Minimization of Memory and Network Contention for Accessing Arbitrary Data Patterns in SIMD Systems

Steven S. Seiden and Mayez A. Al-Mouhamed

ICS-UCI Technical Report 93-29

June 18, 1993

Minimization of Memory and Network Contention for Accessing Arbitrary Data Patterns in SIMD Systems

Steven S. Seiden* and Mayez A. Al-Mouhamed†

ICS-UCI Technical Report 93-29

June 18, 1993

Abstract

Non-uniform memory and network access is a major source of performance degradation in SIMD supercomputers. We investigate the problem of finding general XOR-schemes to minimize memory conflicts and network contention for accessing arrays with arbitrary data *templates*, defined by template *bases*.

The XOR-matrix is defined so that each column corresponds to a distinct vector in the union of all templates bases. A *restriction* of the XOR-matrix to a given template is formed by concatenation of the columns corresponding to the template basis. We prove that a necessary and sufficient condition for conflict-free and network-contention-free access for the *Baseline network* is that certain sub-matrices of every template's restricted matrix be non-singular. A new characterization of the baseline network and XOR-matrices is proposed. Finding an XOR-matrix for accessing arbitrary templates is proved to be an NP-complete problem.

To minimize memory and network contention, a heuristic algorithm is proposed for finding XOR-matrices. The algorithm determines successive rows, from the bottom up. Given the previous row, the algorithm determines: 1) the constraints required by each template's restricted matrix 2) and the row solution by solving a set of simultaneous equations. To avoid backtracking, a randomized approach is used. The time complexity of the heuristic is $O(tpn^2)$, where t, 2^p, and n, are the number of templates, the number of processors, and the number of distinct vectors of template bases, respectively.

Evaluation shows that the proposed XOR-schemes significantly reduce memory and network contention compared to interleaving and XOR-schemes that are optimized for a set of static reference reference templates.

*Department of Information and Computer Science, University of California, Irvine, CA 92717.

†Department of Information and Computer Science, University of California, Irvine, CA 92717. On leave from King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.

Keywords: Memory Conflicts, Multi-Stage Networks, NP-completeness, Parallel Memories, Storage Schemes

1 Introduction

When high performance computers are used, non-uniform access to parallel memories and network contention are responsible for significant performance degradation [15, 1], especially in SIMD systems. Generally, subdividing the main memory into units that operate in parallel does not linearly increase the system bandwidth.

Memory interleaving has been widely used because it optimally maps sequential array addresses into successive memory units. Operations on arrays require most of the available bandwidth for SIMD computers. Sequential addresses form arithmetic sequences, or *strides*, where successive addresses differ by a constant amount. Because significant degradation occurs when the stride is not relatively prime to the number of memories, more sophisticated storage schemes were investigated in order to eliminate or reduce memory and network conflicts. The use of a prime number of memories [15] significantly outperforms regular interleaving but requires computationally expensive [9] address translation.

Improving the performance of scalar access to parallel memories has also been investigated for long-instruction-word (LIW) computers [8]. LIWs have multiple functional units that operate in lock-step and fetch data required by parallel operations from multiple memory modules. Conflict-free allocation of data may not always exist, but a high percentage of memory conflicts can be avoided [8] by using a very low degree of data duplication.

For *load/store* operations in SIMD computers, network and memory contention cause all the PE's to wait until the completion of the last data transfer [5]. By restricting the type of access to some fixed data patterns, conflict-free access [5, 2, 14] to rows, columns, and diagonals of arrays were proposed on the basis of row rotations. In this case, adequate data organization enables load/store operations to execute in a parallel fashion (one cycle) because each PE maps its array element to a distinct path through the network and to a distinct memory unit. Another approach [16] uses matrices of control patterns for the interconnection network to cause the dynamic permutation of data. The drawbacks of these approaches are: dependence on the array size, dependence on the number of memories, and complex address transformations. While these approaches are useful, several researchers recognized [6, 13] that restricting the conflict-free access to some fixed templates was not realistic assumption for general computing.

Based on *skew storage*, a more flexible approach [5, 6, 10, 19], called *XOR-schemes* was proposed based on linear bitwise transformation of addresses. The efficiency of XOR-schemes stem from the fact that modulo-2 arithmetic can be implemented using the logical **and** and **exclusive-or** operators. This approach requires the use of a number of processors and memory units which is a power of 2, in addition to some compile-time knowledge of the accessed templates. By considering mainly the memory requirements, these schemes were further analyzed and several variants were proposed [6, 11, 17]. In the following we present some of these variants.

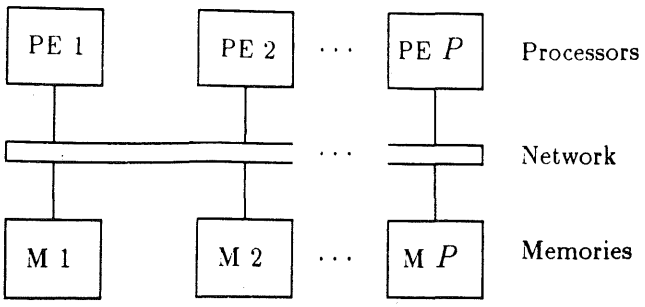


Figure 1: Structure of a SIMD architecture

For vector processors, an XOR-scheme can be found [9] by factoring the stride into two integers, the first being a power of 2, and the other being relatively prime to 2. This results in optimized stride access. In addition, this scheme outperforms regular interleaving for other strides provided that memory units are augmented by some buffering capabilities.

In [6], it is shown that a necessary and sufficient condition for conflict-free memory access of a given template is that the image by the storage scheme of the template basis is linearly independent. While the XOR-scheme of a single template can be easily found, no general methods were proposed for finding the XOR-scheme for composite templates. In some specific cases, the combined scheme is found for a set of templates.

Because of the excessive cost of crossbar switches, multistage networks such as the Benes [3], Omega [14], and Baseline [18], were analyzed with respect to their permissible permutations. In most cases, conflict-free access to the network is obtained for some fixed data templates. For row, column, diagonals, and square blocks, a scheme [4] based on composite linear permutations was proposed for the Omega network.

While all these approaches are useful, they either: 1) treat only the parallel memory characterization, or 2) treat memory and network aspects by considering only a reference set of static templates. In this work, we consider an SIMD computer that consists of a number of processing elements and memory units connected by a network, as displayed in Figure 1. Our objective is to find a methodology for combining the requirements of composite data templates in a general and coherent XOR-scheme. We shall present a complete approach for finding the XOR-matrix that minimizes memory and network contention for accessing arbitrary data templates in SIMD systems. We restrict ourselves to the Baseline network.

This paper is organized as follows. Section 2 describes the notation used and background. Section 3 defines templates and XOR-schemes. Characterizations of the Baseline network and non-singular matrices are presented in Section 4. The memory and network requirements for conflict-free access of arbitrary data templates and their NP-completeness aspects are presented in Section 5. An algorithm for finding XOR-schemes is presented in Section 6. Section 7 presents performance evaluation, and Section 8 concludes this work and outlines future extensions.

2 Notation

Our study of XOR-schemes makes use of concepts from linear algebra. XOR-schemes operate on integers as bit vectors. We set forth the notation we shall use, and briefly review some concepts.

In general, we denote abstract objects such as vector spaces and templates using capital cursive letters, i.e. the vector space $\mathcal{V}$. Set are denoted using capital italic letters, i.e. the set S. Integers and vectors are denoted using small italic letters, i.e. the vector x. Components of vectors are denoted by subscripting. The ith component of the vector x is x_i. Bits of integers are referred to in a similar fashion. The binary representation of a d bit integer x is $x_{d-1} \ldots x_1 x_0$. In other words, the zeroth bit (least significant) of x is x_0, the first bit is x_1 etc....

We can identify the set of integers in the range $[0 : 2^d - 1]$ with the d dimensional vector space over $\mathbb{Z}_2$, the integers modulo 2. The vector $(x_0, x_1, \ldots, x_{d-1})$ corresponds to the integer $x = x_{d-1} \ldots x_1 x_0$. Each bit of the integer is a component of the vector. *Note that the order of the bits in the vector is the reversal of the order in the binary representation.* We adopt this convention to be consistent with preceding literature. For example, 13 in binary is 1101, and so the vector representation of 13 is $(1, 0, 1, 1)$.

In the field $\mathbb{Z}_2$, addition and multiplication are defined thus:

$\oplus$	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

We use the symbol $\oplus$ to denote addition in $\mathbb{Z}_2$, and to denote addition of vectors over $\mathbb{Z}_2$. Note that if we let 1 be the Boolean value **true**, and 0 be the Boolean value **false**, addition corresponds to logical **exclusive-or**, and multiplication corresponds to logical **and**. Adding two vectors in $\mathbb{Z}_2^n$ corresponds to taking the bitwise **exclusive-or** of the binary representations of the corresponding integers. For example, $(1, 1, 0, 1) \oplus (0, 1, 1, 0) = (1, 0, 1, 1)$, this corresponding to the bitwise **exclusive-or** of 11 and 6, which is 13. Multiplying a vector by the scalar 1 gives back that vector, while multiplying a vector by the scalar 0 results in the zero vector.

A set of vectors in a vector space are *linearly independent* if and only if no non-zero *linear combination* of them is zero. Formally, if we have a set of vectors $\{v_0, \ldots, v_{n-1}\}$ then they are linearly independent if and only if the linear combination:

$$a_0 v_0 \oplus \cdots \oplus a_{n-1} v_{n-1}$$

where each a_i is a scalar, is zero exactly when all a_i are zero.

