On the Effectiveness of XOR-Mapping Schemes for Cache Memories

Antonio González*, Mateo Valero*, Nigel Topham† and Joan M. Parcerisa*

* Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
c/ Gran Capità s/n, 08071 Barcelona (Spain)
Email:{antonio,mateo,jmanel}@ac.upc.es

† Department of Computer Science
University of Edinburgh
JCMB, Kings Buildings, Edinburgh (UK)
Email:npt@dcs.ed.ac.uk

Abstract

XOR-mapping schemes were initially proposed in the context of interleaved memories to provide a pseudo-random distribution of memory addresses among multiple memory modules. In that context, the objective was to allow multiple memory references to proceed in parallel by ensuring that they access different memory modules. XOR-mapping schemes can also provide significant benefits for cache memories. In this case, the objective is to avoid conflicts between different data structures that are accessed concurrently during the execution of a program. In spite of this potential benefit, there are very few proposals in the literature that make use of such schemes. This paper evaluates the performance of XOR-mapping schemes for a number of different cache organizations: direct-mapped, set-associative, victim, hash-rehash, column-associative and skewed-associative. It also proposes novel replacement policies for some of these cache organizations. In particular, it presents a low-cost implementation of a pure LRU replacement policy which demonstrates a significant improvement over the pseudo-LRU replacement previously proposed. The paper shows that for a 8 Kbyte data cache, XOR-mapping schemes approximately halve the miss ratio for two-way associative and column-associative organizations. Skewed-associative caches, which already make use of XOR-mapping functions, can benefit from the LRU replacement and also from the use of more sophisticated mapping functions. For two-way associative, column-associative and two-way skewed-associative organizations, XOR-mapping schemes achieve a miss ratio that is not higher than 1.10 times that of a fully-associative cache. XOR mapping schemes also provide a very significant reduction in the miss ratio for the other cache organizations, including the direct-mapped cache. Ultimately, the conclusion of this study is that XOR-based mapping schemes unequivocally provide highly significant performance benefits to most cache organizations.

Keywords: cache memory, XOR-mapping schemes, conflict misses, performance evaluation.

1 Introduction

The idea of using XOR functions to map memory addresses onto a set of memory modules has been studied extensively in the last decade; for example, see [7], [13], [19], [10], [14], [9], [15] and [21]. It has proven to be an effective way to distribute memory addresses to memory modules in a pseudo-random way. In that context, the aim is to allow multiple memory references to proceed in parallel by maximizing the probability that they will access different memory modules. The effect of random distribution can be also beneficial for cache memories if it is used to map memory addresses onto cache data sets. The pseudo-random distribution of addresses improves cache behavior in two important ways:

- Reduces the number of memory conflicts when a data structure is traversed with a stride that is not coprime with the number of sets in the cache.
- Reduces the number of memory conflicts among data structures that are accessed concurrently with strides (distance between consecutive accesses even if they belong to different data structures) that are not coprime with the number of sets.

Despite these potential benefits, there are few proposals in the literature describing XOR-mapping schemes for cache memories. The most notable are the skewed-associative cache [16] [17] and the cache memory of the HP 7100 [1].

In this paper, we present a thorough study of the use of XOR-mapping schemes on a number of different cache organizations: direct-mapped [18], set-associative [18], victim[12], hash-rehash [3], column-associative [4] and skewed-associative [17]. The paper also proposes a low-cost implementation of the LRU replacement policy for use with XOR-mapping functions. It is shown that this replacement policy provides a significant improvement for the column-associative and the two-way skewed-associative cache.

Focussing on miss ratio for the SPEC95 floating point benchmark suite, two different types of XOR-mapping schemes have been evaluated: a simple bitwise XOR of two fields of the address and the polynomial mapping proposed in [15].

For the bitwise XOR scheme, a column-associative cache with LRU replacement, and without swapping, has the lowest miss ratio. This miss ratio is significantly lower than that of a four-way set associative cache and very close to that of a fully-associative cache. A two-way associative cache with an XOR-mapping function yields almost the same hit ratio. We found that a two-way skewed-associative cache has a significantly higher miss ratio when it uses the replacement policy originally proposed by its author. This miss ratio is about the same as that of a victim cache with

an XOR-mapping. However, when using the LRU replacement policy proposed in this paper, the skewed-associative cache achieves a miss ratio very similar to the column-associative and the two-way associative organizations. When swapping is incorporated in a column-associative cache, the overall miss ratio increases slightly, due to the use of the XOR-mapping. However, in this case most of the hits are obtained with a single probe, which may reduce the average access time. A direct-mapped cache exhibits the highest miss ratio, but even in this case, an XOR-mapping function yields very significant improvement.

Polynomial mapping provides marginal advantages for the column-associative and for the two-way associative organizations. However, it is quite effective for the two-way skewed associative cache. With this type of mapping, the two-way skewed-associative cache achieves the lowest miss ratio, which is practically identical to that of a fully associative cache.

Overall, the two-way skewed-associative, column-associative and two-way associative organizations exhibit a similar miss ratio. Miss ratio is not the only parameter to consider when evaluating a cache memory. The most relevant performance metric is the average memory access time, which depends on the access time of the cache memory, the miss ratio and the miss penalty. Genuinely set-associative caches have higher hit times than pseudo-associative caches, though the latter may require two probes to detect a hit. In addition, LRU replacement requires more hardware for the column-associative and skewed-associative organizations. Thus, the most effective organization in practice will depend on the hardware implementation.

The rest of this paper is organized as follows. Section 2 summarizes related work. Some basic concepts are reviewed in section 3, which also describes the evaluation methodology. The performance of conventional mapping functions is evaluated in section 4 for a selection of cache configurations. Section 5 explores the benefits of using XOR-mapping functions in those cache organizations. The implementation of LRU replacement in the presence of an XOR mapping is discussed and evaluated in section 6. The effectiveness of polynomial mapping is analyzed in section 7. Section 8 evaluates the effect of swapping in the column-associative cache. Finally, the main conclusions of this work are summarized in section 9.

