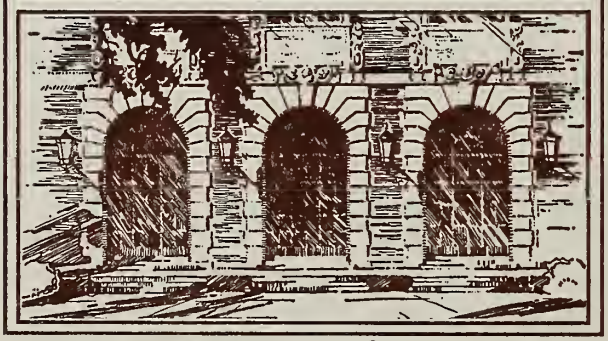
LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84
Il6r
no.553-558
cop. 2



CENTRAL CIRCULATION AND BOOKSTACKS

The person borrowing this material is responsible for its renewal or return before the **Latest Date** stamped below. **You may be charged a minimum fee of $75.00 for each non-returned or lost item.**

Theft, mutilation, or defacement of library materials can be causes for student disciplinary action. All materials owned by the University of Illinois Library are the property of the State of Illinois and are protected by Article 16B of Illinois Criminal Law and Procedure.

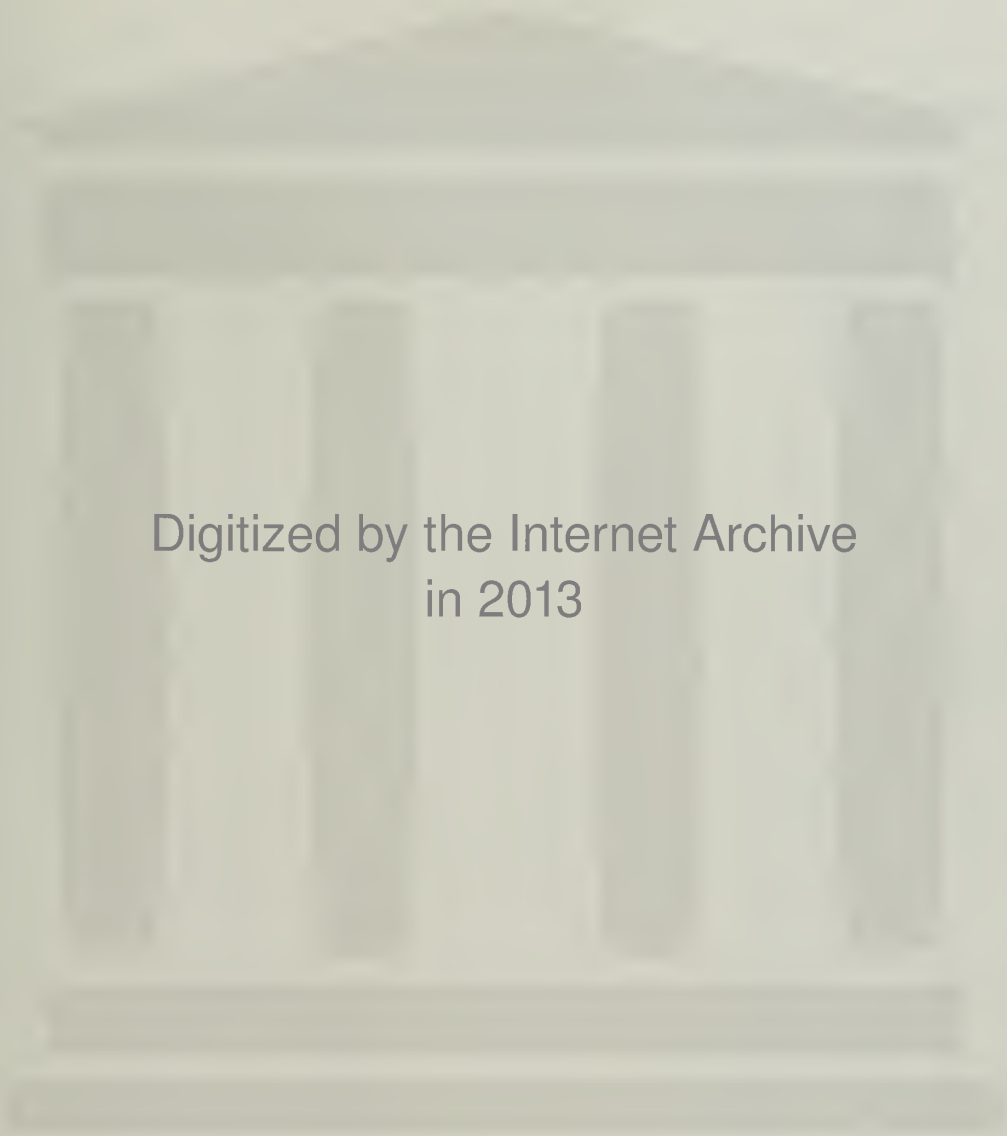
TO RENEW, CALL (217) 333-8400.

University of Illinois Library at Urbana-Champaign

JUL 1 1 2001

JUL 1 1 2001

When renewing by phone, write new due date below previous due date. L162


Digitized by the Internet Archive
in 2013

http://archive.org/details/memoryprocessorc557lawr

510.84
I16r
no.557
cop.2

Math

Report No. UIUCDCS-R-73-557

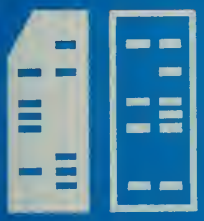
THE LIBRARY OF THE
MAR 15 1973
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

MEMORY-PROCESSOR CONNECTION NETWORKS

by

Duncan Hamish Lawrie

February 1973



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. UIUCDCS-R-73-557

MEMORY-PROCESSOR CONNECTION NETWORKS*

by

Duncan Hamish Lawrie

February 1973

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF GJ 27446 and was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, February 1973.

MEMORY-PROCESSOR CONNECTION NETWORKS

Duncan Hamish Lawrie, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1973

In order to utilize the potential speed of a SIMD type parallel processor it is necessary to arrange data in the memory system so that subsets of this data can be fetched in parallel without memory conflicts. Additionally, we must provide sufficient memory-processor paths to allow the data to be correctly aligned with the processor array. In this paper we present several storage mapping algorithms together with a memory-processor interconnection network. We demonstrate the cost and effectiveness of these and compare them with other networks which have been proposed for this application.

ACKNOWLEDGMENT

The author would like to thank Professor David J. Kuck for his patience and inspiration, the Department of Computer Science for its support, and the ILLIAC IV Project for a long and usually fruitful relationship.

The diligent efforts of Vivian Alsip and Barbara Armstrong in typing this document and the workmanship of the printing and drafting departments are greatly appreciated.

TABLE OF CONTENTS

1. INTRODUCTION

It has been suggested that the use of a large number of processors operating on a single program can be used to produce improvements in speed and possibly cost effectiveness. Recent experiments with this concept have indicated a number of problems which are as yet unsolved. The purpose of this paper is to investigate a possible solution to one of these problems, the problem of how to provide the processors with data at a rate which matches the processors' speed.

It is clear that given N processors, we need on the order of N memories just to provide the raw bandwidth requirements. Given an order of N memories, we must still solve two problems. The first problem is to assign the data to memory in such a way that memory conflicts are minimized; that is, the data must be arranged so that during the fetch of any required set of N data elements, no two of these will be stored in the same memory. This will be called the memory assignment problem. Once the data has been fetched, each datum must be sent to the correct processor. This is the second problem which we will call the data alignment problem.

Independent solutions to the above problems have been suggested in the literature and elsewhere, but to the best of our knowledge no practical and compatible solution to both problems has ever been published. For example, $(\pm 1, \pm \sqrt{N})$, and barrel shifters have been proposed (and built) to solve the data alignment problem [1] and memory assignment algorithms have been proposed to solve the memory assignment problem [2, 3]. Unfortunately, the better solutions of the latter generally require alignments which are more complex than simple uniform shifts (see Chapter 5). More complex networks have been proposed [4, 5] , but their high cost or excessive switching time

make them impractical for our purposes.

In this paper we present several effective memory assignment algorithms and propose a new network which can meet the resulting alignment requirements. We begin by proving some useful results about these networks and then showing their effectiveness in handling certain important alignments. We follow this with a comparison with other solutions and show that our solution is more effective than these previous solutions.

Before continuing we will define some terms and assumptions which will be implicit in the following work. By a network we mean a device which has n input and n output ports. Each datum entering the network remains integral, but the ordering of the data may be altered by the network. We assume that a different ordering (or permutation) of the inputs is required for each successive set of inputs, i.e., a different switching of the network is required for each set of n or less inputs. Thus, the time required to switch the network is a significant factor.

This network may be thought of as lying between the N processors and M memories as in Figure 1, as a separate function box available to the processors as in Figure 2, or as an integral part of the processors.

In general, we assume that at each unit of time, N or fewer of the processors require one datum each. These data are fetched and aligned with the correct processors. We shall also assume that these data are taken from a uniform part of an $N \times N$ matrix, e.g., row, column, diagonal, etc. We will define this more clearly in Chapter 3.

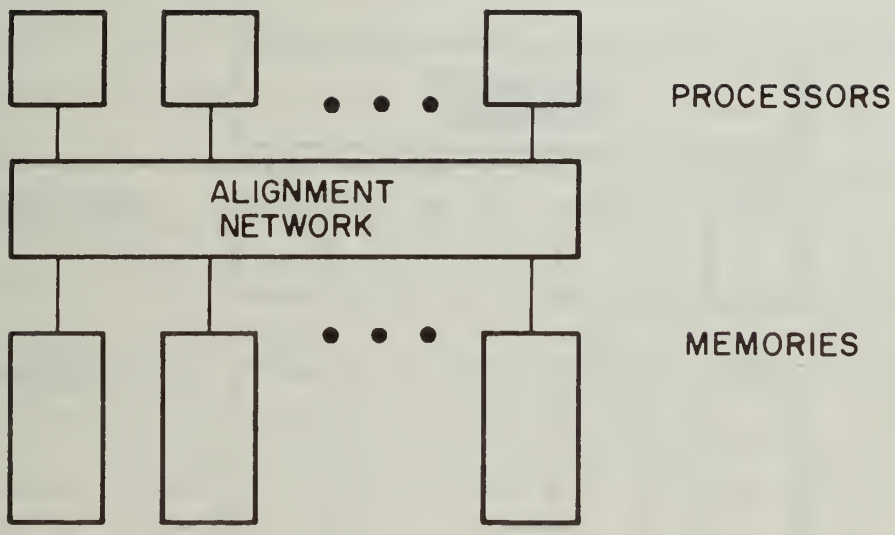


Figure 1. Processor-Network-Memory System

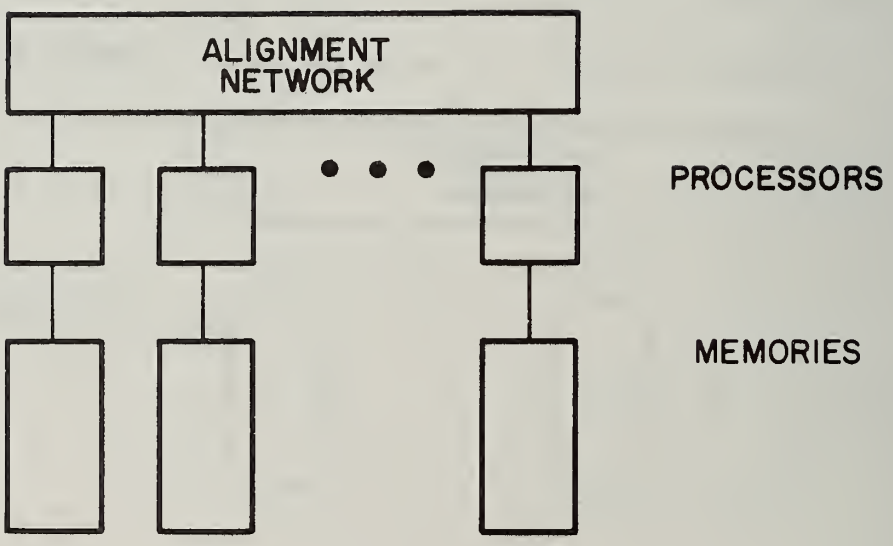


Figure 2. Network-Processor-Memory System

2. Ω NETWORKS, THEORETICAL DEVELOPMENT

The purpose of a switching network, variously known as a permutation, indexing, connection, or sorting network, is to produce connections between input nodes and output nodes. The network may be changed or switched by means of control signals in such a way as to alter these connections to meet various requirements. In this chapter we will investigate in some detail a special class of network, henceforth Ω networks, which we feel are especially useful for connections between a vector of memories and a vector of processors.

Ω networks are based on a mathematical notion of a "Ω base" representation of integers. We begin by investigating some special properties of an Ω base system. We then proceed to show how to build a network from an Ω base and then using our results we show how to determine whether the Ω network can produce certain special connections.

In later chapters we will derive a number of connections which are frequently required during operation of a parallel computer. Then, using the results derived in this chapter, we will show that the Ω network performs well in certain types of parallel computers.

2.1 Fundamentals

We will begin with some fundamental definitions from number theory. Any number x can be expressed as the sum of a multiple of a modulus m and a remainder r:

$$x = mb + r \qquad r < m$$

Given two numbers

$$x = mb_x + r_x$$

$$y = mb_y + r_y$$

we say $x \underset{m}{\equiv} y$ if and only if $r_x = r_y$. Further, we say

$$x \overset{m}{\equiv} y$$

if and only if $b_x = b_y$. We say x divides y

$$x|y$$

if there exists an integer a such that $ax = y$. Further, whether or not x divides y, we define

$$x \div y$$

to be the integer part of the quotient x/y.

The greatest common divisor of x and y,

$$\gcd(x,y)$$

is the largest integer a such that

$$a|x \text{ and } a|y.$$

If $\gcd(x,y) = 1$, we say x and y are relatively prime, x is prime to y, or y is prime to x.

The greatest prime factor, written

$$\text{gpf}(x,y),$$

is the greatest integer factor (divisor) of x which is prime to y. Thus,

$$\gcd(\text{gpf}(x,y),y) = 1.$$

We now present a number of results. Most proofs can be found in Vinogradov [6] or Shanks [7].

P1 $\quad x+a \underset{m}{\equiv} y+a \leftrightarrow x \underset{m}{\equiv} y$

P2 $\quad x \underset{m}{\equiv} y \rightarrow ax \underset{m}{\equiv} ay$

P3 $\quad \gcd(a,m)=1 \wedge ax \underset{m}{\equiv} ay \rightarrow x \underset{m}{\equiv} y$

P4 $\quad a \mid m \wedge ax \underset{m}{\equiv} ay \rightarrow x \underset{m/a}{\equiv} y$

P5 $\quad a \mid m \wedge x \underset{m}{\equiv} y \rightarrow x \underset{m/a}{\equiv} y$

P6 $\quad x \underset{am}{\not\equiv} y \wedge x \underset{m}{\equiv} y \rightarrow x \underset{m}{\not\equiv} y$

P7 $\quad x \underset{m/a}{\equiv} y \wedge (0 \leq x,y < m/a) \rightarrow x \underset{m}{\equiv} y$

P8 $\quad x \underset{m}{\equiv} y \rightarrow ax \underset{am}{\equiv} ay$

P9 $\quad a \mid m \wedge x \overset{m/a}{\equiv} y \rightarrow x \overset{m}{\equiv} y$

P10 $\quad a - x \underset{m}{\equiv} a - y \leftrightarrow x \underset{m}{\equiv} y$

P11 $\quad x \overset{am}{\equiv} y \rightarrow x \div a \overset{m}{\equiv} y \div a$

P12 $\quad x \overset{m}{\not\equiv} y \leftrightarrow ax \overset{am}{\not\equiv} ay$

We will tend to use these results in their contrapositive form. That is,

P2 $\quad ax \underset{m}{\not\equiv} ay \rightarrow x \underset{m}{\not\equiv} y$

Let R_n be an ordered set of factors of n, $R_n = \{\rho_1, \rho_2, \ldots, \rho_k\}$ such that $\rho_1 \times \rho_2 \times \ldots \times \rho_k = n$. Define $\Omega(R_n)$ to be the set

$$\Omega(R_n) = \{\omega_1 \mid \omega_k = 1,\ \omega_i = \omega_{i+1} \times \rho_{i+1},\ 0 < i \leq k-1\},\ \text{so}\ \omega_i = \prod_{j=i+1}^{k} \rho_j.$$

Note that $\omega_o = n$.

In general, any number $x < \omega_o$ can be expressed as

$$x = \sum_{i=1}^{k} x_i \times \omega_i,$$

where $x_i < \rho_i$. We say that $x_1x_2x_3 \ldots x_k$ is the "base Ω" representation of x.

Thus, if $\Omega = \{8,4,2,1\}$, then we get the familiar base 2 number system while if $\Omega = \{12,6,3,1\}$, then we get a somewhat unusual (but useful) number system where

$$0_{10} = 000_\Omega$$

$$1_{10} = 001_\Omega$$

$$2_{10} = 002_\Omega$$

$$3_{10} = 010_\Omega$$

$$4_{10} = 011_\Omega$$

$$5_{10} = 012_\Omega$$

$$6_{10} = 100_\Omega$$

$$7_{10} = 101_\Omega$$

$$8_{10} = 102_\Omega$$

$$9_{10} = 110_\Omega$$

$$10_{10} = 111_\Omega$$

$$11_{10} = 112_\Omega$$

Notice that

$$x \overset{\omega_j}{\equiv} y \rightarrow x_i = y_i \qquad 1 \leq i \leq j$$

and

$$x \overset{\omega_j}{\equiv} y \rightarrow x_i = y_i \qquad j + 1 \leq i \leq k.$$

For example, if $R_n = \{2,5,3\}$, then $\Omega(R_n) = \{30,15,3,1\}$, and

$$5_{10} = 012_\Omega$$

$$4_{10} = 011_\Omega$$

$$17_{10} = 102_\Omega$$

So

$$5 \overset{3}{\equiv} 4 \qquad (\underline{012}_{\Omega} \overset{3}{\equiv} \underline{011}_{\Omega})$$

$$5 \equiv 17 \quad (012_{\Omega} \equiv 102_{\Omega})$$
$$3 \qquad\qquad 3$$

$$5 \overset{15}{\not\equiv} 17 \qquad (\underline{012}_{\Omega} \overset{15}{\not\equiv} \underline{102}_{\Omega}).$$

In most of the following work, when R_n is understood or not important we will refer simply to Ω rather than $\Omega(R_n)$. Further, the unsubscripted lower case letters x, y, z, s, d are usually variables whose range is the set of non-negative integers. The notation {f(x):condition on x} denotes the set of all numbers in the range of the function f where the domain of f is all values of x which satisfy the specified condition. Thus, $\{2x:x \geq 0\}$ is the set of all even, non-negative integers. Subscripted letters usually refer to specific values of the variable.

2.2 The ↑ Relation and its Properties

An n-set represents a mapping or connection between n input nodes and n output nodes.

Definition 1: n-set

An n-set P is a set of pairs of integers, P = {(s,d)}. If $(s,d) \in P$, then we say P connects input s mod n to output d mod n. □

Thus, if P = {(2,1), (7,0), (2,0)} and n=4, then P represents the connection illustrated in Figure 3.

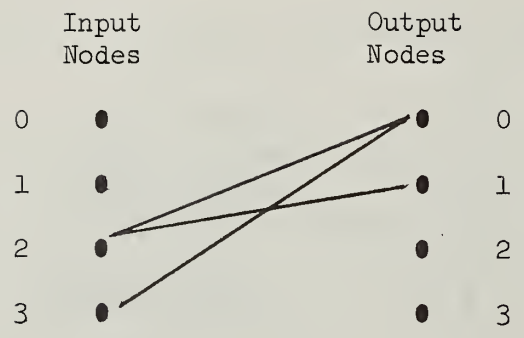


Figure 3. A Connection

Definition 2: $m \uparrow P$

Given an n-set P, we say the integer m passes P, written $m \uparrow P$, if an only if for all (s_i,d_i), $(s_j,d_j) \varepsilon P$

$$s_i \underset{n}{\not\equiv} s_j \text{ and } s_i \underset{m}{\not\equiv} s_j \rightarrow d_i \overset{m}{\not\equiv} d_j. \qquad \square$$

Definition 3: $\Omega \uparrow P$

Given an n-set P and $\Omega = \{\omega_0=n, \omega_1,\ldots, \omega_k=1\}$, if for all $\omega \varepsilon \Omega$, $\omega \uparrow P$, then we say $\Omega \uparrow P$. That is

$$\Omega \uparrow P \leftrightarrow (\forall\omega \varepsilon \Omega, \forall(s_i,d_i), \text{and } \forall(s_j,d_j)\varepsilon P, s_i \underset{n}{\not\equiv} s_j \wedge s_i \underset{\omega}{\equiv} s_j \rightarrow d_i \overset{\omega}{\not\equiv} d_j).$$

The negative form of $\Omega \uparrow P$ is useful and perhaps its statement will make the relation $\uparrow$ clearer.

$$\Omega \overline{\uparrow} P \leftrightarrow \exists\, \omega \varepsilon \Omega \text{ and } \exists(s_i,d_i),(s_j,d_j)\varepsilon P, \text{ such that } s_i \underset{n}{\not\equiv} s_j$$

$$\text{and } s_i \underset{\omega}{\equiv} s_j \text{ and } d_i \overset{\omega}{\equiv} d_j. \qquad \square$$

We will now demonstrate several functions f(P) on n-sets and show that

$$\Omega \uparrow P \rightarrow \Omega \uparrow f(P).$$

Theorem 1: Let $P + c = \{(s+c,d):(s,d)\varepsilon\ P,\ c \text{ an integer constant}\}$.

Then $\Omega \uparrow P \rightarrow \Omega \uparrow P+c$

We shall prove the contrapositive form of this theorem:

$\Omega \overline{\uparrow} P + c \rightarrow \Omega \overline{\uparrow} P$. We have

$\Omega \overline{\uparrow} P + c \rightarrow \exists\, \omega \varepsilon \Omega,\ \exists(s_i+c,\ d_i)$, and $\exists(s_j+c,\ d_j)$, such that

$s_i + c \underset{n}{\not\equiv} s_j + c$, $s_i + c \underset{\omega}{\equiv} s_j + c$, and $d_i \overset{\omega}{\equiv} d_j$. But

$$s_i + c \underset{n}{\not\equiv} s_j + c \overset{Pl}{\rightarrow} s_i \underset{n}{\not\equiv} s_j,$$

and*

$$s_i + c \underset{\omega}{\equiv} s_j + c \overset{Pl}{\rightarrow} s_i \underset{\omega}{\equiv} s_j$$

so $\Omega \overline{\uparrow} P$. □

Theorem 2: Let $P \times a = \{(x\times a,d):(s,d)\varepsilon\ P,\ a \text{ an integer constant}\}$.

If for all $\omega \varepsilon \Omega$, $\gcd(a,\omega) = 1$, then $\Omega \uparrow P \rightarrow \Omega \uparrow P\times a$.

---

* The notation Pl above the implication symbol means that the implication follows from property Pl found at the beginning of this chapter.

Proof:

$\Omega \overline{\uparrow} P \times a \rightarrow \exists\, \omega \in \Omega$, $\exists\ (s_i \times a, d_i)$, and $\exists\, (s_j \times a, d_j)$, such that

$s_i \times a \underset{n}{\not\equiv} s_j \times a$, $s_i \times a \underset{\omega}{\equiv} s_j \times a$, and $d_i \overset{\omega}{\equiv} d_j$. But

$s_i \times a \underset{n}{\not\equiv} s_j \times a \overset{P2}{\rightarrow} s_i \underset{n}{\not\equiv} s_j$, and if $\gcd(a,\omega) = 1$, then

$s_i \times a \underset{\omega}{\equiv} s_j \times a \overset{P3}{\rightarrow} s_i \underset{\omega}{\equiv} s_j$. Thus, $\Omega \overline{\uparrow} P$. □

Corollary 2.1: If $\gcd(a,n) = 1$, then $\Omega \uparrow P \rightarrow \Omega \uparrow P \times a$.

Proof:

We need only show that $\gcd(a,n) = 1 \rightarrow \forall\, \omega \in \Omega$, $\gcd(a,\omega) = 1$ and then apply Theorem 2. Assume there exists an $\omega \in \Omega$, such that

$$\gcd(a,\omega) = b \ .$$

Then

$$b \mid a \text{ and } b \mid \omega,$$

but recall that $\omega \mid n$ (from the definition of Ω), i.e., there exists an integer c such that

$$\omega c = n.$$

Since $b \mid \omega$ then $b \mid \omega c$ or $b \mid n$. Thus, $\gcd(a,n) \geq b$, and so

$$\gcd(a,n) = 1 \rightarrow \forall \omega \in \Omega,\ \gcd(a,\omega) = 1.$$ □

Theorem 3: Let $c - P = \{(c-s,d):(s,d)\varepsilon P\}$. Then $\Omega \uparrow P \rightarrow \Omega \uparrow c - P$.

Proof:

As before, $c - s_i \underset{\omega}{\equiv} c - s_j \overset{P10}{\rightarrow} s_i \underset{\omega}{\equiv} s_j$

so $\Omega \overline{\uparrow} c - P \rightarrow \Omega \overline{\uparrow} P$ □

Let $P = \{(ax + b, cx + d):0 \leq x < \xi\}$, where a,b,c and d are integer constants. We will now prove a theorem which states that if a,c and ξ satisfy certain conditions with respect to Ω, then $\Omega \uparrow P$.

Theorem 4:

Let $P \subseteq \{(ax + b, cx + d) \mid 0 \leq x < \xi\}$ be an n-set and define

$h = gpf(c,n)$

$\gamma_m = gcd(c,m)$

$\eta_m = gcd(c,a\gamma_m)$

$\delta_m = gpf(a\gamma_m/\eta_m,\ m/\gamma_m)$

$\zeta_m = gpf(a,m)$

$R_n = \{\rho_1,\rho_2,\ldots,\ \rho_k\}$

$\Omega(R_n) = \{m_o=n,m_1,m_2,\ldots,\ m_k=1\}$

μ = the largest $m \varepsilon \Omega$ such that $gcd(a,m) = m$.

If the following three conditions hold,

A: $c \leq n$

B: $ah/c \geq 1$ or $\xi \leq hn/c$

C: For all $m \in \Omega$, $\mu \leq m < n$

C1: $a/(\delta_m \eta_m) \leq 1$, or

C2: $\xi \leq_m \zeta_m/a$,

then $\Omega \uparrow P$.

