Spring 1991

Multistage interconnection networks : improved routing algorithms and fault tolerance

Kuo-Yu Chen
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Multistage Interconnection Networks: Improved Routing Algorithms and Fault Tolerance

by
Kuo-Yu Chen

Thesis submitted to the Faculty of the Graduate School of
the New Jersey Institute of Technology in partial fulfillment of
the requirements for the degree of
Master of Science in Electrical Engineering
1991

APPROVAL SHEET

Title of Thesis: Multistage Interconnection Networks:
Improved Routing Algorithms and Fault Tolerance

Candidate: Kuo-Yu Chen
Master of Science in Electrical Engineering, 1991

Thesis and Abstract Approved by the Examining Committee:

Dr. John Carpinelli, Thesis Advisor — Date
Assistant Professor
Department of Electrical and Computer Engineering

Dr. Anthony Robbi — Date
Associate Professor
Department of Electrical and Computer Engineering

Dr. Sotirios Ziavras — Date
Assistant Professor
Department of Electrical and Computer Engineering

New Jersey Institute of Technology, Newark, New Jersey.

Vita

Name: Kuo-Yu Chen

Degree and date to be conferred: M.S.E.E., 1991

Collegiate institutions attended	Dates	Degree	Date of Degree
Tam-Kang University	9/77-5/81	B.S.E.E.	May 1986
Science and Technology			
New Jersey Institute of Technology	9/87-12/88	M.S.E.E.	May 1991

Major: Electrical Engineering

Position held: Candidate of M.S.E.E.

Abstract

Title of Thesis: Multistage Interconnection Networks: Improved Routing Algorithms and Fault Tolerance

Kuo-Yu Chen, Master of Science in Electrical and Computer Engineering, 1991

Thesis directed by: Assistant Professor Dr. John D. Carpinelli

Multistage interconnection networks for use by multiprocessor systems are optimal in terms of the number of switching element, but the routing algorithms used to set up these networks are suboptimal in terms of time. The network set-up time and reliability are the major factors to affect the performance of multistage interconnection networks. This work improves routing on Beneš and Clos networks as well as the fault tolerant capability. The permutation representation is examined as well as the Clos and Beneš networks. A modified edge coloring algorithm is applied to the regular bipartite multigraph which represents a Clos network. The looping and parallel looping algorithms are examined and a modified Tree-Connected Computer is adopted to execute a bidirectional parallel looping algorithm for Beneš networks. A new fault tolerant Clos network is presented.

Acknowledgement

The author wishes to thank his thesis committee for their help and advice. Special thanks are extented to Prof. Carpinelli for providing much time and effort throughout his time at the New Jersey Institute of Technology. Finally, the author wishes to thank his family for their support, help and encouragement throughout his graduate studies.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

As the number of task and complexity of scientific and engineering applications such as weather forecasting, artificial intelligence, expert system are increasing, high-performance computers are increasingly in demand. Although VLSI techniques have considerably increased the performance and reliability of hardware devices for advanced computer architecture, parallel computation as well as fast algorithms are also needed to meet the above demands. Through parallel computers and computing algorithms, the tasks can be divided into subtasks and the subtasks can be executed simultaneously. The interconnection network can provide the parallel communication needs between multiple processors, shared memory modules and I/O systems. The network set-up time and system reliability are the major factors that affect the system performance. To reduce the set-up time and provide the fault tolerant capability are the major goals of this thesis.

1.2 Background

1.2.1 Parallel Computers

Parallel computers can be characterized into three structural classes: pipelined computers, array processors, and multiprocessor systems. In pipelined computers, successive instructions are executed in an overlapped fashion in terms of execution steps per instruction[11,14,15,19]. An array processor is a synchronous parallel computer which consists of multiple processing elements (PEs), a control unit (CU), and an interconnection network (IN). The CU broadcasts instructions to the PEs and all the PEs execute in parallel in a lock-step fashion on different data. The IN provides communication paths between PEs and for processor-memory communication.

Array processors are also known as SIMD machines; that stands for single instruction and multiple data streams. It is specially designed to perform vector computations over matrices or arrays of data[16,24,25,26]. A multiprocessor system is a single computer that contains multiple processors. All processors may share access to memory modules, I/O channels, and peripheral devices. Processors may communicate and cooperate at different levels in solving a given problem. A multiprocessor system is controlled by a single integrated operating system instead of several autonomous computers in a multiple computer system. Unlike the SIMD machines which execute synchronously, the processors in multiprocessor systems can asynchronously, autonomously execute different instructions on different data. Those are considered as MIMD machines[2,7,13].

Multiprocessor systems can be classified into tightly coupled and loosely coupled multiprocessor systems. Figure 1.1 shows the basic organization of a nonhierachical loosely coupled multiprocessor system. In such systems, each processor has a set of I/O devices and a large memory. We refer to a processor, its local memory, and

I/O devices as a computer module. Computer modules communicate through an interconnection network. Since the processor accesses most of the instructions and data from its local memory, a loosely coupled system is often referred to as a distributed system. Processors in tightly coupled multiprocessor systems communicate through shared memory modules. Each processor may have a small local memory and buffer cache. An interconnection network provides the complete connectivities between processors, shared memory modules, and I/O devices. The configuration of a tightly coupled multiprocessor system is shown in figure 1.2.

1.2.2 Multiprocessor Interconnection Networks

Multiprocessor interconnection networks[8,10] are classified into bus interconnection, crossbar interconnection, and multistage interconnection networks. The time shared bus organization, as shown in figure 1.3, is the simplest way to give multiple processors access to a shared memory. Crossbar interconnection networks, shown in figure 1.4, use a crossbar switch of N^2 crosspoints to connect N processors to N memories; this is considered a strictly non-blocking network. Multistage interconnection networks (MINs) have long been studied for use in telephone switching and multiprocessor systems to meet the communication needs of multiprocessor systems in a cost-effective manner. Although crossbar systems prevent bus contention by providing dedicated paths between processors and memories, the hardware cost has limited it to a small number of processors (from four to sixteen processors).

Typically MINs designed for N inputs and N outputs contain $O(N \lg N)$ stages of N/m crossbar switching elements of size $m \times m$. The area complexity of a MIN is $O(N \lg N)$, compared to $O(N^2)$ of the crossbar network. A Baseline network is shown in figure 1.5. But there is a trade-off between cost and performance. The set-up time, $O(N \lg N)$, for a MIN is much larger than the delay time, $O(\lg N)$, when

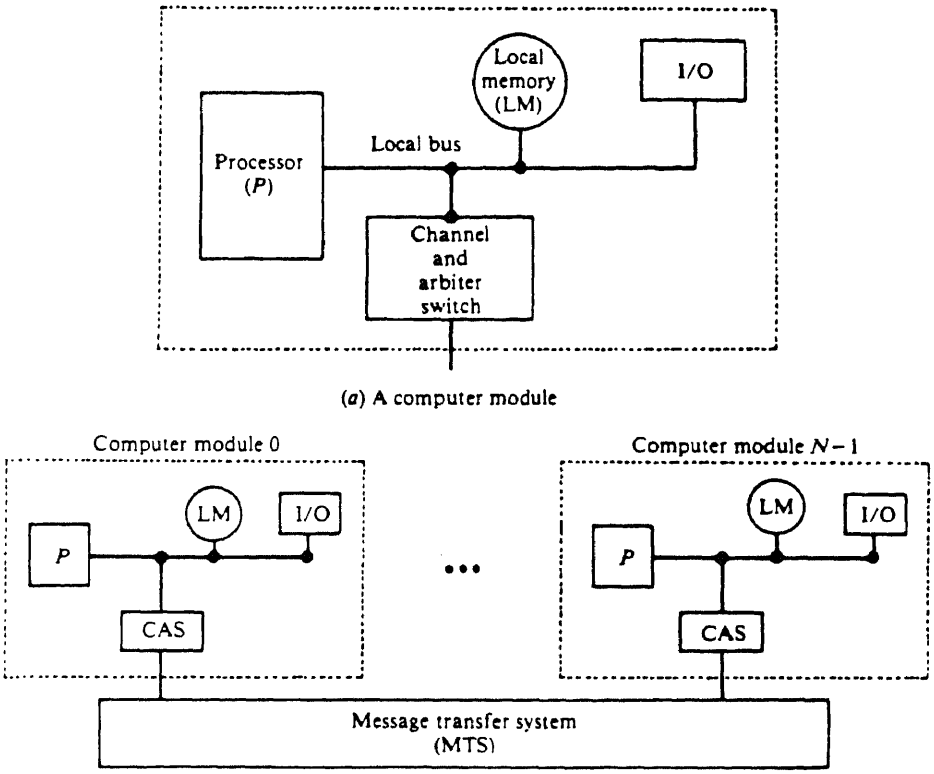


Figure 1.1: A loosely coupled multiprocessor system

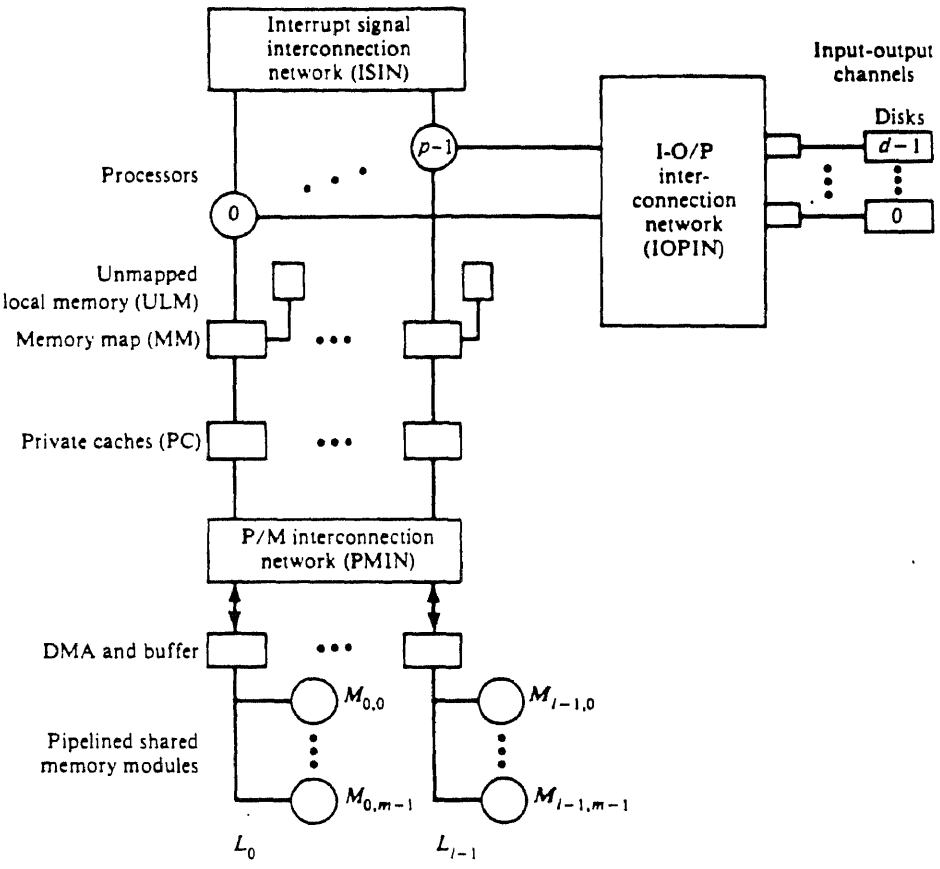


Figure 1.2: A tightly coupled multiprocessor system

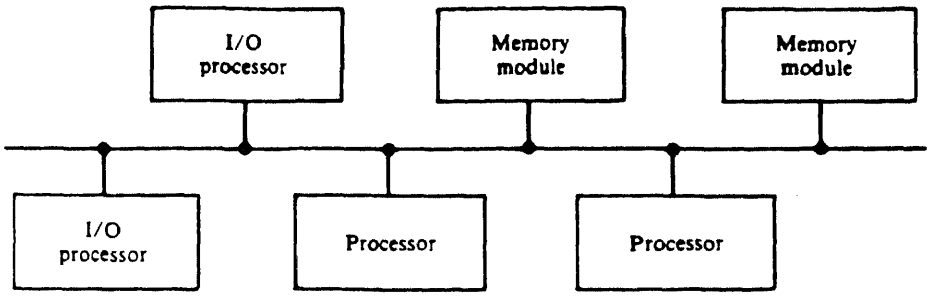


Figure 1.3: A time shared bus system

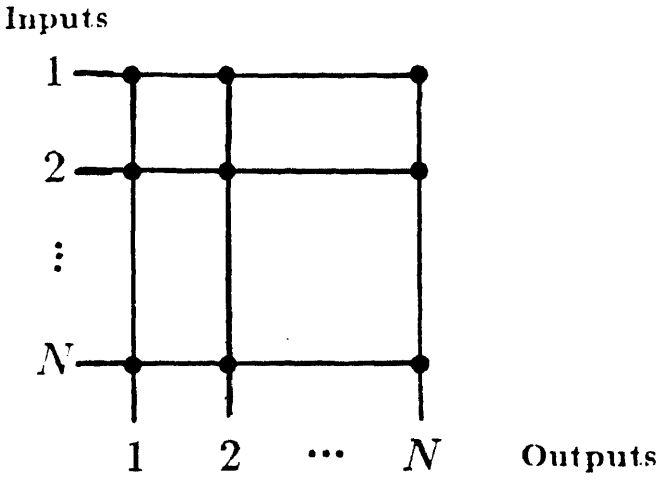


Figure 1.4: A crossbar switching system

routing algorithms are executed in a single processor computer. A lot of research has been directed toward network routing algorithms for different MINs, such as Omega[18], Baseline[30], and shuffle-exchange[27,29]. However, unlike Clos and Beneš networks, those networks have limited permutation capability. Much research has been dedicated to the non-blocking Clos and Beneš networks[3,6,21,22,23,28].

