A Compiler Address Transformation for Conflict-Free Access of Memories and Networks

Mayez Al-Mouhamed(1) Lubomir Bic (2) and Hussam Abu-Haimed (3)

(1, 3) Department of Computer Engineering
King Fahd University of Petroleum and Minerals,
Dhahran 31261, Saudi Arabia (Email: mayez@ccse.kfupm.edu.sa)

(2) Department of Information and Computer Science
University of California
Irvine, CA 92717, USA.

Abstract: A method for mapping arrays into parallel memories so that to minimize serialization and network conflicts for lock-step systems wall be presented. Each array is associated an arbitrary number of data access patterns that can be identified following compiler data-dependence analysis. Conditions for conflict-free access of parallel memories and network are derived for arbitrary power-of-2 data patterns and arbitrary mutistage networks. We propose an efficient heuristic to synthesize combined address transformation (NP complete) which applies to arbitrary linear patterns, arbitrary multistage networks, and arbitrary number of power-of-2 memories. Our method can be implemented us part of the address transformation (Xor and And) or through compiler emulation. Performance of optimized storage schemes is presented for FFT, arbitrary sets of data patterns, non power-of-2 stride access in vector processors, interleaving, and static row-column storages. Our approach is profitable in all the above cases and provides a systematic method for converting array-memo y mapping and network aspects of algorithms from one network topology to another.

A Compiler Address Transformation For Conflict-Free Access of Memories and Networks

Mayez Al-Mouhamed
Comp. Eng. Dept.
King Fahd Univ.
Dhahran , Saudi Arabia

Lubomir Bic
Dept. of Info. and Comp. Sc.
UC Irvine
Irvine, CA 92717

Hussam Abu-Haimed
Comp. Eng. Dept.
King Fahd Univ.
Dhahran, Saudi Arabia

Abstract

A method for mapping arrays into parallel memories so that to minimize serialization and network conflicts for lock-step systems will be presented. Each array is associated an arbitrary number of data access patterns that can be identified following compiler data-dependence analysis. Conditions for conflict-free access of parallel memories and network are derived for arbitrary power-of-2 data patterns and arbirary mutistage networks. We propose an efficient heuristic to synthesize combined address transformation (NP-complete) which applies to arbitrary linear patterns, arbitrary multistage networks, and arbitrary number of power-of-2 memories. Our method can be implemented as part of the address transformation (Xor and And) or through compiler emulation. Performance of optimized storage schemes is presented for FFT, arbitrary sets of data patterns, non power-of-2 stride access in vector processors, interleaving, and static row-column storages. Our approach is profitable in all the above cases and provides a systematic method for coverting array-memory mapping and network aspects of algorithms from one network topology to another.

1 Introduction

The serialization of memory accesses is a major limiting factor in high performance SIMD computers. Conflict-free access [2] to rows, columns, and diagonals of arrays was proposed on the basis of row rotations. The drawbacks are the dependence on the array size, the number of memories, and the complex address transformation. The use of a prime number of memories [5] significantly outperforms interleaving but requires expensive address translation.

Based on *skew storage*, *XOR-schemes* were proposed [7, 4] for eliminating most of the above problems. The scheme can be efficiently used for power-of-2 strides but other strides can also be accessed through the use of few buffers at the memory inputs and outputs. The buffers reduce the effects of transient degradation in pipelined memories.

Linear permutations [3, 6] for mutistage networks have been studied by using non-singular (NS) matrices. 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 [3] based on composite linear permutations was proposed for the Omega network.

Our objective is to find a storage scheme that combines the constraints of composite patterns and the network in synthesizing dynamic storages so that memory and network contentions are minimized.

Sections 2 and 3 presents linear permutations and storage schemes. Sections 4 and 5 presents properties of combined storages. Sections 6 and 7 present one general example and application to FFT. Sections 8 and 9 present application to general patterns and stride access. Section 10 concludes this work.

2 Linear permutations and networks

In a multistage network, routing a source $s = s_{n-1} \dots s_0$ to destination $d = d_{n-1} \dots d_0$ consists of finding a path of switches that connect s to d.

In an Ω_n network, the position of the message at the input is $pos_0(s, d) = s_{n-1} \dots s_0$. we can easily find the position of the message at the output of the ith stage:

$$pos_i(s, d) = s_{n-i-1} \dots s_0 d_{n-1} \dots d_{n-i+1} d_{n-i}$$

We can similarly find the position of the message for other networks. A network input is denoted by $s = (s_{n-1}, \dots, s_0) \in S$ and output is denoted by

$d = (d_{n-1}, \dots, d_0) \in S$. This applies to n-stage networks. A linear permutation is a function $M : S \to S$ for which each source $s \in S$ maps into destination $d = Ms = (d_{n-1}, \dots, d_0)$ where d_i is a linear combination of the bits of s by using the logical *AND* and *XOR* operators as the multiplication and addition, respectively.