The proof will consist of two parts and several subparts. First, we need a preliminary result. Let $(ax + b, cx + d)$, $(ay + b, cy + d)$ be any two elements of P. Then we will show

$$ax + b \underset{n}{\not\equiv} ay + b \rightarrow cx + d \underset{n}{\not\equiv} cy + d$$

We have

$$cx + d \underset{n}{\equiv} cy + d \overset{P1}{\rightarrow} cx \underset{n}{\equiv} cy$$

$$\overset{P3}{\rightarrow} \frac{cx}{h} \underset{n}{\equiv} \frac{cy}{h} \text{ where } h = gpf(c,n)$$

Now since $c < n$ and $h = gpf(c,n)$, then $\frac{c}{h} | n$. So

$$\overset{P4}{\rightarrow} x \underset{\frac{hn}{c}}{\equiv} y$$

Multiplying by a

$$\overset{P8}{\rightarrow} ax \underset{\frac{ahn}{c}}{\equiv} ay.$$

Suppose $\frac{ah}{c} \geq 1$. Then $\overset{P5}{\rightarrow} ax \underset{n}{\equiv} ay$. If $\frac{ah}{c} < 1$, then since $x,y < \xi \leq \frac{hn}{c}$ (condition B), then $ax,ay < \frac{ahn}{c}$. Thus,

$$\overset{P7}{\rightarrow} \; ax \underset{n}{\equiv} ay.$$

Finally,

$$\overset{P1}{\rightarrow} \; ax + b \underset{n}{\equiv} ay + b.$$

Thus,

$$ax + b \underset{n}{\not\equiv} ay + b \rightarrow cx + d \underset{n}{\not\equiv} ay + d.$$

Now we will prove that if for all $m \in \Omega$ where $u \leq m < n$, condition C1 or C2 holds, then for all $m \in \Omega$

$$ax + b \underset{n}{\not\equiv} ay + b \wedge ax + b \underset{m}{\equiv} ay + b \rightarrow cx + d \overset{m}{\not\equiv} cy + d$$

and thus $\Omega \uparrow P$. Specifically, we shall prove the equivalent theorem

$$ax + b \underset{n}{\not\equiv} ay + b \wedge cx + d \overset{m}{\equiv} ay + d \rightarrow ax + b \underset{m}{\not\equiv} ay + b.$$

Using our previous result, we have from condition B

$$ax + b \underset{n}{\not\equiv} ay + b \rightarrow cx + d \underset{n}{\not\equiv} cy + d.$$

Then, by P6 we get

$$cx + d \underset{n}{\not\equiv} cy + d \wedge cx + d \overset{m}{\equiv} ay + d \overset{P6}{\rightarrow} cx + d \underset{m}{\not\equiv} cy + d.$$

$$\overset{P1}{\rightarrow} \; cx \underset{m}{\not\equiv} cy$$

$$\overset{P8}{\rightarrow} \; \frac{cy}{\gamma_m} \not\equiv \frac{cy}{\gamma_m} \pmod{\frac{m}{\gamma_m}} \qquad \text{where } \gamma_m = \gcd(c,m)$$

Now

$$\eta_m = \gcd(c, a\gamma_m).$$

Let

$$\nu_m = c/\eta_m , \qquad u_m = a\gamma_m/\eta_m$$

Then since $\delta_m = \mathrm{gpf}(u_m, m/\gamma_m)$

$$\overset{P4,3}{\rightarrow} \frac{u_m cx}{\gamma_m} \underset{\frac{mu_m}{\delta_m\gamma_m}}{\not\equiv} \frac{u_m cy}{\gamma_m}$$

$$\overset{P2}{\rightarrow} \frac{u_m cx}{\nu_m\gamma_m} \underset{\frac{u_m m}{\delta_m\gamma_m}}{\not\equiv} \frac{u_m cy}{\nu_m\gamma_m}$$

But since

$$\frac{u_m}{\nu_m\gamma_m} = \frac{a\gamma_m}{\gamma_m\eta_m}\frac{\eta_m}{c} = \frac{a}{c}$$

we have

$$ax \underset{\frac{u_m m}{\delta_m\gamma_m}}{\not\equiv} ay$$

Now suppose C1 holds, i.e., $u_m/\delta_m\gamma_m \leq 1$. Then

$$\overset{P5}{\rightarrow} ax \underset{m}{\not\equiv} ay$$

and

$$\overset{P1}{\rightarrow} ax + b \underset{m}{\not\equiv} ay + b.$$

Suppose now that C1 does not hold, i.e., $u_m/\delta_m\gamma_m > 1$, but C2 does hold, i.e.,

$$\xi \leq \frac{m\zeta_m}{a}$$

Let $\beta = u_m/\delta_m\gamma_m$. Then we have

$$ax \underset{\beta m}{\not\equiv} ay \qquad \beta > 1$$

$$\overset{P2}{\rightarrow} \frac{a}{\zeta_m} x \underset{\beta m}{\not\equiv} \frac{a}{\zeta_m} y \quad ,$$

where

$$\zeta_m = gpf(a,m).$$

Since $x < \xi \leq \frac{m\zeta_m}{a}$, then $ax, ay < m\zeta_m$

and

$$\overset{P7}{\rightarrow} \frac{ax}{\zeta_m} \underset{m}{\not\equiv} \frac{ay}{\zeta_m}$$

$$\overset{P3}{\rightarrow} ax \underset{m}{\not\equiv} ay$$

where

$$\zeta_m = gpf(a,m) \rightarrow gcd(\zeta_m, m) = 1$$

Finally,

$$\overset{P1}{\rightarrow} ax + b \underset{m}{\not\equiv} ay + b.$$

Now, suppose μ is the largest $m \in \Omega$ such that $gcd(a,m) = m$, and that C1 or C2 hold for all m, $n > m \geq \mu$. We will now show that for all $m \leq \mu$, $cx + b \overset{m}{\not\equiv} cy + b$. Let $m \in \Omega$ be $\leq \mu$ such that $m = \mu/\rho$ as per the definition of Ω. Since $gcd(a,\mu) = \mu$,

$$ax + b \underset{\mu}{\equiv} ay + b$$

must be true always. Since C1 or C2 hold for μ, we have

$$ax + b \underset{\mu}{\equiv} ay + b \rightarrow cx + d \overset{\mu}{\not\equiv} cy + d.$$

By P9

$$cx + d \overset{\mu}{\not\equiv} cy + d \rightarrow cx + d \overset{\mu/\rho}{\not\equiv} cy + d.$$

Since the original premise is true, then the final conclusion must be true, and since $m = \mu/\rho$,

$$cx + d \overset{m}{\not\equiv} cy + d.$$

Now, in the special case where $\mu = n$, then neither C1 nor C2 need hold. But since $\gcd(a,n) = n$, then

$$ax + b \overset{n}{\equiv} ay + b$$

must be true always. Thus, $\Omega \uparrow P$ trivially. □

Theorems 1, 2, and 3 tell us that if $\Omega \uparrow P$, then $\Omega \uparrow P+c$, $\Omega \uparrow c-P$ and if $\gcd(a,n) = 1$, then $\Omega \uparrow P\times a$. However, they tell us nothing about whether or not $\Omega \uparrow P$. Theorem 4 tells us that if P satisfies certain conditions with respect to Ω, then $\Omega \uparrow P$. We will use all of these theorems later. First, we will present an algorithm for constructing a switching network from a given Ω. We will then conclude this chapter by showing that if $\Omega \uparrow P$, then the network constructed from Ω can produce without conflict the input-output connections specified by P.

2.3 Construction of a Network from Ω

Algorithm 1: Construction of a network from Ω.

Let $\Omega(R_n)$ be as defined earlier, i.e., $\Omega(R_n)$ is the set

$$\Omega(R_n) = \{\omega_i : \omega_k = 1,\ \omega_i = \prod_{j=i+1} \rho_j,\ 0 \leq i \leq k - 1\}.$$

The network will consist of k stages numbered 1,2,..., k from left to

right. The i^{th} stage will consist of n/ρ_i crossbar switches*, each switch being $\rho_i \times \rho_i$. Refer to Figure 4 and number the inputs (left) and outputs (right) of each stage from 0 to n - 1.

Now, for $1 \leq i \leq k-1$, connect output j of stage i to input ℓ of stage i + 1 where

$$\ell = (j \div \omega_{i-1}) \times \omega_{i-1} + (j \bmod \rho_i) \times \omega_i + (j \bmod \omega_{i-1}) \div \rho_i$$

(Recall $x \div y$ is the integer part of the quotient x/y).

Finally, the actual network inputs must be connected to the stage 1 inputs (actually this will be shown in the figures as a renumbering of the stage 1 inputs). Let

$$\hat{R}_n = \{\rho_k, \rho_{k-1}, \ldots, \rho_2, \rho_1\}$$

where

$$R_n = \{\rho_1, \rho_2, \ldots, \rho_{k-1}, \rho_k\},$$

and form

$$\Omega(R_n) = \{\omega_i : \omega_k = 1,\ \omega_i = \omega_{i+1}\ \rho_{i+1},\ 0 \leq i \leq k-1\}$$

and

$$\Omega(\hat{R}_n) = \{\omega_i : \omega_{k=1},\ \omega_i = \omega_{i+1}\ \rho_{k-i+1},\ 0 \leq i \leq k-1\}$$

Define a function $\tau(x)$ on $0 \leq x \leq n-1$ such that if $x = (x_1 x_2 \ldots x_k)_{\hat{\Omega}}$, then $\tau(x) = (x_k \ldots x_2 x_1)_{\Omega}$. Now connect (or renumber) stage 1 input x to $\tau(x)$. □

Figures 5, 6, and 7 are examples of networks constructed in this way.

* A crossbar switch is capable of producing any one-to-one or one-to-many connection of inputs to outputs. An attempt to produce a many-to-one connection results in a "conflict."

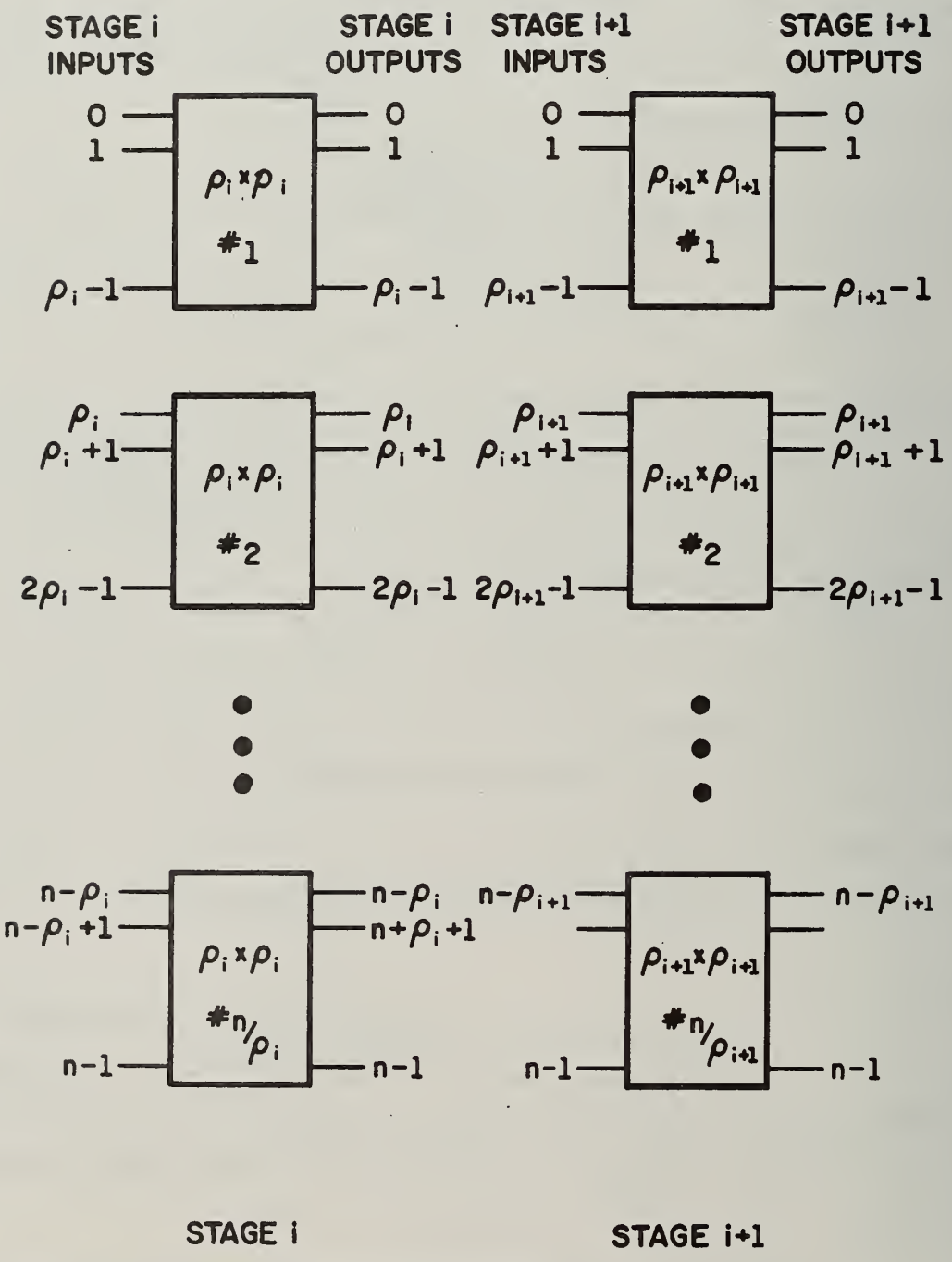


Figure 4. Stages of a Network Constructed from Ω

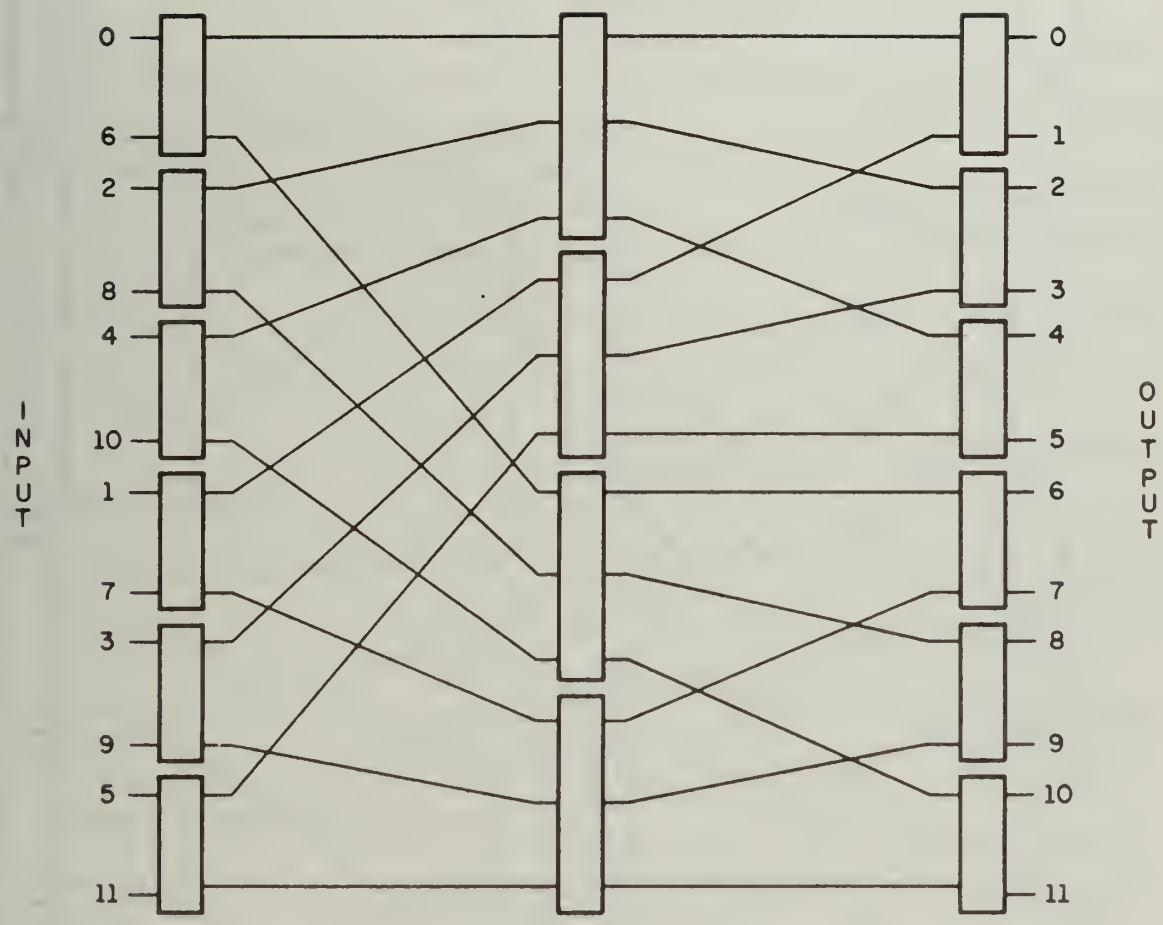


Figure 5. Network Constructed from $\Omega = \{12, 6, 2, 1\}$

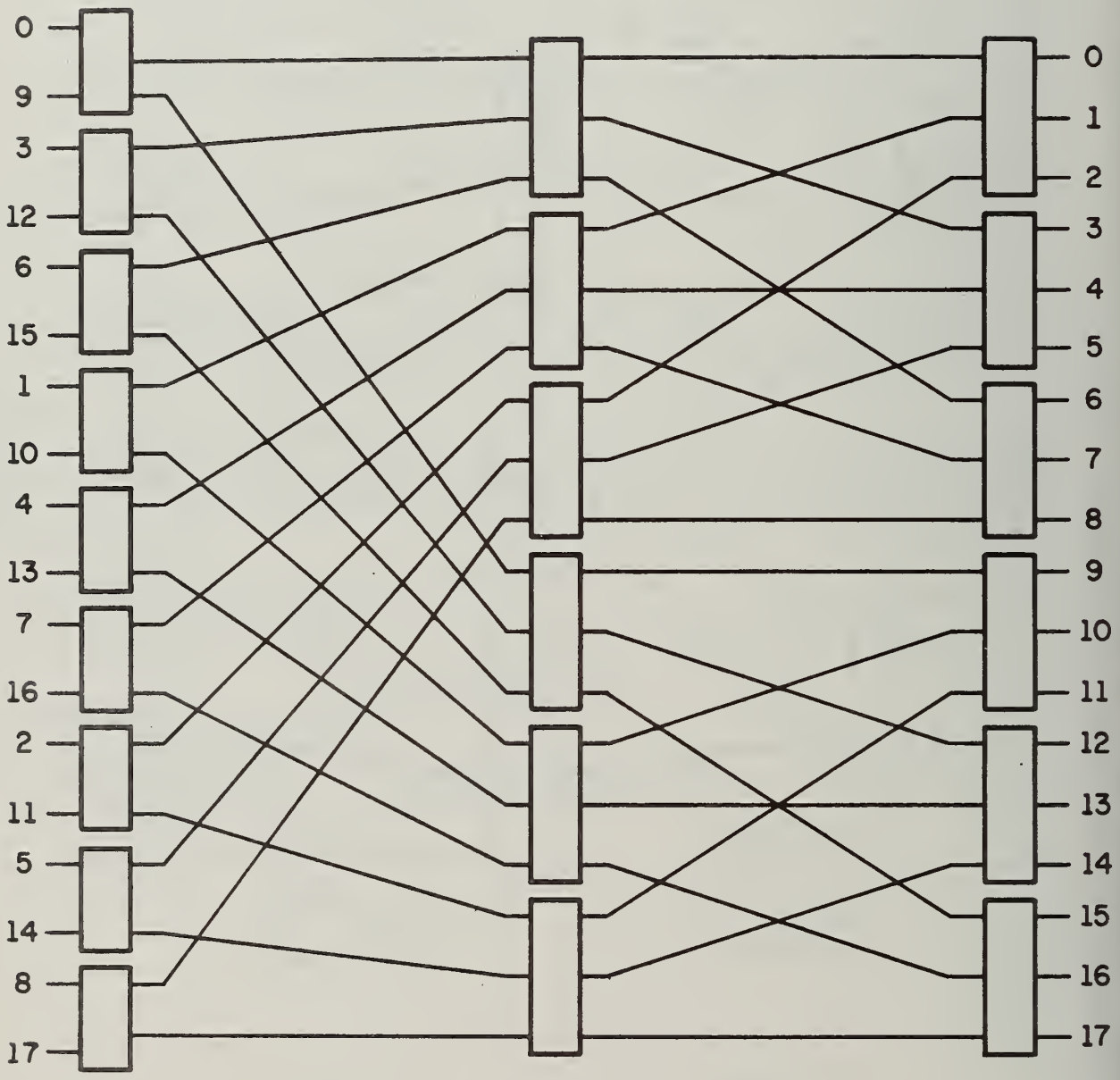


Figure 6. Network Constructed from $\Omega = \{18, 9, 3, 1\}$

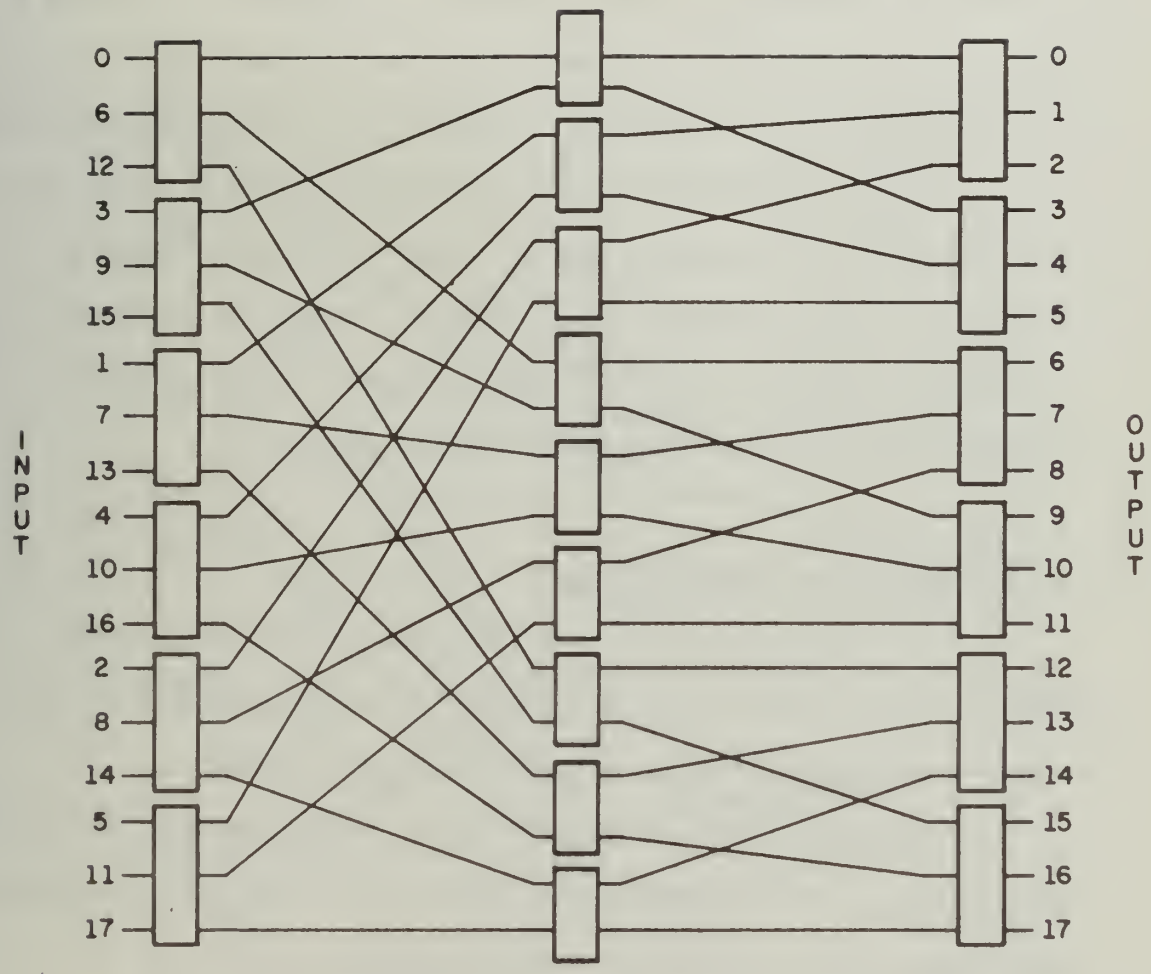


Figure 7. Network Constructed from $\Omega = \{18, 6, 3, 1\}$

Algorithm 2: Network Control Algorithm

Each (s,d) pair in P establishes a path through the network as follows. Assume s and d are expressed in base Ω notation, i.e., $s = s_1 s_2 \cdots s_k$, $d = d_1 d_2 \cdots d_k$. Then, starting at input s, for $i = 1, 2, \ldots, k$, each (s,d) pair "enters" a $\rho_i \times \rho_i$ crossbar switch and is connected to output d_i of that crossbar. Alternately, starting at output d for $i = k, k-1, \ldots, 1$, each (s,d) pair "enters" an output of a $\rho_i \times \rho_i$ crossbar and is connected to input s_{k-i+1} of that crossbar switch. □

Figure 8 shows the necessary switching of the $\Omega = \{18, 6, 3, 1\}$ network for $P = \{(0,7), (0,9), (7,16), (16,13)\}$ where $7 = 101_\Omega$, $9 = 110_\Omega$, $13 = 201_\Omega$, and $16 = 211_\Omega$. Figure 9 shows the switching for $P = \{(12,15), (15,16)\}$, where $15 = 210_\Omega$, and $16 = 211_\Omega$. Note the conflict at the output of stage 2: $H(\Omega) \overline{\uparrow} P$ and $\Omega \overline{\uparrow} P$.

2.4 The Equivalence of $H(\Omega)$ and Ω

It remains for us to show that this network is in some way equivalent to Ω. That is, we would like to show that if $\Omega \uparrow P$, then the network constructed from Ω "passes" P, and conversely.

