A Heuristic Storage for Minimizing Access Time of Arbitrary Data Patterns

M. A. Al-Mouhamed and S. S. Seiden

Department of Computer Engineering
King Fahd University of Petroleum and Minerals (KFUPM)
31261 Dhahran, Saudi Arabia

Department of Information and Computer Science
University of California Irvine,
California 92717, USA

Abstract—The serialization of memory accesses is a major limiting factor in high performance SIMD computers. The data patterns or templates that are accessed by a program can be perceived by the compiler, and, therefore, the design of dynamic storage schemes that minimize conflicts may dramatically improve performance. The problem of finding storage schemes that minimize the access time of arbitrary sets of power-of-two data patterns is proved to be NP-complete. We propose linear address transformations that can be dynamically applied by each processing element for mapping array references onto memories. An efficient approach for combining the constraints of different access patterns into one single linear address transformation is presented. We prove that finding the transformation that minimizes the access time is reducible to N-coloring, where N is the number of parallel memories. Using coloring heuristics, storage schemes are investigated with respect to minimizing the implementation cost (perfect storage) and overall access conflicts (semi-perfect storage). Results show that the perfect-storage may deviate on the average by 20% from the optimum access time in the case of 10 arbitrary data patterns and 16 memories. However, semi-perfect schemes lead to dramatic reduction of the degree of conflict compared to perfect-schemes. The proposed heuristic storage largely outperforms interleaving and row-column-diagonals storages. The method can be implemented as compiler procedure for synthesizing storage schemes that promote parallel access to arbitrary sets of data patterns.
A Heuristic Storage for Minimizing Access Time of Arbitrary Data Patterns

Mayez A. Al-Mouhamed and Steven S. Seiden

Abstract—The serialization of memory accesses is a major limiting factor in high performance SIMD computers. The data patterns or templates that are accessed by a program can be perceived by the compiler, and, therefore, the design of dynamic storage schemes that minimize conflicts may dramatically improve performance.

The problem of finding storage schemes that minimize the access time of arbitrary sets of power-of-two data patterns is proved to be NP-complete. We propose linear address transformations that can be dynamically applied by each processing element for mapping array references onto memories. An efficient approach for combining the constraints of different access patterns into one single linear address transformation is presented. We prove that finding the transformation that minimizes the access time is reducible to N-coloring, where N is the number of parallel memories. Using coloring heuristics, storage schemes are investigated with respect to minimizing the implementation cost (perfect storage) and overall access conflicts (semiperfect storage).

Results show that the perfect-storage may deviate on the average by 50% from the optimum access time in the case of 10 arbitrary data patterns and 16 memories. However, semiperfect schemes lead to dramatic reduction in the degree of conflict compared to perfect-schemes. The proposed heuristic storage largely outperforms interleaving and row-column-diagonals storages. The method can be implemented as compiler procedure for synthesizing storage schemes that promote parallel access to arbitrary sets of data patterns.

Index Terms—Boolean matrices, heuristics, memory organization, NP-complete, parallel memories, performance evaluation, storage schemes.

1 INTRODUCTION

The serialization of memory accesses is a major limiting factor to bandwidth balancing between processors and parallel memories. Memory interleaving maps consecutive addresses into different physical memories so that simultaneous accesses can be performed in one memory cycle. Conflict-free access is possible only when the stride associated with successive references is relatively prime to the number of memories. The use of prime number of memories [1] greatly reduce potential conflicts but at the cost of expensive address calculation.

Budnik and Kuck [2] proposed the rotate-rotation scheme that allows conflict-free access to arbitrary row, and column of arrays. To avoid run-time overhead, Sohi [3] proposed bit-wise Boolean address transformations for vector processors in order to determine the memory number where a given array element should be stored. The scheme can be efficiently used for power-of-two 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.

The image by the storage scheme of all the elements of a given pattern should map into different memories. Therefore, the columns or the rows of the needed transformation matrix should be linearly independent [4]. Norton and Melton [5] synthesized a transformation matrix that allows conflict-free access to a number of power-of-two strides. As the interconnection network should provide data alignment between processors and memories, other constraints [6] can then be used for finding the storage matrix. Boppana and Raghavendra [7] proposed a bit-wise linear transformation matrix (nonsingular) for accessing to the row, column, main-diagonal, and square blocks.