The vector space $\mathbb{Z}_2^n$ is generated by an ordered set of *basis vectors*. The basis of a vector space is linearly independent. Further, the basis *spans* the space, every vector in the space can be represented as a unique linear combination of the basis vectors. One possible basis of $\mathbb{Z}_2^n$ is the set:

$$v_0 = (1, 0, 0, \ldots, 0)$$

$$\begin{aligned} v_1 &= (0,1,0,\ldots,0) \\ \vdots &\ \vdots\ \vdots \\ v_{n-1} &= (0,0,0,\ldots,1) \end{aligned}$$

This basis is known as the canonical basis of $\mathbb{Z}_2^n$. In this paper, we will always use canonical bases. We illustrate these concepts with a small example. Let $\mathcal{V} = \mathbb{Z}_2^2$. The canonical basis of this vector space is:

$$\begin{aligned} v_0 &= (1,0) \\ v_1 &= (0,1) \end{aligned}$$

We can express all four vectors in $\mathcal{V}$ as linear combinations of these two:

$$\begin{aligned} (0,0) &= 0v_0 \oplus 0v_1 \\ (1,0) &= 1v_0 \oplus 0v_1 \\ (0,1) &= 0v_0 \oplus 1v_1 \\ (1,1) &= 1v_0 \oplus 1v_1 \end{aligned}$$

Let ϕ be a function. Then we say ϕ is *linear* if $\phi(x \oplus y) = \phi(x) \oplus \phi(y)$ and $\phi(cx) = c\phi(x)$. Suppose ϕ is a linear function $\phi : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^m$. The function ϕ is represented by an $m \times n$ matrix Φ over $\mathbb{Z}_2$. We apply ϕ to a vector x by matrix multiplication:

$$\phi(x_0, x_1, \ldots, x_{n-1}) = \Phi \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

The upper left-most entry of Φ is $\Phi_{0,0}$. We denote the ith row of Φ by $\Phi_{i,*}$, and the jth column of Φ by $\Phi_{*,j}$. The columns of Φ represent the values of ϕ on the basis vectors of $\mathbb{Z}_2^n$. I.e., if v_j is the jth basis vector, $\Phi_{*,j}$ is the value of $\phi(v_j)$. ϕ will be onto if and only if Φ has full rank.

Since ϕ produces a vector, we wish to consider how ϕ produces each component of that vector. We define $\phi_i(x)$ to be the ith component of the vector $\phi(x)$. In other words:

$$\phi(x_0, \ldots, x_{n-1}) = (\phi_0(x_0, \ldots, x_{n-1}), \ldots, \phi_{m-1}(x_0, \ldots, x_{n-1}))$$

Or more succinctly:

$$\phi(x) = (\phi_0(x), \ldots, \phi_{m-1}(x))$$

The matrix representing ϕ_i is $\Phi_{i,*}$.

Given a subset S of the basis of $\mathbb{Z}_2^n$, we can create a *restriction* of ϕ to S denoted by ϕ^S. Since the basis of $\mathbb{Z}_2^n$ is an ordered set, an ordering of S is imposed by the ordering of

the basis. The matrix representing the restricted function is denoted Φ^S. Formally, if S is a subset of the basis $v_0, \ldots, v_{n-1}$ of $\mathbb{Z}_2^n$, and m is the size of S, then ϕ^S is defined by:

$$\phi^S(x) = (\phi_{i_0}(x), \ldots, \phi_{i_{m-1}}(x))$$

where v_{i_j} is the jth member of S. The matrix Φ^S consists of the columns $\Phi_{*,i_0} \ldots \Phi_{*,i_{m-1}}$, concatenated in that order.

We give a small example. Let $\phi : \mathbb{Z}_2^4 \mapsto \mathbb{Z}_2^3$. We assume that the basis V of $\mathbb{Z}_2^4$ is:

$$\begin{aligned} v_0 &= (1,0,0,0) \\ v_1 &= (0,1,0,0) \\ v_2 &= (0,0,1,0) \\ v_3 &= (0,0,0,1) \end{aligned}$$

Let ϕ be defined by the matrix:

$$\Phi = \begin{array}{c} \begin{array}{cccc} v_0 & v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Note that we have labeled each column with the basis vector to which it corresponds. I.e., the value of $\phi(v_0) = (0,1,0)$, $\phi(v_1) = (1,0,0)$ etc.... Let $S = \{v_0, v_2, v_3\}$. Then the matrix representing ϕ^S is:

$$\Phi^S = \begin{array}{c} \begin{array}{ccc} v_0 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

3 XOR-schemes

We now formally define XOR-schemes. First we formalize the relationship between array positions and vectors. We then formally define templates with respect to XOR-schemes. Finally, we define XOR-schemes themselves.

3.1 Definition of Vector Spaces Used

We wish to identify each array position (a, b) with a unique vector. The row index a can be identified with the vector $(a_0, \ldots, a_{d-1})$ and the column index b can be identified with $(b_0, \ldots, b_{d-1})$. Consequently, array position (a, b) can be uniquely identified with $(a_0, \ldots, a_{d-1}, b_0, \ldots, b_{d-1})$. We also identify each memory unit number c with $(c_0, \ldots, c_{p-1})$. We now define formally the vector spaces to which we have been alluding.

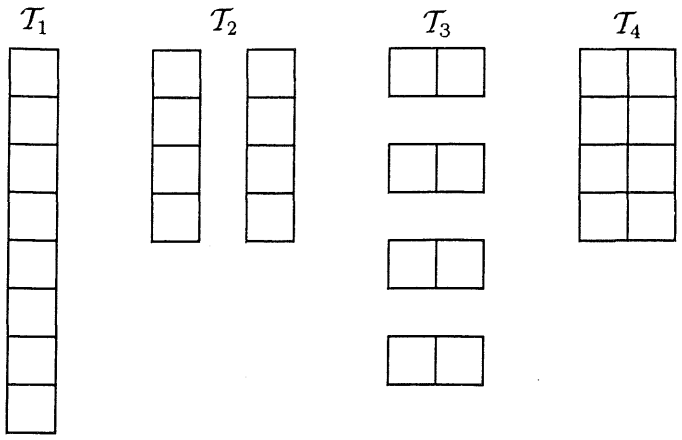


Figure 2: From left to right $\mathcal{T}_1 \ldots \mathcal{T}_4$

We define a vector space $\mathcal{F} = \mathbb{Z}_2^d$ to represent horizontal indices. We define $F = \{f_0, f_1, \ldots, f_{d-1}\}$ to be the canonical basis of $\mathcal{F}$. So we have:

$$\begin{aligned} f_0 &= (1,0,0,\ldots,0), \\ f_1 &= (0,1,0,\ldots,0), \\ &\vdots \quad \vdots \quad \vdots \\ f_{d-1} &= (0,0,0,\ldots,1) \end{aligned}$$

Each row index has a unique representation as a vector in $\mathcal{F}$. We similarly define vector spaces $\mathcal{G}$ for column indices and $\mathcal{H}$ for memory unit numbers, with canonical bases $G = \{g_0, g_1, \ldots, g_{d-1}\}$ and $H = \{h_0, h_1, \ldots, h_{p-1}\}$, respectively.

The Cartesian product of the vector spaces $\mathcal{F}$ and $\mathcal{G}$ is a new vector space $\mathcal{V} = \mathcal{F} \times \mathcal{G}$ with basis $F \cup G$. Let $n = 2d$. We denote this combined basis as $V = \{v_0, v_1, \ldots, v_{n-1}\}$, where $v_0 = f_0$, $v_1 = f_1$, $v_d = g_0$ etc.... This vector space is isomorphic to $\mathbb{Z}_2^n$. Any position (a, b) in the array is uniquely associated with a vector in $\mathcal{V}$. This vector can be expressed as a linear combination of the basis elements:

$$a_0 v_0 \oplus \cdots \oplus a_{d-1} v_{d-1} \oplus b_0 v_d \oplus \cdots \oplus b_{d-1} v_{n-1}$$

We refer to the elements of the basis V using either f's and g's, or v's, depending on which is notationally convenient.

3.2 Data Templates

A *template* is a pattern on array element locations. Templates are used to describe the access patterns of an algorithm. A particular instantiation of a template is called a *template instance*. Given a set of templates, we would like to find a function that allows us to access them all in a *conflict-free* manner. By this we mean that for all given templates, for all

template instances, all of the elements of any template instance map to distinct memory units.

XOR-schemes can handle a large class of useful templates, such as rows, columns, square and rectangular blocks etc.... However, a restriction of XOR-schemes is that they can handle only non-overlapping templates. We now define formally what is meant by a template, with respect to XOR-schemes.

We define a template $\mathcal{T}_i$ by a basis T_i, which is a nonempty subset of V. We assume all template bases are of size p. Notice that there is a definite distinction between the definition of a template $\mathcal{T}_i$, which is a pattern, and its basis T_i, which is a set of vectors.

This is best explained by looking at some examples such as the templates that are illustrated on Figure 2. Let p and d both be 3. Our basis is $V = \{f_0, f_1, f_2, g_0, g_1, g_2\}$, or alternatively $V = \{v_0, v_1, \ldots, v_5\}$. Consider the template $\mathcal{T}_1$ defined by $T_1 = \{f_0, f_1, f_2\}$. The set of template instances described are all non-overlapping columns of eight elements. Every element in a template instance is a linear combination:

$$a_0 f_0 \oplus a_1 f_1 \oplus a_2 f_2 \oplus b_0 g_0 \oplus b_1 g_1 \oplus b_2 g_2$$

where the b's are constant, and the a's are allowed to vary. The position with all the varying bits set to zero is the template instance's *origin*. $((0,0,0,b_0,b_1,b_2)$ is the origin in our example). Intuitively, we are letting the bits of a vary, while the bits of b remain fixed. By allowing different sets of bits to become variable, we generate templates of different shapes. Let $\mathcal{T}_2$ have basis $T_2 = \{f_0, f_1, g_1\}$. Then, in $\mathcal{T}_2$ all template instances are four elements tall. Since g_0 is omitted, this template skips a column. Thus, we have two 4×1 sub-arrays, spaced two columns apart. We define $\mathcal{T}_3$ by $T_3 = \{f_1, f_2, g_1\}$. This template is four 1×2 sub-arrays spaced two rows apart. We let $\mathcal{T}_4$ have basis $T_4 = \{f_0, f_1, g_0\}$. It is a 4×2 sub-array.

