Self-routing Schemes in Parallel Memory Access[1]

by

Rajendra Boppana and C. S. Raghavendra

Technical Report No. CENG 89-36
12 December 1989

e-mail: raghu@surya.usc.edu
phone: (213) 743-5532
Dept. of Electrical Engineering–Systems
University of Southern California
Los Angeles, CA 90089–0781

[1]This research is supported by the NSF grant No. MIP 8452003, and a grant from AT&T.

Self-routing Schemes in Parallel Memory Access

Abstract

In this paper, we give a generalized solution to the problem of conflict free access of various templates of data of a matrix, when stored in memory modules in an SIMD multiprocessor system. The important features of our method are: (a) compact representation of a skewing scheme, (b) simple address computation, (c) use of self-routing schemes to set up the interconnection network, and (d) a general framework for the study of skewing schemes. In our method, each template access of interest will be a linear permutation on the processor address. The linear permutation involved determines the types of templates accessible. For parallel access of the most important templates, namely, row, column, main diagonal, and square blocks, the interconnection network needs to realize only the class of linear-complement permutations. It is known that with Beneš or Omega as the interconnection network, one can efficiently self-route these permutations; this compares favorably with the schemes proposed by other researchers who assume that a crossbar is available for processor-memory interconnections. Hence, the approach given in the paper can be used to solve the data alignment problem for the existing parallel machines such as IBM RP3, BBN Buterlfy machine, NYU Ultracomputer, etc. This is a generalized solution to the data skewing problem, and encompasses the previous efforts by other researchers as special cases.

Key words: data alignment, interconnection network, linear permutations, matrix storage, parallel memory systems, scrambled skewing, self-routing, shuffle-exchange networks.

Contents

List of Figures

1 Introduction

In SIMD multiprocessors, access to shared data in the memory modules plays an important role in the overall performance. For specific problems, knowing the data access patterns by processors, one can allocate data to memory modules initially so that high parallelism can be achieved in data access at run time. By the parallel access of data, we mean that the access of data is free of memory and interconnection network conflicts[1]. This problem of arranging data in the memory modules to allow parallel access is called the data alignment problem; a special case of this problem is, the storing of a matrix of data in memory modules so that various portions of it can be accessed with high parallelism.

Consider an SIMD multiprocessor system (figure 1) with N processors, N memory units, and an interconnection network. We assume that the interconnection network can realize some permutations so that it can, simultaneously, provide N data paths between processors and memory units. Examples of such interconnection networks include crossbar, Beneš, and Omega networks. The problem addressed in paper is: store an $m \times m$ matrix, $m \geq N$, in the memory modules such that various N-(element)subsets (also called, *templates*) of the matrix are retrieved from memory modules without memory and interconnection network conflicts.

In figure 2, we illustrate two methods to store the elements of a 4×4 matrix, $A = (a_{i,j})$, in 4 memory units, numbered $0, \ldots, 3$. Given a storage scheme to store the data matrix A, we can represent it in the form of a mapping matrix. A mapping matrix is obtained from A by replacing each $a_{i,j}$ by the number of the memory unit in which it is stored. In the first storage scheme (the mapping matrix in figure 2(b)), any row can be accessed without memory conflicts, since, in any row, the symbols appearing are distinct. The same can not be said about the column access, because, in any given column, a particular symbol appears 4 times; hence, to access a column of the matrix, four consecutive memory accesses to a memory unit are to be made, before proceeding with the computations using the column. In the second method (figure 2(c)), any row as well as any column can be accessed without memory conflicts. This second method of storing the matrix elements is called the skewed storage method. If various templates of a matrix are to be accessed conflict free, then it is essential that the matrix be stored in some skewed form; simplistic approaches such as the row major order storing (figure 2(b)) of a matrix are not satisfactory.

A good data skewing scheme, in addition to allowing conflict free access of various data templates, should have the following features.

- Given the row and column indices of a data element, the computation of its address

[1]If a memory module contains more than one element of the data to be accessed, then there is a memory conflict in accessing that data. Network conflicts exist, if the interconnection network can not set up, in one pass, paths from processors to memory units to access data that are free of memory conflicts.

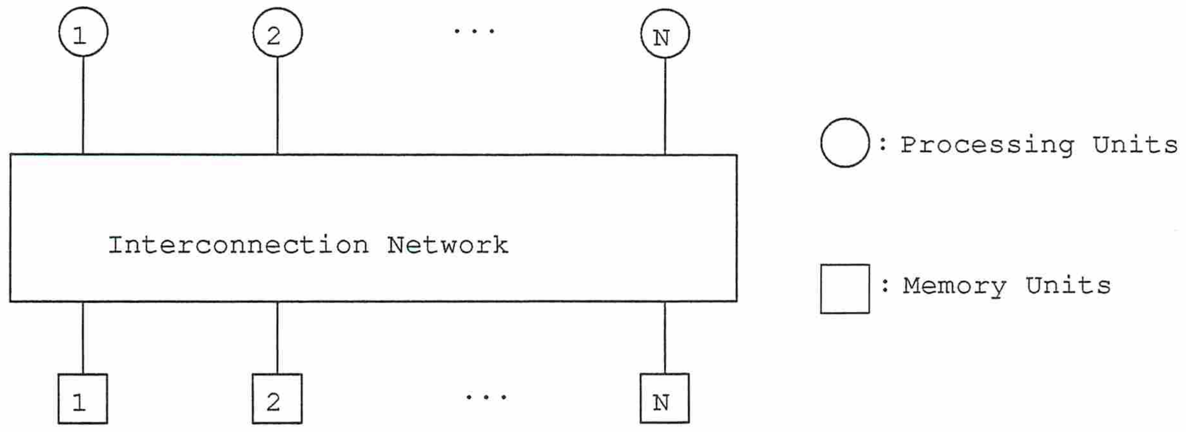

Figure 1: An SIMD multiprocessor.

$$\begin{array}{|cccc|}\hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \qquad \begin{array}{|cccc|}\hline 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ \hline \end{array} \qquad \begin{array}{|cccc|}\hline 0 & 3 & 1 & 2 \\ 1 & 2 & 0 & 3 \\ 2 & 1 & 3 & 0 \\ 3 & 0 & 2 & 1 \\ \hline \end{array}$$

The data matrix A — Mapping matrix 1 — Mapping matrix 2

a — b — c

Figure 2: Two methods of storing a 4×4 data matrix

— the memory module in which the element is stored and location within the module — should be fast and efficient with very little hardware requirements.

- An inexpensive interconnection network such as a multistage interconnection network with simple control algorithm should be sufficient to provide simultaneous data paths between processors and memory modules, during the parallel access of data.
- The skewing scheme should be represented compactly so that overheads are minimized.
- The skewing scheme should allow efficient re-skewing of data between phases of computation when necessary.

A large number of researchers [1, 2, 7, 9, 13, 14, 16, 17, 23, 26] studied the matrix storage problem due to the frequent occurrence of matrix computations in a large number of problems related to linear algebra. Often, researchers assume the availability of an expensive interconnection network such as crossbar and propose solutions to store a data array without memory conflicts; this is not suitable for large matrices. Furthermore, solutions of this type are difficult implement in the existing parallel machines, which contain a multistage interconnection network or a sparsely connected static interconnection network. Some researchers provided schemes that satisfy the above criteria; however, these solutions are inefficient or do not achieve conflict free access of a large variety of data templates.

In this paper, we provide a generalized solution to the data access problem for storing matrices in memory modules. An important feature of the solution is, the data transfer functions[2] that arise in the parallel access of many templates of data are in the class of linear permutations, which is well studied by the researchers. As a result, the limitations of interconnection networks are overcome in the parallel access of many useful subsets of a matrix. In fact, using the solution presented, we devise storage schemes, that can be used in existing parallel machines with the Omega interconnection network such as IBM RP3, BBN Butterfly, NYU Ultracomputer, and Cedar multiprocessor systems. This compares favorably with any of the previously proposed schemes.

The rest of the paper is organized as follows. In the next section, we discuss the related research and the proposed solution method. Next, we explain the notation used in this paper. In section 4, we give the storage scheme, and discuss about the choice of a storage scheme and its effect on the conflict free access of subsets; we also discuss the problem of address generation, and the issue of realizing the transfer functions that arise during the parallel access. In section 5, we devise skewing schemes for which the data transfer functions in the parallel access of templates of a matrix are realized by an

[2]The functions that model the data paths to be set up by the interconnection network for the parallel access of templates as permutations from inputs to outputs of the interconnection network are called the data transfer functions.

Omega or an inverse Omega network. Next, we discuss about the use of the proposed method in distributed memory SIMD systems, the hypercube and the mesh computers We conclude the paper with a discussion and directions for further research.

2 Related work

In this section, we, briefly, discuss the skewing schemes proposed by the other researchers and the solution proposed.

The skewing schemes can, broadly, be classified as: the linear skewing schemes, and the non-linear skewing schemes. In a linear skewing scheme [7, 14, 23, 26], element (i, j) is stored in the memory unit given by a linear combination of i and j; that is, element (i, j) is stored in the memory unit $ai + bj \pmod{N}$, for some fixed integers a and b. Any skewing scheme that is not linear is called a non-linear skewing scheme.

