# ABSTRACT

Title of Dissertation: Networks for Fast and Efficient Unicast and Multicast Communications

Ching-Yi Lee, Doctor of Philosophy, 1992

Dissertation directed by: A. Yavuz Oruç, Associate Professor
Department of Electrical Engineering

This dissertation presents new results on networks for high-speed unicast and multicast communications which play key roles in communication networks and parallel computer systems. Specifically, (1) we present fast parallel algorithms for routing any one-to-one assignment over Beneš networks, (2) we propose new multicasting networks that can efficiently realize any one-to-many assignments, and (3) we give an explicit construction of linear-size expanders with very large expansion coefficients.

Our parallel routing algorithms for Beneš networks are realized on two different topologies. Using these algorithms, we show that any unicast assignment that involves $O(k)$ pairs of inputs and outputs can be routed through an $n$-input Beneš network in $O(\log^2 k + \lg n)$ time without pipelining and $O(\lg k)$ time with pipelining if the topology is complete, and in $O(\lg^4 k + \lg^2 k \lg n)$ time without

pipelining and $O(\lg^3 k + \lg k \lg n)$ time with pipelining if the topology is extended perfect shuffle. These improve the best-known routing time complexities of parallel algorithms of Lev et al, and Nassimi and Sahni by a factor of $O(\lg n)$.

Our multicasting network uses a very simple self-routing scheme which requires no separate computer model for routing. Including the routing cost, it can be constructed with $O(n \lg^2 n)$ bit-level constant fanin logic gates, $O(\lg^2 n)$ bit-level depth, and can realize any multicast assignment in $O(\lg^3 n)$ bit-level time. These complexities match or are better than those of multicasting networks with the same cost that were reported in the literature. In addition to its attractive routing scheme, our multicasting network is input-initiated and can pipeline multicast assignments through itself. With pipelining, the average routing time for $O(\lg^2 n)$ multicast assignments can be reduced to $O(\lg n)$ which is the best among those of the multicasting networks previously reported in the literature.

Our linear-size expanders are explicitly constructed by following a traditional design and analysis technique. We construct a family of linear-size with density 33 and expansion coefficient 0.868. This expansion coefficient is the largest among the linear-size expanders that were similarly constructed. Using these expanders, we also report a family of explicitly constructed superconcentrators with density 208.

# Networks for Fast and Efficient Unicast and Multicast Communications

by

Ching-Yi Lee

Dissertation submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1992

Advisory Committee:

Associate Professor A. Yavuz Oruç, Chairman/Advisor
Professor Robert Newcomb
Associate Professor Prakash Narayan
Assistant Professor Bernard Menezes
Professor Stephen S. Kudla

# Dedication

To

my dear wife Heng-Tzu Lee

for

her endless love and support

# Acknowledgements

I am extremely grateful to my advisor Dr. A. Yavuz Oruç for his advice and guidance during the work of this dissertation. His perspective, enthusiasm, patience and kindness have inspired and benefited me very much. I would like to give my greatest respect and acknowledgment to him here.

I am also grateful to Dr. Robert Newcomb, Dr. Prakash Narayan, Dr. Bernard Menezes and Dr. Stephen Kudla for their serving on my committee and giving corrections of this dissertation.

Special thanks to my parents and parents in-law for providing me mental and financial support. To them, I owe a lot of gratitude.

Finally, I would like to thank my wife Heng-Tzu. Without her, this dissertation would not have been possible.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

CHAPTER
ONE

# INTRODUCTION

In communication networks as well as parallel computer systems, interconnection networks are the key for good performance. Interconnection networks serve as switching fabrics which form the backbone of a communication network and provide networking and switching services to realize switching patterns that arise in the network [Clo53, Ben65, Pip82, Hui90]. In parallel computer systems, interconnection networks provide physical frames for interprocessor and processor-memory communications [Fen81, Bhu87, DeC89, Sie90]. For todays rapid progress of broadband communication networks and high-speed parallel computers, high performance interconnection networks are very critical [Tob90, MZ90, RB90].

Among all switching patterns in communication networks and interconnection requests in parallel computers, one-to-one or unicast assignments, and one-to-many or multicast assignments are the most general. Interconnection networks that can realize unicast and multicast assignments are called, respectively, uni-

casting and multicasting networks in this dissertation, and they have received much attention in the literature [Clo53, Pau62, Ben65, Ofm65, Can71, MJ72, Hwa72, Tho78, Pip82, Oru87a, Oru87b, Lea88, Tur88, Hui90, YM91a]. Traditional designs of these networks typically focus on reducing the hardware cost, but the recent research has also focused on reducing the time needed to realize unicast and multicast assignments. This trend is enforced partly because of the fast advances in semiconductor, integrated circuit (IC) and very large-scale integration (VLSI) technologies, and partly because of the fast intercommunication requirements in communication networks [Tob90, MZ90, RB90] and parallel computer systems [Bhu87, DeC89, Sie90].

In this dissertation, we consider three issues, namely the *unicasting problems*, the *multicasting problems* and the $(\alpha, \beta)$-*expanding problems*. These issues are critical with regard to unicasting and multicasting networks, and have attracted much attention in the literature. Several results on these problems have been reported. First, we give a description of these issues and a survey of key results.

## 1.1 The Unicasting Problems

Unicast assignments are the most typical among all assignments that arise in communication networks and parallel computers, and they appeared in the earliest stage of interconnection network literature [Clo53, Pau62, Ben62, Ben64, Ben65]. The well-known Beneš networks [Ben65] is a unicasting network with $O(n \lg n)$ cost and $O(\lg n)$ depth. Batcher's networks [Bat68] and those that were introduced by Koppelman and Oruç [KO90] and by Douglass and Oruç [DO90] can also be used as unicasting networks, but they require $O(n \lg^3 n)$ cost and $O(\lg^3 n)$ routing time in bit level.

Recently, Jan and Oruç [JO91] proposed a radix permutation network which has $O(n \lg^2 n)$ bit-level cost and can realize any complete unicast or permutation assignment in $O(\lg^2 n \lg \lg n)$ bit-level time. Later, Lu and Oruç [LO92] modified that network and reduced the cost to $O(n \lg n)$. More recently, Chien and Oruç [CO92] reported a binary sorting network which can also be used for permutation switching. That network has $O(n \lg n)$ cost, $O(\lg^3 n)$ depth and $O(\lg^3 n)$ routing time, all in bit level. Although these networks have impressive cost and routing time, it is difficult and inefficient to route incomplete unicast assignments on them mainly because they are designed to route permutation assignments.

Because of the $O(n \lg n)$ cost and $O(\lg n)$ depth, the Beneš network is considered as an attractive unicasting network, and has received much attention in interconnection network literature [Ben65, LPV81, NS81, NS82a, Lee87]. In a way, this network can be considered the forerunner of most multistage interconnection networks that have been extensively studied and used in some real parallel computer systems [Sea89, Kea91].

One problem regarding the Beneš network that remains to have a satisfactory solution is its routing. Many routing algorithms have been reported extending from the $O(n \lg n)$ time looping procedure of Waksman [Wak68], and Opferman and Tsao-Wu [OTW71] to the $O(\lg^2 n)$ time parallel algorithms of Lev et al [LPV81], and Nassimi and Sahni [NS82a]. Other routing algorithms for the Beneš network include matching and edge-coloring schemes [Hwa83, Car86, CO88].

The looping algorithm was developed with the realization that there are exactly two subnetworks in the center stage for the network's inputs to connect to its

outputs, and with the added constraint that no two inputs can be connected to two outputs through the same subnetwork in the center stage unless those outputs belong to different switches in the last stage. Given a permutation assignment from the inputs onto the outputs, one gets around this constraint simply by looping between the switches in the first stage and those in the last stage, and assigning the paths in an alternate fashion to the subnetworks in the center stage. Given that there are $n$ inputs to route, it takes the looping algorithm $O(n)$ time to set the switches in a 3-stage Beneš network, and if the same algorithm is applied recursively to the center-stage subnetworks then all the switches in the fully decomposed Beneš network can be set in $O(n \lg n)$ time.

The routing time of the Beneš network can be reduced in several ways. The most obvious approach is to use a binary tree to set the subnetworks in the center stage in parallel as was done in [CO88]. The root processor of the tree sets the switches in the first level, its children set the switches in the next level, and so on. It is easy to see that the routing time, $T_n$, of this setting scheme obeys the recurrence $T_n = T_{n/2} + O(n)$ which implies that $T_n = O(n)$. The routing time can be reduced further by introducing parallelism into the setting of switches in each level. This can be done by dividing the outputs into equivalence classes such that two outputs will be in the same class if and only if they must be routed through the same center-stage subnetwork. Once the equivalence classes are decided, the switches in the first and last stages can be set in parallel.

The parallel algorithm of Nassimi and Sahni [NS82a] is based on this discovery, and the complexity of their algorithm depends on the complexity of the par-

allel computer model and the number of processors available. Their algorithm takes $O(\lg^2 n)$ time on a completely interconnected $n$-processor computer, and $O(\lg^4 n)$ time on a perfect-shuffle interconnected $n$-processor computer. Lev et al [LPV81] provided similar parallel algorithms in a more general framework by using edge-coloring schemes. Assuming a parallel computer with conflict free access between $O(n)$ processors and $O(n)$ memory elements, their algorithm also takes $O(\lg^2 n)$ time.

While these parallel algorithms are fast, their time complexities are still higher than the $O(\lg n)$ depth of the Beneš network. Furthermore, these algorithms can only route complete or permutation assignments. In case of incomplete assignments where some inputs are idle, they cannot be used unless the idle inputs are given dummy outputs. This takes additional time and may render these algorithms inefficient. For those assignments which involve fewer pairs of inputs and outputs, especially for the sparse-connection assignments which involve only $O(k)$ pairs for $k << n$, it takes more time to assign dummy outputs to the idle inputs. Besides, one does not gain any time to route incomplete assignments since they are treated as complete assignments in these algorithms.

## 1.2 The Multicasting Problems

Multicast assignments are more general than unicast assignments, and they include the unicast assignments. Several communication services such as teleconferencing, cable TV broadcasting, local and metropolitan area networking require multicasting of voice, video and other signals [Tob90, MZ90, RB90]. Multicasting networks, which realize multicast assignments, have been extensively studied in the literature [Ofm65, MJ72, Hwa72, Tho78, Lea88, Tur88,

YM91a, YM91b], and they were obtained by one of three approaches.

The first approach is based on extensions of 3-stage Clos networks [Clo53]. The constructions given in [MJ72, Hwa72, YM91a] are all based on this concept. The main problem with this approach is that to obtain asymptotically minimum cost multicasting networks, one must recursively decompose these extended Clos networks. However, the decomposed networks require complex routing schemes for unicast assignments (see [LPV81]), and realizing multicast assignments on such networks is even more complex. We must also mention the 2-stage broadcast network introduced in [RH85]. While the construction given there is based on a 2-stage decomposition, its routing amounts to finding matchings in bipartite graphs, and thus, is as complex as the routing of 3-stage networks [Hwa83]. Besides, it is not known whether this 2-stage broadcast network, when recursively decomposed, leads to an asymptotically minimum cost network.

The second approach, which was introduced by Ofman [Ofm65], and also used in [Tho78, Tur88, Lea88, Lee88, YM91b], decomposes the design of a multicasting network into two or more subproblems. Typically, the first problem involves the construction of a network, called a *generalizer* in Thompson's work, to generate the requisite copies of the packets to be multicast. Once the copies are made, then a unicasting network is used to route these copies to their destinations. The multicasting networks obtained by this approach have several drawbacks. First, the sorter-based generalizers such as the one used in the *Starlite network* [HK84] have excessive switching cost; the sorter stage requires $O(n \lg^2 n)$ comparators for $n$ packets. Other generalizers such as the one described in [Tur88] use $O(n \lg n)$ switches, but they are blocking networks, and

are routed by adhoc conflict resolution schemes that may lead to uneven delays among packets. Furthermore, routing of copied packets through the unicasting network stage is as complex as in the first approach unless blocking is allowed.

The third approach is based on recursive decompositions of multicasting networks. Unlike the second approach, the rationale behind this approach is to generate the requisite copies of packets while they are being routed to their destinations, rather than use a separate copy network. This approach was introduced by Nassimi and Sahni [NS82b] who constructed an $n$-input multicasting network with $O(kn^{1+1/k} \lg n)$ $2 \times 2$ crossbar switches, $O(k \lg n)$ depth, and $O(k \lg n)$ routing time in word level, for any $k, 1 \leq k \leq \lg n$. (All the definitions will be formally given in Chapter 2.) While the design of this multicasting network is based on elementary switches, its routing relies on a parallel computer model with cube or perfect shuffle topology. In addition, the realization of multicast assignments under this routing scheme requires sorting, and this means that incomplete assignments can be realized only if the unused inputs are assigned to some dummy outputs. Besides, paths on this multicasting network to realize a multicast assignment are established from the output side to the input side, i.e., it is output-initiated, which is not practical as it is often the case that the inputs initiate the requests.

Finally, we should note that multicasting networks can also be obtained by modifying the permutation networks reported in [JO91, CO92]. However, as in the unicasting problems, it is difficult to route incomplete multicast assignments over such networks.

## 1.3 The $(\alpha, \beta)$-Expanding Problems

In [JO91], Jan and Oruç also constructed permutation networks by using a special type of networks, called *linear-size expanders*. But, they did not provide any construction for the expanders. In interconnection network literature, expanders has been proven to be useful in constructing other types of interconnection networks as well, and have been studied extensively [Pin73, Chu78, Bas81, Pip82, Mar73, GG81, AM85, Alo86, NAM87, JM87].

Informally, an $n$-input linear-size expander is a directed bipartite graph with $O(n)$ edges such that any set of less than $\alpha n$ inputs will be adjacent to some set of more than $\beta n$ outputs, where $\alpha \leq \beta$. This is called the $(\alpha, \beta)$-*expanding* property of the expander in Bassalygo's work [BP74, Bas81]. Expanders are important in that they are used in constructing concentrators [Chu78], superconcentrators [Chu78, Pip77], generalizers [Chu78, Pip78a], unicasting networks [Pip82]. strictly and wide-sense nonblocking networks [BP74, Chu78, Pip78b, Pip82, FFP88] and sorting networks [AKS83].

The existence of linear-size expanders was first discovered by Pinsker [Pin73]. By using Konig's theorem on matchings in graphs, and combinatorial arguments, he was able to show that linear-size expanders exist. Based on combinatorial or probabilistic arguments, other results proving the existence of linear-size expanders were also given in [Chu78, Bas81, Pip82]. By using group representations, Margulis [Mar73] first provided an explicit construction for linear-size expanders, but he could not specify the expansion coefficient which is an important parameter to measure a linear-size expander. Subsequently, Gabber and Galil [GG81] modified Margulis' construction and used harmonic analysis to specify the expansion coefficient (it is $(2-\sqrt{3})/4$) of their expander

graphs. More recently, Alon and Milman [AM85, Alo86] used a matrix formulation to express the expansion coefficient as a function of the second largest eigenvalue of a matrix representation of that expander. Using this formulation, Alon et al. [AM85, Alo86, NAM87] and Jimbo and Maruoka [JM87] obtained expander graphs with expansion coefficients 0.412 and 0.466. All these constructions are based on Margulis' expander design, and provide $n$-input expanders where $n$ is a square of any integer.

While these expanders have very low densities, their expansion coefficients are quite small, which may restrict their use in constructing other interconnection networks. Alon [Alo86] suggested a method to construct expanders with large coefficients from expanders with small expansion coefficients. However, the drawback of this approach is that the density of the resulting expander is too high.

## 1.4 Contributions

For the problems described in the previous sections, the contributions of this dissertation are three-fold. First, we design a parallel algorithm for routing any one-to-one (complete or incomplete) assignment over Beneš networks. To route an assignment involving $O(k)$ pairs of inputs and outputs, our algorithm takes $O(\log^2 k + \lg n)$ time without pipelining and $O(\lg k)$ time with pipelining on a complete connection topology, and it takes $O(\lg^4 k + \lg^2 k \lg n)$ time without pipelining and $O(\lg^3 k + \lg k \lg n)$ time with pipelining on a perfect-shuffle-like connection topology. In case of sparse assignments (i.e., when $k << n$), our algorithm achieves the optimum ($O(\lg n)$) routing time for Beneš networks. Compared with the time complexities of $O(\lg^2 n)$ on the complete connection

topology and $O(\lg^4 n)$ on the perfect-shuffle connection topology of the parallel algorithms in [NS82a, LPV81], our algorithm has an $O(\lg n)$ factor of time gain over those algorithms. These results provide the best-known time complexities for routing one-to-one assignments on Beneš networks.

Second, we construct a multicasting network which has a very simple self-routing scheme. It uses very simple circuits consisting of constant fanin logic gates to set the switches in each stage. Including the routing cost, our multicasting network can be constructed with $O(n \lg^2 n)$ bit-level constant fanin logic gates and $O(\lg^2 n)$ bit-level depth, and can realize any multicast (complete or incomplete) assignment in $O(\lg^3 n)$ bit-level time. These complexities match or are better than those of multicasting networks that were reported in the literature. Besides, unlike the multicasting networks reported in [NS82b] which are output-initiated and rely on parallel computer models for routing assignments, our multicasting network is input-initiated (which is more realistic) and requires no other routing computer. In addition, multicast assignments can be pipelined over our network. With pipelining, the routing time can be reduced to $O(\lg n)$, which is the best-known routing time among those of the multicasting networks ever reported.

Third, we explore and explicitly construct a family of linear-size expanders with large expansion coefficients. Our expanders are constructed similarly as those in [Mar73, GG81, AM85, Alo86, NAM87, JM87]. The density is 33, and the expansion coefficient is 0.868, which is larger than the largest expansion coefficient reported in [Mar73, GG81, AM85, Alo86, NAM87, JM87]. Using these expanders, we also construct a family of bounded concentrators with density 25.5 and a family of superconcentrators with density 208.

## 1.5 Dissertation Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we give some basic facts and definitions that will be used throughout this dissertation. We also review some well-known multistage interconnection networks that are relevant to the results of this dissertation. Chapter 3 presents our parallel routing algorithm for Beneš networks. Chapter 4 displays the construction of our multicasting network. Chapter 5 gives our explicit design of linear-size expanders, bounded concentrators and superconcentrators. Chapter 6 is a conclusion which summarizes the results of this dissertation and suggests some problems for future research work.

CHAPTER
TWO

# BASIC CONCEPTS

In this chapter, we state some basic facts and definitions which will be used throughout the dissertation. We also give a brief description of some well-known interconnection networks which are relevant to the work of this dissertation.

## 2.1 Basic Definitions and Facts

### 2.1.1 Networks

An interconnection network with $n$ inputs and $q$ outputs is denoted as an $(n,q)$-network and formally defined as follows.

**Definition 2.1** *An $(n,q)$-network is a directed acyclic graph with $n$ distinguished source vertices, called inputs, $q$ distinguished sink vertices, called outputs, some internal vertices, and some edges which connect these vertices. If $n = q$, an $(n,q)$-network is abbreviated as an $n$-network.* ||

Pictorially, an $n$-network will be depicted as a rectangle, and designated its inputs and outputs by arrows on its sides. The inputs and outputs will be normally numbered 0 through $n-1$ and 0 through $q-1$ from top to bottom (or left to right in some cases) as shown in Figure 1 unless otherwise stated.

An *assignment* for a network is a pairing of its inputs with its outputs such that each output appears in at most one pair. An input is called *busy* in an assignment if it is paired with some output(s), and *idle* otherwise. Similarly, an output is called *busy* in an assignment if it is paired with an input, and called *idle* otherwise. An assignment consisting of $k$ pairs will be called a $k$-assignment. An assignment is called one-to-one or *unicast* if each input appears in at most one pair, and is called one-to-many or *multicast*, otherwise. A *permutation assignment* for an $n$-network is a unicast $n$-assignment. A network is said to *realize* an assignment if, for each pair $(a, b)$ in the assignment, a path of vertices can be formed from input $a$ to output $b$ by determining the edges between adjacent vertices in the network with the constraint that the paths for no two pairs $(a, b)$ and $(c, d)$ overlap at the same edge unless $a = c$.

Paths in a network will be established by specifying some routing information at its inputs. It is assumed that each input holds its own destination information unless otherwise stated. It is also assumed that the routing information for each input is accompanied by some binary coded message that is to be routed from that input to those output(s) specified in the routing information. A message and routing information combined together will be termed a *packet*, and the routing part of a packet will be called the *packet header*. The message part of a packet can have an arbitrary number of bits, but it will be assumed that the content and length of the message(s) entering a network have no impact

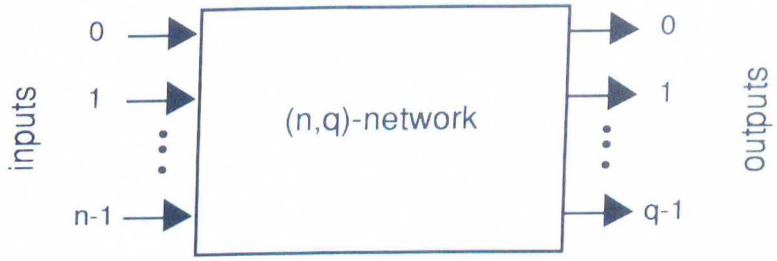



**Figure 1**: Graphical representation of an $(n, q)$-network.

on its routing performance. For $n$ inputs, the number of bits in the routing information will vary between 1 and $2n - 2$ for the networks described in this dissertation. If a packet at input $i$ uses $m + 1$ bits of routing information, its packet header will be denoted as $(r_i, d^i_{m-1}, \ldots, d^i_1, d^i_0)$. The first bit $r_i$, to be called the connecting bit, specifies whether input $i$ is paired with some output, and hence determines if input $i$ is busy or idle. The remaining bits, i.e., $(d^i_{m-1}, \ldots, d^i_1, d^i_0)$, carry the address information of output(s) to which the packet at input $i$ will be sent (i.e., with which input $i$ is paired).

Two of the most useful networks are the *unicasting network* and the *multicasting network* which are formally defined as follows.

**Definition 2.2** *An $n$-network is called a unicasting $n$-network if it can realize all unicast assignments between its inputs and outputs.* ||

**Definition 2.3** *An $n$-network is called a multicasting $n$-network if it can realize all multicast assignments between its inputs and outputs.* ||

For an $n$-network, because there are $n!/(n - k)!$ $k$-assignments between its inputs and outputs, $1 \leq k \leq n$, the total number of unicast assignments which it can realize is $\sum_{k=1}^{n} n!/(n - k)! = O(2^{n \lg n})$. For a multicasting $n$-network,

each of its $n$ output may be paired with none or any one of its $n$ inputs, the total number of multicast assignments that can be realized by the multicasting $n$-network is $(n+1)^n$.

Some other networks are also useful and can be used to construct unicasting and multicasting networks.

**Definition 2.4** *An $(n,q)$-network, $q \leq n$, is called an $(n,q)$-concentrator if, for any $k, 1 \leq k \leq q$, and for any $k$ of its $n$ inputs, it can realize at least one among the $k!$ unicast $k$-assignment that pairs those $k$ inputs with the first $k$ of its $q$ outputs.* ||

**Remark 2.1** *From this definition, it is clear that every $(n,q)$-concentrator is an $(n,q')$-concentrator for any $q' \leq q$. Therefore, we will only give an $n$-concentrator construction in this dissertation. If an $(n,q)$-concentrator is needed, for $q < n$, it can be implemented by chopping away the last $n-q$ outputs of an $n$-concentrator.* ||

**Definition 2.5** *An $n$-network is called an $n$-superconcentrator if, for any $k, 1 \leq k \leq n$, and for any $k$ of its $n$ inputs and any $k$ of its $n$ outputs, it can realize a unicast $k$-assignment that pairs those $k$ inputs with those $k$ outputs.*
*If an $n$-superconcentrator has no more than $dn$ edges where $d$ is a constant independent of $n$, then it is called a linear-size $n$-superconcentrator with density $d$, and will be denoted as an $(n;d)$-superconcentrator.* ||

**Definition 2.6** *An $(n,q)$-network is called an $(n,q;k,\alpha)$-bounded concentrator if it is bipartite, i.e., it has no internal vertex, and has no more than $kn$*

*edges such that, for any subset $X$ of inputs with $|X| \leq \alpha n$, $|\Gamma_X| \geq |X|$, where $\Gamma_X$ is the set of outputs adjacent to the inputs in $X$.* ||

**Definition 2.7** *An $n$-network is called an $(n; k, c, \alpha)$-expander if it is bipartite and has no more than $kn$ edges such that, for any subset $X$ of inputs with $|X| \leq \alpha n$, $|\Gamma_X| \geq [1 + c(1 - |X|/n)]|X|$, where $\Gamma_X$ is the set of outputs adjacent to the inputs in $X$. $c$ is called the expansion coefficient.* ||

### 2.1.2 Performance Measures

Three parameters are typically used to measure the performance of a network, and they can be recursively defined and calculated when the network is recursively constructed. The *cost* of a network is the amount of hardware needed to construct the network. The *depth* of a network is the maximum length of a path from an input to an output. The *routing time* of a network is the maximum amount of time to decide a path from an input to an output.

