Randomized Cache Placement for Eliminating Conflicts

Nigel Topham, *Member*, *IEEE Computer Society*, and
Antonio González, *Member*, *IEEE Computer Society*

Abstract—Applications with regular patterns of memory access can experience high levels of cache conflict misses. In shared-memory multiprocessors conflict misses can be increased significantly by the data transpositions required for parallelization. Techniques such as blocking which are introduced within a single thread to improve locality, can result in yet more conflict misses. The tension between minimizing cache conflicts and the other transformations needed for efficient parallelization leads to complex optimization problems for parallelizing compilers. This paper shows how the introduction of a pseudorandom element into the cache index function can effectively eliminate repetitive conflict misses and produce a cache where miss ratio depends solely on working set behavior. We examine the impact of pseudorandom cache indexing on processor cycle times and present practical solutions to some of the major implementation issues for this type of cache. Our conclusions are supported by simulations of a superscalar out-of-order processor executing the SPEC95 benchmarks, as well as from cache simulations of individual loop kernels to illustrate specific effects. We present measurements of Instructions committed Per Cycle (IPC) when comparing the performance of different cache architectures on whole-program benchmarks such as the SPEC95 suite.

Index Terms—Conflict avoidance, cache architectures, performance evaluation.

1 Introduction

If the upward trend in processor clock frequencies during the last 10 years is extrapolated over the next ten years, we will see clock frequencies increase by a factor of 20 during that period [1]. However, based on the current 7 percent per annum reduction in DRAM access times [2], memory latency can be expected to reduce by only 50 percent in the next 10 years. This potential 10-fold increase in the distance to main memory has serious implications for the design of future cache-based memory hierarchies as well as for the architecture of memory devices themselves.

Each block of main memory can be placed in exactly one set of blocks in cache. The chosen set is determined by the *indexing* function. Conventional caches typically extract a field of m bits from the address and use this to select one block from a set of 2^m. While easy to implement, this indexing function is not robust. The principal weakness is its susceptibility to repetitive conflict misses. For example, if C is the number of cache sets and B is the block size, then addresses A_1 and A_2 map to the same cache set if $\mid A_1/B \mid_C = \mid A_2/B \mid_C$. If A_1 and A_2 collide on the same cache set, then addresses $A_1 + k$ and $A_2 + k$ also collide in cache, for any integer k, except when $m_1 < B - \mid k \mid_B \leq m_2$, where $m_1 = \min(\mid A_1 \mid_B, \mid A_2 \mid_B)$ and $m_2 = \max(\mid A_1 \mid_B, \mid A_2 \mid_B)$. There are two common cases when this happens. First, when accessing a stream of addresses $\{A_0, A_1, \ldots, A_m\}$, if A_i collides with A_{i+k}, then there may be up to $m - k$ conflict misses in this stream. Second, when accessing elements of two distinct arrays b_0 and b_1, if $b_0[i]$ collides with $b_1[j]$, then $b_0[i + k]$ collides with $b_1[j + k]$ except under the conditions outlined above. Set-associativity can help to alleviate such conflicts, but is not an effective solution for repetitive and regular conflicts.

One of the best ways to control locality in dense matrix computations with large data structures is to use a tiled (or "blocked") algorithm. This is effectively a reordering of the iteration space which increases temporal locality. However, previous work has shown that the conflicts introduced by tiling can be a serious problem [3]. In practice, until now, this has meant that compilers which tile loop nests really ought to compute the maximal conflict-free tile size for given values of B, major array dimension N, and cache capacity C. Often, this will be too small to make it worthwhile tiling a loop or, perhaps, the value of N will not be known at compile time. Gosh et al. [4] present a framework for analyzing cache misses in perfectly nested loops with affine references. They develop a generic technique for determining optimum tile sizes and methods for determining array padding sizes to avoid conflicts. These methods require solutions to sets of linear Diophantine equations and depend upon there being sufficient information at compile time to find such solutions.

Table 1 highlights the problem of conflict misses with reference to the SPEC95 benchmarks. The programs were compiled with the maximum optimization level and instrumented with the ATOM tool [5]. A data cache similar to the first-level cache of the Alpha 21164 microprocessor was simulated: 8 KB capacity, 32-byte lines, write-through and no write allocate. For each benchmark, we simulated the first 2^{30} load operations. Because of the no-write-

- *N. Topham is with the Institute for Computing Systems Architecture, Division of Informatics, Edinburgh University, Edinburgh, Scotland. E-mail: npt@dcs.ed.ac.uk.*
- *A. González is with the Department of Computer Architecture, Polytechnic University of Catalonia, c/ Jordi Girona 1-3, Modulo D6, 08034 Barcelona, Spain. E-mail: antonio@ac.upc.es.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 108233.

TABLE 1
Cache Miss Ratios for Direct-Mapped (DM), 2-Way Set-Associative (2W), Column-Associative (CA), Victim Cache (VC), 2-Way Skewed Associative (SA), and Fully-Associative (FA) Organizations. Conflict-Miss Ratio (CMR) Is Also Shown.