We wish to know under what condition an Ω_n network can perform permutation M. We shall abbreviate $pos_i(s, Ms)$ to $pos_i(s)$ which can be written as a matrix product $\tilde{M}[i]s$:

$$\mathrm{pos}_i(s) = (s_{n-i-1} \cdots s_0 d_{n-1} \cdots d_{n-i}) = \tilde{M}[i](s_{n-1} \cdots s_{n-i} s_{n-i-1} \cdots s_0) \tag{1}$$

where $\tilde{M}[i]$ is

$$\begin{pmatrix} 0 & \cdots & 0 & 1 & \cdots & 0 \\ . & \cdots & . & 0 & \cdots & . \\ . & \cdots & . & . & \cdots & . \\ . & \cdots & . & . & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 1 \\ a_{n-1,n-1} & \cdots & a_{n-1,n-i} & a_{n-1,n-i-1} & \cdots & a_{n-1,0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-i,n-1} & \cdots & a_{n-i,n-i} & a_{n-i,n-i-1} & \cdots & a_{n-i,0} \end{pmatrix} \tag{2}$$

Where the $(n-i) \times i$ matrix in the upper-left corner is formed by 0s, the $(n-i) \times (n-i)$ matrix in the upper-right corner is the identity, the $i \times (n-i)$ matrix in the lower-right corner will be denoted by $B[i]$, and the $i \times i$ matrix in the lower-left corner will be denoted by $M[i]$:

$$\mathrm{M}[i] = \begin{pmatrix} a_{n-1,n-1} & \cdots & a_{n-1,n-i} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ a_{n-i,n-1} & \cdots & a_{n-i,n-i} \end{pmatrix} \tag{3}$$

The permutation matrix M that gives the position $pos_n(s) = d = Ms$ of the message at the output of the nth stage is defined as follows:

$$\begin{matrix} a_{n-1,n-1} & . & a_{n-1,n-i} & a_{n-1,n-i-1} & . & a_{n-1,0} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ a_{n-i,n-1} & . & a_{n-i,n-i} & a_{n-i,n-i-1} & . & a_{n-i,0} \\ a_{n-i-1,n-1} & . & a_{n-i-1,n-i} & a_{n-i-1,n-i-1} & . & a_{n-i-1,0} \\ . & . & . & . & . & . \\ a_{0,n-1} & . & a_{0,n-i} & a_{0,n-i-1} & . & a_{0,0} \end{matrix} \tag{4}$$

Note that $M[i]$ is the $i \times i$ sub-matrix in the upper-left corner of M. We present a number of Theorems that characterize permutations. The proof of these Theorems can be found in[1].

Theorem 1 *All inputs of the ith stage map one-to-one to all outputs of the same stage if and only if $M[i]$ is NS.*

We now characterize linear permutations M which the Omega network can perform. An $n \times n$ permutation matrix M is said to be *Strongly-Non-Singular* (SNS) if and only if its restriction $M[i]$ is NS for arbitrary i, where $1 \le i \le n$.

Theorem 2 *A linear permutation M defined over Z_2^n can be performed by Ω_n if and only if M is SNS for Ω_n.*

We generalize the above result to an arbitrary network that belongs to the class of dynamic, full access, unique path, multistage networks.

Theorem 3 *An arbitrary dynamic, full access, unique path, multistage network (γ) can achieve arbitrary linear permutation defined by an $n \times n$ boolean matrix M if and only if M is SNS for γ.*

Note that the position of sub-matrices $M[i]$ can be different for each type of network. For the Omega network, the NS submatrices are located in the upper-left corner of M. For the Baseline network, the NS sub-matrices start at the lower-left corner of M.

We can define a permutation $\tilde{d} = Ms \oplus x$ which we call *Complement-Permutation*, where x is a constant.

Theorem 4 *An arbitrary complement-permutation $\tilde{d} = Ms \oplus x$ passes a multistage network if and only if M is SNS for the network.*

It can also be proved that the set of all complement-permutations associated with all SNS matrices M are all distinct for any multistage network.