2 Related work

There are remarkably few papers on the use of alternative mapping schemes for cache memories. The first computers based on the HP Precision Architecture Processor [6] made use of XOR-mapping functions in order to index the TLB. In these machines, the 11-bit TLB index was obtained by the exclusive OR of two 9-bit fields, one from the virtual page number and the other from the space ID, appended to two other bits of the space ID.

Hashing the process ID with the address bits in order to index the cache memory was evaluated in [2] for a multiprogrammed environment. Results were provided for just one trace, which shown that this scheme could reduce the miss ratio.

In practical systems, like the HP PA 7100, limited and undocumented use of XOR-mapping schemes has occurred, but there is currently no established body of published results analyzing the true benefits of alternative mapping schemes.

More recently, the use of XOR-mapping functions was proposed in skewed-associative caches [16] [17]. A two-way skewed-associative cache consists of two banks of the same size that are accessed simultaneously with two different hashing functions. In that paper, a family of mapping functions was defined as follows. Assume that the cache memory consists of a 2^l lines of 2^b bytes each. A memory address $A=<a_{n-1},a_{n-2},..., a_0>$ comprises the following fields: $A=(A_3, A_2, A_1, A_0)$ such that $A_0=<a_{b-1},..., a_0>$; $A_1=<a_{l+b-2},..., a_b>$; $A_2=<a_{2l+b-3},..., a_{l+b-1}>$; and $A_3=<a_{n-1},..., a_{2l+b-2}>$. Let $\oplus$ denote the bitwise exclusive OR; let $\bullet$ denote the bitwise AND operation and let T be any $(l-1)$-bit number (a good choice for T would be 1010...10); let $\overline{T} = 2^l-1-T$. The family of twin XOR-based mapping functions are defined as:

$$f_0^T: \{0...2^n-1\} \rightarrow \{0...2^{l-1}-1\}$$
$$A=(A_3, A_2, A_1, A_0) \rightarrow ((A_2 \bullet T) \oplus A_1, A_0)$$

$$f_1^T: \{0...2^n-1\} \rightarrow \{0...2^{l-1}-1\}$$
$$A=(A_3, A_2, A_1, A_0) \rightarrow ((A_2 \bullet \overline{T}) \oplus A_1, A_0)$$

In [16], it was proposed a pseudo-LRU replacement by associating a one-bit flag to each line in bank 0. If the requested data is found in bank 0, the corresponding line flag is set, whereas it reset if the data is found in bank 1. On a miss, the flag of the line selected in bank 0 is read and its value determines the bank where the missing data is to be placed.

Using a different workload from that used in this paper, it was observed that the miss ratio of the two-way skewed-associative cache was lower than that of a victim cache (with four lines in the victim buffer) and similar to the miss ratio of a four-way set associative cache.

3 Preliminaries

Whenever a line of main memory is brought into cache a decision must be made on which line, or set of lines, in the cache will be candidates for storing that memory line. This **line placement** policy is one of the least researched aspects of cache design. Direct-mapped caches typically extract a field of l bits from the address and use this to select one line from a set of 2^l. Whilst simple, and trivial to implement, this mapping function is not robust. The principal weakness of this function is its susceptibility to repetitive conflict misses. For example, if C is the cache capacity and B is the line size, then addresses a_1 and a_2 map to the same cache line if $\lfloor a_1/B \rfloor \ mod \ C = \lfloor a_2/B \rfloor \ mod \ C$. If a_1 and a_2 map to the same cache line, then addresses $a_1 + k$ and $a_2 + k$ are guaranteed to also map to identical cache lines, for any integer $k \geq B$. There are two common cases when this happens:

- when accessing a stream of addresses $A = \{a_0, a_1, \ldots, a_m\}$ if a_i collides with a_{i+k}, then there may be up to $(m-k)$ conflict misses in this stream.
- when accessing elements of two distinct arrays b_0 and b_1, if $b_o[i]$ collides with $b_1[j]$ then $b_0[i+k]$ will collide with $b_1[j+k]$, for any integer $k \geq B$.

w-way associativity can be used to alleviate such conflicts. However, if a working set contains $p > w$ conflicts on some cache line, set associativity can only eliminate at most w of those conflicts. Our studies suggest that when conflict misses dominate, the critical factor is not a lack of associativity, but a defective line placement algorithm which fails to disperse data equitably between the available cache lines.

3.1 XOR-mapping schemes

The use of XOR-mapping schemes has been studied extensively in the context of interleaved memories [7], [13], [19], [10], [14], [9], [15] and [21] among others. In this paper we consider two types of XOR-based mapping schemes; those chosen in an *ad hoc* way based on common intuitive notions of how such schemes behave, and a scheme proposed by Rau in [15] which describes a method for constructing XOR mapping schemes based on polynomial arithmetic.

The former type of XOR-mapping computes a cache index by performing a bitwise XOR of two fields of the address of the requested data. We will refer to this type of schemes as *bitwise XOR mapping*. The family of mapping functions proposed in [16] belong to this category.

In this paper we refer to Rau's scheme simply as *polynomial mapping*. Polynomial mapping can be understood by first considering address $A = \langle a_{n-1}, \ldots, a_1, a_0 \rangle$ as a polynomial $A(x) = a_{n-1}x^{n-1}, \ldots, a_1x^1, a_0$, the coefficients of which are in the Galois Field GF(2). The use of polynomial arithmetic, with coefficients restricted in this way, ensures that multiplication and addition of coefficients takes place modulo 2, and thus can be implemented as logical AND and exclusive-OR respectively. The mapping from an address to an l-bit cache index is determined by the polynomial $R(x)$ defined by $A(x) = V(x)P(x) + R(x)$, where $P(x)$ is an irreducible polynomial of order l and $P(x)$ is such that $x^i \mod P(x)$ generates all polynomials of order lower than l. The polynomials that fulfill the previous requirements are called *I-Poly* polynomials. Rau shows how the computation of $R(x)$ can be accomplished by the vector-matrix product of the address and an $n \times l$ matrix H of single-bit coefficients. In GF(2), this product is computed by a network of AND and XOR gates, and if the H-matrix is constant the AND gates can be omitted and the mapping then requires just l XOR gates with fan-in from 2 to n.