	DM	2W	CA	VC	SA	FA	CMR
tomcatv	53.8	48.1	47.0	26.6	22.1	12.5	41.3
swim	56.2	59.1	53.7	33.7	15.1	7.9	48.3
su2cor	11.0	9.1	9.3	9.5	9.6	8.9	2.1
hydro2d	17.6	17.1	17.2	17.0	17.1	17.5	0.1
mgrid	3.8	3.6	4.2	3.7	4.1	3.5	0.3
applu	7.6	6.4	6.5	6.9	6.7	5.9	1.7
turb3d	7.5	6.5	6.4	7.0	5.4	2.8	4.7
apsi	15.5	13.3	13.4	10.7	11.5	12.5	3.0
fpppp	8.5	2.7	2.7	7.5	2.2	1.7	6.8
wave	31.8	31.7	30.7	20.1	16.8	13.9	17.9
go	13.4	8.2	8.6	10.9	7.5	4.8	8.6
m88ksim	4.0	1.6	2.4	3.2	1.5	0.7	3.3
gcc	10.6	7.2	7.3	8.6	6.6	5.3	5.3
compress	17.1	15.8	16.3	16.2	14.3	13.0	4.1
li	8.6	5.4	5.5	7.2	4.9	3.8	4.8
ijpeg	4.1	3.3	3.1	2.3	1.9	1.2	2.9
perl	10.7	7.3	7.5	9.3	6.9	5.2	5.5
vortex	5.3	2.7	2.7	3.8	1.8	1.4	3.9
Ave (FP)	21.3	19.7	19.1	14.3	11.0	8.71	12.6
Ave (Int)	9.22	6.44	6.67	7.67	5.66	4.42	4.80
Average	15.9	13.8	13.6	11.3	8.66	6.80	9.14

allocate feature, the tables below refer only to load operations. Table 1 shows the miss ratio for the following cache organizations: direct-mapped, two-way associative, column-associative [6], victim cache with four victim lines [7], and two-way skewed-associative [8], [9].

Of these schemes, only the two-way skewed-associative cache uses an unconventional indexing scheme, as proposed by its author. For comparison, the miss ratio of a fully associative cache is shown in the penultimate column. The miss ratio difference between a direct-mapped cache and that of a fully associative cache is shown in the right-most column of Table 1, and represents the direct-mapped conflict miss ratio (CMR) [2]. In the case of `hydro2d` and `apsi`, some organizations exhibit lower miss ratios than a fully-associative cache due to suboptimality of LRU replacement in a fully-associative cache for these particular programs. Effectively, the direct-mapped conflict miss ratio represents the target reduction in miss ratio that we hope to achieve through improved indexing schemes. The other type of misses, compulsory and capacity, will remain unchanged by the use of randomized indexing schemes. As expected, the improvement of a 2-way set-associative cache over a direct-mapped cache is rather low. The column-associative cache provides a miss ratio similar to that of a two-way set-associative cache. Since the former has a lower access time but requires two cache probes to satisfy some hits, any choice between these two organizations should take into account implementation parameters such as access time and miss penalty. The victim cache removes many conflict misses and outperforms a four-way set-associative cache. Finally, the two-way skewed-associative cache offers the lowest miss ratio. Previous work has shown that it can be significantly more effective than a four-way conventionally indexed set-associative cache [10].

In this paper, we investigate the use of alternative index functions for reducing conflicts and discuss some practical implementation issues. Section 2 introduces the alternative index functions and Section 3 evaluates their conflict avoidance properties. In Section 4, we discuss a number of implementation issues, such as the effect of novel indexing functions on cache access time. Then, in Section 5, we evaluate the impact of the proposed indexing scheme on the performance of a dynamically scheduled processor. Finally, in Section 6, we draw conclusions from this study.

2 Alternative Indexing Functions

The aim of this paper is to show how alternative cache organizations can eliminate repetitive conflict misses. This is analogous to the problem of finding an efficient hashing function. For large secondary or tertiary caches, it may be possible to use the virtual address mapping to adjust the location of pages in cache, as suggested by Bershad et al. [11], thus avoiding conflicts dynamically. However, for small first-level caches, this effect can only be achieved by using an alternative cache index function.

In the field of interleaved memories, it is well-known that bank conflicts can be reduced by using bank selection functions other than the simple modulo-power-of-two. Lawrie and Vora proposed a scheme using prime-modulus functions [12], Harper and Jump [13], and Sohi [14] proposed skewing functions. The use of XOR functions in parallel memory systems was proposed by Frailong et al. [15], and other pseudorandom functions were proposed by Raghavan and Hayes [16] and Rau et al. [17], [18]. These schemes each yield a more or less uniform distribution of requests to banks, with varying degrees of theoretical predictability and implementation cost. In principle, each of these schemes could be used to construct a conflict-resistant cache by using them as the indexing function. However, in cache architectures, two factors are critical: First, the chosen indexing function must have a logically simple implementation and, second, we would like to be able to guarantee good behavior on all regular address patterns—even those that are pathological under a conventional index function.

In the commercial domain, the IBM 3033 [19] and the Amdahl 470 [20] made use of XOR-mapping functions in order to index the TLB. The first generation HP Precision Architecture processors [21] also used a similar technique.

