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 20% from the optimum access time in the case of 10 arbitrary data patterns and 16 memories. However, semiperfect 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.

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 *row-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.

- *M.A. Al-Mouhamed is with the Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Box 787, Dhahran 31261, Saudi Arabia. E-mail: mayez@ccse.kfupm.edu.sa.*
- *S.S. Seiden is with the Department of Information and Computer Science, University of California, Irvine, CA 92717. E-mail: sseiden@ics.uci.edu.*

Manuscript received May 9, 1994.
For information on obtaining reprints of this article, please send e-mail to: transpds@computer.org, and reference IEEECS Log Number D95261.

2 Background

Any processor in an SIMD system can access any memory through an interconnection network. Assume there are an equal number $P = 2^p$ 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 $2^{d_1} \times 2^{d_2}$, 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 2^3.

```
for i = 0 to 2^d1 - 1
  for j = 0 to 2^d2 - 2
    c(i, j+1) = c(i, j+1) + c(i, j)
  end
end
```

(a) original program

```
for ii = 0 to 2^(d1-3) - 1, 2^3
  s1: all PE_k r1 ← c(ii + k, 0)
    for j = 0 to 2^d2 - 2
      s2: all PE_k r2 ← c(ii + k, j + 1)
      s3: all PE_k r1 ← r1 + r2
      s4: all PE_k c(ii + k, j + 1) ← r1
    end
end
```

(b) SIMD program

Fig. 1. Example of a loop that requires accessing template $\mathcal{T}_1$.

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_1, s_2, and s_4 require parallel access to the *data template* $\mathcal{T}_1$ shown in Fig. 2.

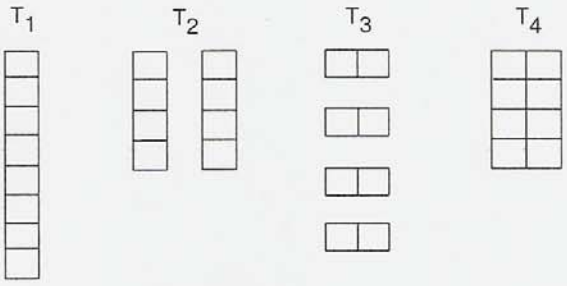

Fig. 2. Example of four templates.

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 left-most 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 $2^d \times 2^d$ such that the element in the ath row and the bth column is denoted by (a, b). The upper left-most element is (0, 0). The sizes of the horizontal and vertical dimensions are both 2^d. An integer x is represented by $x_0x_1 \ldots x_{d-1}$.

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

The Cartesian product of the vector spaces $\mathcal{F}$ and $\mathcal{G}$ s a new vector space $\mathcal{V} = \mathcal{F} \times \mathcal{G}$ with basis $\{f_0, f_1, \ldots f_{d-1}, g_0, g_1, \ldots g_{d-1}\}$. Let $n = 2d$. We denote this combined basis as $V = \{v_0, v_1 \ldots v_{n-1}\}$, where $v_0 = f_0$, $v_1 = f_1$, $v_d = g_0$, etc. This vector space is isomorphic to Z_2^n. Any location (a, b) in memory is uniquely associated with a linear combination of the basis elements $a_0v_0 \oplus \cdots \oplus a_{d-1}v_{d-1} \oplus b_0v_d \oplus \cdots \oplus b_{d-1}v_{n-1}$. Adding two vectors in Z_2^n corresponds to bitwise *exclusive or*. We refer to the elements of the basis V using either fs and gs, or vs, depending on which is notationally convenient.

3 Templates

We define a template $\mathcal{T}_i$ 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 $\mathcal{T}_i$, 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 $V = \{f_0, f_1, f_2, g_0, g_1, g_2\}$, or, alternatively, $V = \{v_0, v_1 \ldots v_5\}$. Consider the template $\mathcal{T}_i$ defined by $T_1 = \{f_0, f_1, f_2\}$. 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 bs are constant, and the as are allowed to vary ($b_0 g_0 \oplus b_1 g_1 \oplus b_2 g_2$ 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 $\mathcal{T}_2$ have basis $T_2 = \{f_0, f_1, g_1\}$. Then, in $\mathcal{T}_2$ all template instances are four elements tall. Since g_0 is omitted, this template skips a column. Thus, we have two 4×1 subarrays, spaced two columns apart. We define $\mathcal{T}_3$ by $T_3 = \{f_1, f_2, g_0\}$. This template is four 1×2 subarrays spaced two rows apart. We let $\mathcal{T}_4$ have basis $T_4 = \{f_0, f_1, g_0\}$. It is a 4×2 subarray. All of these templates are illustrated in Fig. 2.