Linear skewing schemes are not general enough to allow conflict free access to various templates such as row, column, main diagonal, etc., when N is even [7, 14]. Various solutions given to this problem are as follows. Budnik and Kuck [7] studied the matrix storage problem initially and proposed a linear skewing storage scheme that can be used to store an $N \times N$ matrix in a system with N processors and $N + 1$ memory units, where N is a power of 2. A similar scheme is used in the prime memory system of Burroughs Scientific Processor [15], which has 16 processors, 17 memory modules, and a 16×17 crossbar as processor-memory interconnection network. The disadvantages of this approach, for large N, are: (a) address computation is complicated, (b) interconnection network is expensive, and (c) unequal allocation of data to the memory modules, which might complicate the memory management. Lawrie [14] proposed the, now well-known, Omega network and discussed a linear skewing scheme to store a $N \times N$ matrix in $2N$ memory units so that rows, columns, main diagonal, back diagonal, and square blocks of the matrix can be accessed in one pass of through the network. However, the use of twice the minimum number of memory modules necessary is not desirable.

It is possible to design non-linear skewing schemes such that these templates are obtained conflict free, using N memory modules, even when N is a power of 2. However, arbitrary non-linear skewing schemes should be avoided in the interest of fast address computation and reducing the cost of interconnection network.

There is a special class of schemes called, scrambled storage schemes, that satisfies the criteria of a good skewing scheme. The class of scrambled storage schemes are defined for the case N a power of 2 as follows. Element (i, j) of the matrix is stored in the memory module whose binary representation is a linear permutation of of its row and column indices, i and j, respectively. The storage schemes proposed by Batcher [2], Pease [21], Balakrishnan et al. [1], Lee [16, 17], and Kim and Kumar [13] belong to this class. Fairlong et al. [9] discussed this class of schemes in a general frame work and proposed the class of XOR schemes.

In this paper, we consider a special class of scrambled storage schemes, which provide skewed storge using linear permutations on the indices of the elements. These are called *b*itwise *li*near *p*ermutation (blip) schemes. The schemes proposed by Fairlong et al. are similar to the blip schemes and all the other previously proposed scrambled schemes, are also in the class of blip schemes. The blip schemes provide conflict free access to various templates such that the corresponding data transfer functions can be realized by a multistage interconnection network with self-routing. In fact, given the access requirements, one can devise an appropriate blip scheme. The advantages of the proposed solution method compared to those given by other researchers are as given below.

- The proposed method is flexible. It facilitates the choice of an appropriate blip scheme, given the access requirements.
- Representation of a blip scheme is extremely compact. Compact representation of the skewing scheme is important in the interest of reducing the overhead in storing the matrix, in skewed form, into the memory modules before the computation begins.
- Sometimes, the data array is already in the memory modules, however, before using it in the next phase of computations, it may be advantageous to re-skew it. In such cases, blip schemes facilitate efficient re-skewing of the elements of the matrix.
- The blip schemes facilitate self-routing in interconnection networks, to realize the data transfer functions corresponding to the access of various subsets of the data matrix.
- The blip schemes are similar to the schemes given by Fairlong et al. The advantage of blip schemes over the XOR schemes is that it is easier to see the relationship between the access requirements and the criteria for choosing a skewing scheme. Also, address generation is simpler for a blip scheme. Furthermore, we are primarily interested in storing a matrix such that, to achieve conflict free accesses of various subsets, the interconnection network need to realize only a well known class of permutations.

Using theory developed for the blip schemes we, later, show that there are some blip schemes, called restricted blip (*r-blip*) schemes, that actually allow conflict free access of the templates of interest with one pass through either the Omega or the inverse Omega network.

The linear skewing scheme given by Lawrie [14] is comparable to the r-blip schemes, in terms of simplicity of skewing method and realization of the data transfer functions using inexpensive interconnection network. In his method, to store an $N \times N$ matrix, $2N$ memory modules are required and a $2N \times 2N$ Omega network is required to realize the data transfer functions. Compared to this, the r-blip schemes require only half as many memory modules and an $N \times N$ Omega network with some additional circuitry.

3 Preliminaries

In this section, we define the class of linear permutations and explain the notion of templates of a matrix.

In this paper, we assume that there are N processors and N memory units in the system, $N = 2^n$ with $n >= 2$. Each processor (memory unit) in the system is given an index i, $0 \leq i < N$, such that no two processors (respectively, memory units) have the same index. Therefore, if i is the index of a processor, then $i = (i_0 \ \ldots \ i_{n-1})$, an n-bit vector, such that $i = \Sigma_{x=0}^{n-1} 2^x i_x$.

3.1 Linear permutations

In what follows, we define the class of linear-complement permutations of $N = 2^n$ numbers. Let $V = \{0, 1, \ldots, N-1\}$. Also, let x be any number and y be its image under some mapping.

Definition 1 *A permutation on V is said to be a linear permutation* [21], *if there exists a non singular binary matrix $Q_{n \times n}$ such that, for every $x \in V$, its image is given by the equation, $y^T = Qx^T$.*

So, a linear permutation on V is the permutation that maps each $x \in V$ to a number such that each bit in the binary form of this number is a linear combination of the bits of x. If complement of bits is allowed, then the permutation is a linear-complement permutation. To simplify the notation, we may omit the transpose indicator for vectors, the superscript T, when there is no confusion.

Definition 2 *Let $x' = (x_0 \ \ldots \ x_{n-1} \ 1)$. A permutation on V is a linear-complement permutation ($\mathcal{LC}$) if there exists a binary matrix $P = (Q \,|\, k)$, where Q is as defined above and k is an n-bit binary vector, such that, for every $x \in V$, its image is given by the equation, $y = Px'$.*

In the literature [4, 11], the linear permutations are termed as the non-singular linear transformations of the n-dimensional vector space over the field $GF(2)$ — the field consisting of two elements: 0, and 1. A linear-complement permutation with matrix $P = (Q \mid k)$, Q an $n \times n$ boolean matrix and k some n-bit vector, has the same properties as the linear permutation corresponding to Q has.

Let e_x^T be the binary representation of the number 2^x. Then the matrix formed by the column vectors e_x, $0 \leq x < n$, is $I_n = (e_0, \ldots, e_{n-1})$. Let $Q = (q_{i,j})$ be an $n \times n$ boolean matrix and $i = (i_0, \ldots, i_{n-1})$ an n-bit boolean vector. Let q_x, $0 \leq x < n$, represent the column x of Q; so, $Q = (q_0, \ldots, q_{n-1})$. Then, $Qi = i_0 q_0 \oplus \cdots \oplus i_{n-1} q_{n-1}$.

If, in a particular data transfer, for each processor i, $0 \leq i < N$, the memory unit it accesses is given by Qi, for some non-singular boolean matrix Q, then the problem of realizing the data transfer is equivalent to the problem of realizing the corresponding linear permutation by the interconnection network.

3.2 Subsets of a matrix

A *template* T of an $N \times N$ data matrix is a set of N element positions $\{(i_x, j_x) | 0 \leq x < N\}$, with $(i_0, j_0) = (0,0)$ [23]. By the access of a template T, we mean accessing of the N elements of the matrix whose positions are given by the template. Row (T_r), column (T_c), diagonal (T_d), and square block (T_s) templates are the four important templates reported in the literature [7, 14]. These templates are defined as follows.

Definition 3

$$\begin{aligned} T_r &= \{(0,j) | 0 \leq j < N\} \\ T_c &= \{(i,0) | 0 \leq i < N\} \\ T_d &= \{(i,i) | 0 \leq i < N\} \\ T_s &= \{(i,j) | 0 \leq i,j < \sqrt{N}\} \end{aligned}$$

∎

It is clear that T_r, T_c, T_d, and T_s correspond to the positions given by, respectively, row 0, column 0, main diagonal, and the square block $S_{0,0}$ of the data matrix. The square block $S_{i,j}$ of an $N \times N$ matrix, N square of some integer, is the $\sqrt{N} \times \sqrt{N}$ sub matrix with (i,j) in the top left position.

Given, a template $T = \{(0,0), (i_x, j_x) | 1 \leq x < N\}$, the set $\{(a,b), (a \oplus i_x, b \oplus j_x) | 1 \leq x < N\}$, defines the affine template $T(a,b)$ of T. The usefulness of these concepts will be evident, once we define the blip schemes.

3.3 latin squares

Given a data matrix, the matrix obtained by replacing each element by the memory unit number, in which the element is stored, is called the *mapping* matrix. A mapping matrix such that, in each row and column, a number in the set V appears exactly once is termed as a *latin square* of order N in the literature [8, 22]. A *transversal* of a latin square of order N is the set of N positions such that no two positions are in the same row, in the same column, and have the same symbol.

Right diagonal j of a $N \times N$ matrix is the set of symbols, $\{(i, j+i \pmod{N}) \mid 0 \leq i < N-1\}$. Similarly, left diagonal j is the set of symbols, $\{(i, j-i \pmod{N}) \mid 0 \leq i < N\}$. Main diagonal of a matrix is the 0-th right diagonal, and back diagonal is the $(N-1)$-th left diagonal. A diagonal latin square is a latin square in which the main and back diagonals are transversals. A *crisscross* latin square [12] is a latin square in which all the right diagonals for even j and all the left diagonals for odd j are transversals.