The use of pseudorandom cache indexing has been suggested by other authors. For example, Smith [22] compared a pseudorandom placement against a set-associative placement. He concluded that random indexing had a small advantage in most cases, but that the advantages were not significant. In this paper, we show that, for certain workloads and cache organizations, the advantages can be very large.

Hashing the process ID with the address bits in order to index the cache was evaluated in a multiprogrammed

environment by Agarwal in [23]. Results showed that this scheme could reduce the miss ratio.

Perhaps the most well-known alternative cache indexing scheme is the class of *bitwise exclusive-OR* functions proposed for the skewed associative cache [8]. The bitwise XOR mapping computes each bit of the cache index as either one bit of the address or the XOR of two bits. Where two such mappings are required, different groups of bits are chosen for XORing in each case. A two-way skewed-associative cache consists of two banks of the same size that are accessed simultaneously with two different hashing functions. Not only does the associativity help to reduce conflicts but the skewed indexing functions help to prevent repetitive conflicts from occurring.

The *polynomial modulus* function was first applied to cache indexing in [10]. It is best described by first considering the unsigned integer address A in terms of its binary representation $A = a_{n-1}2^{n-1} + a_{n-1}2^{n-2} + \cdots + a_0$. This is interpreted as the polynomial $A(x) = a_{n-1}x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_0$ defined over the field GF(2). The binary representation of the m-bit cache index R is similarly defined by the GF(2) polynomial $R(x)$ of order less than m such that $A(x) = V(x)P(x) + R(x)$. Effectively, $R(x)$ is $A(x)$ modulo $P(x)$, where $P(x)$ is an irreducible polynomial of order m and $P(x)$ is such that $x^i \mathrm{mod} P(x)$ generates all polynomials of order lower than m. The polynomials that fulfil the previous requirements are called Ipoly polynomials. Rau showed how the computation of $R(x)$ can be accomplished by the vector-matrix product of the address and an $n \times m$ matrix H of single-bit coefficients derived from $P(x)$ [18]. 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 m XOR gates with fan-in from 2 to n. In practice, we may reduce the number of input address bits to the polynomial mapping function by ignoring some of the upper bits in A. This does not seriously degrade the quality of the mapping function.

Ipoly mapping functions have been studied previously in the context of stride-insensitive interleaved memories (see [17], [18]) and have certain provable characteristics of significant value for cache indexing. In [24], it was demonstrated that a skewed Ipoly cache indexing scheme shows a higher degree of conflict resistance than that exhibited by conventional set-associativity or other (non-Ipoly) XOR-based mapping functions. Overall, the skewed-associative cache using Ipoly mapping and a pure LRU replacement policy achieved a miss ratio within 1 percent of that achieved by a fully associative cache. Given the advantage of an Ipoly function over the bitwise XOR function, all results presented in this paper use the Ipoly indexing scheme.

3 Evaluation of Conflict Resistance

The performance of both the integer and floating-point SPEC95 programs has been evaluated for column-associative, two-way set-associative (2W), and two-way skewed-associative organizations using Ipoly indexing functions. In all cases, a single-level cache is assumed. The miss ratios of these configurations are shown in Table 2. Given a conventional indexing function, the direct-mapped (DM) and fully associative (FA) cache organizations display, respectively, the lowest and the highest degrees of conflict resistance of all possible cache architectures. As such, they define the bounds within which novel indexing schemes should be evaluated. Their miss ratios are shown in the right-most two columns of Table 2.

TABLE 2
Miss Ratios for Ipoly Indexing on SPEC95 Benchmarks

	IPOLY indexing					mod 2^k indexing	
	col. assoc.		2W	2-way skewed			
	SPL	LRU		pLRU	LRU	FA	DM
tomcatv⋆	17.2	12.8	14.8	13.1	12.6	12.5	53.8
swim⋆	7.7	7.7	7.9	7.8	7.5	7.9	56.2
su2cor	10.5	9.1	9.9	9.4	9.4	8.9	11.0
hydro2d	17.6	17.2	17.1	17.0	17.1	17.5	17.6
mgrid	5.1	4.2	3.8	4.5	4.1	3.5	3.8
applu	7.3	6.5	6.9	6.8	6.4	5.9	7.6
turb3d	8.1	6.0	4.8	4.5	4.2	2.8	7.5
apsi	12.2	11.2	11.4	11.0	10.6	12.5	15.5
fpppp	4.0	2.7	2.8	2.1	2.3	1.7	8.5
wave⋆	14.6	13.8	14.2	13.9	13.7	13.9	31.8
go	9.6	6.6	8.6	7.5	6.7	4.8	13.4
m88ksim	2.6	1.2	1.9	1.0	0.8	0.7	4.0
gcc	8.2	6.3	7.2	6.7	6.1	5.3	10.6
compress	14.5	13.5	13.7	13.9	13.4	13.0	17.1
li	5.5	4.5	6.1	4.9	4.5	3.8	8.6
ijpeg	1.8	1.3	1.7	1.5	1.4	1.2	4.1
perl	8.5	6.7	8.8	7.1	6.4	5.2	10.7
vortex	2.7	1.7	2.0	1.8	1.6	1.4	5.3
Ave (FP)	10.4	9.12	9.37	9.01	8.78	8.71	21.3
Ave (Int)	6.68	5.22	6.26	5.55	5.09	4.42	9.22
Ave⋆	13.2	11.4	12.3	11.6	11.3	11.4	47.3
Average	8.77	7.39	7.99	7.47	7.14	6.80	15.9

The column-associative cache has access-time characteristics similar to a direct-mapped cache but has some degree of pseudo-associativity—each address can map to one of two locations in the cache but, initially, only one is probed. The column labeled SPL represents a cache which swaps data between the two locations to increase the percentage of a hit on the first probe. It also uses a realistic pseudo-LRU replacement policy. The cache reported in the column labeled LRU does not swap data between columns and uses an unrealistic pure LRU replacement policy [10].

