Utilizing Reuse Information in Data Cache Management

Jude A. Rivers, Edward S. Tam, Gary S. Tyson, Edward S. Davidson
Advanced Computer Architecture Laboratory
The University of Michigan
Ann Arbor, Michigan 48109
{jrivers,estam,tyson,davidson}@eecs.umich.edu

Matt Farrens
Dept. of Computer Science
University of California at Davis
Davis, California 95616
farrens@cs.ucdavis.edu

1. ABSTRACT

As microprocessor speeds continue to outgrow memory subsystem speeds, minimizing the average data access time grows in importance. As current data caches are often poorly and inefficiently managed, a good management technique can improve the average data access time. This paper presents a comparative evaluation of two approaches that utilize reuse information for more efficiently managing the first-level cache. While one approach is based on the effective address of the data being referenced, the other uses the program counter of the memory instruction generating the reference.

Our evaluations show that using effective address reuse information performs better than using program counter reuse information. In addition, we show that the Victim cache performs best for multi-lateral caches with a direct-mapped main cache and high L2 cache latency, while the NTS (effective-address-based) approach performs better as the L2 latency decreases or the associativity of the main cache increases.

1.1 Keywords

Multi-lateral caches, effective address, program counter

2. Introduction

Minimizing the *average data access time* is of paramount importance when designing high-performance machines. Unfortunately, access time to off-chip memory (measured in processor clock cycles) has increased dramatically as the disparity between main memory access times and processor clock speeds widen. And as multiple-issue processors continue to increase the number of instructions that can be issued each cycle, this problem is growing in magnitude.

There are many approaches to minimizing the average data access time. The most common is to incorporate multiple levels of cache memory on-chip, but still allocate and replace their blocks, in a manner that is essentially the same as when caches first appeared three decades ago.

Recent studies [6][7][8][11] have shown some ways to configure and manage the first-level (L1) cache. The average cache miss ratio and the average data access time of a program can be improved by incorporating an additional data store within the L1 cache and intelligently managing the state of this *multi-lateral*[1] cache through the use of reuse pattern information. In this paper, we present two ways to more efficiently manage such L1 data cache, one based on the effective address of the data referenced (NTS [7] and MAT [8]) and another based on the program counter of the load instruction generating the reference (CNA [6]). These two approaches are compared to one another and to a third approach (Victim [9]) which does not utilize previous reuse information. We show that for multi-lateral caches with direct-mapped (DM) main caches, the Victim approach performs best, with the NTS approach performing better as the L2 latency decreases or the associativity of the main cache increases.

The rest of this paper is organized as follows. Section 2 discusses earlier approaches to cache management via reuse information. Section 3 discusses the concept of block tours and presents our simulation methodology, metrics, cache configurations, and experiments. In Section 4 we present and analyze the results of the experiments. Section 5 concludes this work with some interesting observations.

3. Background

There are many techniques for reducing or tolerating the average memory access time. Prominent among these are: 1) store buffers, used to delay writes until bus idle cycles in order to reduce bus contention; 2) non-blocking caches, which overlap multiple load misses while fulfilling other requests that hit in the cache [1][2]; and 3) prefetching methodologies, both hardware [3][5] and software [4], that attempt to preload data from memory to the cache before it is needed. While all these schemes contribute to reducing the average data access time, we approach the data access time problem from the premise that L1 data caches are often

[1] The term *multi-lateral* to refers to a level of cache that contains two or more data stores that have disjoint contents and operate in parallel.

poorly and inefficiently managed, and that the average data access time can be reduced by intelligently managing the state of the L1 cache by using reuse pattern information. This approach, in addition to offering improved performance, can also free up die area on the processor chip that can be devoted to other processor optimization issues. Recently, at least 3 different approaches to more efficiently manage the L1 data cache have emerged in the literature.

3.1 The CNA Model

The CNA cache, evolving out of Tyson et al. [6], is a scheme that decides on data placement based on the program counter value of the memory instruction causing the current miss. A data block is marked as being either cacheable or non-cacheable based on the reuse characteristics of some block previously accessed by this same instruction. Cache blocks with multiple references are marked as cacheable, while those blocks which are only referenced once (the initial miss) prior to eviction are marked as non-cacheable in later miss handling. Blocks deemed non-cacheable and not already in the L1 cache are provided a data path to bypass the cache entirely. Though CNA did achieve a significant reduction in bus traffic, there was also a drop in the cache hit ratio relative to a conventional cache of equal size.

3.2 The NTS Model

The NTS cache [7], proposed by Rivers and Davidson, is a location-sensitive cache management scheme that dynamically partitions cache blocks into two groups, temporal (T) and nontemporal (NT), based on their reuse behavior during a past *tour*[2]. A block is considered NT if during a tour in L1, no word in that block is reused. Data blocks classified as NT are subsequently handled through a separate small cache placed in parallel with the main L1 cache. This scheme decides on data placement using reuse information associated with the effective address of the requested block. NTS' effectiveness in reducing the miss ratio, memory traffic and the average access penalty was demonstrated with mostly numeric programs.