4 The blip schemes

In this section, we give the blip storage scheme and discuss the criteria for having various templates free of memory conflicts. We assume that there are N memory modules and processors, and the data matrix to be stored is of size $N \times N$, $N = 2^n$ for some $n \geq 2$.

The blip storage scheme: Element (i, j) is stored in location i of the memory unit with index $i \oplus \pi(j)$, where π is some linear permutation. ▮

This storage scheme facilitates the use of the self-routing algorithms developed for the class of linear permutations in Beneš and shuffle-exchange networks [5]. The main advantage of a blip scheme is, it allows an easy understanding of the scrambled schemes, and simplifies the criteria for conflict free access to various templates.

In figure 3(a), we give the mapping matrix for storing a 16×16 data matrix in 16 memory modules. The linear permutation corresponding to the mapping is given in figure 3(b). Any blip scheme can, compactly, be represented using a boolean matrix; this is an important aspect of the blip schemes.

The storage scheme proposed by Kim and Kumar [13] is a blip scheme, with memory units renumbered. This is easy to see from their construction method. This is the very reason why address generation for their scheme requires only a set of exclusive-or gates. It can be shown that the skewing scheme proposed by Balakrishnan et al. [1] is also a member of the proposed class. Also, the schemes given by Lee [16, 17] are members of the proposed class. It is easy to see that the storage schemes developed by Fairlong et al. [9] can be obtained from this scheme. In fact, any scheme given by their method can be converted to an equivalent blip scheme by renumbering the memory modules.

For the scheme given in [13], representation of the mapping of the elements of the matrix to the memory modules requires either execution of the algorithm that gives the skewing method, or a dedicated hardware unit that generates the address for each element of the matrix. (The module number, in which an element is stored, and the location within in the module constitute the address for the element.) In our approach, the representation is in the form of n binary vectors; the operations required to generate the address of each element are bitwise exclusive-or and bitwise 'and' operations. Compact representation of a storage scheme and constant time computation of address reduce the overhead of placement of data, before the computation begins, in the memory modules.

Since a linear permutation is used to skew data, the transfer functions for the parallel access of the templates, row, column, diagonal, and square block, when free of memory conflicts, will be in the form of a linear-complement permutation. Thus, blip schemes facilitate pipelining of data movement by choosing a Beneš network [3] or a Π [27] network as the processor-memory interconnection network and using the self-routing techniques developed for these networks [5]. Sometimes, data are read into memory modules in a particular fashion; an example is reading the output of image scanners in row major order; another example is transferring data from disk to main memory modules. Also,

0	12	4	8	3	15	7	11	1	13	5	9	2	14	6	10
1	13	5	9	2	14	6	10	0	12	4	8	3	15	7	11
2	14	6	10	1	13	5	9	3	15	7	11	0	12	4	8
3	15	7	11	0	12	4	8	2	14	6	10	1	13	5	9
4	8	0	12	7	11	3	15	5	9	1	13	6	10	2	14
5	9	1	13	6	10	2	14	4	8	0	12	7	11	3	15
6	10	2	14	5	9	1	13	7	11	3	15	4	8	0	12
7	11	3	15	4	8	0	12	6	10	2	14	5	9	1	13
8	4	12	0	11	7	15	3	9	5	13	1	10	6	14	2
9	5	13	1	10	6	14	2	8	4	12	0	11	7	15	3
10	6	14	2	9	5	13	1	11	7	15	3	8	4	12	0
11	7	15	3	8	4	12	0	10	6	14	2	9	5	13	1
12	0	8	4	15	3	11	7	13	1	9	5	14	2	10	6
13	1	9	5	14	2	10	6	12	0	8	4	15	3	11	7
14	2	10	6	13	1	9	5	15	3	11	7	12	0	8	4
15	3	11	7	12	0	8	4	14	2	10	6	13	1	9	5

(a)

$$Q = \left(\begin{array}{cc|cc} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{array}\right)$$

(b)

Figure 3: A blip scheme to store a 16×16 data matrix.

sometimes, the data are in the memory but not in the desired skewed order due to previous computations. So, before the start of computations, the data may need to be re-skewed, so that during computation conflict free access of various templates is obtained. With the given storage scheme, the data movement can be pipelined with simple address computations.

4.1 Conflict free access of the templates of interest

In what follows, for each of the row, column, diagonal, and square block templates, we obtain: the criterion to be satisfied by a blip scheme to provide conflict free storage of the elements in the template, and the permutation to be realized by the interconnection network to provide conflict free access. The issue of fast address generation is discussed in § 4.2. For the remainder of this section, we assume that Q represents the boolean matrix corresponding to the linear permutation defining the blip scheme under discussion. If T is the template free of memory conflicts, then the data transfer function to be realized by the interconnection network for the parallel access of T is denoted as $f(T)$.

Suppose a template T is free of memory conflicts. Then to see which processor gets which element, we rank the elements in the templates, using their indices, in the row major order[3] and assign i-th smallest ranked element to processor i, $0 \leq i < N$. There are other possible assignments, namely, column major order, shuffled row major order, snake-like row major order, etc. But, if the transfer function for a template access (in row major order) is a linear-complement permutation, then the transfer function will still be a linear-complement permutation with any of the other assignments given above. So, in this paper, we concentrate on the access of templates in row major order.

We use the following theorem, to extend the results proved for any template to the affine templates derived from it.

Theorem 1 *If access to a template T is free of memory conflicts, then (a) access of any affine template T' obtained from T is free of memory conflicts, and (b) $(f(T))$ is a linear permutation, if and only if $f(T')$ is a linear-complement permutation.*

Proof: Since, there is one-one correspondence from T to T', part (a) of the theorem follows. The one-one to correspondence from T to T' is a linear-complement permutation. So, part (b) is follows from the definition of linear permutations. ▮

The interconnection network

In this section, we consider the use of multistage dynamic interconnection networks to realize the transfer functions that arise in accessing the templates. Though, the following discussion assumes the interconnection network to be an $N \times N$ Beneš network, it is valid for other networks, for example, the shuffle-exchange type of networks.

[3]In the row major order, element (i, j) is ranked $i \times \sqrt{N} + j$.

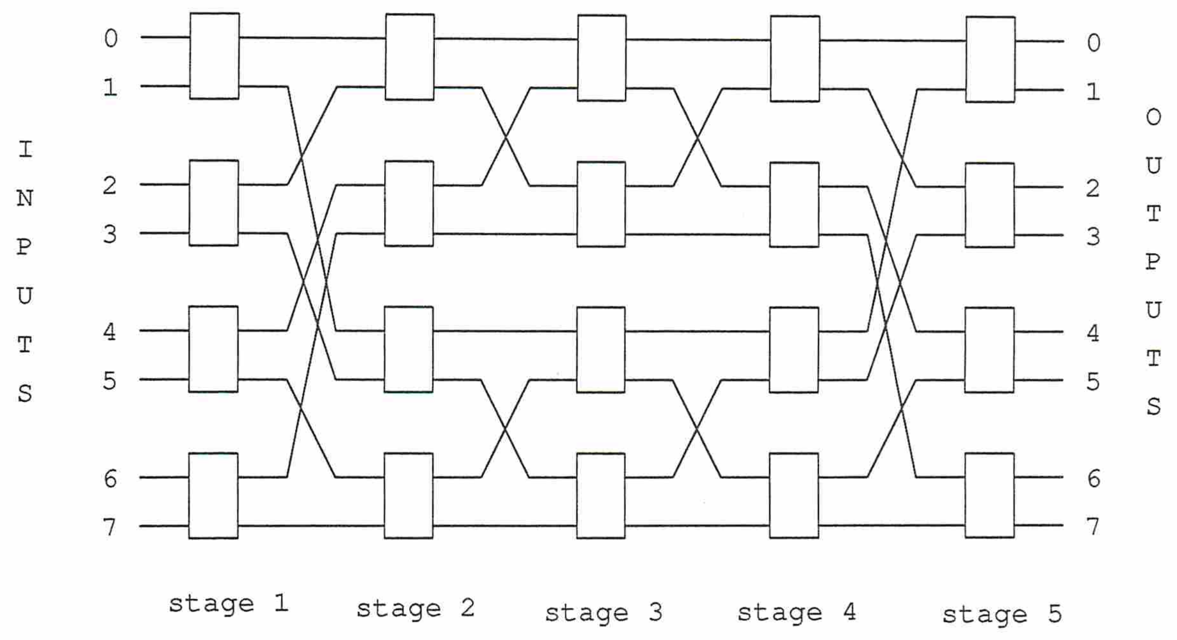

Figure 4: An 8×8 Beneš network.

An 8×8 Beneš network is shown in figure 4. In general, an $N \times N$ Beneš network consists of $2 \log N - 1$ stages of switches, each stage containing $N/2$ switches. Each switch can be set in a cross or straight position. This network can realize any permutation of inputs to outputs, by setting the switches appropriately [3]. However, to realize an arbitrary permutation, the time to set up the network takes a lot of time [25]. So, researchers have proposed methods to set up the network to realize some of the common permutations such that each switch looks at the destination addresses of its inputs and sets itself using some simple logic. Such schemes are called, self-routing schemes [18]. For the Beneš and shuffle-exchange networks, a self-routing algorithm to route any linear-complement permutation is known [5].