It is to be expected that a two-way set-associative cache will be capable of eliminating many random conflicts. However, a conventionally indexed set-associative cache is not able to eliminate pathological conflict behavior as it has limited associativity and a naive indexing function. The performance of a two-way set-associative cache can be improved by simply replacing the index function, while retaining all other characteristics. Conventional LRU replacement can still be used, as the indexing function has no impact on replacement for this cache organization. For two programs, the two-way Ipoly cache has a lower miss ratio than a fully associative cache. This is again due to the

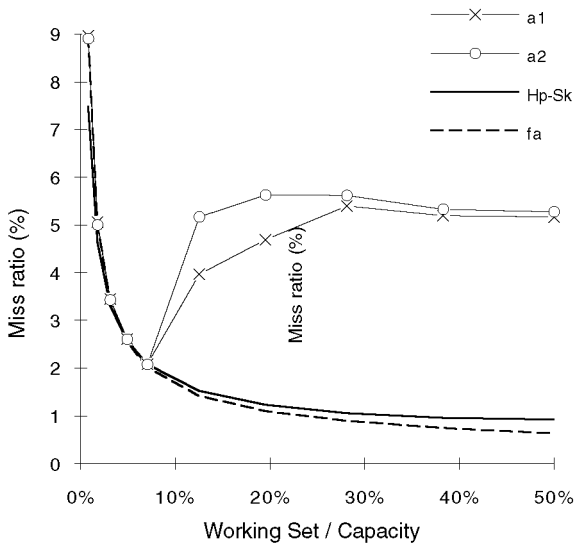

Fig. 1. Miss ratio versus cache loading for 171×171 matrix multiply.

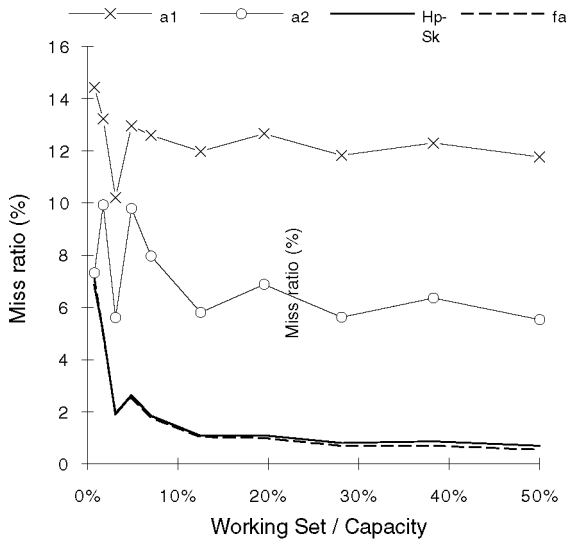

Fig. 2. Miss ratio versus cache loading for 256×256 matrix multiply.

suboptimality of LRU replacement in the fully associative cache and is a common anomaly in programs with negligible conflict misses.

The final cache organization shown in Table 2 is the two-way skewed-associative cache proposed originally by Seznec [8]. In its original form, it used two bitwise XOR indexing functions. Our version uses Ipoly indexing functions, as proposed in [10] and [24]. In this case, two distinct Ipoly functions are used to construct two distinct cache indices from each address. Pure LRU is difficult to implement in a skewed-associative cache, so here we present results for an cache which uses a realistic pseudo-LRU policy (labelled pLRU) and a cache which uses an unrealistic pure LRU policy (labeled LRU). This organization produces the lowest conflict miss ratio, down from 4.8 percent to 0.67 percent for SPECint, and from 12.61 percent to 0.07 percent for SPECfp.

It is striking that the performance improvement is dominated by three programs (`tomcatv`, `swim`, and `wave`). These effectively exhibit pathological conflict miss ratios under conventional indexing schemes. Studies by Olukotun et al. [25], have shown that the data cache miss ratio in `tomcatv` wastes 56 percent and 40 percent of available IPC in 6-way and 2-way superscalar processors, respectively.

Tiling will often introduce extra cache conflicts, the elimination of which is not always possible through software. Now that we have alternative indexing functions that exhibit conflict avoidance properties we can use these to avoid these induced conflicts. The effectiveness of Ipoly indexing for tiled loops was evaluated by simulating the cache behavior of a variety of tiled loop kernels. Here, we present a small sample of results to illustrate the general outcome. Figs. 1 and 2 show the miss ratios observed in two tiled matrix multiplication kernels, where the original matrices were square and of dimensions 171 and 256, respectively. Tile sizes were varied from 2×2 up to 16×16 to show the effect of conflicts occurring in caches that are direct-mapped (a1), 2-way set-associative (a2), fully associative (fa), and skewed 2-way Ipoly (Hp-Sk). The tiled working set divided by cache capacity measures the fraction of the cache occupied by a single tile. Cache capacity is 8 KBytes, with 32-byte lines.

