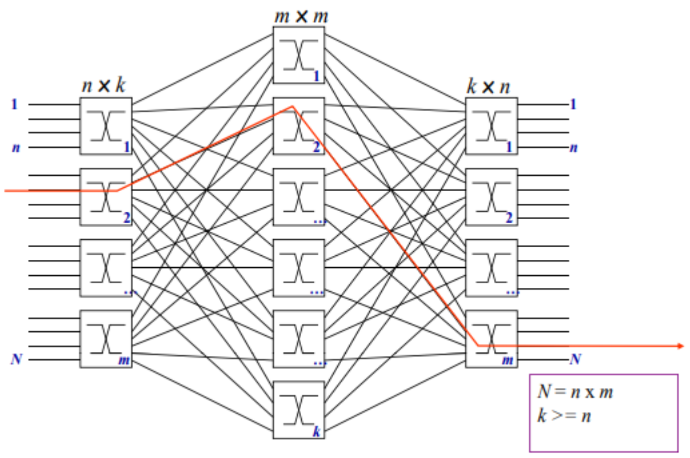

Figure 2.1: 3 stage clos network

The clos network [21] can be non-blocking like a crossbar switch when $k \geq n$ is used. That is, we can identify an arrangement of paths for linking the inputs and outputs through the middle-stage switches for each input-output match. As long as the number of middle stage switches is large enough, the following theorem establishes that adding a new connection does not necessitate reordering existing connections.

2.1.1 Clos Theorem[3]

If $k \geq 2n - 1$, then a new connection can always be added without rearrangement.

Proof: Consider adding the nth connection between 1st stage Ia and 3rd stage Ob as shown in figure2.2 . We need to ensure that there is always some center-stage M available. If $k > (n-1) + (n-1)$, then there is always an M available. i.e., we need $k \geq 2n - 1$.

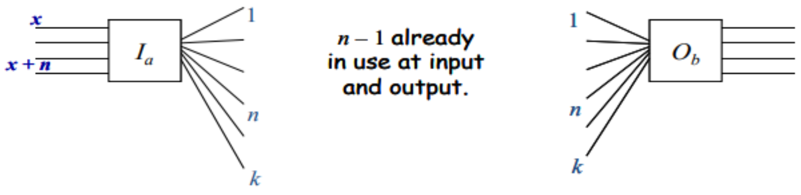

Figure 2.2: Adding the nth connection

2.1.2 Topology

A Clos network is a type of multistage circuit-switching network used in telecommunications that represents a theoretical idealisation of practical multistage switching systems. Edson Erwin[9] created it in 1938, and Charles Clos formalised it in 1952.

A Clos network decreases the number of crosspoints needed to make a large crossbar switch by adding stages. The three integers n, m, and r specify the number of sources that flow into each of r ingress stage crossbar switches; each ingress stage crossbar switch has m outlets; and there are m middle stage crossbar switches in a Clos network design (shown below).

Circuit switching establishes a dedicated communications line for the duration of a connection between two destinations. If the dedicated connections are underutilised, the overall bandwidth available is reduced, but the connection and bandwidth are more predictable, and control overhead is introduced only when the connections are established, rather than with each packet handled, as in modern packet-switched networks.

The number of crosspoints was a decent approximation of the total cost of the switching mechanism when the Clos network was originally established. While this was critical for electromechanical crossbars, it became less so with the introduction of VLSI, which allowed interconnects to be implemented either directly in silicon or inside a small cluster of boards. Clos networks achieved prominence with the emergence of complex data centres with massive connection systems based on optical fibre lines. [5] The Bene network, a subtype of the Clos network, has recently found use in machine learning.

The ingress stage, the middle stage, and the egress stage are the three stages of a Clos network. Each stage is made up of several crossbar switches (see diagram below), which are commonly referred to as crossbars. Between stages, the network uses an r-way perfect shuffle. Each call entering an ingress crossbar switch can be directed to the appropriate egress crossbar switch via any of the available middle stage crossbar switches. If both the link linking the ingress switch to the middle stage switch and the link connecting the middle stage switch to the egress switch are free, a middle stage crossbar is available for a specific new call.

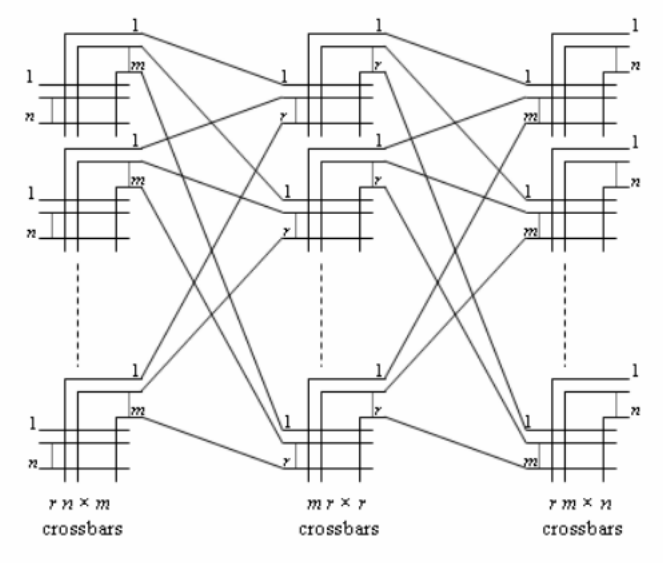

Figure 2.3: clos network topology

The three integers n, m, and r define closed networks. The number of sources that flow into each of the r ingress stage crossbar switches is represented by n. There are m outlets on each entrance stage crossbar switch, and there are m intermediate stage crossbar switches. Each ingress stage switch and each intermediate stage switch have exactly one connection. R egress stage switches are available, each with m inputs and n outputs. Each egress stage switch is connected to each middle stage switch exactly once.The ingress stage therefore includes r switches, each with n inputs and m outputs. Each of the m switches in the intermediate stage has r inputs and r outputs. There are r switches in the egress stage, each with m inputs and n outputs.