Each output of each stage of the Ω network is only accessible to elements of P having certain characteristics. This follows from the construction algorithm and the control algorithm. Let $(xx\ldots x\ a_{j+1} \ldots a_k)_\Omega$ represent the set of all integers z such that

$$z \underset{\omega_j}{\equiv} (a_{j+1}\ a_{j+2} \cdots a_k)_\Omega,$$

and $(a_1 a_2 \ldots a_j\ xx \ldots x)$ represent the set of all integers z, such that

$$z \overset{\omega_j}{\equiv} (a_1 a_2 \ldots a_j\ 00 \ldots 0)_\Omega.$$

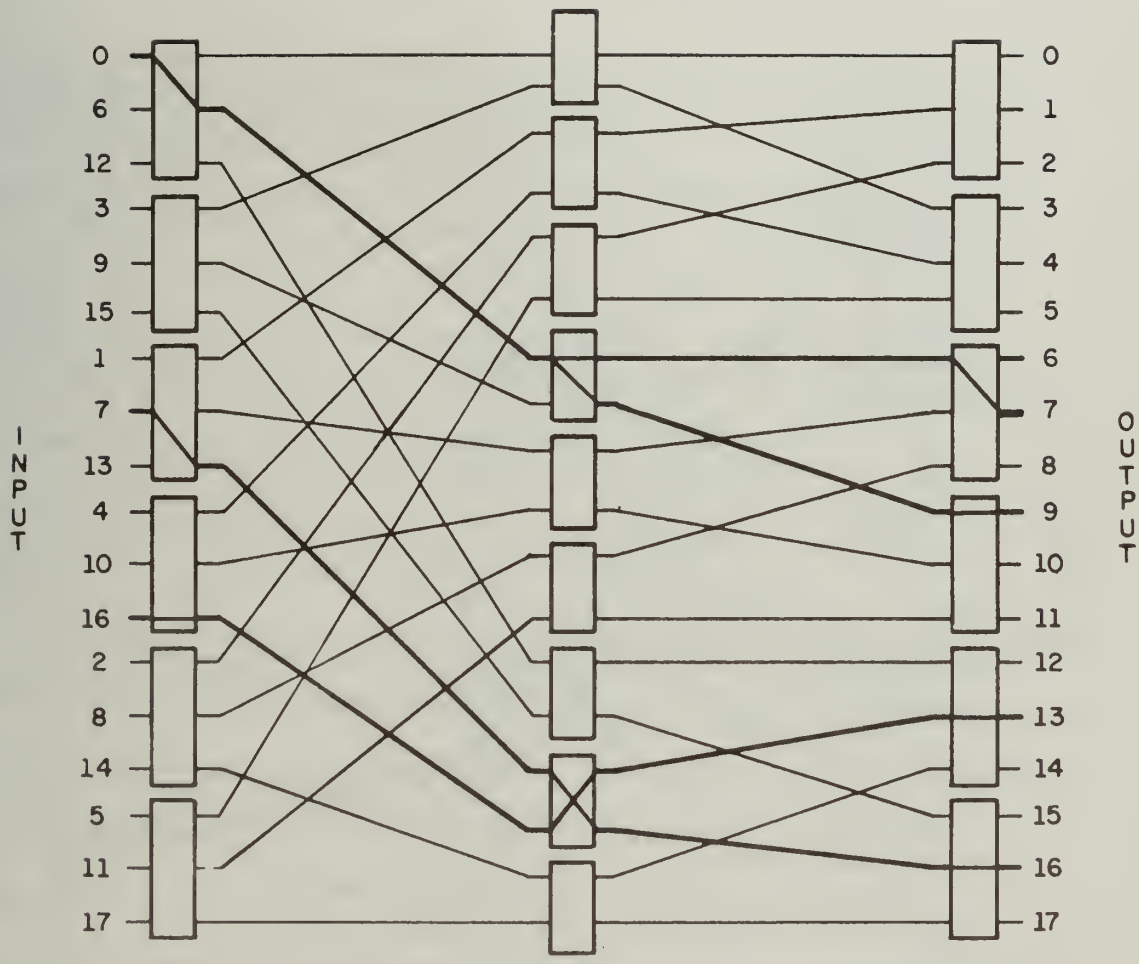


Figure 8. Ω = {18, 6, 3, 1} Switching for P = {(0,7), (0,9), (7,16), (16,15)}

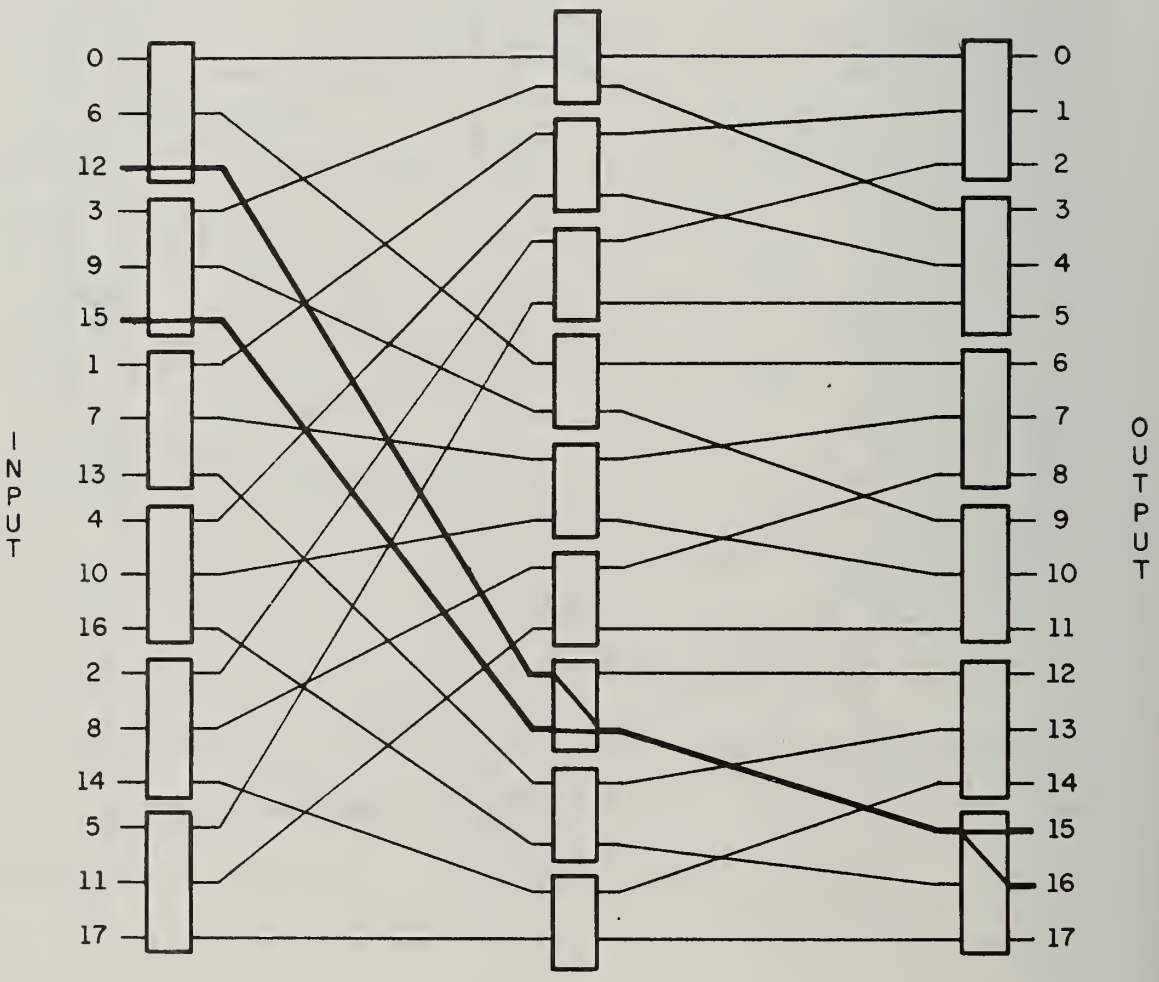


Figure 9. Switching with Conflict for P = {(12,15), (15,16)}

Finally, $(xx\ldots\ x\ s_{j+1}\ s_{j+2}\cdots\ s_k,\ d_1d_2\ldots\ d_j\ xx\ldots\ x)$ represents the set of all (s,d) such that

$$s \underset{\omega_j}{\equiv} (s_{j+1}\ s_{j+2}\ \cdots\ s_k)_\Omega$$

and

$$d \overset{\omega_j}{\equiv} (d_1d_2\ldots\ d_j\ 00\ldots\ 0)_\Omega.$$

Now each output of each stage of the network can be labeled with the representation of the set of (s,d) pairs which are accessible to that output. This is shown in Figure 10 for a specific network.

Theorem 5:

Let $H(\Omega) \uparrow P$ denote the fact that the network $H(\Omega)$ constructed from Ω passes or connects P. Then $\Omega \uparrow P \leftrightarrow H(\Omega) \uparrow P$

Proof:

Assume $H(\Omega) \overline{\uparrow} P$. Then there must be a conflict at the output of one of the crossbar switches. That is, two pairs (s_1,d_1), $(s_2,d_2) \in P$, $s_1 \underset{n}{\not\equiv} s_2$, must be trying to use the same output. Without loss of generality, assume this output is labeled

$$(xx\ldots\ x\ a_{j+1}\ a_{j+2}\cdots\ a_k,\ b_1b_2\ldots\ b_j\ xx\ldots\ x).$$

Then (s_1,d_1) and (s_2,d_2) must both be elements of the set represented by this output label. This implies

$$s_1 \underset{\omega_j}{\equiv} s_2 \text{ and } d_1 \overset{\omega_j}{\equiv} d_2.$$

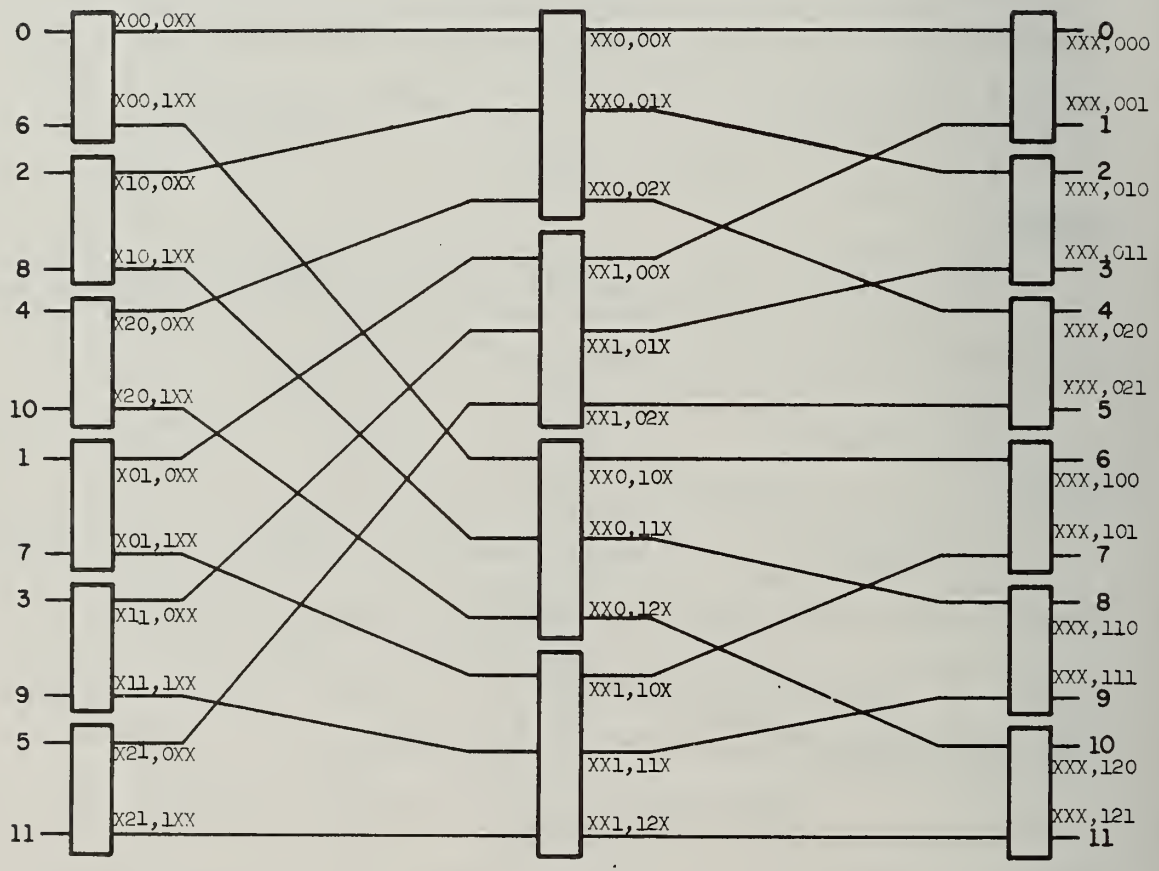


Figure 10. Network for Ω = {12, 6, 2, 1} with Stage Outputs Labelled with (s,d) Classes

But this implies $\Omega \overline{\uparrow} P$. So we have

$$H(\Omega) \overline{\uparrow} P \rightarrow \Omega \overline{\uparrow} P$$

or

$$\Omega \uparrow P \rightarrow H(\Omega) \uparrow P.$$

By a similar argument it is easy to show that if $\Omega \overline{\uparrow} P$, then $H(\Omega) \overline{\uparrow} P$, and thus we have

$$\Omega \uparrow P \leftrightarrow H(\Omega) \uparrow P. \qquad \square$$

Thus far in this chapter we have defined and explored a relation $\uparrow$ between a special kind of set $\Omega(R_n)$ and an arbitrary set of pairs P which represents a set of connections between two sets of nodes. We have shown how to construct a switching network $H(\Omega(R_n))$ from an $\Omega(R_n)$, and we have shown that if $\Omega \uparrow P$, then $H(\Omega) \uparrow P$ and conversely. Thus, given a network $H(\Omega)$ and an n-set P we have some useful tools for determining whether or not $H(\Omega)$ can produce the connections specified by P.

2.5 Minimal Ω Networks

Before leaving this chapter we will present three more results which will be needed in later chapters.

Let

$$R_n = \{\rho_1, \rho_2, \ldots, \rho_i, \rho_{i+1}, \ldots, \rho_k\}$$

and

$$R'_n = \{\rho_1, \rho_2, \ldots, \rho_i \times \rho_{i+1}, \ldots, \rho_k\}$$

Thus, R_n has k elements while R'_n has k - 1 elements.

Theorem 6: $\quad \Omega(R_n) \uparrow P \rightarrow \Omega(R'_n) \uparrow P$

Proof:

Assume $\Omega' \overline{\uparrow} P$. Then $\exists\ \omega \in \Omega'$ and $\exists(s_1,d_1)$, and $\exists(s_2,d_2) \in P$ such that $s_1 \overset{\omega}{\not\equiv}_n s_2$ and $s_1 \overset{\omega}{\equiv} s_2$ and $d_1 \overset{\omega}{\equiv} d_2$. But then $\Omega \uparrow P$, so we have

$$\Omega(R_n) \uparrow P \rightarrow \Omega(R'_n) \uparrow P \qquad \square$$

Theorem 7: The network constructed from $\Omega(R_n)$ has the same number or fewer gates than the network constructed from $\Omega(R'_n)$.

Proof:

A $\rho \times \rho$ crossbar switch has order of ρ^2 gates. So the number of gates in $H(\Omega)$ is just

$$g(\Omega) = \sum_{i=1}^{k} (\rho_i)^2\, n/\rho_i = n \sum_{i=1}^{k} \rho_i,$$

while the number of gates in $H(\Omega')$ is

$$g(\Omega') = \rho_1 n + \rho_2 n + \ldots + \rho_{i-1} n + \rho_i \rho_{i+1} n + \ldots + \rho_k n.$$

Since for $\rho_i,\ \rho_{i+1} > 1$,

$$\rho_i \rho_{i+1} \geq \rho_i + \rho_{i+1}$$

it follows that

$$g(\Omega) \leq g(\Omega'). \qquad \square$$

Notice the special case of Theorem 7 where $\rho_i = \rho_{i+1} = 2$. Then

$$g(\Omega) = g(\Omega')$$

since

$$\rho_i + \rho_{i+1} = \rho_i\, \rho_{i+1}.$$

Theorem 8: Minimal Network

Let $\rho_1, \rho_2, \ldots, \rho_k$ be a prime factorization of n; that is, $\rho_1 \rho_2 \rho_3 \cdots \rho_k = n$, and ρ_i prime numbers. Then the network constructed

from

$$\Omega = \{\omega_i : \omega_i = \rho_{i+1}\ \omega_{i+1},\ 0 \leq i \leq k-1,\ \omega_0 = n\}$$

is minimal for n in terms of gates.

Proof:

This follows easily from Theorem 7. If

$$\Omega' = \{\omega_i : \omega_i = \rho_{i+1}\ \omega_{i+1},\ 0 \leq i \leq k-1,\ \omega_0 = n\},$$

and for some $i = \ell$, $\rho_{\ell+1}$ is not prime, i.e., $\rho_{\ell+1} = ab$, then the Ω generated from $\rho_1, \rho_2, \ldots, \rho_\ell, a, b, \rho_{\ell+1}, \ldots, \rho_k$ yields a network with equal or fewer gates than Ω'. Continued application of this until each ρ_i is prime, therefore, results in a minimal network. □

2.6 Summary

In this chapter we have developed some basic results relating the concept of an Ω set to a class of switching networks. In particular, we showed how to construct a network from Ω and we proved sufficient conditions on P relative to Ω such that $\Omega \uparrow P$. In review

Theorem 1: $\Omega \uparrow P \rightarrow \Omega \uparrow P + c$

Corollary 2.1: $\gcd(a,n) = 1,\ \Omega \uparrow P \rightarrow \Omega \uparrow P \times a$

Theorem 3: $\Omega \uparrow P \rightarrow \Omega \uparrow c - P$

Theorem 5: $\Omega \uparrow P \leftrightarrow H(\Omega) \uparrow P$

Theorem 6: $\Omega \uparrow P \rightarrow \Omega' \uparrow P$

Theorem 7: $g(\Omega) \leq g(\Omega')$

Theorem 8: If all $\rho_i \in R_n$ are prime numbers, then $H(\Omega(R_n))$ is a (gate) minimal network.

In the following chapters we shall examine n-sets which are needed in computer programs, and we will examine in more detail the construction of some networks. In particular, we will compare Ω network with other classes of networks in terms of capability and cost.

3. EFFECTIVENESS OF Ω NETWORKS

One of central problems in utilizing parallel computers is in alignment of data in the memory system. In this paper we are restricting ourselves to the primary memory problem. Here the problem is two-fold:

1. If x and y are two data elements which are required at the same time, then x and y should be stored in separate memory modules in order to avoid a "memory conflict."
2. If x is required by processor p, then x should be stored in a memory module which is "readily accessible" to p via the processor-memory connection network.

For example, refer to Figure 11, which shows an $N \times N$ array stored "straight" in N memories, i.e., element $a_{i,j}$ is stored in memory j. An attempt to fetch a column of this array would result in memory conflicts since the elements of any given column are stored in one memory. In addition, data is only accessible to processor p, $0 \leq p < n-1$ if the data is in memories p or $p \pm 1$ since each processor is only connected to the three closest memories.

In order to solve the first problem, a number of special storage schemes have been proposed (Budnik [2], Kraska [3], Muraoka [8]). In general, these schemes place data in the memories in such a way that the most desirable N-vectors can be fetched without memory conflict.

The purpose of this paper is to solve both of these problems simultaneously. We begin with the following assumptions. We assume that there are N processors and $M \geq N$ memories. The processors and memories are connected together by an $n \times n$ network. The data is an $N \times N$ array where the (i,j)-th element is logically in the i-th row, j-th column of the matrix space.

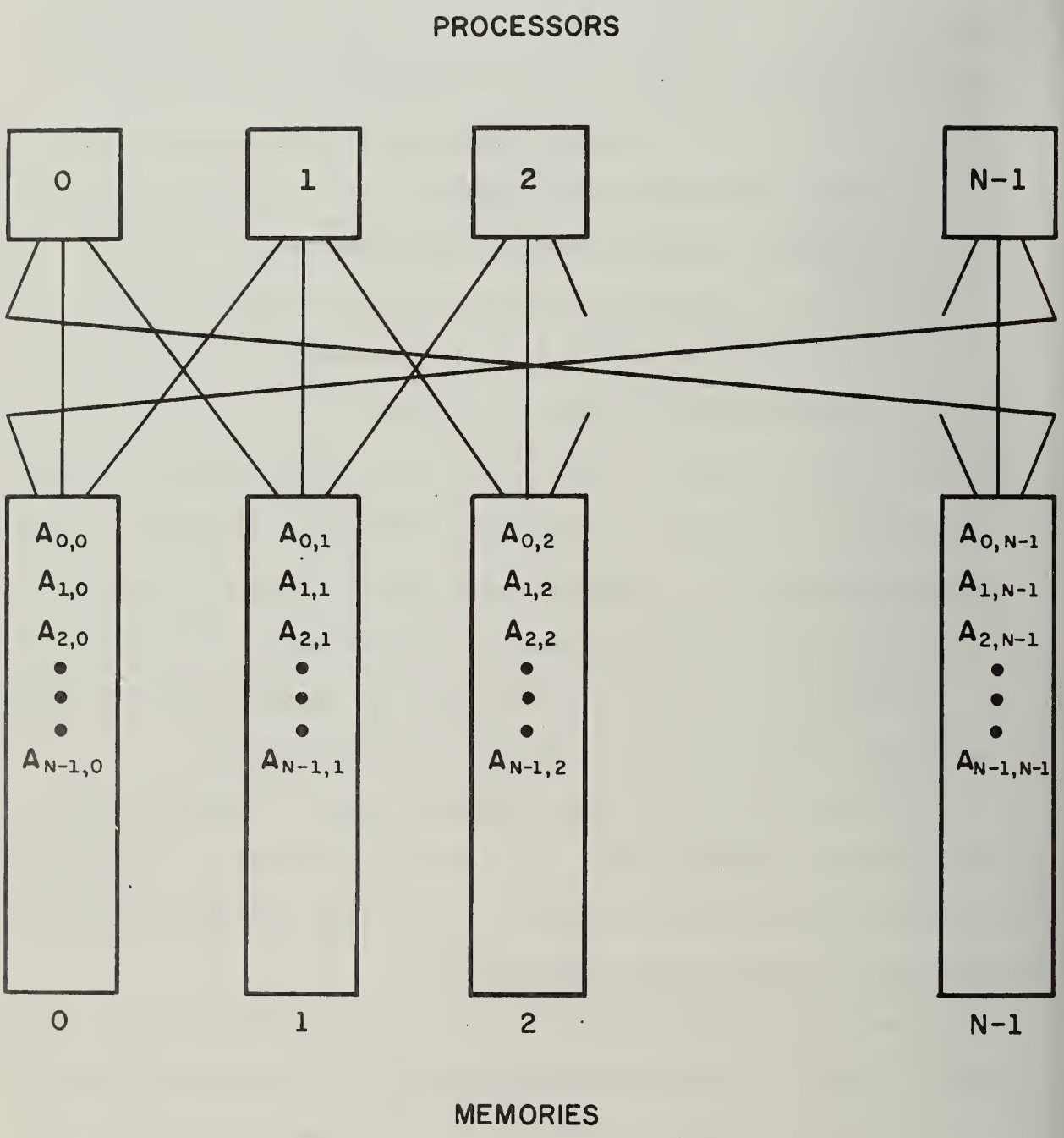


Figure 11. An Example of an N × N Array Stored Straight in a Parallel Memory-Processor System

3.1 Representation of Array Storage Schemes and N-Vectors

Definition 4: Memory Equation

We define the function $\mu(i,j)$ to yield the memory number where data with coordinates i,j is stored. □

For example, if an array is stored "straight", then $\mu(i,j) = j$, whereas if it is stored with skew = 1, then $\mu(i,j) = i + j \pmod M$. In general, we will consider only linear equations $\mu(i,j) = ax + bj + c \pmod M$, where a is called the skew and b is the skip.

Memory equations for some of the more popular storage schemes are given below:*

straight:	j
1-skew:	$i + j$
$\sqrt{N}$-skew:	$\sqrt{N}\, i + j$
1-skew, 2-skip:	$i + 2j$
2-skew, 1-skip:	$2i + j$
$\sqrt{N}$+1-skew:	$(\sqrt{N}+1)i + j$

Definition 5: Vector Equation

The N processors operate on N-vectors of data where the x-th element of the N-vector goes to the x-th processor. We define v(x) to be a function which yields the i,j coordinates of the x-th element of the N-vector. The vector equation for some of the common N-vectors are shown below:+

---

*All arithmetic is mod M.

+All arithmetic is mod N. N is assumed to be a perfect square where necessary.

1. row: $v(x) = (i_o, j_o + x)$

2. column: $v(x) = (i_o + x, j_o)$

3. forward diagonal: $v(x) = (i_o + x, j + x_o)$

4. reverse diagonal: $v(x) = (i_o + x, j - x_o)$

5. $\sqrt{N} \times \sqrt{N}$ partition: $v(x) = (i_o + x \div \sqrt{N}, j_o + x \bmod \sqrt{N})$

6. N broadcast: $v(x) = (i_o, j_o)$

7. $\sqrt{N}$ row broadcast: $v(x) = (i_o, j_o + (x \div \sqrt{N}) \times \sqrt{N})$

8. $\sqrt{N}$ column broadcast: $v(x) = (i_o + (x \div \sqrt{N}) \times \sqrt{N}, j_o)$ □

We should make it clear at this point just what the vector equation means. Notice that i_o, j_o in the above equations determine the first element of the N-vector. For example, if $i_o = 1$ and $j_o = 2$, then the row equation specifies the row beginning with element (1,2). Thus, each vector equation above actually represents a class of vector equations and a particular equation is determined by a choice of i_o and j_o. An N-vector is really represented by a function $v(x, i_o, j_o)$ for all x. For the sake of brevity, we will delete the subscript o on i and j in the following pages.

Combining a vector equation representing a given N-vector with the memory equation which determines the storage map for the data, we get

$$\mu(v(x)) : 0 \leq x \leq N-1 ,$$

which is the memory in which the x-th element of the N-vector is stored. The set

$$\{\mu(v(x)) : 0 \leq x < N\}$$

thus represents the set of memories which contain the given N-vector.

Later we will define a "network connection equation" which determines the memory-processor connection which is required for N processors to access a given N-vector in a particular order. This network connection equation will depend primarily on the memory and vector equations. We will then evaluate various connection networks to determine whether or not they can produce the necessary or common memory-processor connections. Obviously then, we must first know which N-vectors are necessary or common.

3.2 Important N-Vectors

Let us assume for a moment that we intend to build a parallel processor and that we know exactly which problems will be solved on this machine. We could then analyze these problems to determine which N-vectors are important. Using these results, we could then determine the effectiveness of any given connection network in producing the connections determined by these N-vectors.

Unfortunately, there are several problems here. First, we do not have a specific problem set in mind. Second, the determination of important N-vectors from a given problem set is not trivial. In fact, there generally exists more than one algorithm for solving any given problem and each algorithm may require different types of N-vectors. For example, one algorithm for solving the fast Fourier transform requires very strange memory-processor connections (Pease [9]) but a trivial change in this algorithm requires only simple shifts (Stevens [10]).

So where does this leave us? It is our purpose only to present a new connection network and show it is a plausible solution to the memory-processor connection problem. Thus, we will only consider those vector equations and memory equations listed above. We must leave it to the reader to

apply the results of the previous chapter in evaluating the effectiveness of Ω networks on other vector-memory equations.