All these parameters can be expressed in two levels of complexities, namely the word level and the bit level. In word level, the cost of an $n$-network is measured in terms of digital circuits with no more than $O(\lg n)$ constant fanin (independent of $n$) logic gates, the unit of the depth is a constant fanin logic gate, and the unit of the routing time is the delay through a digital circuit which has at most $O(\lg \lg n)$ constant fanin logic gates along a path from any input to any output of the digital circuit. However, it is assumed that the hardware needed to construct an $n$-network can be implemented by constant fanin logic gates, and then the performance parameters can be measured more precisely in bit level. For a network, in bit-level, the cost is the total number of

constant fanin logic gates used in the network, the depth is the largest number of constant fanin logic gates along a path from an input to an output, and the routing time is the maximum amount of time needed to set all the switches in the network for an assignment between the inputs and outputs of the network.

### 2.1.3 Fixed Connections

For network constructions, some fixed one-to-one connections between two sets of $n$ points are useful. They specify connections from a stage of switches of a network to another stage of switches of the network. In this subsection, we describe some fixed connections that will be used later.

Let $X$ and $Y$ be two disjoint sets of points, each with $n$ points, where $n$ is a power of 2. Let $(b^i_{\lg n-1}, \ldots, b^i_1, b^i_0)$ is the binary representation of point $i$ with $b^i_{\lg n-1}$ being the most significant bit, $0 \leq i \leq n-1$.

**Definition 2.8** *For $k = 2^l$, $2 \leq l \leq \lg n$, the k-shuffle connection, $\sigma_k$, from $X$ to $Y$ is a one-to-one mapping from $X$ onto $Y$, given by $\sigma_k(b^i_{\lg n-1}, \ldots, b^i_1, b^i_0) = (b^i_{\lg n-1}, \ldots, b^i_{\lg k}, b^i_{\lg k-2}, \ldots, b^i_0, b^i_{\lg k-1})$, $0 \leq i \leq n-1$.* ||

**Definition 2.9** *For $k = 2^l$, $2 \leq l \leq \lg n$, the k-banyan connection, $\beta_k$, from $X$ to $Y$ is a one-to-one mapping from $X$ onto $Y$, given by $\beta_k(b^i_{\lg n-1}, \ldots, b^i_1, b^i_0) = (b^i_{\lg n-1}, \ldots, b^i_{\lg k}, b^i_0, b^i_{\lg k-2}, \ldots, b^i_1, b^i_{\lg k-1})$, $0 \leq i \leq n-1$.* ||

**Definition 2.10** *The reverse connection, $\rho_n$, from $X$ to $Y$ is a one-to-one mapping from $X$ onto $Y$, given by $\rho_n(b^i_{\lg n-1}, \ldots, b^i_1, b^i_0) = (b^i_0, b^i_1, \ldots, b^i_{\lg n-1})$, $0 \leq i \leq n-1$.* ||

For example, Figure 2 shows these fixed connections on two sets of $n$ points for $n = 16$.

## 2.2 Some Well-Known Networks

In this section, we describe some well-known networks that are relevant to the subjects of this dissertation and will be used in the following chapters.

### 2.2.1 Self-Routing Networks

A network is called *self-routing* if each of its switch can be set by using only the routing information of the incoming packets in its inputs without reference to other switches. Here, we describe three self-routing $n$-networks. Each of these $n$-network has $\lg n$ stages, numbered stage 0, stage 1, ..., stage $\lg n - 1$ from left to right, and each of their stages consists of $n/2$ $2 \times 2$ switches. Every $2 \times 2$ switch can be set in two ways: either *through* state where the two inputs are connected straight through to the two outputs, or *cross* state where the two inputs are connected to opposite outputs.

An $n$-network consisting of $\lg n$ stages is called a *baseline* $n$-network if the fixed connection from stage $i$ to stage $i + 1$ is an $(n/2^i)$-shuffle connection, $i = 0, 1, \ldots, \lg n - 2$. An $n$-network consisting of $\lg n$ stages is called a *banyan* $n$-network if the fixed connection from the inputs to stage 0 is an $n$-shuffle connection and the fixed connection from stage $i$ to stage $i + 1$ is an $(n/2^i)$-banyan connection, $i = 0, 1, \ldots, \lg n - 2$. An $n$-network consisting of $\lg n$ stages is called an *omega* $n$-network if each of the fixed connections from the inputs to stage 0 and from stage $i$ to stage $i + 1$, $i = 0, 1, \ldots, \lg n - 2$, is an $n$-shuffle connection. If the input side and output side of each network are reversed, then

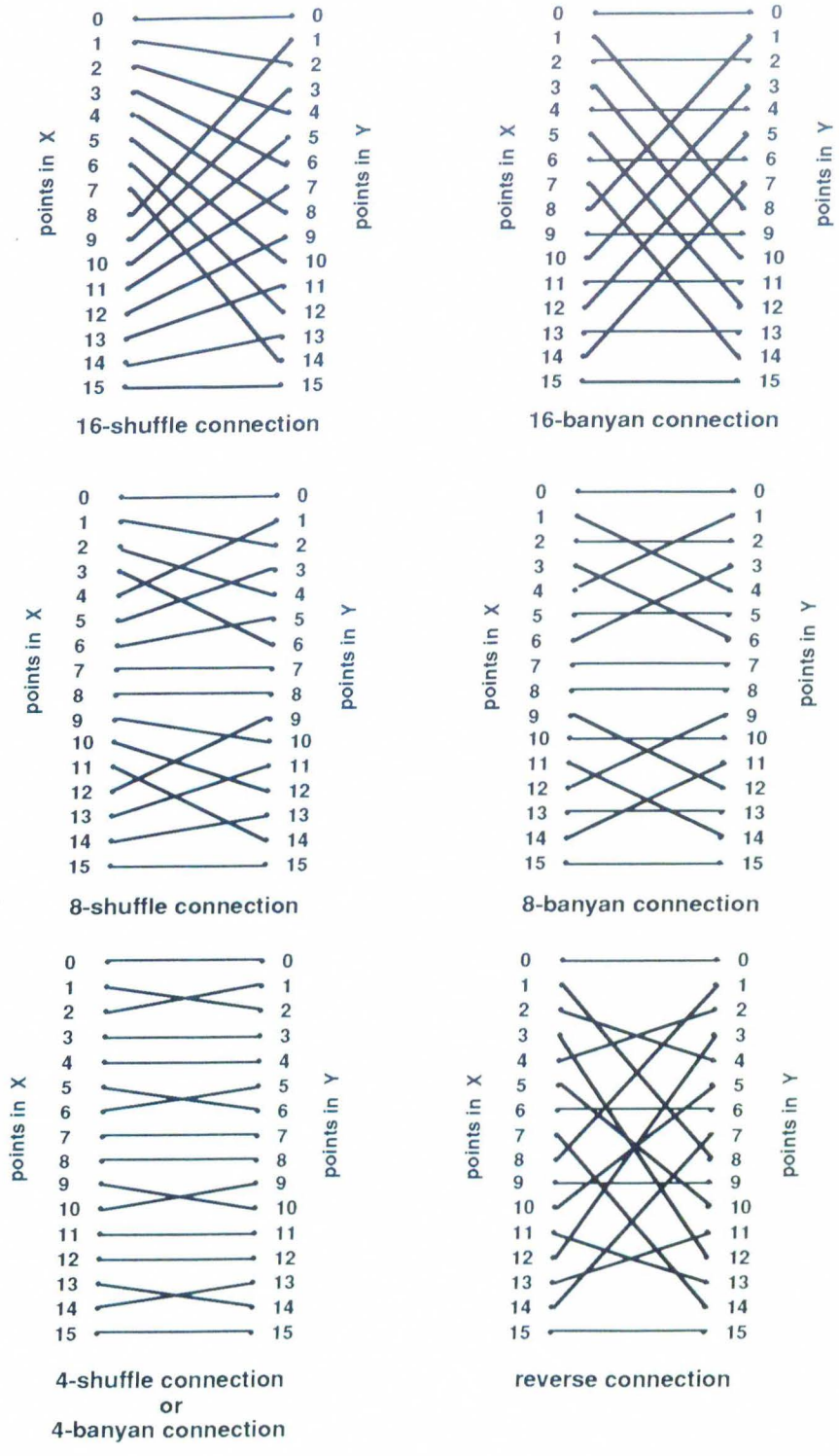



**Figure 2**: Fixed connections on two sets of $n$ points for $n = 16$.

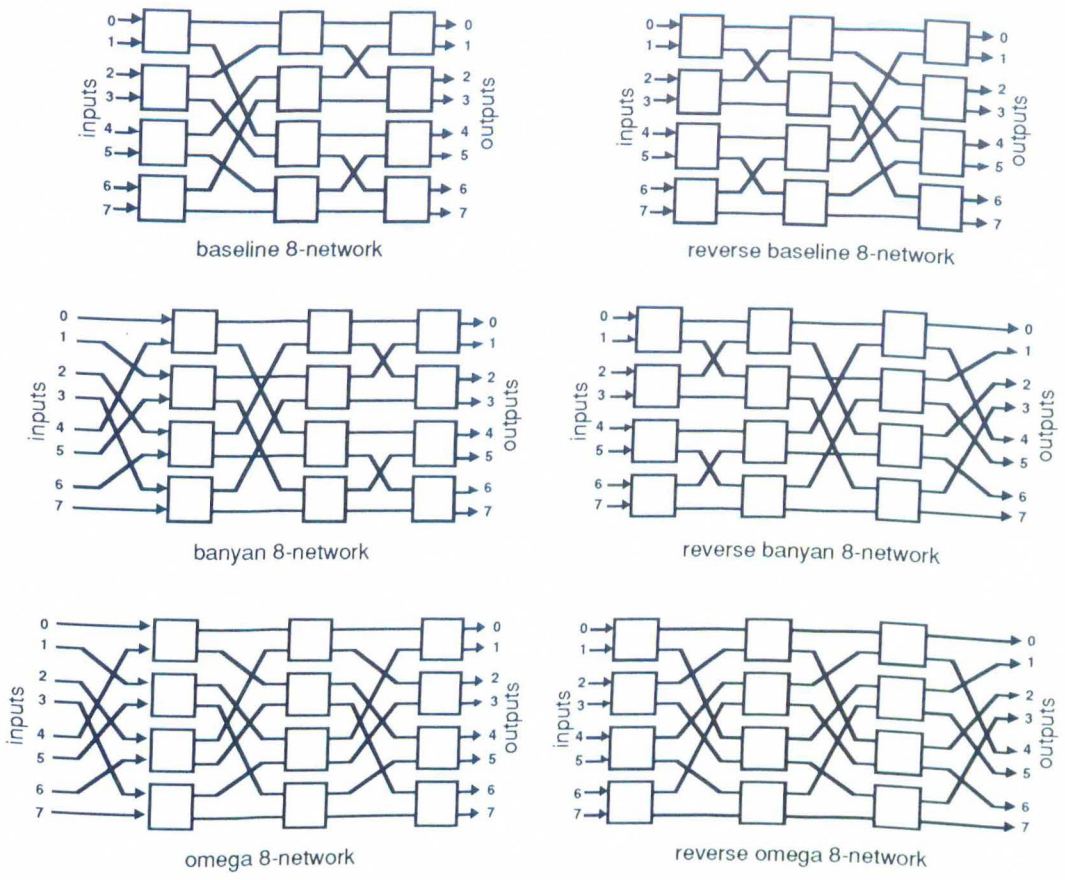



**Figure 3**: Self-routing $n$-networks for $n = 8$.

one obtains a reverse network. Figure 3 illustrates each of these $n$-networks for $n = 8$.

Each of these $n$-networks is self-routing because it possesses the unique path property. However, they are not unicasting $n$-networks. For $n$ inputs, each network can realize only $n^{n/2}$ of the $n!$ $n$-assignments between its inputs and outputs. One can refer to [WF80a, WF80b] for a detailed account of these networks.

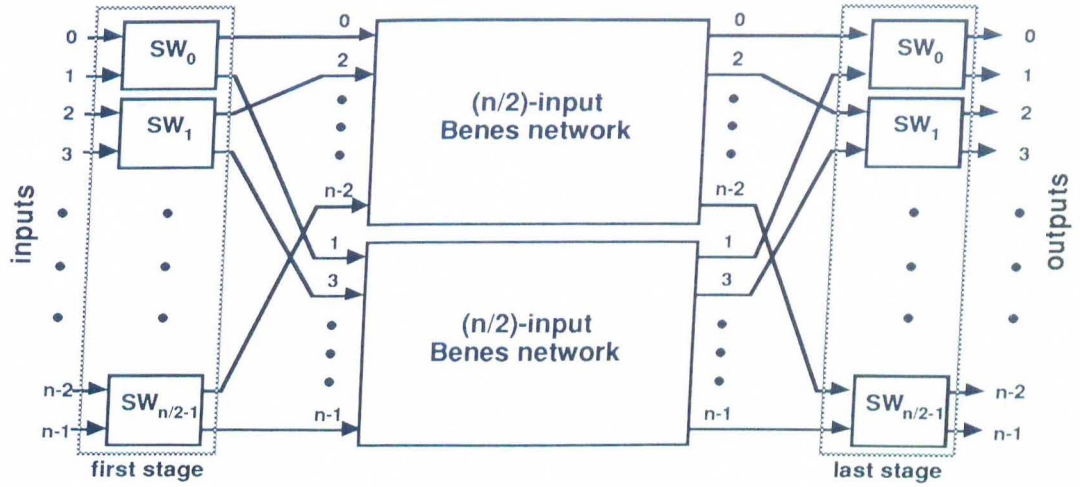



**Figure 4**: The recursive construction of an $n$-input Beneš network.

### 2.2.2 Unicasting Networks

Perhaps, the most well-known unicasting network is the Beneš network, which can be constructed recursively as shown in Figure 4. For $n$ inputs, each of the first and the last stages consists of $n/2$ $2 \times 2$ switches, and the center stag consists of two $n/2$-input Beneš networks. For notational convenience, the inputs and outputs of the upper $n/2$-input Beneš network are numbered $0, 2, \ldots, n-2$, and the inputs and outputs of the lower $n/2$-input Beneš network are numbered $1, 3, \ldots, n-1$, from top to bottom. If the half-sized Beneš networks in the center stage are recursively decomposed then one obtains a $(2 \lg n - 1)$-stage network consisting of $2 \times 2$ switches, assuming that $n$ is a power of 2. Figure 5 depicts the recursively decomposed Beneš network for $n = 8$.

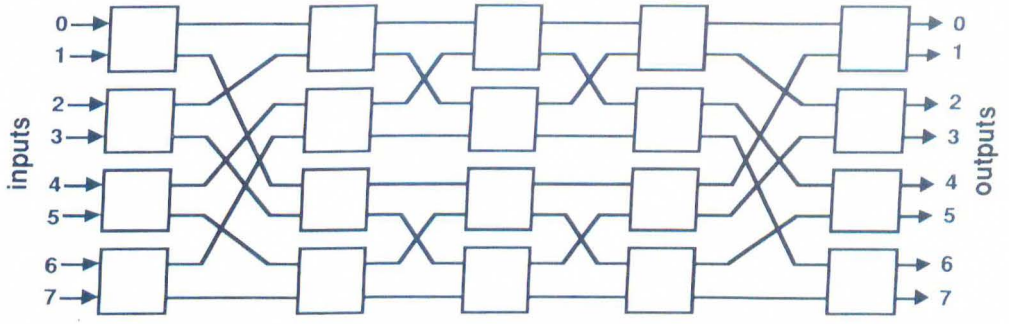



**Figure 5**: A recursive decomposed $n$-input Beneš networks for $n = 8$.

It is clear that an $n$-input Beneš network can also be obtained by concatenating a baseline $n$-network with a reverse baseline $n$-network. Similarly, other unicasting networks could be obtained by concatenating two self-routing networks as shown in the previous subsection. For example, two baseline $n$-networks [WF80a], or an omega $n$-network and a reverse omega $n$-network [Lee85], or a banyan $n$-network and a reverse baseline $n$-network, will result in a unicasting $n$-network.

### 2.2.3 Sorting Networks

An $n$-network is a sorting $n$-network if, for any $k$ numbers at any $k$ of its $n$ inputs, $1 \leq k \leq n-1$, it can realize a unicast assignment that pairs those $k$ inputs to $k$ consecutively numbered outputs such that those $k$ numbers appearing at the $k$ outputs are in an increasing (or decreasing) order. The well-known Batcher's network [Bat68] is a sorting network which consists of $\lg n(\lg n + 1)$ stages of $2 \times 2$ comparators. Another well-known sorting network is the network of H. Stone [Sto71, Knu73], which consists of $\lg^2 n$ stages of $2 \times 2$ comparators where any two consecutive stages are $n$-shuffle connected. Figure 6(a) and (b)

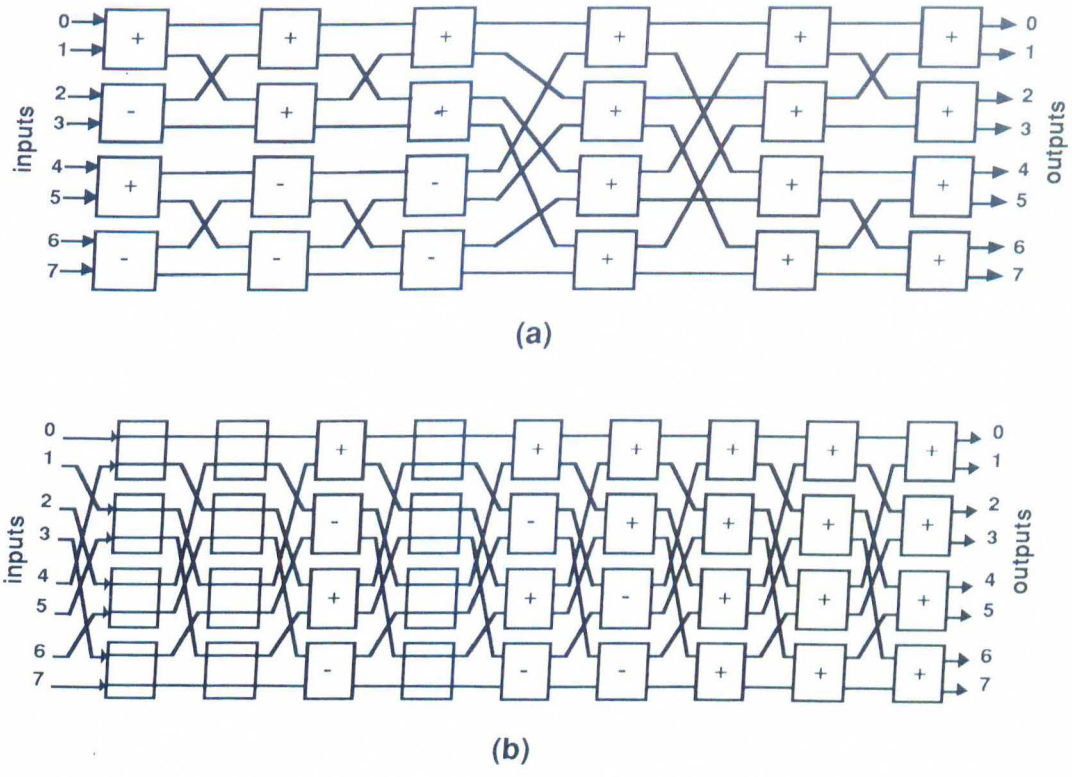



**Figure 6**: (a) A Batcher's sorting network with 8 inputs; (b) An $n$-shuffle connected, $\lg^2 n$-stage sorting network for $n = 8$.

show a Batcher's networks with 8 inputs and an $n$-shuffle connected, $\lg^2 n$-stage sorting network for $n = 8$, respectively.

CHAPTER
THREE

# PARALLEL ROUTING ALGORITHMS FOR BENEŠ NETWORKS

## 3.1 Introduction

Among all kinds of unicasting networks, the Beneš network is more attractive mainly because of its $O(n \lg n)$ bit-level cost and $O(\lg n)$ bit-level depth. But, routing unicast assignments on the Beneš network remains a problem. The looping algorithm [Wak68, OTW71] is not satisfactory simply because the $O(n \lg n)$ time complexity is too large, especially compared with the $O(\lg n)$ depth of the Beneš network. The parallel algorithms reported earlier [LPV81, NS82a] are inefficient in that they can only realize complete or permutation assignments. Besides, their time complexities are still higher than the $O(\lg n)$ depth of the Beneš network.

In this chapter, we present efficient parallel algorithms for routing unicast as-

signments on the Beneš network. Our algorithms can also be pipelined to gain a factor of $O(\lg n)$ speed up over the parallel algorithms of Nassimi and Sahni [NS82a] and Lev et al [LPV81]. Pipelining is made possible by routing the Beneš network stage by stage from left to right and overlapping the routing steps for consecutive stages. Unlike our routing algorithms, the cited parallel algorithms proceed the settings from outer-stage switches to inner-stage switches which makes pipelining difficult.

We realize our routing algorithms on parallel processors interconnected by two different topologies, namely the *complete connection* topology and the *extended perfect-shuffle connection* topology. We show that if every pair of processors are interconnected by a direct arc (i.e., the connection topology is complete) then routing a unicast assignment involving $O(k)$ pairs of inputs and outputs takes $O(\log^2 k + \lg n)$ time without pipelining and $O(\lg k)$ time with pipelining. We also establish that using a weaker topology, the extended perfect-shuffle connection topology (defined in Section 3.3.1), leads to a routing algorithm with $O(\lg^4 k + \lg^2 k \lg n)$ time without pipelining and $O(\lg^3 k + \lg k \lg n)$ time with pipelining. These achieve the optimal $O(\lg n)$ routing time when $k << n$, and they provide the best-known time complexities for routing any one-to-one assignment over the Beneš network.

Also, in our routing schemes, each of the first $\lg n - 1$ stages of an $n$-input Beneš network can be provided with its own special routing module rather than using a single parallel computer to set all the switches. Each routing module is composed of some special purpose processors that can be interconnected by a number of topologies. Each special processor is equipped with a number of $O(\lg n)$-bit registers and some simple arithmetic and logic circuitry to compare

$O(\lg n)$-bit numbers and perform some counting and decoding on them. With these routing modules, once the first $\lg n - 1$ stages are set, the last $\lg n$ stages are then self-routed since they form a reverse baseline $n$-network as stated in Section 2.2.1.

The rest of this chapter is organized as follows. Section 3.2 describes a routing principle which provides the basis of a parallel algorithm for routing on the Beneš network. Section 3.3 discribes this parallel routing algorithm, and Section 3.4 gives its hardware implementation. The chapter is summarized in Section 3.5.

## 3.2 The Routing Principle

In this section, we describe a routing principle which establishes that unicast assignments for the Beneš network can be recursively decomposed into half-sized unicast assignments stage by stage from left to right. This routing principle also permits pipelining of unicast assignments on Beneš network.

### 3.2.1 Notations and Definitions

**Notation:** For $0 \leq i \leq n-1$, if $(b^i_{\lg n-1}, \ldots, b^i_1, b^i_0)$ is the binary representation of $i$, then $\bar{i}$ denotes the integer which has the binary representation $(b^i_{\lg n-1}, \ldots, b^i_1, \bar{b}^i_0)$, and[1] $i$ and $\bar{i}$ are called a *dual pair* of integers. ||

It is assumed that a packet header at input $i$ of an $n$-input Beneš network has $\lg n + 1$ bits, and is denoted as $(r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)$, where $r_i$ is the connecting bit and $(d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)$ is the binary representation of the output paired with

[1] $\bar{b}$ denotes the binary complement of bit $b$.

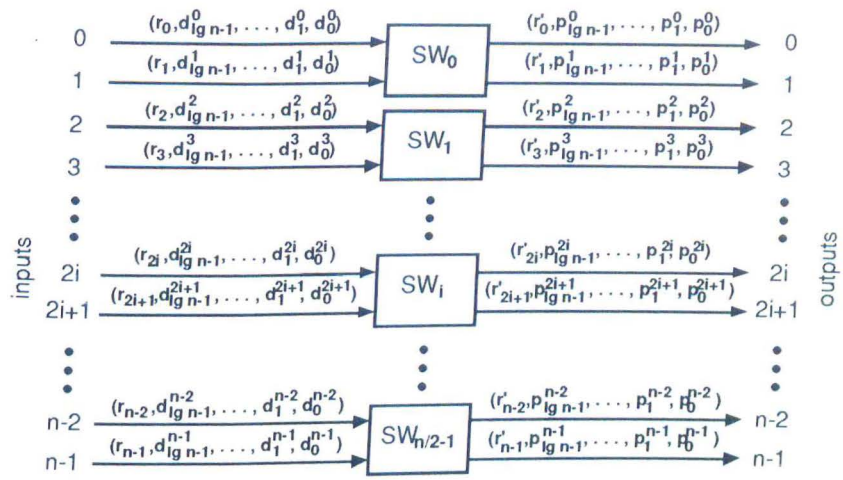



**Figure 7**: $H(n)$: a one-stage $n$-network consisting of $n/2$ $2 \times 2$ switches.

input $i$ with $d^i_{\lg n-1}$ being the most significant bit. In light of this assumption, the following definition makes the notion of unicast assignments more precise.

**Definition 3.1** *A unicast $k$-assignment for an $n$-network is a set $\left\{(i, (r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)) : 0 \leq i \leq n-1\right\}$ such that exactly $k$ $r_i$'s are equal to 1, and $(d^i_{\lg n-1}, \ldots, d^i_1, d^i_0) \neq (d^j_{\lg n-1}, \ldots, d^j_1, d^j_0)$ whenever $i \neq j$ and $r_i = r_j = 1$.* ||

Let $H(n)$ denote a one-stage $n$-network which comprises $n/2$ $2 \times 2$ switches, $SW_0, SW_1, \ldots, SW_{n/2-1}$, as shown in Figure 7, where $n$ is a power of 2.