2.1.3 uses

Clos networks have returned in the design of high-performance switches in data centre fabrics because of the efficiency improvements they offer, despite the fact that electro mechanical switching has given way to newer switching technologies. A Clos network provides non-blocking performance in an interconnected Ethernet switch fabric without the necessity for n-squared ports in a modern setting. [15]

Clos network efficiency is related to the array size, with bigger networks with thousands or tens of thousands of ports reaping the highest benefits.

2.1.4 Characteristics of blocking

The blocking characteristics of the Clos network are defined by the relative values of m and n.

2.1.5 Strict-sense nonblocking Clos networks ($m \geq 2n - 1$): the original 1953 Clos result

The Clos network is strict-sense nonblocking if $mgeq2n1$, which means that an unused input on an ingress switch can always be connected to an unused output on an egress switch without rearranging previous calls. This is the result that Clos' classic 1953 paper was based on. Assume that an ingress switch's input has a free terminal that must be connected to a free terminal on a specific egress switch. In the worst-case scenario, there are n1 other calls active on the ingress switch and n1 other calls active on the egress switch. Assume that each of these calls goes through a different middle-stage switch in the worst-case scenario. As a result, 2n - 2 of the middle stage switches are unable to carry the new call in the worst-case scenario. As a result, another middle stage switch is required to assure strict-sense nonblocking operation, for a total of 2n-1.

2.1.6 Rearrangeably nonblocking Clos networks ($m \geq n$)

The Clos network is rearrangeably nonblocking if $m \geq n$, which means that an unused input on an ingress switch can always be connected to an unused output on an egress switch, although existing calls may need to be rearranged by assigning them to other Clos network centre stage switches.[5] The Clos network is rearrangeably nonblocking if $m \geq n$, which means that an unused input on an ingress switch can always be connected to an unused output on an egress switch, although existing calls may need to be rearranged by assigning them to other Clos network centre stage switches.

The proof uses Hall's marriage theorem[14] which is given this name because it is often explained as follows. Suppose there are r boys and r girls. The theorem states that if every subset of k boys (for each k such that $(0 \leq k \leq r)$ between them know k or more girls, then each boy can be paired off with a girl that he knows. It is obvious that this is a necessary condition for pairing to take place; what is surprising is that it is sufficient.

In the context of a Clos network, each boy represents an ingress switch, and each girl represents an egress switch. A boy is said to know a girl if the corresponding ingress and egress switches carry the same call. Each set of k boys must know at least k girls because k ingress switches are carrying k×n calls and these cannot be carried by less than k egress switches. Hence each ingress switch can be paired off with an egress switch that carries the same call, via a one-to-one mapping. These r calls can be carried by one middle-stage switch. If this middle-stage switch is now removed from the Clos network, m is reduced by 1, and we are left with a smaller Clos network. The process then repeats itself until m = 1, and every call is assigned to a middle-stage switch.

2.1.7 Clos networks with more than three stages

Clos networks may also be generalised to any odd number of stages. By replacing each centre stage crossbar switch with a 3-stage Clos network, Clos networks of five stages may be constructed. By applying the same process repeatedly, 7, 9, 11,... stages are possible.

Beneš network $(m = n = 2)$

A rearrange-ably non blocking network of this type with$m = n = 2$is generally called a Beneš network, even though it was discussed and analyzed by others[who?] before Václav E. Beneš. The number of inputs and outputs is N = r×n = 2r. Such networks have $2log2N - 1$ stages, each containing N/2 2×2 crossbar switches, and use a total of$Nlog2N$ - N/2 2×2 crossbar switches. For example, an 8×8 Beneš network $(withN = 8)$ is shown below; it has $2log28 - 1 = 5$ stages, each containing (N/2)= 4 2×2 crossbar switches, and it uses a total of $Nlog2N$ - N/2 = 20 2×2 crossbar switches. The central three stages consist of two smaller 4×4 Beneš networks, while in the center stage, each 2×2 crossbar switch may itself be regarded as a 2×2 Beneš network. This example therefore highlights the recursive construction of this type of network.

Chapter 3

Butterfly Network

3.1 Introduction to Butterfly network

A butterfly network is a method of connecting several computers to form a high-speed network. A multiprocessor system can employ this type of multistage interconnection network topology to connect different nodes. Unlike other network systems, such as local area networks (LANs) or the internet, the interconnect network for a shared memory multiprocessor system must have low latency and high bandwidth for three reasons:[17]

- Because most messages simply coherence protocol queries and responses without data, messages are brief.
- Because each read-miss or write-miss creates messages to every node in the system to ensure coherence, messages are generated regularly. When the requested data is not in the processor's cache, read/write misses occur, and the data must be acquired from memory or from another processor's cache.
- Because messages are created frequently, it is difficult for processors to mask communication delays.