An XOR-scheme is a linear function $\phi: \mathcal{F} \times \mathcal{G} \mapsto \mathcal{H}$. The function ϕ is represented by a $p \times n$ matrix, which we denote Φ. We apply ϕ to a vector X by matrix multiplication:

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

The ith entry of the jth column of Φ is $\Phi_{i,j}$. (The upper leftmost entry is $\Phi_{0,0}$.) We denote the ith column of Φ by $\Phi_{*,i}$, and the ith row of Φ by $\Phi_{i,*}$. The columns of this matrix represent the values, in terms of the basis H, of ϕ on the members of the basis V, i.e., $\Phi_{*,i}$ is the value of $\phi(v_i)$.

We can also consider ϕ as an ordered set of p functions, $\{\phi_0, \phi_1 \ldots \phi_{p-1}\}$, mapping from $\mathcal{F} \times \mathcal{G}$ to Z_2, where $\phi(X) = \phi_0(X)h_0 \oplus \phi_1(X)h_1 \oplus \cdots \oplus \phi_{p-1}(X)h_{p-1}$. The matrix of ϕ_i is $\Phi_{i,*}$.

Then ϕ allows conflict free access to $\mathcal{T}_i$, if and only if ϕ maps each linear combination of T_i to a unique element of $\mathcal{H}$. Since ϕ is linear, all translations of these linear combinations are also conflict free. In other words, if for one template instance X in $\mathcal{T}_i$, ϕ restricted to X is one to one, then for all template instances X in $\mathcal{T}_i$, ϕ 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 $\phi(X) = \phi_0(X)h_0 \oplus \phi_1(X)h_1 \oplus \cdots \oplus \phi_{p-1}(X)h_{p-1}$, any particular x_i is used at most once, where h_i is the ith 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 $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$. Notice that f_1 appears in all templates bases of this subset. Therefore, let $\phi_0(a_0, a_1, a_2, b_0, b_1, b_2) = a_1$. Further, notice that g_0 appears only in T_3 and f_0 appears only in T_1 and T_2. Therefore, let $\phi_1(a_0, a_1, a_2, b_0, b_1, b_2) = b_0 \oplus a_0$. Finally, notice that g_1 appears only in T_2 and f_2 appears only in T_1 and T_3. We let $\phi_2(a_0, a_1, a_2, b_0, b_1, b_2) = b_1 \oplus a_2$. Let us generalize this procedure.

THEOREM 4.1. *Let Φ be the matrix of a perfect XOR-scheme. If $\Phi_{k,i} = 1$ and $\Phi_{k,j} = 1$ where $i \neq j$, and v_i and v_j are both in T_l, then $\mathcal{T}_l$ cannot not be accessed conflict free.*

PROOF. This is easily seen by a counting argument. ϕ_k can only take on two values. x_i and x_j can each take on two values, for a total of four. Further, because each column of Φ contains at most one nonzero, no other ϕ_k 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. □

Let $T: V \mapsto 2^T$ be a function mapping from a given basis vector to the set of templates bases which contain that vector. More precisely, $T_i \in T(v)$ if and only if $v \in T_i$. We allow $\Phi_{k,i} = 1$ and $\Phi_{k,j} = 1$ only if $T(v_i)$ and $T(v_j)$ are disjoint. The *conflict graph* (V, E) of the template set represents this relationship. The vertices of the graph are the vectors of the basis V. An edge (v_i, v_j) is in E if and only if $i \neq j$ and $T(v_i) \cap T(v_j) \neq \varnothing$. The graph for T_1, T_2, and T_3 is illustrated in Fig. 3.

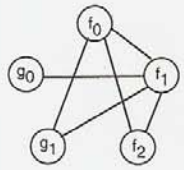

Fig. 3. Conflict graph for templates T_1, T_2, and T_3.