**Definition 3.2** *Given $\left\{(i, (r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)) : 0 \leq i \leq n-1\right\}$ a unicast assignment for $H(n)$, a sequence of switches $SW_{i_0}, SW_{i_1}, \ldots, SW_{i_{p-1}}$ in $H(n)$ is said to form a chain with respect to that assignment if, for $0 \leq q \leq p-2$, one of the inputs of $SW_{i_q}$ and one of the inputs of $SW_{i_{q+1}}$ are paired with a dual pair of outputs and have their connecting bits set to 1, i.e., there exist $x = 2i_q$ or $x = 2i_q + 1$, and $y = 2i_{q+1}$ or $y = 2i_{q+1} + 1$ for which $r_x = r_y = 1$ and $(d^x_{\lg n-1}, \ldots, d^x_2, d^x_1, d^x_0) = (d^y_{\lg n-1}, \ldots, d^y_2, d^y_1, d^y_0)$.*

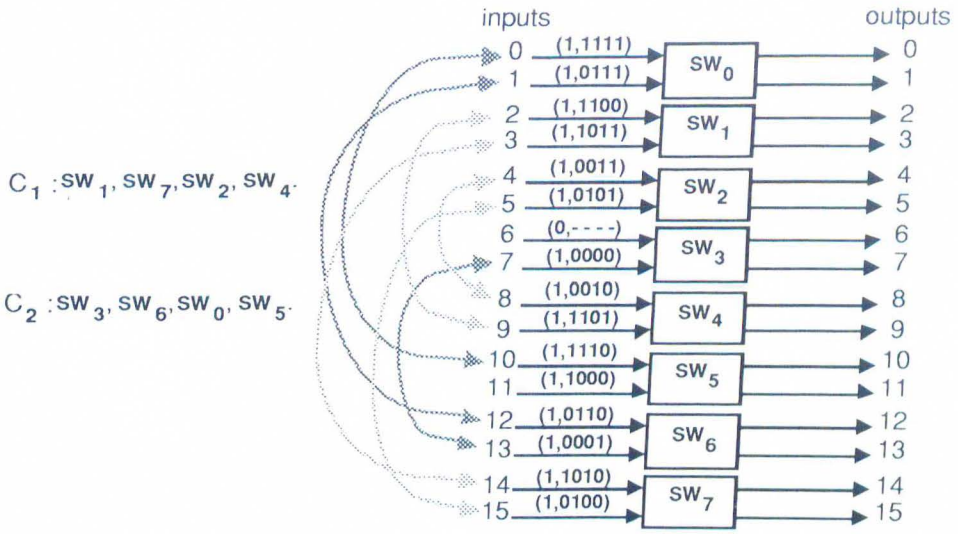



**Figure 8**: Two chains with respect to a unicast 15-assignment for $H(n)$ where $n = 16$.

*Furthermore, $SW_{i_0}, SW_{i_1}, \ldots, SW_{i_{p-1}}$ is said to be a closed chain if the remaining input of $SW_{i_{p-1}}$ and the remaining input of $SW_{i_0}$ are also paired with a dual pair of outputs, and to be an open chain otherwise, and then $SW_{i_0}$ and $SW_{i_{p-1}}$ are called the end switches of that open chain.* ||

The *size* of a chain is the number of switches in the chain. Given a unicast $k$-assignment for $H(n)$, the size of a chain can be as large as $\min\{\lfloor (k+2)/2 \rfloor, n/2\}$ and[2] as small as 1, and the number of chains can be as large as $\min\{k, n/2\}$ (when each chain has size 1) and as small as 1 (when that chain is of size $\lfloor (k+2)/2 \rfloor$ or $\lfloor k/2 \rfloor$). Figure 8 shows a closed chain $C_1$ and an open chain $C_2$ with respect to a unicast 15-assignment for $H(n)$ where $n = 16$.

Each of the two end switches of an open chain, depending on the states (busy

[2] $\lfloor x \rfloor$ denotes the largest integer equal to or smaller than $x$, and $\lceil x \rceil$ denotes the smallest integer equal to or larger than $x$.

or idle) of its inputs, can be in one of the two states: busy or semi-busy. An end switch is called *busy* if both of its inputs are busy, and called *semi-busy* if one of its inputs is busy and the other is idle. Based on the states of the end switches, we distinguish between two types of open chains.

**Definition 3.3** *An open chain is said to be a free open chain if its end switches are both busy or both semi-busy, and to be a half open chain if one of its end switches is busy and the other is semi-busy.* ||

For example, $C_2$ given in Figure 8 is a half open chain since end switch $SW_5$ is busy and end switch $SW_3$ is semi-busy.

### 3.2.2 The Routing Principle

The following theorem states how to decompose a unicast assignment into two half-sized unicast assignments, and will be applied recursively for routing unicast assignments through the first half of stages of the Beneš network.

**Theorem 3.1** *Given* $\left\{(i, (r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)) : 0 \leq i \leq n-1\right\}$ *a unicast* $k$*-assignment for* $H(n)$*, let* $(r'_i, p^i_{\lg n-1}, \ldots, p^i_1, p^i_0)$ *denote the header of the packet at output* $i$ *of* $H(n)$*,* $0 \leq i \leq n-1$*, as shown in Figure 7. There exist settings for* $SW_0, SW_1, \ldots, SW_{n/2-1}$ *such that* $\left\{(2i, (r'_{2i}, p^{2i}_{\lg n-1}, \ldots, p^{2i}_2, p^{2i}_1)) : 0 \leq i \leq n/2-1\right\}$ *is a unicast* $k_0$*-assignment and* $\left\{(2i+1, (r'_{2i+1}, p^{2i+1}_{\lg n-1}, \ldots, p^{2i+1}_2, p^{2i+1}_1)) : 0 \leq i \leq n/2-1\right\}$ *is a unicast* $k_1$*-assignment, where* $k_0 = \lceil k/2 \rceil$ *and* $k_1 = \lfloor k/2 \rfloor$*.*

**Proof:** Suppose that there are $c$ chains, $1 \leq c \leq \min\{k, n/2\}$, with respect to the given unicast $k$-assignment, and suppose that there are $f$ free open

chains among those $c$ chains, $0 \leq f \leq c$. For each chain, by Definition 3.2, its switches can be set such that, for those inputs $i$ and $j$ which have $r_i = r_j = 1$ and $(d^i_{\lg n-1}, \ldots, d^i_2, d^i_1, d^i_0) = (d^j_{\lg n-1}, \ldots, d^j_2, d^j_1, d^j_0)$, one is routed to an even-numbered output and the other is routed to an odd-numbered output, and once a switch in the chain is set then the settings of the other switches in that chain are fixed. Since the first chosen switch can be set in two ways (either *through* or *cross*), there are exactly two ways to set each chain to satisfy the above statement. Besides, different chains can be set mutually independently, i.e., the setting of a chain will not affect the setting of another chain. Therefore, there are $2^c$ ways to set those $c$ chains since each chain has two ways of settings. If the switches in $H(n)$ are set in any one of these $2^c$ ways, then it is easy to verify that $\left\{(2i, (r'_{2i}, p^{2i}_{\lg n-1}, \ldots, p^{2i}_2, p^{2i}_1)) : 0 \leq i \leq n/2 - 1\right\}$ is a unicast $k_0$-assignment and $\left\{(2i+1, (r'_{2i+1}, p^{2i+1}_{\lg n-1}, \ldots, p^{2i+1}_2, p^{2i+1}_1)) : 0 \leq i \leq n/2 - 1\right\}$ is a unicast $k_1$-assignment for some integers $k_0$ and $k_1$ with $k_0 + k_1 = k$. Thus, it suffices to show at least one of the $2^c$ ways of settings will result in $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$. For the closed and free open chains, no matter how they are set, they will increase $k_0$ and $k_1$ by the same number because such chains have even numbers of busy inputs, and a half of the inputs are routed to even-numbered outputs and the other half are routed to odd-numbered outputs. However, a half open chain has an odd number of busy inputs, and its two settings will have different impact on the increases of $k_0$ and $k_1$. One setting, named the *Type-0* setting, will have $k_0$ increase one more than $k_1$, and the other setting, named the *Type-1* setting, will have $k_0$ increase one less than $k_1$. Let $\lceil f/2 \rceil$ of those $f$ half open chains be in their *Type-0* settings and the other $\lfloor f/2 \rfloor$ half open chains be in their *Type-1* settings. Then, obviously, $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$. ||

The previous theorem establishes a routing scheme for the Beneš network. Given a unicast assignment for an $n$-input Beneš network, let the switches in the first stage be set such that the statement of Theorem 1 is satisfied. Then the two established half-sized assignments for the center-stage $n/2$-input Beneš networks are also unicast and can be realized by them, respectively. This implies that if the switches in the first stage are set such that the statement of Theorem 1 is satisfied, then there exist settings for the switches in all of the subsequent stages to realize the unicast assignment. Hence, any algorithm which satisfies the statement of Theorem 1 can be used recursively to set the switches in the first $\lg n - 1$ stages. Thereafter the packets are routed on a self-routing basis through the last $\lg n$ stages to their final destinations. This is because the last $\lg n$ stages of the Beneš network is a reverse baseline $n$-network, which has the well-known property that, between each pair of of its inputs outputs, there is a unique path that can be formed by decoding the output address bit by bit.

## 3.3 The Parallel Routing Algorithm

In this section, we present a parallel routing algorithm for the Beneš network to speed up the routing time. Following the discussion in the previous section, it is only necessary to describe a parallel algorithm for $H(n)$ such that the routing principle stated in Theorem 1 is satisfied, and then the parallel routing algorithm for the Beneš network follows.

In the parallel algorithm for $H(n)$, it is assumed that there are $n$ interconnected processors, $PR(0), PR(1), \ldots, PR(n-1)$, and packet header $(r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)$ is initially input to $PR(i)$, and $PR(i)$ and $PR(i)$ are called a *dual pair* of pro-

cessors and will determine the setting of $SW_{\lfloor i/2 \rfloor}$ of $H(n)$, $0 \leq i \leq n-1$. The time complexity of the parallel algorithm depends on the intercommunication capability of the connection topology between these $n$ processors. The parallel algorithm can run on two connection topologies, namely the *complete* connection topology and the *extended perfect-shuffle* connection topology. It will be shown that any unicast $k$-assignment can be realized over an $n$-input Beneš network in $O(\lg^2 k + \lg n)$ time if the connection topology is complete, and in $O(\lg^4 k + \lg^2 k \lg n)$ time if the connection topology is extended perfect shuffle.

### 3.3.1 The Connection Topologies

For ease of discussion, we first describ three kinds of processes: *move process*, *concentrate process* and *broadcast process*, which will be used in the parallel algorithm for exchanging data between the $n$ interconnected processors. A move process transfers data from some old processors to some new processors in a one-to-one manner, i.e., no data will be moved from one processor to more than one processor, and no processor will receive data from more than one processor. A concentrate process transfers data from some old processors to a specific processor in a many-to-one manner. A broadcast process transfers data from a specific processor to some new processors in a one-to-many manner. In each process, there are destination addresses used to specify the location(s) of new processor(s) to which data are transferred. The time complexities to execute these processes depend on the intercommunication capability of the connection topology between the processors, and are discussed as follows.

The first connection topology between the $n$ processors is the complete con-

nection topology on which there is a connection between any two processors. The second connection topology between $PR(0), PR(1), \ldots, PR(n-1)$ is the *extended perfect-shuffle* connection topology on which each dual pair of processors are connected, and $PR(0), PR(1), \ldots, PR(n/2^m - 1)$ are $(n/2^m)$-shuffle interconnected, for $m = 0, 1, \ldots, \lg n - 2$, i.e., $PR(0), PR(1), \ldots, PR(n-1)$ are $n$-shuffle interconnected, $PR(0), PR(1), \ldots, PR(n/2-1)$ are $n/2$-shuffle interconnected, and so on. Clearly, the complete connection topology uses $O(n^2)$ connections, and the extended perfect-shuffle connection topology uses $O(n)$ connections.

The complete connection topology has a very strong intercommunication capability. It is obvious that each of the three processes can be executed in $O(1)$ time on the complete connection topology. On the other hand, it takes longer time to execute these processes on the extended perfect-shuffle connection topology. A concentrate process or a broadcast process can be finished by repeatedly passing $\lg n$ times through the $n$-shuffle connection between the extended perfect-shuffle interconnected processors, and hence they can be executed in $O(\lg n)$ time. However, it takes three steps to execute a move process on the extended perfect-shuffle interconnected processors. Suppose that there are $k$ processors that have data to be moved in a move process. At the first step, these $k$ data items are moved to $k$ of the first $2^{\lceil \lg k \rceil}$ processors by passing $\lg n$ times through the $n$-shuffle connection. At the second step, these $k$ data items are sorted according to their destination addresses by repeatedly passing $\lceil \lg k \rceil^2$ times through the $(2^{\lceil \lg k \rceil})$-shuffle connection between $PR(0), PR(1), \ldots, PR(2^{\lceil \lg k \rceil} - 1)$, as discussed in Subsection 2.2.3. At the third step, the sorted data are moved to their final destinations by passing $\lg n$ times through the $n$-shuffle connection. Hence, it $O(\lg^2 k + \lg n)$ time to execute a

move process involving $k$ processors on the extended perfect-shuffle connection topology.

### 3.3.2 The Parallel Routing Scheme

To facilitate an understanding of the parallel routing algorithm, we give a parallel routing scheme which outlines the algorithm.

From the proof of Theorem 3.1, inputs that belong to a chain can be partitioned into two *equivalence classes* such that the inputs in one equivalence class are connected to even-numbered outputs and the inputs in the other equivalence class are connected to odd-numbered outputs. Once the equivalence classes of inputs in a chain are established, the determination of the settings for the chain is straightforward. The parallel algorithm will use such equivalence classes to determine the setting of each switch. First, the following proposition shows how to determine the pairs of inputs that are in the same equivalence class.

**Proposition 3.1:** Let $\left\{(i,(r_i,d^i_{\lg n-1},\ldots,d^i_1,d^i_0)) : 0 \le i \le n-1\right\}$ be a unicast assignment for $H(n)$. If $(d^i_{\lg n-1},\ldots,d^i_1,d^i_0) = (d^j_{\lg n-1},\ldots,d^j_1,d^j_0)$, $r_i = r_j = 1$ and $i \neq j$, then inputs $i$ and $\bar{j}$ are in the same equivalence class and inputs $j$ and $\bar{i}$ are in another equivalence class.

**Proof:** By Definition 3.2, $SW_{\lfloor i/2 \rfloor}$ and $SW_{\lfloor j/2 \rfloor}$ are in the same chain since input $i$ of $SW_{\lfloor i/2 \rfloor}$ and input $j$ of $SW_{\lfloor j/2 \rfloor}$ are paired with a dual pair of outputs. Thus, by Theorem 3.1, inputs $i$ and $j$ must be routed in different directions, one connected to an even-numbered output and the other connected to an odd-numbered output. Because $i$ and $\bar{i}$ are a dual pair of inputs and $j$ and $\bar{j}$ are another dual pair of inputs, $i$ and $\bar{j}$ are in the same equivalence class, and so are $j$ and $\bar{i}$. ||

**Remark 3.1** *To manifest the equivalence class relation, each input will be transformed into an ordered quadruples in which the first element is a binary bit used to indicate if this input belongs to a closed or an open chain, the second element corresponds to this input, the third element points to an input that is in the same equivalence class as this input, and the fourth element, to be called the representative of this input, will be used to route this input. If $i$ and $j$ satisfy the hypothesis of Proposition 3.1, two ordered quadruples $\langle 1; i, j; p_i \rangle$ and $\langle 1; j, i; p_j \rangle$ will be established, where their first elements are initialized to 1 and their fourth elements are initialized to $p_i = \min\{i, j\}$ and $p_j = \min\{j, i\}$. If $r_i = 1$ and there is no $r_j = 1$ for which $(d^i_{\lg n-1}, \ldots, d^i_1, d^i_0) = (d^j_{\lg n-1}, \ldots, d^j_1, d^j_0)$ (i.e., input $i$ is paired with an output whose dual output is idle), an ordered quadruple $\langle 1; i, -1; i \rangle$ will be established, where $-1$ is used to denote that this input belongs to a busy end switch of an open chain, or an open chain of size 1. If $r_i = 0$, an ordered quadruple $\langle 0; i, -2; i \rangle$ will be established, where $-2$ is used to denote that this input belongs to a semi-busy end switch of an open chain, or an idle switch.* ||

The parallel algorithm can be roughly divided into four phases. In the first phase, ordered quadruples as defined in Remark 3.1 are established. In the second phase, the representative in each quadruple is computed such that each input knows which chain it belongs to, and all the inputs in the same equivalence class will agree on a common representative. In the third phase, each half open chain is assigned either the *Type-0* setting or *Type-1* setting so that the statement of Theorem 3.1 (i.e., $k_0 = \lceil k/2 \rceil$ and $k_1 = \lfloor k/2 \rfloor$) will be satisfied. In the fourth phase, each switch is set by using the representatives of its two inputs.

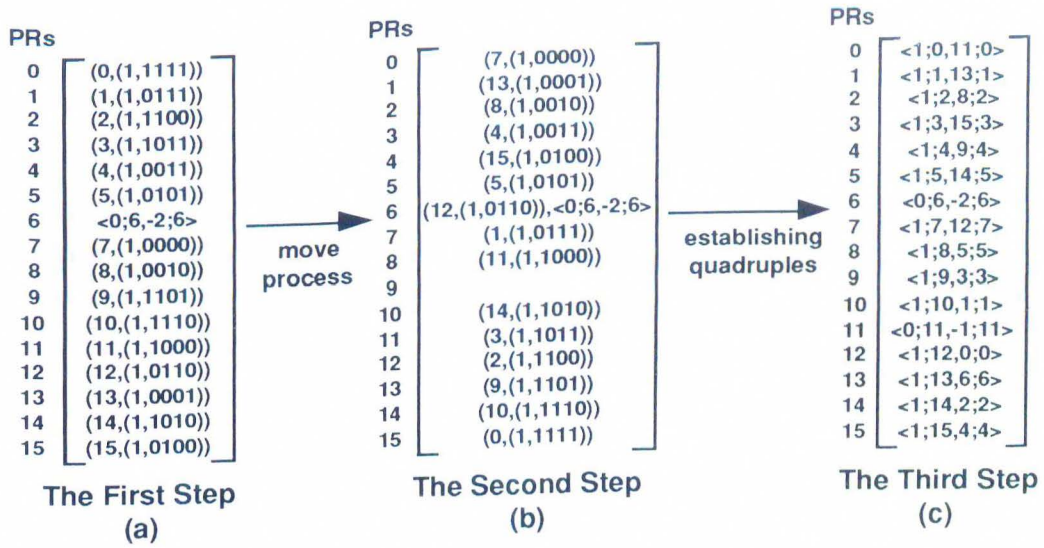



**Figure 9**: An illustration of the three steps in the first phase.

The first phase applies Remark 3.1 to establish ordered quadruples, and can be further decomposed into three steps. At the first step, packet headers are input to the processors, and the idle inputs which belong to the semi-busy switches have their quadruples established. At the second step, packet headers are moved to new processors so that each dual pair of processors can apply Remark 3.1 in parallel. At the third step, the remaining quadruples are established and then moved to new processors specified by their second elements, and their first elements are changed to 0 if their third elements are -1. Using the unicast assignment given in Figure 8 as an example, the first phase is illustrated in Figure 9, where a column with 16 entries is used to express the data stored in the 16 processors at each step.

The second phase is an iterative procedure to compute the representative of each quadruple. This computation is the crux of the parallel algorithm, and it needs to be explained in detail before we go to the description of the second phase.

Any chain, depending upon its type, is decomposed into two sequences of quadruples in the first phase where the quadruples in each sequence belong to the same equivalence class. Let us use the quadruples in each sequence as nodes to form a directed graph in which a directed arc is established from a quadruple to another quadruple if and only if the third element of the former quadruple is equal to the second element of the latter quadruple. Then, a $k$-size closed chain will form two $k$-node *closed subchains* for which each node has an incoming arc and an outgoing arc, and a $k$-size open chain will form two $k$-node *open subchains* for which each source node has an outgoing arc, each sink node has an incoming arc, and each of the other nodes has an incoming arc and an outgoing arc. For example, Figure 10 explicitly depicts the four subchains obtained from the quadruples in (c) of Figure 9.

These subchains will be used to facilitate an understanding of the iterative procedure. Initially, for each quadruple as it is established in the first phase, its first element is 0, its third element is either -1 or -2 and its fourth element points to its second element if this quadruple is the sink quadruple of an open subchain, or otherwise its first element is 1, its second element points to the second element of its *immediate successor* (i.e., the quadruple at distance 1 away from itself) and its fourth element points to the second element of either itself or its immediate successor. In the $m$-th iteration, the elements of each quadruple is updated as follows. If a quadruple is in a closed subchain, then its first element remains 1, its third element points to the second element of its $2^m$-th successor and its last element points to the smallest input among the $2^m$ second elements of its first $2^m$ successors. If a quadruple is in an open subchain, then the way its elements are updated depends on when it is known that this quadruple is in an open subchain. When this quadruple is at distance

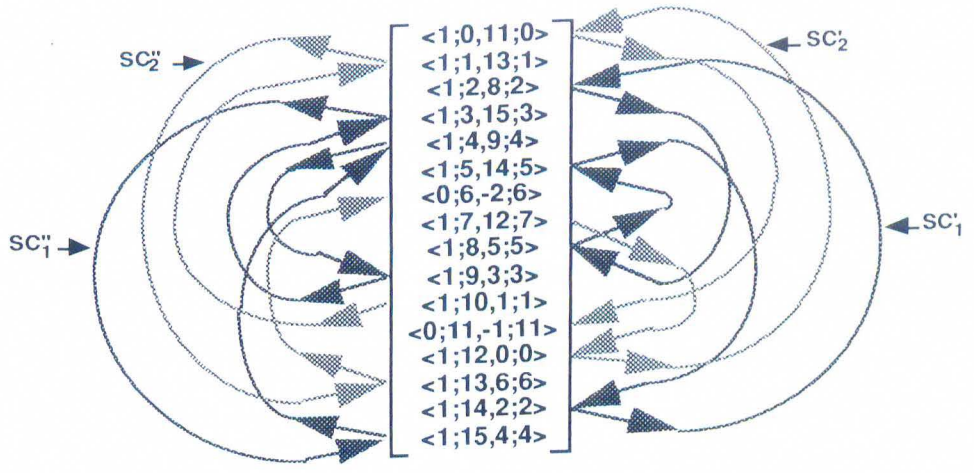



**Figure 10**: The subchains formed from the quadruples in (c) of Figure 9.

less than $2^m$ away from the sink quadruple, this quadruple is recognized to be in an open subchain, and its first element is changed to 0, its third element points to the third element of the sink quadruple and its last element points to the second element of the sink quadruple. On the other hand, when this quadruple is still at least $2^m$ far away from the sink quadruple, its elements are updated in the same way as if this quadruple is in a closed subchain. That is, in each open subchain, the updating information is exponentially propagated from the sink quadruple to the source quadruple. Using this iterative updating procedure, after $\lg\lceil k \rceil$ iterations, any $k$ quadruples that form a closed subchain will select the smallest input among their second elements as their common representative, and any $k$ quadruples that form an open subchain will select the second element of the sink quadruple as their common representative. For example, the four subchains in Figure 10 need two iterations to determine their representatives, and inputs 2, 3, 11 and 6 will be selected as the representatives of the quadruples in $SC'_1$, $SC''_1$, $SC'_2$ and $SC''_2$, respectively.

Now we proceed to describe the second phase of the algorithm. Given a unicast

$k'$-assignment, the second phase is assumed to consist of $\lg\lceil k \rceil$ iterations, where $k = \min\{\lfloor (k'+2)/2 \rfloor, n/2\}$. This assumption is justified since there is no prior information about the exact sizes of the subchains and $\min\{\lfloor (k'+2)/2 \rfloor, n/2\}$ is an upper bound for the sizes of the subchains with respect to the $k'$-assignment as stated in Subsection 3.1.1. Technically, each iteration can be further decomposed into three steps. At the first step, those quadruple whose first elements are 1 are duplicated and the copies are moved to new processors specified by their third elements. At the second step, each processor updates the quadruple(s) that it holds as follows: (1) when it holds only a quadruple, it kills the quadruple if the first element of the quadruple is 1, and otherwise it keeps the quadruple intact; (2) when it holds $\langle 1; l, i; p_l \rangle$ and $\langle 1; i, j; p_i \rangle$, it replaces the first quadruple by $\langle 1; l, j; p'_l \rangle$ where $p'_l = \min\{p_i, p_l\}$, and kills the second quadruple; (3) when it holds $\langle 1; l, i; p_l \rangle$ and $\langle 0; i, *; j \rangle$ where $*$ is either $-1$ or $-2$, it replaces the first quadruple by $\langle 1; l, *; j \rangle$ and keeps the second quadruple intact. At the third step, each updated quadruple is first moved to a new processor specified by its second element if its first element remains 1, and then its first is changed to 0 if its third element is either -1 or -2. For example, we illustrate the second phase in (a) to (e) of Figure 11, where the transitions from (a) to (b) and from (b) to (c) constitute the first iteration, and the transitions from (c) to (d) and from (d) to (e) constitute the second iteration. Note that, after the computation of the second phase, $PR(i)$ and $PR(\bar{i})$ hold the quadruples that corresponds to inputs $i$ and $\bar{i}$ of $SW_{\lfloor i/2 \rfloor}$, respectively. Besides, the chain to which $SW_{\lfloor i/2 \rfloor}$ belongs is clarified by checking the elements of the two quadruples held in $PR(i)$ and $PR(\bar{i})$. If both of their first elements are 1, then $SW_{\lfloor i/2 \rfloor}$ belongs to a closed chain. If both of their third elements are -1 or -2, then $SW_{\lfloor i/2 \rfloor}$ belongs to a free open chain. If one of their third elements is -1

and the other is -2, then $SW_{\lfloor i/2 \rfloor}$ belongs to a half open chain.