3.2 Components

The major components of an interconnect network are:

- One or more processors, as well as their caches, memory, and communication assistance, make up processing nodes.
- Switching nodes (Routers) are communication aids that connect multiple processor nodes in a system. Higher level switching nodes connect to lower level switching nodes in multistage topologies, as seen in figure 3.1, where switching nodes in rank 0 connect

directly to processor nodes and switching nodes in rank 1 connect to switching nodes in rank 0.

- Physical cables that connect two switching nodes are called links. They can be either one-way or bi-directional.

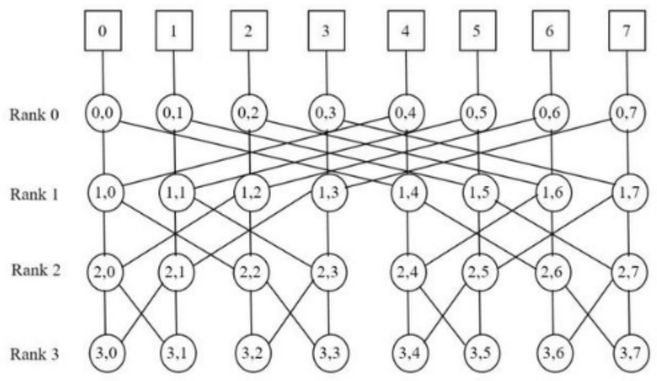

Figure 3.1: Butterfly network for 8 processors

These multistage networks are less expensive than a crossbar but have less congestion than a bus. In a butterfly network, the ratio of switching nodes to processor nodes is more than one. An indirect topology is one in which the ratio of switching nodes to processor nodes is greater than one.[11]

The network derives its name from connections between nodes in two adjacent ranks (as shown in figure 3.1), which resembles a butterfly. Merging top and bottom ranks into a single rank, creates a Wrapped Butterfly Network.[11] In figure 3.1), if rank 3 nodes are connected back to respective rank 0 nodes, then it becomes a wrapped butterfly network.

BBN Butterfly, a massive parallel computer built by Bolt, Beranek and Newman in the 1980s, used a butterfly interconnect network.[19] Later in 1990, Cray Research's machine Cray C90, used a butterfly network to communicate between its 16 processors and 1024 memory banks.[1]

3.3 Butterfly network building

For a butterfly network with p processor nodes, there need to be p($log2$ p + 1) switching nodes. Figure 3.1 shows a network with 8 processor nodes, which implies 32 switching nodes. It represents each node as N(rank, column number). For example, the node at column 6 in rank 1 is represented as (1,6) and node at column 2 in rank 0 is represented as (0,2).[11]

For any 'i' greater than zero, a switching node N(i,j) gets connected to N(i-1, j) and N(i-1, m), where, m is inverted bit on ith location of j. For example, consider the node N(1,6): i equals 1 and j equals 6, therefore m is obtained by inverting the ith bit of 6.

Variable	Binary Representation	Decimal Representation
j	110	6
m	010	6

As a result, the nodes connected to N(1,6) are :

N(i,j)	N(i-1,j)	N(i-1,m)
(1,6)	(0,6)	(0,2)

Thus, N(0,6), N(1,6), N(0,2), N(1,2) form a butterfly pattern. Several butterfly patterns exist in the figure and therefore, this network is called a Butterfly Network.

3.4 Butterfly Network Routing

In a wrapped butterfly network (which means rank 0 gets merged with rank 3), a message is sent from processor 5 to processor 2.[11] In figure 3.2, this is shown by replicating the processor nodes below rank 3. The packet transmitted over the link follows the form:

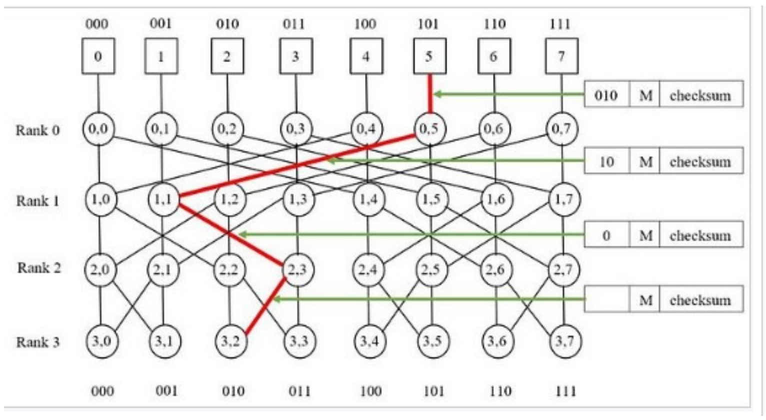

Figure 3.2: Butterfly Network Routing

HEADER	PLAYLOAD	TARILOR

The header contains the destination of the message, which is processor 2 (010 in binary). The payload is the message, M and trailer contains checksum. Therefore, the actual message transmitted from processor 5 is: Upon reaching a switching node, one of the two output

001	m	checksum

links is selected based on the most significant bit of the destination address. If that bit is zero, the left link is selected. If that bit is one, the right link is selected. Subsequently, this bit is removed from the destination address in the packet transmitted through the selected link. This is shown in figure 3.2.