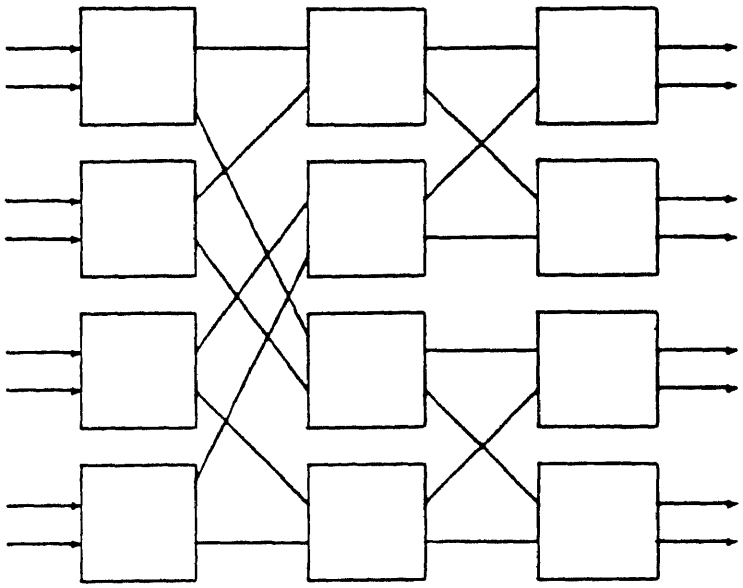


Figure 1.5: A multistage Baseline interconnection network

1.2.3 Fault Tolerant Multistage Interconnection Networks

The reliable operation of an interconnection network is an important factor when evaluating system performance. This issue can be classified into fault diagnosis and fault tolerance. The fault diagnosis concerns fault-detection and fault-location for every fault in a target system. The main concern in fault tolerance is the full connection capability with graceful degradation in spite of the existence of faults. In recent years, much research has been reported on multiple-path MINs to provide alternate path when faults occur[1,17,20].

1.3 Outline

This thesis is dedicated to improve the routing algorithms on Clos and Beneš networks as well as fault-tolerant Clos networks. The rest of the thesis is organized as follows. In Chapter 2, the permutation representation for network settings is introduced. The Clos and Beneš networks are also discussed. The regular bipartite multigraph representation of Clos networks and edge coloring algorithms as applied to decompose bipartite multigraphs are examined in Chapter 3. Also, a modified edge coloring algorithm that includes Euler partition and maximum matching procedures is presented. In Chapter 4, the looping algorithm, run on a single processor computer, and the parallel looping algorithm, run on a Tree-Connected computer, are introduced for routing algorithms of Beneš network. A new bidirectional parallel looping algorithm is presented. In Chapter 5, the fault model and design objectives are discussed and a new fault-tolerant Clos network which provide 3-fault robust capability is presented. A comparison with the original Clos network and Nassar's FTC[20] is discussed. Finally, conclusions and possible further research are presented in Chapter 6.

Chapter 2

Multistage Interconnection Networks

2.1 Introduction

A multistage interconnection network is capable of connecting an arbitrary inlet to an arbitrary outlet. It can be classified into three classes: blocking, rearrangeably non-blocking, or strictly non-blocking. In a blocking network, connections of input-output pairs may result in conflicts in the use of the network links. For example, Omega and Baseline networks are blocking networks[27,29,30]. A rearrangeable non-blocking network can connect all possible connections between inlets and outlets by possibly rearranging its existing paths. For example, Beneš and Clos network belong to this class. A network is strictly non-blocking if it can realize all possible connections without blocking, or modifying the existing paths, such as crossbar networks.

In Section 2.2 the permutation representations for a switch, stage, and network settings are described. The rearrangeable Clos network is discussed in Section 2.3. The rearrangeable Beneš network is examined in Section 2.4.

2.2 The permutation representation

A multistage interconnection network consists of several stages of switches. The outputs of one stage are connected to the inputs of the next stage. Each stage contains several disjoint switches. A switch is the basic element in a network. A switch with N inputs, and N outputs can realize $N!$ one-to-one mappings.

The permutation notation is one of the common methods used to describe the mappings in a switch, stage, and network. A permutation is a one-to-one mapping of a set element onto itself. In standard permutation notation, there are two rows of values. The values in the upper row are mapped onto the elements in the lower row. For example, given a set of values {0, 1, 2, 3} for which 0 maps to 1, 1 to 2, 2 to 3, and 3 to 0, the permutation, P, of the set is written as $P = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \end{pmatrix}$. A permutation must contain one or more disjoint cycles. The following example consists of two cycles.

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} 4 & 5 & 6 & 7 \\ 7 & 4 & 5 & 6 \end{pmatrix} \text{ or } P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 0 & 1 & 2 & 7 & 4 & 5 & 6 \end{pmatrix}$$

The setting of a switch is a set of one-to-one mappings of input onto output, so it can be represented by a permutation. Consider the switch shown in figure 2.1, with four inputs and four outputs. The equivalent permutation representation is $\begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 3 & 2 & 0 \end{pmatrix}$.

As mentioned before, a stage consists of several disjoint switches. Consider a stage with two 3×3 switches; the inputs and outputs of the first switch are labelled 0, 1, 2 and the inputs and outputs of the second switch are labelled 3, 4, 5. The first switch realizes the setting $\begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{pmatrix}$ and the second one realizes $\begin{pmatrix} 3 & 4 & 5 \\ 5 & 3 & 4 \end{pmatrix}$. Combining these two disjoint permutations to form the permutation representation of the stage setting, as shown in figure 2.2, $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 0 & 5 & 3 & 4 \end{pmatrix}$.

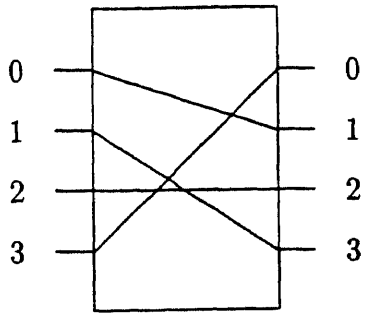


Figure 2.1: A switch setting

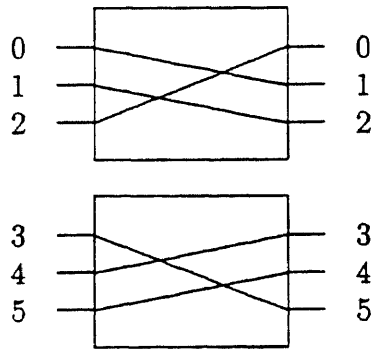


Figure 2.2: A stage setting

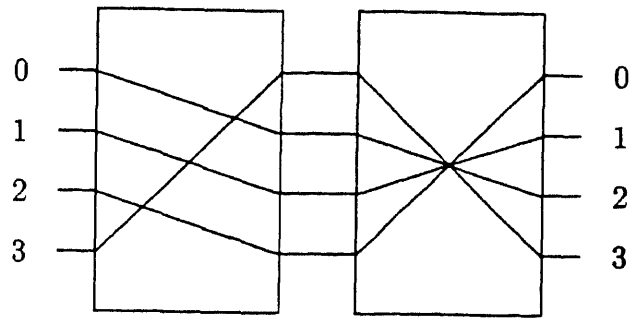


Figure 2.3: A network setting

The permutation notation of a network setting is the ordered composition of the permutation of each stage. Consider a network with four inputs and two stages. The first stage realizes the permutation $P1 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \end{pmatrix}$, and the second stage realizes the permutation $P2 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 \end{pmatrix}$. Inlet 0 of the first stage is mapped onto outlet 1 and the input 1 of the second stage is mapped onto the output 2. Since the output of the first stage is connected to the input of the second stage, the input 0 is mapped onto the output 2 through the entire network. Repeating the same process, the permutation realized by the network is $P = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 3 \end{pmatrix}$. The network setting is shown in figure 2.3.

2.3 Clos Network

A Clos network is a 3-stage network, originally developed for telephone traffic routing[6]. The first stage contains k $m \times n$ switches which are numbered from 0 to $k-1$. The inputs of the ith switch are labelled from $m \cdot i$ to $m \cdot (i+1) - 1$. The second stage consists of n $k \times k$ switches and the third stage contains k $n \times m$

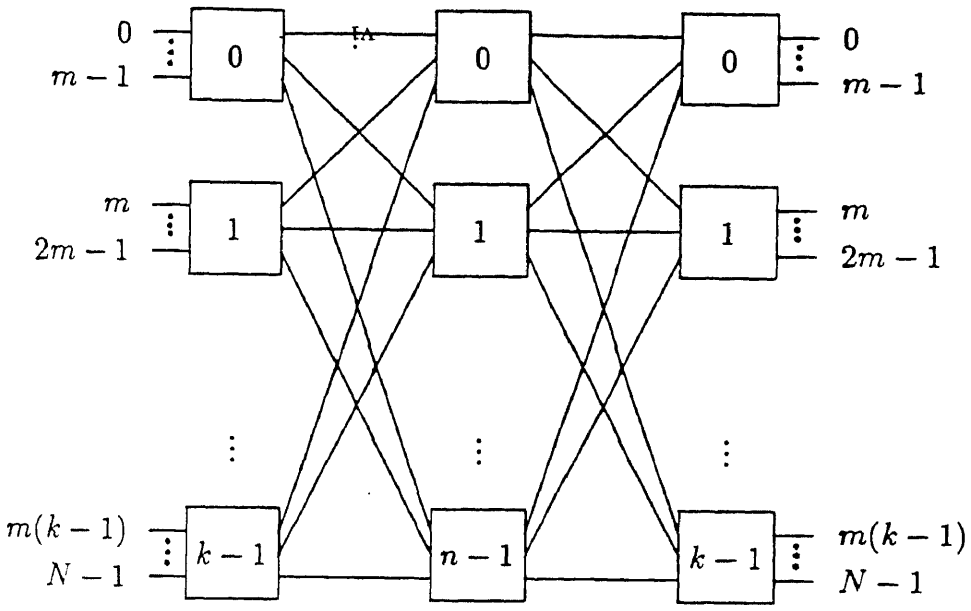


Figure 2.4: 3-stage (m, n, k) Clos network

switches. Each switch can realize any one-to-one mapping of its inputs onto its outputs. Each switch in the center stage derives one input from each switch in the first stage and each switch in the third stage receives one input from each switch in the center stage.

A Clos network with these parameters described above is referred to as an (m, n, k) Clos network with size $N = k \cdot m$; it is shown in figure 2.4. If $n \geq m$, the network is a rearrangeable Clos network. If $n \geq 2m - 1$, the network is a strictly non-blocking Clos network. The number of inputs and outputs to a Clos network is referred as the size N.

In the rest of the thesis, we will consider the regular (m, n, k) Clos network with $m = n$ only.

2.4 Beneš Network

A Beneš network[3] is derived from the 3-stage (m, n, k) Clos network with $m = n = 2$, and $k = \frac{N}{2}$ by decomposing the $k \times k$ switches in the center stage into 3-stage Clos networks and continuing the decomposition until each switch is a 2×2 cell.

To illustrate the decomposition process, consider a Clos network with $n = m = 2$, and $k = 4$ shown in figure 2.5. Each 4×4 switch in the center stage can be considered as a subnetwork and decomposed into a $(2, 2, 2)$ Clos network. Figure 2.6 shows the final result of decomposing a $(2, 2, 4)$ Clos network and that is an 8×8 Beneš network. An N-input Beneš network has $2(\lg N) - 1$ stages and $O(N \lg N)$ 2×2 crossbars.

Waksman[30] had shown that one switch can be removed from the first stage or the last stage of a Beneš network without affecting the rearrangeability. For example, this yields the 4×4 Waksman network, shown in figure 2.7, and the 8×8 network, shown in figure 2.8. The looping algorithms discussed in chapter 4 will use the result of the Waksman network.

