On Methods for Fast and Efficient Parallel Memory Access[1]

C. S. Raghavendra and Rajendra Boppana
Dept. of Electrical Engineering–Systems
University of Southern California, Los Angeles, CA 90089–0781

Abstract

In this paper, we present schemes to store data arrays in parallel memory systems. The salient point of these schemes is, various templates of data can be accessed with maximum concurrency using a well known interconnection network, the Omega network, with some additional hardware. 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 Butterlfy 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.

1 Introduction

In SIMD multiprocessors, access to shared data in the memory units plays an important role in the overall performance. For specific problems, knowing the data access patterns by processors, one can allocate data to memory units 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[2]. This problem of arranging data in the memory units 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 units so that various portions of it can be accessed with high parallelism.

Consider an SIMD multiprocessor system with N processors, N memory units, a centralized control unit, and an interconnection network between processors and memory units [12]. 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 units such that various N-(element)subsets (also called, *templates*) of the matrix are retrieved from memory units without memory and interconnection network conflicts.

In figure 1, 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 1(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 1(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 1(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 the column indices of a data element, computation of its address — the mem-

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

[2]If a memory unit 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.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

(a): The data matrix A

0	1	2	3
0	1	2	3
0	1	2	3
0	1	2	3

(b): Mapping matrix 1

0	3	1	2
1	2	0	3
2	1	3	0
3	0	2	1

(c): Mapping matrix 2

Figure 1: Two methods of storing a 4 x 4 data matrix

ory unit in which the element is stored and location within the unit — 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 units, during the parallel access of data.
- Representation of the scheme should be compact so that overheads are minimized.
- Efficient re-skewing of data between phases of computation, when necessary, should be allowed.

A large number of researchers [1, 2, 8, 9, 13, 14, 15, 16, 17, 20, 22] 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.

A generalized treatment of the storage schemes is given in [6], where a class of storage schemes called the *b*itwise *l*inear *p*ermutation (blip) schemes is defined. An important feature of these schemes is, the data transfer functions[3] 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.

[3]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.

In this paper, we present some special blip schemes that provides conflict free access of rows, columns, main and back diagonals, and square blocks of data arrays using only an Omega network with some additional hardware. Hence, the solution presented can be used in the 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 explain the notation used in this paper. In section 3, we give the blip storage schemes and summarize the known results for these schemes. In section 4, we give some special blip schemes for which the data transfer functions in the parallel access of templates of a matrix are realized by an Omega or an inverse Omega network. These three sections deal with the problem of storing an $N \times N$ matrix in N memory units. In sectioin 5, we discuss the problem of storing larger matrices, and multidimensional matrices. We conclude the paper with a discussion and directions for further research.

2 Preliminaries

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

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

2.1 Linear permutations

Let $V = \{0, 1, \ldots, N-1\}$. A linear permutation on V is a permutation that maps each $i \in V$ to a number $j \in V$ such that each bit in the binary form of j is a linear combination of the bits of i. (Modulo 2 addition is used.)

Definition 1 *A permutation on V is said to be a linear permutation* [19], *if there exists a non singular $n \times n$ binary matrix Q such that, for every $i \in V$, its image j is given by the equation $(j_0, \ldots, j_{n-1})^T = Q(i_0, \ldots, i_{n-1})^T$.* ∎

If complement of bits is allowed, then the permutation is a linear-complement permutation. Formally, this is defined as follows.

Definition 2 *A permutation on V is a linear-complement permutation ($\mathcal{LC}$), if there exists a binary matrix $P = (Q \mid k)$, where Q is as defined above and k is an n-bit column vector, such that, for*

every $i \in V$, its image j is given by the equation $(j_0, \ldots, j_{n-1})^T = P(i_0, \ldots, i_{n-1}, 1)^T$. ∎

In literature [4, 11], 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.

If $i = (i_{n-1}, \ldots, i_0)$, then $\bar{i} = N - 1 \oplus i$, $i_l = (0, \ldots, 0, i_{\frac{n}{2}-1}, \ldots, i_0)$, and $i_u = (0, \ldots, 0, i_{n-1}, \ldots, i_{n/2})$. We denote the column vector $(i_0, \ldots, i_{n-1})^T$ by $\tilde{i}$; and $\tilde{\tilde{i}} = i$. The $n \times n$ identity matrix is given by $I_n = (e_0, \ldots, e_{n-1})$, where $\tilde{e}_x$ is the row vector for the binary representation of 2^x. The n columns of Q, defined above, are denoted as $q_0, \ldots, q_{n-1}$; so, $Q = (q_0, \ldots, q_{n-1})$. Often, given an $i \in V$ and an $n \times n$ boolean matrix Q, we need to compute $\widetilde{Q\tilde{i}}$; to simplify the notation, we denote this as Qi.