The choice of an I-poly polynomial yields properties similar to prime integer modulus functions. Whereas a prime integer modulus function would be prohibitively complex, the I-poly polynomial modulus function has very low complexity; suitable even for computing a cache index.

The use of XOR-mapping schemes requires the computation of several XOR operations to obtain the cache index. Since all the XOR can be done in parallel, the delay of this computation is just one XOR gate. The XOR gates have just two inputs for the bitwise XOR scheme a few more for the polynomial mapping scheme. However, the computation of these XOR operations can be done at the end of the address computation stage of the pipeline. In many current microprocessors, this stage is not the critical stage of the pipeline and therefore this delay may not affect the pipeline cycle time. In addition, the address computation usually computes the address bits from least-significant to most-significant since some kind of carry propagation is required. Since the XOR mapping schemes only use some of the least-significant bits of the address, the XOR gates can operate in parallel with the computation of the most significant-bits and their delay could be completely hidden even if the address computation stage was the critical stage of the pipeline.

3.2 Cache memories

This paper evaluates the performance of XOR-mapping schemes for a number of cache organizations: direct-mapped, two-way associative, victim, hash-rehash, column-associative and two-way skewed-associative. Direct-mapped and set-associative organizations [18] are the most

popular in current microprocessors and we assume that the reader is familiar with them. The two-way skewed-associative cache was described in section 2 . Below there is a short outline of the victim, hash-rehash and column-associative caches.

The hash-rehash cache, proposed by Agarwal *et al.* [3], consists of a conventional direct-mapped cache for which up to two tag probes may be required to find the requested data. First, the cache is accessed with the conventional modulo function, that is using l bits of the address $<a_{b+l-1},a_{b+l-2},..., a_b>$ (2^l is the number of cache lines and 2^b is the line size). If the data is not found, the cache is probed again but with the most significant bit inverted. Thus, the second probe checks the tag for line $<\bar{a}_{b+l-1},a_{b+l-2},..., a_b>$. In case of a second probe hit, the two lines are swapped. Otherwise, the data is brought from the next memory level and it is placed in the first-probe location, whereas the data already there is moved to the second-probe location.

The column-associative cache [4] improves the disappointing miss ratio of the hash-rehash cache by introducing a rehash bit associated with each line. This bit indicates whether the line contains rehashed data, that is, data that is reached in the second probe. When the first probe finds rehashed data, the corresponding line is chosen for replacement. If the rehash bit is zero, then upon a first-time miss the cache is accessed again with the second function. In the case of a second-time hit, the lines are swapped. Otherwise, the data retrieved from memory is placed in the first line and the data already in that line is moved to the line accessed with the second function.

A victim cache [12] consists of a conventional direct-mapped cache with a small fully-associative buffer in the refill path to a second-level cache or main memory. On a cache miss, the line that is evicted from the direct-mapped cache is placed in the victim cache. In the case of a miss in the direct-mapped cache that hits in the victim cache, the lines accessed in both caches are swapped. In the experiments performed in this paper, we assume a victim cache with four lines.

3.3 Evaluation methodology

The results presented in this paper have been obtained through simulation of various data cache organizations using the SPEC 95 floating point benchmark suite. The programs were compiled with the maximum optimization level and instrumented with the ATOM tool [20]. A data cache memory similar to the first-level cache of the Alpha 21164 microprocessor has been assumed: 8 Kilobytes capacity, 32 bytes per line, write-through and no write allocate. For each benchmark we have simulated the first billion (2^{30}) load operations. Because of the no write allocate feature, the performance metrics computed below refer only to load operations.

4 Performance of conventional mapping schemes

Table 1 shows the miss ratio for the following cache organizations: direct-mapped, two-way associative, four-way associative, hash-rehash, column-associative victim and two-way skewed-associative. Of these schemes, only the two-way skewed-associative cache uses an XOR-mapping scheme, as proposed by its author. For comparison, the miss ratio of a fully-associative cache is shown in the penultimate column. The difference between each of columns 2-7 and the fully-associative miss ratio represents the fraction of accesses which result in conflict misses for each listed cache organization. In fact, this difference is slightly negative in the case of 104.hydro2d, due to the sub-optimality of LRU replacement in a fully-associative cache for this particular program. The right-most column shows the conflict miss ratio of the direct-mapped cache. Effectively this represents the target reduction in miss ratio that we hope to achieve through improved mapping schemes.

	direct	**2-way**	**4-way**	**hash-rehash**	**column-assoc.**	**victim**	**2-way skewed**	**fully-assoc.**	**d-m conflicts**
101.tomcatv	53.8	48.1	29.5	51.4	47.0	26.6	22.1	12.5	41.3
102.swim	56.2	59.1	57.1	57.6	53.7	33.7	15.1	7.9	48.3
103.su2cor	11.0	9.1	9.0	11.1	9.3	9.5	9.6	8.9	2.1
104.hydro2d	17.6	17.1	17.3	17.6	17.2	17.0	17.1	17.5	0.1
107.mgrid	3.8	3.6	3.5	6.1	4.2	3.7	4.1	3.5	0.3
110.applu	7.6	6.4	6.0	7.8	6.5	6.9	6.7	5.9	1.7
125.turb3d	7.5	6.5	5.3	7.7	6.4	7.0	5.4	2.8	4.7
141.apsi	15.5	13.3	11.3	18.0	13.4	10.7	11.5	12.5	3.0
145.fpppp	8.5	2.7	2.1	5.9	2.7	7.5	2.2	1.7	6.8
146.wave	31.8	31.7	23.0	35.4	30.7	20.1	16.8	13.9	17.9
Average	**21.32**	**19.76**	**16.42**	**21.87**	**19.11**	**14.27**	**11.05**	**8.71**	**12.61**

Table 1: Miss ratios for the original schemes, with a fully-associative cache for comparison.