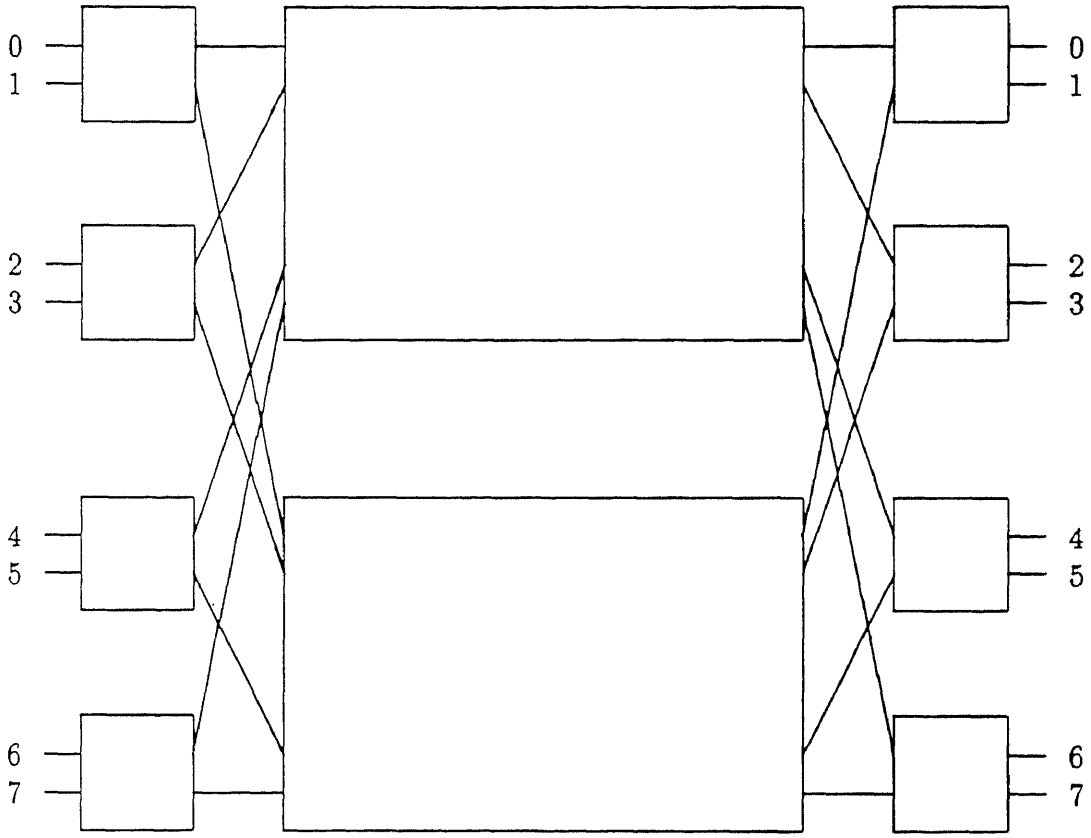


Figure 2.5: $(2, 2, 4)$ Clos network

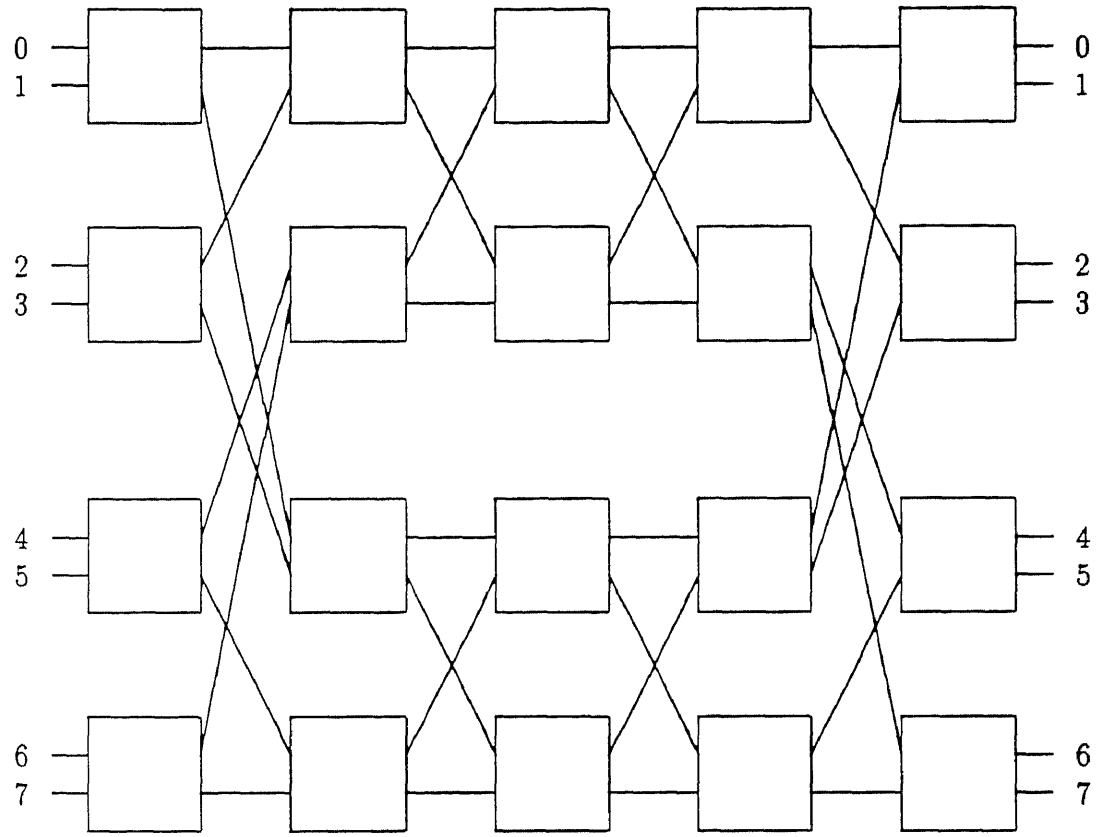


Figure 2.6: 8×8 Beneš Network

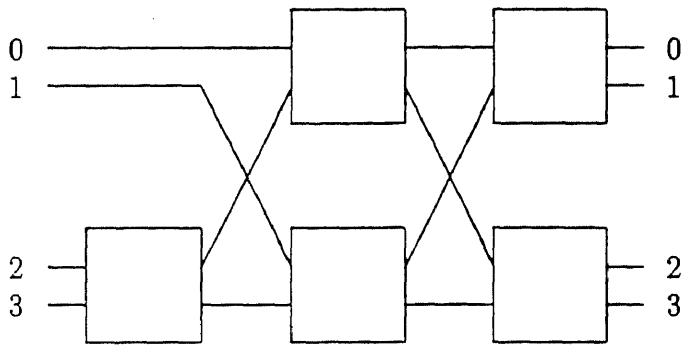


Figure 2.7: 4×4 Waksman network

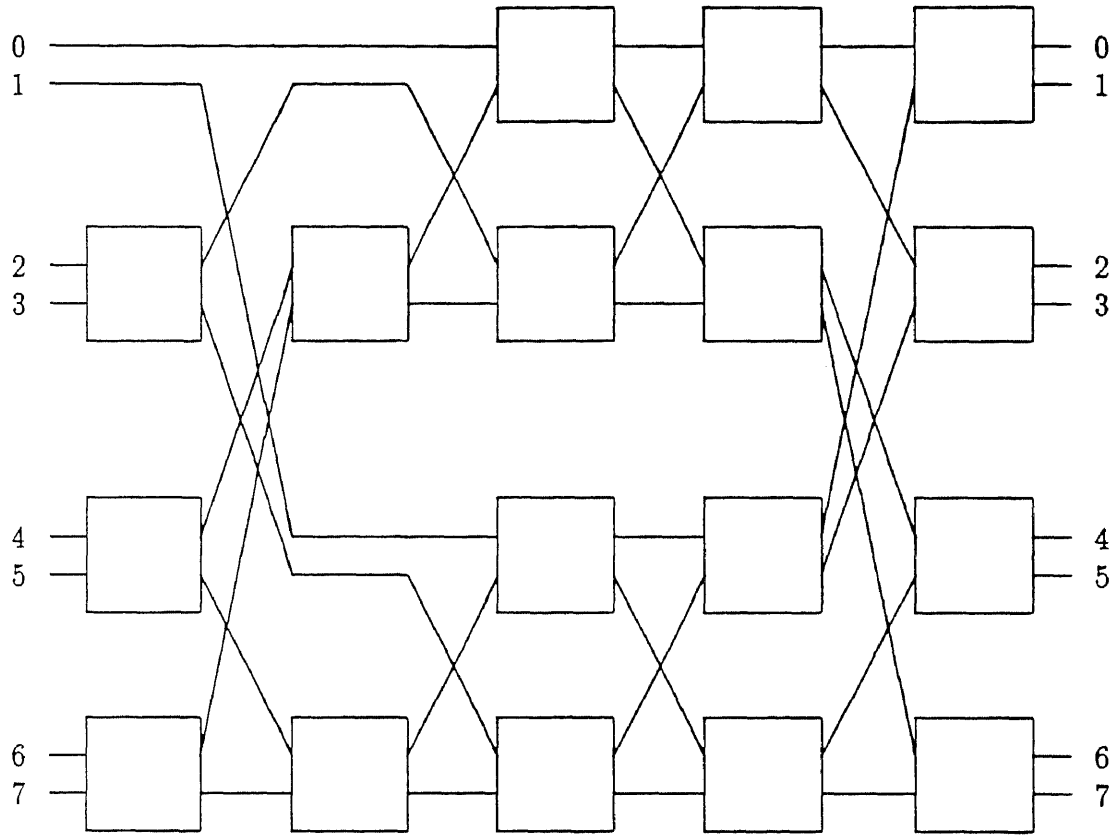


Figure 2.8: 8×8 Waksman network

Chapter 3

Routing on Clos Networks

3.1 Introduction

This chapter examines the edge coloring method in bipartite multigraph and its application on Clos networks. The bipartite multigraph representation of the Clos network is described in Section 3.2. There are two major methods of edge coloring: Vizing's method[31], and Euler partitions. The Euler partition procedure adopted in Gabow's Edge Coloring Algorithm[9] will be discussed in Section 3.3. A matching M is the set of edges of a multigraph such that no two of the edges are incident to the same vertex. In Section 3.4, two methods of matching will be discussed: Gabow's MD procedure[9], which finds a matching that covers all the vertices with maximum degree Δ, and Hopcroft and Karp's Maximum Matching algorithm[12] which finds a matching with the greatest possible number of vertices. In Section 3.5, the Edge Coloring Algorithm developed by Gabow is modified to take the feature of the regular bipartite multigraph and apply it to Clos networks.

3.2 Bipartite Multigraph representation

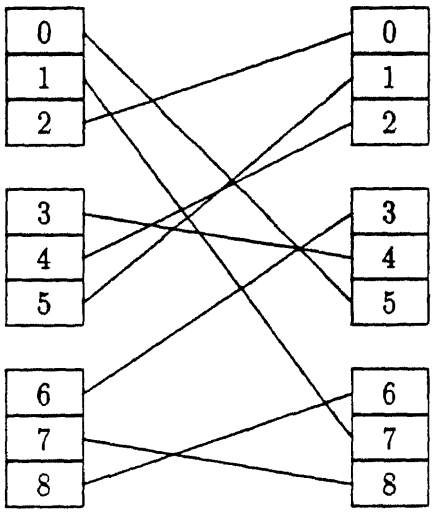
As described in Chapter 2, a 3-stage (m, n, k) Clos network has size $N = m \times k$, k $m \times n$ switches in the first stage, n $k \times k$ switches in the center stage, and k

$n \times m$ switches in the last stage. Since we only consider the regular Clos network with $m = n$, a permutation realized by a Clos network can be represented by a regular bipartite multigraph.

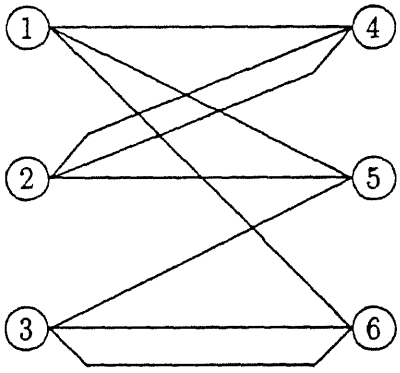
A graph consists of two sets, V, and E. The elements of V are called vertices, points, or nodes. The set E, the edges, contains unordered pairs of distinct vertices (v, u). The graph is denoted by $G(V, E)$. If an edge (v, u) occurs more than once, G is a *multigraph*. An undirected graph is said to be bipartite if its vertices can be partitioned into two disjoint subsets $V1$ and $V2$, such that each edge is incident to a vertex in $V1$ and a vertex in $V2$.

The bipartite multigraph can be represented by a triplet $\{V1, V2, E\}$. The degree of a vertex, v, is equal to the number of edges which are incident to v. The degree of a multigraph is equal to the highest degree of its vertices. If every vertex has the same degree k, a multigraph is called k-regular. A regular (m, m, k) Clos network has m inlets in each switch of the first stage, and m outlets in each switch of the last stage. Each switch in the first and the last stage can be viewed as a vertex with degree m. Each switch in the first stage is an element of set $V1$ and each switch in the last stage is an element of set $V2$. So, the input-output mappings of a regular Clos network can be represented by a regular bipartite multigraph. Consider the permutation $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 7 & 0 & 4 & 2 & 1 & 3 & 8 & 6 \end{pmatrix}$ realized by a $(3, 3, 3)$ Clos network. Figure 3.1.a shows the mappings between the first and last stages, and figure 3.1.b is the corresponding regular bipartite multigraph representation.

As mentioned before, an (m, m, k) Clos network has m $k \times k$ switches in the center stage. Applying edge coloring, discussed in Section 3.5, to partition the m-regular bipartite multigraph into m subgraphs, each subgraph will be the setting of a switch in the center stage. After the settings of the center stage are determined, the settings in the first and the last stage will be easy to determine.



(a)



(b)

Figure 3.1: (a) The mapping of the first and last stage of the Clos network (b) The regular bipartite multigraph representation of figure 3.1.a.

3.3 Euler Partition

The Euler partition divides the edges of a graph into open and closed paths which form a set P. A path, p, is a sequence of edges — (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n). If $v_1 \neq v_n$, p is an open path. If $v_1 = v_n$, p is a closed path. Each vertex with odd degree must be one end of an open path.

A path is formed by the following processes. Start from an arbitrary vertex v_1 of odd degree (or non-zero even degree if no vertex has odd degree) and choose an edge incident to that vertex. Add the edge to path p, then traverse the edge to the other vertex v_2 which the edge is incident to. Delete the edge from the graph. Start from v_2 and continue to choose, traverse, add, and delete edges until a vertex with zero degree is reached. This gives a path p. Then choose a new starting vertex and repeat the same process until the set E in the graph G is empty.

The procedure EP, developed by Gabow[9], is presented below:

PROCEDURE EP;

BEGIN

1. Make P an empty list;
2. Make S an empty queue;
3. Put all vertices of odd degree in S;
4. Put all vertices of non-zero even degree in S;
5. **WHILE** S is non-empty **DO**

   BEGIN

6. Let s be the first vertex in S;
7. Delete s from S;

   COMMENT vertex s may have degree 0, since edges are deleted from G in line 13;

```
8.      IF vertex s has non-zero degree THEN
            BEGIN
9.          Make a new path p empty;
10.         v:= s;
11.         WHILE vertex v has non-zero degree DO
                BEGIN
12.             Let (v, w) be an edge in G;
13.             Delete (v, w) from G;
14.             Put (v, w) in p;
15.             v:= w;
                END;
16.         Put path p in P;
17.         IF vertex s has non-zero degree THEN Put s in S;
            END;
        END;
    END EP;
```

The procedure EP finds an Euler partition in $O(E)$ time and uses $O(E)$ space.

To illustrate this procedure, consider the regular bipartite multigraph in figure 3.1.b. Figure 3.1.b is redrawn in figure 3.2. Start from vertex 1 which has degree 3, traverse the edges 1, 4, 6, 7, 8, 9 to form an open path p_1. Delete those edges from G and start from vertex 2 which now has degree 1. Then traverse edge 5 from v_2 to v_4 to form path p_2 because v_4 has zero degree after deleting edge 5 from G. Finally, start from vertex v_5 and traverse edges 2, 3 to form path p_3. At the end of EP, the list P will contain paths p_1, p_2 and p_3.

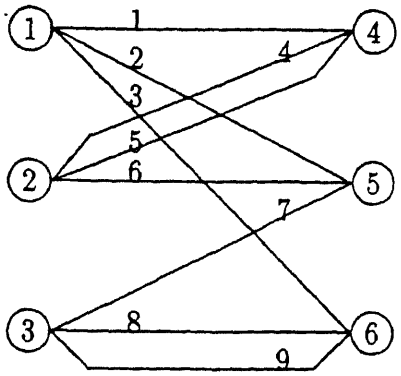


Figure 3.2: Regular bipartite multigraph representation of figure 3.1.a

3.4 Matchings

This section describes Gabow's matching procedure MD, and Hopcroft and Karp's Maximum Matching algorithm. The Maximum Matching will replace the MD procedure in Gabow's Edge Coloring Algorithm for the Clos network.