Subsequently, the third phase assigns types of settings to each of the half open chains, and can be decomposed into three parts. In the first part, the representative of the quadruple held in $PR(i)$ is concentrated to $PR(0)$ if the third element of that quadruple is -1 and if the second and third elements of the quadruple held in $PR(\bar{i})$ is $\bar{i}$ and -2, respectively (i.e., $SW_{\lfloor i/2 \rfloor}$ is the semi-busy end switch of a half open chain and input $i$ is the busy input). In the second part, $PR(0)$ assigns *Type-0* settings to a half of the concentrated representatives and *Type-1* settings to the other half, and broadcasts this information to all the processors. In the third part, for each quadruple, its third element is changed to 0 if its representative is assigned the *Type-0* setting, and to 1 if its representative is assigned the *Type-1* setting. For example, (f) of Figure 11 shows the result of the third phase, where the third elements of the quadruples held in $PR(0), PR(7), PR(11)$ and $PR(12)$ are change to 0 since these quadruples belong to half open chain $C_2$ and $C_2$ is assigned the *Type-0* setting.

Having decided the type of each chain and computed the representative of each quadruple, the fourth phase determines the settings of switches in $H(n)$. Each switch is set by a dual pair of processors which hold the two quadruples that correspond to the inputs of this switch, and the setting depends on the type of the chain to which this switch belongs.

**Case 1**: If $PR(i)$ holds $\langle 1; i, k; j \rangle$ and $PR(\bar{i})$ holds $\langle 1; \bar{i}, l; \bar{j} \rangle$, then $SW_{\lfloor i/2 \rfloor}$ is in a closed chain. Assuming that $SW_{\lfloor j/2 \rfloor}$ is set *through*, $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i - j$ is even and *cross* otherwise.

**Case 2**: If $PR(i)$ holds $\langle 0; i, p; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, p; \bar{l} \rangle$ where $p = -1$ or

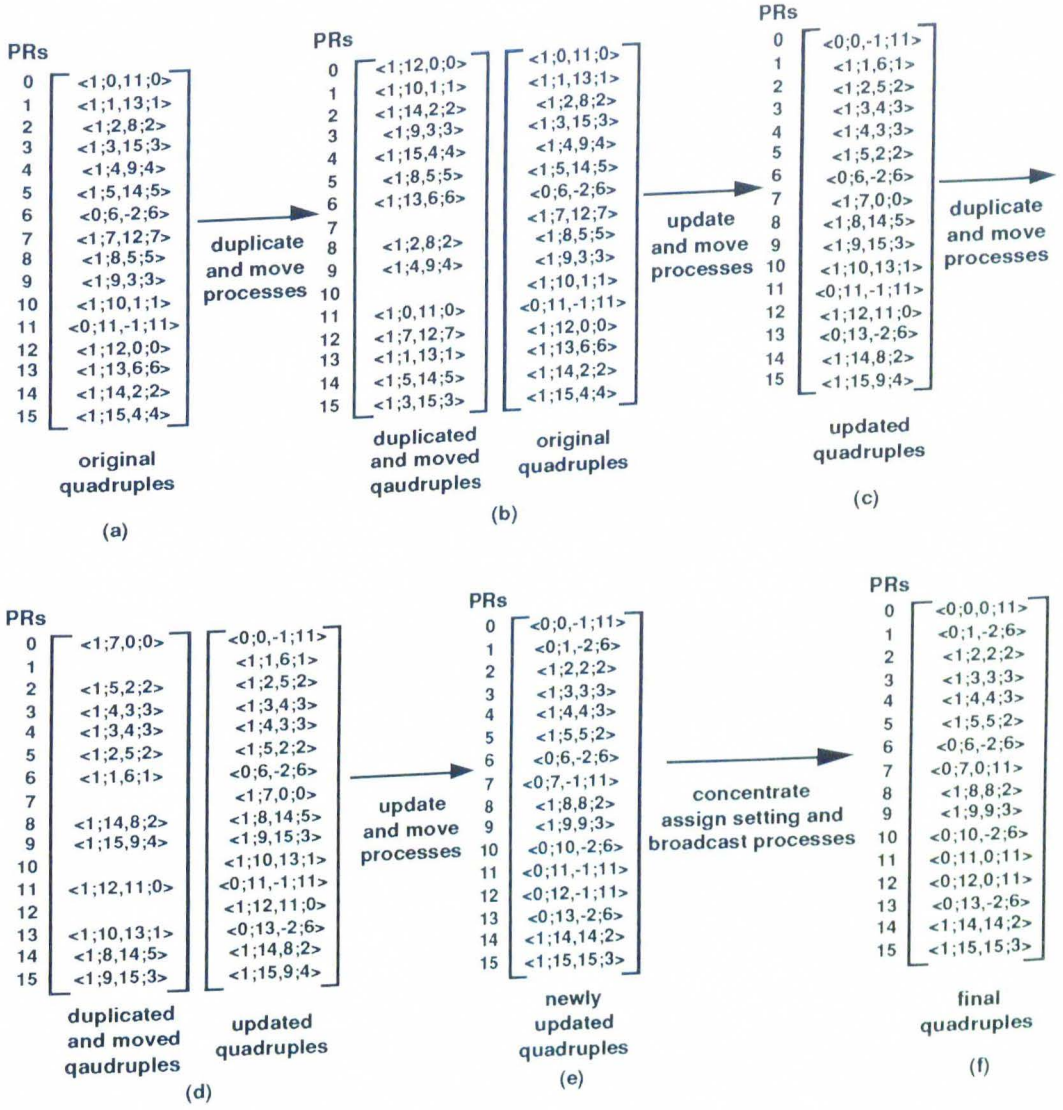



**Figure 11**: The second and third phase of the parallel algorithm that follow the first phase as shown in Figure 9.

$p = -2$, then $SW_{\lfloor i/2 \rfloor}$ is in a free open chain, and $SW_{\lfloor j/2 \rfloor}$ and $SW_{\lfloor l/2 \rfloor}$ are the end switches of the free open chain. Assuming that the smaller one of $SW_{\lfloor j/2 \rfloor}$ and $SW_{\lfloor l/2 \rfloor}$ is set *through*, when $j < l$, $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i - j$ is even and *cross* otherwise, and when $l < j$, $SW_{\lfloor i/2 \rfloor}$ must be *through* if $i - l$ is even and *cross* otherwise.

**Case 3**: If $PR(i)$ holds $\langle 0; i, q; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, -2; l \rangle$ where $q = 0$ or 1, then $SW_{\lfloor i/2 \rfloor}$ is in a half open chain and $SW_{\lfloor j/2 \rfloor}$ is the semi-busy end switch of the half open chain. When $q = 0$ (i.e., the half open chain is assigned the *Type-0* setting), $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i$ is even and *cross* otherwise. When $q = 1$ (i.e., the half open chain is assigned the *Type-1* setting ), $SW_{\lfloor i/2 \rfloor}$ must be set *through* if $i$ is odd and *cross* otherwise.

For example, switches $SW_0, SW_1, SW_4, SW_6$ and $SW_7$ in Figure 8 are set *through* and switches $SW_2, SW_3$ and $SW_5$ are set *cross* by checking the final quadruples held in each dual pair of processors as shown in (f) of Figure 11.

### 3.3.3 The Parallel Algorithm

The following parallel routing algorithm formalizes the steps outlined in the previous subsection.

**Step 1:** Given $(i, (r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0))$ input to $PR(i)$, $PR(i)$ establishes $\langle 0; i, -2; i \rangle$ if $r_i = 0$, $0 \leq i \leq n-1$. Let $k'$ be the number of $r_i$'s whose value are 1, and let $k = \min\{\lfloor (k'+2)/2 \rfloor, n/2\}$. Let $m$ be initialized to 0.

**Step 2:** Move $(i, (r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0))$ to $PR(x)$ if $r_i = 1$ and $(d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)$ is the binary representation of $x$, $0 \leq i \leq n-1$.

**Step 3:** Given that $PR(x)$ holds $(i, r_i)$ and $PR(x)$ holds $(j, r_j)$, $PR(x)$ establishes $\langle 1; i, \bar{j}; p_i \rangle$ and $PR(\bar{x})$ establishes $\langle 1; j, \bar{i}; p_j \rangle$ where $p_i = \min\{i, j\}$ and

$p_j = \min\{j, i\}$ if $r_i = r_j = 1$, or $PR(x)$ establishes $\langle 1; i, -1; i\rangle$ if $r_i = 1$ and $r_j = 0$, or $PR(x)$ establishes $\langle 1; j, -1; j\rangle$ if $r_i = 0$ and $r_j = 1$, $0 \le x \le n-1$. Then, Move $\langle 1; i, j; p_i\rangle$ to $PR(i)$, and $PR(i)$ changes the first element from 1 to 0 if $j = -1$, $0 \le i \le n-1$.

(Step 1, Step 2 and Step 3 constitute the first phase.)

**Step 4:** $m = m + 1$. If $m \le \lceil \lg k \rceil$ then go to Step 5, else go to Step 8.

**Step 5:** Duplicate $\langle 1; l, i; p_l\rangle$ and move a copy to $PR(i)$, $0 \le i \le n-1$. (Those quadruples whose first elements are 1 are duplicated and moved to new processors specified by their third elements.)

**Step 6: Case (1)**– When $PR(i)$ holds only a quadruple: if the first element of the quadruple is 1 then $PR(i)$ kills the quadruple, else $PR(i)$ keeps the quadruple intact, $0 \le i \le n-1$;

**Case (2)**– When $PR(i)$ holds $\langle 1; l, i; p_l\rangle$ and $\langle 1; i, j; p_i\rangle$ : $PR(i)$ replaces the first quadruple by $\langle 1; l, j; p'_l\rangle$ where $p'_l = \min\{p_i, p_l\}$ and kills the second quadruple, $0 \le i \le n-1$;

**Case (3)**– When $PR(i)$ holds $\langle 1; l, i; p_l\rangle$ and $\langle 0; i, *; j\rangle$ where $*$ is $-2$ or $-1$ : $PR(i)$ replaces the first quadruple by $\langle 1; l, *; i\rangle$ and keeps the second quadruple intact, $0 \le i \le n-1$.

**Step 7:** Move $\langle 1; i, j; p_i\rangle$ to $PR(i)$, and $PR(i)$ changes the first element from to 0 if $j = -1$ or $j = -2$, $0 \le i \le n-1$. Then, go to Step 4.

(Step 4, Step 5, Step 6 and Step 7 constitute the second phase.)

**Step 8:** If $PR(i)$ holds $\langle 0; i, -1; j\rangle$ and $PR(i)$ holds $\langle 0; i, -2; i\rangle$, then $PR(i)$ concentrates $j$ to $PR(0)$, $0 \le i \le n-1$. Upon receiving the concentrated numbers, say $j_1, j_2, \ldots, j_f$, $PR(0)$ broadcasts an ordered sequence, $(j_1, j_2, \ldots, j_f)$, to $PR(i)$, $0 \le i \le n-1$. Then, $PR(i)$ changes its quadruple $\langle 0; i, -1; j\rangle$ to $\langle 0; i, 0; j\rangle$ if $j$ is among the first $\lceil f/2 \rceil$ elements of $(j_1, j_2, \ldots, j_f)$, and to

$\langle 0; i, 1; j \rangle$ if $j$ is among the last $\lfloor f/2 \rfloor$ elements of $(j_1, j_2, \ldots, j_f)$, $0 \leq i \leq n-1$.

(Step 8 constitutes the third phase.)

**Step 9: Case 1**: If $PR(i)$ holds $\langle 1; i, j; p_i \rangle$ and $PR(\bar{i})$ holds $\langle 1; \bar{i}, l; p_{\bar{i}} \rangle$, then $PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - p_i$ is even and *cross* otherwise, $0 \leq i \leq n-1$.

**Case 2**: If $PR(i)$ holds $\langle 0; i, p; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, p; l \rangle$ where $p = -1$ or $p = -2$, then they compare $j$ and $l$. When $j < l$, $PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - j$ is even and *cross* otherwise, and when $l < j$, $PR(\bar{i})$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $\bar{i} - l$ is even and *cross* otherwise, $0 \leq i \leq n-1$.

**Case 3**: If $PR(i)$ holds $\langle 0; i, q; j \rangle$ and $PR(\bar{i})$ holds $\langle 0; \bar{i}, -2; l \rangle$ where $q = 0$ or $q = 1$, $PR(i)$ sets $SW_{\lfloor i/2 \rfloor}$ *through* if $i - q$ is even and *cross* otherwise, $0 \leq i \leq n-1$.

(Step 9 constitutes the third phase.)

### 3.3.4 Performance Analysis

In the previous parallel algorithm, move processes dominate the time complexity. As shown in Subsection 3.3.1, each move process can be executed in $O(1)$ time if the $n$ processors are completely interconnected, and in $O(\lg^2 k + \lg n)$ time if the $n$ processors are extended perfect-shuffle interconnected. For a unicast $k$-assignment, since the algorithm uses $O(\lg k)$ move processes, the switches in $H(n)$ can be set in $O(\lg k)$ time if the interprocessor connection topology is complete, and in $O(\lg^3 k + \lg k \lg n)$ time if the interprocessor connection topology is extended perfect shuffle. To realize the unicast $k$-assignment on the Beneš network, it only needs to recursively apply this algorithm $\lceil \lg k \rceil$ times for setting switches in the first $\lceil \lg k \rceil$ stages. This is because the unicast $k$-assignment is decomposed into two half-sized unicast assignments after an application of the

algorithm, and it will be decomposed into $k$ unicast 1-assignments after $\lceil \lg k \rceil$ applications. Therefore, by using this parallel algorithm, the routing time for an $n$-input Beneš network to realize a unicast $k$-assignment is $O(\lg^2 k + \lg n)$ if the interprocessor connection topology is complete, and is $O(\lg^4 k + \lg^2 k \lg n)$ if the interprocessor connection topology is extended perfect shuffle.

Also, unicast assignments can be pipelined over the stages of the Beneš network by using this parallel routing algorithm. That is, when the switch settings in a stage for a unicast assignment are finished, the switch settings in that stage for another unicast assignment can proceed. Pipelining will reduce the average routing time needed to realize a series of unicast assignments. Suppose that there are $\alpha$ consecutive unicast assignments to be realized over an $n$-input Beneš network. Without pipelining, the average routing time to realize an assignment is $O(\lg^2 k + \lg n)$ if the interprocessor connection topology is complete, and is $O(\lg^4 k + \lg^2 k \lg n)$ if the interprocessor connection topology is extended perfect shuffle. But with pipelining, the total routing time to realize these $\alpha$ assignments is $O(\lg^2 k + \lg n + (\alpha - 1) \lg k)$ if the processors is completely interconnected, and is $O(\lg^4 k + \lg^2 k \lg n + (\alpha - 1)(\lg^3 k + \lg k \lg n))$ if the processors is extended perfect-shuffle interconnected, and the average routing time to realize an assignment is reduced to $O(\lg k)$ if the interprocessor connection topology is complete, and to $O(\lg^3 k + \lg k \lg n)$ if the interprocessor connection topology is extended perfect shuffle, for $\alpha \geq \lg n$.

## 3.4 The Hardware Implementation

The parallel routing algorithm stated in the previous section can be implemented on simple hardware circuits as well as parallel computer models. In

this section, we show briefly how to build such routing modules into the stages in the first half of the Beneš network.

Each routing module is an hardware implementation of the parallel algorithm described in Subsection 3.3.3, and will be called a *router*. Each router is composed of a set of special purpose processors that are interconnected by some connection topology. One interconnection choice is the complete connection topology, and another interconnection candidate is the extended perfect-shuffle connection topology, as stated in the previous section. Three types of hardware devices–registers, counters and comparators–and some connection circuitry are provided in each special purpose processor. Registers are used to hold packet headers and quadruples that are input, generated, transferred and updated as stated in the algorithm. Counters are used to record the number of the move processes executed. Comparators are used to update the quadruples while the representatives are computed. And, connection circuitry is used to interconnect these hardware devices in each special purpose processor.

Using such routers, Figure 12 shows the recursive configuration of an $n$-input modified Beneš network in which there is an $(n, n/2)$-router in the first stage. The $(n, n/2)$-router receives $n$ packet headers, $(r_i, d^i_{\lg n-1}, \ldots, d^i_1, d^i_0)$'s, from the inputs of switches in the first stage and then generates $n/2$ binary control bits, $s_j$'s, to set the $n/2$ switches in that stage. With its center-stage subnetwork fully decomposed, the $n$-input modified Beneš network has $2^{k-1}$ $(2n/2^k, n/2^k)$-routers in its $k$-th stage, $1 \leq k \leq \lg n - 1$, and it can thus set its own switches to realize any unicast assignment. Figure 13 shows such a recursively decomposed $n$-input modified Beneš network for $n = 8$. As one can see, the possibility of pipelining over such a recursively decomposed modified Beneš network is

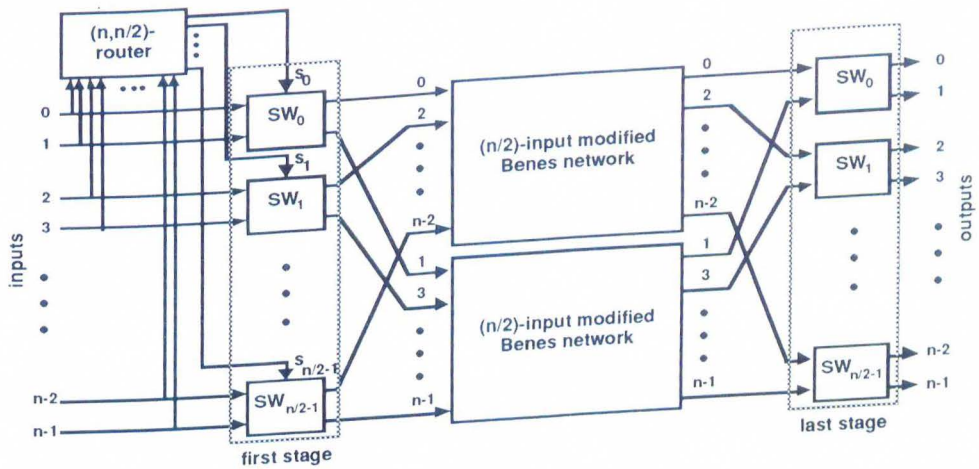



**Figure 12**: The recursive configuration of an $n$-input modified Beneš network.

obvious.

## 3.5 Summary

This chapter has presented an efficient parallel algorithm for routing unicast assignments on Beneš networks. The algorithm generalizes Nassimi and Sahni's parallel algorithm which only routes permutation assignments. The algorithm takes $O(\lg^2 k + \lg n)$ time if each pair of processors are connected by a direct link. On a weaker routing module, where the processors are connected by the extended perfect-shuffle connection, the algorithm takes $O(\lg^4 k + \lg^2 k \lg n)$ time. These times can be further reduced by a factor of $\lg n$ if unicast assignments are pipelined through the stages of the network. For $k << n$, the algorithm even provides the fastest routing time among all unicasting networks.

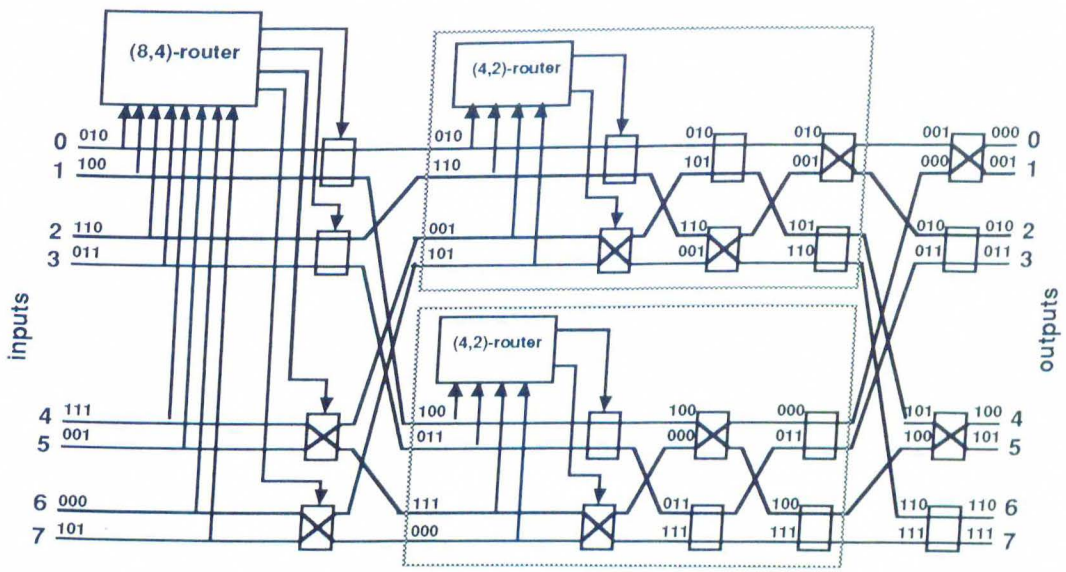



**Figure 13**: The recursively decomposed $n$-input modified Beneš network for $n = 8$.

CHAPTER
FOUR

# EFFICIENT MULTICASTING NETWORKS

## 4.1 Introduction

The Beneš network and its routing algorithms as described in Chapter 3 can only route unicast assignments. In some applications such as teleconferencing and cable broadcasting, messages need not only be routed but also be duplicated as required. Thus, multicasting networks are needed. Multicasting networks are more powerful than unicasting networks in that they can realize multicast as well as unicast assignments. However, as stated in Section 1.2, it still lacks of efficient multicasting networks that can route any multicast assignments (not just complete multicast assignments).

In this chapter, we give a multicasting network which is constructed in terms of simple logic gates and eliminates the need for a parallel computer for routing as the multicasting network given in [NS82b]. Our multicasting network is

recursively constructed by using concentrators as defined in Definition 2.4, and it does not require any sorting as its routing is based on an absolute address decoding scheme. For $n$ inputs, it can be constructed with $O(n \lg^2 n)$ constant fanin logic gates and $O(\lg^2 n)$ bit-level depth, and can realize any multicast assignments in $O(\lg^3 n)$ bit-level time. These complexities either match or are better than the complexities of the multicasting network reported in the literature. Moreover, our multicasting network is input-initiated and is more applicable to the real situations than the output-initiated multicasting networks as in [NS82b]. With the input-initiated property, multicast assignments can be pipelined over our multicasting network, and the average routing time for $O(\lg^2 n)$ multicast assignments can be reduced a $O(\lg^2 n)$ order to $O(\lg n)$ in bit level. Unlike on our multicasting network, multicast assignments are difficult to be pipelined on most of the multicasting networks reported in the literature [NS82b, Hwa72, YM91a, Tho78], mainly because those networks are output-initiated or their switches are set from the outer stages to the inner stages.

The rest of this chapter is organized as follows. Section 4.2 gives a concentrator design that forms the backbone of our multicasting network. Section 4.3 describes the construction of the multicasting network, and Section 4.4 analyzes its performance. The chapter is summarized in Section 4.5.

## 4.2 Concentrator Construction

In this section, we describe a concentrator which is recursively constructed by using another type of networks, called *odd-even splitters*. In routing packets through the splitters and the concentrator, only the connecting bit (as defined

in Subsection 2.1.1) of each packet header is used. For $n$-inputs, our concentrator has $O(n \lg n)$ constant fanin, bit-level logic gates, and $O(\lg n)$ depth and $O(\lg^2 n)$ routing time, both in bit level, where $n$ is a power of 2.

### 4.2.1 Odd-Even Splitter Construction

**Definition 4.1** *An $n$-network is called an $n$-splitter if, for any $k$, $1 \leq k \leq n$, and for any $k$ of its $n$ inputs, it can realize a unicast $k$-assignment that pairs exactly $\lceil k/2 \rceil$ of the $k$ inputs with some $\lceil k/2 \rceil$ outputs in a fixed $n/2$-subset of outputs, and the remaining $\lfloor k/2 \rfloor$ inputs with some $\lfloor k/2 \rfloor$ outputs in the other $n/2$-subset of outputs.* ||

For ease of explaining our network construction, we will use the following narrower version of an $n$-splitter.

**Definition 4.2** *An $n$-splitter is called an odd-even $n$-splitter if the two fixed $n/2$-subsets of outputs in Definition 4.1 comprise odd-numbered outputs and even-numbered outputs, respectively.* ||

Next, we give a construction for the odd-even $n$-splitter.