3.3 The MAT Model

The MAT cache [8], proposed by Johnson and Hwu, is another scheme based on the use of effective addresses; however, it dynamically partitions cache data blocks into two groups based on their frequency of reuse. Cache blocks become tagged as either *Frequently Accessed* or *Infrequently Accessed*. The granularity for grouping is a *macroblock,* defined as a contiguous group of memory blocks considered to have the same usage pattern characteristics. Blocks that are determined to be Infrequently Accessed are handled through a separate small cache. This scheme was shown to give significant speedups over generic caches due to improved miss ratios, reduced bus traffic, and a resulting reduction in the average data access latency.

[2] A block tour refers to any of the time intervals that a block spends in the cache (between an allocation and eviction). A given memory block can have many tours through the cache.

3.4 The Victim Model

A fourth allocation mechanism will also be examined to identify the capability of management without reuse information. The Victim cache [9], proposed by Jouppi, places an additional small, fully-associative cache between the L1 cache and the next level in the hierarchy. Data that is "in cache" must be in the main cache, and data recently evicted from the main cache ("victim" blocks) can be quickly retrieved (swapped in) from the smaller cache on demand. Though this scheme retains recent victims independently of their reuse behavior, it was shown to be capable of reducing miss rates for direct-mapped main cache configurations by effectively increasing the associativity of conflicting cache blocks through the addition of the fully-associative buffer.

4. Simulation Methodology

In order to compare the effectiveness of the different cache management strategies, a simulator and a set of benchmark programs were used. This section describes the dynamic superscalar processor simulator used to evaluate these cache memory structures, the system configuration used, and the environment under which the simulations were performed. In addition, the simulation methods, metrics, and benchmarks used to carry out this study are presented

4.1 Processor and Memory Subsystem

The processor modeled in this study is a modification of one of the simulators in the SimpleScalar [10] toolset. The simulator performs out-of-order (OOO) issue, execution and completion on a derivative of the MIPS instruction set architecture. A schematic diagram of the targeted processor and memory subsystem is shown in Figure 1 with a summary of the chosen parameters and architectural assumptions.

The memory subsystem, modeled by the *mlcache* [13] tool, consists of a separate instruction and data cache, and a perfect secondary data cache or main memory. The instruction cache is perfect and responds in a single cycle. The data cache is modeled as a conventional data cache split into two subcaches (A and B) and placed in parallel within L1. In this *multi-lateral* cache, each subcache is unique with its own configuration: size, set-associativity, replacement policy, etc. The A and B caches are probed in parallel, and are equidistant from the CPU. Both A and B are non-blocking with 32-byte lines and single cycle access times. A standard data cache model would simply configure cache A to the desired parameters and set the B cache size to zero.

Finally, L2 cache is also modeled with an access latency of either four or eighteen cycles depending on the experiment. A 256 bit bus between L1 and L2 allows 32 bytes/cycle data bandwidth. Access from L1 to L2 is fully pipelined, and a miss request can be sent on the L1-L2 bus every cycle for up to 100 pending requests. The L2 cache is modeled as a perfect cache due to the desire to focus this study on the effectiveness of the L1 management strategies.

Fetch Mechanism	fetches up to 16 instructions in program order per cycle
Branch Predictor	perfect branch prediction
Issue Mechanism	out-of-order issue of up to 16 operations per cycle, 256 entry instr. re-order buffer (RUU), 128 entry load/store queue (LSQ); loads may execute when all prior store addresses are known
Functional Units	16 INT ALUs, 16 FP ALUs, 8 INT MULT/DIV, 8 FP MULT/DIV, 8 L/S units
F. U. Latency (total/issue)	INT ALU:1/1, INT MULT:3/1, INT DIV:12/12, FP ALU:2/1, FP MULT:4/1, FP DIV:12/12, L/S:1/1
Instruction Cache	perfect cache, 1 cycle latency
Data Cache	Multi-Lateral L1 (A and B), write-back, write-allocate, 32 byte lines, 1 cycle hit latency, 4 or 18 cycles miss latency, non-blocking, 8 memory ports

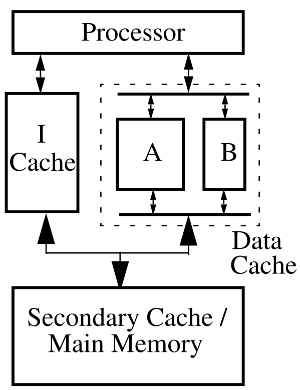

Figure 1. Processor and memory subsystem characteristics.