We are concerned with dynamically reconfigurable storage schemes for SIMD models that minimize the overall access time of an arbitrary set of weighted data patterns. The problem is to find how arrays can be stored into parallel memories in order to enforce the elements of a given data pattern be uniformly distributed over the memories. Given a program that requires access to a set of data patterns, our objective is to find a cost-effective storage scheme that minimizes overall memory access time.

This paper is organized as follows. Section 2 presents the background. The template bases are defined in Sections 3. Sections 4 and 5 present the perfect storage that allows combining data patterns and minimizing the implementation cost. Section 6 presents the semiperfect storage that allows minimizing overall access conflicts. Section 7 presents our method for synthesizing storage schemes. The evaluation of this work and comparison to other proposals are presented in Sections 8 and 9, respectively. The conclusions to this work are presented in Section 10.
2 BACKGROUND

Any processor in an SIMD system can access any memory through an interconnection network. Assume there are an equal number \( P = 2^d \) of processors and memories. Conflicts occur when \( i \) processors \( (i > 1) \) try to access the same memory during a given cycle which requires \( i \) cycles for the memory to serve them. Since all processors run in lock-step, the entire computation is dramatically slowed. It would be desirable to store the data that should be simultaneously accessed into different memories so that parallel access can be achieved.

Suppose that we know a priori the memory access patterns of a given program. We assume that the data to be accessed is a two dimensional array. For example, the program shown in Fig. 1a that uses array \( c \) with size \( 2^d \times 2^m \), where \( d_1 \) and \( d_2 \) are two positive integers. Fig. 1b shows the vectorized program that can run on an SIMD system for which the number of PEs as well as the number of parallel memories is \( 2^d \).

for \( i = 0 \) to \( 2^d - 1 \)
for \( j = 0 \) to \( 2^d - 1 \)
\( s_i; \) all PEs \( r_j \leftarrow c(i \cdot j + 0) \)
end
end

Fig. 1. Example of a loop that requires accessing template \( T_i \).

Due to data dependency between \( c(i, j) \) and \( c(i, j + 1) \) across the iterations, one way to vectorize the program is that all the eight PEs should simultaneously operate on one subcolumn of eight elements of array \( c \) at each iteration. Statements \( s_0 \), \( s_2 \), and \( s_4 \) require parallel access to the data template \( T_i \) shown in Fig. 2.

A template is defined as a pattern of array elements whose addresses are related by some relationship such as those shown in Fig. 2. A template is a set of data elements that should be accessed in parallel, by all the PEs, during the running of the program. We are interested in templates having power-of-two elements. The origin of a template is the coordinate, within the array, of its upper leftmost element. For example the set of all rows (columns) is a template, and each row (column) is a template instance [4], [5].

The memory is assumed to be a single two dimensional array of size \( 2^d \times 2^m \) such that the element in the \( i \)th row and the \( j \)th column is denoted by \( (a, b) \). The upper leftmost element is \((0,0)\). The sizes of the horizontal and vertical dimensions are \( 2^d \). An integer \( t \) is represented by \( x_t 1 \ldots x_{t-1} \).

A row position \( a \) can also be thought of as a vector, over the finite field \( Z_d \), the integers modulo \( 2 \). In \( Z_d \) addition corresponds to logical exclusive or, and multiplication corresponds to logical and. We define a vector space \( \mathcal{F} = Z_2^n \) for horizontal position. Let \( F = \{f_0, f_1, \ldots, f_{2^n-1}\} \) be the canonical basis of \( \mathcal{F} \), i.e., \( f_0 = (0, 0, \ldots, 0) \), \( f_1 = (0, 1, 0, \ldots, 0) \), \( \ldots \), \( f_{2^n-1} = (0, 0, \ldots, 1) \). Each row has a unique representation as a vector in \( \mathcal{F} \). A row \( a \) is expressed as \( a_0 f_0 \oplus a_1 f_1 \oplus \cdots \oplus a_{2^n-1} f_{2^n-1} \) in terms of \( F \). We similarly define vector spaces \( G \) for column positions and \( H \) for memory unit numbers, with canonical bases \( G = \{g_0, g_1, \ldots, g_{2^n-1}\} \) and \( H = \{h_0, h_1, \ldots, h_{2^n-1}\} \), respectively.