3 Characterization of SNS matrices

We now evaluate the number R_n of $n \times n$ SNS matrices. Since R_n is identical for each type of network, we use the notation for Ω_n. It is obvious that $R_1 = 1$. We define algorithm (H-2) that finds and assigns the set of 2×2 matrices which are SNS

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

and so $R_2 = 4$. We now show that:

Theorem 5 *The number of permutations $\tilde{d} = Ms \oplus x$ that an arbitrary n-stage multistage network can perform is $T_n = 2^{n^2}$, where M is an $n \times n$ SNS matrix and x is an $n \times 1$ arbitrary vector.*

Proof We first find the number R_n of $n \times n$ SNS matrices and evaluate the number of permutations $\tilde{d} = Ms \oplus x$ for all values of x.

Suppose we are given an arbitrary $n \times n$ matrix M. Assume that $M[i]$ is SNS. Then by performing row and column operations, we can transform M such that $M[i]$ is the identity matrix thus:

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

We examine the matrix $M[i+1]$. We denote the entry in the lower-right corner as c, the entries above c as $a_0 \ldots a_{i-1}$, and the entries to the left of c as $b_0 \ldots b_{i-1}$. We can determine whether $M[i+1]$ is NS. If all a_i and b_i are zero and c in one, it is easily seen that $M[i+1]$ is NS. For the general case, the fact that $M[i]$ is the identity makes it easy to cancel the a_i's and b_i's using row and column operations. $M[i+1]$ will be NS if and only if after these operations $c = 1$. If $a_j = 1$, add the column containing b_j to the column containing c of M. This changes a_j to zero and c becomes $c \oplus b_j$. Similarly, if $b_j = 1$ and we add the row containing a_j to the row containing c of M, this changes b_j to zero and c becomes $c \oplus a_j$. These operations may affect c as follows: 1) c does not change if $a_j = b_j = 0$ or $a_j \oplus b_j = 1$, or 2) c is flipped if $a_j = b_j = 1$.

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. 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, the total number of ways is simply $\sum_{j=0}^{i} \binom{i}{j} 3^{i-j}$.

If there are R_i ways that $M[i]$ can be NS, then there are $R_i \sum_{j=0}^{i} \binom{i}{j} 3^{i-j} = R_i(3+1)^i = R_i 4^i$ ways that $M[i+1]$ can be NS. Combining this with our value for R_1 we have $R_1 = 1$ and $R_{n+1} = R_n 4^n$. Therefore, we have $R_n = 4^{(n-1)n/2} = 2^{(n-1)n}$. For each SNS matrix M, we can find 2^n distinct vectors for x. Therefore, the number of permutations $\tilde{d} = Ms \oplus x$ is $T_n = R_n 2^n = 2^{n^2}$. ■

4 Combined storage schemes by using SNS matrices

Consider an SIMD computer that consists of 2^n processing elements interconnected to 2^n memories through Ω_n. We assume that the data to be accessed is a one dimensional array denoted by $A = \{a(i) : 0 \leq i \leq 2^k - 1\}$ that is accessed by using a skew storage scheme defined by $d(i) = \Phi i$, where Φ is some boolean matrix, and $d(i)$ is the memory number where array element $a(i)$ is stored. Assume that the dimension of A is 2^k. The binary of i is $v_{k-1}i_{k-1}, \ldots, v_0 i_0$, where $v_{k-1}, \ldots, v_0$ are k canonical vectors of Z_2^k.

Assume Φ is to used for accessing power-of-2 strides, such as strides 1, 2, and 4 with $n = 3$. We denote these patterns by P_1, P_2, and P_3. We may choose matrix Φ as follows

$$\mathrm{d(i)} = \begin{pmatrix} d_2 \\ d_1 \\ d_0 \end{pmatrix} = \begin{array}{c} \begin{array}{ccccc} c_4 & c_3 & c_2 & c_1 & c_0 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \cdot \begin{pmatrix} i_4 \\ i_3 \\ i_2 \\ i_1 \\ i_0 \end{pmatrix}$$

where c_j denotes the jth column of Φ and i is restricted to its 5 least-significant bits. Each pattern can be translated within the array which leads to access different instances of that pattern. For each instance of P_1, the group of bits (i_2, i_1, i_0) takes all possible binary combinations and the remaining bits of i are constant. Each power-of-2 pattern can entirely be specified by some basis. Patterns P_1, P_2, and P_3 have bases $\tilde{P}_1 = \{v_2, v_1, v_0\}$, $\tilde{P}_2 = \{v_3, v_2, v_1\}$, and $\tilde{P}_3 = \{v_4, v_3, v_2\}$, respectively, where $\tilde{P}_i$ denotes the basis of the ith pattern.