**A: Odd-Even Splitter**

To construct an odd-even $n$-splitter, we use a stage of $n/2$ $2 \times 2$ switches and an $(n, n/2)$-network, called an $(n, n/2)$-balancer, put in parallel. Figure 14 shows this odd-even $n$-splitter for $n = 16$. Inputs $2i$ and $2i + 1$ of the odd-even $n$-splitter are connected to the inputs of switch $SW_i$, and the setting of $SW_i$ is controlled by the binary input $s_i$, $0 \leq i \leq n/2 - 1$. If $s_i$ is 0, $SW_i$ is set *through*; otherwise, it is set *cross*. The function of the $(n, n/2)$-balancer, whose inputs

are the connecting bits of the incoming packets, is to decide the values of the control inputs for the $n/2$ switches. Let $r'_i$ be the leading bit of the packet at output $i$ after the setting of $SW_i$, $0 \leq i \leq n$. Then, obviously, $r'_{2i} = r_{2i}$ and $r'_{2i+1} = r_{2i+1}$ if $s_i = 0$, and $r'_{2i} = r_{2i+1}$ and $r'_{2i+1} = r_{2i}$ if $s_i = 1$, $0 \leq i \leq n/2 - 1$.

Before we show the design of the balancer, we will first prove that, given $k$ packets at any $k$ busy inputs, $1 \leq k \leq n$, there is a proper setting for $s_i$, $0 \leq i \leq n/2 - 1$, such that some $\lceil k/2 \rceil$ of the $k$ busy inputs are routed onto some $\lceil k/2 \rceil$ even-numbered outputs and the remaining $\lfloor k/2 \rfloor$ busy inputs are routed onto some $\lfloor k/2 \rfloor$ odd-numbered outputs.

**Theorem 4.1** *Let $s_j$'s, $r_i$'s and $r'_i$'s, $0 \leq j \leq n/2 - 1$, $0 \leq i \leq n - 1$, be defined as above. For any $k$ of $r_i$'s set to 1, $1 \leq k \leq n$, there exists a setting for each $s_i$, such that exactly $\lceil k/2 \rceil$ of $r'_{2i}$'s and exactly $\lfloor k/2 \rfloor$ of $r'_{2i+1}$'s are 1, $0 \leq i \leq n/2 - 1$.* ||

**Proof:** For each pair of connecting bits $(r_{2i}, r_{2i+1})$ at switch $SW_i$, $0 \leq i \leq n/2 - 1$, we consider two cases.

*Case 1:* If both $r_{2i}$ and $r_{2i+1}$ are the same, $(r_{2i}, r_{2i+1})$ is called an *identical pair*. For any identical pair $(r_{2i}, r_{2i+1})$, $r'_{2i} = r'_{2i+1}$ regardless of the value of $s_i$. Hence, regardless of how switch $SW_i$ is set, the identical pair $(r_{2i}, r_{2i+1})$ either leaves the number of 1's at even- and odd-numbered outputs unchanged, or it increases the number of 1's at both by 1. We will arbitrarily set $s_i = 0$.

*Case 2:* If $r_{2i}$ and $r_{2i+1}$ are different, $(r_{2i}, r_{2i+1})$ is called a *complementary pair*. First, mark all the complementary pairs as *unmatched*. Then, for a unmatched complementary pair $(r_{2i}, r_{2i+1})$, find another unmatched complementary pair $(r_{2j}, r_{2j+1})$, choose $s_i$ and $s_j$ such that $r'_{2i}$ and $r'_{2j}$ are different (and so are $r'_{2i+1}$

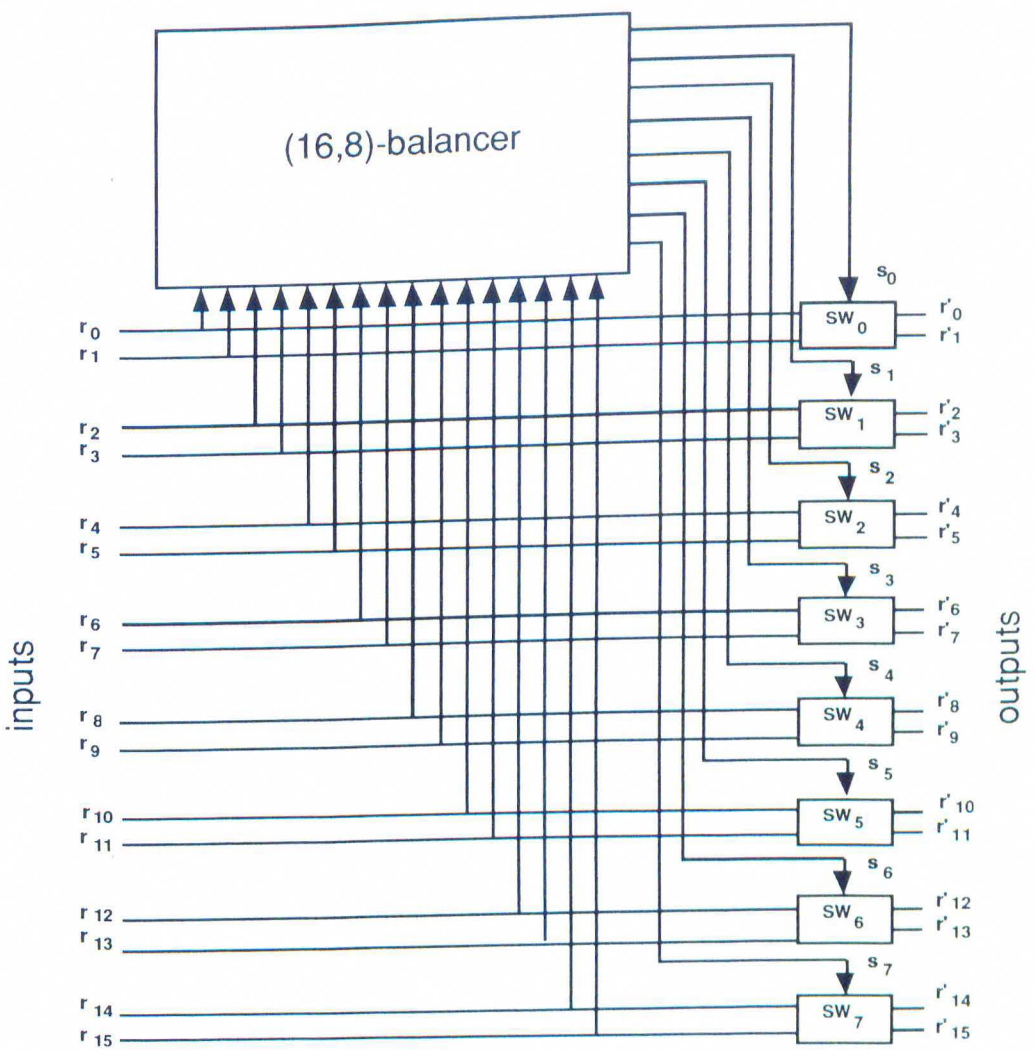



**Figure 14**: An odd-even $n$-splitter for $n = 16$.

and $r'_{2j+1}$), and mark both pairs $(r_{2i}, r_{2i+1})$ and $(r_{2j}, r_{2j+1})$ as *matched*. At the end of this marking process, if $k$ is even, there is no unmatched complementary pair left out, and exactly $k/2$ $r'_{2i}$'s and exactly $k/2$ $r'_{2i+1}$'s will be 1. If $k$ is odd, there must be exactly one unmatched complementary pair $(r_{2i}, r_{2i+1})$ left. In this case, $s_i$ can be chosen such that $r'_{2i} = 1$ and $r'_{2i+1} = 0$, and then exactly $\lceil k/2 \rceil$ $r'_{2i}$'s and exactly $\lfloor k/2 \rfloor$ $r'_{2i+1}$'s will be 1, and the statement follows. ||

**B: Design of Balancer**

The algorithm described in the proof above will be used to construct an $(n, n/2)$-balancer for the odd-even $n$-splitter. In describing this balancer, we will use the terms *identical pair* and *complementary pair* as defined in the proof. Furthermore, a complementary pair $(r_{2i}, r_{2i+1})$ is said to be *zero type* if $r_{2i} = 1$ and $r_{2i+1} = 0$, and to be *one type* if $r_{2i} = 0$ and $r_{2i+1} = 1$.

The $(n, n/2)$-balancer receives $n$ routing bits, $r_0, r_1, \ldots, r_{n-1}$, from which it determines the values of its $n/2$ outputs, $s_0, s_1, \ldots, s_{n/2-1}$. We will use a $\lg n$-level binary tree to implement the $(n, n/2)$-balancer, which is operated by a routing scheme similar to that described in [DO90]. Figure 15 shows this balancer for $n = 16$.

Each routing pair $(r_{2i}, r_{2i+1})$ is connected to a leaf node, denoted LN, at level 0 of the binary tree, $0 \leq i \leq n/2-1$. If $(r_{2i}, r_{2i+1})$ is an identical pair, then the leaf node can decide $s_i$ by itself and set $s_i = 0$. If $(r_{2i}, r_{2i+1})$ is a complementary pair, then the leaf node can not decide how to set $s_i$ and it passes this information to its parent node in the form of a pair of outputs $(z, t_z)$ where $z = 1$ indicates that the leaf node is tied to a complementary routing pair and $t_z$ specifies the type of the complementary pair. If $z = 1$ and $t_z = 0$, then the leaf node has a zero-type complementary pair; and if $z = 1$ and $t_z = 1$, then it has a one-type

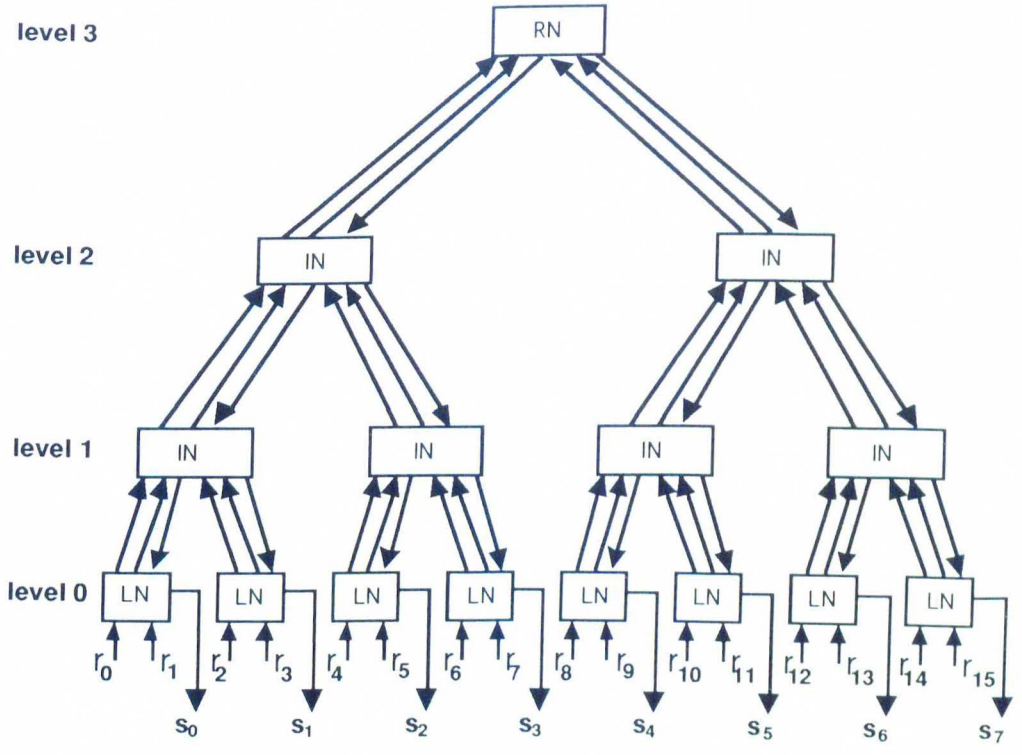


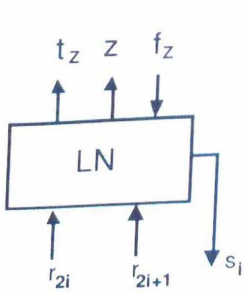

**Figure 15**: The balancer with 16 inputs and 8 outputs.

complementary pair. In both these cases, the leaf node must use its input $f_z$ from its parent node to set its control input. Figure 16(a) and Figure 17(a) show the function and implementation of a leaf node, respectively.

From level 1 to level $\lg n - 2$ of the binary tree, the balancer has intermediate nodes, denoted IN in Figure 15. Each intermediate node receives two pairs of inputs, $(x, t_x)$ and $(y, t_y)$, one from each of its two children. If $x = y = 0$, then none of the descendant leaf nodes of this intermediate node has an unmatched complementary pair, and this node needs not engage in any action. If $x \oplus y = 1$,[1] then exactly one of the descendant leaf nodes of this intermediate node has an unmatched complementary routing pair, and it just passes this information to its parent without any other action. Finally, if $x = y = 1$, exactly two of the descendant leaf nodes of this intermediate node have an unmatched complementary pair, and the intermediate node uses the type information $t_x$ and $t_y$ to set the control inputs of these two leaf nodes. If $t_x \oplus t_y = 1$, i.e., the two complementary pairs are of different types, both control inputs can be set to 0; if $t_x \oplus t_y = 0$, i.e., the two complementary pairs are of the same type, then the control input for the left child is arbitrarily set to 1, and the control input for the right child is set to 0. The function and implementation of an intermediate node are shown in Figure 16(b) and Figure 17(b), respectively.

The root node, RN, at level $\lg n - 1$ receives two pairs of inputs, $(x, t_x)$ and $(y, t_y)$ from its two child nodes. If $x$ and $y$ are the same, then the operation of the root node is the same as that of an intermediate node. If $x$ and $y$ are different, then exactly one of the descendant leaf nodes of the root node has an unmatched complementary pair (i.e., the number of 1's among the connecting

[1] $\oplus$ denotes the binary *EXCLUSIVE OR* operation.

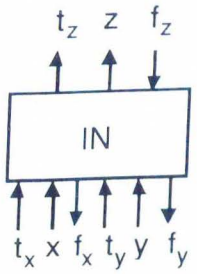



$z = r_{2i} \oplus r_{2i+1}$.

$$t_z = \begin{cases} 1, & \text{if } r_{2i}=0 \text{ and } r_{2i+1}=1; \\ 0, & \text{if } r_{2i}=1 \text{ and } r_{2i+1}=0. \end{cases}$$

$$s_i = \begin{cases} f_z, & \text{if } r_{2i} \oplus r_{2i+1}=1; \\ 0, & \text{otherwise.} \end{cases}$$

(a) Leaf node



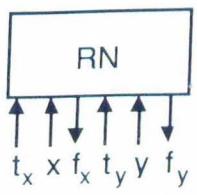

$z = x \oplus y$

$$t_z = \begin{cases} t_x, & \text{if } x=1 \text{ and } y=0; \\ t_y, & \text{if } x=0 \text{ and } y=1; \\ 0, & \text{otherwise.} \end{cases}$$

$$f_x = \begin{cases} f_z, & \text{if } x=1 \text{ and } y=0; \\ 1, & \text{if } x=1, y=1 \text{ and } t_x \oplus t_y=0; \\ 0, & \text{otherwise.} \end{cases}$$

$$f_y = \begin{cases} f_z, & \text{if } x=0 \text{ and } y=1; \\ 0, & \text{otherwise.} \end{cases}$$

(b) Intermediate node

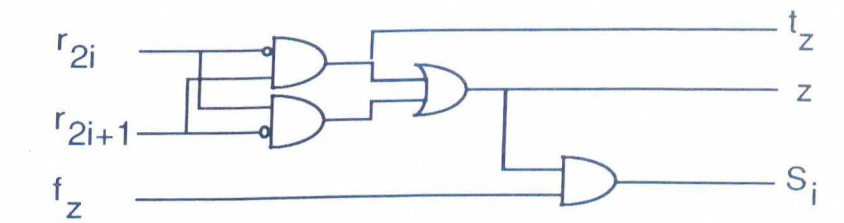



$$f_x = \begin{cases} t_x, & \text{if } x=1 \text{ and } y=0; \\ 1, & \text{if } x=1, y=1 \text{ and } t_x \oplus t_y=0; \\ 0, & \text{otherwise.} \end{cases}$$

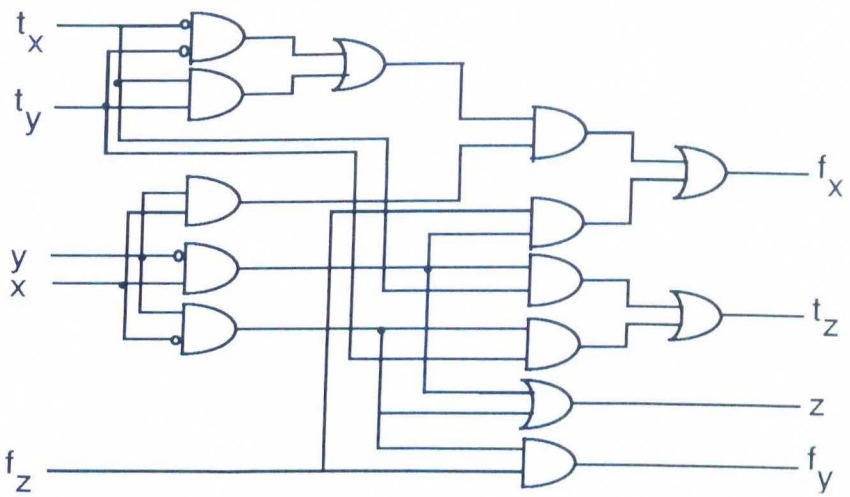
$$f_y = \begin{cases} t_y, & \text{if } x=0 \text{ and } y=1; \\ 0, & \text{otherwise.} \end{cases}$$

(c) Root node

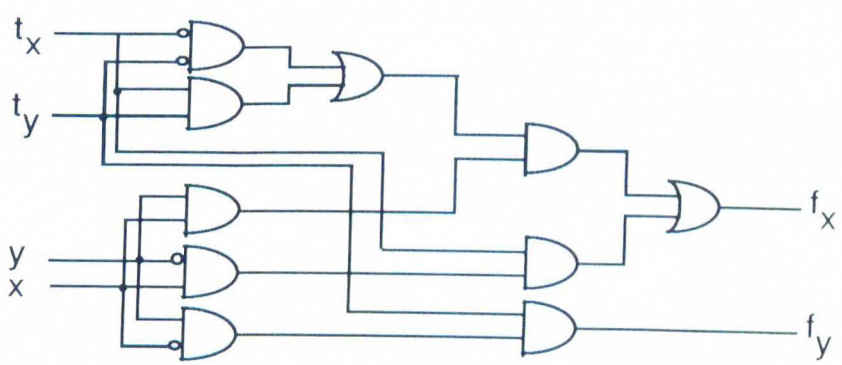
**Figure 16**: The nodes of the balancer and their operations.




(a) Leaf Node




(b) Intermediate Node




(c) Root Node

**Figure 17**: Implementations of the leaf node, intermediate node and root node in terms of logic gates with fanin 2.

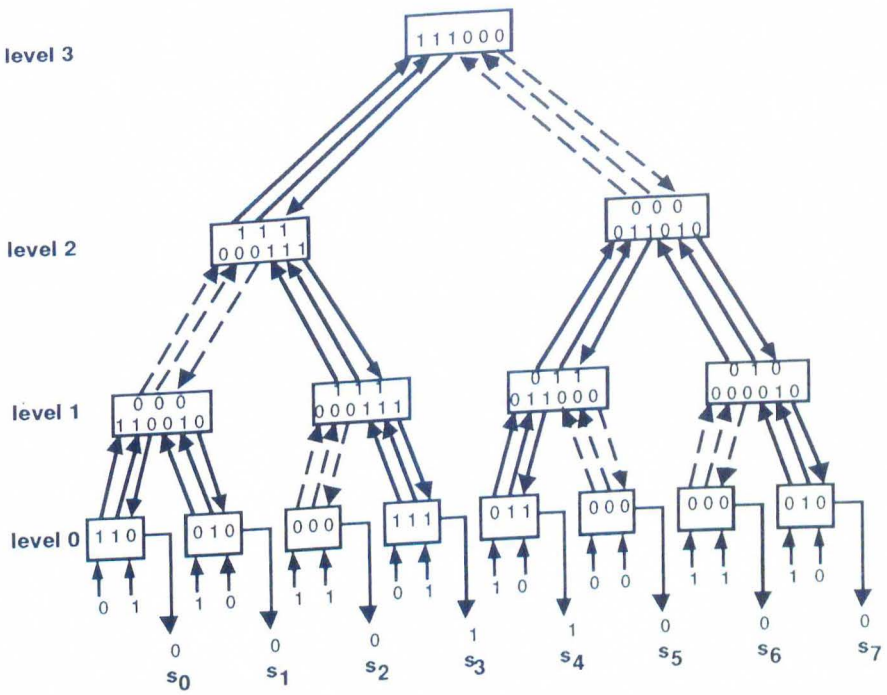



**Figure 18**: Operation of the (16,8)-balancer.

bits in the inputs is odd), and the root node sets the control input of this leaf node according to the type information: it sets the control input to $t_x$ if $x$ is 1, and to $t_y$ if $y$ is 1. The function and implementation of the root node are shown in Figure 16(c) and Figure 17(c), respectively. .

All these steps are illustrated in an example in Figure 18 for a balancer with 16 inputs and 8 outputs, where the dashed lines between the nodes indicate that the unused arcs for the given pattern of connecting bits.

### 4.2.2 Concentrator Construction

Using the odd-even splitters constructed in the previous subsection, an $n$-concentrator is recursively constructed as shown in Figure 19, where $n$ is a power of 2.

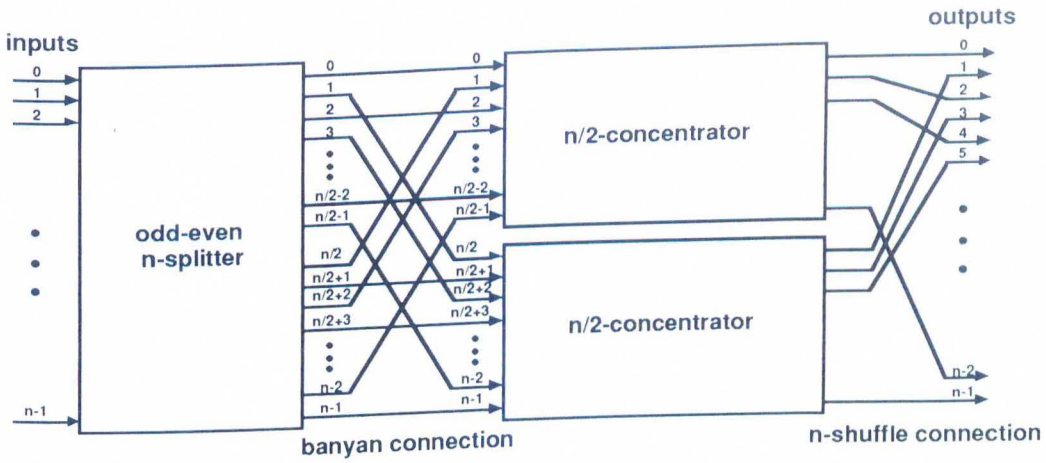



**Figure 19**: The recursive construction of an $n$-concentrator.

**Theorem 4.2** *The $n$-network shown in Figure 19 is an $n$-concentrator.*

**Proof:** Given $k$ packets at any $k$ busy inputs, the odd-even $n$-splitter can simultaneously connect some $\lceil k/2 \rceil$ of the $k$ packets onto some $\lceil k/2 \rceil$ even-numbered outputs, and the remaining $\lfloor k/2 \rfloor$ packets onto some $\lfloor k/2 \rfloor$ odd-numbered outputs. Because the outputs of the $n$-splitter are followed by a banyan connection, the $\lceil k/2 \rceil$ packets at its even-numbered outputs are routed to the inputs of the upper $n/2$-concentrator, and the other $\lfloor k/2 \rfloor$ packets at its odd-numbered outputs are routed to the inputs of the lower $n/2$-concentrator. By induction, the two $n/2$-concentrators can route these $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$ packets onto their first $\lceil k/2 \rceil$ and first $\lfloor k/2 \rfloor$ outputs, respectively. The $n$-shuffle connection, which follows the two $n/2$-concentrators, then moves these $k$ packets into the first $k$ of the $n$ outputs of the $n$-concentrator. ||

The concentrator in Figure 19 can be recursively decomposed into a network consisting of splitters only. Such a decomposition is shown in Figure 20 for $n = 16$. We note that the $(2,2)$-splitters in the last stage do not need balancers since they can be set directly by using the connecting bits they receive. We

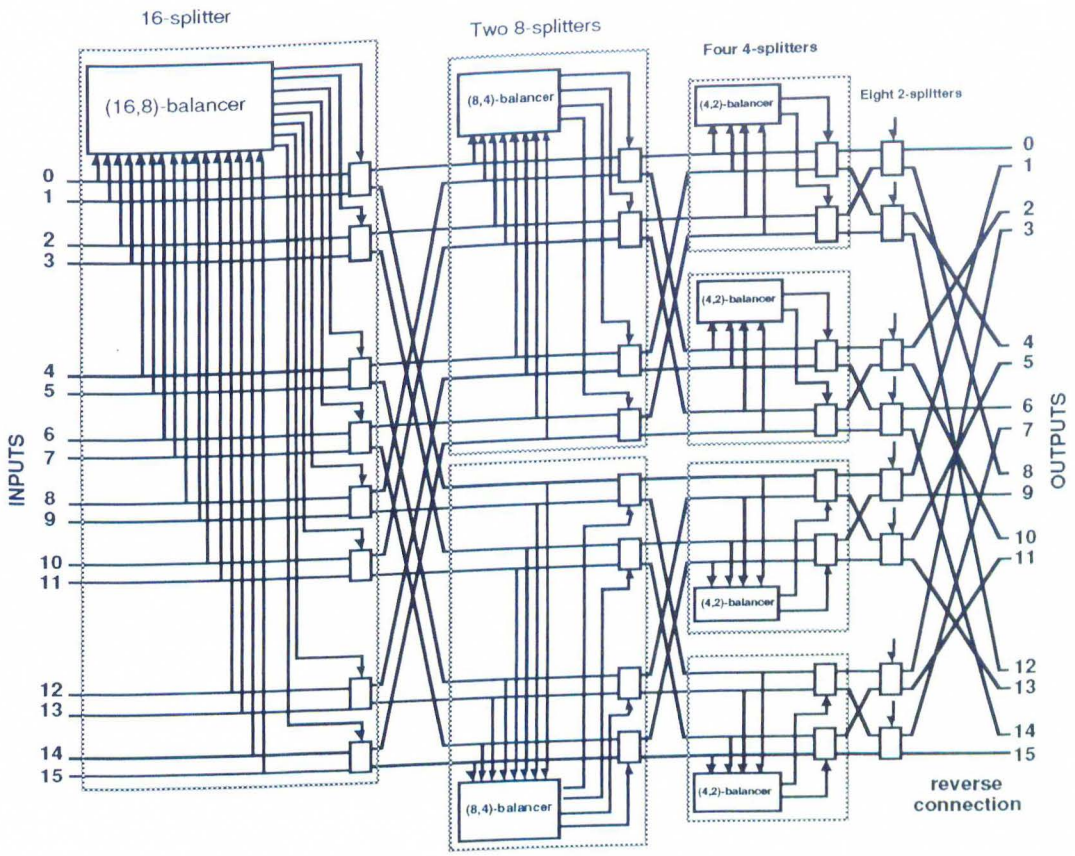



**Figure 20**: An fully decomposed $n$-concentrator for $n = 16$.

also note that the fixed connection following the last stage of $2 \times 2$ switches is a reverse connection which results from the composition of the cascade of shuffle connections that are generated in the recursive decomposition of the 16-concentrator.