From the results in Table 1, we can conclude that set associativity reduces the miss ratio, as expected, although the improvement of a two-way associative cache over a direct-mapped cache is rather low. Comparing the direct-mapped and two-way associative cache with the fully-associative cache suggests that, with the exception of 145.fpppp, these benchmarks show significant clustering in the mapping of memory lines to cache lines under the conventional mapping scheme.

The hash-rehash cache has a miss ratio similar to that of a direct-mapped cache. Although both have similar access times, the hash-rehash scheme requires two cache probes for some hits. Hence, the direct-mapped cache will be more effective. This poor behavior of the hash-rehash cache was

also observed in [4]. The column-associative cache provides a miss ratio similar to that of a two-way associative cache. Since the former has a lower access time but requires two cache probes to satisfy some hits, the choice between these two organization should take into account the particular implementation parameters (access time and miss penalty). The victim cache removes many conflict misses and it out-performs a four-way associative cache. Finally, the two-way skewed-associative cache offers the lowest miss ratio, which is significantly lower than that of a four-way associative cache. The results for the skewed-associative cache are more positive than those observed in [16], where a miss ratio similar to a four-way associative cache was claimed, though using a different workload.

5 Bitwise XOR mapping

XOR-mapping schemes exhibit a behavior which is in some way similar to full associativity but with some restrictions. For instance, in the two-way skewed-associative cache, the set of all addresses that are mapped into the same line of bank 0 are distributed over all the lines in bank 1. Thus, it is similar to having all the lines of bank 1 as alternative locations for a given line in bank 0. However, if one considers a particular memory address, it can be placed in exactly two cache locations (one in bank 0, and the other in bank 1). Below we analyze the performance of bitwise XOR mapping schemes for the other cache organizations. The mapping functions that are evaluated are based on the family of functions proposed in [16]. Section 7 evaluates the performance of the polynomial mapping scheme proposed in [15].

5.1 Direct-mapped

To describe the bitwise XOR mapping function, let us consider a memory address $A=<a_{n-1},a_{n-2},...,a_0>$ composed of the following fields: $A=(A_3, A_2, A_1, A_0)$ such that $A_0=<a_{b-1},..., a_0>$; $A_1=<a_{l+b-1},..., a_b>$; $A_2=<a_{2l+b-1},..., a_{l+b}>$; and $A_3=<a_{n-1},..., a_{2l+b}>$. Let $\oplus$ denote the bitwise exclusive OR. The XOR-based mapping function is defined as follows:

$$f: \quad \{0...2^n-1\} \rightarrow \{0...2^l-1\}$$
$$A=(A_3, A_2, A_1, A_0) \rightarrow (A_2 \oplus A_1, A_0)$$

Table 2 compares the miss ratio of a direct mapped cache with a conventional mapping function to a direct-mapped cache with the mapping function *f* previously defined.

	direct-mapped		
	conventional	**XOR**	**difference**
101.tomcatv	53.8	34.0	-19.8
102.swim	56.2	53.2	-3.0
103.su2cor	11.0	10.5	-0.5
104.hydro2d	17.6	18.3	+0.7
107.mgrid	3.8	4.2	+0.4
110.applu	7.6	8.3	+0.7
125.turb3d	7.5	8.3	+0.8
141.apsi	15.5	13.7	-1.8
145.fpppp	8.5	9.2	+0.7
146.wave	31.8	15.5	-16.3
Average	**21.32**	**17.51**	**-3.81**

Table 2: Miss ratio for a direct mapped cache with the conventional and a bitwise XOR mapping.

It can be seen that the use of an XOR-mapping function provides a large improvement for two of the benchmarks (101 and 146). These are the two benchmarks that also most benefit from a low degree of set-associativity, as can be seen from Table 1. On average, the direct-mapped cache with an XOR-mapping function has a miss ratio lower than that of a column associative cache and almost equal to the miss ratio of a four-way associative cache. Notice however, that five of the ten programs exhibit slightly higher miss ratios. These are all notable for their low conflict miss ratios in a conventional direct-mapped cache. We are seeing the random introduction, with low probability, of conflicts that were not originally present.

5.2 Hash-rehash and column-associative

The mapping functions proposed in [16] for the skewed-associative cache can also be used for a hash-rehash cache and a column associative cache. For these, as for the skewed-associative cache, we define two distinct mapping functions f_0 and f_1. The first probe uses f_0 and, if required, the second probe uses f_1. These functions are as defined in section 2 , using the address decomposition $A=(A_3, A_2, A_1, A_0)$ defined in section 5.1 , and with a binary value of $T = 10101010$.

The cache miss ratios for hash-rehash and column associative caches using f_0 and f_1 are shown in Table 3. We can observe that on average the XOR-mapping functions do not provide any improvement although they are beneficial for two benchmarks (101 and 146). The net deterioration in miss ratio is due to two reasons:

- If reference A produces a cache miss, it is placed in $f_0(A)$. If the data currently in this location corresponds to memory address *B*, it is moved to $f_1(A)$, or discarded. The hash-rehash cache always moves the data, whereas the column-associative cache takes this decision based on the rehash bit. However, it is very likely that $f_1(A) \neq f_0(B)$ and $f_1(A) \neq f_1(B)$. Consequently, the data from address *B* will be moved to a place where it will no longer be accessible and the next reference to B will miss (even if the data is in cache). In addition to degrading performance, this may also cause some consistency problems.
- For a given reference, it may happen that $f_0(A) = f_1(A)$. In this case, reference A does not have an alternative location and we loose the positive effect of pseudo-associativity caused by the use of two mapping functions.

	hash-rehash		**column-associative**	
	original	**XOR**	**original**	**XOR**
101.tomcatv	51.4	38.9	47.0	37.8
102.swim	57.6	63.0	53.7	63.1
103.su2cor	11.1	13.7	9.3	12.7
104.hydro2d	17.6	19.5	17.2	18.4
107.mgrid	6.1	7.1	4.2	4.7
110.applu	7.8	12.6	6.5	11.8
125.turb3d	7.7	19.9	6.4	9.6
141.apsi	18.0	23.9	13.4	15.3
145.fpppp	5.9	11.8	2.7	5.1
146.wave	35.4	25.9	30.7	25.3
Average	**21.87**	**23.63**	**19.11**	**20.39**