The Cartesian product of the vector spaces \( \mathcal{F} \) and \( G \) is a new vector space \( \mathcal{V} = \mathcal{F} \times G \) with basis \( \{f_0, f_1, \ldots, f_{2^n-1}; g_0, g_1, \ldots, g_{2^n-1}\} \). Let \( n = 2d \). We denote this combined basis as \( V = \{v_0, v_1, \ldots, v_{2^n-1}\} \), where \( v_0 = f_0, v_1 = f_1, v_2 = g_0, \ldots \). This vector space is isomorphic to \( Z_2^n \). Any location \((a, b)\) in memory is uniquely associated with a linear combination of the basis elements \( a_0 f_0 \oplus a_1 f_1 \oplus \cdots \oplus a_{2^n-1} f_{2^n-1} \oplus b_0 g_0 \oplus \cdots \oplus b_{2^n-1} g_{2^n-1} \). Adding two vectors in \( Z_2^n \) corresponds to bitwise exclusive or. We refer to the elements of the basis \( V \) using either \( f \)s and \( g \)s, or \( v \)s, depending on which is notionally convenient.

3 TEMPLATES

We define a template \( T_i \) by a basis \( T_i \), which is a nonempty subset of \( V \). Notice that there is a definite distinction between the definition of a template \( T_i \), which is a set of subarrays, and its basis \( T_i \), which is a set of vectors. We assume all templates bases are of size \( p \), i.e., there are \( 2^p \) elements in any given template instance. This is best explained by looking at some examples. Let \( p \) and \( d \) both be 3. Our basis is \( V = \{v_0, v_1, v_2, v_3, v_4, v_5\} \), or, alternatively, \( V = \{v_0, v_1, v_2\} \). Consider the template \( T_i \) defined by \( T_i = \{f_j, f_k, f_3\} \). The set of templates instances described are all nonoverlapping columns of eight elements, the upper left-most template instance having origin \((0, 0)\). 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 \((h_0 g_0 \oplus h_1 g_1 \oplus h_2 g_2) \) is the template instance's origin. Intuitively, we are letting the three least significant bits of \( a \) vary, while the other bits of \( a \) and all bits of \( b \) remain fixed. By allowing different bits to vary, we
generate templates of different shapes. Let \( T_2 \) have basis \( T_2 = \{ f_{0}, f_{1}, g \} \). Then, in \( T_2 \) all template instances are four elements tall. Since \( g \) is omitted, this template skips a column. Thus, we have two \( 4 \times 1 \) subarrays, spaced two columns apart. We define \( T_3 \) by \( T_3 = \{ f_{0}, f_{1}, g, f_{3} \} \). This template is four \( 1 \times 2 \) subarrays spaced two rows apart. We let \( T_3 \) have basis \( T_3 = \{ f_{0}, f_{1}, g, f_{2} \} \). It is a \( 4 \times 2 \) subarray. All of these templates are illustrated in Fig. 2.

An XOR-scheme is a linear function \( \Phi : \mathbb{F} \times \mathbb{G} \rightarrow \mathcal{H} \). The function \( \Phi \) is represented by a \( p \times n \) matrix, which we denote \( \Phi \). We apply \( \Phi \) to a vector \( \mathbf{X} \) by matrix multiplication:

\[
\Phi(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) = \Phi \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}.
\]

The \( i \)th entry of the \( j \)th column of \( \Phi \) is \( \Phi_{ij} \). (The upper leftmost entry is \( \Phi_{00} \).) We denote the \( i \)th column of \( \Phi \) by \( \Phi_{[i]} \) and the \( i \)th row of \( \Phi \) by \( \Phi_{[i]} \). The columns of this matrix represent the values, in terms of the basis \( \mathcal{H} \), of \( \Phi \) on the members of the basis \( \mathcal{V} \), i.e., \( \Phi_{ij} \) is the value of \( \phi(v_j) \).

We can also consider \( \Phi \) as an ordered set of \( p \) functions, \( \{\phi_0, \phi_1, \ldots, \phi_{p-1}\} \), mapping from \( \mathbb{F} \times \mathbb{G} \) to \( \mathbb{Z} \), where \( \phi(X) = \phi_0(X) h_0 \oplus \phi_1(X) h_1 \oplus \cdots \phi_{p-1}(X) h_{p-1} \). The matrix of \( \Phi \) is \( \Phi_{[i]} \).

Then \( \Phi \) allows conflict-free access to \( T_n \), if and only if \( \Phi \) maps each linear combination of \( T_n \) to a unique element of \( \mathcal{H} \). Since \( \Phi \) is linear, all translations of these linear combinations are also conflict-free. In other words, if for one template instance \( X \) in \( T_n \), \( \Phi \) restricted to \( X \) is one to one, then for all template instances \( X \) in \( T_n \), \( \Phi \) restricted to \( X \) is one to one.