To use the interconnection network efficiently, we need to show that there are suitable blip schemes with transfer functions of the important templates in the class of linear permutation. Then, by the theorem 1, the access of any affine template of T requires a corresponding linear-complement permutation to be realized by the interconnection network.

4.1.1 The row and column templates

Access of the row template (i.e., the row 0), T_r, means processor j, $0 \leq j < N$, should obtain the element $(0, j)$, stored in the memory module $0 \oplus \pi(j)$, where π is the linear permutation used for the blip scheme under consideration. Since, Q is the boolean matrix corresponding to π, $\pi(j) = Qj$. So $f(T_r)$ is the linear permutation represented by Q. Access of T_c means processor i gets the element $(i, 0)$ stored in the memory unit i. Hence

$f(T_c)$ is an identity permutation. These points are summarized in the following theorem.

Theorem 2 *For any blip scheme, T_r and T_c are free of memory conflicts and $f(T_r)$ and $f(T_c)$ are linear permutations.* ∎

4.1.2 The diagonal template

Access of the diagonal template T_d (the main diagonal) means processor i obtains the element (i, i) stored in the memory module $i \oplus Qi$. Noting that,

$$i \oplus Qi = I_n i \oplus Qi = (I_n \oplus Q)i$$

we conclude that T_d is free of memory conflicts, if and only if the $Q'' = (I_n \oplus Q)$ is non-singular. Furthermore, whenever Q'' is non-singular, $f(T_d)$ is a linear permutation.

Back diagonal is the template $T_d(0, N-1)$. To access back diagonal, processor i gets the element $(i, \bar{i})$, $\bar{i} = (N-1) \oplus i$, which is stored in the memory module $i \oplus Q\bar{i} = Q\mathbf{1} \oplus i \oplus Qi$, $\mathbf{1}$ is the n-bit column vector with all 1's.

Theorem 3 *For a blip scheme with boolean matrix Q, T_d is free of memory conflicts, if and only if the boolean matrix $Q'' = Q \oplus I_n$ is non-singular. Furthermore, whenever Q'' is non-singular, $f(T_d)$ is a linear permutation given by Q''.* ∎

An example of accessing main diagonal of a 16×16 matrix with skewing method given in figure 3 is shown in figure 5. We assume that to input (left side) of the network, processors are connected, memories to the other end of the network. Input and output lines are numbered top to bottom $0, 1, \ldots, N-1$. Processor (memory) i is connected to input (output) line i. The numbers at the input side represent the module number to which the respective processor is to be connected.

4.1.3 The square block template

Let n be even, so that $\sqrt{N}$ is an integer. Also, for $i = (i_0, \ldots, i_{n-1})$, $0 \le i < N$, let $i_l = (i_0, \ldots, i_{\frac{n}{2}-1}, 0, \ldots, 0)$, and $i_u = (i_{\frac{n}{2}}, \ldots, i_{n-1}, 0, \ldots, 0)$.

Access of T_s (i.e. the elements in the square block $S_{0.0}$) means that processor i obtains the element (i_u, i_l), which is stored in the memory module $i_u \oplus Qi_l$. But,

$$\begin{aligned} Qi_l \oplus i_u &= i_0 q_0 \oplus \cdots \oplus i_{\frac{n}{2}-1} q_{\frac{n}{2}-1} \oplus i_{n/2} e_{n/2} \oplus \cdots \oplus i_{n-1} e_{n-1} \\ &= (q_0, \ldots, q_{\frac{n}{2}-1}, e_0, \ldots, e_{\frac{n}{2}-1}) i \,. \end{aligned}$$

Let $Q' = (q_0, \ldots, q_{\frac{n}{2}-1}, e_0, \ldots, e_{\frac{n}{2}-1})$. Then the processor i, $0 \le i < N$, accesses memory unit $Q'i$. If Q' is non-singular, then this defines a linear permutation.

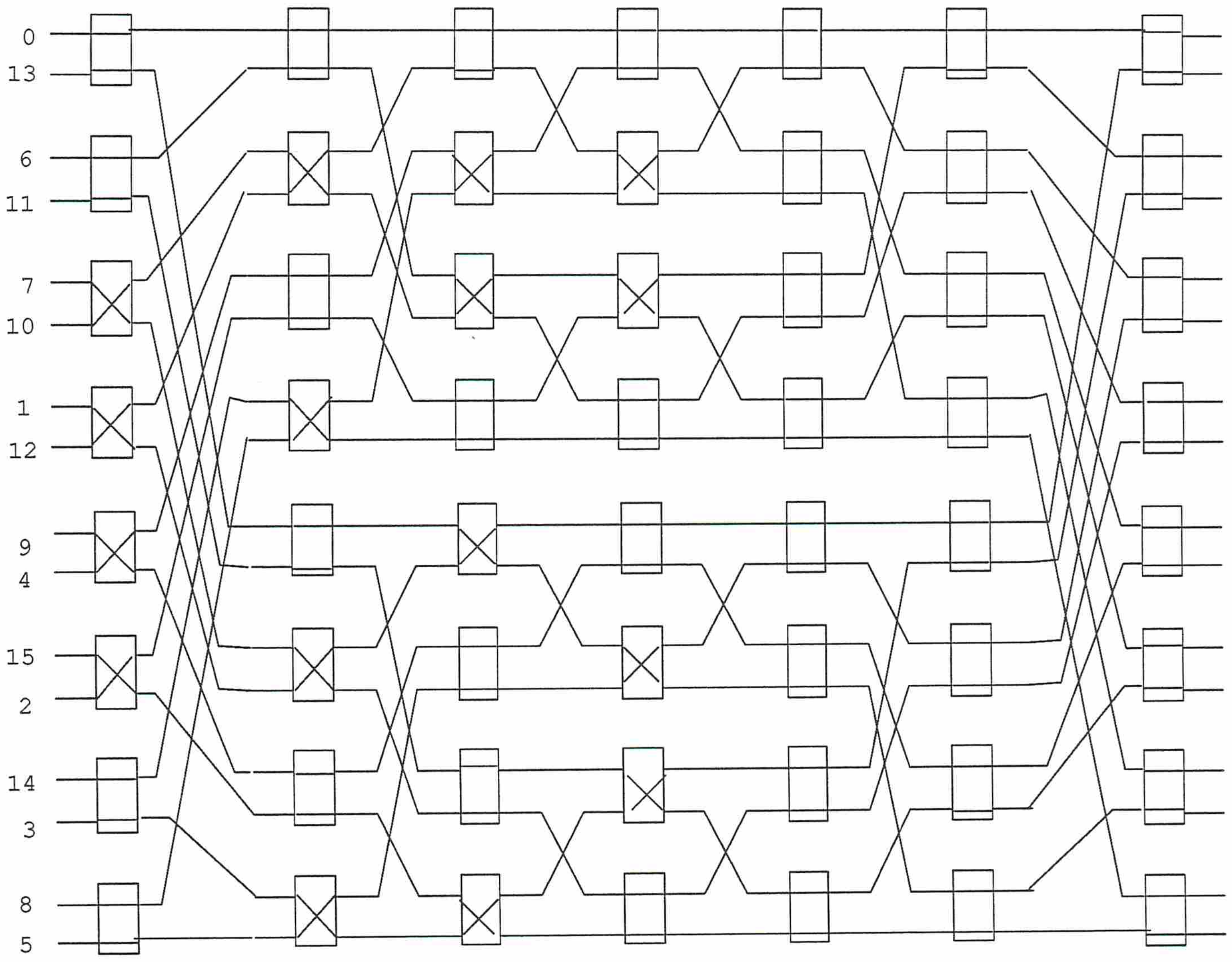

Figure 5: An example of realizing $f(T_d)$ by the Beneš network.

Theorem 4 *For a blip scheme with boolean matrix Q, T_s is free of memory conflicts, if and only if the boolean matrix Q' (obtained from Q such that, for $0 \leq i < n/2$, $q_i' = q_i$, and, for $n/2 \leq i < n$, $q_i' = e_{i-n/2}$) is non-singular. Furthermore, whenever Q' is non-singular, $f(T_s)$ is a linear permutation given by Q'.* ∎

4.1.4 The crumbled rectangles

Fairlong et al. [9] defined a crumbled rectangle of size $(2^r, 2^{n-r})$ by step of $(2^k, 2^l)$ (denoted, $T_{cr(n,r,k,l)}$) to be a template with the following set of elements:

$$\begin{array}{cccc}
(0,0) & (0,2^k) & \dots & (0,(2^r-1)2^k) \\
(2^l,0) & (2^l,2^k) & \dots & (2^l,(2^r-1)2^k) \\
 & \vdots & & \\
((2^{n-r}-1)2^l,0) & ((2^{n-r}-1)2^l,2^k) & \dots & ((2^{n-r}-1)2^l,(2^r-1)2^k)
\end{array}$$

The concept of crumbled rectangles is a generalization of that of the square block template. Extending the argument given for the case of the square block template, we can obtain the criterion for a crumbled rectangle to be conflict free. The following theorem gives this criterion, which is also discussed extensively by Fairlong et al. [9].