The vector equations listed above represent some of the N-vectors which are most frequently encountered in programming the ILLIAC IV computer. Of these, the first two, rows and columns, are so important that we will disregard any system which cannot handle them. Diagonals are also important in many applications. $\sqrt{N}$ partitions represent any submatrix of size equal or less than $\sqrt{N} \times \sqrt{N}$. Of course, sometimes rectangular partitions are referenced where one of the dimensions is greater than $\sqrt{N}$, but we will not consider them here. N broadcast is simply one element broadcast to all the processors. $\sqrt{N}$ broadcast is every $\sqrt{N}$-th element of a row or column sent to $\sqrt{N}$ consecutive processors. This pattern is frequently encountered when, for reasons of storage efficiency, a large matrix is partitioned into $\sqrt{N} \times \sqrt{N}$ blocks and operations are performed in parallel on these blocks.

3.3 Memory Conflicts, Memory and Vector Equations, and the Network Connection Equation

So far we have a memory equation and a vector equation. Thus, the memory where the x-th element of an N-vector is stored is simply $\mu(v(x))$.

Definition 6: Memory Conflict

We say the N-vector $v(x)$ can be fetched without conflict if and only if $x \neq y$ $(0 \leq x, y < N)$ and $\mu(v(x)) \underset{M}{\equiv} \mu(v(y)) \rightarrow v(x) = v(y)$. That is, if two elements of the N-vector $v(x)$ and $v(y)$ occur in the same memory element, then there will be a memory conflict unless $v(x)$ and $v(y)$ are the same. □

Assume $\mu(v(x)) = ax + b$, $0 \leq x < \ell$. A sufficient condition that $v(x)$ can be fetched without conflict is the following.

Theorem 9: If $\mu(v(x)) = ax + b$, $0 \leq x < \ell$, and if

$$\ell \leq M\,\alpha/a$$

where $\alpha = gpf(a,M)$,

then the ℓ-vector $v(x)$, $0 \leq x < \ell$, can be fetched without conflict.

Proof:

We need only to show that for all $0 \leq x, y < \ell$,

$$x \neq y \rightarrow ax + b \underset{M}{\not\equiv} ay + b \ .$$

We have $x \neq y$ and $x,y < \ell \leq \frac{M\alpha}{a} \leq M$

$$\rightarrow x \underset{M}{\not\equiv} y$$

P4
$$\rightarrow \frac{a}{\alpha} x \underset{\frac{aM}{\alpha}}{\not\equiv} \frac{a}{\alpha} y$$

P7
$$\rightarrow \frac{a}{\alpha} x \underset{M}{\not\equiv} \frac{a}{\alpha} y \qquad \text{since } x,y < \frac{M\,\alpha}{a}$$

P3
$$\rightarrow ax \underset{M}{\not\equiv} ay \qquad \text{since } \alpha = gpf(a,M) \text{ then } gcd(\alpha,M) = 1$$

P1
$$\rightarrow ax + b \underset{M}{\not\equiv} ay + b$$

□

Next we need to specify the connection between network ports and memories.

Definition 7: Memory Connection Equation

We define I(m) to be the network port connected to memory m. □

In most cases I(m) = m.

Thus, the network port at which the x-th element of an N-vector will appear is simply

$$I(\mu(v(x))).$$

Now we have said that the x-th element of the N-vector must go to processor x. In order to determine the output port to which this element must be sent we must specify how the processors are connected to the network ports.

Definition 8: Processor Connection Equation

We define $\Phi(x)$ to be the port(s) attached to processor x. □

In most cases $\Phi(x) = x$. However, in case $n = M = 2N$, $\Phi(x)$ is multivalued and may have one of the following two forms:

$$\Phi(x) = 2x \text{ and } 2x + 1$$

or $$\Phi(x) = x \text{ and } x + N.$$

We now have enough information to specify the network connections $P = \{(s,d)\}$ required to send each element of the N-vector to its correct processor.

Definition 9: Network Connection Equation

We define $C(x)$ to be the $(s(x),d(x))$ connection required to route $v(x)$ from memory $\mu(v(x))$ through network input $I(\mu(v(x)))$ to output $\Phi(x)$ and thence to processor x.

Thus $$C(x) = (I(\mu(v(x))), \Phi(x)) .$$ □

Now what have we got? Recall in the previous chapter (Theorem 4) that if

$$P \subseteq \{(f(x),g(x)) : 0 \leq x < \xi\} ,$$

and if f,g and ξ satisfy certain conditions with respect to Ω, then $\Omega \uparrow P$ and so

the network constructed from Ω can produce the connections specified by P. Thus, in many cases we can easily show that a given Ω network will pass C(x) if C(x) satisfies Theorem 4.

3.4 Effectiveness of Ω Networks

In this section we will be concerned with showing the effectiveness of a particular type of Ω network for various memory and storage configurations. First, we shall assume that N, the number of processors, is a power of two.* Second, we will assume that Ω = {n, n/2, ..., 2, 1}, where n, the "size" of the Ω network, is a power of two. This latter assumption is pragmatic. Recall Theorem 8 of the previous chapter. If n is a power of 2, then Ω = {n, n/2, ..., 2, 1} results in a network which is minimal in terms of gates.

Our choice of a power of two for the number of processors is not so defensible. Suffice to say that investigation of any specific system where N is not a power of two is beyond the scope of this paper. Additionally, we only consider values of $N \leq 4096$, except where noted otherwise.

Tables I-III show C(x) for various combinations of $\mu(i, j)$, $v(x)$, $I(m)$ and $\Phi(x)$. Shown along with C(x) are two numbers. The first of these, memory cycles, is the maximum number of memory cycles required to fetch the specified N-vector. For example, if 2 elements of the N-vector lie in the same memory, then at least two cycles of the memory system will be required.

The second number shown with C(x) is the number of network cycles required to pass the N-vector. For example, if $\Omega \overline{\uparrow} P$ but $P = P_1 \cup P_2$ and $\Omega \uparrow P_1$, $\Omega \uparrow P_2$ then two cycles of the network are required.

---

*In some cases, where N must have an integer square root, we assume N is a power of 4.

In some cases the numbers given for memory cycles and network cycles represent (not necessarily least) upper bounds or lower bounds. This is indicated by the symbols $\leq$ and $\geq$ respectively. Table IV is a summary of the results presented in Tables I-III. The numbers shown in Table IV are the maximum of memory cycles or network cycles. As we can see, of the configurations investigated, only Table III-D can handle all 8 of the most frequent types of N-vectors. Before discussing additional pros and cons of these configurations, we will investigate one additional N-vector which sometimes occurs.

Consider the ℓ-vector consisting of the elements of small d×d submatrices along the diagonal (see Figure 12). We assume that the first row of the first submatrix goes in order to the first d processors, the next row of this first submatrix to the second d processors, and so on [11].

In all there can be at most $N \div d^2$ full d×d submatrices in an ℓ-vector where $\ell \leq N$. Thus, there are a total of $\ell = (N \div d^2) \times d^2$ elements. We have

$$v(x) = ((x \bmod d^2) \div d + d(x \div d^2) + i,\ (x \bmod d^2) \bmod d + d(x \div d^2) + j),$$

and for $\mu(i, j) = (\sqrt{N}+1)i + 2j$ (Table III-D) we get

$$\mu(v(x)) = (\sqrt{N}+1)[(x \bmod d^2 \div d + d(x \div d^2) + i] +$$

$$2[(x \bmod d^2 \bmod d + d(x \div d^2) + j].$$

This function has yet to yield to analysis. Instead, an exhaustive check was performed using a computer and assuming the same configuration as in Table III-D for

$$N = 4^k, \text{ where } k = 2, 3, 4, 5.$$

The results are summarized in Table V. A second check was performed for $\Omega = \{2N, N/2, \ldots, 4, 1\}$ and the results were identical to those of Table V.

N processors

N memories

N × N network

$\Phi(x) = x$

$I(m) = m$

$\mu(i,j) = j$ (straight storage)

$C(x) = (\mu(v(x)), x)$

v(x)		note	$\mu(v(x))$	memory cycles	network cycles
(i, j+x)	rows		j+x	1	1
(i+x, j)	columns	a	j	N	1
(i+x, j+x)	fwd. diag.		j+x	1	1
(i+x, j-x)	rev. diag.		j-x	1	1
(i+(x÷$\sqrt{N}$), j+(x mod $\sqrt{N}$)) $\sqrt{N} \times \sqrt{N}$ partitions		b	j+(x mod $\sqrt{N}$)	$\sqrt{N}$	1
(i,j) N broadcast		c	j	1	1
(i, j+(x÷$\sqrt{N}$) × $\sqrt{N}$) $\sqrt{N}$ row broadcast		d	j+(x÷$\sqrt{N}$)×$\sqrt{N}$	1	1
(i+(x÷$\sqrt{N}$) × $\sqrt{N}$, j) $\sqrt{N}$ column broadcast		e	j	$\sqrt{N}$	1

Table I-A

N processors

N memories

N × N network

$\Phi(x) = x$

$I(m) = m$

$\mu(i,j) = i+j$ (skewed storage)

$C(x) = (\mu(v(x)), x)$

v(x)		note	$\mu(v(x))$	memory cycles	network cycles
(i, j+x)	rows		i+j+x	1	1
(i+x, j)	columns		i+j+x	1	1
(i+x, j+x)	fwd. diag.	f	i+j+2x	2	N/2
(i+x, j-x)	rev. diag.	g	i+j	N	1
(i+(x÷$\sqrt{N}$), j+(x mod $\sqrt{N}$)) $\sqrt{N} \times \sqrt{N}$	partitions	h	i+j+x÷$\sqrt{N}$+x mod $\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$
(i,j) N	broadcast	i	i+j	1	1
(i,j+(x÷$\sqrt{N}$) × $\sqrt{N}$) $\sqrt{N}$	row broadcast	j	i+j+(x÷$\sqrt{N}$)×$\sqrt{N}$	1	1
(i+(x÷$\sqrt{N}$) × $\sqrt{N}$, j) $\sqrt{N}$	column broadcast	k	i+j+(x÷$\sqrt{N}$)×$\sqrt{N}$	1	1

Table I-B

N processors

N memories

N × N network

$\Phi(x) = x$

$I(m) = m$

$\mu(i,j) = 3i+j$ (3 skew)

$C(x) = (\mu(v(x)), x)$

$\eta = \sqrt{N}$

v(x)		note	$\mu(v(x))$	memory cycles	network cycles
(i, j+x)	rows		3i+j+x	1	1
(i+x, j)	columns	ℓ	3(i+x)+j	1	1
(i+x, j+x)	fwd. diag.	m	4x+3i+j	4	N/4
(i+x, j-x)	rev. diag.	n	2x+3i+j	2	N/2
(i+(x÷$\sqrt{N}$), j+(x mod $\sqrt{N}$)) $\sqrt{N} \times \sqrt{N}$ partitions			---	-	-
(i,j) N broadcast			3i+j	1	1
(i, j+(x÷$\sqrt{N}$) × $\sqrt{N}$) $\sqrt{N}$ row broadcast			3i+j+(x÷η)×η	1	1
(i+(x÷$\sqrt{N}$) × $\sqrt{N}$, j) $\sqrt{N}$ column broadcast			---	-	-

Table I-C

N processors

N memories

N × N network

$\Phi(x) = x$

$I(m) = m$

$\mu(i,j) = \eta i+j$ ($\sqrt{N}$ skewing)

$C(x) = (\mu(v(x)), x)$

$\eta = \sqrt{N}$

v(x)		note	$\mu(v(x))$	memory cycles	network cycles
(i, j+x)	rows		$\eta i+j+x$	1	1
(i+x, j)	columns	0	$\eta x+\eta i+j$	$\sqrt{N}$	$\sqrt{N}$
(i+x, j+x)	fwd. diag.	p	$(\eta+1)x+\eta i+j$	1	1
(i+x, j-x)	rev. diag.	q	$(\eta-1)x+\eta i+j$	1	1
(i+(x÷$\sqrt{N}$), j+(x mod $\sqrt{N}$)) $\sqrt{N} \times \sqrt{N}$	partitions	r	$x+\eta i+j$	1	1
(i,j) N	broadcast		$\eta i+j$	1	1
(i,j+(x÷$\sqrt{N}$) × $\sqrt{N}$) $\sqrt{N}$	row broadcast	s	$\eta i+j+(x÷\eta)\times\eta$	1	1
(i+(x÷$\sqrt{N}$) × $\sqrt{N}$, j) $\sqrt{N}$	column broadcast	t	$\eta i+j+N(x÷\eta)$	$\sqrt{N}$	1

Table I-D

N processors

N memories

N × N network

$\Phi(x) = x$

$I(m) = m$

$\mu(i,j) = \eta i + j$ ($\sqrt{N}+1$ skewing)

$C(x) = (\mu(v(x)), x)$

$\eta = \sqrt{N}+1$

v(x)		note	$\mu(v(x))$	memory cycles	network cycles
$(i, j+x)$	rows		$\eta i+j+x$	1	1
$(i+x, j)$	columns	u	$\eta(i+x)+j$	1	1
$(i+x, j+x)$	fwd. diag.	v	$(\eta+1)x+\eta i+j$	≈2	-
$(i+x, j-x)$	rev. diag.	w	$(\eta-1)x+\eta i+j$	$\sqrt{N}$	$\sqrt{N}$
$(i+(x\div\sqrt{N}), j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions		x	$\eta(x\div\sqrt{N})+\eta i+\eta(x \bmod \sqrt{N})+\eta j$	>1	-
(i,j) N broadcast			$\eta i+j$	1	1
$(i, j+(x\div\sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast		y	$\eta i+j+(x\div\sqrt{N})\times\sqrt{N}(x\div\sqrt{N})$	1	1
$(i+(x\div\sqrt{N}) \times \sqrt{N}, j)$ $\sqrt{N}$ column broadcast		z	$\eta i+j+N(x\div\sqrt{N})+\sqrt{N}(x\div\sqrt{N})$	1	1

Table I-E

Notes to Table I

a) Since all elements of a column lie in the same memory, i.e., $v(x) = (i+x, j)$, $\mu(v(x)) = j$, that memory must be cycled N times to produce all elements of the column. One might suspect that the networks would also have to be cycled N times. This is not necessarily so. Notice the resulting connection equation $C(x) = (j, x)$ satisfies the condition $\gcd(a, N) = N$ since $a = 0$. Thus, the network can produce the connection $C(x) = (j, x)$, $0 \leq x \leq N - 1$ by Theorem 4 of the last chapter. So how do we reconcile the fact that our theory says the connection is possible while intuition says it is not.

In fact, there is no real reconciliation. If two different data elements are present at the same input port, then any network would have to be cycled at least twice to pass both elements. We shall simply state that the number shown under network cycles represents the number of cycles required, assuming no multiple data at any input port. Extra cycles required due to such multiple data are reflected in the memory cycles column.

b) The equation $C(x) = (j + (x \bmod \sqrt{N}), x)$ is not of the correct form to be covered by Theorem 4. Thus, we must show that $C(x)$ satisfies definition 3. We assume $N = 2^k$ where k is even. Thus, $\sqrt{N} = 2^{k/2}$. Now we assume $x \bmod \sqrt{N} \underset{N}{\not\equiv} y \bmod \sqrt{N}$ (so $x \underset{N}{\not\equiv} y$),

and $x \overset{m}{\equiv} y$. Then, (P6)

$$x \overset{m}{\not\equiv} y.$$

Now, if $m \leq \sqrt{N}$, then obviously $x \bmod \sqrt{N} \underset{m}{\equiv} x$ and $y \bmod \sqrt{N} \underset{m}{\equiv} y$, so

$$x \bmod \sqrt{N} \overset{m}{\not\equiv} y \bmod \sqrt{N}.$$

If $m > \sqrt{N}$, then we have (since $x \bmod \sqrt{N}$, $y \bmod \sqrt{N} < \sqrt{N} < m$)

$$x \bmod \sqrt{N} \underset{N}{\not\equiv} y \bmod \sqrt{N} \overset{P7}{\rightarrow} x \bmod \sqrt{N} \underset{\sqrt{N}}{\not\equiv} y \bmod \sqrt{N}$$

and finally

$$\overset{P5}{\rightarrow} x \bmod \sqrt{N} \underset{m}{\not\equiv} \bmod \sqrt{N} \ .$$

Thus, $\Omega \uparrow P$, where $P = \{(x \bmod \sqrt{N},\ x) : 0 \leq x < N - 1\}$ so by Theorem 1 (previous chapter) $\Omega \uparrow P + j$, where $P + j = \{(j + (x \bmod \sqrt{N}),\ x) : 0 \leq x < N - 1\}$. □

c) See note (a).

d) Again we must show that $C(x) = (j + (x \div \sqrt{N}) \times \sqrt{N},\ x)$ satisfies definition 3, i.e., for all $m \in \Omega$

$$s(x) \underset{N}{\not\equiv} s(y) \text{ and } s(x) \underset{m}{\equiv} s(y) \text{ implies } x \overset{m}{\not\equiv} y \ .$$

First, consider the case $m \geq \sqrt{N}$. Let $\eta = \sqrt{N}$.

We have $j + (x \div \eta)\eta \underset{N}{\not\equiv} j + (y \div \eta)\eta$

$$\overset{P1}{\rightarrow} (x \div \eta)\eta \underset{N}{\not\equiv} (y \div \eta)\eta$$

and $j + (x \div \eta)\eta \underset{m}{\equiv} j + (y \div \eta)\eta$

$$\overset{P1}{\rightarrow} (x \div \eta)\eta \underset{m}{\equiv} j + (y \div \eta)\eta .$$

Thus,

$$\overset{P6}{\rightarrow} (x \div \eta)\eta \overset{m}{\not\equiv} (y \div \eta)\eta$$

P12
$\rightarrow \quad (x \div \eta) \underset{m/\eta}{\not\equiv} (y \div \eta)$

P11
$\rightarrow \quad x \underset{m}{\not\equiv} y \ .$

Now, for $m < \eta$ let $\eta = m\sigma = \sqrt{N}$. We have

$$\eta(x \div \eta) \underset{N}{\not\equiv} \eta(y \div \eta)$$

P4
$\rightarrow \quad (x \div \eta) \underset{\eta}{\not\equiv} (y \div \eta)$

and

$$\eta(x \div \eta) \underset{m}{\equiv} \eta(y \div \eta)$$

or

$$m\sigma(x \div \eta) \underset{m}{\equiv} m\sigma(y \div \eta)$$

P4
$\rightarrow \quad \sigma(x \div \eta) \underset{1}{\equiv} \sigma(y \div \eta)$

P3
$\rightarrow \quad (x \div \eta) \underset{1}{\equiv} (y \div \eta) \ .$

Thus,

P6
$\rightarrow \quad x \div \eta \underset{1}{\not\equiv} y \div \eta$

P11
$\rightarrow \quad x \underset{m\sigma}{\not\equiv} y$

P9
$\rightarrow \quad x \underset{m}{\not\equiv} y \ .$

Thus, $\Omega \uparrow C(x)$. The vector $v(x)$ can be fetched without conflict since $\mu(v(x))$ satisfies definition 6.

e) See notes (a) and (b).

f) Divide $v(x)$ into two $N/2$-vectors:

$$v_1(x) = (i+x,\ j+x),\ 0 \leq < N/2$$

$$v_2(x) = (i+x+N/2,\ j+x+N/2),\ 0 \leq x < N/2 .$$

Then $v(x) = v_1(x) \cup v_2(x)$, and

$\mu(v_1(x))$ and $\mu(v_2(x))$ both satisfy Theorem 9. See also note (n).

g) See note (a).

h) It is easy to see that there are $\sqrt{N}$ elements (the reverse diagonal) of the partition in memory $\mu(v\sqrt{N}-1)) = i+j+\sqrt{N}-1$. Thus, the $\sqrt{N}$ memory cycles. In order to show that the network cycles are $\leq \sqrt{N}$, we divide $v(x)$ into $\sqrt{N}$ $\sqrt{N}$-vectors:

$$v_p(x) = (i+p,\ j+x \bmod \sqrt{N}), 0 \leq x < \sqrt{N},\ 0 \leq p < \sqrt{N} .$$

Then $$v(x) = \bigcup_{p=0}^{\sqrt{N}-1} v_p(x)$$

and by the argument in note (b),

$$\Omega \uparrow C_p(x), \text{ where } C_p(x) = (\mu(v_p(x)),\ x + p\sqrt{N}),\ 0 \leq x < \sqrt{N},\ 0 \leq p < \sqrt{N}.$$

i) See note (a).

j) See note (d).

k) See note (d).

ℓ) Since 3 is prime to any power of 2, $C(x)$ satisfies Theorem 4 and $\mu(v(x))$ satisfies Theorem 9.

m) The required partition for the memory is:

$$v_p(x) = (i+x,\ j+x),\ 0 \leq x < N/4,\ 0 \leq p < 4$$

This allows the N-vector to be fetched in 4 cycles.

The partition required for the network is

$$v_p(x) = (i+p+Nx/4,\ j+p+Nx/4),\ 0 \leq x < 4,\ 0 \leq p < N/4$$

and thus (since $Nx \underset{N}{\equiv} 0$),

$$C_p(x) = (4p+3i+j,\ p+Nx/4), \qquad 0 \leq x < 4,\ 0 \leq p < N/4.$$

n) An argument similar to m applies here.

o) An argument similar to m applies here.

p) If $\sqrt{N} + 1$ is prime to N, then $C(x)$ satisfies Theorem 4 and $\mu(v(x))$ satisfies Theorem 9. $\sqrt{N} + 1$ is prime to 4, 16, 64, 256, 1024, and 4096.

q) This is true for N = 4, 16, 64, 256, 1024, and 4096.

r) Note that $\sqrt{N}(x \div \sqrt{N}) + x \bmod \sqrt{N} = x$.

s) See note (d).

t) The required partition is

$$v_p(x) = (p+i,\ j),\ 0 \leq x < \eta$$
$$0 \leq p < \eta$$
$$\eta = \sqrt{N},$$

giving us $\mu(v_p(x)) = \eta(i+p) + j$, which satisfies definition 6 since

$v_p(x) = v_p(y)$.

To prove $\Omega \uparrow C(x)$, we have

$$\eta i+j+N(x \div \eta) \underset{N}{\equiv} \eta i+j+N(y \div \eta),$$

i.e., the source is the same memory element for all data. See note (a).

u) $\sqrt{N} + 1$ is prime to N = 4, 16, 64, 256, 1024, and 4096. Thus, this result holds for these N by Theorems 4 and 9.

v) The cycles required here vary since $\eta+1 = \sqrt{N}+2$ is not prime to N = 4, 16, 64, 256, or 1024. The memory cycles vary from 4 for N = 4 to 2 for N = 1024, where in general memory cycles is (Theorem 9)

$$\frac{a}{\alpha} \quad \text{where } \alpha = \text{gpf}(a,N).$$

The required number of network cycles is unknown.

w) Note $\eta-1 = \sqrt{N}$. An argument similar to note m applies here.

x) $\eta(x \div \sqrt{N} + x \bmod \sqrt{N}) + \eta(i+j)$

$= \sqrt{N}(x \div \sqrt{N}) + (x \bmod \sqrt{N}) + (x \div \sqrt{N}) + \sqrt{N}(x \bmod \sqrt{N}) + \eta(i+j)$

$= x + (x \div \sqrt{N}) + \sqrt{N}(x \bmod \sqrt{N}) + \eta(i+j)$.

Since $\mu(v(\sqrt{N}-1)) \underset{N}{\equiv} \mu(v(N-\sqrt{N})) \underset{N}{\equiv} \eta(i+j)$, we know that memory cycles are > 1.

y) See note (d).

z) Since $N(x \div \sqrt{N}) \underset{N}{\equiv} 0$,

$$\mu(v(x)) \underset{N}{\equiv} \eta i+j+(x \div \sqrt{N}) \times \sqrt{N} .$$

See note (d).