- The above packet reaches N(0,5). From the header of the packet it removes the leftmost bit to decide the direction. Since it is a zero, left link of N(0,5) (which connects to N(1,1)) gets selected. The new header is '10'.
- The new packet reaches N(1,1). From the header of the packet it removes the leftmost bit to decide the direction. Since it is a one, right link of N(1,1) (which connects to N(2,3)) gets selected. The new header is '0'.
- The new packet reaches N(2,3). From the header of the packet it removes the leftmost bit to decide the direction. Since it is a zero, left link of N(2,3) (which connects to N(3,2)) gets selected. The header field is empty.
- Processor 2 receives the packet, which now contains only the payload 'M' and the checksum.

3.5 Butterfly network parameters

Several parameters help evaluate a network topology. The prominent ones relevant in designing large-scale multi-processor systems are summarized below and an explanation of how they are calculated for a butterfly network with 8 processor nodes as shown in figure 3.1 is provided.

- Bisection Bandwidth: The maximum bandwidth required to maintain network connection between all nodes. This is the smallest number of links that must be cut in order to divide the system into two equal parts. The 8-node butterfly network, for example, can be broken in half by severing four crisscrossing links in the middle. As a result, the system's bisection bandwidth is 4. It's a rough estimate of the bandwidth bottleneck that slows down total communication.

- Diameter: The maximum possible delay (between two nodes) in the system. It can be calculated in network hops, or the number of links a message must traverse to reach the destination node. N(0,0) and N(3,7) appear to be the farthest distant in the 8-node butterfly network, however closer investigation reveals that due to the network's symmetric architecture, travelling from any rank 0 node to any rank 3 node takes only 3 hops. As a result, the system's diameter is 3.
- Links: The total number of links required to build the full network. This is a measure of the overall cost and difficulty of the project. Figure 1 shows an example network that requires 48 links (16 links each between rank 0 and 1, rank 1 and 2, rank 2 and 3).
- Degree: The complexity of each router in the network. This is equal to the number of in/out links connected to each switching node. The butterfly network switching nodes have 2 input links and 2 output links, hence it is a 4-degree network.

3.6 Comparison with other network topologies

This section compares the butterfly network with linear array, ring, 2-D mesh and hypercube networks.[13] Note that linear array can be considered as a 1-D mesh topology. Relevant parameters are compiled in the table[8] ('p' represents the number of processor nodes).

[Network parameter]

topology	Diameter	Bisection width	links	Degree
Linear	$2p-1$	1	$p-1$	2
Ring	$p/2$	2	p	2
2-D mesh	$2(\sqrt{p}-1)$	$\sqrt{p}$	$2\sqrt{p}(\sqrt{p}-1$	4
Hypercube	$log2(p)$	$p/2$	$log2(p)*(p/2)$	log2(p)
Butterfly	$log2(p)$	2^h	$log2(p)*2p$	2

3.6.1 Advantages

- Butterfly networks have a smaller diameter than linear array, ring, and 2-D mesh topologies. This means that a message delivered from one processor will arrive at its destination in fewer network hops in a butterfly network.
- Other topologies offer lower bisection bandwidth than butterfly networks. This means that in order to block global communication in a butterfly network, a greater number of links must be disrupted.
- It has a bigger computer range.

3.6.2 Disadvantages

- Due to the larger number of links required to maintain the network, butterfly networks are more difficult and expensive than other topologies.
- Finally, no single network structure is optimal in all cases. The amount of processor nodes in the system, bandwidth-latency requirements, cost, and scalability all influence the decision.
- The implementation is the distinction between hypercube and butterfly.

Chapter 4

Benes Network

4.1 Introduction

Multi-stage interconnection networks are as one of the fastest communication channels that are using nowadays and they are so useful in designing the parallel computers. .indeed in these networks the connection between the source and destination nodes is formed of multiple switching layer [5]. the way the layers connect to each other provide different kinds of networks. In one grouping the multi-stage interconnection networks device to two group of static and dynamic group. The dynamic ones are based on switching that are applied in run-time to make the route between the source and destination nodes[21][7]. Dynamic multi-stage interconnection networks device into 4 groups: Blocking Network, non-blocking, rearrange able non-blocking and strictly non-blocking networks[20] . Benes network is a kind of dynamic non-

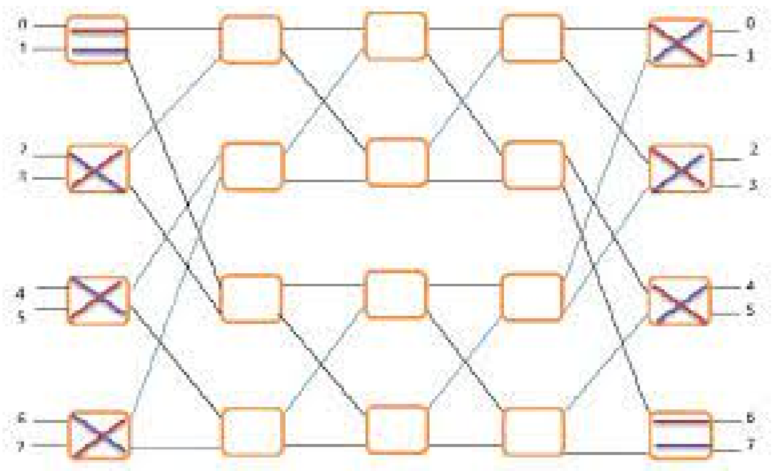

Figure 4.1: Benes network 8*8 input-output

blocking interconnection network after resorting. In this network we won't face the obstruction or if we face it we can close the switches to resolve this problem[18].