Table 3: Miss ratio for the hash-rehash and column-associative caches with the original and the bitwise XOR mapping functions

5.2.1 Enhancing the hash-rehash and column-associative cache

The first problem mentioned above can be solved by inhibiting the swapping of data. Of course, that will cause a significant increase in the percentage of hits that require two probes, but it will provide us with a lower bound on the miss ratio that could be obtained. Besides, swapping may significantly increase pressure on the cache ports, and may cause performance penalties as it is not always possible to hide the swapping during idle cache cycles. For instance, in an ideal out-of-order machine with two memory ports and infinite resources, we have measured that on average two memory ports are busy during 71% of cycles, and only in the 17% of cycles are both idle [8]. An

interesting alternative to swapping is to predict the most likely location of the two possible candidates for a given address. This has been extensively studied by Calder *et al*. who showed that it can be a very effective approach [5].

In order to eliminate the possibility that $f_0(A) = f_1(A)$, we propose to slightly modify the mapping functions such that they always differ in the most significant bit of the result they produce. This most significant bit will be equal to the most significant bit of A_1 for f_0 and it will be inverted for f_1.

The proposed replacement policy is a pseudo-LRU policy inspired by the one proposed in [16]. A one-bit flag is associated with each cache line. When a hit occurs, the flag of the line holding the data is reset to 0 and the flag of the alternate location is set to 1. If a miss occurs, the new line replaces the line whose flag is lower. If both flags are equal, the line at $f_0(A)$ is replaced.

With these changes to the mapping function and replacement policy, and the elimination of swapping, the miss ratios for a column-associative cache are as shown in Table 4.

	column-associative	
	original	**XOR**
101.tomcatv	47.0	20.2
102.swim	53.7	9.7
103.su2cor	9.3	9.2
104.hydro2d	17.2	17.2
107.mgrid	4.2	3.9
110.applu	6.5	6.8
125.turb3d	6.4	5.1
141.apsi	13.4	10.7
145.fpppp	2.7	2.5
146.wave	30.7	15.2
Average	**19.11**	**10.04**

Table 4: Miss ratio for the column-associative caches with the original and the new bitwise XOR mapping function

Notice that with this organization, the effect of the XOR-mapping scheme in the column-associative cache is very impressive, in particular for those programs with the highest miss ratio. The miss ratio of this organization is much lower than that of a four-way associative cache and somewhat lower than that of the skewed-associative cache. To isolate the effect of inverting one bit to obtain always two potential locations for each address, we have performed the simulations just

with the XOR-functions (f_0 and f_1 as defined at the beginning of this section), without the bit inversion and we obtained an average miss ratio of 11.17, which is somewhat higher than that of Table 4.

5.3 Victim cache

In this case, the direct-mapped part uses the XOR-mapping function defined in section 5.1 . The results are shown in Table 5. We can see that the XOR-mapping makes the average miss ratio of the victim cache to be very close to that of the two-way skewed-associative cache. Notice also that the XOR-mapping produces a slight increase in miss ratio for those benchmarks with very few conflict misses. The same behavior was observed for a direct-mapped cache and can be explained again by the random, but infrequent, introduction of new conflict misses.

	victim cache	
	conventional	**XOR**
101.tomcatv	26.6	16.1
102.swim	33.7	21.8
103.su2cor	9.5	9.5
104.hydro2d	17.0	17.3
107.mgrid	3.7	4.0
110.applu	6.9	7.4
125.turb3d	7.0	7.4
141.apsi	10.7	11.1
145.fpppp	7.5	7.9
146.wave	20.1	14.3
Average	**14.3**	**11.6**

Table 5: Miss ratio for the victim cache with the conventional and the bitwise XOR mapping functions

5.4 Two-way associative

In the case of a two-way associative cache, consider an address A composed of four fields $A=(A_3, A_2, A_1, A_0)$ of $n\text{-}2l\text{-}b+2$, $l\text{-}1$, $l\text{-}1$ and b bits respectively. In this case, the XOR-based mapping function is defined as follows:

$$g: \quad \{0...2^n-1\} \rightarrow \{0...2^{l-1}-1\}$$
$$A=(A_3, A_2, A_1, A_0) \rightarrow (A_2 \oplus A_1, A_0)$$

The same mapping function is used to access both banks, as in a conventional set-associative cache, and LRU replacement is used as in this case it can be implemented with low cost. The miss ratios corresponding to this organization are shown in Table 6.

	2-way associative		XOR/full
	conventional	XOR	
101.tomcatv	48.1	17.0	1.36
102.swim	59.1	7.9	1.00
103.su2cor	9.1	9.6	1.07
104.hydro2d	17.1	17.2	0.98
107.mgrid	3.6	3.7	1.06
110.applu	6.4	6.9	1.17
125.turb3d	6.5	4.6	1.64
141.apsi	13.3	11.4	0.91
145.fpppp	2.7	2.7	1.59
146.wave	31.7	14.4	1.04
Average	19.76	9.54	1.10

Table 6: Miss ratios for a two-way associative cache with conventional versus bitwise XOR mapping functions

It can be seen that the bitwise XOR mapping scheme more than halves the miss ratio. For comparison, the rightmost column shows the miss ratio of the two-way associative XOR-mapping scheme relative to the miss ratio of a conventional fully-associative cache. For two programs the two-way XOR cache has lower miss ratio than a fully-associative cache. This is again due to the sub-optimality of LRU replacement in the fully-associative cache, and is a common anomaly in programs with negligible conflict misses.