N processors

2N memories

2N×2N network

$\Phi(x) = x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = j$ (straight storage)

$C(x) = (\mu(v(x)), x), \quad 0 \leq x < N$

$v(x),\ 0 \leq x < N$	note	$\mu(v(x))$	memory cycles	network cycles
$(i,\ j+x)$ rows		$j+x$	1	1
$(i+x,\ j)$ columns	a	j	N	1
$(i+x,\ j+x)$ fwd. diag.		$j+x$	1	1
$(i+x,\ j-x)$ rev. diag.		$j-x$	1	1
$(i+(x\div\sqrt{N}),\ j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions	b	$j+(x \bmod \sqrt{N})$	$\sqrt{N}$	1
(i,j) N broadcast		j	1	1
$(i,j+(x\div\sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast		$j+(x\div\sqrt{N})\times\sqrt{N}$	1	1
$(i+(x\div\sqrt{N}) \times \sqrt{N},\ j)$ $\sqrt{N}$ column broadcast	c	j	$\sqrt{N}$	1

Table II-A

N processors

2N memories

2N×2N network

$\Phi(x) = x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = 2i+j$ (2-skew, 1-skip)

$C(x) = (\mu(v(x)), x), \quad 0 \leq x < N$

$v(x)$, $0 \leq x < N$	note	$\mu(v(x))$	memory cycles	network cycles
$(i, j+x)$ rows		$2i+j+x$	1	1
$(i+x, j)$ columns	d	$2x+2i+j$	1	$\frac{N}{2}$
$(i+x, j+x)$ fwd. diag.		$3x+2i+j$	1	1
$(i+x, j-x)$ rev. diag.		$x+2i+j$	1	1
$(i+(x\div\sqrt{N}), j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions	e	$2(x\div\sqrt{N})+2i+j+x \bmod \sqrt{N}$	$\sqrt{N}$	-
(i,j) N broadcast		$2i+j$	1	1
$(i, j+(x\div\sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast	f	$2i+j+(x\div\sqrt{N})\times \sqrt{N}$	1	1
$(i+(x\div\sqrt{N}) \times \sqrt{N}, j)$ $\sqrt{N}$ column broadcast	g	$2i+j+2\sqrt{N}(x\div\sqrt{N})$	1	1

Table II-B

Notes to Table II

a) See note (a), Table I.

b) See note (b), Table I.

c) See notes (a) and (b), Table I.

d) Theorem 9 is satisfied since

$$x < \frac{M\alpha}{a} = \frac{2N \cdot 1}{2} = N .$$

See note (n), Table I.

e) Partition $v(x)$, $0 \leq x < N$

into

$$v_p(x) = (i+p\sqrt{N},\ j), \quad 0 \leq x < \sqrt{N}, \quad 0 \leq p < \sqrt{N} .$$

Then since $\mu(v_p(x))$ satisfies Theorem 9 and

$$v(x) = \bigcup_{p=0}^{\sqrt{N}-1} v_p(x) .$$

f) See note (d), Table I.

g) We have (after applying P6 and $2\sqrt{N}(x \div \sqrt{N}) = 2\sqrt{N}\, x \div 2N$)

$$2\sqrt{N}\, x \div 2N \overset{m}{\not\equiv} 2\sqrt{N}\, y \div 2N$$

P11
$$\rightarrow \quad 2\sqrt{N}\, x \overset{2mN}{\not\equiv} 2\sqrt{N}\, y$$

P12,9
$$\rightarrow \quad x \overset{m}{\not\equiv} y .$$

N processors

2N memories

2N×2N network

$\Phi(x) = 2x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = j$ (straight storage)

$C(x) = (\mu(v(x)), 2x), \quad 0 \leq x < N$

$v(x)$, $0 \leq x < N$		note	$\mu(v(x))$	memory cycles	network cycles
$(i, j+x)$	rows	a	$j+x$	1	1
$(i+x, j)$	columns	b	j	N	1
$(i+x, j+x)$	fwd. diag.		$j+x$	1	1
$(i+x, j-x)$	rev. diag.		$j-x$	1	1
$(i+(x \div \sqrt{N}), j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$	partitions	c	$j+x \bmod \sqrt{N}$	$\sqrt{N}$	1
(i,j) N	broadcast		j	1	1
$(i, j+(x \div \sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$	row broadcast	d	$j+(x \sqrt{N}) \sqrt{N}$	1	1
$(i+(x \div \sqrt{N}) \times \sqrt{N}, j)$ $\sqrt{N}$	column broadcast	e	j	$\sqrt{N}$	1

Table III-A

N processors

2N memories

2N×2N network

$\Phi(x) = 2x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = 2i+j$ (2-skew, 1-skip)

$C(x) = (\mu(v(x)), 2x), \quad 0 \leq x < N$

$v(x)$, $0 \leq x < N$	note	$\mu(v(x))$	memory cycles	network cycles
$(i, j+x)$ rows		$2i+j+x$	1	1
$(i+x, j)$ columns	f	$2i+2x+j$	1	1
$(i+x, j+x)$ fwd. diag.		$3x+2i+j$	1	1
$(i+x, j-x)$ rev. diag.		$x+2i+j$	1	1
$(i+(x\div\sqrt{N}), j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions	g	$2i+j+2(x\div\sqrt{N})+ x \bmod \sqrt{N}$	$\sqrt{N}$	-
(i,j) N broadcast		j	1	1
$(i, j+(x\div\sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast	h	$2i+j+(x\div\sqrt{N})\ \sqrt{N}$	1	1
$(i+(x\div\sqrt{N}) \times \sqrt{N}, j)$ $\sqrt{N}$ column broadcast	i	$2i+j+2(x\div\sqrt{N})\ \sqrt{N}$	1	1

Table III-B

N processors

2N memories

2N×2N network

$\Phi(x) = 2x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = i+2j$ (1-skew, 2-skip)

$C(x) = (\mu(v(x)), 2x), \quad 0 \leq x < N$

$v(x)$, $0 \leq x < N$	note	$\mu(v(x))$	memory cycles	network cycles
$(i, j+x)$ rows	j	$i+2j+2x$	1	1
$(i+x, j)$ columns		$x+i+2j$	1	1
$(i+x, j+x)$ fwd. diag.		$3x+2j+i$	1	1
$(i+x, j-x)$ rev. diag.		$x+2j+i$	1	1
$(i+(x \div \sqrt{N}), j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions	k	$x \div \sqrt{N}+i+2j+2(x \bmod \sqrt{N})$	$\frac{\sqrt{N}}{2}$	$\frac{\sqrt{N}}{2}$
(i,j) N broadcast		$i+2j$	1	1
$(i,j+(x \div \sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast	ℓ	$i+2j+2(x \div \sqrt{N})\ \sqrt{N}$	1	1
$(i+(x \div \sqrt{N}) \times \sqrt{N}, j)$ $\sqrt{N}$ column broadcast	m	$2i+j+(x \div \sqrt{N})\ \sqrt{N}$	1	1

Table III-C

N processors

2N memories

2N×2N network

$\nu(x) = 2x, \quad 0 \leq x < N$

$I(m) = m, \quad 0 \leq m < 2N$

$\mu(i,j) = \eta i + 2j \qquad (\sqrt{N}+1 \text{ skew, } 2 \text{ skip})$

$C(x) = (\mu(\nu(x)), 2x), \quad 0 \leq x < N$

$\eta = \sqrt{N} + 1$

$\nu(x),\ 0 \leq x < N$	note	$\mu(\nu(x))$	memory cycles	network cycles
$(i,\ j+x)$ rows	n	$\eta i+2j+2x$	1	1
$(i+x,\ j)$ columns	o	$\eta i+2j+\eta x$	1	1
$(i+x,\ j+x)$ fwd. diag.	p	$(\eta+2)x+\eta i+2j$	1	1
$(i+x,\ j-x)$ rev. diag.	q	$(\eta-2)x+\eta i+2j$	1	1
$(i+(x\div\sqrt{N}),\ j+(x \bmod \sqrt{N}))$ $\sqrt{N} \times \sqrt{N}$ partitions	r	$\eta(x\div\sqrt{N})+\eta i+2j+$ $2(x \bmod \sqrt{N})$	1	1
(i,j) N broadcast		$\eta i+2j$	1	1
$(i, j+(x\div\sqrt{N}) \times \sqrt{N})$ $\sqrt{N}$ row broadcast	s	$\eta i+2j+2\sqrt{N}(x\div\sqrt{N})$	1	1
$(i+(x\div\sqrt{N}) \times \sqrt{N},\ j)$ $\sqrt{N}$ column broadcast	t	$\eta i+2j+\eta\sqrt{N}(x\div\sqrt{N})$	1	1

Table III-D

Notes, Table III

a) Note that $C(x)$ is now $(\mu(v(x)), 2x)$.

b) See note (a), Table I.

c) See note (b), Table II.

d) See note (d), Table I.

e) See notes (a) and (b), Table I.

f) Note that $2x + 2i + j$ satisfies Theorem 9, since $x < N$. Also, $C(x) = (2x, 2x)$ satisfies Theorem 4.

g) See note (e), Table II.

h) See note (d), Table I.

i) See note (d), Table I.

j) See note (f), Table III.

k) Note that

$$2(x \bmod \sqrt{N}) = 2x \bmod 2\sqrt{N}.$$

Partition $v(x)$, $0 \leq x < N$ into

$$v_p(x) = (i+(x \div \sqrt{N})+2p,\ j+(x \bmod \sqrt{N})),$$

$$0 \leq x < 2\sqrt{N},$$

$$0 \leq p < \sqrt{N}/2.$$

Then $\mu(v_p(x))$ satisfies Theorem 9 since $x < 2\sqrt{N} < 2N$, and

$$v(x) = \bigcup_{p=0}^{\sqrt{N}/2} v_p(x).$$

and

$$v(x) = \bigcup_{p=0}^{\sqrt{N}/2} v_p(x) \ .$$

These same partitions satisfy Theorem 4.

ℓ) The result follows from note (d), Table I, and P2.

m) See note (d), Table I.

n) Theorem 9 is satisfied since $x < 2N/2$. Theorem 4 is also satisfied (since $\Phi(x) = 2x$).

o) $\sqrt{N} + 1$ is prime to $2N$ for $N = 4, 16, 64, 256, 1024$, or 4096. Thus, Theorems 4 and 9 are satisfied again.

p) $\eta + 2 = \sqrt{N} + 3$. $\sqrt{N} + 3$ is prime to $2N$ for $N = 4, 16, 64, 256, 1024$, or 4096.

q) $\eta - 2 = \sqrt{N} - 1$. $\sqrt{N} - 1$ is prime to $2N$ for $N = 4, 16, 64, 256, 1024$, or 4096.

r) An exhaustive check of the function

$$C(x) = ((\sqrt{N}+1)(x \div \sqrt{N}) + 2(x \bmod \sqrt{N}),\ 2x),\ 0 \leq x \leq N - 1$$

was made by computer for $N = 4, 16, 64, 256, 1024$, and 4096. In all cases, definitions 3 and 6 were satisfied. The required connections then follow by Theorem 1.

s) We have

$$\eta i + 2j + 2\sqrt{N}(x \div \sqrt{N}) \underset{m}{\equiv} \eta i + 2j + 2\sqrt{N}(y \div \sqrt{N})$$

$$\xrightarrow{P1} \quad 2\sqrt{N}(x \div \sqrt{N}) \underset{m}{\equiv} 2\sqrt{N}(y \div \sqrt{N})$$

But

$$\eta i + 2j + 2\sqrt{N}(x \div \sqrt{N}) \underset{2N}{\not\equiv} \eta i + 2j + 2\sqrt{N}(y \div \sqrt{N})$$

$$\xrightarrow{P1} \quad 2\sqrt{N}(x \div \sqrt{N}) \underset{2N}{\not\equiv} 2\sqrt{N}(y \div \sqrt{N})$$

These two results give us

$$\text{P6} \quad \rightarrow \quad 2\sqrt{N}(x \div \sqrt{N}) \overset{m}{\not\equiv} 2\sqrt{N}(y \div \sqrt{N}).$$

Since $2\sqrt{N}(x \div \sqrt{N}) = (2y\sqrt{N}) \div (2N)$,

we get

$$\text{P11} \quad \rightarrow \quad 2\sqrt{N}\, x \overset{2mN}{\not\equiv} 2\sqrt{N}\, y$$

$$\text{P12} \quad \rightarrow \quad 2x \overset{2m\sqrt{N}}{\not\equiv} 2y$$

$$\text{P9} \quad \rightarrow \quad 2x \overset{m}{\not\equiv} 2y \ .$$

Thus, definition 3 is satisfied. Definition 6 is satisfied since it is satisfied for rows (see note (d), Table I).

t) We have

$$\eta i + 2j + \eta\sqrt{N}(x \div \sqrt{N}) \overset{m}{\equiv} \eta i + 2j + \eta\sqrt{N}(y \div \sqrt{N})$$

$$\text{P1} \quad \rightarrow \quad \eta\sqrt{N}(x \div \sqrt{N}) \overset{m}{\equiv} \eta\sqrt{N}(y \div \sqrt{N}), \quad \text{where} \quad \eta = \sqrt{N} + 1$$

Again, η is prime to $2N$ for $2N = 2^k$, so

$$\text{P3} \quad \rightarrow \quad \sqrt{N}(x \div \sqrt{N}) \overset{m}{\equiv} \sqrt{N}(y \div \sqrt{N}) \ .$$

But $$\eta i + 2j + \eta\sqrt{N}(x \div \sqrt{N}) \overset{2N}{\not\equiv} \eta i + 2j + \eta\sqrt{N}(y \div \sqrt{N})$$

$$\text{P1,3} \quad \rightarrow \quad \sqrt{N}(x \div \sqrt{N}) \overset{2N}{\not\equiv} \sqrt{N}(y \div \sqrt{N})$$

So P6 $\rightarrow \quad \sqrt{N}(x \div \sqrt{N}) \overset{m}{\not\equiv} \sqrt{N}(y \div \sqrt{N})$.

Since $\sqrt{N}(x \div \sqrt{N}) = (x\sqrt{N}) \div N$, we get

$$(x\sqrt{N}) \div N \overset{m}{\not\equiv} (y\sqrt{N}) \div N$$

P11 $\rightarrow \quad \sqrt{N}\, x \overset{Nm}{\not\equiv} \sqrt{N}\, y$

P9 $\rightarrow \quad \sqrt{N}\, x \overset{Nm/2}{\not\equiv} \sqrt{N}\, y$

P12 $\rightarrow \quad 2\sqrt{N}\, x \overset{2Nm/2}{\not\equiv} 2\sqrt{N}\, y$

$\rightarrow \quad 2\sqrt{N}\, x \overset{Nm}{\not\equiv} 2\sqrt{N}\, y$

P12 $\rightarrow \quad 2x \overset{m\sqrt{N}}{\not\equiv} 2y$

P9 $\rightarrow \quad 2x \overset{m}{\not\equiv} 2y$.

Thus, definition 3 is satisfied since

$$s(x) \underset{2N}{\not\equiv} s(y) \wedge s(x) \underset{m}{\equiv} s(y) \rightarrow d(x) \overset{m}{\not\equiv} d(y) \ .$$

Definition 6 is satisfied by an argument similar to note (s), Table III.

That is, since there are no memory conflicts for columns, and since the $\sqrt{N}$ column broadcast involves fetching only $\sqrt{N}$ elements out of N in a column, then there cannot be any memory conflicts in a $\sqrt{N}$ column broadcast. That there are no memory conflicts for a column fetch follows from the fact that $\sqrt{N} + 1$ is prime to $2N = 2^k$. Thus, Theorem 9 is satisfied since the greatest factor of $\sqrt{N} + 1$ prime to 2N is $\sqrt{N} + 1$,

$$gpf(\sqrt{N}+1, N) = \sqrt{N} + 1$$

and so it is only required that x be less than $M = 2N$,

$$x < \ell \leq M = 2N .$$

Thus $x \neq y$ and $\mu(v(x)) \underset{M}{\equiv} \mu(v(y))$

or $(\sqrt{N}+1)(x+i) + 2j \underset{M}{\equiv} (\sqrt{N}+1)(y+i) + 2j$

P1
$\rightarrow$ $(\sqrt{N}+1)(x+i) \underset{M}{\equiv} (\sqrt{N}+1)(y+i)$

P3
$\rightarrow$ $x+i \underset{M}{\equiv} y+i$

$\rightarrow$ $(x+i, j) = (y+i, j)$ since $x, y < M$

$\rightarrow$ $v(x) = v(y)$.

Table Number	I-A	I-B	I-C	I-D	I-E	II-A	II-B	III-A	III-B	III-C	III-D
rows	1	1	1	1	1	1	1	1	1	1	1
columns	N	1	1	$\sqrt{N}$	1	N	≥ 2	N	1	1	1
fwd. diag.	1	N/2	N/4	1	$\tilde{2}$	1	1	1	1	1	1
rev. diag.	1	N	N/2	1	$\sqrt{N}$	1	1	1	1	1	1
$\sqrt{N} \times \sqrt{N}$ partition	$\sqrt{N}$	$\sqrt{N}$	-	1	≥ 2	$\sqrt{N}$	$\geq \sqrt{N}$	$\sqrt{N}$	$\geq \sqrt{N}$	$\frac{\sqrt{N}}{2}$	1
broadcast	1	1	1	1	1	1	1	1	1	1	1
$\sqrt{N}$ row broadcast	1	1	1	1	1	1	1	1	1	1	1
$\sqrt{N}$ column broadcast	$\sqrt{N}$	1	-	$\sqrt{N}$	1	$\sqrt{N}$	1	$\sqrt{N}$	1	1	1
memories	N	N	N	N	N	2N	2N	2N	2N	2N	2N
skew	0	1	3	$\sqrt{N}$	$\sqrt{N}+1$	0	2	0	2	1	$\sqrt{N}+1$
skip	1	1	1	1	1	1	1	1	1	2	2
$\Phi(x)$	x	x	x	x	x	x	x	2x	2x	2x	2x

Table IV. Summary

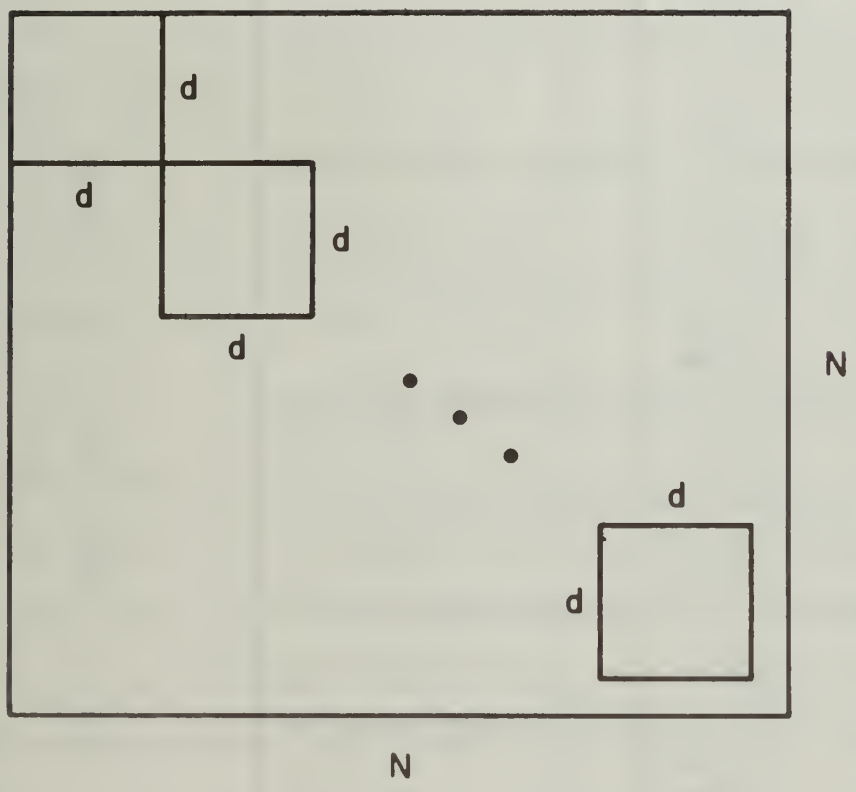


Figure 12. Storage of d × d Submatrices Along the Diagonal

d	2		3		4		5		6		7	
N	memory cycles	network cycles	memory cycles	network cycles	memory cycles	network cycles	memory cycles	network cycles	memory cycles	network cycles	memory cycles	network cycles
16	1	2	1	1	1	1	-	-	-	-	-	-
64	1	2	2	2	2	2	1	2	1	1	1	1
256	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	1	≥ 2	≥ 2	≥ 2	1	≥ 2
1024	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	≥ 2	1	≥ 2	1	≥ 2	≥ 2	≥ 2

Table V. Small Blocks on the Diagonal

3.5 Additional Considerations, Cost

Tables I-V give us a pretty good idea of the performance of the Ω network on certain highly frequent N-vectors. We will not discuss these results any further in this chapter. (In a later chapter we will compare these results with similar results from other types of networks). We will, however, discuss several questions raised by these results.

First, it seems apparent that using 2N memories yields fewer memory conflicts than a system with N memories. In fact, purely combinatorial arguments tell us that of a total of $\binom{N^2}{N} = \frac{(N^2)!}{(N^2-N)!N!}$ possible distinct N-vectors in an $N \times N$ array, the N memory systen can deliver $N!N^N$ without conflict, while the 2N memory system can deliver* $\frac{(2N)!N^N}{N!2^N}$ without conflict.

But we must ask: What is the difference in cost between an N memory system and a 2N memory system? Let us assume that the total storage capacity of both systems is equal, e.g., each memory in the N memory system can store K words while each memory in the 2N memory system can store K/2 words.

There is some evidence which suggests that at least the cost of the basic memory components would be the same. For example, Intel supplies a solid state memory board which can be configured either as $4K \times 18$ bit words or $8K \times 9$ bit words. However, the total cost of a memory system is primarily determined by the cost of power supplies, packaging, interconnections, etc., and thus further analysis of this

---

*The assumption is made that all N^2 elements of the $N \times N$ array are evenly distributed among all N or 2N memories.

problem is beyond the scope of this paper. We will simply assert that it is not unlikely that the cost of a 2N memory system would not be prohibitively greater than the cost of an N memory system when the total cost of the computer system is considered.

Another question which arises is the relative cost of a $2N \times 2N$ Ω network versus an $N \times N$ Ω network. As we shall see in a later chapter, the number of gates in an $N \times N$ Ω network (including control circuitry) is on the order of

$$\frac{5N}{2}(d + \log_2(\log_2 N)) \log_2 N ,$$

where d is the number of bits per data word. The gate ratio of a $2N \times 2N$ Ω network to an $N \times N$ network is approximately 2. (This should be compared with a ratio of 4 for a crossbar switch).

Finally, it might be argued that each memory in a 2N memory system only needs to be half as fast as each memory in an N memory system in order to provide the same effective memory bandwidth. This argument is valid only if successive N-vectors do not interfere with each other. While an analysis of successive N-vector interference is beyond the scope of this work, it seems obvious that this interference could be significant and so the argument justifying slower memories would not hold.

3.6 Summary

In this chapter we have explored several memory systems in conjunction with various memory equations. We have shown that in almost all cases the Ω network performs at least as well as the memories themselves. That is, in almost all cases, if the N-vector can be accessed without memory conflict, then the Ω network can establish the necessary memory-processor connection.

In the next chapter we will discuss actual implementation of Ω networks. This will be followed in later chapters by comparisons with various other switching networks which have been proposed in the literature.

4. CONSTRUCTION OF SEVERAL Ω NETWORKS

In this chapter we will present several implementations of Ω networks. Of course, many implementations are possible but we will present only a few which represent various extremes of design. The first design is probably the simplest design possible. It is somewhat slow, due to the fact that it operates in bit serial mode. It might be useful in applications requiring the switching of bit serial data, such as switching data between tracks of a rotating memory.

The second design is a more general network which might be used for the processor-memory connections discussed earlier. Finally, we will discuss a particular interconnection of processors which is surprisingly related to Ω networks.

4.1 A Bit Serial Ω Network

This Ω network, shown in Figure 13, will be constructed from elements shown in Figure 14. (Notice that the two center NAND gates form a bistable device). The network operates as follows. The network is first reset by using the reset lines which are common to all elements. Associated with each input is a $\log_2 n$ bit number which represents the number of the output port to which that input port is to be connected (the destination tag). This number is inserted bit serially into each input port, most significant bit first. Each element then transmits these bits

$$A = C,\ B = D.$$

As the i-th most significant bit is fed into the network, the strobe signal is turned on momentarily in the i-th stage (from the left). This strobe signal allows the bistable pair of NAND gates to be switched, and if A = 1, then the element will begin transmitting A = D, B = C; otherwise, it will continue to

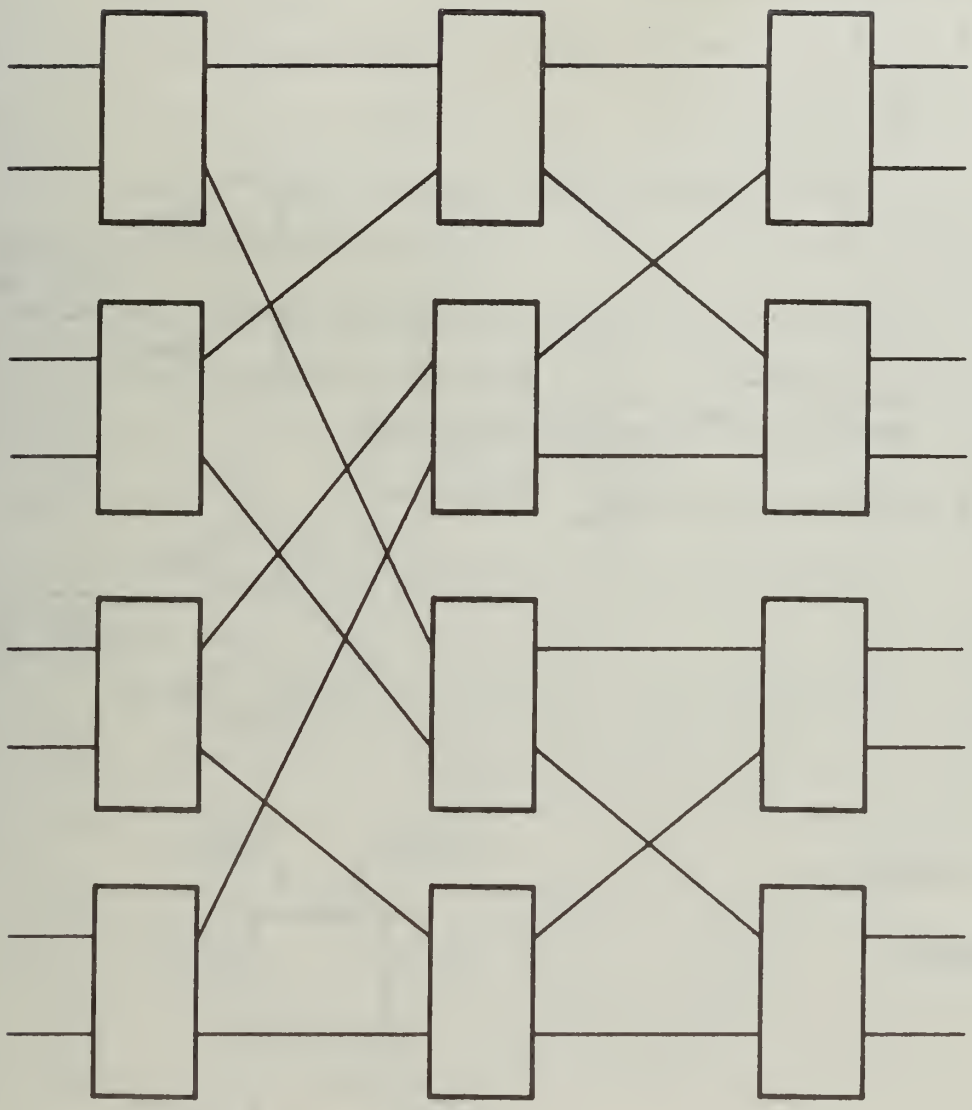


Figure 13. An Ω Network

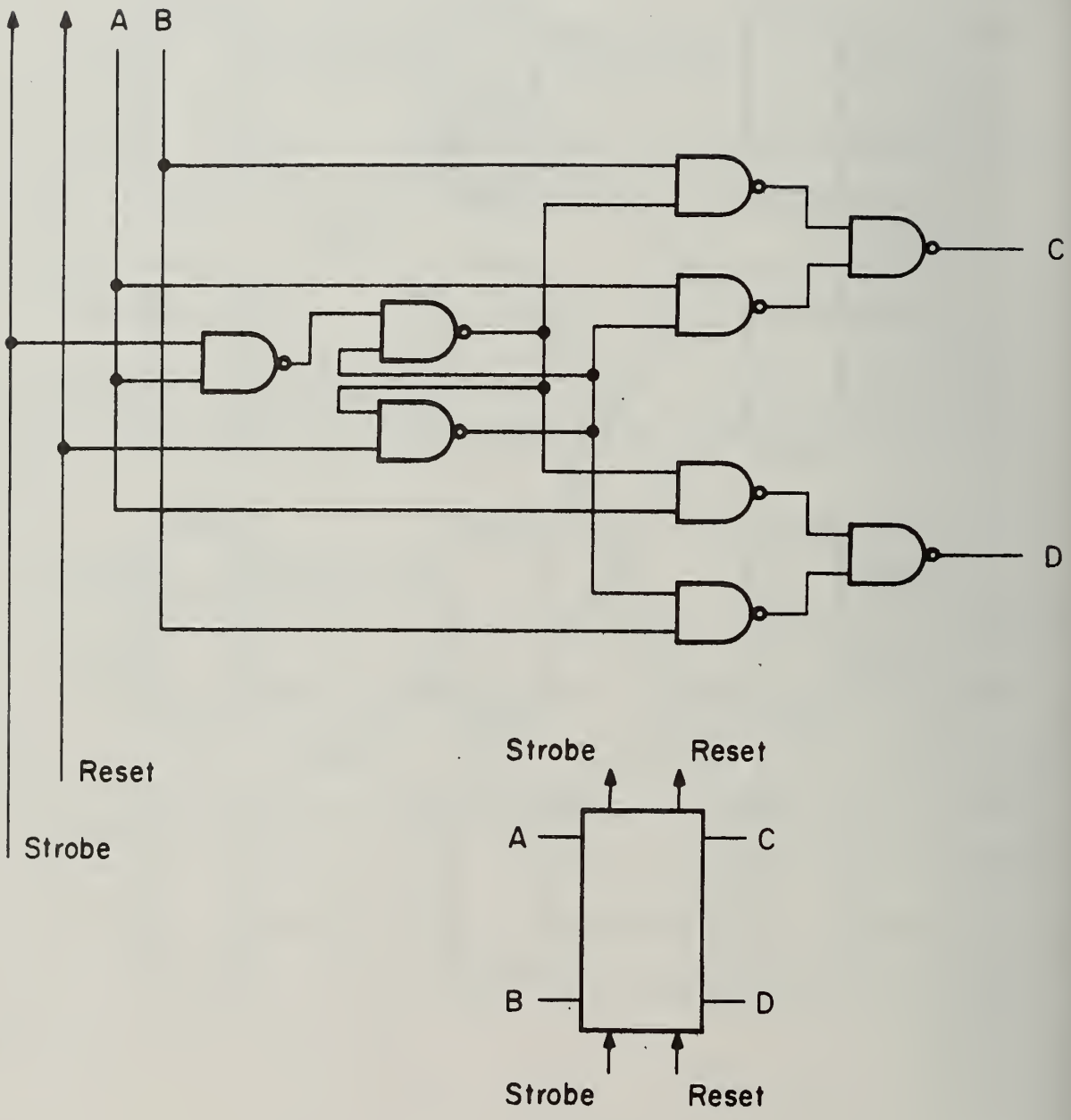


Figure 14. One Element of a Bit Serial Ω Network Constructed from NAND Gates

transmit A = C, B = D.