4 Perfect XOR-Schemes

We say that an XOR-scheme is perfect if and only if all columns \( \Phi_{[i]} \) contain at most one nonzero entry. In other words, in the expression \( \phi_0 = \phi_0(X) h_0 \oplus \phi_1(X) h_1 \oplus \cdots \phi_{p-1}(X) h_{p-1} \), any particular \( x_i \) is used at most once, where \( h_i \) is the \( i \)th canonical vector of the basis of the memory numbers. A perfect XOR-scheme only requires \( n \) XOR gates to be implemented.

We give an example using the templates of Section 3 for which \( n = 6 \) and \( p = 3 \). We would like to find a perfect XOR-scheme for this set of templates. But first, let us consider the subset of templates \( \{ T_1, T_2, T_3 \} \). Notice that \( f_1 \) appears in all templates bases of this subset. Therefore, let \( \phi_0(a_0, a_2, b_0, b_1, a_3) = a_2 \). Further, notice that \( g_0 \) appears only in \( T_2 \) and \( f_0 \) appears only in \( T_1 \) and \( T_3 \). Therefore, let \( \phi(a_0, a_1, b_0, b_1, b_2) = b_0 \oplus a_0 \). Finally, notice that \( g_1 \) appears only in \( T_2 \) and \( f_2 \) appears only in \( T_1 \) and \( T_3 \). We let \( \phi(a_0, a_2, b_0, b_1, b_2) = b_1 \oplus a_2 \). Let us generalize this procedure.

Theorem 4.1. Let \( \Phi \) be the matrix of a perfect XOR-scheme. If \( \Phi_{ij} = 1 \) and \( \Phi_{ik} = 1 \) where \( i \neq j \), and \( \psi_i \) and \( \psi_j \) are both in \( T_n \), then \( T_n \) cannot be accessed conflict-free.

Proof. This is easily seen by a counting argument. \( \Phi_{ij} \) can only take on two values \( x_i \) and \( x_j \), each can take on two values, for a total of four. Further, because each column of \( \Phi \) contains at most one nonzero, no other \( \Phi_{ij} \) varies with \( x_i \) or \( x_j \). Therefore, given a template instance, we are mapping two elements of it to each memory unit in its image.
5 Nonperfect XOR-Schemes

Suppose we do not restrict ourselves to perfect XOR-schemes. Does the set of templates in Fig. 2 have an XOR-scheme? By inspection, we find the matrix of one possible XOR-scheme is:

\[
\Phi = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]

Why does this function allow conflict-free access? Consider \( \Phi \) restricted to the template \( T_i \). The matrix of this restricted function is:

\[
\Phi(T_i) = \begin{pmatrix}
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 1
\end{pmatrix}
\]

Notice that this matrix has rank 3; its columns are linearly independent. The dimension of the image of \( \Phi \) restricted to \( T_i \) is three, and, therefore, conflict-free access is assured. We denote \( \Phi \) restricted to \( T_i \) by \( \Phi(T_i) \). The matrix of \( \Phi(T_i) \) is \( \Phi(T_i) = (\Phi_{ij})_{i,j \in T_i} \). The matrices of \( \Phi \) restricted to \( T_2 \ldots T_4 \) have all rank 3. In general, we need to find a function \( \Phi \) with matrix \( \Phi \), such that \( \Phi(T_i) \) has rank \( |T_i| \) for all templates \( T_i \). How can such XOR-schemes be found? First consider a more general problem.

Suppose we are given:

1) a vector space \( Z = Z_2^4 \),
2) a set of variables \( V = \{v_1, v_2, \ldots, v_4\} \), and
3) a set \( T = \{T_1, T_2, \ldots, T_4\} \), where each \( T_i \) is some member of \( Z^V \) and each variable must appear in some \( T_i \).

The problem is to assign each variable a value in \( Z \), such that for all \( i \), the vectors assigned to the variables in \( T_i \) are linearly independent. We call this problem linear independence satisfiability (LIS). We can prove [8] that LIS is NP-complete. Our proof that LIS is NP-complete does not depend on the fact that \( Z \) is over \( Z_2 \); LIS is NP-complete over any finite field.

It is easily seen that LIS is equivalent to finding a nonperfect XOR-scheme. Note that finding a general XOR-scheme for four memory units is NP-complete, where finding a perfect XOR-scheme for four memory units is polynomial. Also note that we can build independence graphs only for \( p = 2 \). We need to find a model of XOR-schemes for \( p > 2 \), from which good heuristics can be derived.

