A Compiler Transformation to Improve Memory Access Time in SIMD Systems

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: The serialization of memory accesses and network conflicts are two major limiting factors in lock-step parallel memories. We derive conditions for accessing parallel memories which is free of both network and memory conflicts. This applies to accessing arbitrary sets of linear data patterns. We also combine different access patterns (NP-complete) into one single compiler address transformation. The synthesized storage scheme applies to arbitrary linear patterns, arbitrary multistage networks, and arbitrary number of power-of-2 memories. We propose a new heuristic for synthesizing combined XOR-matrices. Performance of optimized storage schemes is presented for sorting and for combining arbitrary sets of power-of-2 patterns.

A Compiler Transformation to Improve Memory Access Time in SIMD Systems

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

The serialization of memory accesses and network conflicts are two major limiting factors in lock-step parallel memories. We derive conditions for accessing parallel memories which is free of both network and memory conflicts. This applies to accessing arbitrary sets of linear data patterns. We also combine different access patterns (NP-complete) into one single compiler address transformation. The synthesized storage scheme applies to arbitrary linear patterns, arbitrary multistage networks, and arbitrary number of power-of-2 memories. We propose a new heuristic for synthesizing combined XOR-matrices. Performance of optimized storage schemes is presented for sorting and for combining arbitrary sets of power-of-2 patterns.

1 Introduction

Non-uniform access to parallel memories and network contention are responsible for significant performance degradation, especially in SIMD systems. The use of a prime number of memories [5] significantly outperforms interleaving but requires expensive address translation. 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.

Based on *skew storage*, *XOR-schemes* were proposed [6, 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 for the Omega network have been studied by using non-singular 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.

Section 2 presents linear permutations and the networks. Section 3 presents properties of storage schemes. Sections 4 and 5 present our method for combining data patterns into one single compiler transformation. In Sections 6 and 7 we present applications to sorting and arbirary sets of data patterns. Section 8 concludes this work.

2 Linear permutations and networks

In a multistage network, routing a source $s = s_{n-1} \ldots s_0$ to destination $d = d_{n-1} \ldots d_0$ 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 $pos_0(s,d) = s_{n-1} \ldots s_0$. we can easily find the position of the message at the output of the ith stage:

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

We can similarly find the position of the message for other networks. A network input is denoted by $s = (s_{n-1}, \ldots, s_0) \in S$ and output is denoted by $d = (d_{n-1}, \ldots, d_0) \in S$. This applies to n-stage networks. A linear permutation is a function $M : S \rightarrow S$ for which each source $s \in S$ maps into destination $d = Ms = (d_{n-1}, \ldots, d_0)$ 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 $\tilde{M}[i]s$:

$$\mathrm{pos_i(s)} = (s_{n-i-1} \ldots s_0 d_{n-1} \ldots d_{n-i}) = \tilde{M}[i](s_{n-1} \ldots s_{n-i} s_{n-i-1} \ldots s_0) \quad (1)$$

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

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

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 $M[i]$:

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

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:

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

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 non-singular (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 $1 \leq i \leq n$.

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 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 $M[i+1]$ depends 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.

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

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

$$\mathrm{M}_{\mathrm{P1}} \cdot \begin{pmatrix} i_2 \\ i_1 \\ i_0 \end{pmatrix} \oplus \mathrm{x} \qquad (5)$$

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 access to instances of P_1, P_2, and P_3, respectively. The permutation matrices associated with P_2 and P_3 are:

$$\mathrm{M}_{\mathrm{P}_2} = \begin{array}{c} \begin{array}{ccc} c_3 & c_2 & c_1 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \end{array} \quad \mathrm{M}_{\mathrm{P}_3} = \begin{array}{c} \begin{array}{ccc} c_4 & c_3 & c_2 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

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. 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. Algorithm (H-n) 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. Let the basis of P_j be $v_{\ell_{n-1}} \cdots v_{\ell_0}$, and $y_k = (\tilde{\Phi}_{P_j})_{n-k-1,\ell_i}$. Then the equation is:

$$x_{\ell_i} \oplus \Sigma_{k=0}^{i-1} x_{\ell_k} y_k = 1 \qquad (6)$$

   (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 Sorting

Bitonic sorting can be implemented on the basis of the four group of access patterns shown in Figure 1 for a 16-point array and 8 memory-PE. For each group, every processor is to read a pair of items from memory, sort them, and store the sorted items back into memory. Each pair of items (0-1) involved in one group are connected by an edge in Figure 1. The problem

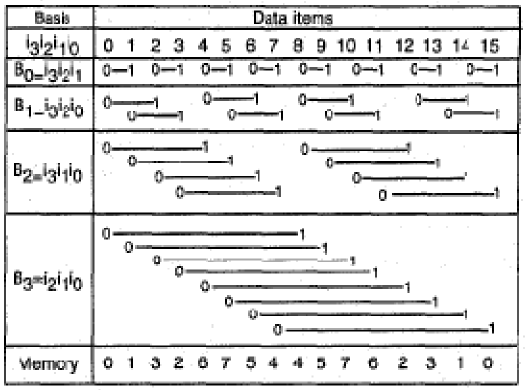

Figure 1: Sorting 16-items over 8 memories

is to find the mapping of data elements to memories and the implied access patterns.