4.2 The Relative Cache Effects Ratio

An important metric for evaluating any cache management scheme is the cache hit/miss ratio. However, in OOO processors with multi-ported non-blocking caches, effective memory latencies vary according to the number of outstanding miss requests. To evaluate the effectiveness of the L1 cache structure using special management techniques, the Relative Cache Effects Ratio (RCR) seems appropriate. The RCR for a given processor running cache configuration *X* relative to cache configuration *base*, is given by:

$$RCR_X = \frac{Cycle\ Count_X - Cycle\ Count_{Perfect\ Cache}}{Cycle\ Count_{base} - Cycle\ Count_{Perfect\ Cache}}$$

where Cycle $\text{Count}_{\text{Perfect Cache}}$ refers to the total number of cycles needed to execute the same program on the same processor with a perfect cache configuration. The RCR, a normalized metric between the *base* cache and the perfect cache, is 1 for the *base* cache configuration and 0 for the perfect cache configuration. Cache configurations that perform better than the base have RCR between 0 and 1, with lower RCR being better. A cache configuration that performs worse than the *base* has RCR > 1.

4.3 The Block Tour and Reuse Concept

The effectiveness of a cache management scheme can also be measured by its ability to minimize the cumulative number of block tours during a program run. Individual cache block tours are monitored and classified based on the reuse patterns they exhibit. A tour that sees a reuse of any word of the block is considered dynamic temporal; a tour that sees no word reuse is dynamic nontemporal. Both dynamic temporal and dynamic nontemporal tours can be further classified as either spatial, when more than one word was used, or nonspatial, when no more than a single word was used. This allows for the classification of all tours into four groups: 1) nontemporal nonspatial (NTNS), 2) nontemporal spatial (NTS), 3) temporal nonspatial (TNS), and 4) temporal spatial (TS). Good management schemes should result in fewer tours and a higher percentage of data references to TS tours; NTNS and NTS tours are problematic, and a majority of references to such tours have a high likelihood of causing excessive cache-to-memory traffic and cache pollution. To minimize the impact of bad tours, a good multi-lateral cache management scheme should utilize an accurate block behavior prediction mechanism for deciding on data placement.

4.4 Benchmarks

We selected eight programs (5 integer and 3 floating point) from the SPEC95 benchmark suite. These programs have varying memory requirements, and the simulations were done using the training data sets. Each program was run to completion (with the exception of **perl**, which for reasons of limited time was terminated after the first 1.5 billion instructions).

4.5 Simulated Cache Memory Models

Performing a realistic comparison among the program counter (PC) and effective address (EA) schemes requires detailed memory simulators. To ensure a fair comparison and evaluation, we placed the management schemes on the same platform. For example, CNA as originally defined consisted of a single cache while the other two schemes used a main cache and an auxiliary buffer. We added a buffer to the CNA model to make the comparisons fair.

Our implementations of the NTS, MAT and CNA cache management schemes are discussed below. We refer to the main cache as cache A, and the auxiliary buffer as cache B, and both caches are placed equidistant from the CPU.

4.5.1 The NTS Model

The NTS cache structure (see Figure 2i) consists of a large A cache, a small B cache, and a detection structure (DS) for keeping track of reuse information and for guiding data placement in the L1 structure. The DS is distributed in three structures. The detection unit (DU), which serves as a clearinghouse for block placement in the L1 structure, is decoupled from both cache A and B and contains 32 entries. Each entry has 33 bits - the 32-bit effective address of a

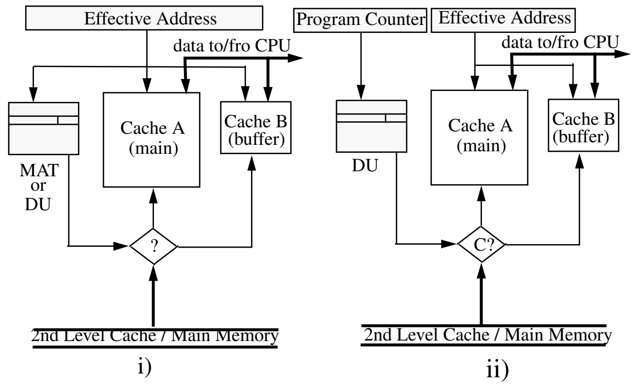

Figure 2. Schematic diagrams for the operations of i) the NTS and MAT models and ii) the CNA model.

block and an extra bit denoting the block's temporality. Caches A and B each have an attached temporality detector (TD). A TD is a hardware bit-map structure laid out as a matrix of bits with breadth equal to the number of words in a cache block plus one, and depth equal to the number of blocks in the cache. The most significant bit of a TD entry serves as the block's temporality decider. Whenever a block is allocated in the L1 structure, its TD bits are reset and the bit corresponding to the accessed word is set.