6 Semiperfect XOR-Schemes

We investigate a class of XOR-schemes which are somewhere between perfect and general XOR-schemes. We call this class of XOR-schemes semiperfect. We say that an XOR-scheme \( \Phi \) represented by a matrix \( \Phi \), is semiperfect if and only if for all templates \( T_i \) in \( T \), the matrix of \( \Phi \) restricted to \( T_i \) contains at most one column with two nonzero entries, and the rest of the columns have one or zero nonzero entries.

Theorem 6.1. Let \( \Phi \) be a matrix over \( Z_2 \) with at most one nonzero entry in each column. Let \( \Phi' \) be defined by:

\[
\Phi'_{ij} = \begin{cases} 
1 & i = c \text{ and } j = k \\
\Phi_{ij} & \text{otherwise}
\end{cases}
\]

for some fixed \( c \) and \( k \). Then \( \text{rank}(\Phi') \geq \text{rank}(\Phi) \).

Proof. If \( \Phi'_{i,k} = 1 \), then \( \Phi' = \Phi \), and, therefore, \( \text{rank}(\Phi') = \text{rank}(\Phi) \). Otherwise, if some other column \( \Phi'_{i,j} \) has a nonzero in row \( c \), then add \( \Phi'_{i,c} \) to \( \Phi'_{i,k} \), giving a new matrix \( \Phi'' \), which has the same rank as \( \Phi' \). Since this other column must have at least one nonzero entry, the only change to \( \Phi'' \) is that \( \Phi''_{i,k} = 0 \). Now \( \Phi'' = \Phi \), and, therefore, \( \text{rank}(\Phi'') = \text{rank}(\Phi) \). If no other column \( \Phi'_{i,j} \) has a nonzero in row \( c \), then the rank of \( \Phi' \) is one greater than that of \( \Phi \). \( \Box \)

Given a perfect XOR-scheme for a set of templates (in which some templates are not accessed conflict free), we can use the preceding theorem to create a semiperfect XOR-scheme with a decreased number of conflicts, by selectively adding ones to its matrix. In fact, the example given in Section 5 is a semiperfect XOR-scheme. We call this process of selectively adding ones augmenting.

Given a perfect XOR-scheme which is not conflict free, we want to know if it can be augmented to a conflict free semiperfect scheme. Unfortunately, answering this is NP-complete. Each column \( \Phi_{i,j} \) will either be augmented or not. For each template \( T_i \), at most one column of \( \Phi(T_i) \) may be augmented. Given these restrictions, we wish to find a subset of columns to augment such that all \( \Phi(T_i) \) are augmented. For \( p = 3 \), this problem is exactly ONE-IN-THREE SAT, which is NP-complete [9].

7 Heuristic Approaches

To \( p \)-color conflict graphs, we extend the weight function to the edges and vertices of the graph. The weight of an edge is proportional to the number of extra CPU cycles that will be spent if the vertices of that edge are identical colored (assuming that all other edge constraints are met). Thus, the weight of an edge is \( \omega(v_i, v_j) = \sum_{v_k \in T_i} \alpha(T_k) \), where \( \alpha(T_k) \) is the access frequency of \( T_k \). The weight of a vertex is defined by \( \omega(v_i) = \max_{v_j} \{\omega(v_i, v_j)\} \). The weighted graph for \( T \) is shown in Fig. 5.

Fig. 5. Weighted conflict graph.
Graph coloring heuristics are used for solving problems such as scheduling [10] and register allocation [11]. One of the simplest heuristics is the greedy coloring algorithm [12]. We present two heuristics called Highest-Weighted-Conflict-First (HWCF) and Most-Immediate-Conflict-First (MISF) for generating perfect XOR-schemes. These are modifications of the basic greedy method for weighted graphs.

A color number is an integer in the range \([0 \ldots p - 1]\). A vertex number is an integer in the range \([0 \ldots n - 1]\). A template number is an integer in the range \([0 \ldots t - 1]\). An array \(cost(v, c)\) indexed by vertex numbers and colors is used. Initially, \(cost(v, c) = 0\) for all \(v\) and all \(c\).

HWCF works as follows. Put all vertices in a priority queue \(H\) sorted in the decreasing order of the weight. Pick the maximally weighted vertex from \(H\), say \(v\), and assign it color \(c\) so that \(cost(v, c)\) is the least among all colors (least conflict). The next step is to update the cost values of the vertex's neighbors. For this \(cost(w, c)\) is augmented by \(\alpha(w, v)\) for each vertex \(w\) that is adjacent to \(v\) and edge \((w, v)\) is removed from the adjacency list as it will need no further consideration. Repeat until all vertices are expedited.