THEOREM 4.2. *For a set of templates $\mathcal{T}$, a conflict free XOR-scheme for P memory units exists, if and only if the conflict graph of the templates is p-colorable.*

PROOF. Suppose the conflict graph is p-colorable. Then, let $\Phi_{i,j} = 1$ if and only if vertex v_j has color i. ϕ is perfect and conflict free, since for all i, if $\Phi_{i,j} = 1$ and $\Phi_{i,k} = 1$, then v_j and v_k cannot be in the same template. Conversely, given a conflict free XOR-scheme ϕ, color the conflict graph of its templates by giving a vertex v_i the color j if and only if $\Phi_{j,i} = 1$. No vertex is assigned more than one color, because each column of Φ contains at most one nonzero entry. No two adjacent vertices are assigned the same color, because this would imply that the XOR-scheme is not conflict free. □

Indeed, the graph in Fig. 3 is three-colorable. The matrix of the perfect XOR-scheme is:

$$\Phi = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & g_0 & g_1 & g_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

However, consider what happens when $\mathcal{T}_4$ is added. The graph of $T_1 \ldots T_4$ is shown in Fig. 4. This graph is not three-colorable, because g_0, f_0, f_1, and f_2 form a clique, and, thus, no perfect XOR-scheme exists. We can prove [8] that finding a perfect XOR-scheme for a set of templates and arbitrary p is NP-complete. Note that two-coloring is polynomial, and, thus, finding a perfect XOR-scheme for four memory units is tractable.

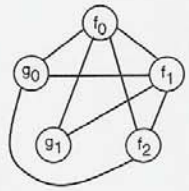

Fig. 4. Conflict graph for templates $T_1 \ldots T_4$.

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:

$$\Phi = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & g_0 & g_1 & g_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

Why does this function allow conflict free access? Consider ϕ restricted to the template $\mathcal{T}_1$. The matrix of this restricted function is:

$$\Phi(T_1) = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{array}.$$

Notice that this matrix has rank 3; its columns are linearly independent. The dimension of the image of ϕ restricted to T_1 is three, and, therefore, conflict free access is assured. We denote ϕ restricted to T_i by $\phi(T_i)$. The matrix of $\phi(T_i)$ is $\Phi(T_i) = \left(\Phi_{*,j}\right)_{v_j \in T_i}$. The matrices of ϕ restricted to $T_2 \ldots T_4$ have all rank 3. In general, we need to find a function ϕ with matrix Φ, such that $\Phi(T_i)$ has rank $|T_i|$, for all templates $\mathcal{T}_i$. How can such XOR-schemes be found? First consider a more general problem.

Suppose we are given:

1) a vector space $Z = Z_2^p$,
2) a set of variables $V = \{v_0, v_1, \ldots v_{n-1}\}$, and
3) a set $T = \{T_1, T_2, \ldots T_t\}$, where each T_i is some member of 2^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 ϕ, represented by a matrix Φ, is semiperfect if and only if for all templates $\mathcal{T}_i$ in $\mathcal{T}$, the matrix of ϕ restricted to $\mathcal{T}_i$ 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 Φ be a matrix over Z_2 with at most one nonzero entry in each column. Let Φ' be defined by:*

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

for some fixed c and k. Then rank(Φ') ≥ rank(Φ).

PROOF. If $\Phi_{c,k} = 1$, then $\Phi' = \Phi$, and, therefore, rank(Φ') = rank(Φ). Otherwise, if some other column $\Phi'_{*,x}$ has a nonzero in row c, then add $\Phi'_{*,x}$ to $\Phi'_{*,k}$ giving a new matrix Φ'', which has the same rank as Φ'. Since this other column must have at most one nonzero entry, the only change to Φ'' is that $\Phi''_{c,k} = 0$. Now $\Phi'' = \Phi$, and, therefore, rank(Φ'') = rank(Φ). If no other column $\Phi_{*,x}$ has a nonzero in row c, then the rank of Φ' is one greater than that of Φ. □

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 $\Phi_{*,i}$ will either be augmented or not. For each template $\mathcal{T}_j$, at most one column of $\Phi(\mathcal{T}_j)$ may be augmented. Given these restrictions, we wish to find a subset of columns to augment such that all $\Phi(\mathcal{T}_j)$ 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 identically colored (assuming that all other edge constraints are met). Thus, the weight of an edge is $\omega(v_i, v_j) = \sum_{v_i, v_j \in T_k} \omega(\mathcal{T}_k)$, where $\omega(\mathcal{T}_k)$ is the access frequency of $\mathcal{T}_k$. The weight of a vertex is defined by $\omega(v_i) = \max_{v_j}\{\omega(v_i, v_j)\}$. The weighted graph for $\mathcal{T}$ is shown in Fig. 5.