A matching of a graph is a set of edges such that no two of the edges are connected. A maximum matching contains the greatest possible number of edges. For a k-regular bipartite multigraph denoted (V_1, V_2, E), a maximum matching must contain k edges and cover all of the vertices in sets V_1 and V_2. Let Δ be the maximum degree of a bipartite multigraph; there exists a minimal edge coloring which uses exactly Δ colors[4], and a matching can be colored by one color.

Gabow's matching procedure MD finds a matching M that covers every vertex with degree Δ. A matching can be found as follows:

1. For $i = 1, 2$, find a maximum matching M_i that covers every vertex of degree m in V_i.

2. Put all edges of $M_1 \cap M_2$ in M.

3. Put all edges of $M_1 \oplus M_2$ in N. (A connected component C in N is a path with edges alternately in M_1 and M_2.)

4. For each connected component C in N, put all edges of $M_i \cap C$ in M, where i is determined by C. If C is an open path of odd length, then choose i so $\mid M_i \cap C \mid$ is maximum. If C is an open path with even length, exactly one end of C is a vertex of degree Δ. Choose i so that M_i covers that vertex. If C is a closed path, choose i arbitrarily.

After step 4, M is the matching that covers every vertex of degree Δ. The procedure MD is presented below:

```
PROCEDURE MD;
BEGIN
1.  FOR i := 1, 2 DO COMMENT find a matching M_i that covers every vertex
        of maximum degree Δ in S_i;
    BEGIN
2.      Let T be the set of vertices in S_i that do not have maximum degree;
3.      let H be the multigraph G − T;
4.      let M_i be a maximum matching on H;
    END;
    COMMENT form M from M_1 and M_2;
5.  M := M_1 ∩ M_2;
6.  N := M_1 ⊕ M_2;
7.  FOR each connected component C of N DO
    BEGIN
8.      Let C be the sequence of edges e_1, e_2, ..., e_r;
9.      Without loss of generality assume C starts with a vertex of degree Δ.
```

10. **FOR** $i := 1$ **STEP** 2 **TO** r **DO**

11. Put e_i in M;

END;

END MD;

Lines 1-4 perform step 1, line 5 implements step 2, line 6 executes step 3, and lines 7-11 complete step 4. The total run time is $O(V^{\frac{1}{2}}E)$ and the algorithm uses $O(E)$ space. To illustrate this procedure, consider the bipartite multigraph shown in figure 3.3 which has maximum degree 3 in vertices 2, 4, 5 and 6. After step 1, M_1 and M_2 are shown in figure 3.4.a. M is empty after step 2. In step 3, N consists of two connected components shown in figure 3.4.b. Figure 3.4.c shows the final matching M after step 4.

Gabow's MD is suited to irregular bipartite multigraphs only. For a regular bipartite multigraph, only line 4 of MD procedure is needed to find the matching M (or maximum matching) which covers all vertices of degree Δ. Since each vertex has the same degree in a regular bipartite multigraph, the set T in line 2 is always empty, and H is always equal to G. When M_1 is found in line 4, actually it is the final result M for procedure MD.

In line 4, Gabow used Hopcroft and Karp's Maximum Matching algorithm[12] to find the maximum matching M_i. So we will use the Maximum Matching algorithm to replace the MD procedure in Gabow's Edge Coloring algorithm[9] to apply to a regular bipartite multigraph.

Hopcroft and Karp's Maximum Matching algorithm is presented below:

step 0: $M \leftarrow \Phi$.

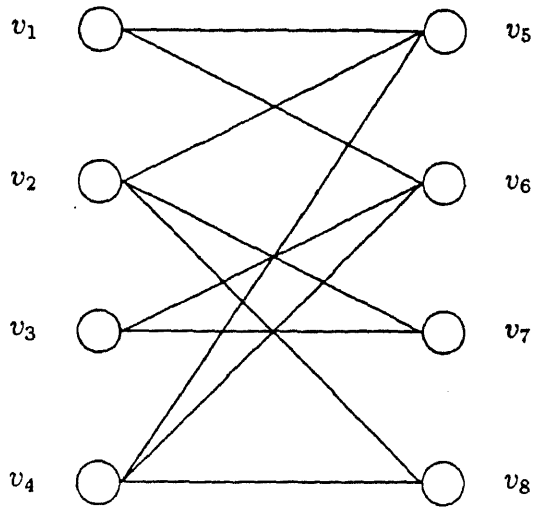


Figure 3.3: A bipartite multigraph with maximum degree 3

step 1: Let $l(M)$ be the length of a shortest augmenting path relative to M. Find a maximal set of paths $\{Q_1, Q_2, \ldots, Q_t\}$ with the properties that

(a) for each i, Q is an augmenting path relative to M and $\mid Q_i \mid = l(M)$;

(b) the Q_i are vertex disjoint.

Halt if no such paths exist.

step 2: $M \leftarrow M \oplus Q_1 \oplus Q_2 \oplus \ldots \oplus Q_t$; go to step 1.

A path (without repeated vertices)

$$P = (v_1, v_2), (v_2, v_3), ..., (v_{2k-1}, v_{2k})$$

is called an augmenting path if v_1 and v_{2k} are both free and its edges are alternately in $E - M$ and in M. A vertex is free if it is incident to no edge in M. Two

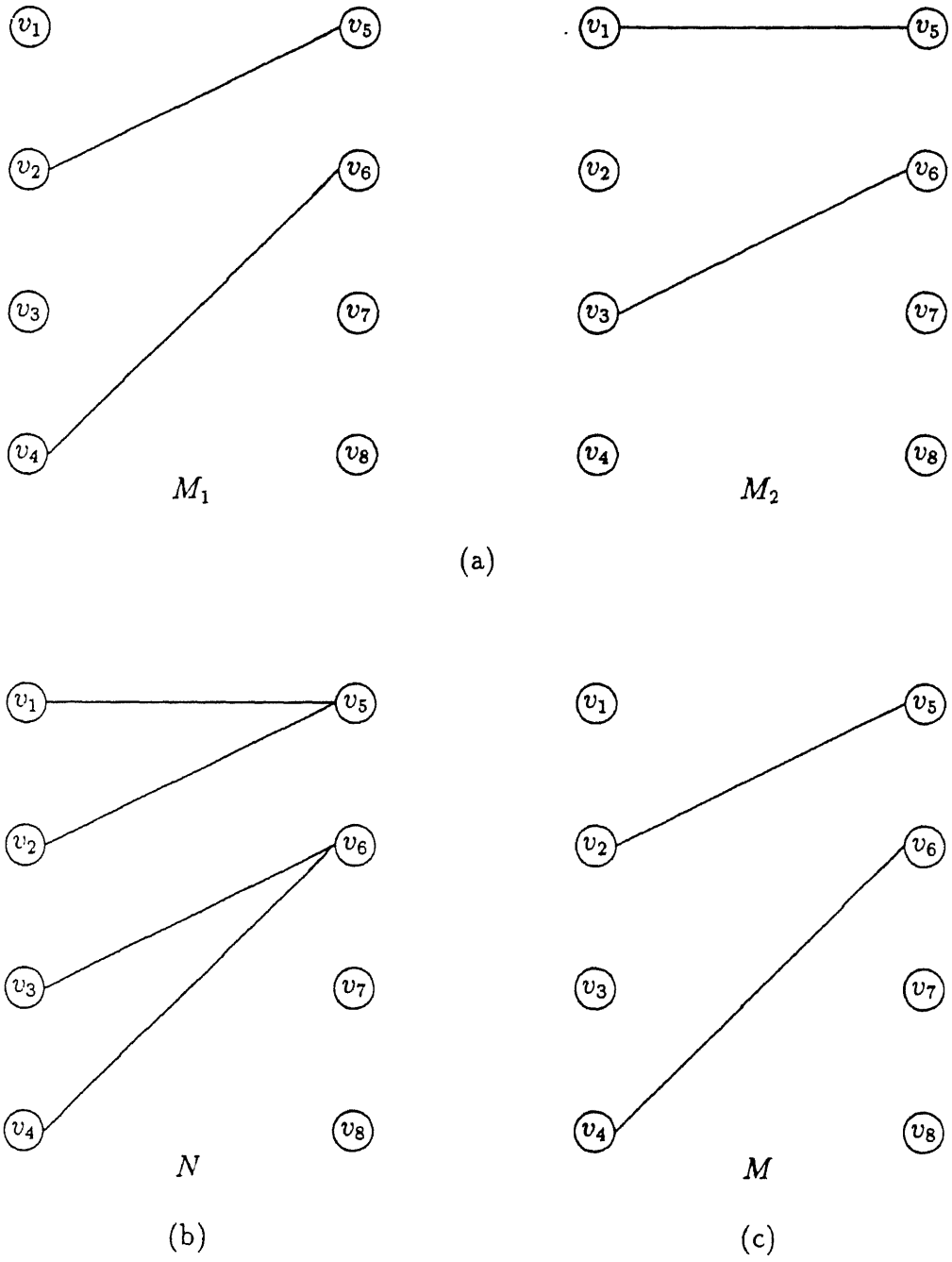


Figure 3.4: (a) M_1 and M_2 of the bipartite multigraph shown in figure 3.3 after step 1 of MD procedure. (b) After step 3. (c) Final result of M.

vertices are jointed if they are incident to the same edge. The Maximum Matching algorithm runs in $O(V^{\frac{1}{2}}E)$ time.

3.5 Modified Edge Coloring on Clos Network

This section describes the modified version of Gabow's Edge Coloring algorithm EC to find a minimal edge coloring of a regular bipartite multigraph.

A colored graph does not contain any vertex with edges of the same color incident to it. If Δ is the maximum degree of a regular bipartite multigraph, then there exists a minimal edge coloring which uses exactly Δ colors. Therefore an m-regular bipartite multigraph can be colored by using only m colors. As noted in Section 3.2, an (m, m, k) Clos network has m $k \times k$ switches at the center stage, and can be represented by a m-regular bipartite multigraph. So each color in a regular bipartite multigraph will represent a mapping of one center-stage switch in a Clos network. After the mapping in the center stage has been determined, it will be easy to set up the switches in the first and the last stages.

The EC algorithm uses a divide-and-conquer technique which divides graph G into two subgraphs, G_1 and G_2. To divide G, use the Euler Partition procedure described in Section 3.3 to get path set P, then put each edge of each path alternately in G_1 and G_2. Let Δ be the maximum degree in G; then the maximum degree in G_1, or G_2 is $\lfloor \frac{\Delta}{2} \rfloor$ or $\lceil \frac{\Delta}{2} \rceil$. IF Δ is even, EP can get a minimal coloring. When Δ is odd, and both subgraphs have maximum degree $\lceil \frac{\Delta}{2} \rceil$, the total number of colors of G is $\Delta + 1$ which is not minimal. To solve this problem, apply Hopcroft and Karp's Maximum Matching algorithm to find a maximum matching if Δ is odd. Then assign one color to the matching and delete it from G. After G has maximum degree $\Delta - 1$, the EP can be used to get G_1 and G_2. The modified EC algorithm is presented below:

PROCEDURE EC;

BEGIN

PROCEDURE REC(Δ);

COMMENT REC recursively colors a regular bipartite multigraph G that contains no vertices of degree 0. Δ is the maximum degree of a vertex. L_1 and L_2 are lists, local to REC, that store the edges of G;

BEGIN

1. **IF** Δ is odd **THEN**

BEGIN

COMMENT make M a matching that covers every vertex of degree Δ, and color the edges in M;

2. **IF** $\Delta = 1$ **THEN** $M := G$ **ELSE** MM;

3. Let e be a new color;

4. **FOR** each edge $e \in M$ **DO**

BEGIN

5. color(e):= c;

6. Delete e from G;

END;

END;

7. **EP**; **COMMENT** put the edge of G into an euler partition P;

8. **IF** P is not empty **THEN**

BEGIN

9. Make L_1 and L_2 empty lists;

10. **FOR** each path p in P **DO**

BEGIN

11. Let p be the sequence of edges $e_1,\ldots,e_r$;

12. **FOR** $i := 1$ **TO** r **DO**

13. **IF** i is odd **THEN** put e_i in L_1

ELSE put e_i in L_2;

END;

14. **FOR** $i := 1, 2$ **DO**

BEGIN

15. Let G be the multigraph consists of the edges in L_i and the vertices incident to them;

16. **REC**($\lfloor \frac{\Delta}{2} \rfloor$); **COMMENT** color the edges in L_i;

END;

END;

END REC;

17. Delete all vertices of degree 0 from G;

18. Let Δ be the maximum degree of a vertex;

19. **REC**(Δ);

END EC;

The modified EC algorithm finds a minimal edge coloring of a regular bipartite multigraph in $O(V^{\frac{1}{2}}E)$ time and $O(E)$ space. The run time for a Clos network is $O(k^{\frac{1}{2}}N \lg m)$.

Chapter 4

Routing on Beneš Networks

4.1 Introduction

In this chapter, a modified Tree-Connected Computer (TCC) and improved Parallel Looping Algorithm are presented. In Section 4.2, the well known Looping algorithm, which uses a single processor to determine the switch settings of a Beneš network, is described. In Section 4.3, the TCC used for PARALOOP[5] is discussed and a modified version of TCC, TTCC, with two processors is presented. The parallel looping algorithm PARALOOP is examined in Section 4.4. In Section 4.5, a bidirection parallel looping algorithm, BiPARALOOP, is presented. This algorithm, executed on the TTCC, can achieve a 50% speedup over the original TCC.

4.2 Looping Algorithm

The Looping Algorithm is used to set up a Beneš network. As mentioned in Chapter 2, a Beneš network is derived from a 3-stage $(2, 2, 2^r)$ Clos network by decomposing the two switches in the center stage into two 3-stage Clos networks and continuing the decomposition until each switch is a 2×2 cell. What the Looping Algorithm does is similar to the decomposition process. It decomposes

a permutation realized by a Beneš network into two subgroups (the mapping of the two center subnetworks) and determines the switch settings in the first, and the last stage. The Looping Algorithm is sequentially called to decompose the two center subnetworks and recursively called to set the center stage subnetwork until the subnetwork consists of one switch.