A Benes network with N input and N output is shown Benes (N×N) which is made of a 2×2 switch. This network has $2logN-1$ switch layer and each layer has N/2 switch. Figure 4.1 illustrates a Benes network (8×8)

In Benes Network the most important issue is quick and safe routing. Determination of the state of switches to transform the data from input to output is the goal of routing in this network[9][6]. Various routing algorithm for Benes network are offered like looping algorithm that introduced by tsao-wu and opferman in 1971, that starts from outer layer and moves to inner layer determines the way from input to output, however it might needs to return and change switches[14]. Fast method is based on analyze of permutation in every step and determine the state of switches[12]. Hassan-José method is based on bits and determines the state of every switches by use of coloring graph[3][10].

In 2009 chakrabarty and his colleagues introduced the matrix-based algorithm[4] that for each layer, there is one matrix and with simple algorithms determine the state of LogN-1 switches of the first layer and the other layers control with self-routing algorithm, but the problem of this algorithm is the back warding that make it less effective[2][8].

4.1.1 Tabular information of benes and butterfly network

In the below table we can see the all the details of benes and butterfly network.

Network	Diameter	switch size	no. of switches	congestion
benes	$2logN+1$	$2*2$	$2NlogN$	1
butterfly	$logN+2$	$2*2$	$N(logN+1)$	$\sqrt{N} or \sqrt{(N/2)}$

4.1.2 pseudo code design for benes network

First of all, we have to decide about the inputs to the program.

Algorithm 1 Pseudo code implementation of benes Network

```
Require: configuration matrix = config matrix as int $confmatrix[8][6]$
    int $input[8]$ ¡- this array will be having input on which we want to perform operation
Ensure: Loop will run for all input lines of input file and copy to configuration
    matrix. So, the definition of input is below-
```

1. 8 input integer that we are going to take as an input to the n input. We will store 8 integer in array of size 8.

   Input[8];
2. Next comes configuration matrix. Here we have N/2 *(2logN -1) number of elements in the matrix. i.e., here in this case 8/2*(2log8-1) = 4*5 number of rows and column. I have taken a 1-dimentional array of size 20.

   ConfigurationMatrix[20];
3. One more 2-dimentional matrix I have taken where I have stored the values of previous changed position of the columns.

   Table[8][6]
4. So, the above is my inputs to the program.
5. In designing the 8*8 Benes network input-output flow. I have taken the table and in the 0th column I have put all the input and it will forward to next column as defined in the circuit and swap its positions according to the input of configurationMatrix. And this operation will continue till the end. And in the last row we the output.
6. Based on the configuration matrix we will generate a permutation of input in the form of output. That output I will take and save in a file.
7. We can get 2^{20} number of configuration matrix and input is 8 so, input permutation will be 8! As a output.
8. It is obviously2^{20} is much greater than 8!. So, definitely we will be getting some outputs same for different configurationMatrix.

Chapter 5

Result and Simulation

In 8*8 Benes Network, We got the output as permuted values and we counted all the occurrences permuted in the below table.

So, in the above table we can see the number of times each permuted output has occurred.

Before applying the Heuristics we got the below table.

permuted output	no. of times
16	14336
32	12288
8	8192
64	2816
40	2048
128	512
256	127
257	1

After applying the Heuristics we have got the below table.

So, we can clearly observe from the data that we can reduce the redundancy of Benes Network.

permuted output	no. of times
16	13312
32	12544
8	9216
64	512
56	256
256	80
192	32
65537	1

Chapter 6

Conclusion

We have all the permutation values of 8 input in the 8*8 Benes Network. We counted the occurrences of each permuted value. The graph was then plotted, yielding the image below. 6.1.