3.3 XOR-Schemes

An XOR-scheme is a linear function $\phi : \mathcal{F} \times \mathcal{G} \mapsto \mathcal{H}$, which is represented by a $p \times n$ matrix Φ.

An XOR-scheme ϕ allows conflict-free access to $\mathcal{T}_i$ if and only if ϕ maps each linear combination of T_i to a unique element of $\mathcal{H}$ [6]. In other words, ϕ^{T_i} must be onto. For conflict-free access to all templates, ϕ^{T_i} must be onto for all T_i.

4 The Baseline Network

We consider SIMD systems in which the PE's are interconnected to the memories by using *multi-stage networks* that basically are permutation networks such as the Baseline, Omega, Delta, etc. Their basic element is the 2×2 switch, but the characteristics of the permutations each network can pass depend on its topology. The *Baseline network* is considered here as the data-alignment system between the memory units and the processors. To simplify the notation, we assume that processor addresses and corresponding data travel through the

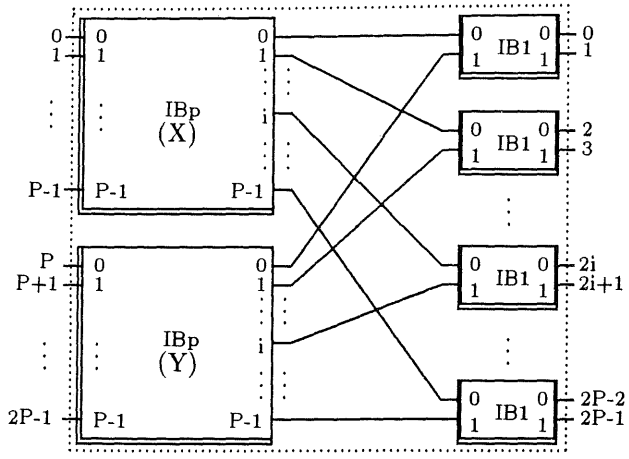
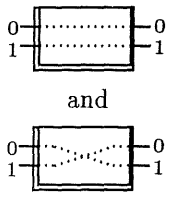


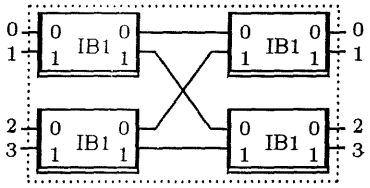
Figure 3: Structure of IB(p+1)

inverted-baseline and baseline, respectively. In this section, we define the inverted baseline network, and study its structure and properties.

The 1-stage inverted baseline network IB1 consists of a single switch with two inputs and two outputs. The switch can perform both of the possible permutations of outputs to inputs, $\binom{01}{01}$ and $\binom{01}{10}$:



The 2-stage network IB2 consists of four switches connected in the following manner:



Note that this network can perform 16 permutations, but the total number of possible permutations is 24. For instance, IB2 cannot perform the identity permutation $\binom{0123}{0123}$. In general, the p-stage inverted baseline network IBn has $P = 2^p$ inputs and outputs. We define IB$(p+1)$ recursively in terms of IBp as is shown in Figure 3. IB$(p+1)$ uses two copies of IBp, which we shall call X and Y, and P individual switches. The inputs of X are inputs 0 through $P-1$ of the network. The inputs of Y are inputs P through $2P-1$ of the network. The ith single switch has its outputs 0 and 1 as outputs $2i$ and $2i+1$ of the

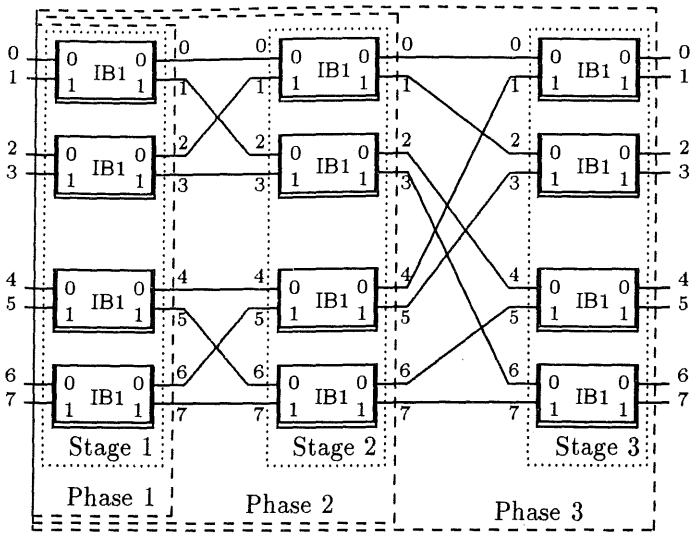


Figure 4: Stages and phases of IB3

network, respectively. Input 0 of this switch is connected to the ith output of X, and input 1 connected to the ith output of Y.

We define a *sub-network* of a network to be some subset of the switches in that network. From the definition of IBp, we know that it consists of two sub-networks which are copies of IB$(p-1)$, and 2^{p-1} sub-networks which are copies of IB1. However, this is not the only way to partition the network into sub-networks. We may also consider the network in terms of its *stages*. For instance, the three stages of IB3 are illustrated in Figure 4. Note that each stage consists of 4 switches, and has 8 inputs and outputs. The inputs of the first stage are the inputs of the network, and the outputs of the last stage are the outputs of the network. In general, it is easily seen that IBp will have 2^{p-1} switches in each of its stages. Proof of this follows directly from the recursive definition of IBp.

We consider other useful sub-networks. We can see from the previous example that stage 1 consists of four copies of IB1 (which are just single switches), and stages 1 and 2 combined consist of two copies of IB2. We call the sub-network consisting of stages 1 through i a *phase*, or more specifically phase i. We see that in a p stage network, phase i will consist of 2^{p-i} copies of IBi. Proof of this also follows directly from the definition of IBp. We call these copies *sub-phases*, and number them from top to bottom. Sub-phase 0 receives inputs 0 through 2^i-1, and produces outputs 0 through 2^i-1; Sub-phase 1 receives inputs 2^i through $2\cdot 2^i-1$, and produces outputs 2^i through $2\cdot 2^i-1$; etc.... The phase structure of IB3 can also be seen in Figure 4.

4.1 Routing

We study how a message passes through the network from a given source to a given destination. By doing so, we will gain a better understanding of which permutations the network can handle.

Consider a message passing from input 0 of IB3 to output 5. We refer the reader to Figure 4. The message passes through the first switch of stage 1 and reaches output 1 of stage 1. This output is connected to input 2 of stage 2. The message then passes through the second switch of stage 2 and exists stage 2 via output 2. Finally, the message enters input 4 of stage 3, passes through the third switch, and exists through output 5. Let us express this in a more concise form, using numbers in binary:

$$000 \rightarrow 001 \rightarrow 010 \rightarrow 101$$

The leftmost number is the source, and the rightmost is the destination. The numbers in between represent the position of the message after each stage. Consider some other examples. The routing from 4 to 7 is:

$$100 \rightarrow 101 \rightarrow 111 \rightarrow 111$$

and from 7 to 0 is:

$$111 \rightarrow 110 \rightarrow 100 \rightarrow 000$$

Now consider again the routing from 0 to 5, and notice how the source position is transformed into the destination position:

$$000 \rightarrow 00\boxed{1} \rightarrow 0\boxed{10} \rightarrow \boxed{101}$$

We have placed a box around the bits of the destination position. At each stage, the bits of the destination are 'shifted in' to replace those of the source. Reconsidering the previous examples, we see that this pattern of 'shifting in' holds. We prove that it holds in the general case.

Theorem 4.1 *Given a p-stage network, let* $\mathrm{pos}_i(s,d)$ *be the position of a message passing from input* s *to destination* d *after the* i*th stage. Then:*

$$\mathrm{pos}_i(s,d) = s_{p-1} \ldots s_i d_{p-1} \ldots d_{p-i} \tag{1}$$

in binary representation, where d_j *is the* j*th bit of* d*, and* s_j *is the* j*th bit of* s*. (The least significant bit is the zeroth, etc...)*

Proof The proof is by induction. The base case is $\mathrm{pos}_p(s,d) = d = d_{p-1} \ldots d_0$. We assume that:

$$\mathrm{pos}_{i+1}(s,d) = s_{p-1} \ldots s_{i+1} d_{p-1} \ldots d_{p-i-1}$$

and we show that:

$$\mathrm{pos}_i(s,d) = s_{p-1} \ldots s_i d_{p-1} \ldots d_{p-i}$$

Since the message enters phase $(i+1)$ of the network at:

$$s = s_{p-1} \ldots s_0$$

and exits at:

$$s_{p-1} \ldots s_{i+1} d_{p-1} \ldots d_{p-i-1}$$

it is apparent that the message passes through sub-phase:

$$T = s_{p-1} \ldots s_{i+1}$$

The starting position of the message relative to this sub-phase is

$$t = s_i \ldots s_0$$

The exiting position relative to this sub-phase is:

$$e = d_{p-1} \ldots d_{p-i-1}$$

Note that:

$$\begin{array}{rcl} \mathrm{pos}_{i+1}(s,d) & = & T \cdot 2^{i+1} + e \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 0 \ \ldots\ldots \ 0 \\ & + & 0 \ \ldots\ldots\ldots \ 0 \ \ d_{p-1} \ \ldots \ d_{p-i-1} \end{array}$$

and that:

$$\begin{array}{rcl} s & = & T \cdot 2^{i+1} + t \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 0 \ \ldots\ldots \ 0 \\ & + & 0 \ \ldots\ldots\ldots \ 0 \ \ s_i \ \ldots\ldots \ s_0 \end{array}$$