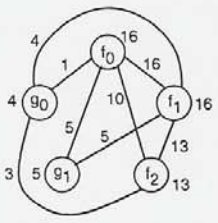

Fig. 5. Weighted conflict graph.

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 updates the cost values of the vertex's neighbors. For this $cost(w, c)$ is augmented by $\omega(w, v)$ 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 expended.

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 $O(p(t+n) + n^2 t)$ time, where t, 2^p, and n, are the number of templates, the number of processors, and the number of distinct vectors of the template bases, respectively.

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 $\Phi_{*,i}$, then a one cannot be added to any column $\Phi_{*,j}$, where $\Phi_{*,i}$ and $\Phi_{*,j}$ are both in some $\Phi(\mathcal{T}_k)$. Or, alternatively, v_i and v_j are both in T_k. Whenever a one is added to a column $\Phi_{*,i}$, mark all $\mathcal{T}_k$ such that v_i is in T_k as *blocked*.

Initially, no columns are blocked. We consider templates in order of their weight (maximal first). $\mathcal{T}_i$ is conflict free, go to the next template. Otherwise, two columns of $\Phi(\mathcal{T}_i)$ are necessarily identical. Adding a nonzero to one of the columns will increase the rank of $\Phi(\mathcal{T}_i)$, 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 $\Phi_{*,j}$. We examine the rows of $\Phi(\mathcal{T}_i)$. If some row k contains only zeros, we set $\Phi_{k,j} = 1$, and add all the vectors (columns) which appear in some template with v_j to the blocked set. We then proceed to the next template. The time complexity [8] of SP is $O\left(p^2 t \log t + \alpha p n\right)$ where α 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 $\omega(\mathcal{T}_i)$ be the number of accesses to $\mathcal{T}_i$. A lower bound $A_{\min}$ on the number of accesses can be defined as $A_{\min} = \sum \omega(\mathcal{T}_i)$. Unfortunately, there is no guarantee that $A_{\min}$ is achievable. Therefore, the optimum access time (A_{opt}) of a perfect scheme, for a given set of templates 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 $P_s = (A_s/A_{\text{opt}} - 1)100$ be the percentage of extra memory accesses beyond the optimal that s requires. The number of memory accesses for a perfect XOR-scheme is:

$$A_s = \sum_{\mathcal{T}_i \in \mathcal{T}} \omega(\mathcal{T}_i) 2^{\left(p - \text{rank}(\Phi(\mathcal{T}_i))\right)}.$$

We tested heuristics HWCF and MICF 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 10^5, inclusive. Heuristics HWCF and MICF were run on each template set. The results of this simulation are displayed in Fig. 6.

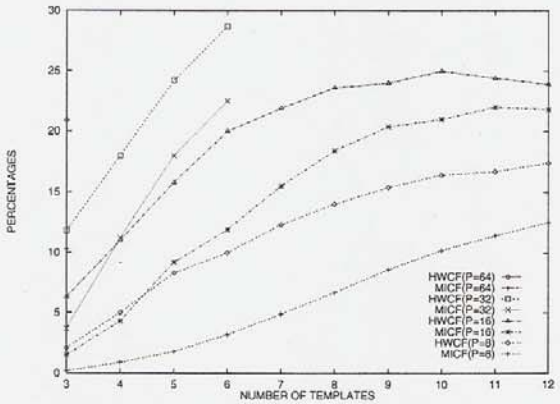

Fig. 6. Plots of P_{HWCF} and P_{MISF} for perfect storages.

Average P_{HWCF} and P_{MISF} are shown for $3 \le t \le 12$ and $8 \le P \le 64$. One thousand cases were run for each instance

TABLE 1
PERCENTAGES OF TEST CASES WITHIN PERFORMANCE RANGES

	0-4	5-9	10-14	15-19	20-24	25-29	30-34
HWCF & SP	76.2	5.2	3.3	1.9	0.9	0.5	0.2
MICF & SP	78.7	3.5	1.9	0.9	0.4	0.2	0.1

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_{HWCF} and S_{MISF}. Fig. 7 shows the plots of S_{HWCF} and S_{MISF}. 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 1s 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.