Cache A holds temporal (T) data and cache B holds nontemporal (NT) data; each access probes them in parallel. For accesses that hit in either cache, data is returned to the processor in one clock cycle. Simultaneously, the TD entry of the corresponding block is monitored and the bit corresponding to the referenced word is set. If this bit was set previously, signifying a word reuse, then the most significant bit of the TD entry will also be set, indicating that this is a temporal block. Accesses that miss in the L1 structure must reference the next level of memory, and their effective addresses are used to index into the DU to determine where the new block should be placed in the L1 structure. A block for which no information is found in the DU is assumed to be temporal and placed in the A cache by default.

Whenever a block is evicted from the L1 structure, its reuse status, based on the most significant bit of its TD entry, is cached in the DU together with its EA. DU entries are managed in an LRU fashion, allowing blocks that have not been accessed for long periods of time to fall out of the DU. This permits a block to change its usage characteristics based on recent program execution trends, allowing for better placement within the cache structure on subsequent tours. No communication or block swaps are allowed between caches A and B during a tour. Hence, on a miss, if the block's effective address hits in the DU and is determined to be NT, the block is placed in the B cache and will remain in the B cache until evicted, even if it were to show temporal behavior on this tour.

4.5.2 The MAT Model

In this implementation (see Figure 2i), the MAT cache [8] structure also consists of a large A cache and a small B cache at the first level. In addition, there is a Memory Address Table (MAT) for keeping track of reuse information and for guiding data placement in the L1 structure. The MAT is a direct-mapped structure containing 1K entries, each entry consisting of a macroblock address and an n-bit counter. An 8-bit counter and a 1K macroblock size was used for this study, as in the original study.

On a memory access, caches A and B are checked in parallel for the requested data. At the same time, the counter in the corresponding MAT entry for the accessed block is incremented. The counter in MAT entry serves as an indicator of the "usefulness" of a given macroblock, and helps to decide whether a block in that macroblock should be placed in the A or B cache.

On a cache miss, the macroblock address of the incoming block is used as an index into the MAT. If an entry exists, its counter value is compared against the counter of the block that would be evicted if the incoming block were to be placed in the A cache. If the counter value of the incoming block is higher than that of the current resident block, the incoming block replaces the current block in the A cache. This indicates that the incoming block has shown more "usefulness" in earlier tours than the current resident block, and should thus be given higher priority in the larger main cache. If the counter value of the incoming block is less than that of the current resident block, the incoming block is placed in the smaller B cache.

Finally, if no entry exists for the incoming block, the block is placed in the A cache and a new entry is created for it in the MAT, with its counter initialized to zero. If no entry is found for the conflicting block currently in cache A, its counter value is assumed to be 0, permitting the new block to easily replace it. When an entry is not found in the MAT for a resident block in cache A, it means that another macroblock that maps to the same set in the MAT has been accessed more recently, and the current block is therefore less likely to be used in the near future.

As with the NTS cache, communication between the A and B caches is disallowed. Unlike the NTS cache however, the MAT structure is updated for every access to the cache instead of only on evictions. The MAT is typically a direct-mapped structure, as opposed to the 32-way DU used by the NTS cache in this study.

4.5.3 The CNA Model

The CNA cache structure modeled (see Figure 2ii) is as similar as possible to the NTS cache. However, the criteria for data placement differ. The CNA cache determines block placement based on the past performance of the instruction (PC) requesting the data, instead of the effective address of the block. The CNA cache uses a detection structure (DS) similar to the NTS cache. The detection unit (DU) for the CNA cache is the same as that for the NTS cache, except that it is indexed by the instruction's PC. Both cache A and cache B also carry temporal detectors (as is done in the NTS cache). In addition to the bit-map structure, each TD entry also contains the PC that brought the block into the cache.