When a bitwise XOR mapping is used, the average miss ratio of the two-way associative cache is slightly better than that of a column-associative cache and much better than that of the skewed associative cache. This may seem to contradict the results in [16], where Seznec observed that the two-way skewed-associative cache had a lower miss ratio than a two-way associative cache with the same mapping function for both banks. The reason for this difference is twofold. Firstly, Seznec used function f_0^T described in section 2 to index the two-way associative cache. This function indexes the cache using $l - 1 + \lfloor (l-1)/2 \rfloor$ bits, whereas his two-way skewed-associative cache was indexed using $2l - 2$ bits. One of the most important benefits of XOR-mapping schemes is that they avoid conflicts among data structures that are accessed simultaneously with the same stride but whose initial addresses differ in a sum of powers of two. If these powers of two correspond to bits that are used by the mapping function, the conflicts may be avoided. Thus, to be fair, one should

compare cache organizations that use the same number of bits as input to the mapping function. Both the two-way skewed-associative cache in Table 1 and the two-way associative cache in Table 6 use the same number of bits. The second reason for the difference with Seznec's results is that he used a different workload, with a much smaller working set, since his miss ratios are much lower.

The results in Table 6 suggest that in the case of a two-way associative cache, it is more effective the use of more bits in each mapping function than having two different indexing functions.

5.5 Restricted hashing

A drawback of the XOR-mapping scheme is that it may interfere with the use of a physically tagged cache, which may be desirable for coherency reasons [11]. To remove address translation from the critical path it is common to have a virtually-indexed cache with physical address tags. This typically means that the cache is indexed using only unmapped virtual address bits. This constrains the minimum page size to be at least the size of the first-level cache, a practice that is not restrictive given current cache capacities.

However, the XOR-mapping scheme requires the use of more bits of the address and therefore, heightens the constraint on the page size. One way to overcome this problem is to use fewer bits to compute the mapping. In the case of a skewed associative cache, it was shown that this produces a small reduction in performance [16]. Table 7 compares the miss ratio of a column-associative cache using the mapping functions described in section 5.2 with the miss ratio obtained when using only the four least-significant bits of A_2 to perform the bitwise XOR with A_1. It can be observed that the increase in miss ratio is not significant.

6 An affordable implementation of LRU replacement

The use of two different XOR-mapping functions creates an effect similar to full associativity, as previously discussed. This suggests that an LRU replacement policy may be expensive to implement, and has motivated previous work on pseudo-LRU replacement policies [16]. However, implementing LRU replacement in column-associative or skewed-associative caches is not as expensive as in the case of a fully-associative cache. One way to implement LRU for the caches that use two different mapping functions is to add a time stamp to each cache line. A count of memory references is maintained, and every time a cache line is accessed its time stamp is updated

	column-associative	
	full XORing	**partial XORing**
101.tomcatv	20.2	23.2
102.swim	9.7	11.8
103.su2cor	9.2	9.4
104.hydro2d	17.2	17.1
107.mgrid	3.9	3.7
110.applu	6.8	6.8
125.turb3d	5.1	6.1
141.apsi	10.7	12.4
145.fpppp	2.5	2.4
146.wave	15.2	15.5
Average	**10.04**	**10.85**

Table 7: Miss ratio for the column-associative caches with the full and partial bitwise XOR mapping

with the value of the reference counter. When a miss occurs, the candidate for replacement which has the lowest time stamp is chosen for replacement. In the case of a two-way skewed-associative or a column-associative cache this requires a single comparison between two integer fields.

This replacement policy produces a noticeable benefit in the performance of the column-associative cache and the two-way skewed-associative cache, as shown in Table 8, especially for benchmarks 101 and 102.

	column-associative		2-way skewed-associative	
	pseudo-LRU	**LRU**	**pseudo-LRU**	**LRU**
101.tomcatv	20.2	16.4	22.1	20.0
102.swim	9.7	8.6	15.1	12.3
103.su2cor	9.2	9.0	9.6	9.1
104.hydro2d	17.2	17.1	17.1	17.1
107.mgrid	3.9	3.9	4.1	3.9
110.applu	6.8	6.4	6.7	6.3
125.turb3d	5.1	4.6	5.4	4.9
141.apsi	10.7	10.0	11.5	10.5
145.fpppp	2.5	2.5	2.2	2.2
146.wave	15.2	14.6	16.8	16.3
Average	**10.04**	**9.31**	**11.05**	**10.24**

Table 8: Miss ratios for the column-associative cache and the two-way skewed-associative cache comparing pseudo-LRU with LRU replacement.

The cost associated with this LRU replacement depends on the number of bits devoted to the time-stamp. The simulations reported in Table 8 ensure that the time-stamp never overflows. All time stamps were 64-bit integers; sufficient to uniquely identify all simulated memory references, but clearly impractical. A more practical scheme, that uses a small number of bits both in the counter and the time-stamp would work by shifting the counter and all the time-stamps one bit to the right whenever the reference counter overflowed. We simulated this scheme for the column-associative cache using just 8 bits for the counter and the time stamps. The results are practically identical to those obtained with an unrestricted time stamp (the average miss ratio was 9.32).

One potential criticism of our comparison between the column-associative and the skewed-associative caches is that the former uses one bit more of the address to compute the cache index. To isolate this effect we simulated the column-associative cache using $2l-2$ address bits in the mapping function (the same as the skewed-associative cache), and without bit inversion. This produced an average miss ratio of 9.36, indicating no significant difference.

We also investigated the potential benefits of using more bits of the address to compute the mapping and the results are very similar to the ones previously obtained. For instance, using $3l$ bits the average miss ratio is 9.98 for the column-associative cache (slightly worse than for $2l-2$ bits). This suggest that the working set of the benchmarks is not much higher than 2^{2l+b}, i.e., 2 Mbytes.

7 Polynomial mapping

We have investigated the performance of the XOR-mapping scheme proposed by Rau in [15], which is based on polynomial arithmetic and which will be referred to as polynomial mapping. The performance of polynomial mapping has been evaluated for the column associative, the two-way associative and the two-way skewed-associative organizations. For all of them, Table 9 compares the miss ratios of the previous XOR mapping functions based on the bitwise XOR of two bit strings (*XOR*) with that obtained using polynomial mapping functions (*Poly*). In all cases, an LRU replacement is assumed. The miss ratio of a fully-associative cache is also shown for comparison.