Generally, the data dependences in the program dictate the data pattern that must must be accessed by each PE in order to honor the dependence. The array $a(.,.)$ shown in Figure 1 must be accessed by different data patterns depending on the dependence in the loop (on top of each array). Each of the cases (a) to (d) shows the partitioning of the array by a data pattern whose instances correspond to the shown frames. The component of the basis for each pattern are pointed by arrows in the address vector for each partitioning of the array. Therefore, an array must be accessed by a number of distinct data patterns which can be identified following compiler dependence analysis. Finding the access patterns for an array can be a compiler decision that can be taken once the data dependence analysis is complete.

The union of all pattern bases is $\{v_0, \ldots, v_4\}$. Generally, the storage matrix Φ should then be an $n \times p$ matrix, where p is the number of distinct canonical

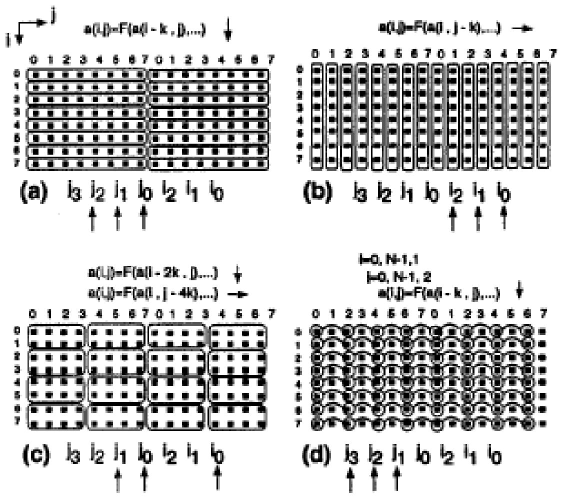
a(i,j)=F(a(i - k , j),...)
a(i,j)=F(a(i , j - k),...)
a(i,j)=F(a(i - 2k , j),...)
a(i,j)=F(a(i , j - 4k),...)
i=0, N-1,1
j=0, N-1, 2
a(i,j)=F(a(i - k , j),...)
(a)
(b)
(c)
(d)

Figure 1: Patterns dependences requiring: row access (a), column access (b), block access (c), and row-stride access (d)

vectors in the union of all pattern bases. When accessing P_1, function $d(i) = \Phi i$ can be decomposed into the following sum:

$$d(i) = \begin{array}{c} \begin{matrix} c_2 & c_1 & c_0 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \cdot \begin{pmatrix} i_2 \\ i_1 \\ i_0 \end{pmatrix} \oplus \begin{array}{c} \begin{matrix} c_4 & c_3 \end{matrix} \\ \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array} \cdot \begin{pmatrix} i_4 \\ i_3 \end{pmatrix} =$$

$$M_{P_1} \cdot \begin{pmatrix} i_2 \\ i_1 \\ i_0 \end{pmatrix} \oplus x \qquad (5)$$

where M_{P_1} is the left-hand matrix of the sum and the right-hand term is constant (x) when accessing P_1. M_{P_1} is said to be Φ *restricted to pattern* P_1. Equation 5 is a complement-permutation whenever M_{P_1} is SNS. The processors numbers $(s_0, s_1, \ldots, s_7)$ are identified with the values taken by (i_2, i_1, i_0), (i_3, i_2, i_1) and (i_4, i_3, i_2) during an access to some instance of P_1, P_2, and P_3, respectively. The permutation matrices associated with P_2 and P_3 are:

$$M_{P_2} = \begin{array}{c} \begin{matrix} c_3 & c_2 & c_1 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array} \quad M_{P_3} = \begin{array}{c} \begin{matrix} c_4 & c_3 & c_2 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

The product $M_{P_1} i$ also takes all possible binary values because M_{P_1} is NS which causes the elements of P_1 be distributed over the memories. Network alignment is guarantee for Ω_3 because M_{P_1}, M_{P_2}, and M_{P_3} are SNS for Ω_3. Thus the storage scheme defined by Φ is network contention-free as well as free of memory conflicts.

5 Storage schemes and multistage networks

The problem of finding a linear storage scheme which allows conflict-free and network-contention-free access for a given set of patterns is tractable for $p = 2$, but NP-complete for $p > 2$. The problem of finding an assignment of vectors in Z_2^n such that the matrices corresponding to all tuples are SNS is called *non-singular satisfiability* (NSS). It is proved that NSS is NP-complete for $n \geq 3$ [1].