Now assume there are no conflicts as defined in Chapter 2. Then it can be shown (refer to Figure 10 in Chapter 2) that both streams entering an element of the i-th stage (from the left) must be equal in the first i-1 most significant bits and must be unequal in the i-th most significant bit position. Thus, the i-th stage is strobed when the i-th most significant bits are present at the inputs. (If there are no conflicts, then these bits are unequal). If A = 0 at this time, then the element remains in state A = C, B = D. Otherwise, A = 1 and the element enters state A = D, B = C. The strobe is now turned off which effectively locks the element in this state until the **reset** signals are used. Thus, the network is being switched according to Algorithm 2 of Chapter 2 as long as there are no conflicts.

Now there are two problems. First, if a conflict does arise, the network will produce an erroneous connection. (The destination tags could be examined as they emerge from the outputs to determine if the correct connection was established). Second, it is impossible to set up any one-to-many connections. (Recall these are allowed in the generalized Ω network presented in Chapter 2). Nevertheless, this network may prove useful in some applications.

Notice in Figure 14 that each element requires 9 NAND gates. Thus, the total number of gates required for the network is $9 \times \frac{n}{2} \log_2 n$ or $\frac{9}{2} n \log_2 n$. Further examination of Figure 14 reveals that 3 gate delays are needed for switching and 2 for transmission through each element. This, together with the bit serial nature of transmission, reveals that

$$\sum_{i=1}^{\log_2 n} 3 + 2i = 2\log_2 n + (\log_2 n)^2$$

gate delays are required to properly switch the network.

This could be reduced to $5\log_2 n$ by the addition of appropriate latching registers and clock signals.

We will now turn to a more complicated Ω network.

4.2 A Better Network

In the previous network when a conflict arises, one of the inputs is switched in the "wrong" direction. This wrongly switched input can, in a later stage, cause other inputs to be wrongly switched. The network which we will now discuss prevents this by associating a validity signal with each input. As soon as an input is switched in the wrong direction, its validity signal is turned off and thus the wrongly switched input is prevented from influencing later switching decisions.

Additionally, this network will be capable of producing one to many connections (broadcasts) as discussed in Chapter 2. This is accomplished by using source tags rather than destination tags. A source tag is a number associated with each output port which represents the input to which that output port is connected.

This network is divided into one "control plane" and one or more "data planes" as shown in Figure 15. Signals generated by the control plane are used to control the switching of the data planes. Each plane is similar (at least topologically) to Figure 13. Notice that the control signals will flow from right to left while the data will be transmitted from left to right.

We begin by designing the "control plane" for this Ω network. The control plane generates signals which are fed to the switches in each data plane of the Ω network. Refer to Figure 16. On the right edge are n shift registers of $\log_2 n$ bits each. Each of these shift registers contain the number of the input port to which this output is to be connected. The least

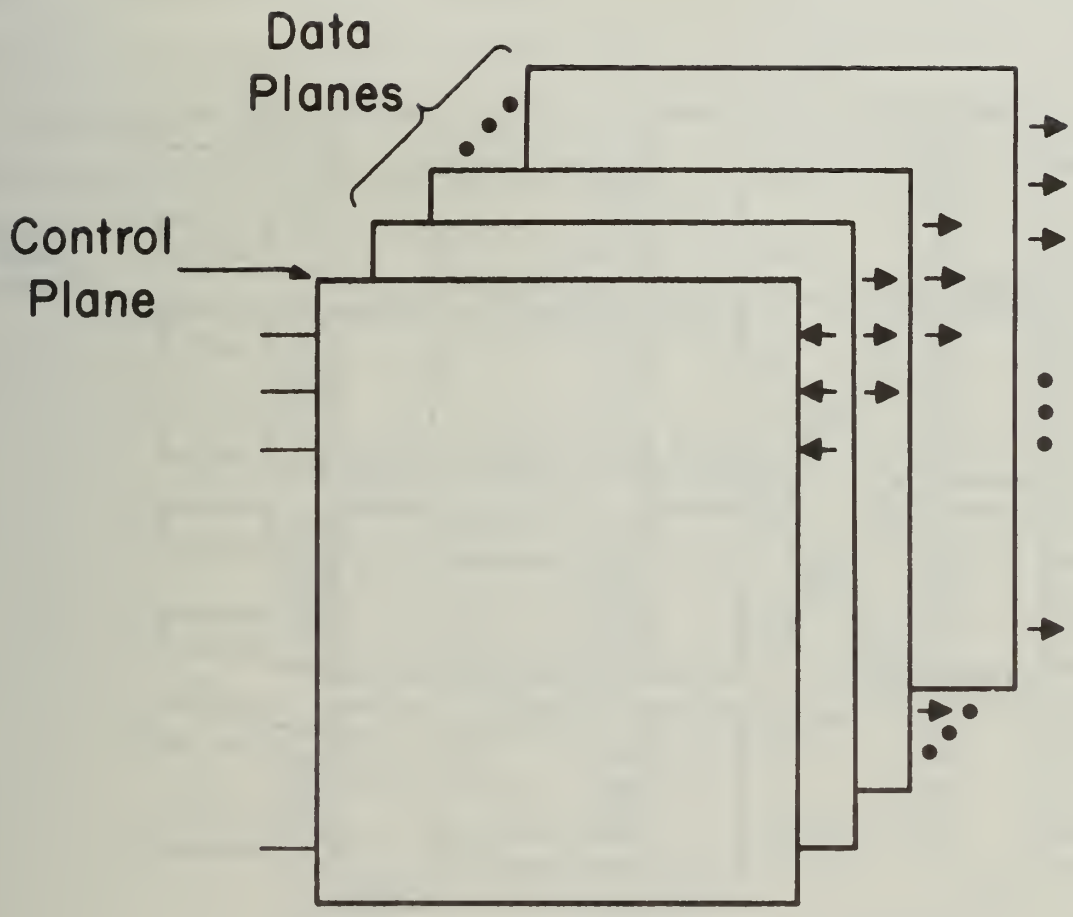


Figure 15. The Control and Data Planes of an Ω Network

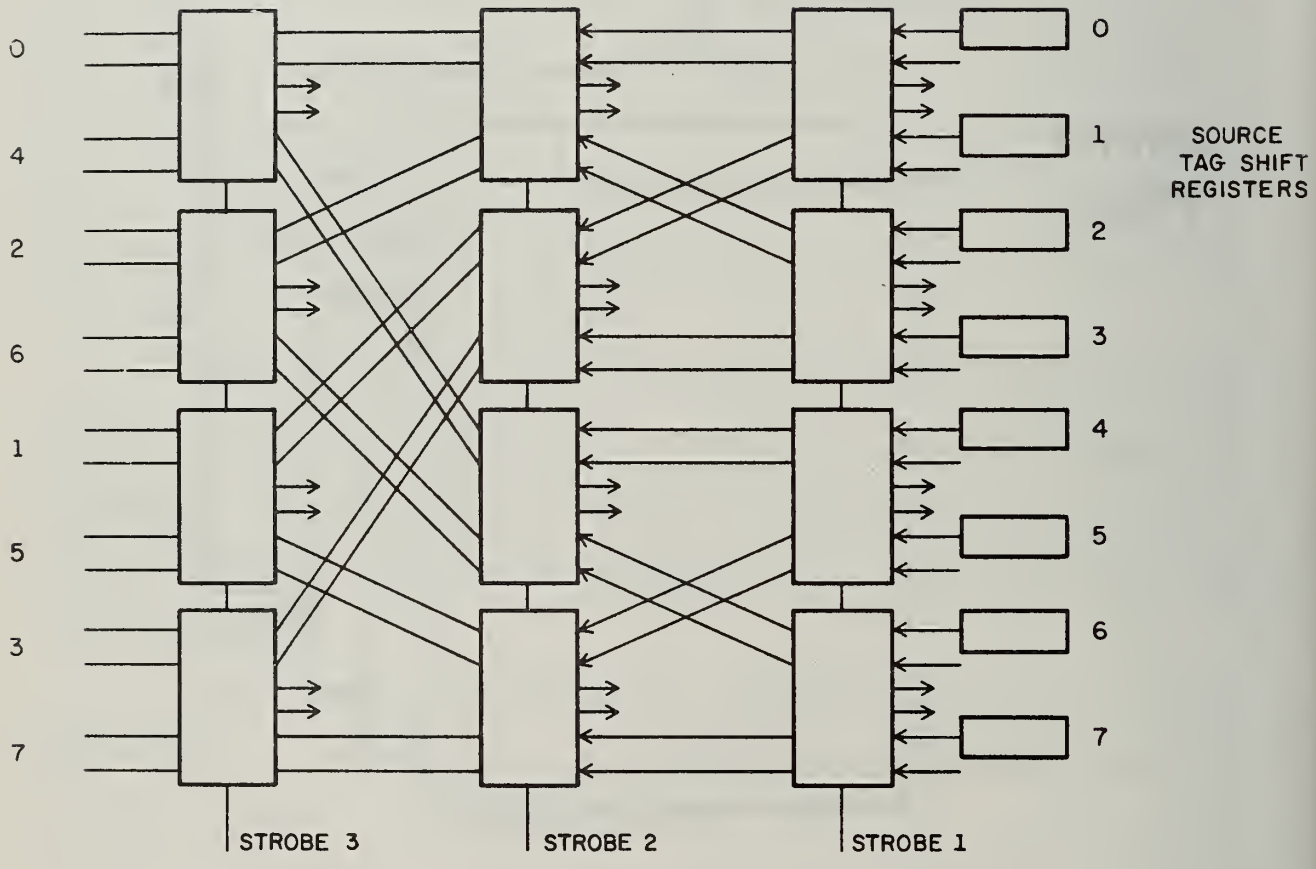


Figure 16. Control Plane for an 8 × 8 Ω Network

significant bit of each of these registers is connected to the inputs of the control plane as shown in Figure 16. Generally, each stage of the control plane will be "set up" during one major clock. The entire plane is thus set in $\log_2 n$ major clocks. During the i-th major clock, the i-th stage (from the right) is strobed allowing the signal from the current least significant bit to set the switching and memory of each block in the stage. Then the source tag registers are shifted allowing the next least significant bit to be switched through the i-th stage to the (i+1)-th stage where this process is repeated.

The following signals are used in each element of the control plane (refer to Figure 17):

Strobe:	enables setting of the flip flops
A:	upper output tag
B:	lower output tag
V_A:	upper output validity
V_B:	lower output validity
C:	upper input tag
D:	lower input tag
V_C:	upper input validity
V_D:	lower input validity
F_C:	upper switching signal for data switch
F_D:	lower switching signal for data switch
Reset:	resets both flip flops

A special situation arises if an attempt is made to switch both inputs to the same output in this control plane. This may arise if a broadcast connection is being set up (i.e., two network outputs requesting connection to the same network input) or in the case of a genuine conflict. When

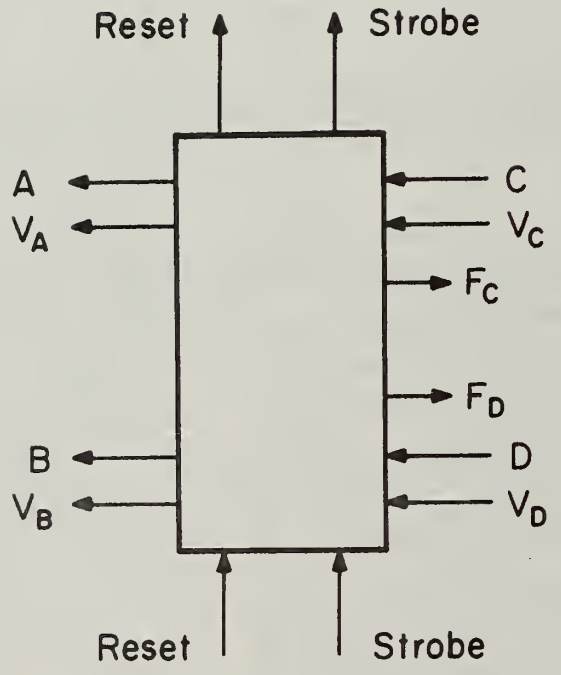


Figure 17. One Control Plane Switching Element

this happens, the signals which will control the data planes are set to connect the broadcast connection but the switch of the control plane is set to transmit only one of the tag signals correctly. The other tag signal is transmitted incorrectly but its associated validity signal is set to zero. (Note that the validity signals may also be used to indicate that a given network output port requires no connection).

Thus, each source tag travels bit by bit from right to left through the control plane. Each bit causes a given stage to switch the next bit through to the next stage.

The logic diagram of one element of the control plane is shown in Figure 18. The two flip flops are set or reset by their respective input (C or D) provided they are enabled by the strobe signal. (These flip flops are the same as the three gate devices shown in Figure 14). The outputs of these flip flops control subsequent transmission of this element in addition to the switching of corresponding elements of each data plane (see Figure 20). Associated with each tag input signal (C or D) are two validity signals (V_C or V_D). These validity signals indicate whether or not the corresponding tag signal is valid. (A tag signal is invalid if it was incorrectly switched in a previous stage or if the corresponding network output port is not requesting a connection).

The logic equations are determined as follows. First, the flip flops are strobed and set or reset as C and D are set or reset. The various transmission states of this element are shown in Figure 19. Numbers on the right of each box correspond, top to bottom to C, V_C, D, and V_D, respectively. Lines in each box represent connections and the numbers on the right indicate transmitted validity signals (V_A, V_B). Don't care conditions were chosen to

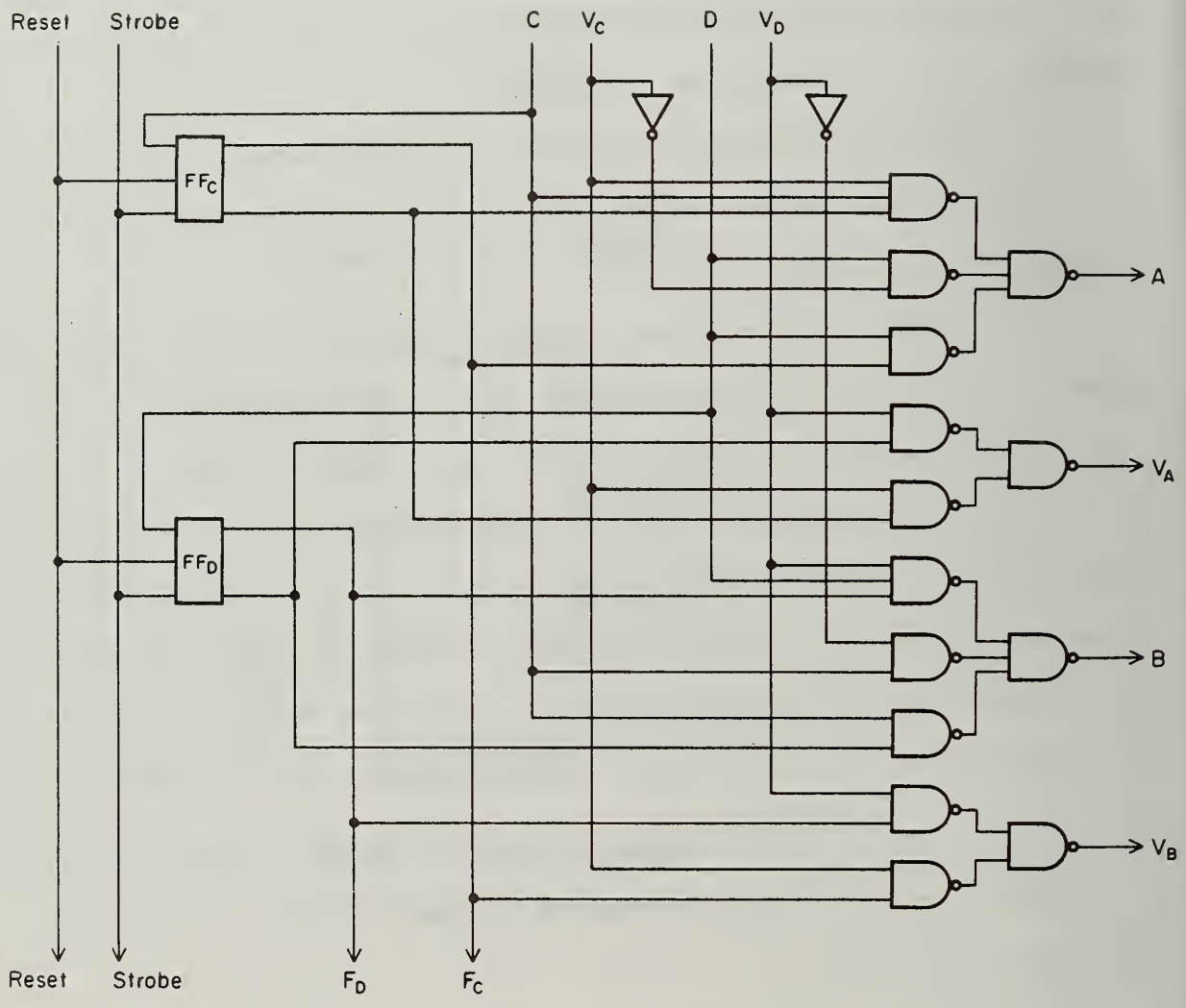


Figure 18. Control Plane Switching Element Circuitry

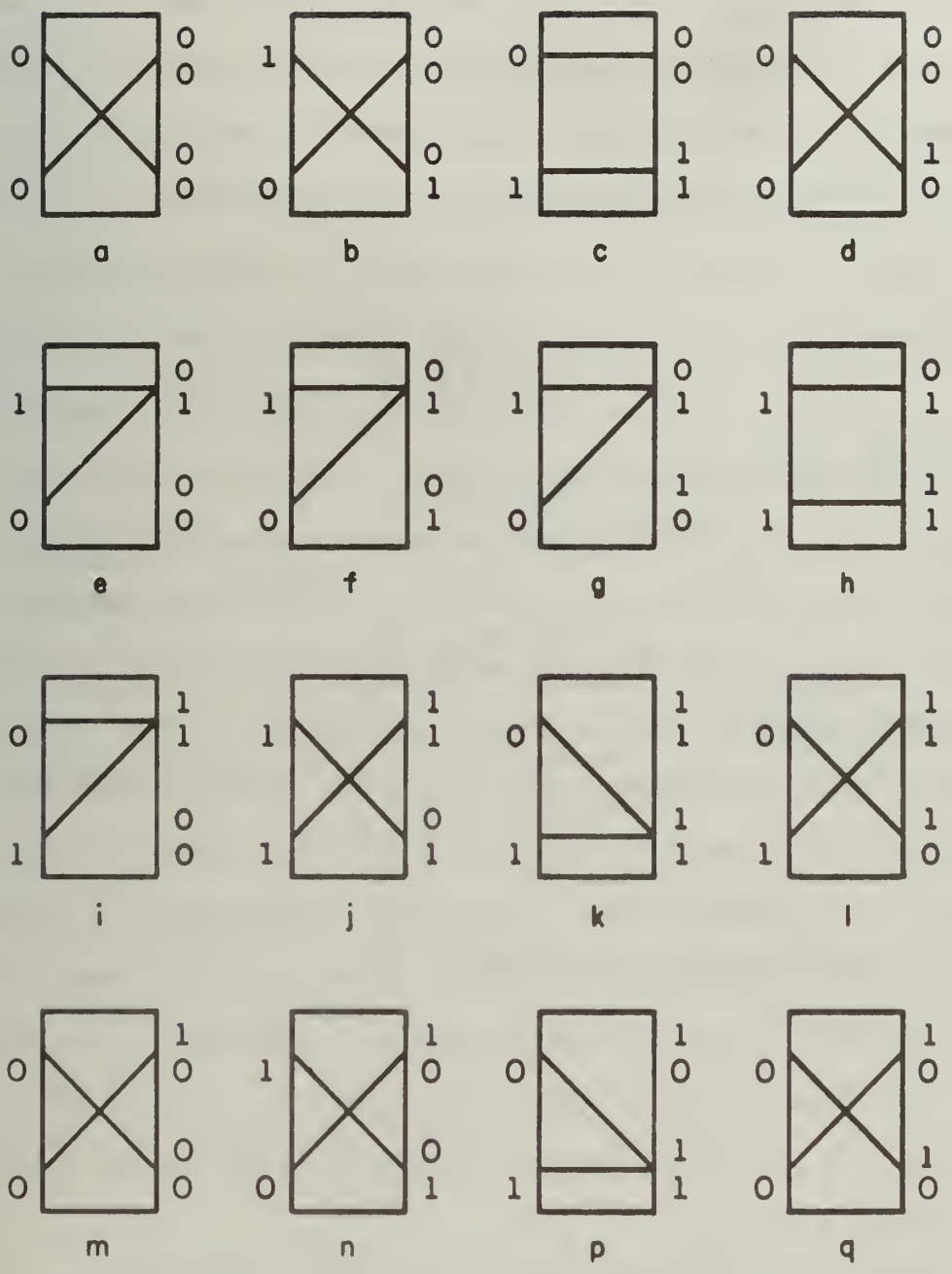


Figure 19. Transmission States of One Control Element

minimize gate counts.

As mentioned earlier, the signals from one of these control planes will be used to control the switching of one or more data planes. Each data plane consists of $\log_2 n$ stages of $n/2$ elements. One such element is shown in Figure 20. Notice that the switching of this element is controlled by the signals F_C, F_D, which are generated by the corresponding element of the control plane (see Figure 3). Figure 21 shows the states of this data switching element. The numbers above each box represent values of F_C and F_D, respectively.

In summary then, each output requests connection to a particular input port by placing the number of that port in the source tag register. The control plane is then clocked $\log_2 n$ times after which the data planes are set to provide some or all of the specified connections between inputs and outputs. If the requested connection does not create conflicts in the sense defined in Chapter 2, then the resulting connection will be the requested connection. If there is a conflict, then some output(s) will not be connected to their requested input. For example, refer to Figure 16. Assume output 0 requests connection to input 0 and output 2 to input 4. Then the first element of the middle stage of Figure 16 will be placed in state f (Figure 19). At this point the request from output 2 will be effectively cancelled, since its validity signal is turned off and in fact output 2 will be connected via the data planes to input 0 instead of 4.