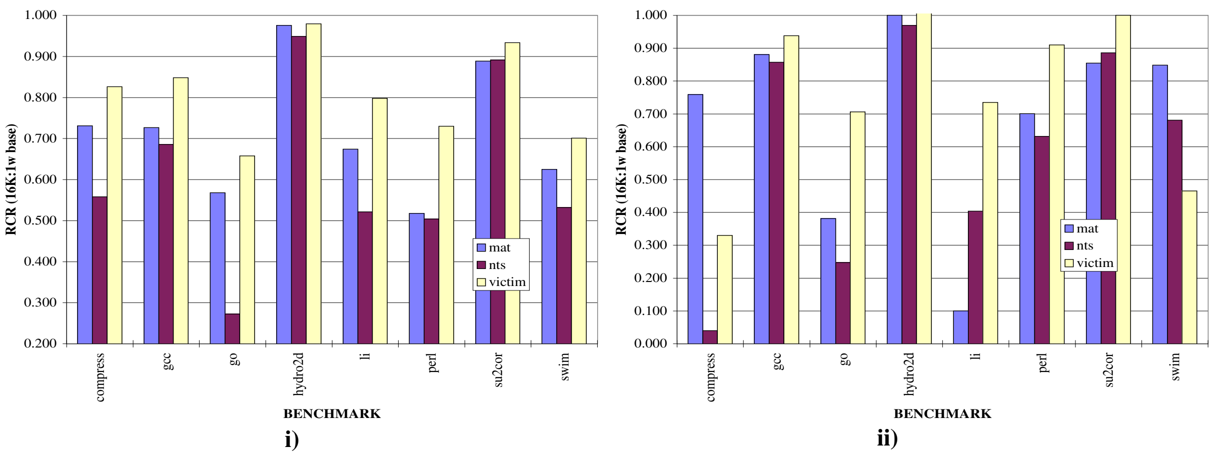

Figure 3. Comparative RCR data for the MAT, NTS and Victim cache schemes. i) is for a 16 instruction in-order issue processor while ii) is the same processor doing OOO issue. For both graphs, the A cache is 16KB direct-mapped and the B cache is 2KB FA. Block size is 32 bytes. L2 cache 4 cycles from the processor.

The DU is updated in a similar manner to the NTS scheme. When a block is replaced, the temporality bit of its entry (accessed by the PC of the instruction that loaded it) is set to the value of the block temporality tag in the TD. Thus, if that instruction subsequently misses, the loaded block will be placed in the B cache if the instruction's PC hits in the DU and the temporality bit indicates NT; otherwise, the block is placed in the A cache. If the instruction misses in the DU a new entry is created for this instruction in the DU.

4.5.4 The Victim Model

The Victim cache structure modeled is similar to the original design. There is a large A cache and a smaller, fully-associative B cache. All data from the next level of memory is placed in the A cache. On a hit in cache A, the requested data is returned to the processor in the next cycle. On a miss to cache A, the new block is fetched and the evicted block is placed in the B cache, possibly replacing one of its blocks. On a B cache hit, the requested block is placed in the A cache and the block evicted due to this migration is placed in the B cache -- this process is called a *swap*. Data evicted from the B cache returns to the next level of memory.

In our studies, we evaluate the effect of incurring various latencies for swap operations. For the traditional Victim cache implementation, we assume an extra cycle latency for a swap operation. For a multi-lateral Victim cache, we assume no latency for a swap operation, as a multi-lateral design will likely optimize the time to access data such that accesses to data resident in either cache are returned in the next cycle.

5. Experimental Results

In this section, the results of the comparative evaluation between the two effective address reuse information schemes is presented. We use the better of the effective address schemes to compare against the program counter scheme, and attempt to explain the disparities that exist between using program counter vs. effective address reuse information.

5.1 Managing with Effective Address Reuse Information

Figure 3 shows the RCR of each benchmark for the MAT, NTS, and Victim models. For these graphs, a 16KB direct-mapped cache A and a 2KB fully-associative (FA) cache B were used for each scheme. A 16KB direct-mapped cache provides the base for comparison. These experiments assume a perfect L2 cache only 4 cycles away from the processor. (This configuration was chosen to more closely reflect the configuration used in [8]). The NTS scheme uses a 32-entry DU in addition to the necessary temporality detector bits as explained earlier. The MAT scheme uses a 1K entry memory address table with a macroblock size of 1KB and a uniform block size among both cache A and B. (Note: Our experiments show that, in general, the 1K-entry DM MAT structure outperforms a 32-entry FA MAT structure). The Victim scheme used here incurs an additional cycle latency for a hit to the victim buffer.

Figure 3i shows comparative RCR data for an evaluation using a 16-wide issue, in-order processor while Figure 3ii provides similar data for the same 16-wide processor with out-of-order issue. As one would expect, these management schemes are more effective in in-order issue environments than in OOO environments. Cache misses are more likely to cause processor stalls in in-order environments, while in more latency-tolerant OOO processors, the latencies resulting from cache misses are likely to be overlapped with useful work and therefore do not appear too prominent.

Comparing the effective address schemes reveals that NTS outperforms MAT among all the programs, except for **su2cor** in the in-order processor configuration and **li** and **su2cor** in the OOO processor configuration. This performance gap can be explained by how these two schemes approach L1 data placement. On a cache miss, both

NTS and MAT attempt to predict where to place the missing data by indexing into the detection unit and the memory address table, respectively. While the NTS scheme's prediction is based solely on the missing data's temporal reuse in the previous tour, the MAT uses the reference frequency of the previous tour and that of the macroblock in its current tour. A block with a short-lived high reference frequency (e.g. nontemporal spatial data) may then be fixed in the A cache, which can lead to a gradual degradation in performance.

Another performance issue concerns the granularity for classifying tours. Whereas the NTS scheme tracks the reuse behavior of individual blocks, the MAT uses the larger macroblock granularity (1KB, in this study). The reuse frequency patterns of any single 32 byte block in a macroblock can be used to generalize the frequency metric of the larger 1KB chunk of data. The consequences may be poor placement decisions that result in many unnecessary block tours. Table 1 presents a tour analysis for **go** and **su2cor** for the in-order processor experiment. The performance of NTS relative to MAT is well demonstrated in **go**. Not only does NTS appear to manage the tours in this program better, but it actually reduces the number of tours MAT experiences by 32%. In the case of **su2cor**, where MAT performs only slightly better than NTS, it is clear that the application itself has much more nontemporal spatial data (> 14.5%), and neither scheme cuts the tours seen by a 16KB direct-mapped cache by more than 10%.