We present an efficient algorithm for finding a linear storage scheme for a given pattern set. The idea is to construct the storage matrix Φ one row at a time, from top to bottom. We assume that, for each pattern, the matrix $\Phi_{P_i}[j]$ is SNS and attempt to construct the row $\Phi_{p-j,*}$ so that each $\Phi_{P_i}[j+1]$ is SNS. We use the idea from the proof of Theorem 5, in the construction of algorithm (H-n) which is as follows

1. Determine the upper two rows of the matrix.
2. Create each remaining row, working from top to bottom.
   For i in 2 to $n-1$ loop:
   (a) For each pattern P_j do:
      i. Obtain a matrix $\tilde{\Phi}_{P_j}$ by reducing the matrix Φ_{P_j} to the identity in its upper-left corner, using only row operations.
      ii. Use the ith column of this matrix to determine the equation associated with this pattern.
   (b) Solve the system of simultaneous equations. Assign entry $\Phi_{n-i-1,k}$ the value x_k.

The time complexity of this algorithm is $O(n^2)$. Since there are potentially several alternatives at Steps 1 and (b), one possibility is to use backtracking to exhaustively search for a solution.

6 Example

We illustrate the algorithm by an example with $p = 3$. Suppose we are given patterns P_1, P_2, P_3, and P_4 so that their bases are:

$$\begin{aligned} \tilde{P}_1 &= \{v_2, v_1, v_0\}, \quad \tilde{P}_2 = \{v_3, v_2, v_1\} \\ \tilde{P}_3 &= \{v_5, v_4, v_3\}, \quad \tilde{P}_4 = \{v_4, v_3, v_1\} \end{aligned}$$

We construct the upper two rows of Φ using algorithm H-2. For each Φ_{P_i}, we wish to ensure that the 2×2

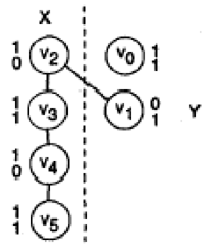

Figure 2: Finding the first two rows by coloring

sub-matrix in its upper-left corner is SNS. We can accomplish this by constructing a matrix for the reduced pattern set:

$$\begin{aligned}\tilde{P}_1' &= \{v_2, v_1\}, \quad \tilde{P}_2' = \{v_3, v_2\}\\ \tilde{P}_3' &= \{v_5, v_4\}, \quad \tilde{P}_4' = \{v_4, v_3\}\end{aligned}$$

Note that $\tilde{P}_i'$ is just $\tilde{P}_i$ with the lowest ordered vector removed. The vectors appearing first in some pattern basis $\tilde{P}_i'$ are $X = \{v_5, v_4, v_3, v_2\}$. The vectors appearing only second in some pattern are $Y = \{v_1\}$. Finally, v_0 does not appear first or second and thus can be assigned any value. The conflict graph, with one possible coloring, is shown in Figure 2.

The upper two rows of Φ have been determined. We let $x_5 \ldots x_0$ be the values in the lowest row:

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

We must now ensure that each 3×3 matrix that corresponds to some pattern is NS. So considering an instance of P_1, we must assign x_2, x_1, and x_0 in such a way that the matrix

$$\Phi_{P_1} = \begin{array}{c} \begin{array}{ccc} v_2 & v_1 & v_0 \end{array} \\ \begin{pmatrix} 1 & 0 & 1\\ 0 & 1 & 1\\ x_2 & x_1 & x_0 \end{pmatrix} \end{array}$$

is NS. The first step is to get the identity matrix in the upper right 2×2 sub-matrix by using only row operations. This requirement is satisfied for Φ_{P_1}. Using Theorem 5, we have $b_0 = x_1$, $b_1 = x_2$, $c = x_0$, $a_0 = 1$, and $a_1 = 1$.

Matrix Φ_{P_1} will be NS, and thus SNS, 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 Z_2, that is $b_0 \oplus b_1 \oplus c = 1$. Or, in terms of x's $x_2 \oplus x_1 \oplus x_0 = 1$.

Now consider Φ_{P_2}:

$$\Phi_{P_2} = \begin{array}{c} \begin{array}{ccc} v_3 & v_2 & v_1 \end{array} \\ \begin{pmatrix} 1 & 1 & 0\\ 1 & 0 & 1\\ x_3 & x_2 & x_1 \end{pmatrix} \end{array}$$

We need to get the identity matrix in the upper-left 2×2 sub-matrix, using only row operations. We achieve this by adding row 1 to row 2 and, next, we add row 2 to row 1. We denote this reduced matrix $\tilde{\Phi}_{P_2}$ according to the format of theorem 5:

$$\tilde{\Phi}_{P_2} = \begin{array}{c} \begin{array}{ccc} v_3 & v_2 & v_1 \end{array} \\ \begin{pmatrix} 1 & 0 & 1\\ 0 & 1 & 1\\ x_3 & x_2 & x_1 \end{pmatrix} \end{array}$$

For Φ_{P_2} to be NS, we must have $x_3 \oplus x_2 \oplus x_1 = 1$. The remaining conditions for Φ_{P_3} and Φ_{P_4} are $x_5 \oplus x_3 = 1$ and $x_4 \oplus x_3 \oplus x_1 = 1$, respectively.