In many cases it would be desirable to detect this condition. This can be done by including a $\log_2 n$ bit source tag along with each element of data at the inputs. After the data is sent through the network each output port checks the source tag received against the source tag requested. Lack

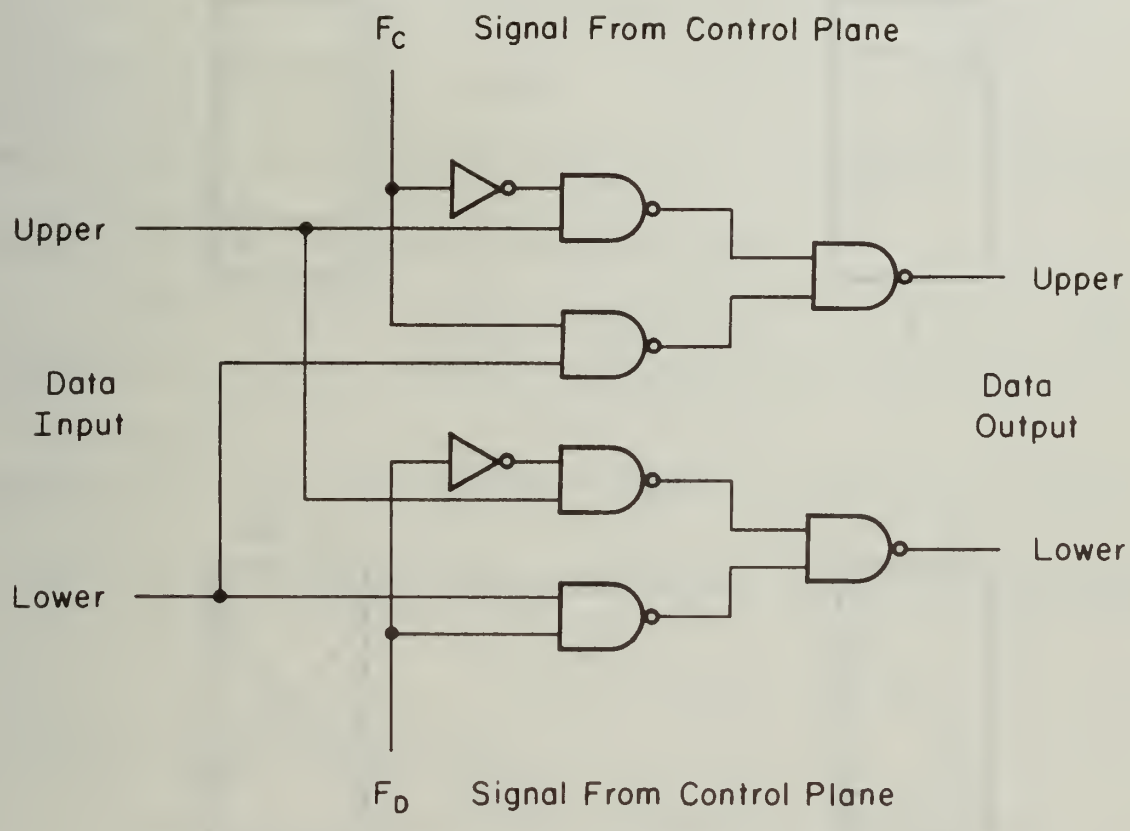


Figure 20. One Element of a Data Plane Using NAND Gates

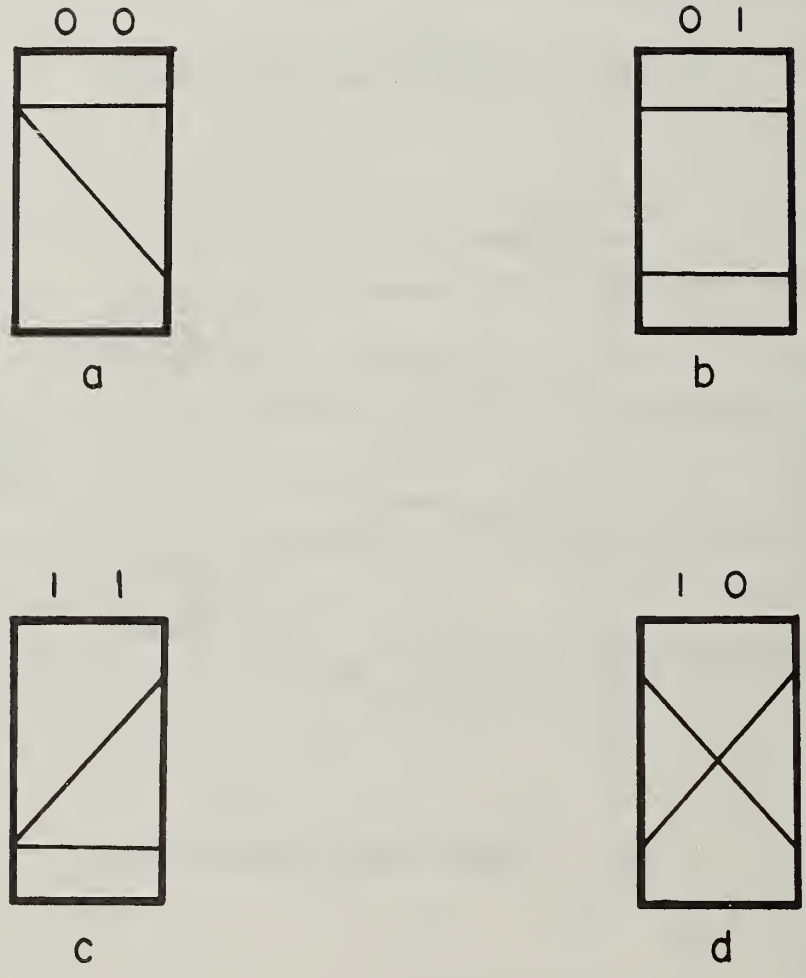


Figure 21. States of One Data Switching Element

of equality indicates a conflict occurred, and the process must be repeated to process the unsatisfied requests.

We have discussed in some detail the construction of the control and data planes of an Ω based network which allows broadcast patterns and is capable of detecting conflicts. If we assume that a flip flop requires 3 gates, then a control plane requires $\frac{n}{2}\log_2 n$ elements of 22 gates (see Figure 16) for a total of $11n \log_2 n$ gates. Each data plane has $\frac{n}{2}\log_2 n$ elements of 8 gates for a total of $4n \log_2 n$ gates. If there are d data planes, then the entire network requires $(4d+11)n \log_2 n$ gates.

Examination of Figure 18 reveals that transmission through a stage requires 3 gate delays and switching of a stage required 3 gate delays. Thus, it requires $\sum_{i=1}^{\log_2 n} 3(i-1) + 3 = \frac{3}{2}(\log_2 n + (\log_2 n)^2)$ gate delays to switch the entire network. This could be changed to $6\log_2 n$ by the addition of appropriate latches between stages.

4.3 Processor-Processor Connections

As readers may have noticed, the topology of the Ω network is equivalent to that of the last $\log_2 n$ stages of both the bitonic sorting networks discussed by Batcher [4] and the binary switching networks discussed by Benes [5]. This in itself is somewhat surprising. It is even more surprising to discover that the interconnections between stages turn out to be the "perfect shuffle" connection discussed by Pease[9] and Stone [12]. We will not present a proof of this result but it can be easily seen by examining Figure 22, which is the same binary Ω network we have been discussing all along. While it appeared earlier (Figure 13) that each pair of stages was

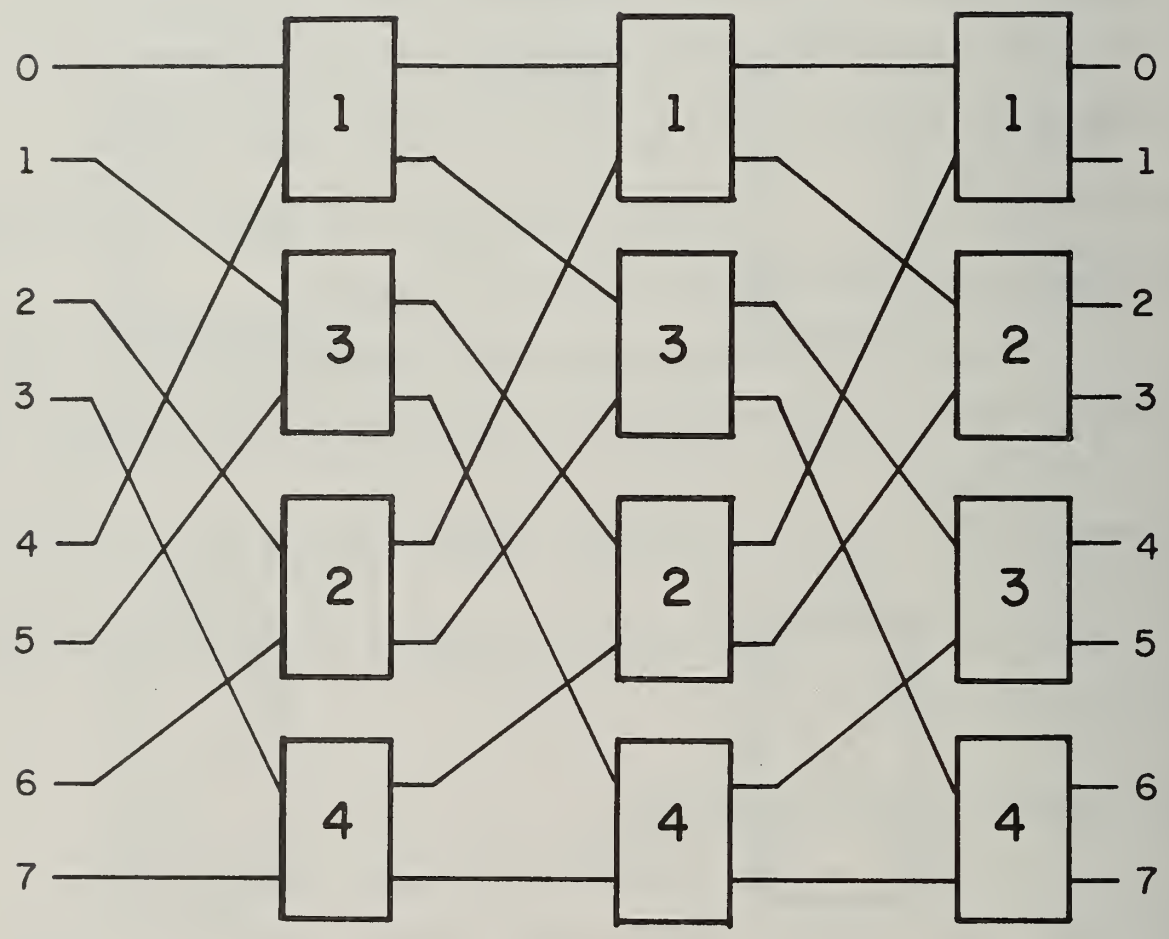


Figure 22. An Ω Network with Identical Interstage Connections Revealed

interconnected differently, (it was expedient to draw them this way to facilitate understanding of how they worked), in fact, a reordering of elements in each stage allows them to be drawn as shown in Figure 22. Thus, it becomes clear that the stage interconnections are identical for each stage. (The numbers on each element indicate their original order as shown in Figure 13).

Since each stage is identical, it appears that it might be possible to save more gates and build just one stage of an Ω network, add some registers, and simply recycle this stage $\log_2 n$ times. We will now examine some particularly effective ways of doing this.

Assume that we have N processors interconnected by the perfect shuffle, as shown in Figure 23. This interconnection was proposed by Stone [12]. The new results, effectively proved in Chapter 2, is that shifts and other patterns required by our solution to the memory conflict problem can be performed in exactly $1 + \log_2 N$ cycles. This is accomplished as follows.

Each processor will have as many as two data words which need to be transmitted to another processor (possibly itself). Associated with each data word will be a $\log_2 N$ bit destination tag and a one bit validity indicator. Assume these interconnections are wide enough to allow parallel transmission of d bits of data, $\log_2 N$ bits of tag and one extra bit. During each cycle the contents of the output registers (see Figure 24) which contain data, tag and validity information are sent via the shuffle connection to the input registers of other processors. Then, depending on the tags and validity bits, the two input registers in each processor may exchange or not as they are gated to the output registers. (It should be noted that one to many connections are not possible using this scheme, but another scheme similar to section 2 of this

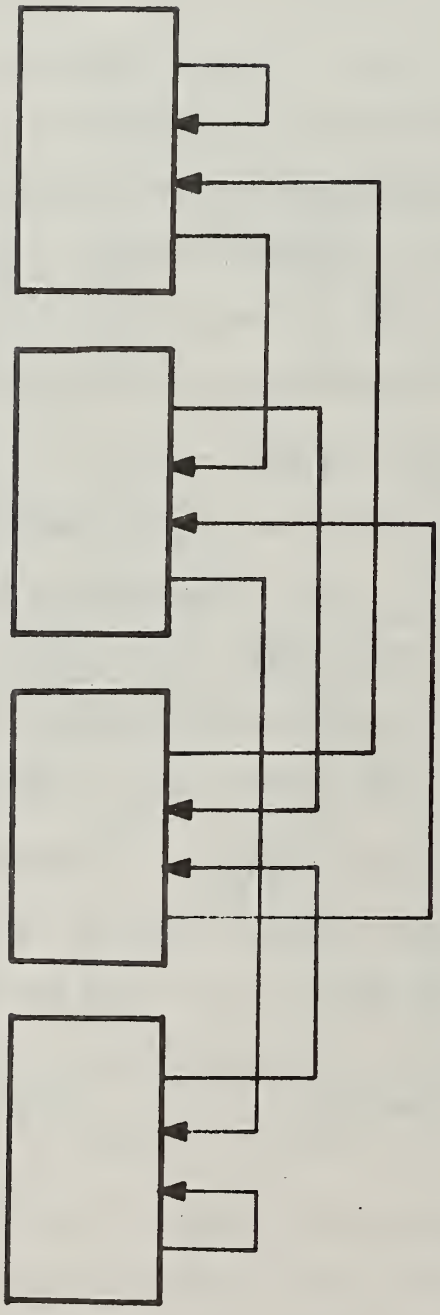


Figure 23. Four Processors Connected by the Perfect Shuffle

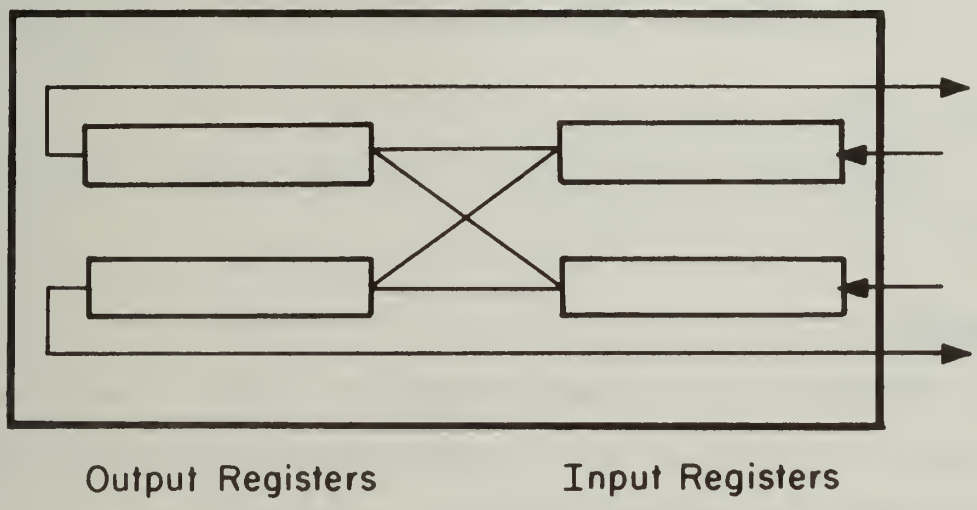


Figure 24. Internal Registers Required for the Processor Interconnection Scheme

chapter is possible which does allow broadcast type connections).

From Chapter 2, we know that any permutation of n numbers which satisfies definition 3 of that chapter can be produced in $\log_2 n$ cycles. Since $n = 2N$, this becomes $1 + \log_2 N$ cycles. In addition, we know that any permutation can be done in $\log_2 n(\log_2 n+1)/2$ cycles. This latter result follows from Batcher's work [4], and the facts that the stage interconnections in the Batcher network are perfect shuffles and that the only data dependent information required by a Batcher element can be obtained from the inputs to that element. It is not known at this time whether or not $\log_2 n(\log_2 n+1)/2$ is a least upper bound, given the above constraints.

One additional result which follows from Batcher is that we can sort 2N numbers using this same hardware.

4.4 Summary

It is difficult to summarize the results of this chapter in terms of actual numbers. In one case we took existing processors, possibly modified them, and formed an Ω network. The primary cost here would not be determined by gates (which presumably are already available in the processors) but by wires, drivers, connectors, etc. In two other cases we actually designed separate Ω networks. We can say that any permutation which satisfies definition 3 of Chapter 2 can be produced in time on the order of $\log_2 n$ cycles and with an order of n gates/stage.

Thus, we have a number of options available:

I. Processor-processor connections.

A. Advantages: possibly cheaper due to the use of already existing gates and can be used to implement a bitonic

sorter with almost no extra logic.

B. Disadvantage: during each cycle, some input must travel a distance of N/2 processors. If the processors are large or there are a lot of them, the wire lengths involved may be a problem.

II. Separate networks.

A. Multistage.

1. Advantages: size and thus wire lengths are smaller.

2. Disadvantage: cost in terms of gates.

B. Single stage.

1. Advantages: size and wire lengths are smaller than processor-processor connections; uses fewer gates than the multistage networks.

2. Disadvantage: probably slower than multistage due to extra clocking requirements and I/O registers.

C. Multistage pipelined.

1. Advantages: size and wire length are smaller than processor-processor connections, higher bandwidth.

2. Disadvantages: cost in terms of gates for the necessary interstage registers and in terms of extra transmission time due to these registers and extra clocking.

It should also be pointed out that it is not necessary to build these networks using binary elements. Larger elements, consistent with the particular implementation technology, can be used with no loss in capability (Theorem 6, Chapter 2) and yield a possibly faster network.

In the next chapter we will review some other networks which might be adaptable to memory-processor connection networks and we will compare these

networks with Ω networks in terms of cost, speed and effectiveness.

5. CONSTRUCTION AND PROPERTIES OF OTHER NETWORKS

In this chapter we will examine five networks which have been proposed as data alignment networks.

5.1 Uniform Shift Networks

A uniform shift network has the capability of "shifting" all inputs ± 1 or $\pm S$ (mod n) positions through any stage or cycle* of the network. However, during any cycle it must shift all inputs by the same distance. It is easy to show that the maximum number of cycles required to perform an arbitrary uniform shift is $\lfloor \frac{1}{2} \lceil \frac{n}{S} \rceil \rfloor + \lfloor \frac{S}{2} \rfloor - 1$. This upper bound is minimized if $S = \sqrt{n}$ (assuming n is a square).+ This type of network generally performs well for uniform shifts, the time being bounded by $\sqrt{n}-1$ cycles. But if non-uniform shifts are required, then we run into trouble.

In the case of ILLIAC IV, it was generally true that significant changes had to be made either to the storage mapping function or to the algorithm itself in order to force the alignment patterns to be uniform shifts. If this could not be done, then a software routine had to be used which resulted in n shifts of +1. Any problem which relied on this routine usually performed so badly that it was no longer considered for use on ILLIAC IV.

Let us examine our alignment requirements in terms of uniform shifts. Recall (Chapter 3) that we characterize our alignment requirements by a function

$$C(x) = (\mu(v(x)), \Phi(x)) ,$$

where we considered the cases $\Phi(x) = x$ or $\Phi(x) = 2x$. In terms of shift

---

*Since all stages would be identical, we may assume only one stage is used but is recycled to necessary number of times.

+The network used in ILLIAC IV is a ± 1, ± 8 uniform shifter.

distances then, the x-th element of the **n** vector will require a shift of

$$S(x) = \Phi(x) - \mu(v(x)) \pmod{n} .$$

Now, if $\Phi(x)$ is independent of x, then it is a uniform shift; otherwise it is not. Table VI summarizes the results of Tables I-III of Chapter 3, in terms of uniform or non-uniform shifts. An × represents a non-uniform shift while O represents a uniform shift. As we can see, in no case are more than two of the n-vectors listed accessible by uniform shifts. (We have not even considered broadcast patterns since this kind of network cannot produce them).

There are several ways of building such a network. One way is to use a single stage as shown in Figure 25. (Figure 25 actually shows two stages in order to more easily show the connections. One may mentally fold the second stage into the first). Figure 26 shows one of the n elements. Clearly, the interstage wiring for this type of network is more complex than that of the Ω network. This network requires 5n gates/stage (see Figure 20 of Chapter 4).

Finally, we may consider connecting the N processors together with a ± 1, $\pm S$ connection and using the processors as a single, recycleable stage. Each processor requires 4 input and 4 output lines, while using Ω connections required only two input and two output lines. Further, the Ω connections allowed us to align 2N data words while the ± 1, $\pm S$ connection only allows us to shift N words.

Let us propose that we add ± 1 connections to the Ω connections. Then, we have a system whose cost is approximately the same as the ± 1, $\pm\sqrt{N}$ system, but which can do any uniform shift in $\leq \log_2 N$ cycles as opposed to $\leq \sqrt{N}$ cycles for the ± 1, $\pm\sqrt{N}$ system. Additionally, the ± 1, Ω system can produce permutations which are not uniform shifts.

In spite of the apparently overwhelming evidence against the uniform

Table Number	I-A	I-B	I-C	I-D	I-E	II-A	II-B	III-A	III-B	III-C	III-D
rows	0	0	0	0	0	0	0	×	×	0	0
columns	×	0	×	×	×	×	×	×	0	×	×
forward diagonals	0	×	×	×	×	0	×	×	×	×	×
reverse diagonals	×	×	×	×	×	×	0	×	×	×	×
$\sqrt{N} \times \sqrt{N}$ partitions	×	×	×	0	×	×	×	×	×	×	×
memories	N	N	N	N	N	2N	2N	2N	2N	2N	2N
skew	0	1	3	$\sqrt{N}$	$\sqrt{N}+1$	0	2	0	2	1	$\sqrt{N}+1$
skip	1	1	1	1	1	1	1	1	1	2	2
$\Phi(x)$	x	x	x	x	x	x	x	2x	2x	2x	2x

Table VI. Uniform (0) or Non-Uniform (×) Shift Alignment Patterns

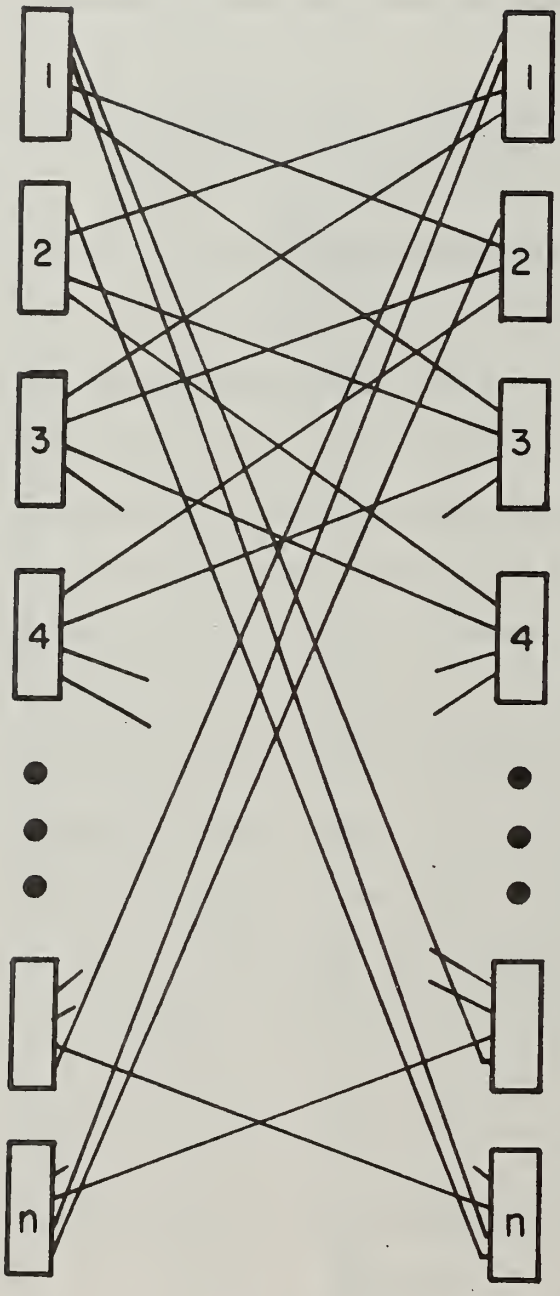


Figure 25. A $(\pm 1, \pm 2)$ Uniform Shift Network

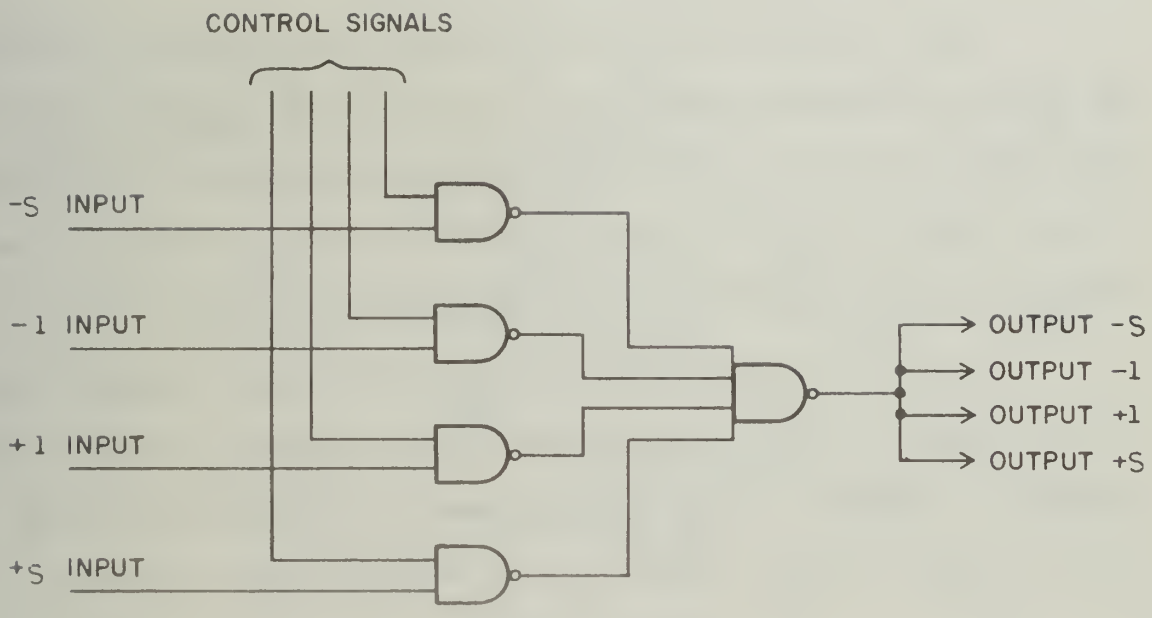


Figure 26. One Element of a ($\pm 1, \pm S$) Shift Network Using NAND Gates

shift networks, we will examine one more such network, the barrel shifter.

5.2 The Barrel Shifter

The barrel shifter is capable of doing any uniform shift. It is conceptually a simple device. It consists of $\log_2 n$ non-identical stages. The i-th stage is capable of switching all inputs either 0 or 2^{i-1} positions. This is shown in Figure 27 for a four port network.

Thus, the barrel shifter requires $\log_2 n$ stage delays for any uniform shift. It requires 3n gates/stage. This is to be compared with the Ω network which requires 4n gates/stage and $\log_2 n$ stage delays. Unfortunately, the stage interconnections of the barrel shifter are not uniform and thus we cannot build just a single stage and recycle it $\log_2 n$ times.

The only advantage of the barrel shifter is that the upper bound on the time required to perform any uniform shift is less than the same bound for the ± 1, $\pm S$ shifter. But the latter network has a smaller lower bound. The barrel shifter is clearly inferior for our purposes to the Ω network.