For dimension 171, the miss ratio initially falls for all caches as tile size increases. This is due to increasing spatial locality, up to the point where self-conflicts begin to occur in the conventionally indexed direct-mapped and two-way set-associative caches. The fully associative cache suffers no self-conflicts and its miss ratio decreases monotonically to less than 1 percent at 50 percent loading. The behavior of the skewed 2-way Ipoly cache tracks the fully associative cache closely. The qualitative difference between the Ipoly cache and a conventional two-way cache is clearly visible.

For dimension 256, the product array and the multiplicand array are positioned in memory so that cross-conflicts occur in addition to self-conflicts. Hence, the direct-mapped and 2-way set associative caches experience little spatial locality. However, the Ipoly cache is able to eliminate cross-conflicts, as well as self-conflicts, and it again tracks the fully associative cache.

4 Implementation Issues

The logic of the GF(2) polynomial modulus operation presented in Section 2 defines a class of hash functions which compute the cache placement of an address by combining subsets of the address bits using XOR gates. This means that, for example, bit 0 of the cache index may be computed as the XOR of bits 0, 11, 14, and 19 of the original address. The choice of polynomial determines which bits are included in each set. The implementation of such a function for a cache with an 8-bit index would require just eight XOR gates with fan-in of 3 or 4.

While this appears remarkably simple, there is more to consider than just the placement function. First, the function itself uses address bits beyond the normal limit imposed by typical minimum page size restriction. Second, the use of pseudorandom placement in a multilevel memory hierarchy has implications for the maintenance of Inclusion. In [24], we explain these two issues in more depth

and show how the virtual-real two-level cache hierarchy proposed by Wang et al. [26] provides a clean solution to both problems.

A cache memory access in a conventional organization normally computes its effective address by adding two registers or a register plus a displacement. Ipoly indexing implies additional circuitry to compute the index from the effective address. This circuitry consists of several XOR gates that operate in parallel and therefore the total delay is just the delay of one gate. Each XOR gate has a number of inputs that depend on the particular polynomial being used. For the experiments reported in this paper, the number of inputs is never higher than 5. The XOR gating required by the Ipoly mapping may increase the critical path length within the processor pipeline. However, any delay will be short since all bits of the index can be computed in parallel. Moreover, we show later that, even if this additional delay induces a full cycle penalty in the cache access time, the Ipoly mapping provides a significant overall performance improvement. *Memory address prediction* can be also used to avoid the penalty introduced by the XOR delay when it lengthens the critical path. Memory addresses have been shown to be highly predictable. For instance, in [27], it was shown that the addresses of about 75 percent of the dynamically executed memory instructions from the SPEC95 suite can be predicted with a simple tabular scheme which tracks the last address produced by a given instruction and its last stride. A similar scheme that could be used to give an early prediction of the line that is likely to be accessed by a given load instruction is outlined below.

The processor incorporates a table indexed by the instruction address. Each entry stores the last address and the predicted stride for some recently executed load instruction. In the fetch stage, this table is accessed with the program counter. In the decode stage, the predicted address is computed and the XOR functions are performed to compute the predicted cache line. This can be done in one cycle since the XOR can be performed in parallel with the computation of the most significant bits of the effective address. When the instruction is subsequently issued to the memory unit, it uses the predicted line number to access the cache in parallel with the actual address and line computation. If the predicted line turns out to be incorrect, the cache access is repeated with the actual address. Otherwise, the data provided by the speculative access can be loaded into the destination register.

A number of previous papers have suggested address prediction as a means to reduce memory latency [28], [29], [30], or to execute memory instructions and their dependent instructions speculatively [31], [27], [32]. In the case of a miss-speculation, a recovery mechanism similar to that used by branch prediction schemes is then used to squash miss-speculated instructions.

5 Effect of Ipoly Indexing on IPC

In order to verify the impact of polynomial mapping on realistic microprocessor architectures, we have developed a parametric simulator for a four-way superscalar processor with out-of-order execution. Table 3 summarizes the functional units and their latencies used in these experiments. The reorder buffer contained 32 entries, and there were two separate physical register files (FP and Integer), each with 64 physical registers. The processor had a lockup-free data cache [33] that allowed eight outstanding misses to different cache lines. Cache capacities of 8 KB and 16 KB were simulated with 2-way associativity and 32-byte lines. The cache was write-through and no-write-allocate. The cache had two ports, each with a two-cycle hit time and a miss penalty of 20 cycles. This was connected by a 64-bit data bus to an infinite level-two cache. Data dependencies through memory were speculated using a mechanism similar to the arb of the Multiscalar [34] and the HP PA-8000 [35]. A branch history table with 2K entries and 2-bit saturating counters was used for branch prediction.