An $N \times N$ Beneš network has N inlets and N outlets numbered $0, 1, \ldots, N-1$. As defined in [23], a and b are called dual values or duals if $\lfloor \frac{a}{2} \rfloor = \lfloor \frac{b}{2} \rfloor$. $a = \overline{b}$ means that a is the dual of b. So, $D = \{(0,1),(2,3),\ldots,(N-2,N-1)\}$ is the set of duals. In the Looping Algorithm, the duals will be sent to or received from the opposite subnetwork $(C0, C1)$. The following procedure is the implementation of the Looping Algorithm on Beneš networks.

PROCEDURE LOOP(P, N);

BEGIN

$C0 := \Phi$; $C1 := \Phi$; $CYCLE := \Phi$; $a := 0$;

WHILE $\mid P \mid > 0$ DO

BEGIN

$$C0 := C0 + \left\{ \begin{pmatrix} a \text{ div } 2 \\ p(a) \text{ div } 2 \end{pmatrix} \right\};$$

$$P := P - \left\{ \begin{pmatrix} a \\ p(a) \end{pmatrix} \right\};$$

$$C1 := C1 + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)}) \text{ div } 2 \\ \overline{p(a)} \text{ div } 2 \end{pmatrix} \right\};$$

$$P := P + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)}) \\ \overline{p(a)} \end{pmatrix} \right\};$$

$$CYCLE := CYCLE + \{a\} + \{p^{-1}(\overline{p(a)})\};$$

IF $\overline{p^{-1}(\overline{p(a)})} \notin CYCLE$ **THEN**

$a := \overline{p^{-1}(\overline{p(a)})}$;

ELSE $a := x$ where $x \notin CYCLE$;

END;{ while }

IF | $C0$ | > 2 **THEN**

BEGIN

LOOP($C0, \frac{N}{2}$);

LOOP($C1, \frac{N}{2}$);

END;{ if }

END{ procedure }

Let P be the permutation realized by a Beneš network, and

$$P = \left\{ \begin{pmatrix} 0 \\ p(0) \end{pmatrix}, \begin{pmatrix} 1 \\ p(1) \end{pmatrix}, \ldots, \begin{pmatrix} N-1 \\ p(N-1) \end{pmatrix} \right\}.$$

The Looping Algorithm arbitrarily chooses one element of P such as $\begin{pmatrix} 0 \\ p(0) \end{pmatrix}$ and puts it into $C0$, the upper subnetwork. Since the dual values must go to a different subnetwork, the dual of $p(0)$, $\overline{p(0)}$ must go to $C1$. So, the element $\begin{pmatrix} a \\ \overline{p(0)} \end{pmatrix}$ is added to $C1$, the lower subnetwork. Before adding an element to $C0$ or $C1$, the 0, $p(0)$, $\overline{p(0)}$, and a have to shift right one bit of its binary representations. Since input a is sent to $C1$, $\overline{a}$ is routed to $C0$. The mapping $\begin{pmatrix} \overline{a} \\ p(\overline{a}) \end{pmatrix}$ is assigned to $C0$. Then follow the same procedure until P has been decomposed completely to get $C0$ and $C1$. If a permutation consists of more than one cycle, arbitrarily choose another element and start the same process when a cycle is finished.

The run time is $O(N)$ per iteration and there are $2 \lg N - 1$ stages, so the run time is $O(N \lg N)$ for the entire network. As mentioned in Section 2.4, one switch can be removed from the Beneš network to yield a Waksman network. This corresponds to arbitrarily choosing one element from permutation P.

To illustrate this procedure, consider the permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 7 & 3 & 2 & 6 & 1 & 0 & 4 \end{pmatrix}$$

realized by an 8×8 Beneš network. The right-shift function is skipped here. Arbitrarily choose $a = 0$. For the first round of the WHILE loop, $| P | = 8$ and $\begin{pmatrix} a \\ p(a) \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \end{pmatrix}$ is placed into $C0$. The dual of output 5 is 4, so $\begin{pmatrix} 7 \\ 4 \end{pmatrix}$ is added to $C1$. Now, the set $CYCLE$ contains (0, 7) and P remains $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 3 & 2 & 6 & 1 & 0 \end{pmatrix}$. The dual of input 7 is 6, which belongs to P, so $a = 6$. The second round of WHILE loop begins. At the end of the third round, the set $CYCLE$ contains (0, 7, 6, 5, 4, 1) that forms a cycle and P remains $\begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}$. Then arbitrarily choose $a = 2$ and start the forth round to get $C0 = \begin{pmatrix} 0 & 6 & 4 & 2 \\ 5 & 0 & 6 & 3 \end{pmatrix}$, $C1 = \begin{pmatrix} 7 & 5 & 1 & 3 \\ 4 & 1 & 7 & 2 \end{pmatrix}$. The switch settings of the first stage are easy to get by routing the inputs in $C0$ to the upper subnetwork and the inputs in $C1$ to lower subnetwork. The switch settings in the last stage can be determined by deriving the outputs in $C0$ from the upper subnetwork and the outputs in $C1$ from the lower subnetwork. The LOOP is recursively called to finish all the switch settings. The settings are shown in figure 4.1.

4.3 The Tree-Connected Computer

Two versions of the Tree-Connected Computer (TCC) are discussed in this section. They are used to run parallel Looping Algorithms.

In an ordinary TCC, all processors are connected as in a binary tree. Processor $P(x, y)$, $0 \leq x \leq (\lg N) - 2$ and $0 \leq y \leq 2^x - 1$, has two child processors $P(x + 1, 2y)$, $P(x + 1, 2y + 1)$. N is the size of the Beneš network. $P(0, 0)$ is the root processor and processors $P((\lg N) - 1, y)$, $0 \leq y \leq (\frac{N}{2}) - 1$, are the leaf processors which have no child processors. A TCC for the 8-input Beneš network is shown in figure 4.2.

A modified version of the TCC with two processors per node is called the TTCC. Nodes are connected as in a binary tree. Node $T(x, y)$, $0 \leq x \leq (\lg N) - 2$, $0 \leq$

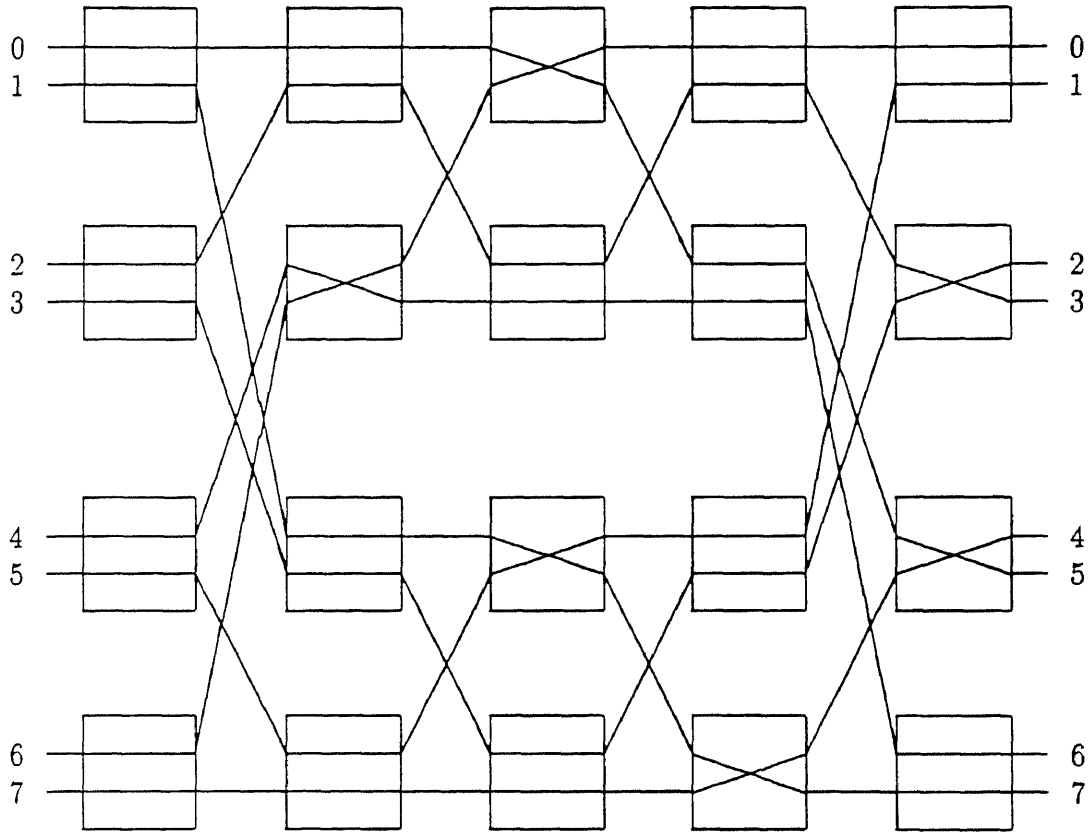


Figure 4.1: The switch settings of an 8×8 Beneš network for a permutation $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 7 & 3 & 2 & 6 & 1 & 0 & 4 \end{pmatrix}$

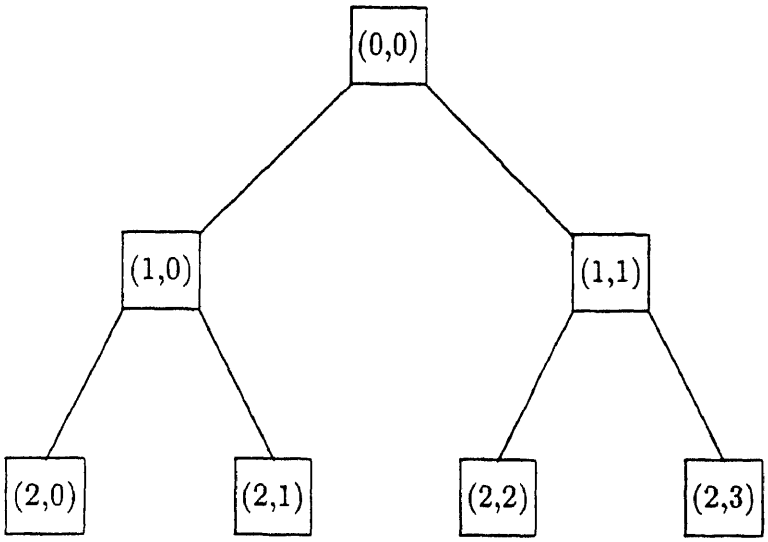


Figure 4.2: Tree-Connected Computer for $N = 8$

$y \leq 2^x - 1$, has two child nodes $T(x+1, 2y)$, $T(x+1, 2y+1)$. $T(0,0)$ is the root node and nodes $N((\lg N) - 1, y)$, $0 \leq y \leq (\frac{N}{2}) - 1$, are the leaf nodes which have no child nodes. Node $T(x,y)$ has two processors, $P_1(a_1, b_1)$ and $P_2(a_2, b_2)$, where $a_1 = a_2 = x$, $b_1 = 2y$, and $b_2 = 2y + 1$; P_1 and P_2 share a dual-port memory where the permutation is stored. A TTCC for the 8-input Beneš network is shown in figure 4.3.

The time needed to transfer data between processors or nodes is O(1). There are $O(N)$ connections in the TCC and TTCC. Each connection has $O(\lg N)$ width and the overall complexity is $O(N \lg N)$. Compare this to the complete interconnected computer of Nassimi and Sahni[21] that has $O(N^2 \lg N)$ complexity which is not feasible for a large value of N. The time needed to transfer data between processors or nodes is O(1).

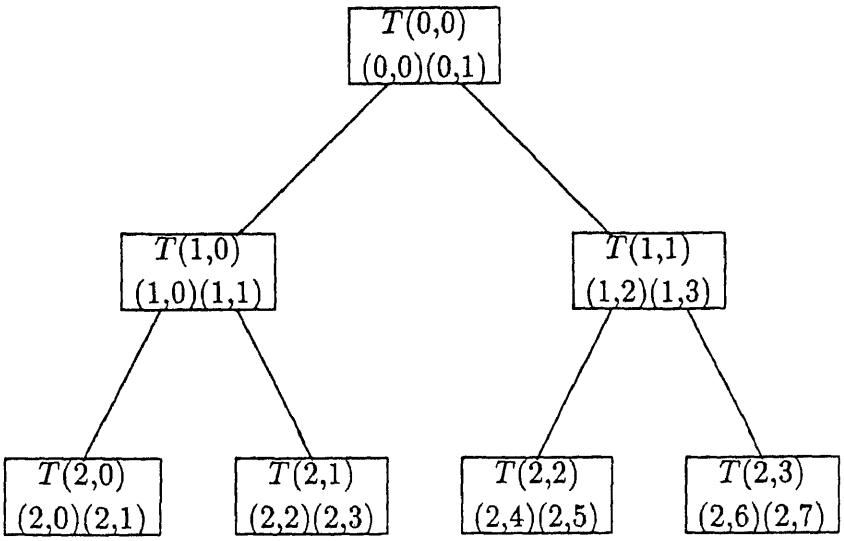


Figure 4.3: 2-processor TCC (TTCC) for $N = 8$

4.4 Parallel Looping Algorithm

As with the Looping Algorithm described in Section 4.2, there are two independent subsets, $C0$ and $C1$, after each iteration. So, the center subnetworks can be set in parallel by a TCC.