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 of the system.

2.2 Subsets of a matrix

A *template* T of an $N \times N$ data matrix is a set of N element positions $\{(\lambda_x, \mu_x) \mid 0 \leq x < N\}$, with $(\alpha_0, \mu_0) = (0, 0)$ [20]. 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 [8, 14]. These templates are defined as follows.

Definition 3 $T_r = \{(0, j) | 0 \leq j < N\}$
$T_c = \{(i, 0) | 0 \leq i < N\}$
$T_d = \{(i, i) | 0 \leq i < N\}$
$T_s = \{(i, j) | 0 \leq i, j < \sqrt{N}\}$ ∎

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 $\sqrt{N} \times \sqrt{N}$ submatrix with (i, j) in the top left position.

Given, a template $T = \{(0, 0), (\lambda_x, \mu_x) \mid 1 \leq x < N\}$, the set $\{(a, b), (a \oplus \lambda_x, b \oplus \mu_x) | 1 \leq x < N\}$, defines the affine template $T(a, b)$ of T.

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)

$$Q_4 = \left(\begin{array}{cc|cc} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right)$$

(b)

Figure 2: A blip scheme to store a 16×16 data matrix.

3 The blip schemes

In this section, we give the blip storage scheme and discuss the criteria for having various templates free of memory conflicts. ($N \times N$ data matrix is assumed.)

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. ∎

Blip schemes facilitate 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 2(a), we give the mapping matrix for storing a 16×16 data matrix in 16 memory units, as specified by a blip scheme. The boolean matrix Q_4 corresponding to the linear permutation used in the blip scheme is given in figure 2(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 units. For more details, see [6].

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 [23] network as the processor-memory interconnection network and using the self-routing techniques developed for these networks [5]. Furthermore, the data movement can be pipelined with simple address computations. For these reasons, re-skewing of data from one blip scheme to another is fast and efficient [6].

3.1 Conflict free access of the templates of interest

In what follows, we summarize the known results about the blip schemes. These results, including the issue of fast address generation, are discussed in detail in [6]. 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[4] 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 it 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 the 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.*

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

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

The following theorems give the criteria for a blip scheme to provide conflict free access of various templates. More details can be found in [6].

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.* ∎

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''.* ∎

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'.* ∎

3.2 Address generation

For fast access of any template of a matrix, besides having each element of the template in a distinct memory unit, a fast scheme to generate addresses of the elements in the template should be provided. When a blip scheme is used, for accessing the important templates, each processor can generate the memory unit number corresponding to its data, independently, in constant time with only n exclusive-or gates.

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 unit 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 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, i); 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.

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 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 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 $\Omega_N \uparrow f(T_x)$ ($\Omega_N^{-1} \uparrow f(T_x)$).

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 modified Omega network.

The interconnection network

An 8 processor system with the modified Omega network is shown in figure 3. 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.

4.1 Characterization of Omega passable linear permutations

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

Pease [19] proved that a linear permutation π is realizable in an indirect binary n-cube network (identical to the inverse Omega network [18]), if and only if it is composed of two linear permutations: an upper triangular permutation followed by a lower triangular permutation, that is, $\pi = \alpha\upsilon$ (composition of permutations is right to left). So, a linear permutation π is passable in an Omega network, if and only if $\pi = \upsilon\alpha$. An equivalent criterion is given below to characterize the Omega passable linear permutations. This is a special case of the more general property proved by Varma [21] to count the number of passes required to realize a linear permutation in the Omega network.

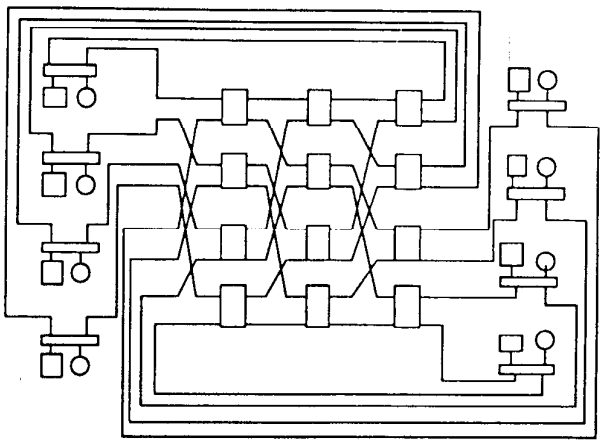

Figure 3: An 8 processor system with the modified Omega network.