Since sub-phase T is a copy of IB$(i+1)$, consider again Figure 3. If $t < 2^i$ (i.e. $s_i = 0$) then the message originated in the upper copy of IBi, as illustrated in the figure. Its exiting position relative to this upper copy must have been:

$$f = \lfloor e/2 \rfloor = d_{p-1} \ldots d_{p-i}$$

Therefore:

$$\begin{array}{rcl} \mathrm{pos}_i(s,d) & = & T \cdot 2^{i+1} + f \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 0 \ \ldots\ldots\ldots \ 0 \\ & + & 0 \ \ldots\ldots\ldots \ 0 \ \ 0 d_{p-1} \ \ldots\ldots \ d_{p-i} \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 0 d_{p-1} \ \ldots\ldots \ d_{p-i} \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ s_i d_{p-1} \ \ldots\ldots \ d_{p-i} \end{array}$$

If $t \geq 2^i$ (i.e. $s_i = 1$) then the message originated in the lower copy of IBi, as illustrated in the figure. Its exiting position relative to this lower copy must have been:

$$f' = 2^i + \lfloor e/2 \rfloor = 1 d_{p-1} \ldots d_{p-i}$$

So, once again:

$$\begin{array}{rcl} \mathrm{pos}_i(s,d) & = & T \cdot 2^{i+1} + f' \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 0 \ \ldots\ldots\ldots \ 0 \\ & + & 0 \ \ldots\ldots\ldots \ 0 \ \ 1 d_{p-1} \ \ldots\ldots \ d_{p-i} \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ 1 d_{p-1} \ \ldots\ldots \ d_{p-i} \\ & = & s_{p-1} \ \ldots\ldots \ s_{i+1} \ s_i d_{p-1} \ \ldots\ldots \ d_{p-i} \end{array}$$

■

4.2 Linear Permutations

Let ϕ be a linear function from $\mathbb{Z}_2^p$ onto $\mathbb{Z}_2^p$. Then ϕ is a permutation on $\mathbb{Z}_2^p$, and we say that ϕ is a *linear permutation.* ϕ is defined by a $p \times p$ matrix Φ.

We may also consider ϕ as a permutation of the numbers $[0 : 2^p - 1]$, by identifying each integer x, having binary representation $x_{p-1} \ldots x_0$, with the vector $(x_0, \ldots, x_{p-1})$. We wish to know if the inverted baseline network can perform ϕ. From Equation (1) we find that:

$$\mathrm{pos}_i(s, \phi(s)) = (\phi_{p-i}(s), \ldots, \phi_{p-1}(s), s_i, \ldots, s_{p-1})$$

$\mathrm{pos}_i(s, \phi(s))$ is the position of a message going from s to $\phi(s)$, after the ith stage. Since we always refer to a message going from s to $\phi(s)$, we shall abbreviate $\mathrm{pos}_i(s, \phi(s))$ to $\mathrm{ps}_i(s)$. If Φ is the matrix of ϕ, the matrix of $\mathrm{ps}_i(s)$ is:

$$\begin{pmatrix}
\Phi_{p-i,0} & \cdots & \Phi_{p-i,i-1} & \cdots & \cdots & \cdots & \cdots & \Phi_{p-i,p-1} \\
\vdots & & \vdots & & & & & \vdots \\
\Phi_{p-1,0} & \cdots & \Phi_{p-1,i-1} & \cdots & \cdots & \cdots & \cdots & \Phi_{p-1,p-1} \\
0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 \\
0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 \\
\vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\
0 & \cdots & 0 & 0 & 0 & \cdots & 1 & 0 \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1
\end{pmatrix} \tag{2}$$

We define:

$$\Phi[i] = \begin{pmatrix}
\Phi_{p-i,0} & \cdots & \Phi_{p-i,i-1} \\
\vdots & & \vdots \\
\Phi_{p-1,0} & \cdots & \Phi_{p-1,i-1}
\end{pmatrix} \tag{3}$$

Note that $\Phi[i]$ is just the $i \times i$ sub-matrix in the upper-left corner of matrix (2), and the $i \times i$ sub-matrix in the lower-left of Φ.

Theorem 4.2 ps_i *is onto if and only if* $\Phi[i]$ *is non-singular.*

Proof ps_i will be onto if and only if its matrix (2) is non-singular. The matrix (2) contains the identity matrix in its lower-right hand corner. We can cancel any entry $\Phi_{j,k} = 1$ where $p - i \leq j \leq p - 1$ and $0 \leq k < i$, by adding row k to row j. This leaves the identity in the lower right quadrant, matrix (3) in the upper-left quadrant, and zeros in the remaining quadrants. Therefore, the non-singularity of (2) depends on its upper-left quadrant, which is matrix (3). ∎

We now characterize linear permutations ϕ which the inverted baseline network can perform. We say that an $p \times p$ matrix Φ is *sub-non-singular* if and only if $\Phi[i]$ is non-singular for $1 \leq i \leq p - 1$.

Theorem 4.3 *A linear permutation ϕ on $\mathbb{Z}_2^p$ can be performed by IBp, if and only if its matrix Φ is sub-non-singular.*

Proof If two messages are mapped to the same output of a single switch in the network, the permutation will fail. If some sub-matrix $\Phi[i]$ is singular, then ps_i is not onto, and two messages will be mapped to the same output of some switch in the ith stage. ■

4.3 Sub-Non-Singular Matrices

We find the number of $p \times p$ matrices M over $\mathbb{Z}_2$ which are sub-non-singular. Suppose the number of such matrices of a given size is small compared to the number of non-singular matrices of that size. Then the types of templates that could be handled using XOR-schemes on a computer with an inverted baseline network computer is severely restricted. We show this is not the case. Furthermore, since the proof we give is constructive, it will aid us in devising an algorithm for creating XOR-schemes.

Let us denote the number of $p \times p$ matrices which are sub-non-singular by S_p. Examining the small cases, it is obvious that $S_1 = 1$. With a little work, we find the set of 2×2 matrices which are sub-non-singular:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

and so $S_2 = 4$. We now show that:

Theorem 4.4 *Let S_p be the number of sub-non-singular $p \times p$ matrices over $\mathbb{Z}_2$. Then:*

$$S_p = 2^{(p-1)p} \tag{4}$$

Proof Suppose we are given an arbitrary $p \times p$ matrix M. Assume that $M[i]$ is sub-non-singular. Then by performing row and column operations, we can transform M such that $M[i]$ is the mirror image of the identity matrix thus:

$$\begin{pmatrix} \vdots & & & & & \vdots & \\ b_{i-1} & b_{i-2} & \cdots & b_1 & b_0 & c & \cdots \\ 0 & 0 & \cdots & 0 & 1 & a_0 & \\ 0 & 0 & & 1 & 0 & a_1 & \\ \vdots & \vdots & & \vdots & \vdots & & \\ 0 & 1 & \cdots & 0 & 0 & a_{i-2} & \\ 1 & 0 & \cdots & 0 & 0 & a_{i-1} & \cdots \end{pmatrix}$$

We examine the matrix $M[i+1]$. We denote the entry in the upper-right corner as c, the entries below c as $a_0, \ldots, a_{i-1}$, and the entries to the left of c as $b_0, \ldots, b_{i-1}$. By examining these quantities, we can determine whether $M[i+1]$ is non-singular. If all a_i and b_i are zero,

and c in one, it is easily seen that $M[i+1]$ is non-singular. But what if this not the case? The fact that $M[i]$ is the mirror-identity makes it easy to cancel the a_i's and b_i's using row and column operations. $M[i+1]$ will be non-singular, if and only if, after these operations $c = 1$. If $a_j = 1$, add the column containing b_j to column $(i-1)$ of M. This changes a_j to zero. Similarly, if $b_j = 1$ and we add the row containing a_j to row $(p-1-i)$ of M, this changes b_j to zero. These operations may affect c as follows:

- If $a_j = b_j = 0$ there is no change to c.
- If $a_j = 1$, $b_j = 0$ there is no change to c.
- If $a_j = 0$, $b_j = 1$ there is no change to c.
- If $a_j = b_j = 1$ then c is flipped.

In the last case, both a_j and b_j are one. If we choose to cancel a_j first, the value of $b_j = 1$ is added to c, changing it from a one to a zero, or vice-versa. If we choose to cancel b_j first, the value of $a_j = 1$ is added to c, and c is again changed. In all other cases, we can cancel a_j and b_j without affecting c. The non-singularity of $M[i+1]$ will therefore depend on two factors: the initial value of c, and the number of flips. Counting the number of ways we get 0 flips, we find that we can do so in 3^i ways, because there are three ways each a-b pair can be assigned without causing a flip. There are $i3^{i-1}$ ways we can get one flip, and $i(i-1)3^{i-2}/2$ ways we can get two flips. In general, there are $\binom{i}{j}3^{i-j}$ ways we can get j flips. If c is initially zero, $M[i+1]$ is non-singular exactly when there are an odd number of flips. This can happen in $\sum_{0\le j\le i,\text{odd}i}\binom{i}{j}3^{i-j}$ different ways. If c is initially one, $M[i+1]$ is non-singular exactly when there are an even number of flips. This happens in $\sum_{0\le j\le i,\text{even}i}\binom{i}{j}3^{i-j}$ ways. The total number of ways is simply $\sum_{j=0}^{i}\binom{i}{j}3^{i-j}$.