Theorem 5 *For a blip scheme with boolean matrix Q, the access of the crumbled rectangle $T_{cr(n,r,k,l)}$ is free of memory conflicts, if and only if the boolean matrix $Q^* = (q_k, \dots, q_{k+r-1}, e_l, \dots, e_{n-r+l-1})$ is non-singular. Furthermore, whenever Q^* is non-singular, $f(T_{cr(r,n,k,l)})$ is a linear permutation.* ∎

The discussion given by Fairlong et al. about the requirements for conflict free access to various crumbled rectangle is applicable to the blip schemes as follows. Suppose a skewing scheme is developed using their method. Then, to find the equivalent blip scheme, renumber the memory modules such that the leftmost column of the mapping matrix defined by their scheme is an identity permutation. The resultant mapping matrix corresponds to the blip scheme with linear permutation given by the permutation of topmost row of the mapping matrix.

The discussion on the chess board templates, given in [9], can be applied to the blip schemes.

4.2 Address generation

For fast access of any template of a matrix, besides having each element of the template in a distinct memory module, a fast scheme to generate the module numbers should be provided. When a blip scheme is used, for the important templates, memory unit number can be generated by each processor, independently, in constant time with only $\log N$ exclusive-or gates. This can be explained as follows.

The most flexible approach is to store the boolean matrix, corresponding to the blip scheme used, in each processor, in a fast memory. This requires $(\log N)^2$ bits of storage. Now, given row and column indices of an element, the memory module in which it is stored is computed easily. However, if only the four data templates, T_r, T_c, T_d, and T_s, are used in the course of computation, then space and time can be saved with the following approach. We assume that, in processor i, the n-bit boolean vectors Qi, $Q\bar{i}$, and Qi_l, $\bar{i}$ and i_l are as defined in § 4.1.3, are stored.

For the access of the row template, processor i has to compute Qi, which is already present in the processor's local memory. For the access of any affine row template $T_r(a,b)$, processor i needs to compute the address of the element (a,i), which is $a \oplus Qi$. With the broadcast of the vector a, each processor can compute address of the element assigned to it in parallel and in constant time using $\log N$ exclusive-or gates. In the case of an affine column template $T_c(a,b)$ (column b) processor i computes the address of $i \oplus Qb$. If Qb is broadcast, each processor can compute address in parallel.

To access the main diagonal elements, processor i has to compute $i \oplus Qi$, the address of the element (i,j); this can be done in constant time. To access the back diagonal, processor i needs to compute $i \oplus Q\bar{i}$, which can be done in constant time.

Let n be even and, for any i, i_u and i_l be as defined in § 4.1.3. For the access of the square block template (square block $S_{0,0}$) processor i has to compute the address of the element (i_u, i_l), which is given by $i_u \oplus Qi_l$. Each processor can compute this in constant time. For an affine square block template $T_s(a,b)$ (square block $S_{a',b'}$, $a' = a_u \times \sqrt{N}$ and $b' = b_u \times \sqrt{N}$), to get the memory unit number, each processor would perform the above computation and then perform exclusive-or of the result with the vector $(a,' \oplus Qb')$, which is broadcast by the control unit.

4.3 Some examples

In what follows, we take some of the schemes proposed by the researchers, give the equivalent blip schemes, and analyze the accessibility of templates with the theory developed in this paper.

Example 1:

The scheme proposed by Lee [16] corresponds to the blip scheme with

$$Q = \left(\begin{array}{c|c} 0 & I_{n/2} \\ \hline I_{n/2} & 0 \end{array}\right).$$

Since, Q satisfies theorem 4, the square blocks template is free of memory conflicts. However, $Q'' = I_n \oplus Q$ is singular, since it is of the form given below. Hence, this scheme can not provide conflict free access to the diagonal template.

$$Q'' = \left(\begin{array}{c|c} I_{n/2} & I_{n/2} \\ \hline I_{n/2} & I_{n/2} \end{array}\right).$$

Example 2:

The scheme proposed by Lee [17] corresponds to the blip scheme with $Q = (q_0, \ldots, q_{n-1})$, where $q_0 = 3$, $q_{n-1} = 1$, and, for $1 \leq i < n-1$, $q_i = e_{i+1}$. For $n = 6$, Q and Q'' are as shown below.

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \qquad Q'' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

In general, it is easy to show that Q fails to satisfy the theorem 4, hence it can not provide conflict free access of square block templates. But, Q'' is non-singular, hence, the diagonal template is free of memory conflicts.

Example 3:

Fairlong et al. [9], discussed a scheme to store 16×16 matrix as an example of the method proposed by them. This scheme corresponds to the blip scheme with

$$Q = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

It is easy to see that Q satisfies theorem 4, hence allows conflict free access of the square block templates. But, Q'' is singular, hence the diagonal has memory conflicts.

Example 4:

For the case $N = 16$, the storage scheme using the Kim and Kumar's method [13] corresponds to the blip scheme (shown in figure 3) with

$$Q = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Here, Q satisfies theorems 3 and 4, hence it provides conflict free access of both diagonal and square block templates. In general, one can show that their construction method yields a blip scheme with conflict free access of diagonal and square block templates.

4.4 Conflict free access of some composite templates

Sometimes, it may be desirable to access subsets of the data that are not templates. However, they may be regular and nice and have the same form as that of some template.

We call such templates as composite templates. In this section, we consider two composite templates: square blocks $\{S_{i,j} \mid i_l = 0 or j_l = 0\}$ and diagonal other than main and back diagonals.

One can extend the address generation methods discussed in § 4.2 to these composite templates, by storing only the three vectors Qi, Qi_l, and $Q\bar{i}$ in processor i, $0 \leq i < N$. The trnasfer functions are not in the class of linear-complement permutations. However, some efficient routing techniques can be devised using using the known routing techniques. Any composite square block can be accessed using a Beneš or a Π network with the self-routing techniques given for linear-complement permutations in at most two passes through the network. Any off diagonal that is free of memory conflicts will have the composite permutation composed of a linear-complement permutation followed by a cyclic shift permutation as its transfer function. Clearly, more work is warranted on the realization of transfer functions for the parallel access of composite templates.

4.4.1 The composite square block templates

Kim and Kumar's method is such that any square block $S_{i,j}$ is conflict free whenever $i \equiv 0 \pmod{\sqrt{N}}$, or $j \equiv 0 \pmod{\sqrt{N}}$. While this is difficult to understand by looking at their construction method, it is very easy to obtain the necessary and sufficient conditions for the property to be true when their method is viewed as a blip scheme.

The set of positions in square block $S_{i,j}$, $i \equiv 0 \pmod{\sqrt{N}}$, are obtained form the set of positions in the square block $S_{0,j}$ by exclusive-or operation of $(i, 0)$ on each position in the square block $S_{0,j}$. So, if the set of square block $S_{0,j}$, $0 \leq j \leq (N - \sqrt{N})$, are conflict free, then any square block $S_{i,j}$, $i \equiv 0 \pmod{\sqrt{N}}$ is conflict free. Similar property holds for the square blocks $S_{i,j}$, $j \equiv 0 \pmod{\sqrt{N}}$.

First we obtain the criterion for conflict free access of any square block $S_{0,j}$. We know that square block $S_{0,0}$ is conflict free when the theorem 4 is satisfied. For, $S_{0,1}$ to be conflict free, we need that symbols in the rightmost column of $S_{0,1}$ (consists of the topmost $\sqrt{N}$ elements of the column $\sqrt{N}$ of the mapping matrix) be a permutation of the the symbols in the leftmost column of $S_{0,0}$. But, the topmost $\sqrt{N}$ entries in that column the mapping matrix are of the form $i \oplus q_{n/2}$, $0 \leq i < \sqrt{N}$. So, for conflict free access of $S_{0,1}$, the vector $q_{n/2}$ should have value less than $\sqrt{N}$. This argument can be continued to show that each of the vectors $q_{n/2}, \ldots, q_{n-1}$ should be of value less than $\sqrt{N}$, if each of the square blocks $S_{0,j}$, $0 \leq j \leq (N - \sqrt{N})$ are to be conflict free. Similar argument can be applied to get the criterion for the square block $S_{i,0}$, $0 \leq i \leq (N - \sqrt{N})$, to be conflict free. In this case, it is required that each of the vectors $q_0, \ldots, q_{\frac{n}{2}-1}$ should be of value greater than or equal to $\sqrt{N}$. Thus the the criterion for conflict free access of any square block S_{ij}, $i \equiv 0 \pmod{\sqrt{N}}$, or $j \equiv 0 \pmod{\sqrt{N}}$, is as given in the following theorem.

Theorem 6 *Let Q be the boolean matrix corresponding to a storage scheme. Then, the storage scheme has the property that any square block S_{ij} such that $i \equiv 0 \pmod{\sqrt{N}}$, or*

$j \equiv 0 \pmod{\sqrt{N}}$, is conflict free, if and only if Q can be partitioned such that

$$Q = \left(\begin{array}{c|c} 0 & Q_2 \\ \hline Q_1 & 0 \end{array} \right)$$

where Q_1, Q_2 are non-singular $\frac{n}{2} \times \frac{n}{2}$ boolean matrices. ∎

The boolean matrix in figure 3(b) can be partitioned as given in the above theorem. Hence, in the mapping matrix, figure 3(a), any square block S_{ij}, $i \equiv 0 \pmod 4$, or $j \equiv 0 \pmod 4$, is conflict free.