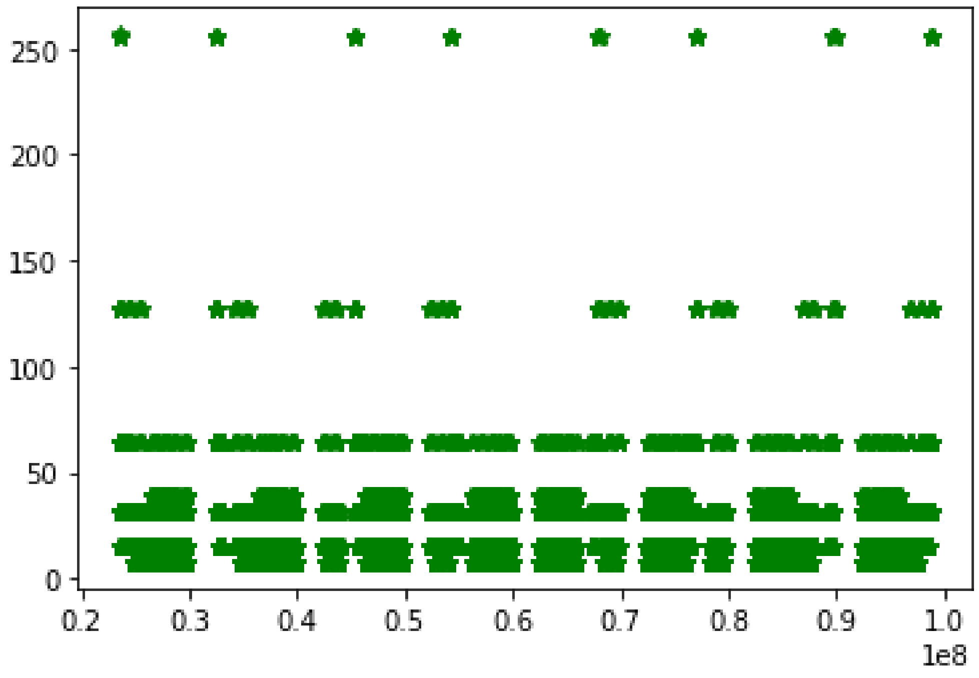

Figure 6.1: Distribution of all permuted value count

Heuristic-If the configuration matrix's column 2 and column 4 are mirror images of each other, the permuted value will be the same.

This above Heuristic we have applied on the 8*8 input benes network and then we have got the below distribution 6.2.

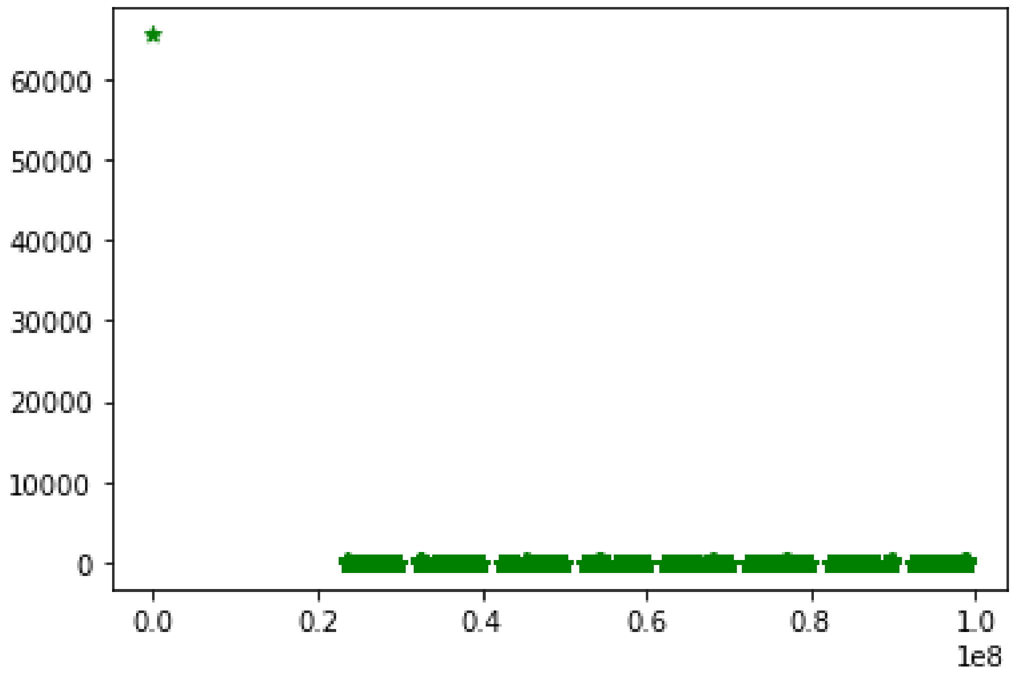

Figure 6.2: Distribution of all permuted value count after applying the lemma

We have got 65537 same permuted values and successfully able to detect and neglect this.

We used heuristics to reduce duplication (the number of times the same permutation occurs).

Chapter 7

Future work

In the following work, we will implement a 16*16 benes network and plot the distribution graph for it. We will continue to work to reduce redundancy in this network.

For k=even value (i.e. 10, 12) we will implement Benes Network.

Appendix A

A.1 8*8 Benes Network Implementation in c++

Listing A.1: Code to implement input file

```
#include <iostream>
#include <bits/stdc++.h>
#include <bitset>

using namespace std;

int main()
{
  ofstream file("inputfile.txt");
  bitset<20> bs;
  for (long i = 0; i < 1048576; i++)
  {
    bs = i;
    file << bs << endl;
  }
  file.close();
  return 0;
}
```