Though the graphs shown in Figure 3 demonstrate that the Victim model is worse than both effective address schemes, except for **compress** and **swim** in the OOO case, it must be noted that we assumed a perfect L2 cache which is only 4 cycles away. This performance might be expected given the L2 cache's low access latency and the victim scheme taking extra cycle penalties for swapping. If future L2 caches are going to be fast and sufficiently large, with the possibility of adequately capturing most of the application's working set, the effective address management schemes can provide better performance than the Victim cache. To determine the effects of using slower, larger L2 caches, we change our L2 latency to 18 cycles for the rest of our experiments. We also use a multi-lateral Victim cache with no latency for swaps, and consider only OOO processor environments beyond this point.

5.2 Managing with Program Counter vs. Effective Address Reuse Information

In this section we look at how the program counter approach compares with the effective address approach. We compare the NTS scheme with a comparable CNA scheme. This provides a fair platform for evaluations since the NTS and the CNA schemes are very compatible, differing only in their DU indexing mechanisms.

Figure 4 presents comparative data for the CNA scheme, the NTS scheme and the Victim scheme, relative to a 16KB direct-mapped cache. In these experiments, cache A is 16KB direct-mapped and cache B is 2KB fully-associative

Caching Scheme	Total Tours	Change in Tours	% Refs to Tour Groups			
			NTNS	NTS	TNS	TS
16K:1w	7,183,166	-	1.40	0.90	11.32	86.38
NTS:1wFA	1,879,584	-73.83%	0.19	0.45	4.41	94.94
MAT:1wFA	2,785,270	-61.23%	0.41	0.52	5.09	93.98

Caching Scheme	Total Tours	Change in Tours	% Refs to Tour Groups			
			NTNS	NTS	TNS	TS
16K:1w	25,799,451	-	0.50	14.98	7.73	76.79
NTS:1wFA	23,321,331	-9.61%	0.14	14.64	7.11	78.11
MAT:1wFA	23,244,175	-9.90%	0.16	14.73	5.17	79.94

Table 1: Tour Analysis for Go (top) Su2cor (bottom), 16-issue in-order processor model

for the 3 schemes. Also, the L2 cache is 18 cycles away from the processor. With the L2 latency changed from 4 to 18, the Victim cache approach outperforms both the NTS and CNA schemes, except for **compress** where NTS is still the best. The Victim scheme's performance can be attributed to the victim buffer's ability to dynamically add associativity to the hot sets in cache A.

Figure 4 also shows that the NTS scheme performs better than the CNA scheme, except for **su2cor** and **swim**. A given block in the cache can be used by many different instructions, each with varying usage characteristics. Consequently, a block in the CNA scheme may be suboptimally placed in the cache due to its correlation to blocks previously referenced by the requesting PC, leading to more unnecessary tours. For instance, a block may be brought into the cache at the beginning of a large routine. Depending on the flow of program execution, the block may be reused many times by many different instructions or not reused at all. All of these characteristics will be attributed to a single entry in the DU, tied to the PC of the instruction that caused the block to be loaded. In fact, a particular instruction may even cause multiple tours of a single block, and these tours could result in different temporal and nontemporal groupings. When each of these tours end, the instruction's entry in the DU is updated with that particular tour's behavior. This directly affects the placement of the next tour requested by this instruction.

However, program counter management schemes may be good if a given instruction always loads data whose usage is *strongly biased* [12] in one direction, i.e. if these tours are almost all temporal or almost all nontemporal. In this case, accurate behavior predictions for future tours will result in good block placement for that instruction. However, if the data blocks loaded by the instruction have differing usage characteristics (i.e. are *weakly biased* [12]) then placement decisions of its blocks will be poor.

5.2.1 Updating the Detection Unit: on every access vs. only on eviction

We studied the effect on updating history information with

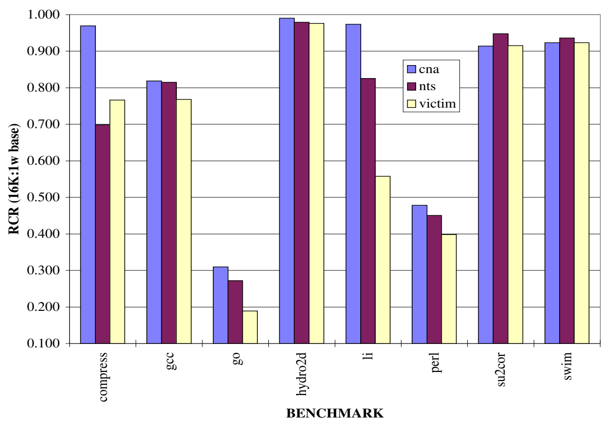

Figure 4. Comparative RCR for CNA, NTS and Victim. The A cache is 16KB DM and the B cache is 2KB fully associative. The perfect L2 cache is 18 cycles from the processor.

an experiment where instead of waiting until a tour's end to update the detection unit information, we do an update whenever a block is accessed. In Figure 5, we present the comparative RCR performance data for update on every access vs. update on eviction for both CNA and NTS. On the average, each of these schemes show better performance when they update only on block eviction.