5.3 The Batcher Network

The Batcher network [4] was first proposed as a sorting network (see Figure 29). However, it should be clear that any network capable of sorting n numbers is also capable of switching n numbers. We will not go into the details of this network. It is sufficient to say that the networks consist of $\log_2 n(1+\log_2 n)/2$ stages. Each stage is (or can be) interconnected by the perfect shuffle and each of the n/2 elements of each stage is capable of ordering its two inputs into descending or ascending sequence. Each element can be built with 13 NOR gates for a total of 13n/2 gates/stage (see [4]). This is for the "control" plane. Data planes can be identical to

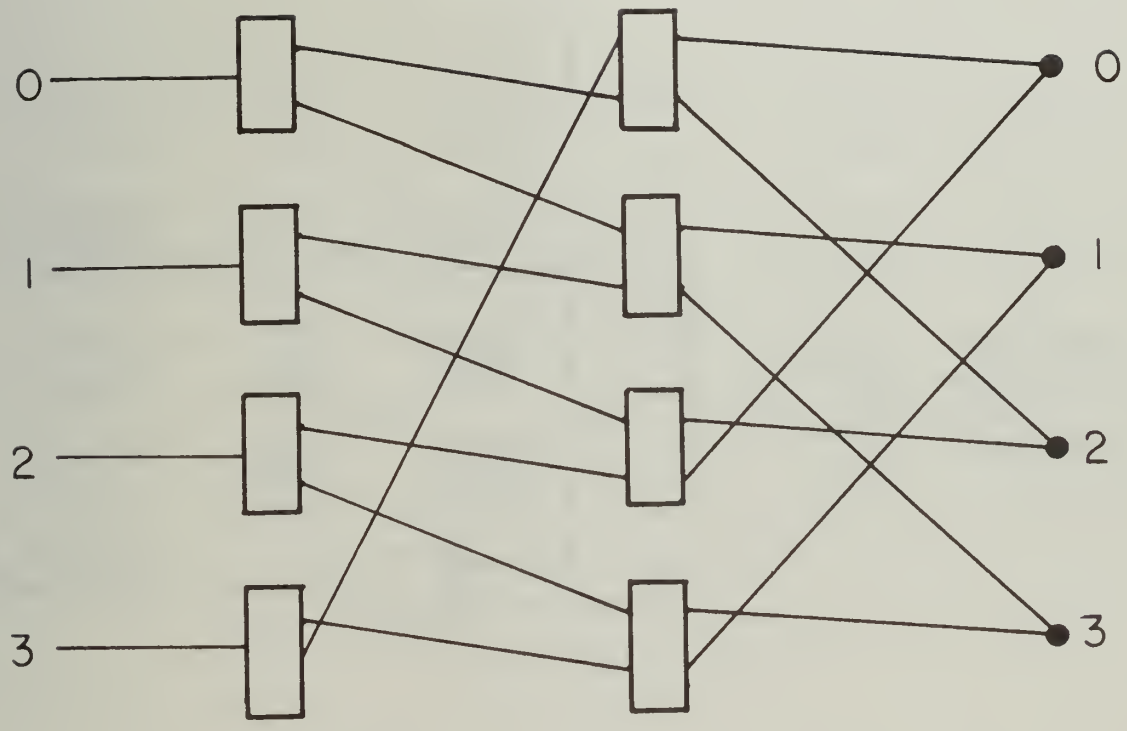


Figure 27. A Four Port Barrel Shifter

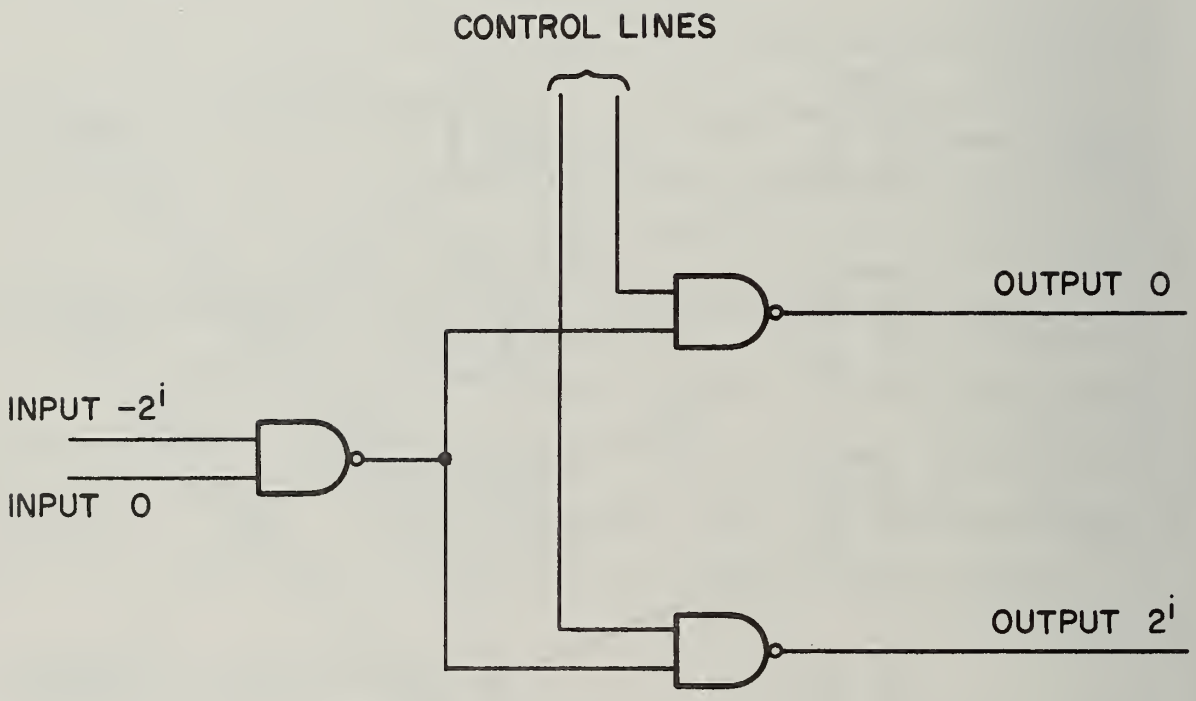


Figure 28. One Barrel Switch Element

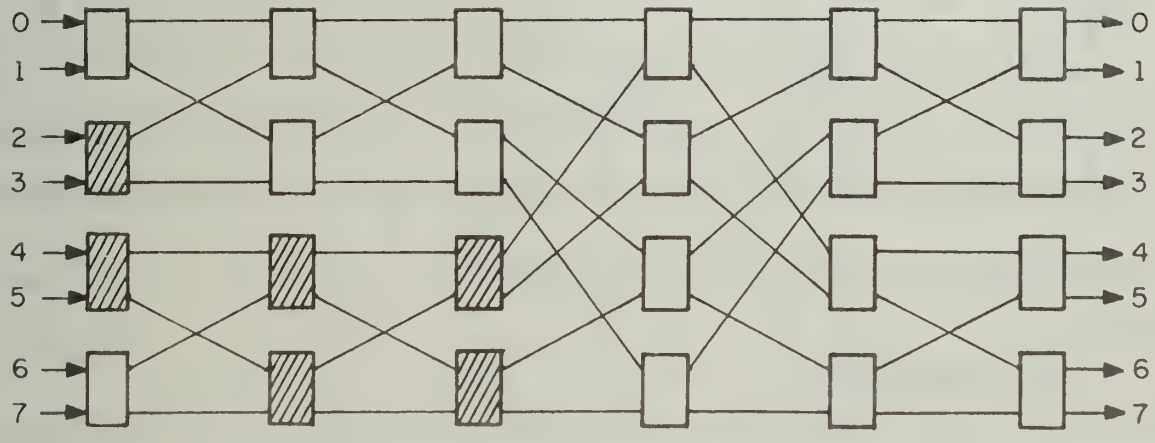


Figure 29. An 8 × 8 Batcher Network. Crosshatched Elements Sort in Descending Order

those of the Ω network except there are more stages.

We could build only one stage and recycle it $\approx(\log_2 n)^2$ times in order to save gates. However, one additional problem arises. Each element will require some extra logic since during some cycles it must produce a descending order while during others it must produce an ascending order. We will ignore this problem.

Finally, we could use the processors themselves as Batcher elements. Now we have a system which is practically identical to that of section 3, Chapter 4. Batcher's results tell us that using this system we can do any permutation in an order of $(\log_2 n)^2$ cycles. The results of Chapter 2 tell us that if the permutation satisfies definition 3 of Chapter 2 (which includes uniform shifts), then it can be done in $\log_2 n$ cycles. Other permutations may require fewer or greater than $\log_2 n$ cycles.

5.4 The Benes Network

While Benes did not invent this network, his extensive discussion of this network prompted us to name it after him [5]. It was originally intended for telephone traffic switching.

The data planes of a Benes network consist of $(2\log_2 n)-1$ stages of $n/2$ elements interconnected by the perfect shuffle and with the exception of the number of stages they are identical to the data planes of the Ω and Batcher networks (see Figure 30).* Benes shows that the typology of such a network is sufficient to produce any permutation of inputs. Unfortunately, the determination of the switching of each element in the network is complex. The best known result [13] requires time on the order of $n \log_2 n$

---

*Actually, Benes considered these networks in a more general form. We consider here only binary networks.

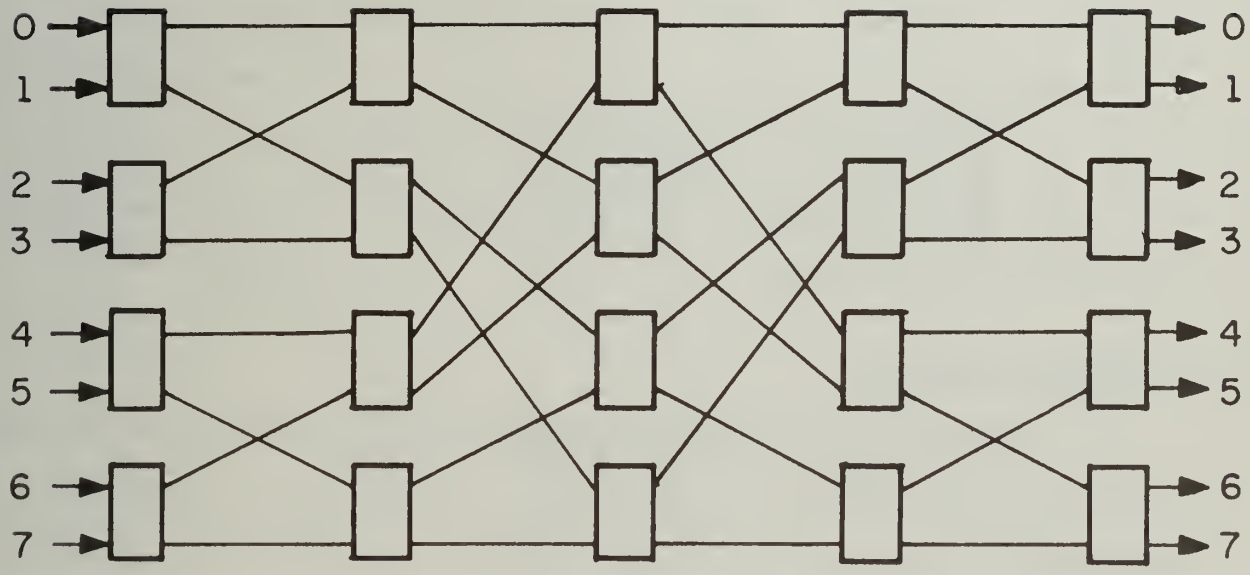


Figure 30. A Binary 8 × 8 Benes Network

operations. Thus, it would appear that this network would not be practical in situations where the control signals have to be determined in real time. However, in some instances it might be possible to compute these signals ahead of time, say in a compiler, and in these cases the Benes network might be useful.

5.5 The Crossbar Switch

A conceptual diagram of a crossbar switch is shown in Figure 31. This could be implemented as shown in Figure 32 using fan-in logic and limiting fan-in and fan-out to 2. Control signals for this network are derived directly from the source tags. This circuit can produce any set of one-to-one or one-to-many connections. It requires an order of n^2 gates and log n gate delays for transmission.

5.6 Network Comparison

At this point we have discussed these networks sufficiently to allow us to compare them as much as possible with Ω networks. We will begin by assuming that we will build a complete network, i.e., we will not build just a single stage which can be recycled. We will follow this by a comparison based on single stage design. In this first case we will not consider the ±1, ±S uniform shifter since it is clearly nonsensical to build such a network. For each of the Ω, barrel shift, Batcher and Benes networks we will compare control gates, data gates, switching time, data transmission time, and capability. Table VII shows the relative values of these parameters. The expressions listed represent only relative magnitudes and do not represent actual values.

Now let us assume we interconnect the N processors in by ±1, $\pm\sqrt{N}$ connections or by the perfect shuffle connections. Table VIII summarizes these results. The only difference between the Ω, Batcher, and Benes "networks" is

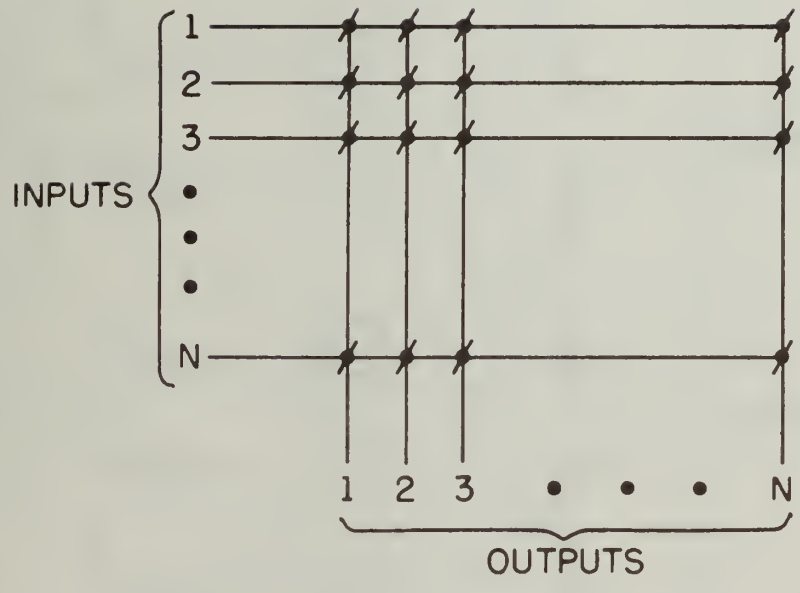


Figure 31. A Conceptual Diagram of an N × N Crossbar Switch

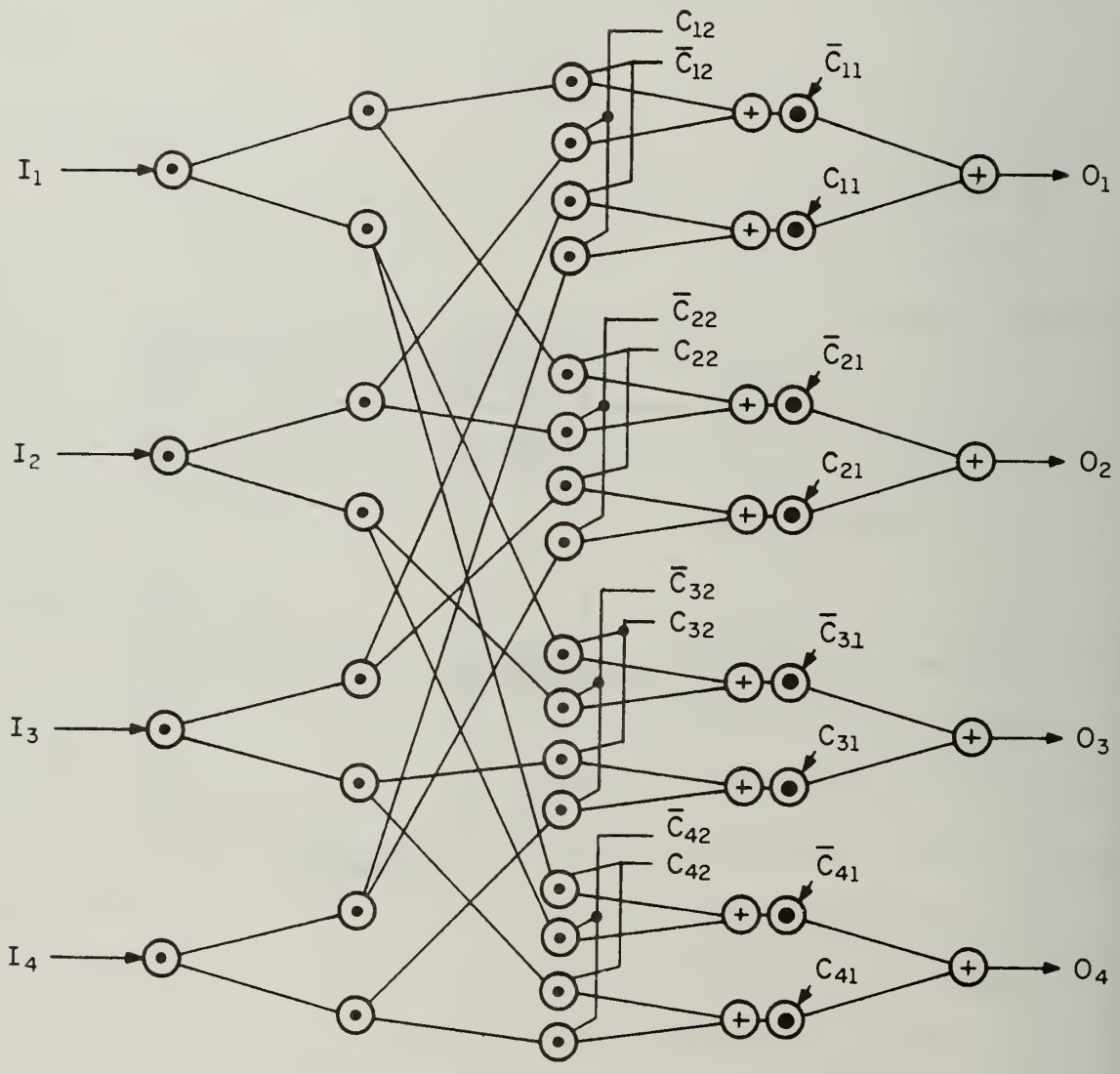


Figure 32. A 4 × 4 Crossbar Using AND-OR Components and Controlled by Source Indexing

	Control Gates	Data Gates	Control Time	Data Transmission Time	Capabilities
Barrel Shifter	$0^{(a)}$	$n \log n$	1	$\log n$	uniform shifts
Ω	$n \log n$	$n \log n$	$\log n$	$\log n$	any permutation satisfying definition 3 of Ch. 2 (includes uniform shifts)
Batcher	$n(\log n)^2$	$n(\log n)^2$	$(\log n)^2$	$(\log n)^2$	any permutation
Benes	$> n \log n^{(b)}$	$2n \log n$	$n \log n$	$2\log n$	any permutation
Crossbar	---	n^2	1	$\log n$	any permutation

(a) Negligible.

(b) Two log n bit, n word memories are required for the "fast" switching algorithm as well as a finite state machine.

Table VII. Comparison of Five Networks

	Input/Output Line Pairs Per Processor	Cycles Required for Uniform Shift	Upper Bound on Cycles Required for an Arbitrary Permutation	Switching Time Per Cycle
$\pm 1, \pm\sqrt{N}$	4	$1 \leq c \leq \sqrt{N}-1$	N-1	0
Batcher - Ω	2	$\log_2 n$	$(\log_2 n)^2$	1
Benes	2	$2\log_2 n$	$2\log_2 n$	n
± 1, Batcher - Ω	4	$1 \leq c \leq \log_2 n$	$(\log_2 n)^2$	1

Table VIII. Comparison of Processor Interconnection Schemes

the decision process required in each processor during each cycle. In the case of the Ω network, this requires examination of one bit each of two $\log_2 N$ bit tags in each processor during each cycle. For the Batcher network this requires arithmetic comparison of two $\log_2 N$ bit tags in each processor during each cycle. Since these two are so similar, we have combined them in Table VIII. The Benes network using Opferman and Tsao-Wu's algorithm requires a control time on the order of $n \log_2 n$ but cannot be done based on examination of the two data elements in each processor at each cycle. The switching time shown in Table VIII simply reflects the total time divided be the number of cycles.

It should be pointed out that the ± 1, $\pm \sqrt{N}$ connection can shift N inputs and during each cycle only one of the four I/O line pairs is used. The shuffle connections allow permutation of $n = 2N$ numbers and during each cycle both I/O line pairs are used. We have shown in Chapter 3 that using 2N memories resulted in the ability to fetch significantly more N-vectors. Thus, the ability to handle any N out of 2N inputs may be significant.

6. CONCLUSION

The problem of data alignment for a large parallel computer has been investigated before, but previous results generally relate only to interprocessor connections or switching networks; memory assignment; or (rarely) control algorithms for switching networks. For example, Benes [5] investigated the properties of some $2\log_2 n$ stage binary switching networks which might provide a solution except that no control algorithm is known which is fast enough to allow the fast word-after-word switching required in a highly parallel computer. Kuck and Budnik [2] proposed a solution to the memory assignment problem which allows a wide variety of vectors to be fetched from a parallel memory system without conflicts, but the alignment and indexing problems indicate that this solution is probably not practical.

We have attempted to consider all these problems simultaneously in order to arrive at a realistic solution. In this paper we have presented such a solution. In order to prove the merits of this solution, we began in Chapter 2 by deriving some theoretical properties of a particular type of network which we called an Ω network. We showed, for example, a necessary and sufficient condition (definition 3) for a given permutation to be producible by a given Ω network. Then, in Chapter 3, we derived some important N-vectors, several storage assignment schemes, and considered several memory systems. We showed in each case whether or not certain N-vectors could be fetched in parallel without conflict and aligned by a binary Ω network. We found that by using twice as many memories as processors, a $(\sqrt{N}+1)$-skew, 2 skip memory assignment, and a binary Ω network we could fetch and align at least the four most important N-vectors (rows, columns, diagonals and $\sqrt{N} \times \sqrt{N}$ positions) without conflicts.

Finally, in Chapter 5, we discussed several other types of networks

and their relationships to the Ω network. Here we showed that in general the uniform shift networks are not flexible enough to be useful in general parallel computing applications and, in fact, need not be considered since Ω networks are more flexible and either cost less or at least do not cost significantly more than uniform shift networks. In addition, comparison with more general networks (Benes [5] and Batcher [4]) which are capable of producing any arbitrary permutation showed the Ω network to be cheaper and faster.

We also discussed the possibility of interconnecting the processors together in such a way as to form one stage of an Ω network. By recycling this single stage the correct number of times we could effectively simulate an Ω, Batcher, or Benes network. As it turns out, this particular interconnection of processors was proposed by Stone [12] and called the perfect shuffle. Using the results of Chapter 2, we were able to prove the new result that certain important permutations (including uniform shifts) could be done in exactly $\log_2 N$ cycles where N is the number of processors. In comparison, the ILLIAC IV ± 1, $\pm \sqrt{N}$ connections allow any uniform shift in time $\leq \sqrt{N}$. If we add ± 1 connections to the perfect shuffle connection we get a system with the same number of I/O line pairs (probably the major cost item) and which can handle any uniform shift plus other important permutations in time $\leq \log_2 N$.

Thus, while we may not have found a panacea, we have demonstrated a class of network which provides a feasible solution to the data alignment and memory conflict problems in a parallel vector processing machine.

LIST OF REFERENCES

[1] G. H. Barnes, et. al., "The Illiac IV Computer," IEEE Transactions on Computers, Vol. C-17, pp. 746-757; August 1968.

[2] P. Budnik, and D. J. Kuck, "The Organization of Parallel Memories," IEEE Transactions on Computers, C20, p. 1566; December 1971.

[3] P. W. Kraska, "Array Storage Allocation," M.S. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 344; August 1969.

[4] K. E. Batcher, "Sorting Networks and Their Applications," Proceedings of the Spring Joint Computer Conference, 1968; pp. 307-314.

[5] V. E. Benes, Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, New York; 1965.

[6] I. M. Vinogradov, Elements of Number Theory, Dover Publications; 1954.

[7] D. Shanks, Solved and Unsolved Problems in Number Theory, Vol. I, Spartan Books, Washington, D. C.; 1962.

[8] Y. Muraoka, "Storage Allocation Algorithms in the TRANQUIL Compiler," M.S. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Report No. 297; January 1969.

[9] M. C. Pease, "An Adaption of the Fast Fourier Transform for Parallel Processing," Journal of the ACM, Vol. 15, pp. 252-264; April 1968

[10] J. E. Stevens, "A Fast Fourier Transform Subroutine for Illiac IV," Center for Advanced Computation, University of Illinois at Urbana-Champaign, Document No. 17; October 1971.

[11] D. J. Kuck, and A. H. Sameh, "Parallel Computation of Eigenvalues of Real Matrices," Proceedings of the IFIP Congress, p. TA-1-24; 1971.

[12] H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, C20, pp. 153-161; February 1971.

[13] D. C. Opferman, and N. T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks," Bell System Technical Journal, Vol. 50, pp. 1579-1618; May-June 1971.

VITA

Duncan Hamish Lawrie was born in Chicago, Illinois, on April 26, 1943. He received the B.A. degree from DePauw University in Greencastle, Indiana, and the B.S.E.E. degree from Purdue University, both in 1966, and the M.S. degree in Computer Science from the University of Illinois in 1969. While at the University of Illinois Mr. Lawrie held a NASA Traineeship in Computer Science and a Research Assistantship in the Department of Computer Science. He also worked on the Illiac IV project as Senior Research Programmer in charge of language development and computer operations. He is a member of Tau Beta Pi, Eta Kappa Nu, the Association for Computing Machinery, the Institute of Electrical and Electronic Engineers, and is an associate member of Sigma Xi. He has published two papers, "The Use and Performance of Memory Hierarchies: A Survey," in Software Engineering, Vol. I, Academic Press, New York (1970), and "Interconnection Networks for Processors and Memories in Large Systems," presented at the COMPCON 72 convention in San Francisco, California (1972), both with D. J. Kuck.

BLIOGRAPHIC DATA IEET	1. Report No. UIUCDCS-R-73-557	2.	3. Recipient's Accession No.
Title and Subtitle MEMORY-PROCESSOR CONNECTION NETWORKS			5. Report Date February 1973 6.
Author(s) Duncan Hamish Lawrie			8. Performing Organization Rept. No. UIUCDCS-R-73-557
Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. US NSF GJ 27446
. Sponsoring Organization Name and Address National Science Foundation Washington, D. C.			13. Type of Report & Period Covered Doctoral - 1972 14.

. Supplementary Notes

. Abstracts

In order to utilize the potential speed of a SIMD type parallel processor it is necessary to arrange data in the memory system so that subsets of this data can be fetched in parallel without memory conflicts. Additionally, we must provide sufficient memory-processor paths to allow the data to be correctly aligned with the processor array. In this paper we present several storage mapping algorithms together with a memory-processor interconnection network. We demonstrate the cost and effectiveness of these and compare them with other networks which have been proposed for this application.

. Key Words and Document Analysis. 17a. Descriptors

Indexing Networks
Sorting Networks
Switching Networks
Crossbar Switch
Memory-Processor Connection

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	121
	20. Security Class (This Page) UNCLASSIFIED	22. Price

MAR 13 1973

OCT 24 1913

UNIVERSITY OF ILLINOIS-URBANA
3 0112 052121743