To perform a fair comparison we applied the randomization scheme using the same number of bits of the original address as input to all the mapping functions; in all the cases this is 19 bits (14 without considering the bits that indicate the displacement inside the cache line). For the polynomial mapping functions, we chose the I-poly polynomials that require the fewest number of XOR entries for its implementation. The four H-matrices used to perform the polynomial mappings are shown in figure 1. For the column-associative cache, H_1 and H_2 define the mapping of the two

	column-associative		2-way associative		2-way skewed-assoc.		fully-assoc.
	XOR	**Poly**	**XOR**	**Poly**	**XOR**	**Poly**	
101.tomcatv	13.8	12.8	17.0	14.8	20.0	12.6	12.5
102.swim	8.3	7.7	7.9	7.9	12.3	7.5	7.9
103.su2cor	9.1	9.1	9.6	9.9	9.1	9.4	8.9
104.hydro2d	17.1	17.2	17.2	17.1	17.1	17.1	17.5
107.mgrid	4.0	4.2	3.7	3.8	3.9	4.1	3.5
110.applu	6.6	6.5	6.9	6.9	6.3	6.4	5.9
125.turb3d	5.5	6.0	4.6	4.8	4.9	4.2	2.8
141.apsi	10.6	11.2	11.4	11.4	10.5	10.6	12.5
145.fpppp	4.0	2.7	2.7	2.8	2.2	2.3	1.7
146.wave	14.7	13.8	14.4	14.2	16.3	13.7	13.9
Average	**9.36**	**9.12**	**9.54**	**9.37**	**10.24**	**8.78**	**8.71**

Table 9: Miss ratios for a column associative cache, a two-way associative cache and a two-way skewed-associative cache for the two XOR-mapping schemes: bitwise XOR (XOR) and polynomial mapping (Poly).

indexing functions used by this organization. H_3 corresponds to single function utilized by the two-way associative cache. Finally, H_3 and H_4 define the two different mapping functions used by skewed-associative cache.

Regarding the column-associative and the two-way associative results, we can conclude from Table 9 that the scheme based on using polynomial mapping provides a marginal advantage over the bitwise XOR scheme. However, as the former requires wider XOR gates (i.e. more inputs) the simpler XOR scheme may be preferable. The marginal advantage of the polynomial mapping scheme can be explained in a number of ways. Firstly, both schemes are really quite similar; the principal advantage of polynomial mapping is the guarantee of optimal behavior on address patterns that lead to pathological conflict misses in a conventional mapping scheme. Such optimality may not be a feature of bitwise XOR schemes, but pathological cache behavior is also not a dominant feature of the SPEC95 suite. Anyway, both schemes achieve a miss ratio that is very close to that of a fully-associative cache.

On the other hand, the polynomial mapping provides a significant improvement for the skewed-associative cache. For three of the benchmarks (101, 102 and 146) this improvement is quite important. For the others, the reduction in miss ratio is very small, if any, since the miss ratio of the original mapping was already very close to that of a fully-associative cache. Overall, the skewed-associative cache using polynomial mapping and a pure LRU replacement achieves a miss ratio practically identical to that of a fully-associative cache.

$$H_1 = \begin{bmatrix} 1&0&1&1&0&0&0&0 \\ 0&1&0&1&1&0&0&0 \\ 0&0&1&0&1&1&0&0 \\ 0&0&0&1&0&1&1&0 \\ 0&0&0&0&1&0&1&1 \\ 1&1&1&1&1&0&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \qquad H_2 = \begin{bmatrix} 0&0&1&1&1&0&0&1 \\ 1&0&0&0&1&0&1&0 \\ 0&1&0&0&0&1&0&1 \\ 1&0&1&1&0&1&0&0 \\ 0&1&0&1&1&0&1&0 \\ 0&0&1&0&1&1&0&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 1&0&0&0&0&1&1 \\ 1&1&0&0&0&0&0 \\ 0&1&1&0&0&0&0 \\ 0&0&1&1&0&0&0 \\ 0&0&0&1&1&0&0 \\ 0&0&0&0&1&1&0 \\ 0&0&0&0&0&1&1 \\ 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{bmatrix} \qquad H_4 = \begin{bmatrix} 1&1&0&0&1&0&0 \\ 0&1&1&0&0&1&0 \\ 0&0&1&1&0&0&1 \\ 1&0&0&1&0&0&0 \\ 0&1&0&0&1&0&0 \\ 0&0&1&0&0&1&0 \\ 0&0&0&1&0&0&1 \\ 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{bmatrix}$$

Figure 1: H matrices used by the polynomial mapping functions of Table 9.

8 Swapping in the column-associative cache

In the previous sections, the column-associative cache did not incorporate the swapping feature. As a result we can expect a lower miss ratio but a higher percentage of hits requiring two probes. Table 10 compares the performance of the column-associative cache both with and without swapping, using a bitwise XOR mapping scheme taking $2l - 2$ bits. In this case, when a reference to address *A* misses in cache, it is brought to $f_0(A)$. If *B* is the address of the data currently in that location, either it is moved to its alternative location ($f_0(B)$ or $f_1(B)$) or it is discarded if its alternative location has been used more recently. In the same way, when data is found in the second probe ($f_1(A)$) it is moved to $f_0(A)$ and the data currently in this location is moved or discarded following the same criteria as in the case of miss. In any case, data is always placed in an accessible location.

	column-associative	
	without swapping	**with swapping**
101.tomcatv	13.8	16.6
102.swim	8.3	9.4
103.su2cor	9.1	10.1
104.hydro2d	17.1	17.6
107.mgrid	4.0	4.0
110.applu	6.6	7.1
125.turb3d	5.5	6.8
141.apsi	10.6	12.3
145.fpppp	4.0	5.9
146.wave	14.7	17.3
Average miss	**9.36**	**10.70**
% first time hit	**52%**	**96%**

Table 10: Miss ratio for the column-associative cache with bitwise XOR mapping, LRU replacement and with/without swapping

It can be seen that swapping increases the miss ratio by a factor of 1.14, but also ensures that almost all hits can be achieved with a single probe.