Let P be the permutation realized by an $N \times N$ Beneš network and

$$P = \left\{ \begin{pmatrix} 0 \\ p(0) \end{pmatrix}, \begin{pmatrix} 1 \\ p(1) \end{pmatrix}, \cdots, \begin{pmatrix} N-1 \\ p(N-1) \end{pmatrix} \right\}.$$

The implementation of parallel Looping Algorithm[5] is shown below:

PROCEDURE PARALOOP(P, N);

BEGIN

$C0 := \Phi$; $C1 := \Phi$; $CONSYMBOLS := \Phi$; $a := 0$;

WHILE $\mid P \mid > 0$ **DO**

BEGIN

$C0 := C0 + \left\{ \begin{pmatrix} a \bmod (N/2) \\ p(a) \bmod (N/2) \end{pmatrix} \right\}$;

$P := P - \left\{ \begin{pmatrix} a \\ p(a) \end{pmatrix} \right\}$;

$C1 := C1 + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)}) \bmod (N/2) \\ \overline{p(a)} \bmod (N/2) \end{pmatrix} \right\}$;

$P := P + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)}) \\ \overline{p(a)} \end{pmatrix} \right\}$;

IF $\overline{p^{-1}(\overline{p(a)})} \notin CONSYMBOLS$ **THEN**

BEGIN

$CONSYMBOLS := CONSYMBOLS + \{p^{-1}(\overline{p(a)})\}$;

$a := \overline{p^{-1}(\overline{p(a)})}$;

END;

ELSE

$a := x$ where $x \notin CONSYMBOLS$;

IF $\mid C0 \mid > 2$ **THEN**

PARABEGIN

PARALOOP$(C0, \frac{N}{2})$;

PARALOOP$(C1, \frac{N}{2})$;

END;{parabegin}

ELSE; {exit}

END{procedure}

It is assumed that the original permutation P with N elements is stored in the root processor. At the end of the WHILE loop, the mappings of the first and last stages are created and the permutations, $C0$, and $C1$, for the two inner subnetworks are formed. $C0$ and $C1$ are stored in the child processors of the root processor and computed simultaneously by running PARALOOP$(C0, \frac{N}{2})$ and

PARALOOP($C1, \frac{N}{2}$). The PARALOOP(P, N) is called recursively to process the subnetworks until the subnetwork consists of one switch. Figure 4.4 depicts how the procedure is executed. The C_{ij} in processor $p(i,j)$ corresponds to the permutation for the jth subnetwork at ith level. Processors $p(i,j)$, $0 \leq j \leq 2^i - 1$, are executing simultaneously to decompose its permutation for its child processors. P is the original permutation.

The run time of PARALOOP is O(N) and results a speedup of O($\lg N$) over the Looping Algorithm.

4.5 Bidirection Parallel Looping Algorithm

The Bidirection Parallel Looping Algorithm is run on the 2-processor Tree-Connected Computer (TTCC). It takes the features that dual values are sent to different subsets ($C0, C1$), and $C0$ and $C1$ are independent.

As mentioned in Section 2.2, a permutation consists of one or more than one cycle. The Looping Algorithm decomposes each cycle into $C0$ and $C1$ by starting from an arbitrarily chosed input-output mapping and tracing down the cycle to the other end of the cycle (the dual of the starting input). When the starting input is chosen and added to $C0$, its dual value must go to $C1$. So, we can use two processors to decompose each cycle concurrently.

BiPARALOOP(P, N) /* Processor $p1$ */

{

1. $C0 = \Phi$; $C1 = \Phi$; $a = 0$;
2. **for** (;;) /* infinite loop */

{

3. $C0 = C0 + \left\{ \begin{pmatrix} a/2 \\ p(a)/2 \end{pmatrix} \right\}$;

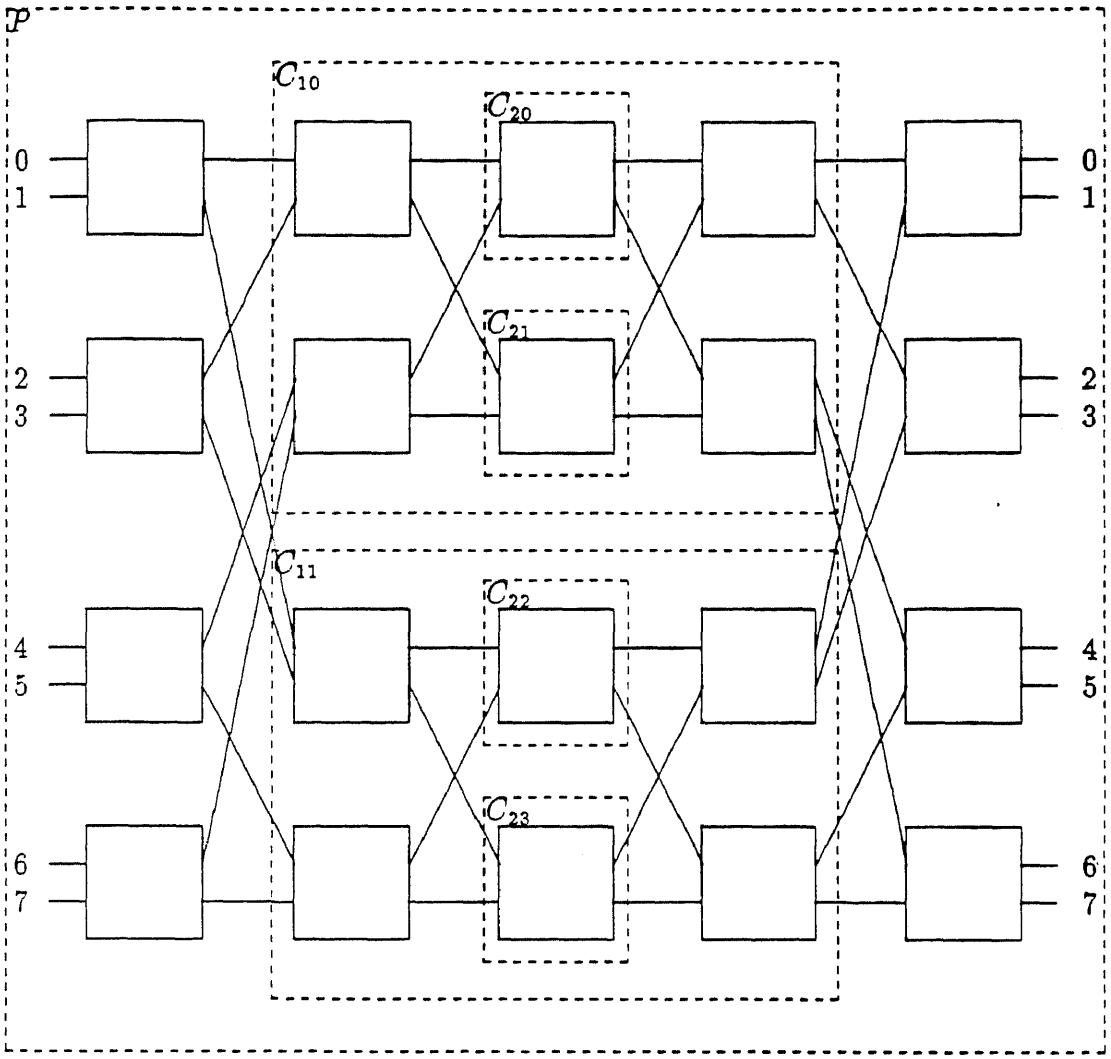


Figure 4.4: (a) 8-input Beneš network

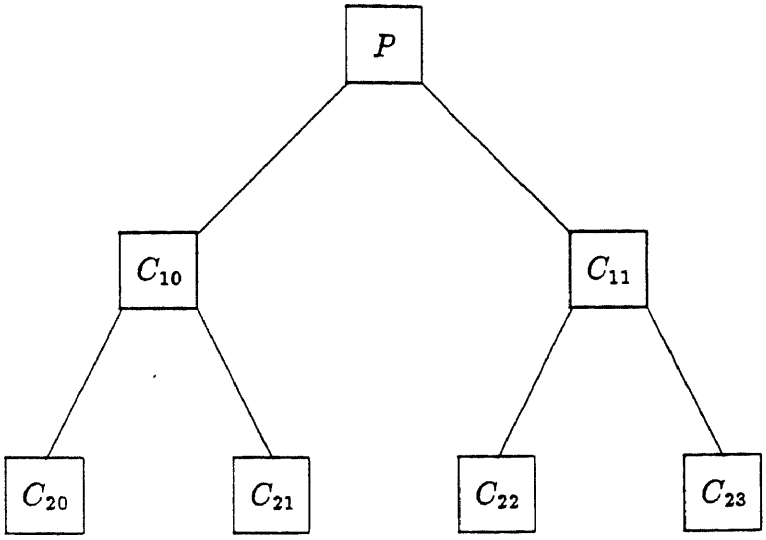


Figure 4.4: (b) Execution of the Parallel Looping Algorithm for $N = 8$

4. $P = P - \left\{ \begin{pmatrix} a \\ p(a) \end{pmatrix} \right\}$;

5. **if** $(p^{-1}(\overline{p(a)}) \in P)$

{

6. $C1 = C1 + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)})/2 \\ \overline{p(a)}/2 \end{pmatrix} \right\}$;

7. $P = P - \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)}) \\ \overline{p(a)} \end{pmatrix} \right\}$;

8. **if** $((a = p^{-1}(\overline{p(a)})) \notin P)$

{

9. **if** $(\mid P \mid > 0)$

{

10. $a =$ minimal even value in P;

}

11. **else break**;

}

}

12. **else if** $(\mid P \mid > 0)$

```
            {
13.             a = minimal even value in P;
            }
14.         else break;
        }
15.     if (| C0 | > 0)
        {           /* parallel begin */
16.         BiPARALOOP(C0, N/2);
17.         BiPARALOOP(C1, N/2);
        }           /* end of parallel */
18. }           /* end of procedure */
```

The differences of BiPARALOOP on $p2$ are listed below:

line 1: $C0 = \Phi; C1 = \Phi; a = 1;$

line 3: $C1 = C1 + \left\{ \begin{pmatrix} a/2 \\ p(a)/2 \end{pmatrix} \right\};$

line 6: $C0 = C0 + \left\{ \begin{pmatrix} p^{-1}(\overline{p(a)})/2 \\ \overline{p(a)}/2 \end{pmatrix} \right\};$

line 10: $a =$ minimal odd value in P;

line 13: $a =$ minimal odd value in P;

The original permutation P is stored in the root node, and $C0$ and $C1$ are stored and executed in its child nodes. To illustrate the BiPARALOOP, consider the switch setttings in figure 4.1. Figure 4.5 shows how the BiPARALOOP is executed. The bold lines are set by $p1$. The other lines are routed by $p2$.

In Parallel Looping Algorithm, the WHILE loop runs $\frac{N}{2}$ times. In BIPARALOOP, the FOR loop executes $\frac{N}{4}$ times. So this algorithm can result in a 50%

speedup over the Parallel Looping Algorithm.

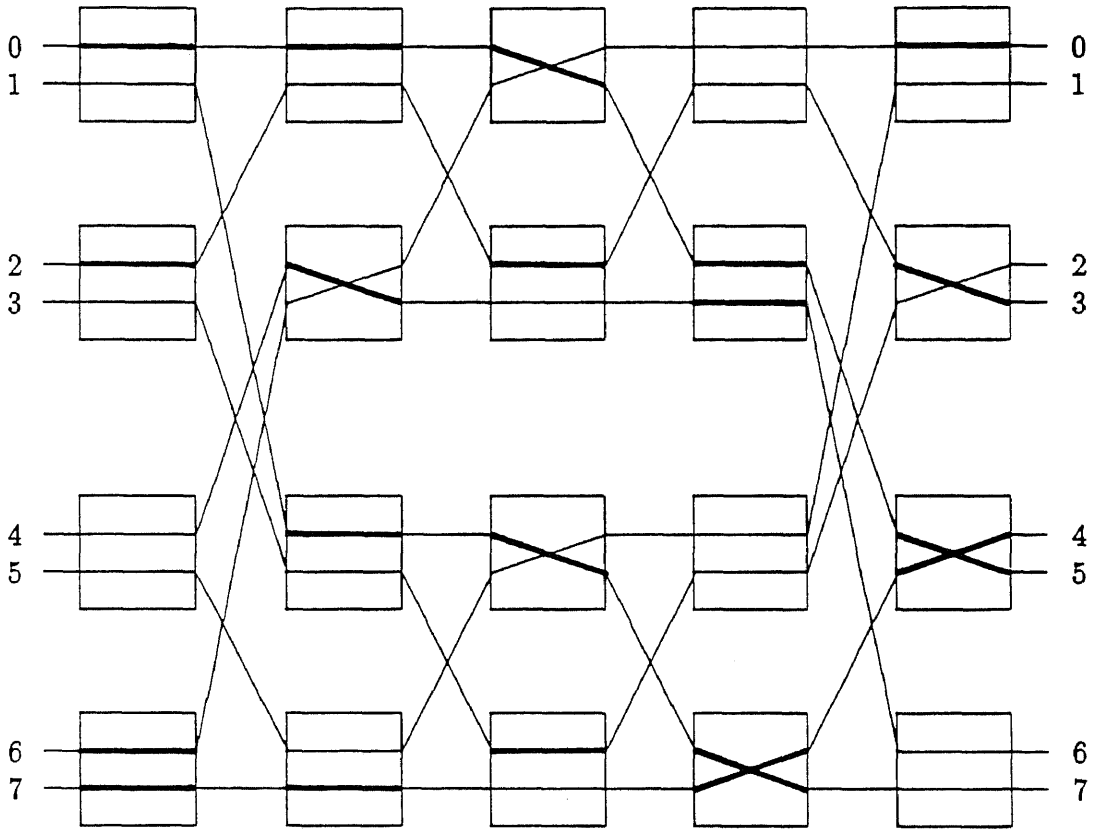


Figure 4.5: The switch settings of $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 7 & 3 & 2 & 6 & 1 & 0 & 4 \end{pmatrix}$ via BiPARALOOP algorithm

Chapter 5

Fault-Tolerant Clos Network

5.1 Introduction

An ordinary (m, n, k) Clos network has some degree of fault tolerance capability if $n > m$[20]. But this exists only if the fault occurs in the center stage. A Fault-Tolerant Clos network which provides alternate paths to bypass a fault at any stage is presented in this chapter. In Section 5.2, the fault model and design objectives are described. The construction of the FTC as well as the extra hardware elements are presented in Section 5.3. In Section 5.4, fault recovery is described. A comparison to the ordinary Clos network and Nassar's FTC[20] is discussed in Section 5.5.

5.2 Fault Model and Design Objectives

A fault model defines the possible physical failures of the fault tolerant network. It also implies its fault tolerance capabilities. The fault model is defined below:

1. Any switch can fail.

2. Any interstage link can fail.

3. Additional hardware can not fail.

The Fault-Tolerant Clos network has full recovery, so the network can achieve the prefault connectivity. The faults are assumed to occur independently and the faulty elements can not be used again.

The design objectives are:

1. single switch fault tolerance in each stage,
2. single link fault tolerance in each interstage link,
3. low switch and link complexity,
4. the same switch size of an ordinary Clos network,
5. the same routing algorithm.

The first three objectives are easily met by other fault tolerance networks[17,20] but the FTC presented here can achieve all five objectives.