We have assumed that all 4^i values of a's and b's are possible. We show that this assumption is valid, at all steps. The proof is by induction. Consider M after $M[i]$ has been transformed to the mirror-identity:

$$\begin{pmatrix}
x_{j-1,i-1} & \cdots & x_{j-1,0} & d_{j-1} & y_{j-1,0} & \cdots & y_{j-1,j-1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots \\
x_{0,i-1} & \cdots & x_{0,0} & d_0 & y_{0,0} & \cdots & y_{0,j-1} \\
b_{i-1} & \cdots & b_0 & c & e_0 & \cdots & e_{j-1} \\
0 & \cdots & 1 & a_0 & z_{0,0} & \cdots & z_{0,j-1} \\
\vdots & & \vdots & \vdots & \vdots & & \vdots \\
1 & \cdots & 0 & a_{i-1} & z_{i-1,0} & \cdots & z_{i-1,j-1}
\end{pmatrix}$$

where $j = p - i - 1$. Any previous operations could affect only the a's, b's, x's and z's. The values of c, the d's, e's and y's have not been changed. In canceling the a's and b's, we may change the values of the d's and e's. Our inductive hypothesis is that there exist initial entry values such that all values of the a's, b's, x's and z's are possible. This is true for

$i = 1$, because we have yet to perform any operations, all entries of M are still at their initial values. Assuming the inductive hypothesis for i, we show that it is true for $i + 1$. Consider an arbitrary d_k. The value of d_k may be changed when we cancel some b_ℓ by adding the row containing a_ℓ to the row containing c. This adds $x_{\ell,k}$ to d_k. Since all values of d_k and $x_{\ell,k}$ are arbitrary, it is easily seen that d_k can assume an arbitrary value after this operation. Since several $x_{\ell,k}$'s may be added to d_k, we extend this argument by induction to cover an arbitrary number of operations. We argue similarly for arbitrary e_k.

If there are S_i ways that $M[i]$ can be non-singular, then there are:

$$S_i \cdot \sum_{j=0}^{i} \binom{i}{j} 3^{i-j} = S_i \cdot (3+1)^i = S_i \cdot 4^i$$

ways that $M[i+1]$ can be non-singular. Combining this with our value for S_1 we have:

$$\begin{aligned} S_1 &= 1 \\ S_{p+1} &= S_p \cdot 4^p \end{aligned}$$

Since S_p is always a power of four, let $s_p = \log_4 S_p$. Then $s_0 = 0$ and $s_{p+1} = s_p + p$. We have:

$$\begin{aligned} s_p &= \sum_{i=0}^{p-1} i \\ &= \frac{(p-1)p}{2} \end{aligned}$$

Therefore, we have:

$$S_p = 4^{(p-1)p/2} = 2^{(p-1)p}$$

■

Note that the proof of this theorem implies a $\Theta(p^3)$ algorithm for determining the sub-non-singularity of a matrix. The naïve algorithm is to test each of the p sub-matrices independently. Since the best known algorithm for determining non-singularity takes more than $\Theta(n^2)$ time, the naïve algorithm would take more than $\Theta(p^3)$ time.

We now compare the number of non-singular matrices to the number of sub-non-singular matrices.

Theorem 4.5 *Let U_p be the number of $p \times p$ matrices over $\mathbb{Z}_2$ that are non-singular. Then:*

$$U_p = \prod_{i=1}^{p} (2^p - 2^{i-1}) \tag{5}$$

Proof If we are given a $p \times p$ non-singular matrix then the first column contains one or more non-zero entries. There are:

$$\sum_{i=1}^{p} \binom{p}{i}$$

ways that this can occur. Being that this is the case, pick some non-zero arbitrarily. Let a be the row containing this non-zero. Cancel the remaining non-zeros, using row operations.

Consider the second column. The entry in row a may be arbitrary. We must have at least one non-zero in some other row in this column. The number of ways this can occur is:

$$2\sum_{i=1}^{p-1}\binom{p-1}{i}$$

In general, considering the jth column, we will already have $j-1$ non-zeros in previous columns, each in a different row. For these rows, arbitrary entry values are allowed in column j. In the remaining rows, at least one non-zero must appear. So we have:

$$2^{j-1}\sum_{i=1}^{p-j+1}\binom{p-j+1}{i}$$

distinct sets of entries for this column which ensure non-singularity.

We now show that all these sets of entries are possible, the sequence of row operations performed do not eliminate any possibilities. Consider the column j symbolically. We denote the initial values in this column $a_0, a_1, \ldots, a_{p-1}$. The final values are $a'_0, a'_1, \ldots, a'_{p-1}$. Since we perform only row operations, each a'_k is the sum of a_k and some other a's. So each a'_k can be represented by a linear equation:

$$\begin{aligned} a'_0 &= b_{0,0}a_0 \oplus b_{0,1}a_1 \oplus \cdots \oplus b_{0,p-1}a_{p-1} \\ a'_k &= b_{k,0}a_0 \oplus b_{k,1}a_1 \oplus \cdots \oplus b_{k,p-1}a_{p-1} \\ a'_{p-1} &= b_{p-1,0}a_0 \oplus b_{p-1,1}a_1 \oplus \cdots \oplus b_{p-1,p-1}a_{p-1} \end{aligned}$$

And so given an assignment to the b's, the question is whether we can find a set of a values which result in a given arbitrary set of a' values. We can make this assignment if and only if the matrix $B = (b_{k,\ell})$ is non-singular. Suppose B is singular. Then we can perform some sequence of row operations which cancels some row ℓ of B. However, since we perform only row operations on the original matrix, this implies that if we perform this same sequence of operations on the original matrix, the entire row ℓ is canceled. This implies that the original matrix is singular.

Putting this all together, we have:

$$\begin{aligned} U_p &= \left[\sum_{i=1}^{p}\binom{p}{i}\right]\cdot\left[2\sum_{i=1}^{p-1}\binom{p-1}{i}\right]\cdots\left[2^{p-1}\sum_{i=1}^{1}\binom{1}{1}\right] \\ &= \prod_{j=1}^{p}2^{j-1}(2^{p-j+1}-1) \\ &= \prod_{j=1}^{p}(2^p - 2^{j-1}) \end{aligned}$$

■

Corollary 4.1 *The ratio of $p \times p$ non-singular matrices to matrices is:*

$$V_p = \prod_{i=1}^{p} \frac{2^i - 1}{2^i} = \frac{1}{2} \cdot \frac{3}{4} \cdots \frac{2^p - 1}{2^p} \tag{6}$$

Furthermore, V_p converges as p goes to infinity.

Proof Let $R_p = 2^{p^2}$. This is the number of distinct $p \times p$ matrices over $\mathbb{Z}_2$. We have:

$$\begin{aligned} \frac{U_p}{R_p} &= \frac{\prod_{i=1}^{p}(2^p - 2^{i-1})}{R_p} = \frac{\prod_{i=1}^{p} 2^p(1 - 2^{i-1-p})}{R_p} = \frac{2^{p^2} \prod_{i=1}^{p}(1 - 2^{i-1-p})}{R_p} \\ &= \prod_{i=1}^{p}(1 - 2^{i-1-p}) = \prod_{i=1}^{p}(1 - 2^{-i}) = \prod_{i=1}^{p}\left(\frac{2^i}{2^i} - \frac{1}{2^i}\right) \\ &= \prod_{i=1}^{p} \frac{2^i - 1}{2^i} \end{aligned}$$

To show that V_p converges, we use the fact that:

$$\prod_{i=1}^{\infty}(1 + a_i) \qquad a_i < 0 \text{ for all } i$$

will converge if and only if:

$$\sum_{i=1}^{\infty} a_i$$

converges. Since $(2^i - 1)/2^i = 1 - 2^{-i}$ we see that (6) will converge if and only if:

$$\sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^i$$

converges. This geometric series is known to converge to 1. ∎

We compute V_p for some small values of p:

p	V_p	p	V_p	p	V_p
1	0.5000000000	12	0.2888586115	23	0.2887881295
2	0.3750000000	13	0.2888233504	24	0.2887881123
3	0.3281250000	14	0.2888057220	25	0.2887881037
4	0.3076171875	15	0.2887969084	26	0.2887880994
5	0.2980041504	16	0.2887925017	27	0.2887880972
6	0.2933478355	17	0.2887902984	28	0.2887880962
7	0.2910560556	18	0.2887891967	29	0.2887880956
8	0.2899191179	19	0.2887886459	30	0.2887880954
9	0.2893528696	20	0.2887883705	31	0.2887880952
10	0.2890702984	21	0.2887882328	32	0.2887880952
11	0.2889291508	22	0.2887881639	33	0.2887880951

5 XOR-schemes and the Network

Suppose ϕ is an XOR-scheme for a set of templates. We need to know that a given template will not only be accessed conflict-free, but that the XOR-scheme will map each template instance in such a way that the network can perform the mapping. We say that ϕ is *network-contention-free* if and only if all template instances of all templates are mapped by ϕ in such a way that the network can perform the mappings.