Definition 4 *For a boolean matrix $Q = (q_{i,j})_{n \times n}$, the leading principal submatrix $Q(k)$, $0 \leq k < n$, 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 "trailing" principal submatrix $Q(k)^e$, $0 \leq k < n$, 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 factored into non-singular lower triangular and upper triangular boolean matrices without pivoting, if and only if each of its leading principal submatrices are non-singular.*

Proof of a more general case of this result is given by Golub and Van Loan (theorem 3.2.1, [10]). ∎

Theorem 5 *A linear permutation with boolean matrix Q is passable in the inverse Omega network, if and only if each of the leading principal submatrices of Q are non-singular.*

Proof follows from the above lemma and the Pease's result discussed above. ∎

To apply the above theorem to the Omega passable linear permutations, we use the following result due to Parker [18]. Let ρ denote the bit-reversal permutation and f be an Ω_N^{-1} passable permutation. Then, $\Omega \uparrow \rho f \rho$.

Corollary 1 *A linear permutation with boolean matrix Q is passable in the Omega network, if and only if each of the trailing principal submatrices of Q is non-singular.*

Proof follows from theorem 5 and the Parker's result. ∎

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

Theorem 6 *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.

4.2 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. As an example, L_4 and U_4 are given below. 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.

$$L_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad U_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Theorem 7 *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 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 is immediate by the application of theorem 4. ∎

Lemma 3 *Each of the leading principal submatrices of Γ_n is non-singular.*

Proof: For $0 \le i < n/2$, $\Gamma_n(i)$ is a lower triangular matrix, hence non-singular. Now, consider $\Gamma_n(n/2)$, given below.

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

In this submatrix, 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 are independent as well, since 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, it is non-singular. This argument can be extended to show that, for $n/2 \le i < n$, $\Gamma_n(i)$ 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 for $f(T_d)$. 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 general,

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

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

Proof: Use the following transformation on Γ_n''. Replace row i, $0 \le i < n/2$, of Γ_n'' with the sum of rows i and $i + n/2$. The resulting matrix, given below, is non-singular. Hence, Γ_n'' is non-singular.

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

Lemma 5 *Each of the trailing principal submatrices of Γ''_n is non-singular.*

Proof: Consider $\Gamma''_n(k)^c$, $0 \leq k < n/2$, given below.

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

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, for $0 \leq k < n/2$, $\Gamma''_n(k)^c$ is non-singular. For $n/2 \leq k < n$, $\Gamma''_n(k)^c$ 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 8 *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], 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. 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 $\tilde{\delta}_0 = N-2$, $\tilde{\delta}_{n-1} = N-1$, 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 trailing principal submatrices 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 leading principal submatrices of Δ''_n is a non-singular lower triangular matrix. And, Δ''_n itself is non-singular. We summarize this in the following theorem.

Theorem 9 *The blip scheme defined by Δ_n gives conflict free access of row and column templates 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.* ∎

5 Other applications

We now briefly discuss other applications of the blip shemes.

Blip schemes for distributed memory systems

The blip schemes can be used with no changes for 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. With the use of blip schemes, the data transfer functions will be in the form of linear-complement permutations for the templates T_r, T_c, T_d, and T_s. For the hypercube and the mesh computers, the self-routing techniques developed for them [7] can be used to realize the transfer functions of these templates.

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. The issues related to address generation and routing in interconnection networks are similar to the case $m = N$.

Blip schemes to store multi-dimensional matrices

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, suggested by Fairlong et al., is to apply 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.

6 Summary and conclusions

In this paper, we considered a class of scrambled skewing schemes, called as blip schemes. These schemes have 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, (c) simple address generation methods, and (d) fast and efficeint re-skewing of data. These aspects are crucial for the use of a storage scheme.

Later, we devised, using the theory developed for the class of blip schemes, 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 units and an interconnection network that is approximately half as big as that 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. Self-routing schemes in parallel memory access. Technical Report CENG 89-36, Dept. of EE-Systems, Univ. of Southern Cal., Univ. Park, Los Angeles, CA 90089-0781, December 1989.

[7] R. Boppana and C. S. Raghavendra. Optimal self-routing of linear-complement permutations in hypercubes. In *The Fifth Distibuted Memory Computing Conference*, 1990.

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

[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] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins Univ. Press, 1989.

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

[12] K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[13] K. Kim and V. K. Prasanna 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.-L. 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. S. Parker. Notes on shuffle/exchange-type switching networks. *IEEE Trans. on Computers*, c-29(3):213–222, 1980.

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

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

[21] A. Varma, 1989. Personal communication.

[22] 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.

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