We are best equipped to decide the color of a vertex when some of its neighbors have already been colored. The uncolored neighbors of vertices in the colored set are coloring candidates. Micf works as follows. Pick the maximally weighted vertex from \(H\), say \(v\), and color it as in HWCF. Next, add the vertices adjacent to \(v\) to a priority queue \(H_{adj}\) that is empty before any vertices are colored. We repeatedly remove the maximal vertex from \(H_{adj}\) and color it, until the current component is completely colored (the graph may not be connected). We have completely colored the current component. We proceed to the next component by deleting the colored vertices from \(H\) and setting \(H_{adj} = \emptyset\). This proceeds until \(H\) becomes empty.

Analysis of the time complexity [8] shows that HWCF and MISF run in \(O(p(t + n) + n^3)\) time, where \(t\), \(n\), and \(p\), are the number of templates, the number of processors, and the number of distinct vectors of the template bases, respectively.

We now introduce a heuristic SP for augmenting a perfect XOR-scheme, which has conflicts, to a semiperfect scheme, with fewer conflicts. We assume that a perfect XOR-scheme is given. SP works as follows. If a one is added to a column \(\Phi_{x, y}\), then a one cannot be added to any column \(\Phi_{x, y'}\), where \(x\) and \(y\) are both in some \(\Phi_{y} (T_x)\). Or, alternatively, \(y\) and \(y'\) are both in \(T_x\). Whenever a one is added to a column \(\Phi_{x, y}\) mark all \(T_x\) such that \(y\) is in \(T_x\) as blocked.

Initially, no columns are blocked. We consider templates in order of their weight (maximal first). \(T_x\) is conflict free, go to the next template. Otherwise, two columns of \(\Phi_{y} (T_x)\) are necessarily identical. Adding a nonzero to one of the columns will increase the rank of \(\Phi_{y} (T_x)\), provided that the row to which the nonzero is added contains only zeros, and that the column is not blocked. In the case that one of the columns is blocked, we augment the other one. If neither is blocked, we may choose to augment either. We choose the one which is contained in the least number of templates. Let the column so chosen be \(\Phi_{x, y}\). We examine the rows of \(\Phi_{y} (T_x)\). If some row \(k\) contains only zeros, we set \(\Phi_{x, y} = 1\), and add all the vectors (columns) which appear in some template with \(y\) to the blocked set. We then proceed to the next template. The time complexity of SP is \(O(p(t + n) + n^3)\), where \(\alpha\) is the number of calls to SP.

8 Performance Evaluation

We experimentally evaluate the performance of the proposed schemes and compare them to other known schemes.

Let \(a(T)\) be the number of accesses to \(T\). A lower bound \(a_{\text{min}}\) on the number of accesses can be defined as \(a_{\text{min}} = \sum a(T)\). Unfortunately, there is no guarantee that \(a_{\text{min}}\) is achievable. Therefore, the optimum access time \(A_{\text{opt}}\) of a perfect scheme, for a given set of templates pairs each with a given frequency, is found by using a branch-and-bound algorithm.

We evaluate the performance of a perfect XOR-scheme by comparing the number of accesses required with the optimal perfect XOR-scheme. Denote by \(A_s\) the average number of accesses of storage scheme \(s\) and let \(P = (A_s / A_{\text{opt}} - 1)100\) be the percentage of extra memory accesses beyond the optimal that \(s\) requires. The number of memory accesses for a perfect XOR-scheme is:

\[
A_s = \sum_{T \in T} \sum_{c \in C} a(T_c)^{\alpha(T_c)}
\]

We test heuristics HWCF and MISF using a Monte Carlo simulation. Template sets were generated randomly. Given \(n\), \(p\), and \(t\), we generated randomly \(t\) unique templates consisting of \(p\) unique vectors, selected randomly from a basis of \(n\) vectors. Each template was given a random weight between 1 and \(10^3\), inclusive. Heuristics HWCF and MISF were run on each template set. The results of this simulation are displayed in Fig. 6.