Listing A.2: implementation of Benes Network 8*8 input output

```
#include<bits/stdc++.h>
#include <fstream>
#include <iostream>

using namespace std;


int table[8][6] = {{0, 0, 0, 0, 0, 0},
                   {1, 2, 2, 2, 4, 1},
                   {2, 4, 1, 1, 1, 2},
                   {3, 6, 3, 3, 5, 3},
                   {4, 1, 4, 4, 2, 4},
                   {5, 3, 6, 6, 6, 5},
                   {6, 5, 5, 5, 3, 6},
                   {7, 7, 7, 7, 7, 7}};

// this function print the array.
void printArray(int arrref[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arrref[i] << "␣";
    }
    cout << endl;
}
\par

string operation(int inputref[8], int confref[20])
{
    int p;
    //string s="";
    int output[8];
    int tableref[8][6];
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            tableref[i][j] = table[i][j];
        }
    }

    //printArray(inputref, 8);
    //printArray(confref, 20);
    for (int i = 0; i < 8; i++)
    {
        tableref[i][0] = inputref[i];
    }
    for (int c = 0; c < 5; c++)
    {
        for (int r = 0; r < 8; r += 2)
        {
            if (confref[p] == 1)
            {
                int temp = tableref[r][c];
                tableref[r][c] = tableref[r + 1][c];
                tableref[r + 1][c] = temp;
            }
            p++;
        }

        for (int i = 0; i < 8; i++)
        {
            int val = tableref[i][c + 1];
            tableref[i][c + 1] = tableref[val][c];
        }
    }

    for(int i=0;i<8;i++){
        output[i]=tableref[i][4];
    }
   // printArray(output,8);

    ostringstream os;
    for(int i: output){
        os << i;
    }
    string str(os.str());
    //cout << str << endl;
    return str;
}
```

```
int main()
{
    int input[8] = {2, 3, 4, 5, 6, 7, 8, 9};

    int configurationMatrix[20];
    string line, outline;
    ifstream MyReadFile("inputfile.txt");
    ofstream MyWriteFile("resultfile.txt");
    if (MyReadFile.is_open())
    {
        while (getline(MyReadFile, line))
        {
            cout << line << endl;

            for (int i = 0; i < line.length(); i++)
            {
                configurationMatrix[i] = line[i] - 48;
            }
            outline=operation(input, configurationMatrix);
            outline+='\n';
            MyWriteFile << outline ;
        }
    }
    MyReadFile.close();
    MyWriteFile.close();
    return 0;
}
```

A.2 16*16 Benes Network implementation in C++

Listing A.3: Code to implement input file

```
// this is the program to find all the possible value of 5
#include <iostream>
#include <bits/stdc++.h>
#include <bitset>

using namespace std;

int main()
{
  ofstream file("inputfile.txt");
  bitset<56> bs;
  for (long i = 0; i < 72057594037927936; i++)
  {
    bs = i;
    file << bs << endl;
  }
  file.close();
  return 0;
}
```

Listing A.4: implementation of benes network 16*16

```
/*
below program is implementation of benes network of 16*16
*/
#include<bits/stdc++.h>

using namespace std;

int table [16][8] =   {{0    ,0   ,0   ,0   ,0   ,0   ,0
,0   },
                        {1    ,2   ,2   ,2   ,2   ,4   ,8

,1   },
                        {2    ,4   ,4   ,1   ,1   ,1   ,1

,2   },
                        {3    ,6   ,6   ,3   ,3   ,5   ,9

,3   },
                        {4    ,8   ,1   ,4   ,4   ,2   ,2

,4   },
                        {5    ,10  ,3   ,6   ,6   ,6   ,10  ,5

},
                        {6    ,12  ,5   ,5   ,5   ,3   ,3

,6   },
                        {7    ,14  ,7   ,7   ,7   ,7   ,11  ,7

},
                        {8    ,1   ,8   ,8   ,8   ,8   ,4

,8   },
                        {9    ,3   ,10  ,10  ,10  ,12  ,12  ,9

},
                        {10   ,5   ,12  ,9   ,9   ,9   ,5

,10  },
                        {11   ,7   ,14  ,11  ,11  ,13  ,13  ,11  },
                        {12   ,9   ,9   ,12  ,12  ,10  ,6

,12  },
                        {13   ,11  ,11  ,14  ,14  ,14  ,14  ,13  },
                        {14   ,13  ,13  ,13  ,13  ,11  ,7

,14  },
                        {15   ,15  ,15  ,15  ,15  ,15  ,15  ,15  }}

//we use below method to print the array containing charac
void printCharArray(char arrref[], int n)
```

```
{
    for (int i = 0; i < n; i++)
    {
        cout << arrref[i] << "␣";
    }
    cout << endl;
}

// we use below method to print the array containing integ
void printintArray(int arrref[], int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << arrref[i] << "␣";
    }
    cout << endl;
}

// below we do our operation, we pass variable one is inpu
string operation(char inputref[16], int confref[56]){
    int p;// this acts as a counter when p is 56 then it w
    char output[16];
    char inputToCopy[16];// this variable is used to copy
    int tableref[16][8];
    for(int i=0;i<16;i++){
        //cout<<inputref[i] << "␣";
        inputToCopy[i]=inputref[i];
    }

    for (int i = 0; i < 16; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            tableref[i][j] = table[i][j];
        }
    }
    printCharArray(inputref, 16);
    //printintArray(confref, 56);

    for (int c = 0; c < 7; c++)
    {
        for (int r = 0; r < 16; r += 2)
        {
            if (confref[p] == 1)
            {
                swap(inputref[r],inputref[r+1]);
            }
            p++;
        }

        for (int i = 0; i < 16; i++)
        {
            int val = tableref[i][c + 1];
            output[i] = inputref[val];
        }
        for(int i=0;i<16;i++){
            cout<<inputref[i] << "␣";
            inputref[i]=output[i];
        }
        cout << endl;
    }
    /*
    for(int i=0;i<16;i++){
        cout<<tableref[i][5];
        output[i]=tableref[i][6];
    }
    */
    //printCharArray(output,16);
    for(int i=0;i<16;i++){
        //cout<<inputref[i] << "␣";
        inputref[i]=inputToCopy[i];
    }

    ostringstream os;
    for(int i=0;i<16;i++){
        os << output[i];
    }
    string str(os.str());
    cout << str << endl;

    return str;
}
```

```
int main(){

    char input[16]={'a' ,'b' ,'c' ,'d' ,'e' ,'f' ,'g' ,'h'
    int configurationMatrix[56];
    string line, outline;
    ifstream MyReadFile("inputfile.txt");
    ofstream MyWriteFile("resultfile.txt");
    if (MyReadFile.is_open())
    {
        while (getline(MyReadFile, line))
        {
            cout << line << endl;

            for (int i = 0; i < line.length(); i++)
            {
                configurationMatrix[i] = line[i] - 48; //4
                //cout<<configurationMatrix[i];
            }
            outline=operation(input, configurationMatrix);
            outline+='\n';
            MyWriteFile << outline ;
        }
    }
    MyReadFile.close();
    MyWriteFile.close();
    return 0;


}
```