## 4.3 The Multicasting Network

Given the $n$-concentrator just described, we proceed with the construction of a multicasting network in this section.

### 4.3.1 Multicasting Network Construction

The multicasting network is shown in Figure 21. It is constructed recursively where the non-recursive part consists of a distributor stage and two $(n, n/2)$-concentrators. The distributor is an array of $n$ $(1,2)$-copiers where each $(1,2)$-copier can map its input to none, any one or both of its two outputs, and the concentrators can be realized by the construction given in the previous section. We first establish that this network is a multicasting $n$-network.

**Theorem 4.3** *The network shown in Figure 21 is a multicasting $n$-network.*

**Proof:** Consider an arbitrary, but fixed multicast assignment of the inputs to the outputs. If an input in this assignment is paired with an output in the first half of outputs then route it to the upper $(n, n/2)$-concentrator, if it is paired with an output in the second half of outputs then route it to the lower $(n, n/2)$-concentrator; and if it is paired with outputs in both half of outputs then route it to both concentrators. Now, the point is that no matter how the multicast assignment is fixed, each concentrator can receive no more than $n/2$ packets, since otherwise an output must be assigned to more than one input which is impossible. Thus, the $(n, n/2)$-concentrators can route the packets they receive to the inputs of the multicasting $(n/2)$-networks, and the statement follows by induction. ||

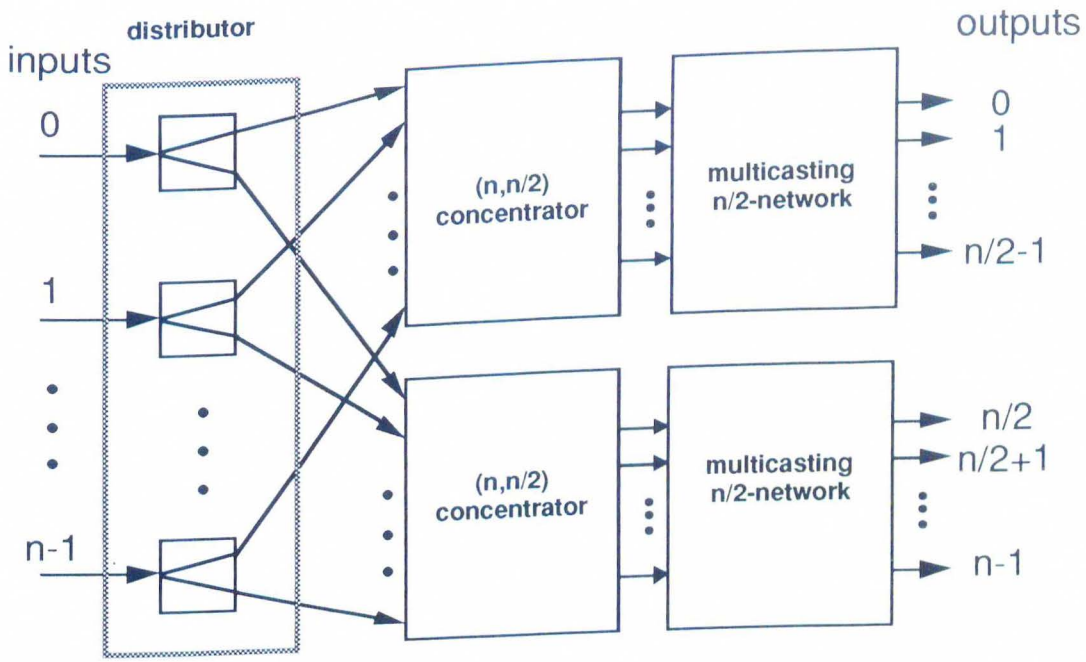



**Figure 21**: The recursive construction of a multicasting $n$-network.

Unlike the previously reported multicasting networks, the multicasting network given here does not use any dedicated copy network. Rather, it generates the copies of the packets as they are being routed to their destinations. As we shall see subsequently, this approach reduces the amount of destination address information that each input must use in order to route its packet to the outputs it desires.

### 4.3.2 Routing Multicast Assignments

In this subsection, we show how multicast assignments are routed through the previous multicasting $n$-network by using $O(n)$ bits of routing information per input.

**Definition 4.3** *For a given multicast assignment, a multicast pattern for an input $i$ (or for a packet at that input) in a multicasting $n$-network is a binary sequence $(a_0, a_1, \ldots, a_{n-1})$, where $a_j = 1$ if and only if input $i$ is paired with output $j$.* ||

**Proposition 4.1:** In a multicasting $n$-network, any given input can have any one of $2^n$ multicast patterns.

**Proof:** It follows from Definition 4.3. ||

**Proposition 4.2:** If an input in a multicasting $n$-network must specify the outputs to which it is assigned independently of other inputs, then it must use at least $n$ bits.

**Proof:** By Proposition 4.1, an input may have any one of $2^n$ multicast patterns. If the input is to specify any one of these patterns without relying on the specifications of the other inputs, then it obviously needs $\lg 2^n = n$ bits. ||

We note that the independence assumption used here is very much like assuming that each input must use at least $\lg n$ bits to specify the output to which it is assigned in a unicasting $n$-network where each input can be paired with at most one output. We also note that if the independence assumption is relaxed, then it is possible that all $(n+1)^n$ assignments for a multicasting $n$-network can be coded only with $\lg(n+1)^n = O(n \lg n)$ bits. In fact, this coding scheme can be enforced if the assignments are specified from outputs to inputs, since each output can only be connected to at most one input, and thus $O(\lg n)$ bits of source address per output will suffice. However, in the multicasting network construction given here, it is assumed that the inputs are the active ports and initiate the connection requests. In this case, we do not know of any coding scheme that works with $O(\lg n)$ destination bits per input to

specify any multicast $n$-assignment although some multicast assignments can be indeed be specified by $O(\lg n)$ destination bits per input. For example, the multicast $n$-assignments for which each input is paired with a set of consecutively numbered outputs can be specified by $2\lg n$ destination bits per input with the first $\lg n$ bits specifying the number of outputs paired with an input and the second $\lg n$ bits specifying the smallest numbered output. Therefore, we assume that the inputs independently specify their multicast patterns by using $O(n)$ bits per input. We will describe two destination coding schemes, both using $O(n)$ destination bits per input, to realize multicast assignments.

First, we establish the following propositions which will be used in the two destination coding schemes. Let $(a_0, a_1, \ldots, a_{n-1})$ be a multicast pattern of a packet at an input of the multicasting $n$-network as shown in Figure 21. It is obvious that the packet is routed to the first half of outputs if and only if the elements $a_0, a_1, \ldots, a_{n/2-1}$ are not all zero, or equivalently, $a_0 + a_1 + \cdots + a_{n/2-1} = 1$,[2] and, it is routed to the second half of outputs if and only if $a_{n/2} + a_{n/2+1} + \cdots + a_{n-1} = 1$. More generally, the following proposition holds.

**Proposition 4.3:** Suppose that the outputs in the network of Figure 21 are partitioned into $2^k$ groups of $n/2^k$ consecutive outputs from top to bottom, $1 \leq k \leq \lg n$. Then, a packet at an input whose multicast pattern is $(a_0, a_1, \ldots, a_{n-1})$ is routed to the $i$th group of outputs, $0 \leq i \leq 2^k - 1$, if and only if at least one of the $a_j$, $in/2^k \leq j < (i+1)n/2^k$ is 1. ||

Define a binary variable $d_i^k$ on $k$ and $i$, $0 \leq i \leq 2^k - 1$, as $d_i^k = 1$ if and only if at least one of the $a_j$, $in/2^k \leq j < (i+1)n/2^k$ is 1. Let $O_i^k$ denote the $i$th

[2]Here + denotes the binary $OR$ operation.

group of outputs defined in Proposition 4.3. We have the following equivalent proposition.

**Proposition 4.4:** Suppose that a packet at an input of the multicasting $n$-network shown in Figure 21 has a multicast pattern $(a_0, a_1, \ldots, a_{n-1})$. Let $d_i^k$ and $O_i^k$ be defined as above, $0 \leq i \leq 2^k - 1$, $1 \leq k \leq \lg n$. Then, the packet would be routed to $O_i^k$ if and only if $d_i^k = 1$. ||

To route a packet with a multicast pattern $(a_0, a_1, \ldots, a_{n-1})$ through the multicasting $n$-network, the $(1,2)$-copiers along the path(s) of the packet can use $d_i^k$'s to decide their settings. When it is recursively decomposed, the multicasting $n$-network has $\lg n$ distributor stages. If these distributor stages are numbered $1, 2, \ldots, \lg n$, from left to right, then stage $k$ has $2^{k-1}$ distributors, numbered $0, 1, \ldots, 2^{k-1} - 1$ from top to bottom, $1 \leq k \leq \lg n$. Each $(1,2)$-copier in the $i$th distributor at stage $k$ uses the pair $(d_{2i}^k, d_{2i+1}^k)$ it receives to decide its own setting as described subsequently.

### A: Routing with $2n - 2$ Destination Bits

In the first coding scheme, each packet at an input is given a $(2n-2)$-bit destination address $(d_0^1, d_1^1, d_0^2, d_1^2, d_2^2, d_3^2, \ldots, d_0^{\lg n}, d_1^{\lg n}, \ldots, d_{n-1}^{\lg n})$. These $2n-2$ destination bits are defined as in Proposition 4.4. Figure 22 shows an example of this coding scheme for $n = 8$. Given such a coding of destination addresses, routing multicast assignments is straightforward. For each value of $k$, $d_i^k$, $0 \leq i \leq 2^k - 1$, specify $2^k$ destination bits used by the distributors at stage $k$. If a $(1,2)$-copier in the $i$th distributor at stage $k$ is on a path of a packet, then it uses $d_{2i}^k$ and $d_{2i+1}^k$ that it receives to route that packet, $0 \leq i \leq 2^{k-1} - 1$, $1 \leq k \leq \lg n$. If $(d_{2i}^k, d_{2i+1}^k) = (0,1)$ then that packet is routed to the lower output of the copier, if $(d_{2i}^k, d_{2i+1}^k) = (1,0)$ then it is routed to the upper

| input | Multicast Pattern $a_0a_1a_2a_3a_4a_5a_6a_7$ | Destination Code $d_0^1d_1^1$ $d_0^2d_1^2d_2^2d_3^2$ $d_0^3d_1^3d_2^3d_3^3d_4^3d_5^3d_6^3d_7^3$ |
|---|---|---|
| 0 | 10000010 | 11/1001/10000010 |
| 1 | 00000000 | 00/0000/00000000 |
| 2 | 00000100 | 01/0010/00000100 |
| 3 | 01110000 | 10/1100/01110000 |
| 4 | 00000000 | 00/0000/00000000 |
| 5 | 00001000 | 01/0010/00001000 |
| 6 | 00000001 | 01/0001/00000001 |
| 7 | 00000000 | 00/0000/00000000 |

**Figure 22**: Destination coding for multicast routing.

output, and if $(d_{2i}^k, d_{2i+1}^k) = (1,1)$ then it is routed to both outputs. Along with that packet, the copier also routes a valid destination code for the remaining stages on the path(s) of that packet. The determination of the destination code depends on the output(s) that that packet is routed to. If the packet is routed to the upper output of the copier, then the first half of the destination code for each of the remaining stages is retained, and the second half is discarded, and if it is routed to the lower output, then the second half of the destination code for each of the remaining stages is retained and the first half is discarded. Therefore, at all distributor stages, the leftmost two bits of the destination code that arrives with a packet at a $(1,2)$-copier are used to decide the setting of the copier.

This routing scheme is illustrated in Figure 23 by using the multicast pattern for input 3 shown in Figure 22 as an axample, where input 3 is routed to outputs 1, 2 and 3, and only the first four outputs are depicted. The $(1,2)$-copiers (CPs) that are adjacent in the tree are separated by a concentrator stage which is not shown. A more complete description of the routing scheme

for a multicasting 8-network, including the concentrator stages, is shown in Figure 24. The multicasting network is obtained by decomposing the two multicasting 4-networks based on the construction given in Figure 22. The multicast assignment that is realized is the one given in Figure 23. It is assumed that, in routing their busy inputs, the concentrators operate as described in Section 4.2.

**B: Routing with $n$ Destination Bits**

The length of the destination code can be reduced from $2n-2$ to $n$, by increasing the complexity of routing at each copier node. In this case, the packet at each input carries its own $n$-bit multicast pattern $(a_0, a_1, \ldots, a_{n-1})$ as its destination address. The $(1,2)$-copiers must rely on more bits in order to determine which way to route the packets they receive. More specifically, the copiers in the $k$th distributor stage must examine $n/2^{k-1}$ bits, $1 \leq k \leq \lg n$, to determine their settings. Other than this, this multicast pattern coding scheme is very similar to the previous one, and its details are omitted.

**Remark 4.1** *We note that using $O(n)$ bits per input to encode multicast assignments is quite efficient as compared to the coding schemes used in multicasting networks that rely on separate copy networks [Lee88]. Those destination coding schemes require $O(n \lg n)$ bits to specify a multicast pattern for each input, or $O(n^2 \lg n)$ for all the inputs as compared to $O(n^2)$ bits of our coding schemes.* ||

## 4.4 Performance Analysis

In this section, we determine the cost, depth and routing time of the multicasting network described in the previous section. As before, we continue to

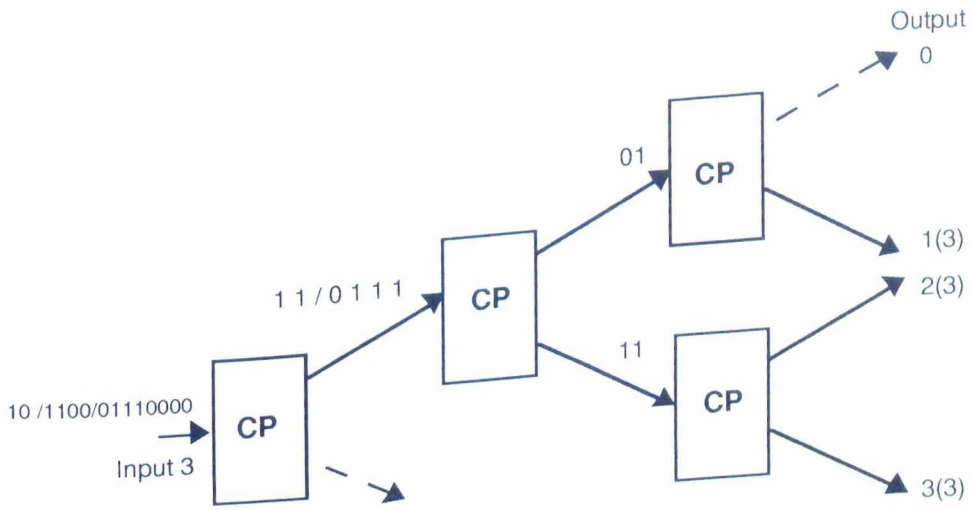



**Figure 23**: Illustration of the multicast routing scheme.

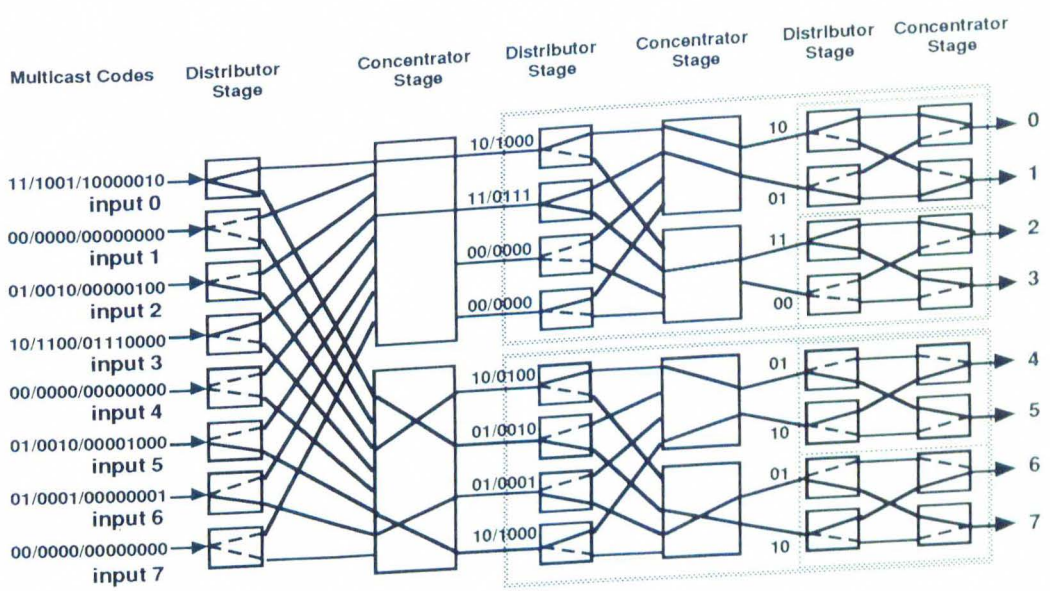



**Figure 24**: A multicasting 8-network shown to realize the multicast assignment given in Figure 22.

assume in this section that all our networks have a power of two number of inputs and outputs.

First, we compute the cost, depth, and routing time, $C_{conc}(n), D_{conc}(n), T_{conc}(n)$ for the $n$-concentrator. As described in Section 4.2, this $n$-concentrator is recursively constructed by using an odd-even $n$-splitter from which we have

$$\begin{aligned} C_{conc}(n) &= C_{split}(n) + 2C_{conc}(n/2), \\ D_{conc}(n) &= D_{split}(n) + D_{conc}(n/2), \\ T_{conc}(n) &= T_{split}(n) + T_{conc}(n/2), \end{aligned}$$

where $C_{split}(n), D_{split}(n), T_{split}(n)$ denote the cost, depth, and routing time of the odd-even $n$-splitter, in that order. The odd-even $n$-splitter is composed of an array of $n/2$ $2 \times 2$ switches and an $(n, n/2)$-balancer, where the balancer is used to set the switches. Once the switches are set, the packets can pass through the switches. Hence, the routing time of the odd-even $n$-splitter is decided by the routing time of the balancer, and the depth of the splitter is the depth of a $2 \times 2$ switch. Assuming that a $2 \times 2$ switch can be implemented by using six bit-level logic gates with fanin 2, and depth 2, we have

$$\begin{aligned} C_{split}(n) &= 3n + C_{balancer}(n), \\ D_{split}(n) &= 2, \\ T_{split}(n) &= T_{balancer}(n), \end{aligned}$$

where $C_{balancer}(n), T_{balancer}(n)$ denote the cost and routing time of the balancer, respectively. The $(n, n/2)$-balancer has $n-1$ nodes and the longest path from an input to an output is $2 \lg n$. As shown in Figure 17, each node in the balancer can be implemented by no more than 14 bit-level logic gates with fanin 2, and within any node, the largest number of 2-input logic gates along a path

from an input to an output is 4. Hence, $C_{split}(n) \leq 17n$ and $T_{split}(n) \leq 8 \lg n$. Substituting all these expressions into the cost, depth and routing time recurrences for the $n$-concentrator, and solving the recurrences with $C_{conc}(2) = 6$, $D_{conc}(2) = 2$, and $T_{conc}(2) = 1$, we find

$$\begin{aligned} C_{conc}(n) &\leq 17n \lg n - 14n, \\ D_{conc}(n) &= 2 \lg n, \\ T_{conc}(n) &\leq 4 \lg^2 n + 4 \lg n. \end{aligned}$$

We note that for $n = 2$, the $n$-concentrator reduces to a $2 \times 2$ switch from which the values for $C_{conc}(2), D_{conc}(2)$, and $T_{conc}(2)$ follow.

With these expressions, we can now compute the cost, depth, and routing time of our multicasting $n$-network. Let $C_{multi}(n), D_{multi}(n)$, and $T_{multi}(n)$ denote its cost, depth and routing time. As shown in Figure 21, the multicasting $n$-network is recursively constructed by using an $(n, 2n)$-distributor and two $(n, n/2)$-concentrators. Hence,

$$\begin{aligned} C_{multi}(n) &= C_{distr}(n) + 2C_{conc}(n) + 2C_{multi}(n/2), \\ D_{multi}(n) &= D_{distr}(n) + D_{conc}(n) + D_{multi}(n/2), \\ T_{multi}(n) &= T_{distr}(n) + T_{conc}(n) + T_{multi}(n/2). \end{aligned}$$

The $(n, 2n)$-distributor is an array of $n$ $(1, 2)$-copiers and each $(1, 2)$-copier can be implemented by two logic gates of fanin 2. Hence, $C_{distr}(n) = 2n, D_{distr}(n) = 1$ and $T_{distr}(n) = 1$. Substituting these, and the expressions for $C_{conc}(n), D_{conc}(n)$, and $T_{conc}(n)$ into the above e recurrences, and solving them with $C_{multi}(2) = D_{multi}(2) = T_{multi}(2) = 2$, we have

$$C_{multi}(n) \leq 17n \lg^2 n,$$

$$
\begin{aligned}
D_{multi}(n) &= lg^2 n + 2\lg n + 1, \\
T_{multi}(n) &\leq 4/3\lg^3 n + 4\lg^2 n + 11/3\lg n.
\end{aligned}
$$

The routing time given above can be further reduced by pipelining consecutive multicast assignments over the multicasting $n$-network. Suppose that there are $\alpha$ consecutive multicast assignments to be routed through the multicasting $n$-network. An assignment can enter a stage after the previous assignment leaving that stage, and hence routings in different stages can be overlapped. From the previous calculation, the odd-even $n$-splitter needs $O(\lg n)$ routing time, and is the "bottle-neck stage" of this multicasting $n$-network. Thus, it takes $O(\lg^3 n + (\alpha - 1)\lg n)$ time to route these $\alpha$ consecutive multicast assignments, and the routing time per assignment is reduced to $O(\lg n)$ if $\alpha \geq \lg^2 n$.

## 4.5 Summary

In this chapter, we presented a multicasting network. Unlike the previously reported multicasting networks, this network has a very simple design, and can be routed very fast. Including the routing cost, it has $O(n\lg^2 n)$ cost, $O(\lg^2 n)$ depth, and $O(\lg^3 n)$ routing time without pipelining, and $O(\lg n)$ routing time with pipelining, all in bit level. These complexities compare favorably with those of the best generalized connectors reported earlier.

While this chapter did not explicitly treat unicast assignments, those can also be realized by the multicasting $n$-network described. If the multicasting $n$-network is sought to realize only unicast assignments, then the distributor design and the destination coding can both be simplified (only $\lg n$ destination bits per input is sufficient). This, however, does not reduce the cost, depth and routing time complexities of the network.

This chapter also gave a construction for an $n$-concentrator that has $O(n \lg^2 n)$ cost, $O(\lg^2 n)$ depth, and $O(\lg^2 n)$ routing time without pipelining, and $O(\lg n)$ routing time with pipelining, all in bit level. If these complexities can be reduced, as of the networks to be constructed in the next chapter, then the complexities of the multicasting $n$-network can also be scaled down by the same factor of reduction.