The on-every-access update strategy can adversely impact performance. With many concurrent accesses to different blocks (in the NTS scheme) or many different memory accessing instructions (in the CNA scheme) occurring and creating new DU entries within a short time period, great space contention and frequent replacements can occur within the DU. As a result, useful block or instruction usage reuse characteristics may be lost prematurely, significantly impacting the placement of future blocks. With the update on eviction strategy however, the DU is only updated when a block is evicted from the cache structure; new DU entries will be created less frequently and tour history information will remain longer in the DU. While increasing the size of the DU may improve the performance of the update on-every-access scheme, for a fixed, small size DU, the on-eviction policy performs consistently better, except for CNA on **compress** and NTS on **su2cor**.

5.2.2 History Counter

We can keep the reuse information for an entry in the DU using saturating counters. For this experiment, we chose to evaluate 1- and 2-bit saturating counters. When an entry in the DU is updated to reflect that the entry has exhibited temporal behavior, the counter is incremented. Similarly, the counter is decremented when the entry has exhibited nontemporal behavior. In this case, for a block entering the cache that has an entry in the DU, the most significant bit indicates which cache it should be placed in. If this leading bit is 1, the block is placed in the A cache and if it is 0, in the B cache.

Increasing the number of bits of information stored provides

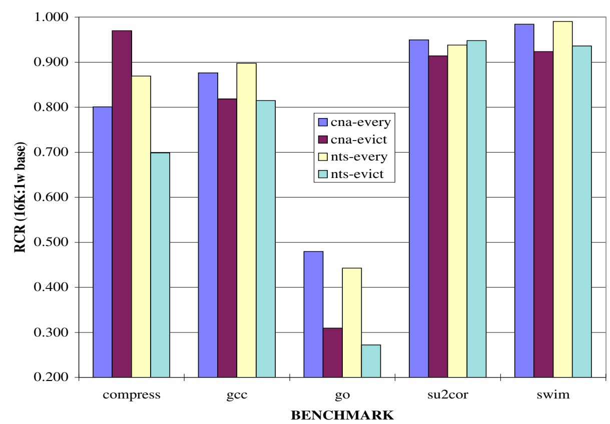

Figure 5. Effectiveness of CNA and NTS with DU update on every access vs. only on eviction. The A cache is 16KB DM and the B cache is 2KB fully associative. The perfect L2 cache is 18 cycles from the processor.

a more cumulative multi-tour history for a given entry. We find from our experiments that varying the number of history bits we keep does not meaningfully alter performance for the configurations that we have evaluated. A single-bit counter delivers similar performance to that of a two-bit counter. Keeping long history therefore does not appear beneficial for data placement prediction in the cache structure in these schemes. Keeping less history can be beneficial because the usage characteristics in a particular DU entry are more responsive to changes in behavior; keeping more history allows the behavior characterization to persist past exceptional tours. On average, these two effects seem to balance out.

5.3 The NTS scheme vs. the Victim scheme

We saw in Figure 4 that the multi-lateral Victim cache, with a direct-mapped A cache, generally performs better than the corresponding NTS structure. Current and proposed processors are including more associativity in the primary cache. Therefore, we extend our original simulations to examine the effect of increasing associativity in the A cache.

Figure 6 presents RCR data comparing the NTS scheme against the Victim scheme. The base for comparison is a 16K direct-mapped cache. The first three bars per application refer to the NTS scheme while the following three are for the Victim scheme. The three bars per scheme refer to cache A associativity of 1-way, 2-way, and 4-way. Cache B remains fully associative for all cases. As associativity increases beyond one for cache A, the NTS approach generally performs better than corresponding Victim cache schemes. The reason for this trend is that with increasing associativity in cache A, conflicts due to limited associativity are reduced, limiting the effectiveness of the Victim management approach. Thus, while only minimal benefit can be obtained by further targeting a reduction in conflicts due to limited associativity, a multi-lateral design in which more detailed reuse information is available can further improve performance.