Theorem 5.1 *Let ϕ be an XOR-scheme for a template set $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_i\}$, with matrix Φ. Then ϕ is conflict-free and network-contention-free for the inverted baseline network if and only if Φ^{T_i} is sub-non-singular for all i.*

Proof Consider ϕ restricted to some template $\mathcal{T}_i$. Then this restricted ϕ must be a linear permutation, if it is to be conflict-free. The matrix Φ^{T_i} must therefore be sub-non-singular. So Φ^{T_i} must be sub-non-singular for all i. ■

5.1 NP-Completeness

We show that the problem of finding an XOR-scheme which allows conflict-free and network-contention-free access for a given set of templates is tractable for $p = 2$, but NP-complete for $p > 2$. We consider the following abstract problem. We are given:

- A vector space $\mathcal{Z} = \mathbb{Z}_2^p$.
- A set of n variables $V = \{v_i \mid 0 \leq i \leq n-1\}$.
- A set T of p-tuples of variables, $T = \{(v_{i_0}, \ldots, v_{i_{p-1}}) \mid 0 \leq i_j \leq n-1\}$. Each v_i must appear in some tuple.

The vectors assigned to the variables of a tuple form the columns of the a $p \times p$ matrix. For example, if (v_0, v_1) is a tuple, and v_0 and v_1 are assigned $\binom{0}{1}$ and $\binom{1}{1}$, respectively, then the matrix corresponding to this tuple is $\binom{01}{11}$. We want to find an assignment of vectors in $\mathcal{Z}$ such that the matrices corresponding to all tuples are sub-non-singular. We call this problem Sub-Non-Singular Satisfaction (SNSS).

This problem can easily be solved for $p = 2$. A 2×2 matrix is sub-non-singular only if its lower left entry is non-zero. So if a variable appears as the first item in a tuple, that variable must be assigned one of two values, $\binom{0}{1}$ or $\binom{1}{1}$. If a variable appears as the second item in a tuple, and is not as the first item in any other tuple, then it may be assigned one of three values, $\binom{0}{1}$, $\binom{1}{0}$ or $\binom{1}{1}$. Any two variables which appear in the same tuple must be assigned different vectors, or the corresponding matrix will be singular. We build a graph to represent this problem. Each vertex in the graph represents a variable. There is an edge between two vertices if and only if the variables they appear together in some tuple. We call this the *conflict graph* of the problem. Let X be the set of vertices corresponding to

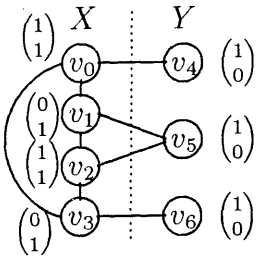


Figure 5: Example for SNSS with $p = 2$

variables that appear first in some tuples, and thus can be assigned only two values. Let Y be the remaining vertices, which can be assigned three values. No two vertices in Y are adjacent, because each edge corresponds to a matrix which must have a first column. Therefore all edges will be between two vertices in X, or between a vertex in X and one in Y. We two-color the vertices in X. The vertices in Y can all be colored with the third color. This coloring corresponds directly to assigning values to the variables. We call this algorithm XIB2. Construction of the conflict graph requires $O(t)$ time, and coloring requires $O(n)$ time, so the complexity of algorithm XIB2 is $O(n + t)$.

We illustrate this with a small example. Suppose we have variables $v_0, \ldots, v_6$ and:

$$\begin{aligned} T_1 &= (v_0, v_4), \quad T_2 = (v_1, v_5), \quad T_3 = (v_2, v_5), \quad T_4 = (v_3, v_6) \\ T_5 &= (v_0, v_1), \quad T_6 = (v_1, v_2), \quad T_7 = (v_2, v_3), \quad T_8 = (v_3, v_0) \end{aligned}$$

The set of vertices which appear in the first position of some tuple is $X = \{v_0, v_1, v_2, v_3\}$. The set of vertices which appear only in the second position of tuples is $Y = \{v_4, v_5, v_6\}$. The conflict graph of this problem is illustrated in Figure 5. One possible coloring is shown in the figure.

Theorem 5.2 *Sub-Non-Singular Satisfaction is NP-complete for $p = 3$.*

Proof Given an assignment of vectors to variables, we can easily verify that each tuple corresponds to a sub-non-singular matrix, in polynomial time. So SNSS is in NP.

We now show that SNSS for $p = 3$ is NP-hard. The proof is by reduction from four-coloring [12]. Let G be an arbitrary graph. Let n be the number of vertices in G, and m be the number of edges. We wish to know if G has a four-coloring. Construct a variable for each vertex. The variables $v_0, \ldots, v_{n-1}$ correspond directly to the vertices $v_0, \ldots, v_{n-1}$. Construct also $2m$ 'slack' variables, $s_0, \ldots, s_{2m-1}$. For each edge $e_i = (v_j, v_k)$ in G, we construct two tuples (v_j, s_{2i}, v_k) and (v_k, s_{2i+1}, v_j). Note that since both v_j and v_k are the first item in a tuple, they both must be assigned vectors which have a non-zero in their bottom entry. So the matrices corresponding to the two tuples constructed for e_i are:

$$E_i = \begin{array}{c} \begin{matrix} v_j & s_{2i} & v_k \end{matrix} \\ \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ 1 & ? & 1 \end{pmatrix} \end{array} \quad \text{and} \quad E_i' = \begin{array}{c} \begin{matrix} v_k & s_{2i+1} & v_j \end{matrix} \\ \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ 1 & ? & 1 \end{pmatrix} \end{array}$$

The vectors that may be assigned to v_j and v_k are:

$$z_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad z_1 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \quad z_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad z_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Note that v_j and v_k will be linearly independent if and only if they are assigned different vectors from this set. Furthermore, this is the only restriction that must hold between v_j and v_k.

We show that assigning values to the variables $v_0, \ldots, v_{n-1}, s_0, \ldots, s_{2m-1}$, subject to the above restriction, corresponds exactly to four-coloring G. Suppose G is four-colorable, then if vertex v_i has color j we assign z_j to vector v_i. The slack variables can then easily be assigned, since each slack variable appears in only one tuple. Suppose we have an assignment of vectors to variables, then if variable v_i is assigned z_j we assign vertex i color j. The coloring is proper, since if two adjacent vertices have the same color this implies that the matrix corresponding to the edge between them is non-singular. Since SNSS can be used to solve any four-coloring problem, via a polynomial time transformation, SNSS is NP-hard. ■

Corollary 5.1 *Sub-Non-Singular Satisfaction is NP-complete for all $p \geq 3$.*

Proof (Sketch) We can easily generalize the proof for $p = 3$, so that in general an SNSS problems corresponds exactly to an arbitrary 2^{p-1}-coloring problem. Each edge in an arbitrary graph corresponds to two $p \times p$ matrices. The first and last columns of these matrices correspond to the colors of the vertices incident on that edge. The middle columns of the matrices are slack columns. ■

6 Heuristic Approach to SNSS

We present an efficient algorithm for finding an XOR-scheme for a given template set, if one exists. This algorithm outperforms naïve backtracking. Further, by not allowing the algorithm to backtrack, we derive a practical algorithm for finding approximate XOR-schemes, XOR-schemes in which some templates may not be accessed conflict-free.

The idea is to construct the matrix of the XOR-scheme one row at a time, from the bottom up. We assume that, for each template, the matrix $\Phi^{T_i}[j]$ is sub-non-singular, and attempt to construct the row $\Phi_{p-j,*}$ so that each $\Phi^{T_i}[j+1]$ is sub-non-singular. We use algorithm XIB2, along with the idea from the proof of Theorem 4.4, in the construction.

We illustrate the algorithm by an example with $p = 3$. Suppose we are given the template set:

$$\begin{aligned} T_1 &= \{v_0, v_1, v_2\}, \quad T_2 = \{v_1, v_2, v_3\} \\ T_3 &= \{v_3, v_4, v_5\}, \quad T_4 = \{v_1, v_3, v_4\} \end{aligned}$$

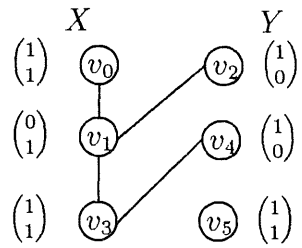


Figure 6: Example

We construct the bottom two rows of Φ using algorithm XIB2. For each Φ^{T_i}, we wish to ensure that the 2×2 sub-matrix in its lower-left corner is sub-non-singular. We can accomplish this by constructing a matrix for the reduced template set:

$$\begin{aligned} T_1' &= \{v_0, v_1\}, \quad T_2' = \{v_1, v_2\} \\ T_3' &= \{v_3, v_4\}, \quad T_4' = \{v_1, v_3\} \end{aligned}$$

Note that T_i' is just T_i with the highest ordered vector removed. The vectors appearing first in some template T_i' are $X = \{v_0, v_1, v_3\}$. The vectors appearing only second in some template (or in no template) are $Y = \{v_2, v_4, v_5\}$. The conflict graph, and one possible coloring, are shown in Figure 6. Since v_5 does not appear in any T', we arbitrarily assign it $\binom{1}{1}$. The bottom two rows of our matrix have been determined. We let $x_0, \ldots, x_5$ be the values in the top row:

$$\Phi = \begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{pmatrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

We must now ensure that each 3×3 matrix is non-singular. So considering for instance $\mathcal{T}_1$, we must assign x_0, x_1, and x_3 in such a way that the matrix:

$$\Phi^{T_1} = \begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 \end{matrix} \\ \begin{pmatrix} x_0 & x_1 & x_2 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

is non-singular. The first step is to get the mirror identity matrix in the lower right 2×2 sub-matrix, using only row operations. If column operations are used, the values of the x's are affected. We achieve this in our example by adding the second row to the third:

$$\hat{\Phi}^{T_1} = \begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 \end{matrix} \\ \begin{pmatrix} x_0 & x_1 & x_2 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \tag{7}$$

Note that we do not want to change the original matrix, and so we denote this row reduced matrix $\hat{\Phi}^{T_1}$. We now have the matrix in the format specified by the theorem. In terms of Theorem 4.4, we have:

$$\begin{aligned} b_0 &= x_1, \quad b_1 = x_0 \\ c &= x_2 \\ a_0 &= 1, \quad a_1 = 1 \end{aligned}$$