![Fig. 6. Plots of \(P_{\text{HWCF}}\) and \(P_{\text{MISF}}\) for perfect storages.](image)

Average \(P_{\text{HWCF}}\) and \(P_{\text{MISF}}\) are shown for \(3 \leq t \leq 12\) and \(8 \leq p \leq 64\). One thousand cases were run for each instance.
of $P$ and $t$. The speed of the branch-and-bound algorithm prohibited us from completing the figures. The number of distinct vectors of all the generated template bases was fixed at $n = 17$ (17 is prime). Note that, in general, HWCF was outperformed by MICF. Both heuristics degrade in a smooth fashion with increasing numbers of templates and increasing template size. However, for a dozen templates and 16 memory modules, both heuristics deviate on the average by more than 20% from the optimum solution.

The semiperfect heuristic SP has been applied to each of the perfect schemes that are found using HWCF, MICF, and branch-and-bound, respectively. The percentage deviation from branch-and-bound of semiperfect schemes are denoted by $S_{SP}$ and $S_{SPf}$. Fig. 7 shows the plots of $S_{HWCF}$ and $S_{SPf}$. For example, using six templates and 32 memories, the average increasing over the optimum in the access time (MICF) of a template is 0.058 cycles (Fig. 7) if the optimum access is one cycle. The semiperfect heuristic strongly reduces the degree of conflict and decreases its deviation by nearly 50% from that obtained for perfect schemes. This significant improvement is achieved at the cost of incorporating few percents of additional 1's in the storage matrix, i.e., minimizing the number of gates to implement the transformation matrix. For all the studied cases, the cost of upgrading a perfect scheme to semiperfect is below 5% the implementation cost of the perfect scheme.

<table>
<thead>
<tr>
<th></th>
<th>0-4</th>
<th>5-9</th>
<th>10-14</th>
<th>15-19</th>
<th>20-24</th>
<th>25-29</th>
<th>30-34</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWCF &amp; SP</td>
<td>76.2</td>
<td>5.2</td>
<td>3.3</td>
<td>1.9</td>
<td>0.9</td>
<td>0.5</td>
<td>0.2</td>
</tr>
<tr>
<td>MICF &amp; SP</td>
<td>78.7</td>
<td>3.5</td>
<td>1.9</td>
<td>0.9</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 1: Percentages of Test Cases within Performance Ranges

Finally, we show that our approach to perfect storage finds optimum combined address transformations for arbitrary sets of power of two strides. Denote by $s_t = [f_{[t}, f_{2t}, ... , f_{P-1}t]$ the basis of a template that allows conflict-free access to stride $2^t$ with $2^t$ memory modules. Similarly, the basis of stride $2^{s_1}$ is $s_{s_1} = [f_{s_1}, f_{2s_1}, ..., f_{P-1s_1}]$. The bases $s_t$ and $s_{s_1}$ share $p - 1$ canonical vectors that are $f_{s_1}, ..., f_{P-1s_1}$. Therefore, the vectors of $s_t$ and $s_{s_1}$ are p-colorable by using our perfect storage scheme. Moreover, our heuristic approach to perfect storage schemes enables finding an optimum combined address transformation for arbitrary sets of power of two strides.

Analysis of the distribution for the deviations of $S_{HWCF}$ and $S_{SPf}$ indicates (Table 1) that nearly 76% and 78% of the population is within the 4% deviation boundary, respectively. It gives the percentages of test cases (for the entire study) which fell within the given performance ranges. Comparison of the proposed scheme is carried out with respect to traditional row-major interleaving and to a fixed-pattern scheme similar to the one proposed in [6]. The selected fixed-pattern scheme allows conflict-free access to rows, columns, and both diagonals. For the generated templates, the average access time for interleaving ranges from 6 to 18.64 fold the optimum access time for $P = 8, 16, 31, 64$. The average access time of the fixed-pattern scheme ranges from 4.23 to 5.84 fold the optimum access time for the same memories. The fixed-pattern scheme is superior to the row-major scheme, but it is largely outperformed by the semiperfect scheme. This shows that our storage scheme significantly reduces the access time in the case of arbitrary types of templates.

Finally, we show that our approach to perfect storage finds optimum combined address transformations for arbitrary sets of power of two strides. Denote by $s_t = [f_{[t}, f_{2t}, ... , f_{P-1}t]$ the basis of a template that allows conflict-free access to stride $2^t$ with $2^t$ memory modules. Similarly, the basis of stride $2^{s_1}$ is $s_{s_1} = [f_{s_1}, f_{2s_1}, ..., f_{P-1s_1}]$. The bases $s_t$ and $s_{s_1}$ share $p - 1$ canonical vectors that are $f_{s_1}, ..., f_{P-1s_1}$. Therefore, the vectors of $s_t$ and $s_{s_1}$ are p-colorable by using our perfect storage scheme. Moreover, our heuristic approach to perfect storage schemes enables finding an optimum combined address transformation for arbitrary sets of power of two strides.