9 Conclusions

We have presented a thorough evaluation of XOR-mapping schemes for cache memories using the SPEC 95 floating-point benchmark suite. We have shown that XOR-mapping schemes provide a very high improvement across a broad range of different cache organizations: direct-mapped, set-associative, column-associative and victim cache. We have also evaluated their effect on the hash-rehash cache and presented performance measures of the skewed-associative cache.

We have presented a low-cost implementation of LRU replacement suitable for caches with two or more distinct mapping functions based on XOR-mapping schemes, and shown that it yields significant improvement over previously proposed pseudo-LRU replacement schemes.

Two class of mapping functions have been considered. The first one is based on the bitwise exclusive OR of two bit strings. The second class is the polynomial mapping proposed in [15] in the context of interleaved memories.

For the first class of mapping functions, among the different schemes evaluated, the lowest miss ratio is achieved by the column associative cache, closely followed by the two-way set associative cache, the two-way skewed-associative cache and the victim cache. All of them achieve a miss ratio much lower than that of a conventional four-way associative cache and close to that of a fully-associative cache. For example, a two-way associative cache achieves an average miss ratio that is just 1.09 times that of a fully-associative cache. Similarly, a column-associative cache can achieve a miss ratio between 1.07 and 1.23 times that of a fully-associative cache, depending on whether swapping is implemented. For comparison, a conventional direct-mapped cache has a miss ratio that is 2.45 times that of a fully-associative cache.

Regarding polynomial mapping, we have shown that it provides a marginal advantage over the simpler bitwise XOR schemes for the two-way associative and column-associative organizations. However, for the skewed-associative cache it achieves a significant reduction in miss ratio. Combining the effects of a pure LRU replacement and polynomial mapping, the miss ratio of the two-way skewed associative cache is reduced from 1.27 to 1.01 times that of a fully associative cache.

Comparing the three most effective organizations, i.e., skewed-associative, column-associative and set-associative, we can see that all achieve a very similar miss ratio. Each one may be preferable for different reasons: a skewed-associative has the lowest miss ratio, the column-associative has the lowest hit time and the set-associative requires less hardware to implement a LRU replacement.

In overall, we can conclude that XOR-mapping schemes are an extremely powerful technique for eliminating conflict misses.

References

[1] T. Asprey *et al*., "Performance Features of the PA7100 Microprocessor", *IEEE Micro,* vol. 13, no. 3, June 1993, pp. 22-35.

[2] A. Agarwal, *Analysis of Cache Performance for Operating Systems and Multiprogramming,* Kluwer Academic Publishers, 1989, pp. 120-122.

[3] A. Agarwal, J. Hennessy and M. Horowitz, "Cache Performance of Operating Systems and Multiprogramming", *ACM Trans. on Computer Systems,* vol. 6, Nov. 1988, pp. 393-431.

[4] A. Agarwal and S.D. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches", in *Proc. Int. Symp. on Computer Architecture,* 1993, pp. 179-190.

[5] B. Calder, D. Grunwald and J. Emer, "Predictive Sequential Associative Caches", in *Proc Int. Symp. on High Performance Computer Architecture,* 1996, pp. 244-253.

[6] D.A. Fotland *et al*., "Hardware Design of the First HP Precision Architecture Computeres", *Hewlet-Packard Journal,* vol. 38, no. 3, March 1987, pp. 4-17.

[7] J.M. Frailong, W. Jalby and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories", In *Proc. Int. Conf. on Parallel Processing,* 1985, pp. 276-283.

[8] J. González and A. González, "Identifying Contributing Factors to ILP", to appear in *Proc. Euromicro 96,* 1996.

[9] D.T. Harper III, "Reducing Memory Contention in Shared Memory Multiprocessors", in *Proc. Int. Symp. on Computer Architecture,* 1991, pp. 66-73.

[10] D. Harper III and D. Linebarger, "A Dynamic Storage Scheme for Conflict-Free Vector Access", in *Proc. Int. Symp. on Computer Architecture,* 1989, pp. 72-77.

[11] J.L. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann Publishers, 1996.

[12] N. P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers", in *Proc. Int. Symp. on Computer Architecture,* 1990, pp. 364-373.

[13] A. Norton and E. Melton, "A Class of Boolean Linear Transformations for Conflict-free Power-of-two Stride Access", In *Proc. Int. Conf. on Parallel Processing,* 1987, pp. 247-254.

[14] B.R. Rau, M.S. Schlansker and D.W.L Yen, "The Cydra 5 Stride-Insensitive Memory System", In *Proc Int. Conf. on Parallel Processing,* 1989, pp. 242-246.

[15] B.R. Rau, "Pseudo-Randomly Interleaved Memories", in *Proc. Int. Symp. on Computer Architecture,* 1991, pp. 74-83.

[16] A. Seznec, "A Case for Two-way Skewed-associative Caches", in *Proc. Int. Symp. on Computer Architecture,* 1993, pp. 169-178.

[17] A. Seznec and F. Bodin, "Skewed-associative Caches", In *Proc. Int. Conf. on Parallel Architectures and Languages (PARLE),* 1993, pp. 305-316.

[18] A. J. Smith, "Cache Memories", *ACM Computing Surveys,* vol. 14, no. 4, Sept. 1982, pp. 473-530.

[19] G. S. Sohi, *Logical Data Skewing Schemes for Interleaved Memories in Vector Processors,* Computer Science Technical Report #753, University of Wisconsin-Madison, Sept. 1988.

[20] A. Srivastava and A. Eustace, "ATOM: A System for Building Customized Program Analysis Tools", in *Proc. SIGPLAN Conf. on Programming Language Design and Implementation,* 1994.

[21] M. Valero et al., "Increasing the Number of Strides for Conflict-free Vector Access", in *Proc. Int. Symp. on Computer Architecture,* 1992, pp. 372-381