Matrix (7) will be non-singular, and thus sub-non-singular, if $c = 1$ and the number of pairs $a_i = b_i = 1$ is even, or if $c = 0$ and the number of pairs $a_i = b_i = 1$ is odd. Since the values of the a's are fixed, we can express this as a linear equation over $\mathbb{Z}_2$.

$$b_0 \oplus b_1 \oplus c = 1$$

Or, in terms of x's:

$$x_0 \oplus x_1 \oplus x_2 = 1$$

Row reducing Φ^{T_2} we get:

$$\begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 \end{matrix} \\ \begin{pmatrix} x_1 & x_2 & x_3 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

And so for Φ^{T_2} to be non-singular, we must have:

$$x_1 \oplus x_2 \oplus x_3 = 1$$

The remaining conditions are:

$$\begin{aligned} x_3 \oplus x_5 &= 1 \\ x_1 \oplus x_3 \oplus x_4 &= 1 \end{aligned}$$

One solution for this system of simultaneous equations is:

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 0, \quad x_3 = 0, \quad x_4 = 0, \quad x_5 = 1$$

So the final matrix is:

$$\Phi = \begin{array}{c} \begin{matrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

In general, the algorithm is:

1. Determine the bottom two rows of the matrix using algorithm XIB2.

2. Create each remaining row, working from the bottom up.
   For i in 2 to $p-1$ loop:

   (a) For each template $\mathcal{T}_j$ do:

      i. Obtain a matrix $\hat{\Phi}^{T_j}$ by reducing the matrix Φ^{T_j} so that it has the mirror identity matrix in its lower-left corner, using only row operations. Operations do not affect the matrix Φ.

      ii. Use the ith column of this matrix to determine the equation associated with this template. Let the basis of $\mathcal{T}_j$ be $v_{\ell_0}, \ldots, v_{\ell_{p-1}}$, and $y_k = \hat{\Phi}^{T_j}_{p-k-1,\ell_i}$. Then the equation is:

$$x_{\ell_i} \oplus \bigoplus_{k=0}^{i-1} x_{\ell_k} y_k = 1 \tag{8}$$

   (b) Solve the system of simultaneous equations. Assign entry $\Phi_{p-i-1,k}$ the value x_k.

We call this algorithm XIB. Note that we do not have to row reduce from scratch each time we perform Step i. If we have available the result of the previous iteration, we can row reduce this partially reduced matrix, and reduce the time complexity of this step from $O(p^3)$ to $O(p^2)$. There are several points to note about algorithm XIB:

- In Step 1, there may be several different ways which algorithm XIB2 could color the conflict graph. The selection of a coloring may affect the ability of XIB to find a solution in Step 2.
- In Step (b), the set of equations may not have a unique solution. The selection of a solution may affect the ability of the algorithm to find a solution for a later row.

Since there are potentially several alternatives at Steps 1 and (b), one possibility is to use backtracking to exhaustively search for a solution. It is also possible to use heuristics to guide the selection at these steps, with or without backtracking. We consider only non-backtracking solutions, since the high cost of backtracking prohibits its use in practical systems.

6.1 Analysis

We analyze the time complexity of algorithm XIB, in the case where no backtracking is allowed.

Algorithm XIB makes an initial call to algorithm XIB2, and then runs in $p-2$ phases. Let t be the number of templates. Each phase constructs t equations. Constructing each equation requires $O(p^2)$ time, if we use the partially reduced matrix from the previous phase. The total construction time is $O(tp^2)$. Solving the resulting set of simultaneous equations can be done in $O(tn^2)$ time, since we have t equations in n unknowns. Since $p \leq n$, the total time required for a phase is $O(tn^2)$. The total complexity of algorithm XIB is therefore $O(ptn^2)$, where t, 2^p, and n, are the number of templates, the number of processors, and the number of distinct vectors of the template bases, respectively.

6.2 Approximate Solutions

Suppose that a conflict-free and network-contention-free XOR-scheme cannot be found. We wish to find an XOR-scheme which minimizes the 'amount' of conflict. More precisely, each of our templates is given a weight, and we wish to find an XOR-scheme where the sum of the weights of the violated templates is minimized. There are two places were algorithm XIB can fail to find a solution, Steps 1 and (b).

Suppose that algorithm XIB fails in Step 1. This occurs if the subgraph X is not two-colored. In this case, we wish to find an approximate two-coloring of X. We simplify the situation by assuming that each template has a weight of 1. Each edge in the graph corresponds directly to some template, and so to minimize the weight of violated templates, we wish to find a two-coloring of X which violates the minimum number of edges. Garey, Johnson, and Stockmeyer [7] have shown this problem to be NP-complete.

Suppose that algorithm XIB fails in Step (b). This occurs if a solution to a set of equations is not found. In this case, we wish to find an approximate solution to the set of equations. Each equation in the set corresponds directly to some template, and so we wish to find an approximate solution which violates the minimum weighted set of equations. We show that this problem is NP-hard: We are given:

- A field F.
- A set of variables $V = \{v_0, \ldots, v_{n-1}\}$.
- A set of simultaneous linear equations over V, each with a weight.

The problem is to find an approximate solution to the equations, in which the sum of the weights of the violated equations is minimized. We call the problem Approximate Linear Solution (ALS).

Theorem 6.1 *Approximate Linear Solution is NP-hard.*

Proof We can use an algorithm for ALS to solve the problem of finding an optimal approximate two-coloring, which is NP-complete [7]. Suppose we are given an arbitrary graph G, and we wish to find an approximate two-coloring which violates the minimum number of edges. Let $F = \mathbb{Z}_2$. For each vertex v_i of G we create a variable v_i. For each edge (v_i, v_j), we create an equation:

$$v_i \oplus v_j = 1$$

and give it weight 1. It is easily seen that an optimal approximate solution to the resulting system of equations corresponds directly to an optimal approximate two-coloring of G. ■

And so finding a best approximate XOR-scheme is impractical. We need a metric by which approximate XOR-schemes can be judged. An XOR-scheme will be network-contention-free if the XOR-scheme is an onto mapping at each stage of the network, for all templates. Consider the mapping of a single template. If the template is mapped network-contention-free assign it a cost of one. If contention occurs at one stage, then every switch of

that stage is mapping both of its inputs onto one of its outputs. The inputs must be serialized, and thus its cost is two. In general, if contention occurs at c stages the cost is 2^c. Given the matrix of an XOR-scheme restricted to some template $M = \Phi^T$, we must determine the number of stages of the network at which contention will occur. If contention first occurs at stage i, then $M[i]$ has rank less than i. Contention will occur at stage $i+1$ if and only if the rank of $M[i+1]$ is not greater than that of $\Phi^T[i]$. If the rank of each subsequent matrix is greater than its predecessor, no further contention will occur. In general, at each stage j where contention occurs, we have $\text{rank}(M[j]) = \text{rank}(M[j-1])$. We define $\text{rank}(M[0]) = 0$. Let:

$$C_i = \begin{cases} 0 & \text{if } \text{rank}(M[j]) = \text{rank}(M[j-1]) \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

We define the *subrank* of a $p \times p$ matrix M as follows:

$$\text{subrank}(M) = \sum_{i=1}^{p} C_i \tag{10}$$

Suppose we are given a weight for each template basis. This weight reflects the frequency of access of the template. Such information might be determined from profiling information, or be estimated by the compiler. The cost of an XOR-scheme Φ for a set of templates with bases $T_1, \ldots, T_t$, and weights $w_1, \ldots, w_t$ is:

$$\text{cost}(\Phi) = \sum_{i=1}^{t} w_i 2^{p - \text{subrank}(\Phi^{T_i})} \tag{11}$$

In particular, the cost of an XOR-scheme that is network-contention-free will be $\sum w_i$.

6.3 A Randomized Algorithm

Consider the following implementation of algorithm XIB:

- In Step 1, if more than one possible coloring of the graph exists, select a coloring randomly (all colorings are equally likely). If no coloring exists, create an approximate two-coloring, by alternately coloring vertices along a random walk through the graph.
- In Step (b), if more than one solution exists, pick one randomly. If no solution exists, pick an approximate solution randomly.
- Algorithm XIB is performed repeatedly, until an XOR scheme within pre-set performance parameters is found, or an iteration limit is reached. In the later case, the best XOR-scheme found is used.

We call this randomized algorithm RXIB.

t	$p=3$	$p=4$	$p=5$	$p=6$
3	100.0	100.0	100.0	100.0
4	99.8	99.8	99.7	98.8
5	99.4	98.9	97.4	96.8
6	98.5	96.7	93.9	88.9
7	96.2	94.7	89.0	82.1
8	95.7	88.1	78.8	67.7
9	89.3	81.5	68.4	53.5
10	86.2	73.8	55.7	34.9
11	83.0	65.6	43.4	21.4
12	76.1	50.9	29.7	8.7

Figure 7: Percentages of cases where an optimum XOR-scheme was found

7 Performance Evaluation

Evaluation is carried out by: 1) statistically testing the proposed approach and comparing to other schemes by considering the amount of memory and network contention as the performance function, and 2) comparing our methodology to other proposals.

7.1 Experimental evaluation

We implemented RXIB, and tested it on randomly generated sets of template bases. We iterated until a network-contention-free XOR-scheme was found, or the limit of ten tries was exceeded. The number of processors p ranges from 2^3 to 2^6, and the number templates t ranges from 3 to 12. One thousand cases were generated for each combination of these parameters. All templates were given a weight of 1. Two criteria are presented. First, the percentages of cases where an optimum solution was found are displayed in Figure 7, i.e. a memory and network contention-free XOR-scheme was found. Optimum access time for any given template is one cycle; optimally accessing t templates requires t cycles.