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:

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

All these permutations can be achieved in an Ω_3 network without conflicts.

7 Application to FFT

In the following we present one implementation of a 16-point FFT over 16 memories/PEs. Data input $a(i)$, data output $b(i)$, and intermediate values of $b(i)$ are stored into memory m_j, where $0 \leq i, j \leq 15$. The complement-permutation for Ω_4 (Figure 3) is $P_k(i) = Ii + c_k$, where $c_1 = (1000)$, $c_2 = (0100)$, $c_3 = (0010)$, and $c_4 = (0001)$ which is is valid only for Ω_4.

A 16-input Baseline network is defined by $B_4 = E_1\sigma_4^{-1}E_2\sigma_3^{-1}E_3\sigma_2^{-1}E_4$, where E_i is the ith stage. This implies that the position of a message, issued at $s = s_3s_2s_1s_0$, is $pos_1 = s_3s_2s_1d_3$, $pos_2 = d_3s_3s_2d_2$, $pos_3 = d_3d_2s_3d_1$, and $pos_4 = d_3d_2d_1d_0$ at the output of the stages. Note that I is not SNS for B_4. Matrix $M = M_4$ is SNS for B_4 if and only if all square submatrices on the anti-diagonal are NS. To guarantee conflict-free access to the network and the memories we need M_i to be an SNS matrix for the target network. The square sub-matrices within the frames shown in Figure 4 are to be chosen as NS. Matrix $M = M_4$ is SNS for B_4 if and only if all the framed sub-matrices are NS.

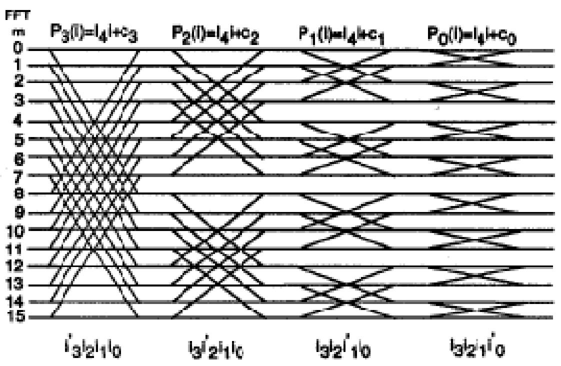

Figure 3: Permutations for FFT over Ω_4

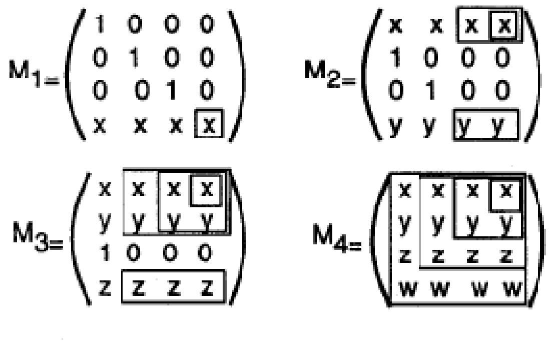

Figure 4: Non-singular matrices for B_4

We apply algorithm H-n to $pos_i = M_i s$ for finding the sub-matrices which should be SNS. Denote by v_k the complement vector for B_4 which corresponds the c_k in the Ω_4. To find the vectors v_k, for Ω_4 we have $P_k(i) = I_4 i + c_k$ which can be written as $P_k(i) = M^{-1}Mi + c_k = M^{-1}(Mi + Mc_k) = M^{-1}(Mi + v_k)$ which implies that $v_k = Mc_k$. For B_4, we have $P_k(i) = Mi+v_k = M(i+c_k)$ which means that either of $Mi+v_k$ or $M(i+c_k)$ can be used to access element $FFT(i + 2^k \ modulo\ N)$. One possible solution for achieving the four complement-permutations $p_k(i) = Mi \oplus c_k$, for $1 \le k \le 4$ and $0 \le i \le 15$, that is found by algorithm H-n is

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

and three complement vectors that are

$$c_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad c_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \quad c_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \quad c_4 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

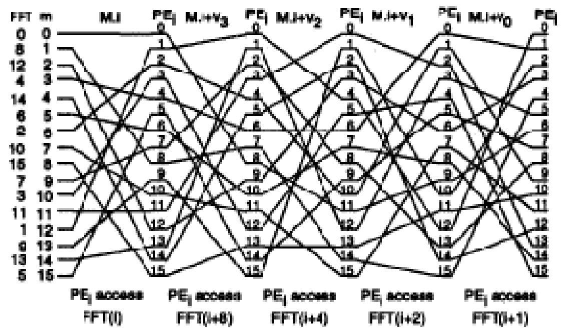

Figure 5: Permutations required for FFT over B_4

The permutations $p_k(i) = Mi \oplus c_k$ are shown in Figure 5, where the columns labeled (FFT) and (M) denote the data elements and memory-PE numbers, respectively. The ith data element is mapped to memory Mi. Each data element is mapped to a unique memory because M is NS. All the complement-permutations p_k are conflict-free for B_4 because M is also SNS.