The memory address prediction scheme was implemented by a direct-mapped table with 1K entries, indexed by instruction address. To reduce cost, the entries were not tagged, although this increases interference in the table. Each entry contained the last effective address of the most recent load instruction to index into that table entry, together with the last observed stride. In addition, each entry contained a 2-bit saturating counter to assign confidence to the prediction. Only when the most-significant bit of the counter is set would the prediction be considered correct. The address field was updated for each new reference, regardless of the prediction. However, the stride field was updated only when the counter went below 10_2, i.e., after two consecutive mispredictions.

Table 4 shows the IPC and miss ratios for six configurations.[1] All IPC averages are computed using an equally-weighted harmonic mean. The baseline configuration is an 8 KB cache with conventional indexing and no address prediction (NP, third column). The average IPC for this configuration is 1.27 from an average miss ratio of 16.53 percent. With Ipoly indexing, the average miss ratio falls to 9.68 percent. If the XOR gates are not in the critical path, IPC rises to 1.33 (NX, fifth column). Conversely, if the XOR gates are in the critical path and a one cycle penalty in the cache access time is assumed, the resulting IPC is 1.29 (WX, sixth column). However, if memory address prediction is then introduced (WP, seventh columnn), IPC is the same as for a cache without the XOR gates in the critical path (NX). Hence, the memory address prediction scheme can offset the penalty introduced by the additional delay of the XOR gates when they are in the critical path, even under the conservative assumption that a whole cycle of latency is added to each load instruction. Finally, Table 4 also shows the performance of a 16 KB 2-way set-associative cache *without* Ipoly indexing (second column). Notice that the addition of Ipoly indexing to an 8 KB cache yields over 60 percent of the IPC increase that can be obtained by doubling the cache size.

These IPC measurements exhibit small absolute differences, but this is because the benefit of Ipoly indexing is perceived by a only small subset of the benchmark programs. Most programs in SPEC95 exhibit low conflict

1. For each configuration we simulated 10^8 instructions after skipping the first 2×10^9.

TABLE 3
Functional Units and Instruction Latencies

Functional unit	Latency	Repeat rate
1 simple integer	1	1
1 complex integer	9 (mult) 67 (div)	1 67
2 eff. address	1	1
1 simple FP	4	1
1 FP mult	4	1
1 FP div/sqrt	16 (div) 35 (sqrt)	16 35

miss ratios. In fact, the SPEC95 conflict miss ratio for an 8 KB 2-way set-associative cache is less than 4 percent for all programs except `tomcatv`, `swim`, and `wave5`. The two penultimate rows of Table 4 show independent IPC averages for the benchmarks with high conflict miss ratios (Ave ☆) and those with low conflict miss ratios (Ave †). This highlights the ability of polynomial mapping to reduce the miss ratio and significantly boost the performance of problem cases. One can see that the polynomial mapping provides a significant 27 percent improvement in IPC for the three bad programs even if the XOR gates are in the critical path and memory address prediction is not used. With memory address prediction, Ipoly indexing yields an IPC improvement of 33 percent compared with that of a conventional cache of the same capacity and 16 percent higher than that of a conventional cache with twice the capacity. Notice that the polynomial mapping scheme with prediction is even better than the organization without prediction, where the XOR gates do not extend the critical path. This is due to the fact that the memory address prediction scheme reduces by one cycle the effective cache hit time when the predictions are correct, since the address computation is overlapped with the cache access (the computed address is used to verify that the prediction was correct). However, the main benefits observed come from the reduction in conflict misses. To isolate the different effects, we have also simulated an organization with the memory address prediction scheme and conventional indexing for an 8 KB cache (WP, column 4). If we compare the IPC of this organization with that in column 3, we see that the benefits of the memory address prediction scheme due solely to the reduction of the hit time are almost negligible. This confirms that the improvement observed in the Ipoly indexing scheme with address prediction derives from the reduction in conflict misses. The averages for the 15 programs which exhibit low levels of conflict misses show a small (1.7 percent) deterioration in average IPC when Ipoly indexing is used and the XOR gates are in the critical path. This is due to a slight increase in the average hit time rather than an overall increase in miss ratio (which, on average, falls by 2 percent). For these programs, the reduction in aggregated miss penalty does not outweigh the slight extension in critical path length.

TABLE 4
Comparative IPC Measurements (Simulated)

	mod 2^k indexing			Ipoly indexing		
	16 KB	8 KB		8 KB		
		NP	WP	NX	WX	
					NP	WP
tomcatv⋆	1.18	1.03	1.04	1.33	1.30	1.36
swim⋆	1.30	1.06	1.08	1.53	1.49	1.57
su2cor†	1.28	1.24	1.26	1.24	1.21	1.25
hydro2d†	1.14	1.13	1.15	1.13	1.11	1.15
mgrid†	1.63	1.61	1.63	1.57	1.55	1.59
applu†	1.51	1.50	1.53	1.50	1.46	1.52
turb3d†	1.85	1.80	1.82	1.81	1.78	1.82
apsi†	1.13	1.08	1.09	1.08	1.07	1.09
fpppp†	2.14	2.00	2.00	1.98	1.93	1.94
wave⋆	1.37	1.26	1.28	1.51	1.48	1.54
go†	1.00	0.87	0.88	0.87	0.83	0.84
m88ksim†	1.56	1.53	1.53	1.52	1.49	1.51
gcc†	1.16	1.04	1.05	1.03	0.98	0.99
compress†	1.13	1.12	1.13	1.11	1.07	1.10
li†	1.40	1.30	1.32	1.33	1.26	1.31
ijpeg†	1.31	1.28	1.28	1.29	1.28	1.30
perl†	1.45	1.26	1.27	1.24	1.19	1.21
vortex†	1.39	1.27	1.28	1.30	1.25	1.27
Ave (FP)	1.42	1.34	1.35	1.44	1.41	1.46
Ave (Int)	1.29	1.19	1.20	1.20	1.15	1.17
Ave ⋆	1.28	1.11	1.13	1.46	1.42	1.49
Ave †	1.38	1.30	1.32	1.30	1.27	1.30
Average	1.36	1.27	1.28	1.33	1.29	1.33