5.3 Design of the Fault-Tolerant Clos Network

Recall from Chapters 2 and 3 that a 3-stage (m, n, k) Clos network with size $N = m \times k$ must have k $m \times n$ switches in the first stage, n $k \times k$ switches in the center stage, and k $n \times m$ switches in the last stage. The edge coloring algorithm determines the mapping of the center stage first and then sets the first and last stages to complete the input-output mappings through the 3-stage network. The FTC is used to completely recover the connections after faults occur.

A Clos network with $n > m$ has some degree of fault tolerance[20]. The FTC presented here uses an ordinary (m, m, k) Clos network. The idea of the new FTC is to isolate the faulty switches by bypassing the inputs to and receiving from the standby switches. The FTC is 3-fault robust since it can completely recover the

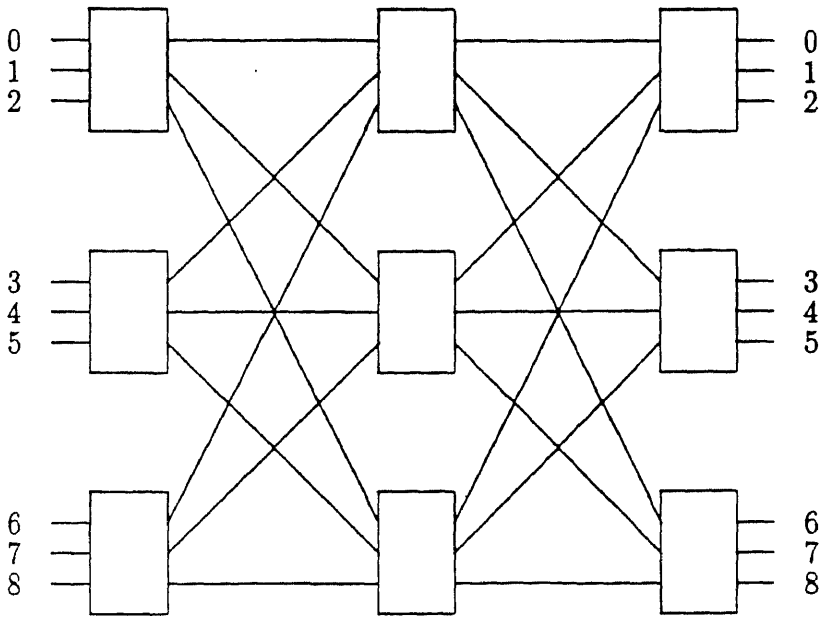


Figure 5.1: A (3,3,3) Clos network

network when one fault occurs in each stage. A (3,3,3) Clos network is shown in figure 5.1 and its FTC network is shown in figure 5.2.

Before describing the construction of the FTC, we have to describe the extra 2×2 switches added to the inputs and outputs of each switch. The 2×2 switch shown in figure 5.3 is used to do the function of isolating. Figure 5.4 shows the four states of the 2×2 switch.

$IN0$ is connected to the ordinary outlet of the previous stage and $OUT0$ is connected to the inlet of the next stage. $IN1$ and $OUT1$ are the extra links connected to the standby switch. Figure 5.5 shows an example of these connections.

Each 2×2 switch has four states controlled by $IN0$ and $OUT0$. These four states are described below:

1. **State (0,0)** : When P is non-faulty and N is non-faulty, the linkage is the same

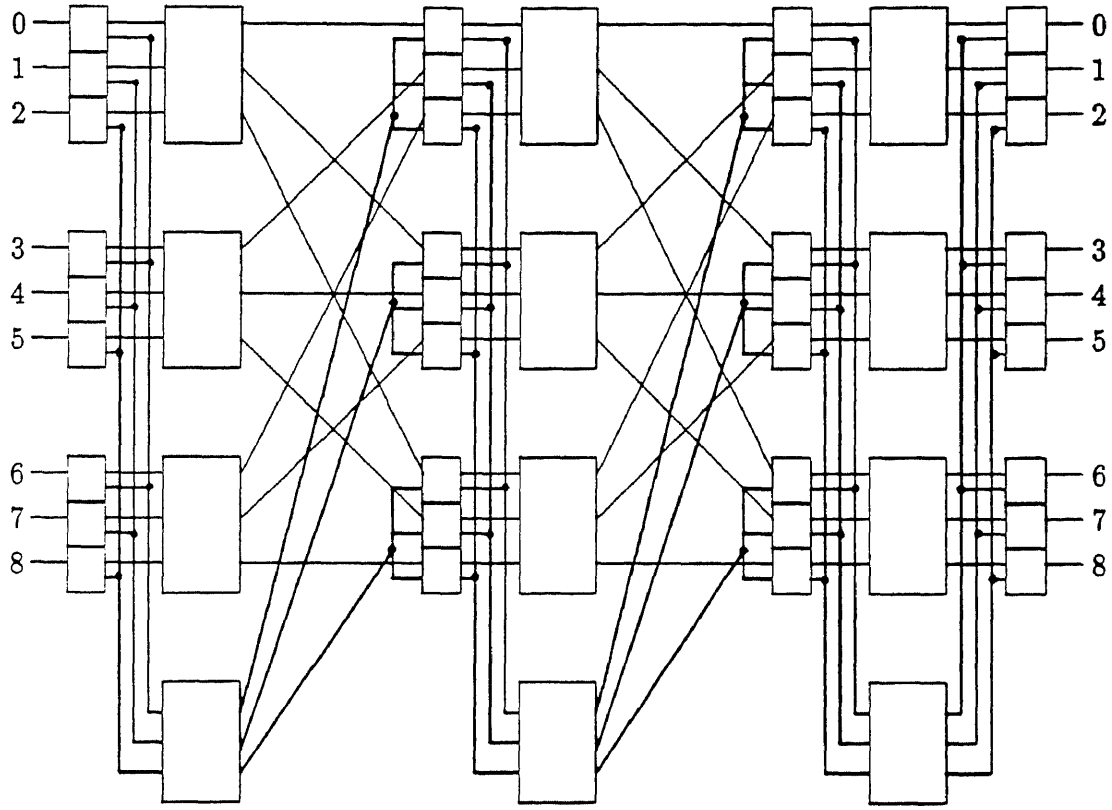


Figure 5.2: A (3,3,3) Fault-Tolerant Clos network

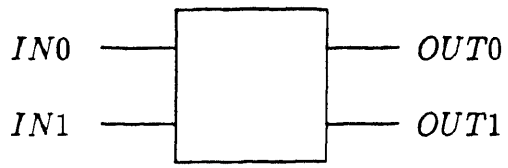


Figure 5.3: A 2×2 switch

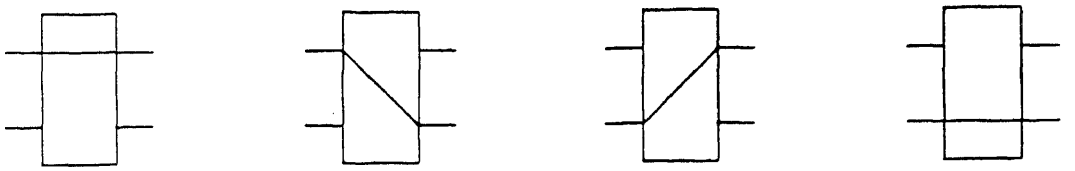


Figure 5.4: The four states of a 2×2 switch

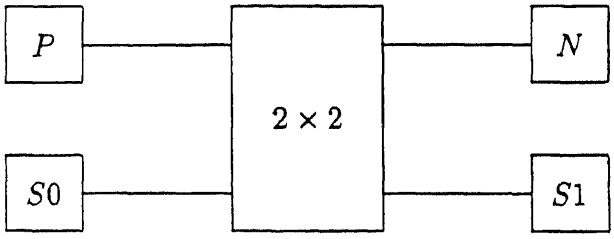


Figure 5.5: A connection between 2×2 switch and stages. P stands for the previous stage, N means the next stage, $S0$ and $S1$ are the standby switches.

as the ordinary network. See figure 5.4.a.

2. **State (0,1) :** When P is non-faulty and N is faulty, the inputs of switch N will be redirected to the standby switch by connecting $IN0$ to $OUT1$. See figure 5.4.b.

3. **State (1,0) :** When P is faulty and N is non-faulty, the inputs of switch N will be linked to the outputs of the standby switch in the previous stage by connecting $IN1$ to $OUT0$. See figure 5.4.c.

4. **State (1,1) :** When P and N are faulty, the inputs of the faulty switch N in the next stage will be redirected to the standby switch and linked to the outputs of the standby switch in the previous stage by connecting $IN1$ to $OUT1$. See figure 5.4.d.

The construction of an (m, m, k) Fault-Tolerant Clos network, shown in figure 5.6, is formed by the following processes:

1. Start with a 3-stage (m, m, k) Clos network of size N.

2. Assume there is one null stage before the first stage and one after the last stage, then put N 2×2 switches between every two stages.

3. Put one extra $m \times m$ switch in the first stage and the last stage, and one $k \times k$ switch in the center stage.

4. Connect $IN0$, $OUT0$ of each 2×2 switch to stages like the connections described in figure 5.5.

5. For the first three columns of N 2×2 switches, connect each $OUT1$ of the ith 2×2 switch of each ordinary switch to the ith input of the standby switch.

6. For the fourth column of N 2×2 switches, connect each $IN1$ of the ith 2×2 switch of each ordinary switch to the ith output of the standby switch.

7. Ground all the $IN1$ in the first column.

8. Ground all the $OUT1$ in the fourth column.

9. Connect the ith output of the standby switch at the first stage to k $IN1$'s at the second column of 2×2 switch which are connected to the ith ordinary switch in the center stage.

10. Connect the ith output of the standby switch at the second stage to N $IN1$'s at the third column of 2×2 switch which are connected to the ith ordinary switch in the last stage.

5.4 Fault Recovery

The ability of the FTC is to find an alternate path to complete the mappings of the prefault permutation at any given time. At most one faulty switch can occur at each stage at a time. When one or more faults occur, the following three steps can recover from the faults.

1. Change the state of the extra 2×2 switches.

2. Apply the edge coloring algorithm to the same permutation.

3. Set the switches.

In step 3, it is assumed that the control unit knows which switches are faulty and sets the appropriate standby switch. If no fault occurs, only steps 2 and 3 have to be performed. In this situation, the routing process is the same as described in

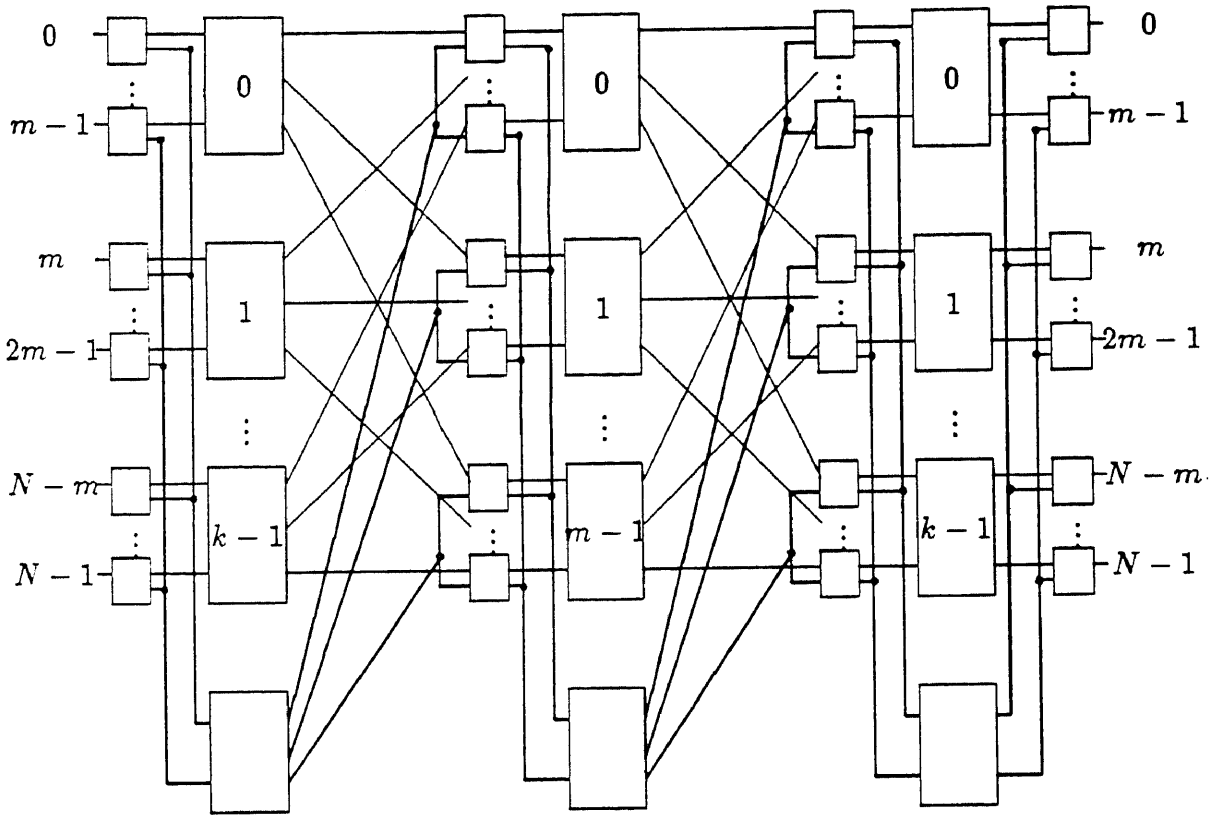


Figure 5.6: An (m, m, k) FTC

Chapter 3. So only one extra operation, step 1, is needed to reconfigure the FTC if faults occur.

As mentioned in Section 5.3, the state of the 2×2 switch is controlled by $IN0$ and $OUT0$. When a fault occurs at a switch, the $IN0$ and $OUT0$ which connect to the faulty switch will change the state of the 2×2 switch. The faulty switch must notify the control unit so the control unit can transfer the setting from the faulty switch to the standby switch. The fault detection and location techniques are beyond the range of this thesis.