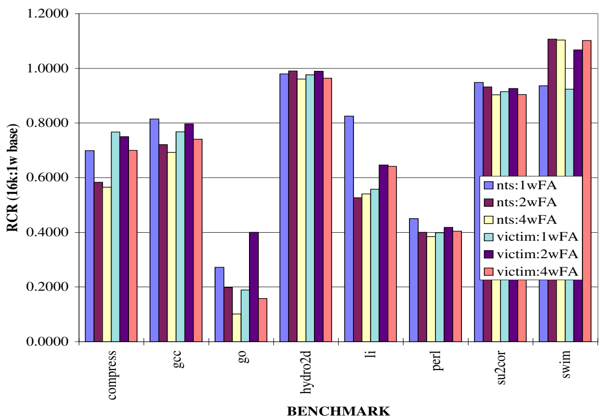

Figure 6. Comparing the NTS and Victim schemes over varying associativity. The A cache is 16KB and the B cache is 2KB. The perfect L2 cache is 18 cycles from the processor.

While we have presented results for a fully-associative B cache throughout this paper, we have performed a sample of these same experiments with an 8-way associative B cache and found that performance drops minimally relative to the fully-associative B cache. Thus, these multi-lateral caches can be built with k-way A cache and 8-way B caches, resulting in a reasonable 8+k compares per cycle at minimal performance degradation.

6. Conclusions

Two different ways to use reuse history to manage an on-chip data cache, one based on the effective address of the data reference (MAT and NTS) and another which manages the cache based on the program counter of the instruction generating a miss (CNA), were evaluated. These approaches were compared to each other and to an approach which does not exploit any reuse information (Victim).

For a direct-mapped main cache with a low (4 cycle) latency L2 cache, an intelligent cache management scheme performs better than a traditional victim cache. This is mainly due to the ratio of the delay to access the victim cache vs. the L2 access latency. When we move to a high (18 cycle) latency L2 cache, the Victim approach performs better than the other schemes because of its ability to effectively distribute associativity as necessary. However, as associativity grows in the main cache, conflicts due to limited associativity are reduced, limiting the effectiveness of the Victim management approach. For these cache structures, a good general management with the NTS (effective address-based) approach provides better performance by targeting placement using reuse information.

While the CNA (PC-based) approach also performs well, the NTS approach is generally better, and updating the detection unit (DU) on block evictions generally performs better than a scheme where updates occur for each access. Additionally, keeping a single bit counter for both the CNA and the NTS schemes provide comparable performance to the same schemes maintaining more history (e.g. a 2-bit saturating counter).

Effective address-based cache management with updating of the DU on eviction gives the best performance across all the cache structures evaluated with low L2 latency. The Victim approach performs best for multi-lateral caches with a direct-mapped main cache and a high L2 latency, while the NTS approach performs better as the L2 latency decreases or the associativity of the main cache increases.

7. References

[1] G. S. Sohi and M. Franklin, “High-Bandwidth Data Memory Systems for Superscalar Processors,” *Proceedings of ASPLOS-IV*, pp. 53-62, April 1991.

[2] D. Kroft, “Lockup-Free Instruction Fetch/Prefetch Cache Organization,” *Proceedings of ISCA-8*, May 1981.

[3] J.-L. Baer and T.-F. Chen, “An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty,” *Proceedings of Supercomputing’91*, Nov. 1991.

[4] D. Callahan, K. Kennedy, and A. Porterfield, “Software Prefetching,” *Proceedings of ASPLOS-IV*, pp. 40-52, April 1991.

[5] T.-F. Chen and J.-L. Baer, “Reducing Memory Latency via Non-Blocking and Prefetching Caches,” *Proceedings of ASPLOS-V*, pp. 51-61, April 1992.

[6] G. Tyson, M. Farrens, J. Matthews, and A.R. Pleszkun, “A Modified Approach to Data Cache Management,” *Proceedings of MICRO-28*, pp. 93-103, Dec. 1995.

[7] J.A. Rivers and E.S. Davidson, “Reducing Conflicts in Direct-Mapped Caches with a Temporality-Based Design,” *Proceedings of the 1996 ICPP*, pp. 154-163, August 1996.

[8] T. Johnson and W.W. Hwu, “Run-time Adaptive Cache Hierarchy Management via Reference Analysis,” *Proceedings of ISCA-24*, pp. 315-326, June 1997.

[9] N.P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,” *Proceedings of ISCA-17*, pp. 364-373, June 1990.

[10] D. Burger and T. M. Austin, “Evaluating Future Microprocessors: the SimpleScalar Tool Set,” Tech. Report #1342, University of Wisconsin, June 1997.

[11] A. Gonzalez, C. Aliagas, and M. Valero, “A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality,” *Proceedings of ICS'95*, July 1995.

[12] C-C. Lee, I-C. K. Chen, and T.N. Mudge, “The Bi-Mode Branch Predictor,” *Proceedings of MICRO-30*, pp. 4-13, Dec. 1997.

[13] E. S. Tam, J. A. Rivers, G. S. Tyson, and E. S. Davidson, “mlcache: A Flexible Multi-Lateral Cache Simulator,” *Proceedings of MASCOTS’98*, July 1998.