6 Conclusions

In this paper, we have discussed the problem of cache conflict misses and surveyed the options for reducing or eliminating those conflicts. We have described pseudorandom indexing schemes based on polynomial modulus functions and have shown them to be robust enough to virtually eliminate the repetitive cache conflicts caused by bad strides inherent in some SPEC95 benchmarks, as well as eliminating those introduced into an application by the tiling of loop nests.

We have highlighted the major implementation issues that arise from the use of such novel indexing schemes. For example, Ipoly indexing uses more address bits than a conventional cache to compute the cache index. Also, the use of different indexing functions at level-1 and level-2 caches results in the occasional eviction at level-1 simply to maintain Inclusion. We have explained that both of these problems can be solved using a two-level virtual-real cache hierarchy. Finally, we have proposed a memory address prediction scheme to avoid the penalty due to the small potential delay in the critical path introduced by the pseudorandom indexing function.

Detailed simulations of an out-of-order superscalar processor have demonstrated that programs with significant numbers of conflict misses in a conventional 8 KB 2-

way skewed-associative cache perceive IPC improvements of 33 percent (with address prediction) or 27 percent (without address prediction). This is up to 16 percent higher than the IPC improvements obtained simply by doubling the cache capacity. Furthermore, from the programs we analyzed, those that do not experience significant conflict misses on average see only a 1.7 percent reduction in IPC when Ipoly indexing appears on the critical path for computing the effective address, and address prediction is used. If the indexing logic does not appear on the critical path, no deterioration in overall average performance is experienced by those programs.

We believe the key contribution of pseudorandom indexing is the resulting predictability of cache behavior. In our experiments, we found that Ipoly indexing reduces the standard deviation of miss ratios across SPEC95 from 18.49 to 5.16. This could be beneficial in real-time systems where unpredictable timing, caused by the possibility of pathological miss ratios, presents problems. If conflict misses are eliminated, the miss ratio depends solely on compulsory and capacity misses which, in general, are easier to predict and control. Conflict avoidance could also be beneficial when iteration-space tiling is used to improve data locality.

ACKNOWLEDGMENTS

This work was supported in part by the European Commission ESPRIT project 24942, by the British Council (grant 1016), and by the Spanish Ministry of Education (Acción Integrada Hispano-Británica 202 CYCIT TIC98-0511). The authors would like to express their thanks to Jose González and Joan Manuel Parcerisa for their help with the simulation software and to the anonymous referees for their helpful comments.