To illustrate the fault recovery, consider the permutation

$$P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 2 & 3 & 6 & 8 & 1 & 4 & 0 & 7 \end{pmatrix}$$

realized in an ordinary Clos network and an FTC when faults occur. Figure 5.7 shows the switch settings in a non-faulty Clos network. Let $X(i,j)$ is the faulty switch, the jth switch at the ith stage. Assume switches $X(0,1)$, $X(1,2)$, $X(2,2)$ fail at the same time; figure 5.8 depicts this situation.

The link failure can be viewed as a failure of the switches which connect to the failed link directly.

5.5 Comparison

Since the FTC presented in the chapter does not change the size of the permutation and uses the same edge coloring algorithm, it has the same run time complexity, $O(kN \lg m)$. However, it does introduce some propagation delay due to the extra 2×2 switches.

The FTC presented in this chapter offers some improvements compared to the Nassar's FTC[20] on a permutation of size N.

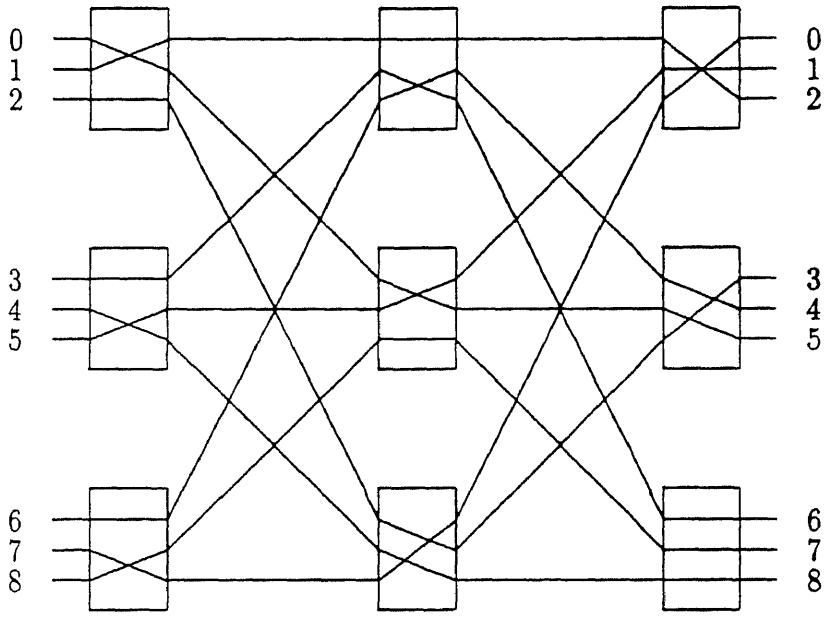


Figure 5.7: The switch settings of permutation $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 2 & 3 & 6 & 8 & 1 & 4 & 0 & 7 \end{pmatrix}$ realized by a (3,3,3) Clos network.

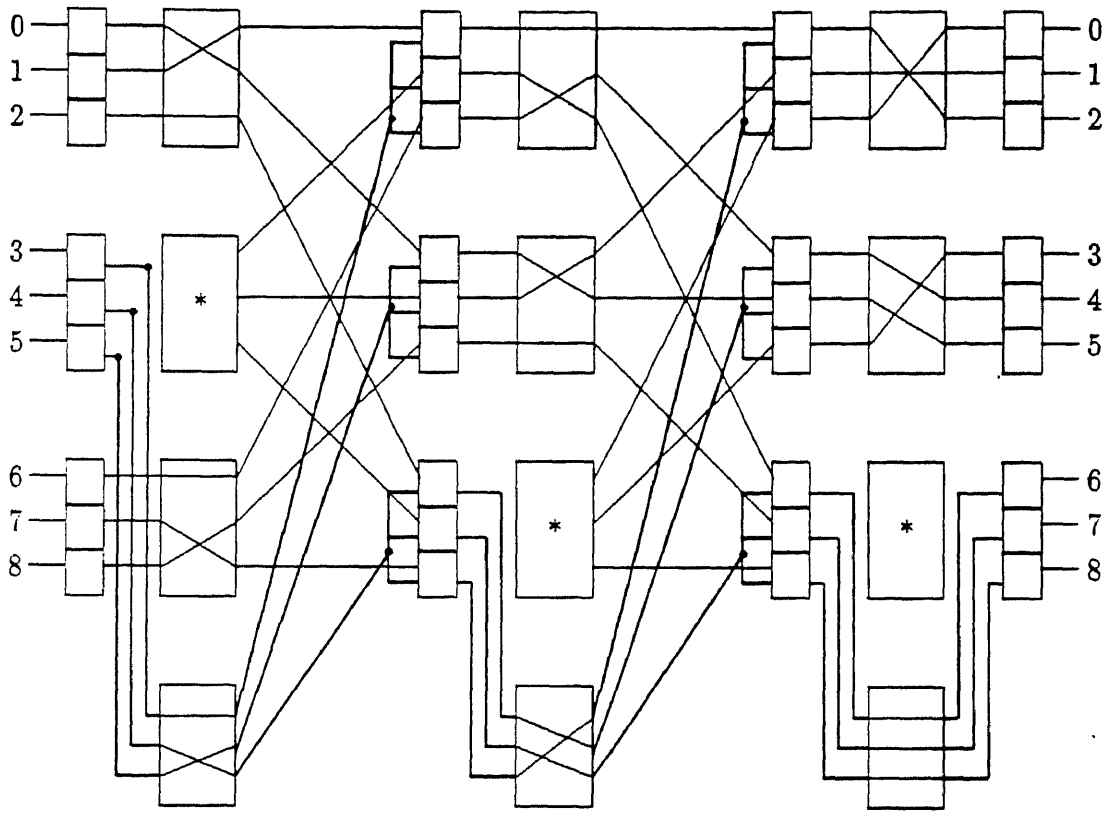


Figure 5.8: The switch setting of permutation $P = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 2 & 3 & 6 & 8 & 1 & 4 & 0 & 7 \end{pmatrix}$ realized by a (3,3,3) FTC. The star, *, marks the faulty switches.

1. There is only one extra operation to reconfigure the FTC when faults occur at any stage—change the state of the 2×2 switch. Nassar's FTC needs three additional operations to reconfigure the FTC if faults occur at the outer stages—change the state of the multiplexers and demultiplexers, terminal relabelling and permutation translation.

2. The ordinary Clos network needs two different sizes of switches, $m \times m$ and $k \times k$. The FTC presented here needs two extra $m \times m$ switches, one $k \times k$ switch, and $4N$ 2×2 switches to provide alternate paths. But Nassar's FTC uses five different sizes of switch, $m \times (m+1)$, $(k+1) \times (k+1)$, $(m+1) \times m$, multiplexer and demultiplexer.

3. There is no permutation translation or size change. But Nassar's FTC expands the permutation size from N to $N + k$. This requires more computing time for the routing algorithm as well as storage space for the permutation. Also the time to do the permutation translation when an outer stage switch fails increases.

Chapter 6

Conclusions

Improved routing algorithms for Clos networks and Beneš networks as well as fault tolerant Clos networks have been developed.

A permutation with size N realized by a 3-stage (m, m, k) Clos network with size $N = m \times k$ is represented by an m-regular bipartite multigraph. A modified Edge Coloring Algorithm that consists of Euler Partition and Maximum Matching procedure is used to decompose the m-regular bipartite multigraph into m subgraphs. Each subgraph represents the mapping of a switch in the center stage of the Clos network. The original Edge Coloring algorithm contains matching procedure MD developed by Gabow. MD procedure uses Hopcroft and Karp's Maximum Matching algorithm to find two maximum matchings, M_1 and M_2, that cover all vertices of maximum degree in V_1 and V_2. For a regular bipartite multigraph, each vertex has the same degree. So, the M_1 or M_2 covers all of the graph in a multigraph and is the maximum matching needed in EC algorithm.

The Looping algorithm and Parallel Looping algorithm (PARALOOP) for the Beneš network were examined. The Bidirectional Parallel Looping algorithm (Bi-PARALOOP) was presented. The Looping algorithm is executed in a single processor; the Parallel Looping algorithm is executed in a Tree-Connected Computer

(TCC) to route the two independent inner subnetworks simultaneously instead of sequentially in the Looping algorithm. The BiPARALOOP algorithm is executed in a modified TCC, TTCC, with two processors per node to run each cycle of a permutation concurrently instead of sequentially in the TCC with one processor per node. The BIPARALOOP on TTCC can gain 50% speedup over the PARALOOP on TCC.

The Tree-Connected Computer is well matched to the parallel routing algorithm for the Beneš network. The Euler partition procedure in edge coloring algorithms decomposes a graph into two subgraphs. How the TCC can be utilized to execute the Euler Partition to determine the switch settings in the center stage of Clos network will be an interesting research topic.

The Fault-Tolerant Clos network presented in this thesis considerably increases the reliability of the network by using little additional hardware and does not significantly degrade the performance under both normal and faulty conditions. Through the isolation of the faulty switches, we can redirect the input-output connections to alternate paths. It needs only one additional operation to reconfigure the network; the Nassar's FTC needs at most three extra operations. There is no change in the size of switches as well as the size of the permutation in the new FTC. Nassar's FTC expands the permutation size from N to $N + K$ and needs a permutation translation when faults detected. Those will use more space and increase the computing time. This new FTC uses three different sizes of switch but Nassar's FTC uses five different sizes of switch.

Bibliography

[1] Agrawal, D. "Testing and Fault Tolerance of Multistage Interconnection Networks," *Computer*, April, 1982. pp. 41-53.

[2] Baer, J. L. "Multiprocessing Systems," *IEEE Transactions on Computers*, vol. C-25, no. 12. December, 1976. pp. 1271-1277.

[3] Beneš, V. E. *Mathematical Theory of Connecting Networks and Telephone Traffic*, New York, Academic Press, 1965.

[4] Berge, C. *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[5] Carpinelli, J. D. *Interconnection Networks: Improved Routing Methods for Clos and Beneš Networks*, Ph.D. Thesis, Rensselaer Polytechnic Institue, Troy, NY, August, 1987.

[6] Clos, C. "A Study of Non-Blocking Switching Networks," *Bell Systems Technical Journal*, vol. 32, no. 2. March, 1953. pp. 406-424.

[7] Enslow, P. H. "Multiprocessor Organization," *Computer Surveys*, vol. 9, no. 3. March, 1977. pp. 103-129.

[8] Feng, Tse-Yun. "A Survey of Interconnection Networks," *Computer*, vol. 14, no. 12. December, 1981. pp. 12-27.

[9] Gabow, H. N. "Using Euler Partition to Edge Color Bipartite Multigraphs," *International Journal of Computer and Information Sciences*, vol. 5, no. 4, 1976. pp. 345-355.

[10] Hwang, K., and Briggs, F. A. *Computer Architecture and Parallel Processing*, New York, Mc-Graw Hill, 1984.

[11] Hwang, K., Su, S. P., and Ni, L. M. "Vector Computer Architecture and Processing Techniques," *Advanced in Computers*, vol. 20, Yovits, ed., New York, Academic Press, 1981, pp. 115-197.

[12] Hopcraf, J. E., and Karp, R. M. "An $n^{\frac{5}{2}}$ Algorithm for Maximum Machings in Bipartite Graphs," *SIAM Journal on Computing*, vol. 2, no. 4. December, 1973. pp. 225-231.

[13] Jones, A. K., and Genringer, E. F. "Cm* Multiprocessor Project: A Research Review," *Technical Report CMU-CS-80-131*, Carnegie-Mellon University, July, 1980.

[14] Kober, R., and Kuznia, C. "SMS—A Multiprocessor Architecture for High-Speed Numerical Calculations," *Proceedings of International Conference on Parallel Processing*, 1978, pp. 18-23.

[15] Kogge, P. M. *Architecture of Pipelined Computers*, New York, McGraw-Hill, 1981.

[16] Kuck, D. J. *The Structure of Computers and Computations*, vol. 1, New York, Wiley, 1978.

[17] Kumar, V. P., and Reddy, S. M. "Augumented Shuffle-Exchange Multistage Interconnection Networks," *Computer*, June, 1987. pp. 30-40.

[18] Lawrie, D. H. "Access and Alignment of Data in an Array Processor," *IEEE Transactions on Computers*, vol. C-24, no. 12. December, 1975. pp. 1145- 1155.

[19] Li, H. F. "Scheduling Trees in Parallel Pipelined Processings Enviroments," *IEEE Transactions on Computers*, November, 1977. pp. 1101-1112.

[20] Nassar, H. *Fault-Tolerant Interconnection Networks for Multiprocessor Systems*, Ph.D. Thesis, New Jersey Institute of Technology, Newark, NJ, 1989.

[21] Nassimi, D., and Sahni, S. "Parallel Algorithms to Set Up the Beneš Permutation Network," *IEEE Transactions on Computers*, vol. C-31, no. 2. February, 1982. pp. 148-154.

[22] ———. "A Self-Routing Beneš Network and Parallel Permutation Algorithms," *IEEE Transactions on Computers*, vol. C-30, no. 5. May, 1981. pp. 332-340.

[23] Opferman, D. C., and N. T. Tsao-Wu. "On a Class of Rearrangeable Switching Networks, Part I: Control Algorithm," *Bell Systems Technical Journal*, vol. 50, no. 5. May, 1971. pp. 1579-1600.

[24] Paul, G. "Large- Scale Vector/Array Processors," *IBM Research Report*, RC 7306, September, 1978.

[25] Siegel, H. J. "A Model of SIMD Machines and a Comparison of Various Interconnection Networks," *IEEE Transactions on Computers*, vol. C-28, no. 12. December, 1979. pp. 907-917.

[26] ———. "Interconnection Networks for SIMD Machines," *IEEE Transactions on Computers*, June, 1979b. pp. 57-65.

[27] Stone, H. S. "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, vol. C-20, no. 2. Feburary, 1971. pp. 153-161.

[28] Waksman, A. "A Permutation Network," *Journal of the ACM*, vol. 15, no. 1. January, 1968. pp. 159- 163.

[29] Wu, C. L., and Feng, T. Y. "The University of the Shuffle-Exchange Network," *IEEE Transactions on Computers*, vol. C-30, no. 5. May, 1981. pp. 324-332.

[30] _____. "On a Class of Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. C-29, no. 8. August, 1980. pp. 694-702.

[31] Vizing, V. "On an Estimate of the Chromatic Class of a p-graph," *Diskret. Analiz.*, No. 3, 1964, pp. 25-30.