4.4.2 The off diagonals

In a matrix of size $N \times N$, there are N right diagonals and N left diagonals. Hedayat [10] showed the non-existence a latin square of even order such that every right diagonal and left diagonal is accessible. As an alternative, Hwang [12] defined crisscross latin squares. He showed that crisscross latin squares exist whenever N is a multiple of 4, and gave a method to construct such latin squares. We will modify the procedure given by Hwang to obtain crisscross latin squares that correspond to the blip schemes.

In the original method given by Hwang, he constructed a set of N positions in which 0 would occur. Form this he obtained the set of positions in which the numbers $1, 2, \ldots, N-1$ occur by simple arithmetic operation on the set of positions obtained for 0. Due to the modulo N arithmetic, the rows and columns of such a latin square are not linear permutations. See figure 6(a) for an 8×8 crisscross latin square obtained by Hwang's procedure. For the case N a power of 2, a blip scheme can be devised to obtain a crisscross latin square. First obtain the positions of the symbol 0 as given by Hwang. This can be shown to be some linear permutation π due to the construction method. (For the 8×8 case, the boolean matrix in figure 6(c) corresponds to this linear permutation.) Now, the crisscross latin square is given by the blip scheme whose linear permutation is π. The 8×8 crisscross latin square in figure 6(b) is obtained by the modified procedure. It can be shown that this blip scheme gives conflict free access to any even (odd) right (left) diagonal of the matrix. The transfer function for the diagonals, other than the main and back diagonals, is a permutation of the form, a linear-complement permutation followed by a cyclic shift operation.

5 Some special blip schemes

We now devise some special blip schemes called r-blip (restricted blip) schemes for which $f(T_x)$, $x \in \{r, c, d, s\}$, can be realized by an $N \times N$ Omega (Ω_N) or an inverse Omega (Ω_N^{-1}) network. If Ω_N (Ω_N^{-1}) can realize $f(T_x)$, we denote this (Lawrie's notation) as $\Omega_N \uparrow f(T_x)$ ($\Omega_N^{-1} \uparrow f(T_x)$).

0	2	4	6	7	1	3	5
1	7	5	3	2	0	6	4
2	4	6	0	1	3	5	7
3	1	7	5	4	2	0	6
4	6	0	2	3	5	7	1
5	3	1	7	6	4	2	0
6	0	2	4	5	7	1	3
7	5	3	1	0	6	4	2

(a)

0	6	4	2	7	1	3	5
1	7	5	3	6	0	2	4
2	4	6	0	5	3	1	7
3	5	7	1	4	2	0	6
4	2	0	6	3	5	7	1
5	3	1	7	2	4	6	0
6	0	2	4	1	7	5	3
7	1	3	5	0	6	4	2

(b)

$$Q = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

(c)

Figure 6: Two examples of crisscross latin squares

We organize this section as follows. First, we present the Omega network with some additional circuitry. Next, we characterize the linear permutations passable in the Omega and the inverse Omega networks. Then, we discuss an r-blip scheme that can be used with the Omega network and some additional circuitry.

The interconnection network

In this section, we assume that the Omega network is available as the interconnection network between processor and memory units. In the process of realizing the data transfer functions, we may use the interconnection network as an Omega network, when processors are connected to the left end of the network and memory units to the right end of the network, or as an inverse Omega network, when processors are connected to the right end of the network and memory units to the left end of the network. This can be achieved with N additional 2×2 switches. By setting these N switches to straight position, the interconnection network between processors and memory units is the Omega network; by setting these switches to cross position, the inverse Omega network is available. This can easily be done with a 1-bit control line by the central control unit. An 8 processor system with this interconnection network is shown in figure 7.

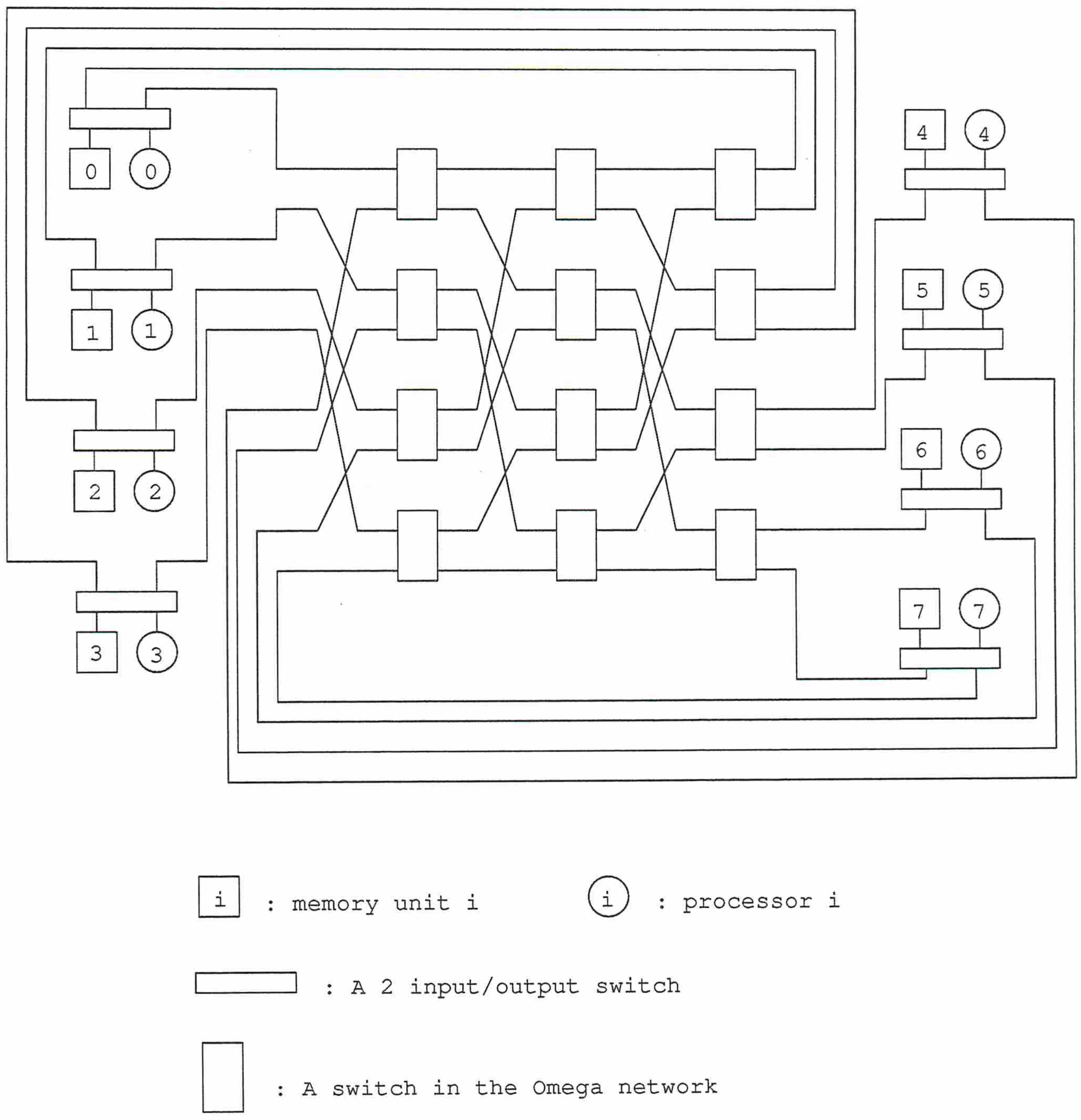

Figure 7: An 8 processor system with the modified Omega network.

5.1 Routing linear permutations in the Omega and the inverse Omega networks

Characterization of Ω_N (Ω_N^{-1}) passable linear permutations enables us to devise the r-blip schemes.

Let us define a lower triangular linear permutation to be a linear permutation with boolean matrix a non-singular lower triangular matrix. Similarly upper triangular linear permutation is defined.

Pease [21] proved that a linear permutation is realizable in an indirect binary n-cube network (identical to the inverse Omega network [20]), if and only if it is composed of two linear permutations: the upper triangular permutation followed by lower triangular permutation. So, a linear permutation is passable in an Omega network, if and only if it can be decomposed into a lower triangular linear permutation followed by an upper triangular permutation. The theorem below gives an equivalent criterion to characterize the Omega passable linear permutations. This is a special case of the more general property proved by Varma [24] to count the number of passes required to realize a linear permutation in Omega.