This example shows how one can convert linear permutations for implementing FFT from one network to another. The benefit of our approach is to re-allocate the data so that the needed permutations become achievable for each multistage network.

8 General patterns

Testing algorithm H-n is carried on for arbitrary sets of data patterns. We iterated H-n until a network-contention-free XOR-scheme was found, or the limit of ten backtracking tries was exceeded. The studied range of memories ($N = 2^n$) is $8 \le N \le 256$ and the number of patterns p ranges from 3 to 16. One hundred cases were generated for each combination of these parameters.

The average number of clocks achieved by the XOR-scheme found by using H-n is displayed in Figure 6. Our scheme finds solutions requiring nearly one clock access for small numbers of patterns and moderate numbers of processors. The access time increases smoothly with increasing either the number of memories or the number of patterns.

For the same patterns, interleaving causes the average access time to be 6 to 26 fold that found by H-n when the number of memories ranges from 8 to 256. The static row-column-diagonals [3] scheme has an average access time that is 4 to 7 fold the that found by H-n.

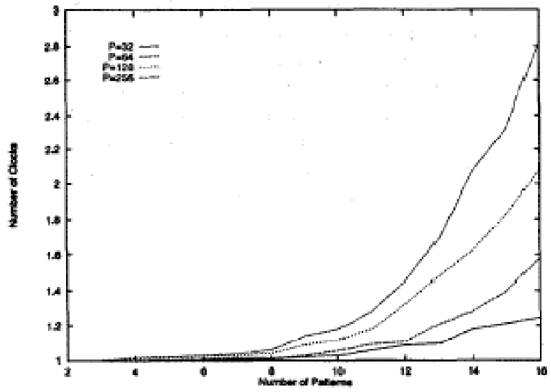

Figure 6: Average number of clocks for memory-network access

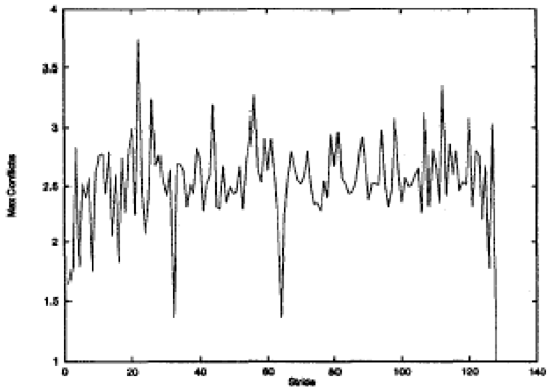

Figure 7: Sohi's maximum conflicts

9 Optimizing for stride access

In the following we examine finding storage schemes for accessing arbitrary strides in vector processors.

Algorithm H-n can find *optimum combined address transformations* for arbitrary groups of power-of-2 strides [1]. Conflict-free access to arbitrary strides is harder to achieve in the general case. However, the use of specific linear storage schemes that minimize the degree of sequentialization in accessing arbitrary stride provide useful throughput in vector computers.

Harper [4] and Sohi [7] showed that high memory throughput can be achieved when few buffers are used at the memory inputs and outputs. The buffers reduce the effects of transient degradation in pipelined memories. In this case, Sohi selected an XOR matrix that allows higher throughput than those obtained by using interleaving or row-rotation. Norton [6] proposed a specific scheme for the IBM-RP3 parallel memory.

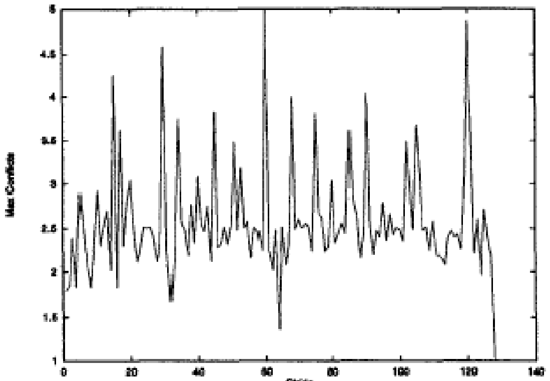

Figure 8: Norton's maximum conflicts

The linear storage schemes selected by Sohi and Norton for 8 memories are:

$$\mathrm{M_{sohi}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathrm{M_{norton}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Sohi's matrix has equal number of ones in each row and any group of 3 successive columns form a NS matrix. The latter condition allows conflict-free access to power-of-two strides but can also be used for arbitrary strides. Sohi's matrix is a particular case of complement-permutations that can be found by using Algorithm H-n. We present two schemes denoted by M_{gray} and M_{cond} which are:

$$\mathrm{M_{gray}} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\mathrm{M_{cond}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