Bibliography

[1] Sunitha S adhav. *Advanced Computer Architecture and Computing. Technical Publications. pp. Section 3–22*, chapter intro, page 192. oninternet, 2009.

[2] Abdulaziz S Almazyad. Optical omega networks with centralized buffering and wavelength conversion. *Journal of King Saud University-Computer and Information Sciences*, 23(1):15–28, 2011.

[3] Hasan Cam and Jose A. B. Fortes. Work-efficient routing algorithms for rearrangeable symmetrical networks. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):733–741, 1999.

[4] Amitabha Chakrabarty, Martin Collier, and Sourav Mukhopadhyay. Matrix-based nonblocking routing algorithm for beneš networks. In *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pages 551–556. IEEE, 2009.

[5] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[6] Xu Jia, Feng Xin, and Wang Ru Chuan. Adaptive spray routing for opportunistic networks. *International Journal on Smart Sensing and Intelligent Systems*, 6(1), 2013.

[7] Abbas Karimi, Abbas Afsharfarnia, Faraneh Zarafshan, and SAR Al-Haddad. A novel clustering algorithm for mobile ad hoc networks based on determination of virtual links' weight to increase network stability. *The Scientific World Journal*, 2014, 2014.

[8] Abbas Karimi, Kiarash Aghakhani, Seyed Ehsan Manavi, Faraneh Zarafshan, and SAR Al-Haddad. Introduction and analysis of optimal routing algorithm in benes networks. *Procedia Computer Science*, 42:313–319, 2014.

[9] Chi-Ping Lee, Chien-Ping Chang, Jiun-Shiou Deng, Min-Hao Li, Ming-Feng Lu, Yang-Tung Huang, and Ping-Yu Kuei. A strictly nonblocking network based on nonblocking 4× 4 optical switches. *Optical Switching and Networking*, 9(1):1–12, 2012.

[10] Kyungsook Yoon Lee. A new benes network control algorithm. *IEEE Transactions on Computers*, 100(6):768–772, 1987.

[11] Morgan Kaufmann Publishers Leighton, F.Thomson. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, chapter intro, page 19. oninternet, 1992.

[12] Enyue Lu and SQ Zheng. Fast reconfiguration algorithms for time, space, and wavelength dilated optical benes networks. *The International Journal of Parallel, Emergent and Distributed Systems*, 22(1):39–58, 2007.

[13] H. Sarbazi-Azad M. Arjomand. *Performance Evaluation of Butterfly on-Chip Network for MPSoCs", International SoC Design Conference*, chapter intro, page 19. oninternet, 2008.

[14] David C Opferman and Nelson T Tsao-Wu. On a class of rearrangeable switching networks part i: Control algorithm. *The Bell System Technical Journal*, 50(5):1579–1600, 1971.

[15] Ville Rantala, Teijo Lehtonen, and Juha Plosila. Network on chip routing algorithms. ", 12 2008.

[16] Lionel Ni Science Direct, José Duato. Interconnecting networks, 2003. Multistage interconnection networks (MINs).

[17] Solihin Publishing Solihin 2009 and Consulting LLC. *Fundamentals of Parallel Computer Architecture: Multichip and Multicore Systems*, page 371–372. oninternet, 12 2009.

[18] Ajithkumar Thamarakuzhi and John A Chandy. 2-dilated flattened butterfly: A nonblocking switching topology for high-radix networks. *Computer Communications*, 34(15):1822–1835, 2011.

[19] Brown T.LeBlanc M., Scott C. *Large-Scale Parallel Programming: Experience with the BBN Butterfly Parallel Processor*, chapter intro, page 19. oninternet, 1988-01-01.

[20] Chu-Sing Yang and LP Zu. Two expansible multistage interconnection networks. In *Proceedings of 1994 International Conference on Parallel and Distributed Systems*, pages 373–378. IEEE, 1994.

[21] K Yangon, F Tsc-Yun, and S Seung-Woo. On a class of concatenated (2log2n-1)-stage interconnection network. In *IEEE Conferences*, pages 537–543, 1996.