Definition 4 *For a boolean matrix $Q = (q_{i,j})_{n\times n}$, the expanding sub-matrix k, $0 \leq k < n$, of Q is defined as follows.*

$$\begin{pmatrix} q_{0,0} & \cdots & q_{0,k} \\ \vdots & & \\ q_{k,0} & \cdots & q_{k,k} \end{pmatrix}$$

Definition 5 *Similarly, for a boolean matrix $Q = (q_{i,j})_{n\times n}$, the contracting sub matrix k, $0 \leq k < n$, of Q is defined as follows.*

$$\begin{pmatrix} q_{k,k} & \cdots & q_{k,n-1} \\ \vdots & & \\ q_{n-1,k} & \cdots & q_{n-1,n-1} \end{pmatrix}$$

Lemma 1 *A boolean matrix $Q = (q_{i,j})_{n\times n}$ can be decomposed into non-singular upper triangular and lower triangular boolean matrices without pivoting, if and only if each of the expanding sub-matrices is non-singular.*

Proof: First, assume that Q can factored without pivoting. We can factor Q into UL, using the modified form of Gauss's algorithm [21] such that, in step i, $0 \leq i < n$, the first i rows are reduced to lower triangular. For this we can define elementary column operations similar to the elementary row operations. Note that here modulo 2 arithmetic is used. Now, at each step of the Gauss's algorithm, the pivot element is non-zero by assumption. The expanding sub matrix with bottom right corner element as the pivot element is a lower triangular matrix, which is non-singular. This lower triangular matrix is obtained from the corresponding expanding sub matrix of Q with some elementary column operations. Since, the elementary row or column operations do not change the non-singularity of a matrix, we conclude that the expanding sub matrix of Q is non-singular as well.

If each of the expanding sub matrices of Q is non-singular, then it is easy to see that Q can be factored without pivoting. ▮

Theorem 7 *A linear permutation with boolean matrix Q is passable in the inverse Omega network, if and only if each of the expanding sub-matrices of Q is non-singular.*

Proof: Directly follows from the above lemma and the Pease's result discussed above. ▮

Let ρ denote the bit-reversal permutation, ω^{-1} be an Ω_N^{-1} passable permutation. Then, the corresponding Ω_N passable permutation (ω) is given by the following theorem, proved by Parker [20].

Theorem 8 $\omega = \rho\omega^{-1}\rho$. ▮

Corollary 1 *A linear permutation given by the boolean matrix Q is passable in the inverse Omega network, if and only if each of contracting the sub-matrices of Q is non-singular.*

Proof: The result follows from the above two theorems. ▮

In addition to these, we state the following theorem for the Omega network. Similar result holds for the inverse Omega network.

Theorem 9 *If a permutation π is passable in the Omega network, then any permutation obtained from π by complementing some bits is also passable in the Omega network.* ▮

If a template T has the data transfer function $f(T)$, then any affine template derived form T has a data transfer function that is obtained from $f(T)$ with some bits complemented. So, if we show that, for a template T, $f(T)$ is passable in the Omega or the inverse Omega network, then $f(T(a,b))$, $T(a,b)$ is an affine template of T, is also passable in the network.

5.2 An r-blip scheme

Let L_x (U_x) be the $x \times x$ non-singular lower (upper) triangular boolean matrix, with all entries below (above) the diagonal as 1. Let $L''_x = L_x \oplus I_x$. U''_x is defined similarly. Let ρ_x be the $x \times x$ boolean matrix corresponding to the *bit reversal* permutation on 2^x numbers. As an example, L_4, U_4, and ρ_4 are given below.

$$L_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad U_x = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \rho_x = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

For any blip scheme, the transfer function of the column template $f(T_c)$ is always an identity permutation.

Theorem 10 *For any blip scheme, $\Omega_N \uparrow f(T_c)$ and $\Omega_N^{-1} \uparrow f(T_c)$.* ∎

Now, consider a blip scheme defined by the following boolean matrix Γ_n. Assume that N is an even power of 2. We will show that this is an r-blip scheme.

$$\Gamma_n = \left(\begin{array}{c|c} L_{n/2} & I_{n/2} \\ \hline L_{n/2} & 0 \end{array}\right)$$

For $N = 16$, Γ_4 is,

$$\left(\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{array}\right)$$

We note the following.

Lemma 2 *For the r-blip scheme given by Γ_n, $f(T_r)$ and $f(T_s)$ are the same.*

Proof: Result is immediate by the application of theorem 4. ∎

Lemma 3 *Each of the expanding sub-matrices of Γ_n is non-singular.*

Proof: For any i, $0 \leq i < n/2$, the expanding sub matrix i of Γ_n is a lower triangular matrix, hence non-singular. The expanding sub matrix $n/2$ is of the following form.

$$\left(\begin{array}{cccc|c} & & & & 1 \\ & L_{n/2} & & & 0 \\ & & & & \vdots \\ & & & & 0 \\ \hline 1 & 0 & \dots & 0 & 0 \end{array}\right)$$

In this, the first row can not be a linear combination of any of the other rows, since it has a 1 in position $(0, n/2)$ and no other row has a 1 in this column. The remaining rows, considering the first $n/2$ columns, form an $\frac{n}{2} \times \frac{n}{2}$ boolean matrix that is same as $L_{n/2}$ with the rows cyclically shifted upward by 1 step. So, the expanding sub matrix $n/2$ is non-singular. This argument can be given to show that any expanding sub matrix i, $n/2 \leq i < n$, is non-singular. ▮

Hence, for the blip scheme defined by Γ_n, $\Omega_N^{-1} \uparrow \{f(T_r), f(T_s)\}$.

$\Gamma''_n = \Gamma_n \oplus I_n$ defines the permutation $f(T_d)$. As an example, Γ''_4 is given below.

$$\left(\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right)$$

In the general case,

$$\Gamma''_n = \left(\begin{array}{c|c} L''_{n/2} & I_{n/2} \\ \hline L_{n/2} & I_{n/2} \end{array}\right)$$

Lemma 4 *Γ''_n, n even, is non-singular.*

Proof: Use the following transformation on Γ''_n. Replace row i, $0 \leq i < n/2$, of Γ''_n with the sum of rows i and $i + n/2$. The resulting matrix is as given below.

$$\left(\begin{array}{c|c} I_{n/2} & 0 \\ \hline L_{n/2} & I_{n/2} \end{array}\right)$$

It is clear that, this is non-singular. Hence, Γ''_n is non-singular. ▮

Lemma 5 *Each of the contracting sub matrices of Γ''_n is non-singular.*

Proof: Consider the contracting sub matrix k, $0 \leq k < n/2$. This will be of the following form.

$$\left(\begin{array}{c|c|c} L''_{n/2-k} & 0 & I_{n/2-k} \\ \hline 0 & I_k & 0 \\ \hline L_{n/2-k} & 0 & I_{n/2-k} \end{array}\right)$$

It is clear that the above matrix is non-singular, if the sub matrix obtained from it by deleting the middle k columns and k rows is non-singular. Since the sub matrix so obtained is of the form Γ''_{n-2k}, by the above lemma, it is non-singular. Hence the contracting sub matrix k of Γ''_n is non-singular, when $0 \leq k < n/2$. For $n/2 \leq k < n$, the contracting sub matrix k of Γ''_n is simply an identity matrix. Hence the lemma holds. ▮

Hence, for the blip scheme defined by Γ_n, $\Omega_N \uparrow \{f(T_d)\}$.

Theorem 11 *Γ_n defines an r-blip storage scheme.* ∎

Another r-blip scheme similar to the above has its boolean matrix as follows.

$$\left(\begin{array}{c|c} U_{n/2} & I_{n/2} \\ \hline U_{n/2} & 0 \end{array}\right)$$

We note that, for the blip schemes proposed by Balakrishnan et al. [1], Lee [16, 17], Kim and Kumar [13], and the example scheme given by Fairlong et al. [9], neither Ω_N nor Ω_N^{-1} network can realize $f(T_r)$. It is interesting to note that the first proposed blip scheme, given by Batcher [2], which is called the flip scheme by him, has I_n as the boolean matrix. So, either Ω_N or Ω_N^{-1} can be used to realize $f(T_r)$. However, this scheme is too simple, and does not allow parallel access of the templates T_d and T_s.

An interesting non r-blip scheme

The following blip scheme gives conflict free access to many off diagonals that are not affine diagonal templates (see § 4.4.2). However, with this scheme, the square block template is not free of memory conflicts. So, this scheme is useful when conflict free access of square block template is not needed. The boolean matrix, Δ_n, for this scheme is such that $\delta_0 = N-2$, $\delta_{n-1} = N-1$ and, for all other i, $\delta_i = e_{i+1}$. Here, δ_i indicates the column i of Δ_n. Given below is Δ_5.

$$\Delta_5 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

It can be shown that, the contracting sub matrices of Δ_n are non-singular. The linear permutation corresponding to the access of the diagonal is given by $\Delta_n'' = \Delta_n \oplus I_n$. Δ_5'' is as follows.

$$\Delta_5'' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

In general, except for $k = n-1$, each of the expanding sub-matrices of Δ'' is a non-singular lower triangular matrix. And, Δ_n'' itself is non-singular. We summarize this in the following theorem.

Theorem 12 *The blip scheme defined by Δ_n gives conflict free access of rows and columns by operating the interconnection network as the Omega network and conflict free access of diagonal template by operating the interconnection network as the inverse Omega network.* ∎

6 Other applications and extensions

In this section, we briefly discuss other applications of the blip schemes.