These schemes can be easily generalized because their basis matrices (3×3) that appear in their right hand side are known as the *Gray* and *Conditional-Exchange* transformations. The remainder of these matrices can be constructed by row rotation of the basis matrix.

To compare these schemes we used the degree of conflict which is the maximum number of cycles required to access a given stride with random origin address. A stride access is defined by the sequence of addresses $a, a+s, a+2s, \ldots, a+(2^n-1)s$, where a is the origin, s is the stride, and 2^n is the number of memories.

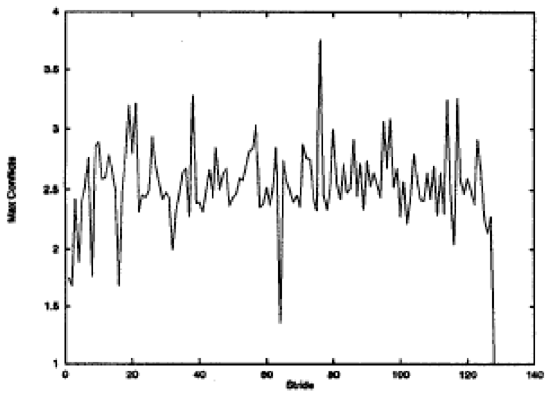

Figure 9: Gray maximum conflicts

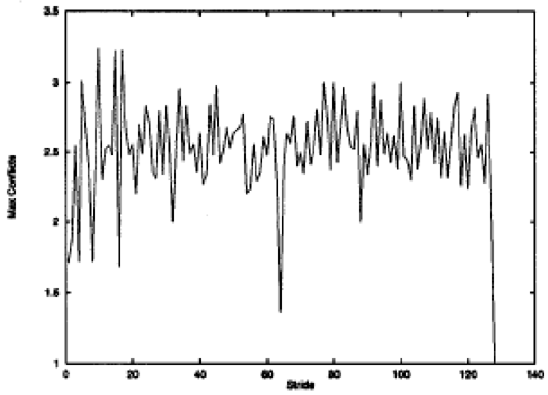

Figure 10: Conditional-exchange maximum conflicts

The origin was set randomly for each run and the results averaged over 100 runs. Figures 7, 8, 9, and 10 show the plots of the degree of conflict versus the stride ($1 \leq s \leq 128$) for M_{sohi}, M_{norton}, M_{gray}, and M_{cond}, respectively. All these schemes are fundamentally equivalent. Norton's scheme has many peaks which reach a degree of conflict of 5. Sohi's scheme achieves similar performance to that of the Gray scheme and has less fluctuation than the others. The Conditional-Exchange matrix achieves the lowest peak conflict.

10 Conclusion

Given an arbitrary set of power-of-2 data patterns, we have addressed the problem of finding compiler address transformations for storing arrays in parallel memories so that any instance of a pattern can be memory conflict-free and accessed without contention through an arbitrary multistage network. To automate the above process we proposed a compiler operator for synthesizing combined storages for arbitrary sets of power-of-2 data patterns so that memory and network conflicts are minimized. Algorithms such as FFT and sorting written for a given network can be converted with simple address emulation to other networks. Some of the results may also apply to secondary memory organization.

11 acknowledgment

Thanks to the Research Committee and the College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for partially supporting this research.

References

[1] M. Al-Mouhamed and L. Bic. Combining linear data patterns for accessing parallel memories through arbitrary multistage networks. *To appear in IEEE Trans. on Parallel and Distr. Sys.*, 1995.

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

[3] R. V. Boppana and C. S. Raghavendra. Efficient storage schemes for arbitrary size square matrices in parallel processors with shuffle-exchange networks. In *Proceedings of the International Conference on Parallel Processing*, pages 365–368, 1991.

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

[5] D. Lawrie and C.R. Vora. The prime memory system for array accesses. *IEEE Transactions on Computers*, C-31(12):435–442, May 1982.

[6] A. Norton and E. Melton. A class of boolean linear transformations for conflict-free power-of-two stride access. In *Proceedings of the International Conference on Parallel Processing*, pages 247–254, 1987.

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