Second, average percent deviations of the cost of the best XOR-scheme found from the cost of a memory and network contention-free scheme is displayed in Figure 8. In other words, if c is the cost of best XOR-scheme found for a particular template set, the deviation is $100 \cdot (c/t - 1)$. The deviations displayed in the figure are averaged over one thousand cases. Our scheme finds near optimum solutions for small numbers of templates and moderate numbers of processors. For the cases where the optimum solution is not found, the average deviation of our scheme from the optimum access time is moderate in all studied cases. The degradation smoothly increases with either increasing the number of templates or the number of processors.

In a second experiment, testing was carried out with respect to combining power of 2 stride access. Template sets, consisting of templates representing strides $2^0, \ldots, 2^t$, were created for t ranging from 1 to 6, and p ranging from 3 to 6. Algorithm XIB found a memory

t	$p=3$	$p=4$	$p=5$	$p=6$
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.1	0.3
5	0.1	0.2	0.5	0.7
6	0.2	0.6	1.0	2.0
7	0.6	0.8	1.6	2.9
8	0.6	1.6	3.1	5.2
9	1.3	2.3	4.2	7.1
10	1.5	3.0	5.7	11.5
11	1.7	3.9	7.8	14.5
12	2.3	5.4	9.9	19.4

Figure 8: Average percentage increasing over the optimum access time

and network-contention free XOR-scheme for all of the above combined sets of strides.

Comparison of the proposed scheme was carried out with respect to row-major interleaving (INT) and to a static-storage-scheme (SSS) that is optimized for a reference set of templates: row, column, and both diagonals [4]. These schemes were tested in similar conditions to our approach as stated above. The INT scheme causes the average access time to be 6, 9.37, 13.59, and 18.64 fold the optimum access time for 2^3, 2^4, 2^5, and 2^6 processors, respectively. Similarly, the SSS scheme causes the average access time to be 4.23, 5.31, 5.79, and 5.84 fold the optimum access time for the same number of processors. No significant dependence over the number of templates was observed for INT and SSS. By comparing to the previous schemes, our proposed XOR-matrix significantly minimizes the memory and network contention for arbitrary data templates under the studied conditions.

7.2 Comparison to other Contributions

In most related research [6], network aspects are not considered and the problem is reduced to conflict-free access to parallel memories. The work presented in [6] has the merit of finding the necessary and sufficient condition for conflict-free access of parallel memories for one template but no method is presented for finding the XOR-scheme in case of composite templates nor are the network issues addressed.

For SIMD vector processors that inherently use pipelined bus architecture, the problem [9, 10, 19] is to improve multi-stride access to interleaved memories. These approaches deal with memory organization and memory buffering in order to maximize memory throughput. A general method is proposed in [9, 10] for finding the dynamic address transformation that optimizes the access for one stride and increases the throughput for other strides compared to low order interleaving. In [19], similar transformation is proposed through the use of all the address bits and a XOR-matrix so that its most right square sub-matrix is of full rank.

In our case, we have presented an efficient approach for finding XOR-schemes for combined data templates. The XOR-matrix is defined so that each of its columns corresponds

to a distinct vector of the basis of the template set, i.e. non-redundant representation. This methodology shows how arbitrary composite templates can be handled by a unique XOR-matrix for which the condition to conflict-free access can be easely formulated. While it is simple to find the XOR-matrix of one template, we proved that finding combined XOR-scheme is an NP-complete problem.

In some other contributions where the network aspects are considered, the problem is restricted to finding dedicated XOR-schemes for a well defined set of templates [4, 17]. For example, minimizing memory and network contention for a subset of rows, columns, diagonals, and square blocks was proposed in [4]. This leads to XOR-schemes that are optimized for a given set of reference templates. In [17], network contention has been analyzed with respect to conflict-free access to a fixed set of strides. This enables finding the XOR-matrix to be used as part of the processor's address translation. Our proposed method has the advantage of being a general approach incorporating both memory and network requirements in a XOR-scheme. Heuristic approaches have been proposed to minimize the global conflict in accessing arbitrary data templates. While only the Baseline network is considered here, all other multi-stage networks require similar characterization [4].

The proposed method for synthesizing XOR-schemes can be used as a compiler tool to generate dynamic address translation matrices to be used with array referencing. By identifying the accessed templates at compile-time, each array is associated an XOR-matrix that is optimized for a given type of data templates. Within our assumptions, this general approach may dramatically reduce contention in SIMD systems.

8 Conclusions

We investigated the problem of finding general XOR-schemes to dynamically minimize memory and network contention in accessing arrays with arbitrary data templates in SIMD computers.

To enable the use of multi-stage networks, we assumed that the number of processors and memories is a power of 2. This approach requires previous knowledge of the accessed templates for each array. The proposed XOR-scheme is based on a non-redundant representation, each column of the XOR-matrix corresponds to a distinct vector of the template bases. A flexible XOR-matrix scheme was defined for combined data templates.

Characterization of the Baseline multi-stage network was presented, with respect to the requirements on the address transformation. The notion of non-singular binary matrices has been extensively analyzed because of its critical importance towards achieving conflict-free access to both memory and multi-stage networks. Each restriction of the XOR-matrix to a given template should be sub-non-singular in order to guarantee conflict-free access. We proved that finding the XOR-matrix for accessing arbitrary data templates is an NP-complete problem.

To minimize memory and network contention, a heuristic algorithm was proposed for finding the XOR-matrix with the above stated constraints for each template restriction. The heuristic operates on successive rows and propagates the constraints of each template to the

next row. The row solution of the XOR-matrix was found by global constraint satisfaction along that row. To avoid backtracking, a randomized approach is used. Evaluation shows that the proposed XOR-schemes significantly reduce the memory and network contention compared to interleaving and XOR-schemes that are optimized for a set of static reference templates.

The contributions of this work are: 1) a general and compact XOR-scheme for combined data templates, 2) characterization of necessary and sufficient conditions for conflict free-access of memory and network, 3) an efficient algorithm for automating the process of finding the combined XOR-matrix.

By applying the proposed approach at the compiler level, significant speedup is expected compared to the traditional memory interleaving technique and other static schemes. Future extension may address the problem of finding more efficient heuristics to further reduce the average deviation from the optimum solution. Reducing the time complexity of the heuristic is interesting to speedup compile-time processing. Extending the present approach to arbitrary multi-stage networks and other type of networks is one of our objectives.

9 Acknowledgments

Thanks to Professor Daniel Hirschberg for listening critically to the various proofs and for his remarks concerning the presentation of this report.

References

[1] BAILEY, D. Vector computer memory bank contentions. *IEEE Transactions on Computers C-36* (Mar 1987), 293–298.

[2] BATCHER, K. The multidimensional access memory in STARAN. *IEEE Transactions on Computers C-26* (Feb 1977), 174–177.

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

[4] BOPPANA, R. V., AND RAGHAVENDRA, C. S. Efficient storage schemes for arbitrary size square matrices in parallel processors with shuffle-exchange networks. In *Proceedings of the International Conference on Parallel Processing* (1991), pp. 365–368.

[5] BUDNIK, P., AND KUCK, D. The organization and use of parallel memories. *IEEE Transactions on Computers C-20*, 12 (Dec 1971), 1566–1569.

[6] FRAILONG, J. M. J. W., AND LENFANT, J. XOR-schemes: A flexible data organization in parallel memories. In *Proceedings of the International Conference on Parallel Processing* (1985), pp. 276–283.

[7] GAREY, M. R., JOHNSON, D. S., AND L., S. Some simplified NP-complete graph problems. *Theoretical Computer Science 2* (1976), 237–267.

[8] GUPTA, R., AND SOFFA, M. L. Compile-time techniques for improving scalar access performance in parallel memories. *IEEE Transactions on Parallel and Distributed Systems 2*, 2 (Apr 1991), 138–148.

[9] HARPER III, D. T. Block, multistride vector, and FFT accesses in parallel memory systems. *IEEE Transactions on Parallel and Distributed Systems 2*, 1 (Jan 1991), 43–51.

[10] HARPER III, D. T. Increased memory performance during vector accesses through the use of linear address transformations. *IEEE Transactions on Computers 41*, 2 (Feb 1992), 227–230.

[11] HARPER III, D. T., AND JUMP, J. Vector access performance in parallel memories using a skewed storage scheme. *IEEE Transactions on Computers C-36*, 12 (Dec 1987), 1440–1449.

[12] KARP, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, 1972, pp. 85–103.

[13] KUCK, D. J., AND SAMEH, A. H. Parallel computation of eigenvalues of real matrices. *Information Processing 71, North Holland* (1972), 1266–1272.

[14] LAWRIE, D. Access and alignment of data in an array processor. *IEEE Transactions on Computers C-24*, 12 (Dec 1975), 1145–1155.

[15] LAWRIE, D., AND VORA, C. The prime memory system for array accesses. *IEEE Transactions on Computers C-31*, 12 (May 1982), 435–442.

[16] LENFANT, J. Parallel permutations of data: a benes network. *IEEE Transactions on Computers C-27* (Jul 1978), 637–647.

[17] NORTON, A., AND MELTON, E. A class of boolean linear transformations for conflict-free power-of-two stride access. In *Proceedings of the International Conference on Parallel Processing* (1987), pp. 247–254.

[18] SIEGEL, H. J. Interconnection networks for SIMD machines. *Computer*, 12 (Jun 1979), 57–67.

[19] SOHI, G. S. High-bandwidth interleaved memories for vector processors–A simulation study. *IEEE Transactions on Computers 42*, 1 (Jan 1993), 34–44.