6.1 Use of blip schemes for band matrices

Often, matrices to be stored are sparse and non-zero entries of the matrix are in the diagonals around the main diagonal. Such matrices are called the band matrices. A matrix of band b, b odd, is a sparse matrix with non-zero entries only along the right diagonals $0, 1, \ldots, (b-1)/2, (N-(b-1)/2), \ldots, (N-1)$. It is wasteful of resources to store and treat them as dense matrices. So, it is desirable to store the band matrices using diagonals. Furthermore, often computations with band matrices involve simultaneous access and computation of elements in a diagonal. The proposed method can be used in this situation to get conflict free access to the diagonals of the matrix as follows. Let the size of band of the matrix be b. Then, we can treat the problem of storing this matrix as that of storing a matrix of size $b \times N$ obtained as follows. Right diagonal j, $0 \leq j \leq (b-1)/2$, is row j, and right diagonal $(N-(b-1)/2)+j$, $0 \leq j < (b-1)/2$, is row $(b+1)/2+j$ in the $b \times N$ matrix. Now, when this matrix is stored in the memory using a linear permutation scheme, conflict free access to any row (that is, any diagonal of the original sparse matrix) is obtained.

6.2 Use of blip schemes in SIMD distributed memory systems

The theory of blip schemes developed in this paper, can be applied with no changes to some of the popular SIMD multicomputer systems such as the hypercube the mesh connected computer, and the shuffle-exchange computer. In this case, the storage problem is to store a matrix of data, without duplication, in local memories of processors such that the data movements among processors are some well-defined transfer functions. The only issue to be considered here is, how to realize $f(T)$ by the interconnection network among processors. One approach is to simulate an appropriate multistage interconnection network and use the self-routing algorithms developed for that multistage network. Another approach, relevant for the hypercube and the mesh computers, is to use the self-routing techniques developed for these computers to realize $f(T)$ for the parallel access of the template T. In [6] self-routing algorithms to realize the the class of linear-complement permutations in the boolean n-cube and the mesh connected computers are developed.

6.3 Use of blip schemes for the vector processors

Norton and Melton [19] discuss a storage scheme for the vector processors. Their scheme guarantees conflict free access to any power of 2 stride. It can be seen that their class also belongs to the class of schemes proposed here. The criterion to get conflict free access to a power of 2 stride is obtained by considering the appropriate crumbled rectangle. Whenever, such a stride is free of memory conflicts, the transfer function to be realized by the interconnection network is a linear-complement permutation.

6.4 Use of blip schemes to store matrices of larger size

For the case when the data matrix is of size $m \times m$, $m > N$ and a power of 2, partition the matrix in terms of square blocks of size $N \times N$. Then, one can use the storage method discussed earlier to skew the elements in the $N \times N$ square block at $(0, 0)$ of the matrix. Other square blocks of the matrix can be skewed using a linear-complement permutation with the same Q as that used in the skew of the square block at $(0, 0)$. Then, one can describe the skew of the square blocks using a linear permutation. By looking at the access requirements, one can choose a linear permutation to skew the square blocks, and a linear permutation to skew the elements at square block $(0, 0)$. This will completely define the skew of the $m \times m$ array. Then the issues related to address generation and routing in interconnection networks are similar to the case $m = N$.

6.5 Use of the blip schemes for storing multi-dimensional data arrays

There are two approaches to apply the theory of blip schemes to devise storage schemes for multidimensional matrices. In the first approach, a multidimensional matrix can be converted into a 2-dimensional matrix, which is very much like converting a 2-dimensional matrix into a linear array by row major ordering or some similar ordering. Then, the approach given for the case of larger size matrices can be used to devise an appropriate storage scheme. The other approach is to use that suggested by Fairlong et al., namely, skew on each dimension of the matrix with the modification that for no skewing is used for the first dimension. While this seems to be more flexible, it has the disadvantage that it complicates the criteria for conflict free access of various templates.

7 Summary and conclusions

In this paper, we considered a class of scrambled skewing schemes. The method proposed has the advantage of: (a) facilitating simple control of inexpensive interconnection networks, in accessing various subsets of a matrix, (b) compact representation of the storage scheme, and (c) simple address generation methods. These aspects are crucial for the use of a storage scheme. We showed that a multistage interconnection network such as the Beneš network or the Omega network can be used to realize the data transfer functions that arise during the parallel access of various templates of the matrix. Compact representation and simple address generation of a skewing scheme obviate the need to store the complete mapping matrix in the memory, and reduce the overhead of placing data in the memory, before the computation begins. Another advantage of the proposed method is it facilitates efficient re-skewing of data between different phases of computation; re-skewing may be necessary to ensure that, in each computation phase, the required templates are conflict free and allow self-routing methods to set up the interconnection network. The treatment given here shows that various efforts by researchers become special cases of the proposed method.

We later devised, using the theory developed, some special schemes that allow parallel access of rows, columns, main diagonal, back diagonal, and square blocks of a matrix using the well known Omega network with some additional hardware. Compared to the Lawrie's linear skewing method, these special schemes are better since parallel access of data is achieved using half as many memory modules and an interconnection network that is approximately half as big as that to be used in the Lawrie's scheme. An additional advantage of the proposed schemes is, simpler and faster address generation.

This approach is very promising. Further work in this direction would be to develop systematic procedures to obtain the linear permutation schemes that allow conflict free access of the templates of interest. Also, one can analyze the transfer functions for subsets that are regular but not affine templates and investigate the methods to realize them using an Omega or a similar network.

References

[1] M. Balakrishnan, R. Jain, and C. S. Raghavendra. On array storage for conflict-free memory access for parallel processors. In *Int'l Conf. on Parallel Processing*, pages 103–107, 1988.

[2] K. E. Batcher. The multidimensional access memory in STARAN. *IEEE Trans. on Computers*, c-26(2):174–177, 1977.

[3] V. E. Beneš. *Mathematical theory of connecting networks and telephone traffic.* Academic Press, 1965.

[4] G. Birhkoff and S. MacLane. *A survey of modern algebra.* Macmillan, 4 edition, 1977.

[5] R. Boppana and C. S. Raghavendra. On self–routing in Beneš and $(2n - 1)$-stage shuffle exchange networks. In *Int'l Conf. on Parallel Processing*, pages 196–200, 1988.

[6] R. Boppana and C. S. Raghavendra. Optimal self-routing of linear-complement permutations in hypercubes. *Tech. Report*, EE-systems, USC, 1989.

[7] P. Budnik and D. J. Kuck. The oorganization and use of parallel memories. *IEEE Trans. on Computers*, c-20(12):1566–1569, 1971.

[8] J. Denes and A. D. Keedwell. *Latin Squares and Their Applications.* Academic Press, 1974.

[9] J. M. Fairlong, W. Jalby, and J. Lenfant. Xor-schemes: A flexible data organization in parallel memories. In *Int'l Conf. on Parallel Processing*, pages 276–283, 1985.

[10] A. Hedayat. A complete solution to the existence and nonexistence of knut vik designs and orthogoanl knut vik designs. *Journal of Combinatorial Theory, Series A*, 22:331–337, 1977.

[11] K. Hoffman and R. Kunze. *Linear Algebra.* Prentice-Hall, 2 edition, 1971.

[12] F. K. Hwang. Crisscross latin squares. *Journal of Combinatorial Theory, Series A*, 27:371–375, 1979.

[13] K. Kim and V. K. P. Kumar. Perfect latin squares and parallel array access. In *Int'l Symp. on Comput. Arch.*, pages 372–379, 1989.

[14] D. H. Lawrie. Access and Alignement of Data in an Array Processor. *IEEE Trans. on Computers*, c–24(12), 1975.

[15] D. H. Lawrie and C. R. Vora. The prime memory system for array access. *IEEE Trans. on Computers*, c-31(5):435–442, 1982.

[16] D. Lee. Scrambled storage for parallel memory systems. In *Int'l Symp. on Comput. Arch.*, pages 232–239, 1988.

[17] D.-L. Lee. On access and alignment of data in a parallel processor. *Information Processing Letters*, 33(1):11–14, 1989/90.

[18] D. Nassimi and S. Sahni. A self–routing Beneš network and parallel permutation algorithms. *IEEE Trans. on Computers*, c–30(5), 1981.

[19] A. Norton and E. Melton. A class of boolean linear transformations for conflict free power-of-two stride access. In *Int'l Conf. on Parallel Processing*, pages 247–254, 1987.

[20] D. S. Parker. Notes on shuffle/exchange-type switching networks. *IEEE Trans. on Computers*, c-29(3):213–222, 1980.

[21] M. C. Pease, III. The indirect binary n–cube microprocessor array. *IEEE Trans. on Computers*, c–26(5), 1977.

[22] H. J. Ryser. *Combinatorial mathematics.* The Math. Assoc. of America, 1963.

[23] H. D. Shapiro. Theoretical limitations on the efficient use of parallel memories. *IEEE Trans. on Computers*, c-27(5):421–428, 1978.

[24] A. Varma. . *Personal communication*, 1989.

[25] A. Waksman. A permutation network. *J. Assoc. Comput. for Mach.*, 15(1), 1968.

[26] H. A. G. Wijshoff and J. V. Leeuwen. The structure of periodic storage schemes for parallel memories. *IEEE Trans. on Computers*, c-34(6):501–505, 1985.

[27] P.-C. Yew and D. H. Lawrie. An easily controlled network for frequently used permutations. *IEEE Trans. on Computers*, c–30(4), 1981.