CHAPTER
FIVE

# EXPANDERS, BOUNDED CONCENTRATORS AND SUPERCONCENTRATORS

## 5.1 Introduction

The cost of the multicasting network described in Chapter 4 can be reduced from $O(n \lg^2 n)$ to $O(n \lg n)$ if the concentrators can be constructed with $O(n)$ instead of $O(n \lg n)$ cost. One approach to construct such $O(n)$-cost concentrators is to employ so called linear-size expanders. The key step to accomplish this is to construct expanders with large enough expansion coefficients. Although linear-size expanders have explicitly been constructed before [Mar73, GG81, AM85, Alo86, NAM87, JM87], their expansion coefficients are quite small (0.466 for the expander in [NAM87, JM87]).

In this chapter, we investigate the possibility of explicit constructions of linear-size expanders with larger expansion coefficients than those of the previously

constructed expanders. Specifically, we construct a new family of linear-size expanders with density 33 and expansion coefficient 0.626 when $\alpha = 1$ and 0.868 when $\alpha = 0.5$. These expanders are obtained by using the method of Alon et al. [NAM87] and Jimbo and Maruoka [JM87] that ties the expansion coefficient of an expander to the second largest eigenvalue of a matrix representation of that expander. Using these expanders, we also obtain a linear-size bounded concentrator with density 25.5 and a linear-size superconcentrator with density 208.

The rest of this chapter is organized as follows. Section 5.2 gives the construction of our expander, and determines its expansion coefficient. Section 5.3 outlines how our expander can be used to obtain linear-size concentrators and superconcentrators. Section 5.4 summarizes this chapter.

## 5.2 Linear-Size Expander Construction

In this section, we will construct a linear-size $n$-input expander with density 33 where $n = m^2$ for any positive integer $m$. This construction follows the original formulation given by Margulis [Mar73] and the analysis in [Alo86, JM87].

### 5.2.1 Expander Construction

Let $Z_m = \{0, 1, \ldots, m-1\}$ denote the modulo–$m$ integer ring with the usual modulo addition and multiplication operations, and let $A_m = Z_m \times Z_m$ where $\times$ denotes the Cartesian product. Define the following permutations on $A_m$:

$$\begin{aligned} \pi_1(x, y) &= (x + 6y, y), \\ \pi_2(x, y) &= (x + 6y, y - 1), \end{aligned}$$

$$
\begin{aligned}
\pi_3(x,y) &= (x+6y, y+1),\\
\pi_4(x,y) &= (x+6y, y+2),\\
\pi_5(x,y) &= (x+2y, -2x-3y),\\
\pi_6(x,y) &= (x+2y, -2x-3y-1),\\
\pi_7(x,y) &= (x+2y, -2x-3y+1),\\
\pi_8(x,y) &= (x+2y, -2x-3y+2),\\
\pi_9(x,y) &= (x, 6x+y),\\
\pi_{10}(x,y) &= (x-1, 6x+y),\\
\pi_{11}(x,y) &= (x+1, 6x+y),\\
\pi_{12}(x,y) &= (x+2, 6x+y),\\
\pi_{13}(x,y) &= (x-2y, 2x-3y),\\
\pi_{14}(x,y) &= (x-2y-1, 2x-3y),\\
\pi_{15}(x,y) &= (x-2y+1, 2x-3y),\\
\pi_{16}(x,y) &= (x-2y+2, 2x-3y),
\end{aligned}
$$

for any $(x,y) \in A_m$, where the arithmetic operations are defined over $Z_m$. It is easy to verify that $\pi_i$, $1 \leq i \leq 16$, are permutations over $Z_m \times Z_m$.

Now define a 32-regular graph $G = (V, E)$ on the vertex set $V = A_m$ by joining each vertex $(x,y) \in V$ to vertices $\pi_i(x,y)$ and $\pi_i^{-1}(x,y)$, for $i = 1, 2, \ldots, 16$. Then, based on $G$, construct a bipartite graph $G = (I, O, E)$ with the set of inputs $I = V$ and the set of outputs $O = V$ such that, for any $(x,y) \in I$ and any $(x',y') \in O$, $((x,y),(x',y')) \in E$ if and only if $(x,y) = (x',y')$ or $((x,y),(x',y'))$ is an edge of $G$. Obviously, $G$ has no more than $33n$ edges.

Applying the results of Alon et al, we have the following theorem.

**Theorem 5.1** *Given $G$ and $\bar{G}$ defined above, let $A_G$ be the adjacency matrix of $G$ and let $\lambda$ be the second largest eigenvalue of the matrix $A = (1/32)A_G$. Then, $\bar{G}$ is an $(n, 33, c_\alpha, \alpha)$-expander with*

$$c_\alpha = \frac{8(1-\lambda)}{\lambda + 4\alpha(1-\lambda) + \sqrt{\lambda^2 + 8\alpha(1-\lambda)(2-\lambda)}}$$

*for any $\alpha, 0 \leq \alpha \leq 1$.*

**Proof:** See Theorem 2.1 to Corollary 2.3 in [NAM87]. ||

For a fixed $\alpha$, it is not too difficult to check that $c_\alpha$ is a decreasing function of $\lambda$ so that an upper bound on $\lambda$ will yield a lower bound on the expansion coefficient $c_\alpha$. In the next subsection, we derive an upper bound on $\lambda$, and then calculate the expansion coefficient of this expander.

### 5.2.2 Expansion Coefficient Calculation

For notational convenience, we shall assume that each pair in $A_m$ as a two-element column vector. Given $n = m^2$, let $N = \{0, 1, \ldots, n-1\}$. Define an automorphism $\nu$ between $N$ and $A_m$ such that, $\forall j \in N, \nu(j) = \begin{pmatrix} j_1 \\ j_2 \end{pmatrix} \in A_m$, where $j = j_1 + mj_2, 0 \leq j_1, j_2 \leq m-1$, (i.e., $(j_1, j_2)$ is an $m$-ary code of $j$). Hence, for any permutation $\pi_i$ on $A_m, i = 1, 2, \ldots, 16$, there is a counterpart permutation $\sigma_i$ on $N$ defined by $\sigma_i(j) = \nu^{-1}\pi_i\nu(j) = \nu^{-1}(\pi_i(\nu(j))), \forall j \in N$. For each permutation $\sigma_i$, we have a permutation matrix $M_{\sigma_i} = [m^i_{j,k}]$ whose entries are given by $m^i_{j,k} = \delta(\pi_i\nu(j), \nu(k)), 0 \leq j, k \leq n-1$, where $\delta(\cdot, \cdot)$ is the well-known Kronecker's delta function. Let $M^t_{\sigma_i}$ denote the transpose matrix of $M_{\sigma_i}$. It is not difficult to verify that $M^t_{\sigma_i}$ is the permutation matrix associated

with the inverse permutation $\sigma_i^{-1} = \nu^{-1}\pi_i^{-1}\nu$. Therefore, $A_G$ (the adjacency matrix of $G$) is the summation of these 32 permutation matrices, and matrix $A = (1/32)A_G$ is given by

$$\begin{aligned} A &= \frac{1}{32}\sum_{i=1}^{16}\left\{M_{\sigma_i}^t + M_{\sigma_i}\right\} \\ &= A_1 + A_2, \quad \text{where } A_1 = \frac{1}{32}\sum_{i=1}^{16} M_{\sigma_i}^t \text{ and } A_2 = A_1^t. \end{aligned}$$

In the construction of the expander, we use two types of permutations on $A_m$, namely *linear permutations* and *affine permutations*. The linear permutations are expressed by multiplying a $2\times 2$ invertible matrix over $Z_m$ with each element of $A_m$, and the affine permutations are expressed by adding a constant vector to linear permutations. More specifically, we use the following matrices and vectors:

$$B_1 = B_2^{-1} = \begin{bmatrix} 1 & 6 \\ 0 & 1 \end{bmatrix}, \quad B_3 = B_4^{-1} = \begin{bmatrix} 1 & 2 \\ -2 & -3 \end{bmatrix}$$

$$B_5 = B_6^{-1} = \begin{bmatrix} 1 & 0 \\ 6 & 1 \end{bmatrix}, \quad B_7 = B_8^{-1} = \begin{bmatrix} 1 & -2 \\ 2 & -3 \end{bmatrix}$$

$$a_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad a_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

In terms of these matrices and vectors, the permutations on $N$ are rewritten as:

$$\begin{aligned} \sigma_1 &= \nu^{-1}B_1\nu, & \sigma_2 &= \nu^{-1}(-a_2 + B_1\nu), \\ \sigma_3 &= \nu^{-1}(a_2 + B_1\nu), & \sigma_4 &= \nu^{-1}(2a_2 + B_1\nu), \\ \sigma_5 &= \nu^{-1}B_3\nu, & \sigma_6 &= \nu^{-1}(-a_2 + B_3\nu), \\ \sigma_7 &= \nu^{-1}(a_2 + B_3\nu), & \sigma_8 &= \nu^{-1}(2a_2 + B_3\nu), \end{aligned}$$

$$\sigma_9 = \nu^{-1}B_5\nu, \quad \sigma_{10} = \nu^{-1}(-a_1 + B_5\nu),$$
$$\sigma_{11} = \nu^{-1}(a_1 + B_5\nu), \quad \sigma_{12} = \nu^{-1}(2a_1 + B_5\nu),$$
$$\sigma_{13} = \nu^{-1}B_7\nu, \quad \sigma_{14} = \nu^{-1}(-a_1 + B_7\nu),$$
$$\sigma_{15} = \nu^{-1}(a_1 + B_7\nu), \quad \sigma_{16} = \nu^{-1}(2a_1 + B_7\nu).$$

Now, define a unitary matrix $\Omega = [\omega_{j,k}]_{n\times n}$ over the complex numbers by $\omega_{j,k} = (1/m)\omega^{<\nu(j),\nu(k)>}$, $0 \le j,k \le n-1$, where $\omega = exp[(2\pi/m)\sqrt{-1}]$ and $<,>$ denotes the inner product of vectors. Because $A$ is symmetric and the sum of entries in any row or any column of $A$ is 1, we know that

$$\Omega^* A\Omega = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & H & \\ 0 & & & \end{bmatrix} = \hat{A}_1 + \hat{A}_2$$

where $H = [h_{j,k}]_{(n-1)\times(n-1)}$ is a Hermitian matrix, $\hat{A}_1 = \Omega^* A_1\Omega$ and $\hat{A}_2 = \Omega^* A_2\Omega$. The largest eigenvalue of $A$ is 1 since the sum of entries in each row or each column of $A$ is 1, and hence the second largest eigenvalue of $A$ is equal to the largest eigenvalue of $H$.

Subsequently, we specify each entry of $H$ and then to determine an upper bound on the largest eigenvalue of $H$. First, we need the following two lemmas of Jimbo and Maruoka [JM87]:

**Lemma 1:** If $\pi$ is a linear permutation on $A_m$, i.e. if there exists some $2\times 2$, invertible matrix $B$ over $Z_m$ such that the counterpart permutation $\sigma = \nu^{-1}\pi\nu$ of $\pi$ is given by $\sigma = \nu^{-1}B\nu$, then $\Omega^* M_\sigma^t \Omega = M_{\hat{\sigma}}$, where $\hat{\sigma} = \nu^{-1}((B^{-1})^t)^{-1}\nu$.

**Lemma 2:** If $\pi$ is an affine permutation on $A_m$, i.e. if there exists some $2\times 2$, invertible matrix $B$ over $Z_m$ and some two-element constant column vector $a$ such that the counterpart permutation $\sigma = \nu^{-1}\pi\nu$ of $\pi$ is given by $\sigma = \nu^{-1}(a +$

$B)\nu$, then $\Omega^* M_\sigma^t \Omega = diag\left\{\omega^{<-a,\nu(j)>}\right\}_{j=0}^{n-1} M_{\tilde{\sigma}}$, where $\tilde{\sigma} = \nu^{-1}((B^{-1})^t)^{-1}\nu$ and $diag\{\cdot\}$ denotes a diagonal matrix.

Applying these lemmas, we can compute $\tilde{A}_1$ as

$$\begin{aligned}
\tilde{A}_1 &= (1/32)\sum_{i=1}^{16} \Omega^* M_{\sigma_i}^t \Omega \\
&= \frac{1}{32} diag\{1 + \omega^{<a_2,\nu(j)>} + \omega^{<-a_2,\nu(j)>} \\
&+ \omega^{<-2a_2,\nu(j)>}\}[M_{\nu^{-1}((B_1^{-1})^t)^{-1}\nu} + M_{\nu^{-1}((B_3^{-1})^t)^{-1}\nu}] \\
&+ \frac{1}{32} diag\{1 + \omega^{<a_1,\nu(j)>} + \omega^{<-a_1,\nu(j)>} \\
&+ \omega^{-<2a_1,\nu(j)>}\}[M_{\nu^{-1}((B_5^{-1})^t)^{-1}\nu} + M_{\nu 1((B_7^{-1})^t)^{-1}\nu}]
\end{aligned}$$

Furthermore, using the facts that $((B_1^{-1})^t)^{-1} = B_5, ((B_3^{-1})^t)^{-1} = B_7, < a_1, \nu(j) >= j_1$ and $< a_2, \nu(j) >= j_2$, where $(j_1, j_2)$ is the $m$-ary code of $j$, we obtain

$$\begin{aligned}
\tilde{A}_1 &= (1/32) diag\left\{1 + \omega^{j_1} + \omega^{-j_1} + \omega^{-2j_1}\right\}[M_{\sigma_1} + M_{\sigma_3}] \\
&+ (1/32) diag\left\{1 + \omega^{j_2} + \omega^{-j_2} + \omega^{-2j_2}\right\}[M_{\sigma_5} + M_{\sigma_7}]
\end{aligned}$$

Similarly, using the facts that $\tilde{A}_2 = \tilde{A}_1{}^*$ and $M_{\sigma_{2i-1}}^* = M_{\sigma_{2i-1}}^t = M_{\sigma_{2i-1}^{-1}}$ for $i = 1, 2, 3, 4$, we compute $\tilde{A}_2$ as

$$\begin{aligned}
\tilde{A}_2 &= (1/32) diag\left\{1 + \omega^{-j_1} + \omega^{j_1} + \omega^{2j_1}\right\}\left[M_{\sigma_1^{-1}} + M_{\sigma_3^{-1}}\right] \\
&+ (1/32) diag\left\{1 + \omega^{-j_2} + \omega^{j_2} + \omega^{2j_2}\right\}\left[M_{\sigma_5^{-1}} + M_{\sigma_7^{-1}}\right]
\end{aligned}$$

The $j,k$-th entry of $H$ is the summation of the $j,k$-th entries of $\tilde{A}_1$ and $\tilde{A}_2$, and is hence given by

$$\begin{aligned}
h_{j,k} &= \frac{1}{32}(1 + \omega^{j_1} + \omega^{-j_1} + \omega^{-2j_1})\,[\delta(B_1\nu(j), \nu(k)) + \delta(B_3\nu(j), \nu(k))] \\
&+ \frac{1}{32}(1 + \omega^{-j_1} + \omega^{j_1} + \omega^{2j_1})\,[\delta(B_2\nu(j), \nu(k)) + \delta(B_4\nu(j), \nu(k))] \\
&+ \frac{1}{32}(1 + \omega^{j_2} + \omega^{-j_2} + \omega^{-2j_2})\,[\delta(B_5\nu(j), \nu(k)) + \delta(B_7\nu(j), \nu(k))] \\
&+ \frac{1}{32}(1 + \omega^{-j_2} + \omega^{j_2} + \omega^{2j_2})\,[\delta(B_6\nu(j), \nu(k)) + \delta(B_8\nu(j), \nu(k))]
\end{aligned}$$

for $1 \le j,k \le n-1$.

To upper bound the largest eigenvalue $\lambda$ of $H$, we define a real-valued matrix $C = [c_{j,k}]_{(n-1)\times(n-1)}$ by

$$\begin{aligned} c_{j,k} &= \frac{1}{32}|1+\omega^{-j_1}+\omega^{j_1}+\omega^{2j_1}|\sum_{i=1}^{4}\delta(B_i\nu(j),\nu(k)) \\ &+ \frac{1}{32}|1+\omega^{-j_2}+\omega^{j_2}+\omega^{2j_2}|\sum_{i=5}^{8}\delta(B_i\nu(j),\nu(k)) \end{aligned} \tag{5.1}$$

where $1 \le j,k \le n-1$. Then $c_{j,k} \ge |h_{j,k}|, 1 \le j,k \le n-1$, and $C$ is symmetric because $H$ is Hermitian. By the well-known principal of Rayleigh, there exists an upper bound on the largest eigenvalue $\lambda_C$ of $C$, and, in turn, $\lambda$ (the largest eigenvalue of $H$) is upper bounded by $\lambda_C$ (See Propositions 3.5 & 3.6 in [JM87] for detail). Specifically, we have the following inequality:

$$\lambda \le \max_{1\le j\le n-1}\sum_{k=1}^{n-1}\gamma_{j,k}c_{j,k}$$

where $\gamma_{j,k}$ is any real valued function of $j$ and $k$, which satisfies the conditions that $\gamma_{j,k} > 0, \gamma_{j,k} = 1/\gamma_{k,j}, 1 \le j,k \le n-1$. Let $D_j = \sum_{k=1}^{n-1}\gamma_{j,k}c_{j,k}$. Then $\lambda \le \max_{1\le j\le n-1} D_j$.

Now, in order to upper bound $\lambda$, we need a real-valued function $\gamma_{j,k}$. First, let $\| \begin{pmatrix} \cdot \\ \cdot \end{pmatrix} \|$ be a norm on $A_m$ defined by $\| \begin{pmatrix} j_1 \\ j_2 \end{pmatrix} \| = Re(\omega^{j_1}) + Re(\omega^{j_2})$ for each $(j_1,j_2) \in A_m$, where $Re(\cdot)$ denotes the real part of a complex number. Then, define $\gamma_{j,k}$ by

$$\begin{aligned} \gamma_{j,k} &= \begin{cases} 1/\sqrt{3.4} & \text{if } \| \nu(j) \| > \| \nu(k) \| \\ 1 & \text{if } \| \nu(j) \| = \| \nu(k) \| \\ \sqrt{3.4} & \text{if } \| \nu(j) \| < \| \nu(k) \| \end{cases} \\ &= (1/\sqrt{3.4})\mu(\| \nu(j) \| - \| \nu(k) \|) + \delta(\| \nu(j) \| - \| \nu(k) \|) \\ &+ \sqrt{3.4}\mu(\| \nu(k) \| - \| \nu(j) \|) \end{aligned}$$

where $\delta(\cdot)$ is the Dirac delta function and $\mu(\cdot)$ is the step function defined as $\mu(x) = 1$ if $x > 0$; otherwise $\mu(x) = 0$.

From Equation 5.1, we find that there are at most 8 distinct $k$'s such that $c_{j,k} \neq 0$ for any given $j, 1 \leq j \leq n-1$. Specifically, they are $k_i = \nu^{-1} B_i \nu(j), i = 1, 2, \ldots, 8$. Hence, the corresponding $\gamma_{j,k_i}$'s are expressed as $\gamma_{j,k_i} = (1/\sqrt{3.4})\mu(E_i) + \delta(E_i) + \sqrt{3.4}\mu(-E_i)$ where $E_i = \| \nu(j) \| - \| \nu(k_i) \| = \| \nu(j) \| - \| B_i \nu(j) \|$ for $i = 1, 2, \ldots, 8$.

Combining these, $D_j$ can be written as

$$D_j = \frac{1}{32}|1 + \omega^{-j_1} + \omega^{j_1} + \omega^{2j_1}| \sum_{i=1}^{4} \left[ \frac{1}{\sqrt{3.4}}\mu(E_i) + \delta(E_i) + \sqrt{3.4}\mu(-E_i) \right]$$
$$+ \frac{1}{32}|1 + \omega^{-j_2} + \omega^{j_2} + \omega^{2j_2}| \sum_{i=5}^{8} \left[ \frac{1}{\sqrt{3.4}}\mu(E_i) + \delta(E_i) + \sqrt{3.4}\mu(-E_i) \right]$$

From now on, for calculation convenience, we will change the discrete version of $D_j$ into the continuous version. Given any $\nu(j) = \begin{pmatrix} j_1 \\ j_2 \end{pmatrix}, 0 \leq j_1, j_2 \leq m-1$, let $x = 2\pi j_1/m$ and $y = 2\pi j_2/m$. Since $\omega = exp[(2\pi/m)\sqrt{-1}]$, we have $|1 + \omega^{-j_1} + \omega^{j_1} + \omega^{2j_1}| = 4|cos(x)cos(x/2)|$ and $|1 + \omega^{-j_2} + \omega^{j_2} + \omega^{2j_2}| = 4|cos(y)cos(y/2)|$. Furthermore, because $\| \nu(j) \| = Re(\omega^{j_1}) + re(\omega^{j_2}) = cos(x) + cos(y)$, $E_i$'s can be expressed as

$$\begin{aligned}
E_1 &= 2sin(3y + x)sin(3y), \\
E_2 &= 2sin(3y - x)sin(3y), \\
E_3 &= 8sin(\frac{3y + x}{2})sin(\frac{x + y}{2})cos^2(\frac{x + y}{2}), \\
E_4 &= 8sin(\frac{3x + y}{2})sin(\frac{x + y}{2})cos^2(\frac{x + y}{2}), \\
E_5 &= 2sin(3x + y)sin(3x), \\
E_6 &= 2sin(3x - y)sin(3x), \\
E_7 &= 8sin(\frac{3y - x}{2})sin(\frac{y - x}{2})cos^2(\frac{y - x}{2}),
\end{aligned}$$

$$E_8 = 8sin(\frac{3x-y}{2})sin(\frac{x-y}{2})cos^2(\frac{x-y}{2}). \quad (5.2)$$

Therefore, we get the continuous version $D(x,y)$ of $D_j$ given by

$$D(x,y) = \frac{1}{8}|cos(x)cos(x/2)|\sum_{i=1}^{4}\left[\frac{1}{\sqrt{3.4}}\mu(E_i) + \delta(E_i) + \sqrt{3.4}\mu(-E_i)\right] + \frac{1}{8}|cos(y)cos(y/2)|\sum_{i=5}^{8}\left[\frac{1}{\sqrt{3.4}}\mu(E_i) + \delta(E_i) + \sqrt{3.4}\mu(-E_i)\right] \quad (5.3)$$

Now, $\lambda$ is bounded by $\lambda \leq \max_{-\pi\leq x,y\leq\pi} D(x,y)$, where $x$ and $y$ are not both 0. In addition, we note that $D(x,y)$ is symmetric, i.e., $D(x,y) = D(y,x) = D(-x,y) = D(x,-y)$. Therefore,

$$\lambda \leq \max_{(x,y)\in R} D(x,y), \text{ where } R = \{(x,y)\,|\,0 \leq y \leq x \leq \pi, x+y \neq 0\}.$$

We will show that the right-hand side of the above inequality is upper bounded by 0.705. From Equation 5.3, we first note that $D(x,y)$ depends both on the signs of $E_i$'s and the values of $|cos(x)cos(x/2)|$ and $|cos(y)cos(y/2)|$. We then partition $R$ into the following subranges:

$$\begin{aligned}
R_1 &= \{(x,y)|x+3y \geq 2\pi\} \\
R_2 &= \{(x,y)|x+3y < 2\pi, x-3y \leq 0, 3x-y \geq 2\pi\} \\
R_3 &= \{(x,y)|x+3y < 2\pi, x-3y \leq 0, 3x-y < 2\pi, 3x+y \geq 2\pi\} \\
R_4 &= \{(x,y)|x-3y > 0, 3x+y \geq 3\pi\} \\
R_5 &= \{(x,y)|x-3y > 0, 2\pi \leq 3x+y < 3\pi\} \\
R_6 &= \{(x,y)|x-3y > 0, \pi \leq 3x+y < 2\pi\} \\
R_7 &= \{(x,y)|x-3y \leq 0, 3x+y < 2\pi, x+3y \geq \pi\} \\
R_8 &= \{(x,y)|x-3y \leq 0, 3x+y \geq \pi, x+3y < \pi\} \\
R_9 &= \{(x,y)|3x+y < \pi\}.
\end{aligned}$$

In each subrange, we can check the sign of each $E_i, i = 1, 2, \ldots, 8$ and calculate the maximums of $|cos(x)cos(x/2)|$ and $|cos(y)cos(y/2)|$. Subsequently, it is easy to verify that $D(x, y)$ is upper bounded by 0.705 in each subrange, and then $D(x, y)$ is upper bounded by 0.705 in $R$. Also, $\lambda$ is upper bounded by 0.705.

Finally, applying this result to Theorem 5.1, we have

$$c_\alpha \geq \frac{2.36}{0.705 + 1.18\alpha + \sqrt{3.06\alpha + 0.4971}}.$$

Taking $\alpha = 1$, our construction yields an $(n, 33, 0.632, 1)$-expander, and taking $\alpha = 0.5$, it yields an $(n, 33, 0.868, 0.5)$-expander. Both these expanders have larger expansion coefficients than the expanders constructed similarly [Mar73, GG81, AM85, Alo86, NAM87, JM87].