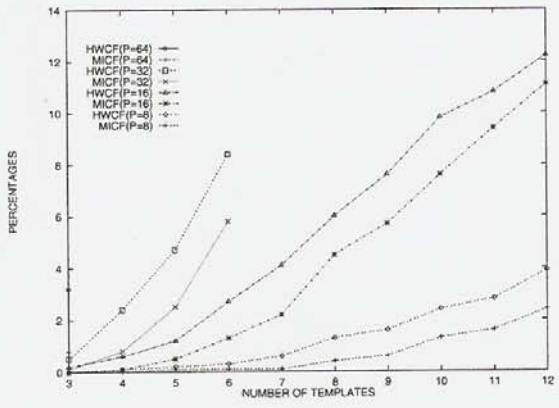

Fig. 7. Plots of S_{HWCF} and S_{MISF} for semiperfect storages.

Analysis of the distribution for the deviations of S_{HWCF} and S_{MISF} 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 $P \in \{8, 16, 31, 64\}$. 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 $s_k = \{f_k, f_{k+1}, \ldots, f_{k+p-1}\}$ the basis of a template that allows conflict-free access to stride 2^k with 2^p memory modules. Similarly, the basis of stride 2^{k+1} is $s_{k+1} = \{f_{k+1}, f_{k+2}, \ldots, f_{k+p}\}$. The bases s_k and s_{k+1} share $p - 1$ canonical vectors that are $f_{k+1}, \ldots, f_{k+p-1}$. Therefore, the vectors of s_k and s_{k+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 $\{2^i : 0 \le i \le 7\}$ and eight memories is:

$$\Phi = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 & f_9 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}.$$

9 COMPARISON TO OTHER APPROACHES

By considering a given set of templates, Frailong 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 RP3 multiprocessor that allows conflict-free access to some power of two strides. Sohi [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

[1] D. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers*, vol. 24, no. 12, pp. 1,145-1,155, Dec. 1975.
[2] P. Budnik and D. Kuck, "The Organization and Use of Parallel Memories," *IEEE Trans. Computers*, vol. 20, no. 12, pp. 1,566-1,569, Dec. 1971.
[3] G.S. Sohi, "High-Bandwidth Interleaved Memories for Vector Processors—A Simulation Study," *IEEE Trans. Computers*, vol. 42, no. 1, pp. 34-44, Jan. 1993.
[4] J.M. Jalby, W. Frailong, and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories," *Proc. Int'l Conf. Parallel Processing*, pp. 276-283, 1985.
[5] A. Norton and E. Melton, "A Class of Boolean Linear Transformations for Conflict-Free Power-of-Two Stride Access," *Proc. Int'l Conf. Parallel Processing*, pp. 247-254, 1987.
[6] K. Batcher, "The Multidimensional Access Memory in STARAN," *IEEE Trans. Computers*, vol. 26, no. 2, pp. 174-177, Feb. 1977.
[7] R.V. Boppana and C.S. Raghavendra, "Efficient Storage Schemes for Arbitrary Size Square Matrices in Parallel Processors with Shuffle-Exchange Networks," *Proc. Int'l Conf. Parallel Processing*, pp. 365-368, 1991.
[8] M. Al-Mouhamed and S. Seiden, "A Cost-Effective Heuristic Storage for Minimizing Access Time of Arbitrary Data Templates," Technical Report ICS-UCI 93-30, Univ. of California, Irvine, June 18, 1993.
[9] T.J. Schaefer, "The Complexity of Satisfiability Problems," *Proc. 10th Ann. Symp. Theory of Computing*, pp. 216-226, 1978.
[10] J. McHugh, *Algorithmic Graph Theory*. Prentice Hall, 1990.
[11] G.J. Chaitan, "Register Allocation and Spilling via Graph Coloring," *ACM SIGPLAN Notices*, vol. 17, no. 2, pp. 201-207, 1982.
[12] B. Bollobàs, *Graph Theory: An Introductory Course*. Springer-Verlag, 1979.

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. Seiden received his BS and MS degrees in information and computer science from the University of California, Irvine (UCI), in December 1989 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.