Two accesses are needed for each group: 1) the left-hand items which are all accessed in step 0, and 2) the right-hand items which are all accessed in step 1. The bases of the patterns are $B_0 = i_3 i_2 i_1$, $B_1 = i_3 i_2 i_0$, $B_2 = i_3 i_1 i_0$, and $B_3 = i_3 i_2 i_0$ and correspond to access patterns (P_0, P_1, P_2, P_4) shown in Figure 1. For an Ω_3 network, the leading NS matrices are located at the upper left edge of the storage matrix. Therefore, algorithm H-2 assigns $\binom{11}{01}$ as the upper left corner of the storage matrix. Algorithm H-n completes the previous sub-matrix and finds a complete solution so that each of the restricted sub-matrices (Φ_{P_i}) is SNS:

$$\Phi = \left(\begin{array}{cccc} i_3 & i_2 & i_1 & i_0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

and the restricted sub-matrices to each pattern are:

$$\Phi_{P_0} = \left(\begin{array}{ccc} i_3 & i_2 & i_1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array}\right) \quad \Phi_{P_1} = \left(\begin{array}{ccc} i_3 & i_2 & i_0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right)$$

$$\Phi_{P_2} = \left(\begin{array}{ccc} i_3 & i_1 & i_0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right) \quad \Phi_{P_3} = \left(\begin{array}{ccc} i_2 & i_1 & i_0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right)$$

Each data element $a(i)$, for $0 \leq i \leq 15$ is stored into memory Φi at offset $off = i_3$:

Items/Patterns	Each P_i fetches
items 0 of P_0	$a(k_2 k_1 k_0 0)$
items 1 of P_0	$a(k_2 k_1 k_0 1)$
items 0 of P_1	$a(k_2 k_1 0 k_0)$
items 1 of P_1	$a(k_2 k_1 1 k_0)$
items 0 of P_2	$a(k_2 0 k_1 k_0)$
items 1 of P_2	$a(k_2 1 k_1 k_0)$
items 0 of P_3	$a(0 k_2 k_1 k_0)$
items 1 of P_3	$a(1 k_2 k_1 k_0)$

Mem	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
$a(i)$	0	1	3	2	7	6	4	5
	15	14	12	13	8	9	11	10
P_0	0	1	1	0	1	0	0	1
	1	0	0	1	0	1	1	0
P_1	0	0	1	1	1	1	0	0
	1	1	0	0	0	0	1	1
P_2	0	0	0	0	1	1	1	1
	1	1	1	1	0	0	0	0
P_3	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1

Table 1: Storage scheme and access patterns

Table 1 shows the storage of the data elements by Φ and the first (marked by 0) and second (marked by 1) access to each pattern P_j. The elements of each pattern P_j map to distinct memories because ϕ_{P_j} is NS. The elements of each P_j can be aligned to the PEs without conflict because each Φ_{P_j} is SNS for Ω_3.

7 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 $8 \leq N \leq 256$ 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 2. 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. This approach largely outperfoms interleaving (6 to 26 fold) and the static row-column-diagonals [3] scheme (4 to 7 fold).

8 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 (Sorting

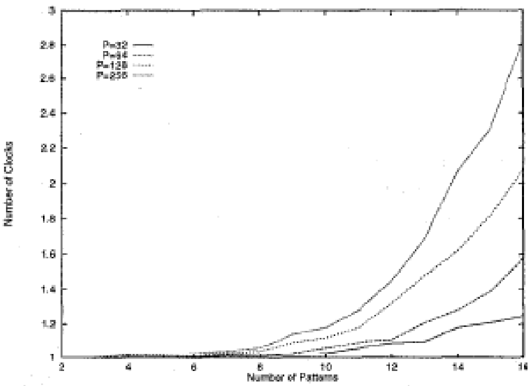

Figure 2: Average number of clocks for memory-network access

and FFT) written for a given network can be converted with little address emulation (hardware or software) to other networks. Some of the results may also apply to secondary memory organization.

References

[1] M. Al-Mouhamed and L. Bic. Combining linear data patterns for accessing parallel memories through arbitrary multistage networks. *Under review with 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] G. S. Sohi. High-bandwidth interleaved memories for vector processors–A simulation study. *IEEE Transactions on Computers*, 42(1):34–44, Jan 1993.