For example, the address transformation matrix that is found by the perfect storage scheme for strides $\{2^t : 0 \leq t \leq 7\}$ and eight memories is:

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

9 Comparison to Other Approaches

By considering a given set of templates, Fräilong et al. [4] analyzed the conditions for conflict-free access of data patterns. They proved that the columns or the rows of the needed address transformation matrix should be linearly independent. Our approach extends the concepts introduced by the above researchers and proposes an efficient method for combining arbitrary templates within one single address transformation. We have identified the necessary and sufficient conditions that a combined storage scheme should satisfy in order to minimize access conflicts.

Norton and Melton [5] proposed a transformation matrix for the RPI multiprocessor that allows conflict-free access to some power of two strides. Sohl [3] used manually synthesized address transformations (bit-wise) for stride access in vector processors. He proved that few memory buffers can reduce the effects of transient degradation in pipelined
memories when arbitrary strides are accessed. By using bitwise linear transformation matrices, Boppana and Raghavendra [7] proposed a conflict-free storage scheme to the row, column, main-diagonal, and square blocks. While all the above schemes are manually synthesized, we proposed heuristic approaches for automatic (compiler) synthesis of general storage schemes that minimize overall access time of arbitrary data templates. We proved that finding the address transformation for a given set of data patterns is reducible to N-coloring, where N is the number of memory modules. We also proposed heuristics for synthesizing storage schemes for arbitrary data templates. More sophisticated coloring heuristics can be used as well for refining the solution.

10 Conclusions

The performance of SIMD computers can be dramatically affected by the serialization of memory accesses. Perceiving data templates and their access frequency is within compiler capability. The use of this knowledge by a cost-effective heuristic storage scheme has been proposed in an attempt for minimizing access time.

In this work, we generalized the concept of conflict-free access to data patterns in SIMD systems, proposed an efficient approach combining arbitrary templates within one single address transformation, and found the necessary and sufficient conditions for conflict-free access.

Our objective was to find heuristic approaches for automatic synthesis of general storage schemes. For this, we proved that finding the address transformation for a given set of data patterns is reducible to graph coloring. We also proposed heuristics for synthesizing address transformations for arbitrary data patterns. Evaluation of this approach has experimentally proved to be effective in reducing the amount of conflicts while using reasonable implementation cost. The contribution of our work are:

1) a nonredundant XOR-matrix for arbitrary combined templates,
2) use of conflict graphs to represent the optimization problem, and
3) an efficient heuristic for minimizing the access time.

To our knowledge, our work is the first attempt to automate the process of synthesizing dynamic storage schemes that allows minimizing overall access time of arbitrary power-of-two data templates.

The proposed dynamic storage schemes are intended to be part of the processor segment translation table. Since the allocation scheme is invisible to the programmer, reduced algorithm complexity and reduced design time are immediate developments of this research. One possible future extension to this work is to use more sophisticated coloring heuristics, incorporate network and data alignment requirements, and study of application in secondary memory organization.

Acknowledgments

Thanks to Professor Daniel Hirshberg from the Department of Information and Computer Science, University of California, Irvine, for listening critically to the various proofs.

Mayez A. Al-Mouhamed acknowledges support from the Research Committee and the College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia. This work was done during Dr. Al-Mouhamed’s sabbatical leave at the Department of Information and Computer Science, University of California, Irvine.

References


Mayez A. Al-Mouhamed received his BS, MS, and PhD degrees in electrical engineering from the University of Paris XI, Paris, France, in 1975, 1977, and 1982, respectively. Since 1983, he has been on the faculty of the Department of Computer Engineering, College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. His research interests are in computer architecture, parallel architectures and algorithms, robotics, and vision. He has two registered European patents in the area of robotics.

Steven S. Selden received his BS and MS degrees in information and computer science from the University of California, Irvine (UCI), in December 1988 and June 1994, respectively. From February 1990 until September 1991, he was a member of the technical staff at United Research Inc., Irvine, California, where he worked in optimizing compiler design. Since September 1991, he has been a PhD student at UCI. He is expected to complete his PhD in June of 1997. His research interests are in the design and analysis of algorithms, data structures, and, more specifically, he worked on hash functions, online algorithms, and scheduling.