REFERENCES

[1] Semiconductor Industry Assoc., "The National Technology Roadmap for Semiconductors," 1994.
[2] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan-Kauffman 1996.
[3] M. Lam, E. Rothberg, and M. Wolf, "The Cache Performance and Optimization of Blocked Algorithms," *Proc. ASPLOS IV,* pp. 63-74, 1991.
[4] S. Gosh, M. Martonosi, and S. Malik, "Cache Miss Equations: An Analytic Representation of Cache Misses," *Proc. Int'l Conf. Supercomputing,* Vienna, Austria, pp. 317-342, July 1997.
[5] A. Srivastava and A. Eustace, "ATOM: A System for Building Customized Program Analysis Tools," *Proc. SIGPLAN Conf. Programming Language Design and Implementation,* 1994.
[6] A. Agarwal and S. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches," *Proc. Int'l Symp. Computer Architecture,* pp. 179-190, 1993.
[7] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. Int'l Symp. Computer Architecture,* pp. 364-373, 1990.
[8] A. Seznec, "A Case for Two-Way Skewed Associative Caches," *Proc. Int'l Symp. on Computer Architecture,* pp. 169-173, 1993.
[9] A. Seznec, F. Bodin, "Skewed-Associative Caches," *Proc. Int'l Conf. Parallel Architectures and Languages (PARLE),* pp. 305-316, 1993.
[10] A. González, M. Valero, N. Topham, and J. Parcerisa, "Eliminating Cache Conflict Misses Through XOR-Based Placement Functions," *Proc. Int'l Conf. Supercomputing,* Vienna, Austria, pp. 76-83, July 1997.
[11] B. Bershad, D. Lee, T. Romer, and J. Chen, "Avoiding Cache Conflict Misses Dynamically in Large Direct-Mapped Caches," *Proc. ASPLOS VI,* pp. 158-170, 1994.
[12] D. Lawrie and C. Vora, "The Prime Memory System for Array Access," *IEEE Trans. Computers,* vol. 31, no. 5, pp. 435-442, May 1982.
[13] D.T. Harper III and J.R. Jump, "Vector Access Performance in Parallel Memories," *IEEE Trans. Computers,* vol. 36, no. 12, pp. 1,440-1,449, Dec. 1987.
[14] G. Sohi, "Logical Data Skewing Schemes for Interleaved Nemories in Vector Processors," Technical Report 753, Univ. of Wisconsin-Madison, 1988.
[15] J. Frailong, W. Jalby, and J. Lenfant, "XOR-Schemes: A Flexible Data Organization in Parallel Memories," *Proc. Int'l Conf. Parallel Processing,* pp. 276-283, 1985.
[16] R. Raghavan and J. Hayes, "On Randomly Interleaved Memories," *Proc. Supercomputing '90,* pp. 49-58, 1990.
[17] B. Rau, M. Schlansker, and D. Yen, "The Cydra 5 Stride-Insensitive Memory System," *Proc. Int'l Conf. Parallel Processing,* pp. 242-246, 1989.
[18] B. Rau, "Pseudo-Randomly Interleaved Memories," *Proc. Int'l Symp. Computer Architecture,* pp. 74-83, 1991.
[19] IBM *IBM 3033 Processor Complex: Theory of Operations Manual—Processor Storage Control Function, vol. 4,* 1978.
[20] Amdahl Corp. *470V/6 Machine Reference Manual,* 1976.
[21] D.A. Fotland, et al., "Hardware Design of the First HP Precision Architecture Computers," *Hewlett-Packard J.,* vol. 38, pp. 4-17, Mar. 1987.
[22] A. Smith, "Cache Memories," *ACM Computing Surveys,* vol. 14, pp. 473-530, Sept. 1982.
[23] A. Agarwal, *Analysis of Cache Performance for Operating Systems and Multiprogramming,* pp. 120-122. Kluwer Academic, 1989.
[24] N. Topham, A. González, and J. González, "The Design and Performance of a Conflict-Avoiding Cache," *Proc. Int'l Symp. Microarchitecture,* pp. 71-80, Dec. 1997.
[25] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The Case for a Single-Chip Multiprocessor," *Proc. ASPLOS-VII,* Oct. 1996.
[26] W.-H. Wang, J.-L. Baer, and H. Levy, "Organization and Performance of a Two-Level Virtual-Real Cache Hierarchy," *Proc. Int'l Symp. Computer Architecture,* 1989.
[27] J. González and A. González, "Speculative Execution Via Address Prediction and Data Prefetching," *Proc. Int'l Conf. Supercomputing,* Vienna, Austria, pp. 196-203, July 1997.
[28] M. Golden and T. Mudge, "Hardware Support for Hiding Cache Latency," Technical Report CSE-TR-152-93, Univ. of Michigan, 1993.
[29] T. Austin, D. Pnevmastikatos, and G. Sohi, "Streamlining Data Cache Access with Fast Address Calculation," *Proc. Int'l Symp. Computer Architecture,* pp. 369-380, 1995.
[30] T. Austin, G. Sohi, "Zero-Cycle Loads: Microarchitecture Support for Reducing Load Latency," *Proc. Int'l Symp. Microarchitecture,* pp. 82-92, 1995.
[31] J. González and A. González, "Memory Address Prediction for Data Speculation," *Proc. EUROPAR '97,* pp. 1,084-1,091, 1997.
[32] Y. Sazeides, S. Vassiliadis, and J. Smith, "The Performance Potential of Data Dependence Speculation and Collapsing," *Proc. Int'l Symp. Microarchitecture,* pp. 238-257, Dec. 1996.
[33] D. Kroft, "Lockup-Free Instruction Fetch/Prefetch Cache Organization," *Proc. Int'l Symp. Computer Architecture,* pp. 81-87, 1981.
[34] M. Franklin and G. Sohi, "ARB: A Mechanism for Dynamic Reordering of Memory References," *IEEE Trans. Computers,* vol. 45, no. 5, pp. 552-571, May 1996.
[35] D. Hunt, "Advanced Performance Features of the 64-bit PA-8000," *Proc. CompCon '95,* pp. 123-128, 1995.

Nigel Topham received his BSc degree in computer science in 1982 and his PhD in computer science in 1985, both from the University of Manchester, England. He is currently a lecturer in the Division of Informatics at Edinburgh University, Scotland, where he is also the director of the Institute for Computing Systems Architecture. His research interests include the architecture of processor and memory systems, the interactions between compilers and architectures, and software tools for prototyping compiler optimizations. Dr. Topham is a member of the IEEE Computer Society.

Antonio González received his degree in computer science in 1986 and his PhD in computer science in 1989, both from the Polytechnic University of Catalonia, Barcelona, Spain. He is currently an associate professor in the Computer Architecture Department at the Polytechnic University of Catalonia. His research interests center on computer architecture, compilers, and parallel processing, with a special emphasis on processor microarchitecture, memory hierarchy, and instruction scheduling. Dr. González is a member of the IEEE Computer Society and the ACM.