**Remark 5.1** *The selection of our permutations is based on an observation as follows. If different permutations on $A_m$ are chosen, the upper bound on the second largest eigenvalue $\lambda$ may be different. In fact, different permutations will yield different $E_i$'s in Equations 5.2 and a different $D(x, y)$ in Equation 5.3. In turn, it will generate a different partition on $R = \{(x, y) \mid 0 \leq y \leq x \leq \pi, x + y \neq 0\}$. Finally, in each subrange $R_i$'s, $D(x, y)$ is upper bounded by a different number. By trial and error, we found that our permutations will have the smallest upper bound on $D(x, y)$ in each subrange and hence on $\lambda$.* ||

## 5.3 Bounded Concentrators and Superconcentrators

In this section, we use the expander constructed in the earlier section to obtain explicit constructions of linear-size concentrators and superconcentrators. The constructions follow those in [Pip77, GG81].

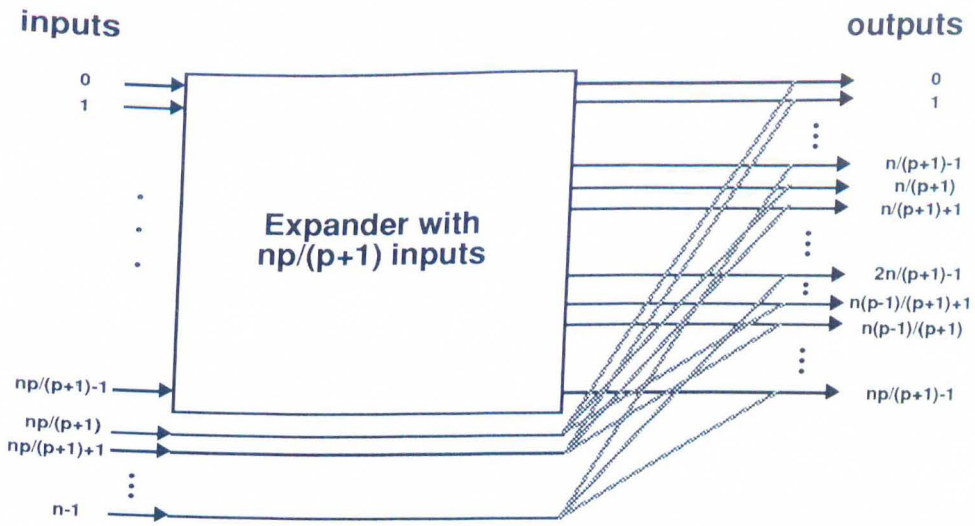



**Figure 25**: An explicit construction of bounded concentrators.

### 5.3.1 Bounded Concentrator Construction

A linear-size bounded concentrator can be constructed by using a linear-size expander. Figure 25 shows such a construction of a bounded concentrator which has $n$ inputs and $pn/(p+1)$ outputs. For the last $n/(p+1)$ inputs, each of them is joined by $p$ edges to $p$ outputs, and no two of them are joined to the same output.

**Theorem 5.2** *The network constructed in Figure 25 is an* $(n, pn/(p+1); k', 0.5)$-*bounded concentrator if the expander is an* $(pn/(p+1); k, c, (p^2-1)/2p^2)$-*expander with an expansion coefficient* $c \geq 2p^2/(p^2+1)(p-1)$, *where* $k' = p(k+1)/(p+1)$.

**Proof:** Obviously, the network as shown in Figure 25 is bipartite and has $p(k+1)n/(p+1)$ edges. By Definition 2.6, it suffices to show that, for any subset $X$ of inputs with $|X| \leq 0.5n$, $|\Gamma_X| \geq |X|$, where $\Gamma_X$ is the set of outputs

adjacent to inputs of $X$. Let $X_1$ and $X_2$ be the intersections of $X$ and the subsets of the first $pn/(p+1)$ inputs and the last $n/(p+1)$ inputs, respectively. Then, $|X| = |X_1| + |X_2|$. If $|X_2| \geq |X|/p$, $|\Gamma_X| \geq |X|$ since the number of outputs adjoint to inputs $X_2$ is exactly $p|X_2|$. Otherwise, $|X_1| \geq ((p-1)/p)|X|$, and let $X'$ be a $((p-1)/p)|X|$-subset of $X_1$. It is clear that $X'$ is a subset of inputs of the $(pn/(p+1); k, c, (p^2-1)/2p^2)$-expander and $|X'| \leq ((p^2-1)/p^2)(pn/(p+1))$ because $|X| \leq 0.5n$. From the definition of expander (See Definition 2.7), $|\Gamma_{X'}| \geq [1 + c(1-(p^2-1)/p^2)]|X'|$, where $\Gamma_{X'}$ is the set of outputs adjacent to inputs of $X'$. Therefore, $|\Gamma_X| \geq |\Gamma_{X'}| \geq (p/(p-1))|X'| = |X|$ if the expansion coefficient $c \geq 2p^2/(p^2+1)(p-1)$. ||

Taking $p = 3$ and substituting the previously constructed $(3n/4; 33, 0.91, 4/9)$-expander into the construction in Figure 25, we obtain an $(n, 3n/4; 25.5, 0.5)$-bounded concentrator.

### 5.3.2 Superconcentrator Construction

Using two $(n, pn/(p+1); k, 0.5)$-bounded concentrator, Figure 26 shows a recursive construction of an $n$-input superconcentrator in which there is an edge joining input $i$ and output $i$, $0 \leq i \leq n-1$.

**Theorem 5.3** *The network constructed in Figure 26 is an $(n; d)$-superconcentrator for $d = (p+1)(2k+1)$.*

**Proof:** For any $k$ inputs and any $k$ outputs, $1 \leq k \leq n$, let input $i$ be paired with output $i$ if input $i$ is one of the $k$ inputs and output $i$ is one of the $k$ outputs, and let $X$ and $Y$ be the sets of the remaining inputs and outputs that can not be paired in this way, respectively. Then, $|X| = |Y|$,

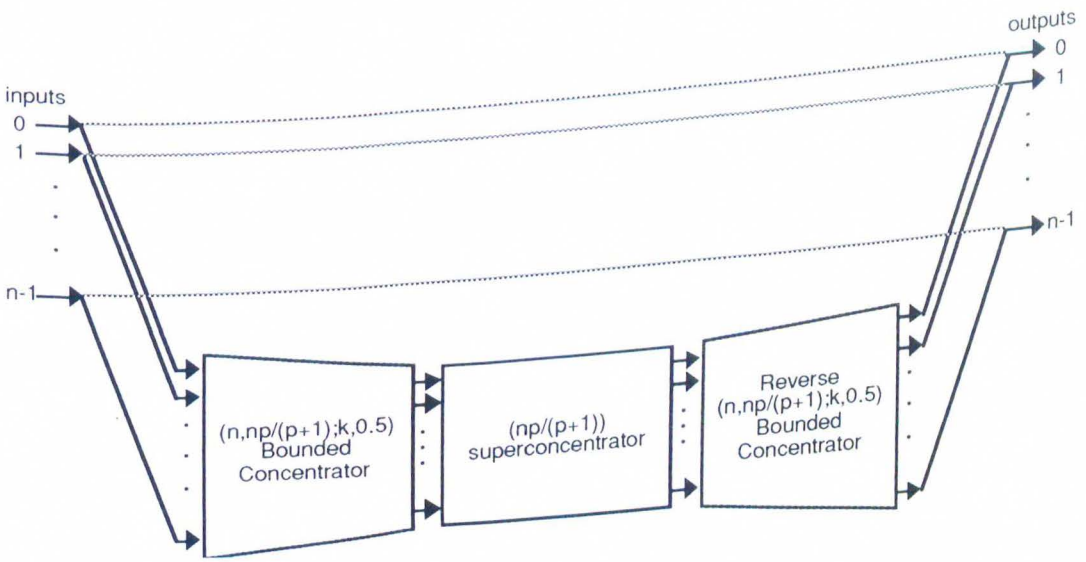



**Figure 26**: A recursive construction of an $n$-input superconcentrator.

$|X| + |Y| \leq k$ and $|X|, |Y| \leq n/2$. Now find a unicast assignment from $X$ onto some set, say $X'$, of outputs of the $(n, pn/(p+1); k, 0.5)$-bounded concentrator, and find a unicast assignment from $Y$ onto some set, say $Y'$, of inputs of the reverse $(n, pn/(p+1); k, 0.5)$-bounded concentrator. Then, by induction, the sandwiched $(pn/(p+1))$-superconcentrator can realize a unicast assignment from $X'$ onto $Y'$. Combining the unicast assignments from $X$ onto $X'$, from $X'$ onto $Y'$ and from $Y'$ onto $Y$, we have a unicast assignment from $X$ onto $Y$. Therefore, the network in Figure 26 is a superconcentrator. If $S(n)$ denotes the number of edges of such an $n$-input superconcentrator, then $S(n) = S(pn/(p+1)) + (2k+1)n$. Solving this recurrence, we have the density of the superconcentrator given by $d = (p+1)(2k+1)$. ||

Using the $(n, 3n/4; 25.5, 0.5)$-bounded concentrators constructed in the previous section, we have a family of superconcentrators with density 208. We must note that this density is not the smallest density that is known; Alon et al [NAM87], and Jimbo and Maruoka [JM87] already constructed superconcentrators with density 122. However, we should also note that it is still

possible to reduce the density by using our expander in a superconcentrator design different from that given in [GG81].

## 5.4 Summary

The expander construction given in this chapter has a very large expansion coefficient but its density is also quite large as compared to the densities of the previous expander constructions. For explicit constructions of $m^2$-expanders, the following problems remain open:

(a) Does there exist an expander with a smaller density, but with about the same expansion coefficient as the expander given here?

(b) Does there exist an expander with density $\leq 20$ and expansion coefficient $\geq 0.7$? If so, this will give a superconcentrator with density at most 85.

(c) More generally, can one find a tight lower bound on the expansion coefficient as a function of the density?

The answers to these questions will largely settle the density question for concentrators and superconcentrators.

CHAPTER
SIX

# CONCLUSIONS

## 6.1 Summary of Contributions

We have presented parallel routing algorithms with the fastest routing time for the Beneš networks. Using these algorithms, an $n$-input Beneš network can route any unicast $k$-assignment, $1 \leq k \leq n$, in $O(\lg^2 k + \lg n)$ time without pipelining and $O(\lg k)$ time with pipelining if each pair of processors are directly connected, and in $O(\lg^4 k + \lg^2 k \lg n)$ time without pipelining and $O(\lg^3 k + \lg k \lg n)$ time with pipelining if the processors are extended perfect-shuffle connected. We list the comparison of various parallel routing algorithms for the Beneš networks in Tabel 1.

We have also introduced new networks with self-routing schemes for routing any multicast assignment. For $n$ inputs, the multicasting network has $O(n \lg^2 n)$ bit-level constant fanin logic gates and $O(\lg^2)$ bit-level depth, and can realize any multicast assignments in $O(\lg^3 n)$ bit-level time without pipelining

| Algorithm | Topology | Routing Time for $k$-Assignments |
|---|---|---|
| Nassimi and Sahni [NS82a] | complete | $O(\lg^2 n)$ |
| | perfect shuffle | $O(\lg^4 n)$ |
| Lev et al [LPV81] | complete | $O(\lg^2 n)$ |
| Our Algorithm | complete | $O(\lg^2 k + \lg n)$ |
| | | $O(\lg k)$ pipelinling |
| | perfect shuffle | $O(\lg^4 k + \lg^2 k \lg n)$ |
| | | $O(\lg^3 k + \lg k \lg n)$ pipelining |

Table 1: Comparison of parallel routing algorithms for Beneš networks.

and $O(\lg n)$ bit-level time with pipelining for $O(\lg^2 n)$ multicast assignments. Tabel 2 shows the complexities of three different multicasting networks.

Finally, we have given an explicit construction of a family of linear-size expanders with density 33 and expansion coefficient 0.868, a family of bounded concentrators with density 25.2 and a family of superconcentrators with density 208. These networks can be used to reduce the cost of our multicasting networks to $O(n \lg n)$ in bit level. Tabel 3 compares various designs of linear-size expanders.

## 6.2 Future Research

Our parallel routing algorithm for the Beneš networks can run not only on the complete and the extended perfect-shuffle topologies as described in this dissertation, but also on other topologies. One future research is to consider other topologies that will fit the algorithm, and another direction of future research is to build hardware prototype routers to implement and demonstrate the algorithm.

It seems that our algorithm has possibly provided the fastest routing time for the Beneš network, and that it is difficult to change the structure of the algorithm to get a major gain of routing time. With minor adjusting, our algorithm also fits other unicasting networks that can be recursively constructed as the Beneš network, such as the omega plus reverse omega networks and the baseline plus baseline networks. Hence, it is also difficult to obtain a faster routing time for these networks than that for the Beneš network. However, one possible exception is the $2 \lg n - 1$ stages shuffle/exchange network because it is not recursively constructable. Although this network has been proved to be a unicasting network [FJ91], it still lacks a routing algorithm and even a parallel

| Construction | Cost | Depth | Routing Time |
|---|---|---|---|
| Nassimi and Sahni [NS82b] | $O(n \lg^3 n)$ | $O(\lg^2 n)$ | $O(\lg^3 n)$ |
| Thompson [Tho78] | $O(n \lg n)$ | $O(\lg n)$ | $O(n \lg n)$ |
| Our Network | $O(n \lg^2 n)$ | $O(\lg^2 n)$ | $O(\lg^3 n)$<br>$O(\lg n)$ pipelining |

Table 2: Complexities of various multicasting network designs in bit level.

routing algorithm. One future research is to investigate this possibility.

As for multicasting networks, we have showed an efficient method that uses $O(n)$ bits per input to encode any multicast assignment. But, we still do not know how to reduce it to $O(\lg n)$ bits per input. Hence, a future research is to find a more efficient encoding method for input-initiated multicasting networks. Instead, as shown in this dissertation, output-initiated multicasting networks can be used, and $O(\lg n)$ bits of source address coding per output are sufficient. A direction for future research is to investigate the advantages of this approach and construct output-initiated multicasting networks with lower cost and depth than the input-initiated counterparts.

Regarding linear-size expanders, it would be worthwhile to construct linear-size expanders with smaller density and larger expansion coefficients, and to use

| Construction | Density | Expansion Coefficient |
|---|---|---|
| Margulis [Mar73] | 5 | unknown |
| Gabber and Galil [GG81] | 5 | 0.007 |
| Alon et al [NAM87] and | 9 | 0.412 |
| Jimbo and Maruoka [JM87] | 13 | 0.464 |
| Our Expander | 33 | 0.868 |

Table 3: Comparison of various linear-size expanders.

them to construct concentrators, superconcentrators, unicasting and multicasting networks. Our bounded concentrator construction improves that of Gabber and Galil, and our superconcentrator construction follows that of Pippenger. However, better constructions of bounded concentrators are still possible and would be worthwhile for future research. As for the superconcentrators, one future research is to explore improved constructions over those of Pippenger, Chung and Bassalygo.

We note that there are still no efficient routing algorithms for linear-size expanders and those networks that are constructed accordingly. Edge coloring schemes are possible, but they are inefficient. Hence, it is worthwhile to design efficient routing algorithms for those networks.

Finally, we note that this dissertation has not touched on the issues of strictly and wide-sense nonblocking networks. These networks are important in telephone switchings. Cantor's network [Can71], which has $O(n \lg^2 n)$ cost and $O(\lg n)$ depth, can be viewed as the representative of strictly nonblocking networks. However, the existing routing algorithms for Cantor's networks are not satisfactory. Lin and Pippenger [LP91] presented an $O(\lg^5 n)$ time parallel routing algorithm for Cantor's networks by using $O(n)$ processors and $O(n \lg^2 n)$ memory space. This time complexity is too high compared with the $O(\lg n)$ depth of the network. Therefore, one area of future research is to design better routing algorithms for Cantor's networks, and another is to build other nonblocking networks with efficient routing schemes.

# BIBLIOGRAPHY


[AKS83] M. Ajtai, J Komlos, and E. Szemeredi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, January 1983.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[AM85] N. Alon and V.D. Milman. $\lambda_1$, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory*, B 38:73–88, 1985.

[Bas81] L. Bassalygo. Asymptotically optimal switching circuits. *Problems of Information Transmission*, 17:206–211, July-September 1981.

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Proceedings of AFIPS Spring Joint Conference*, pages 307–314, 1968.

[Ben62] V. E. Beneš. On rearrangeable three-stage connecting networks. *Bell Systems Technical Journal*, 41:1481–1493, September 1962.

[Ben64] V. Beneš. Optimal rearrangeable connecting networks. *Bell Systems Technical Journal*, 43:1641–1656, July 1964.

[Ben65] V. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press Publishing Company, New York, 1965.

[Bhu87] L.N. Bhuyan. Interconnection networks for parallel and distributed processing. *IEEE Computer Magazine*, 20:9–12, June 1987.

[BP74] L. Bassalygo and M. Pinsker. Complexity of optimum nonblocking switching without reconnections. *Problems of Information Transmission*, 9:64–66, January-March 1974.

[Can71] D. G. Cantor. On non-blocking switching networks. *Networks*, 1:366–377, 1971.

[Car86] C. Cardot. Comments on a simple algorithm for the control of rearrangeable switching networks. *IEEE Transactions on Communications*, COM-34:395, April 1986.

[Chu78] F. R. K. Chung. On concentrators, superconcentrators, generalizers, and nonblocking networks. *Bell Systems Technical Journal*, 58:1765–1777, October 1978.

[Clo53] C. Clos. A study of nonblocking switching networks. *Bell Systems Technical Journal*, 32:406–425, February 1953.

[CO88] J. Carpinelli and A. Y. Oruç. Applications of edge-coloring algorithms to routing in parallel computers. In *Proceedings of 3rd International Conference on Supercomputing*, Santa Clara, CA, May 1988.

[CO92] M. V. Chien and A. Y. Oruç. Adaptive binary sorting schemes and associated interconnection networks. Technical Report UMIACS-TR-92-10,CS-TR-2830, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1992.

[DeC89] A. DeCegama. *Parallel Processing Architectures and VLSI Hardware*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

[DO90] B. Douglass and A. Y. Oruç. Self-routing and route balancing in connection networks. Technical Report UMIACS-TR-90-32,CS-TR-2421, Institute for Advanced Computer Studies, University of Maryland, College Park, MD., 1990.

[Fen81] T. Feng. A survey of interconnection networks. *IEEE Computer Magazine*, 14:12–27, December 1981.

[FFP88] P. Feldman, J. Friedman, and N. Pippenger. Wide-sense nonblocking networks. *SIAM Journal on Algebraic and Discrete Methods*, 1:158–173, February 1988.

[FJ91] C.M. Fiduccia and E.M. Jacobson. Universal multistage networks via linear permutations. Technical Report SRC-TR-91-050, Supercomputing Research Center, Institute for Defense Analyses, 17100 Science Drive, Bowie, MD 20715-4300, Dec. 1991.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[HK84] A. Huang and S. Knauer. Starlite: A wide band digital switch. In *Proceedings of GlobeCom*, 121-125, November 1984.

[Hui90] J. Y. Hui. *Switching and Traffic Theory for Integrated Broadband Network*. Kluwer Publishing Company, Boston, MA, 1990.

[Hwa72] F. K. Hwang. Rearrangeability of multiconnection three-stage networks. *Networks*, 2:301–306, 1972.

[Hwa83] F. K. Hwang. Control algorithms for rearrangeable Clos networks. *IEEE Transactions on Communications*, COM-31:952–954, August 1983.

[JM87] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7:343–355, 1987.

[JO91] C. Y. Jan and A. Y. Oruç. Fast self-routing permutation switching on an asymptotically minimum cost network. Technical Report UMIACS-TR-91-127,CS-TR-2753, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 1991.

[Kea91] J. Konicek and et al. The organization of the cedar system. In *International Conference on Parallel Processing*, pages 49–56, St. Charles, IL., August 1991.

[Knu73] D. E. Knuth. *The Art of Computer Programming, Vol 3: Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[KO90] D. Koppelman and A. Y. Oruç. A self-routing permutation network. *Journal of Parallel and Distributed Computing*, 6:140–151, August 1990.

[Lea88] C. Lea. A new broadcast switch. *IEEE Transactions on Communications*, COM-36:1128–1137, October 1988.

[Lee85] K. Y. Lee. On the rearrangeability of $2 \lg n - 1$ stage permutation networks. *IEEE Transactions on Computers*, C-34:412–425, May 1985.

[Lee87] K. Y. Lee. A new Benes network control algorithm. *IEEE Transactions on Computers*, C-36:768–772, June 1987.

[Lee88] T. Lee. Nonblocking copy networks for multicast switching. *IEEE Journal on Selected Areas in Communications*, 6:1455–1465, December 1988.

[LO92] M. Lu and A. Y. Oruç. Efficient networks for realizing point-to-point assignments in parallel processors. Technical Report UMIACS-TR-92-63,CS-TR-2910, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, June 1992.

[LP91] G. Lin and N. Pippenger. Parallel algorithms for routing in nonblocking networks. In *Proc. ACM Symposium on the Theory of Computing*, pages 272–277, 1991.

[LPV81] G. F. Lev, N. Pippenger, and L. Valiant. A fast parallel routing algorithm for routing in permutation networks. *IEEE Transactions on Computers*, C-30:93–100, February 1981.

[Mar73] G. A. Margulis. Explicit constructions of concentrators. *Problems of Information Transmission*, 9(4):325–332, 1973.

[MJ72] G. M. Masson and B. W. Jordan. Generalized multistage connection networks. *Networks*, 2:191–209, 1972.

[MZ90] N. Maxemchuk and M. El Zarki. Routing and flow control in high speed wide area networks. *Proceedings of the IEEE*, 78:205–221, January 1990.

[NAM87] N. Alon, Z. Galil and V.D. Milman. Better expanders and superconcentrators. *Journal of Algorithms*, 8:337–347, 1987.

[NS81] D. Nassimi and S. Sahni. A self-routing Benes network and parallel permutation algorithms. *IEEE Transactions on Computers*, C-30:332–340, May 1981.

[NS82a] D. Nassimi and S. Sahni. Parallel algorithms to set up the Beneš network. *IEEE Transactions on Computers*, C-31:148–154, February 1982.

[NS82b] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. *Journal of the ACM*, 29:642–667, July 1982.

[Ofm65] J. P. Ofman. A universal automaton. *Transactions of the Moscow Mathematical Society*, 14:200–215, 1965.

[Oru87a] A. Y. Oruç. Designing permutation networks through coset decompositions of symmetric groups. *Journal of Parallel and Distributed Computing*, 3:402–422, August 1987.

[Oru87b] A. Y. Oruç. Rearrangeable networks based on double coset decompositions of symmetric groups. In *Proc. 21st Information Systems and Sciences Conference*, pages 453–458, Johns Hopkins University, Baltimore, MD, 1987.

[OTW71] D. Opferman and N. Tsao-Wu. On a class of rearrangeable switching networks. *Bell Systems Technical Journal*, 50:1579–1618, May-June 1971.

[Pau62] M. C. Paull. Reswitching of connection networks. *Bell Systems Technical Journal*, 41:833–855, May 1962.

[Pin73] M. S. Pinsker. On the complexity of a concentrator. In *Proc. of 7th International Teletraffic Congress*, pages 318/1–318/4, Stockholm, Sweden, 1973.

[Pip77] N. Pippenger. Superconcentrators. *SIAM Journal on Computing*, 6:298–304, 1977.

[Pip78a] N. Pippenger. Generalized connectors. *SIAM Journal on Computing*, 7:510–514, November 1978.

[Pip78b] Pippenger. On rearrangeable and nonblocking switching networks. *Journal of Computer and System Sciences*, 17:145–162, September 1978.

[Pip82] N. Pippenger. Telephone switching networks. *Symposia in Applied Mathematics*, volume 26, pages 101–133. American Mathematical Society, 1982.

[RB90] I. Rubin and J. Baker. Media access control for high speed local area and metropolitan communication networks. *Proceedings of the IEEE*, 78:168–203, January 1990.

[RH85] G. W. Richards and F. K. Hwang. A two-stage rearrangeable broadcast switching network. *IEEE Transactions on Communications*, COM-33:1025–1034, October 1985.

[Sea89] H. J. Siegel and et al. Using the multistage cube network topology in parallel supercomputers. *Proceedings of the IEEE*, 77:1932–1953, December 1989.

[Sie90] H.J. Siegel. *Interconnection Networks for Large-Scale Parallel Processing–Theory and Case Studies*. McGraw-Hill Publishing Company, New York, 1990.

[Sto71] H. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C-20:153–161, February 1971.

[Tho78] C. D. Thompson. Generalized connection networks for parallel processor intercommunication. *IEEE Transactions on Computers*, C-27:1119–1125, December 1978.

[Tob90] F. Tobagi. Fast packet switch architectures for broadband integrated services digital networks. *Proceedings of the IEEE*, 78:133–167, January 1990.

[Tur88] J. Turner. Design of a broadcast packet switching network. *IEEE Transactions on Communications*, COM-36:734–743, June 1988.

[Wak68] A. Waksman. A permutation network. *Journal of the ACM*, 15:159–163, January 1968.

[WF80a] C. L. Wu and T. Y. Feng. On a class of multistage interconnection networks. *IEEE Transactions on Computers*, C-29:694–702, August 1980.

[WF80b] C. L. Wu and T. Y. Feng. The reverse-exchange interconnection network. *IEEE Transactions on Computers*, C-29:801–811, September 1980.

[YM91a] Y. Yang and G. Masson. Nonblocking broadcast switching networks. *IEEE Transactions on Computers*, C-40:1005–1015, September 1991.

[YM91b] Y. Yang and G. Masson. Broadcast ring sandwich networks. Technical Report JHU-91/01, Department of Computer Science, Johns Hopkins University, Baltimore, MD, 1991.