Evaluating the Performance of Active Cache Management Schemes

Edward S. Tam, Jude A. Rivers, Vijayalakshmi Srinivasan,
Gary S. Tyson, and Edward S. Davidson
Advanced Computer Architecture Laboratory
Electrical Engineering and Computer Science Department
The University of Michigan
Ann Arbor, MI 48109-2122
{estam,jrivers,sviji,tyson,davidson}@eecs.umich.edu

Abstract

In this paper we examine the performance of two multi-lateral cache schemes; one makes block allocation decisions correlated to the reference behavior of regions of memory (NTS), the other correlated to the reference behavior of memory accessing instructions (PCS). To determine the efficacy of exploiting these reference correlation schemes to improve cache management, we compare the performance of these multi-lateral schemes to a multi-lateral configuration that uses a near-optimal (but non-implementable) replacement policy, pseudo-opt. In addition to miss ratio, three metrics are used to evaluate the performance of these schemes, relative to pseudo-opt: 1) prediction accuracy in determining reference locality, 2) actual usage accuracy, i.e. how likely a block in an implementable scheme and the near-optimal scheme exhibit the same reuse characteristic, and 3) tour length of a line in the cache. Results show that while the NTS and PCS schemes outperform traditional cache management schemes, they fall short of the pseudo-opt performance; this is due to their simple prediction strategies and because their active management addresses only block allocation and not replacement.

***Keywords**: multi-lateral cache, active management, performance evaluation*

1. 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 continue to widen. There are many approaches to minimizing the average data access time, but the most common solution is to incorporate multiple levels of cache memory on-chip, and allocate cache lines on a miss.

Recent studies [1][2][3][4] have indicated that there may be better ways to configure and manage the first-level (L1) cache. *Active cache management*, which consists of active block allocation and replacement, can improve the performance of a given size cache. Active management uses information, such as a block's reference behavior, to make block allocation and replacement decisions; this is opposed to the traditional, passive management used in caches, in which blocks are allocated in a cache structure via a simple modulo mapping and replaced according to static policies. More specifically, active block allocation exploits individual reference behavior and places those blocks in the cache so as to maximize their usefulness and reduce any harm that their allocation may impart to the cache state. Active block replacement identifies the most useful blocks in the cache and attempts to retain them in the cache in preference to less useful blocks.

In order to accommodate varying block usage within a given cache structure, several schemes [2][3][4] have been proposed that incorporate an additional data store within the L1 cache structure to manage the state of the resulting *multi-lateral*[1] [5] cache by exploiting reuse pattern information. These structures perform active block allocation, though they do not perform active replacement. Each scheme instead relegates block replacement decisions to a simple hardware replacement algorithm. It has been shown that multi-lateral designs perform as well as or better than larger, single data store caches while requiring less die area [5][6].

In this paper we evaluate the efficacy of exploiting reference correlation schemes to improve cache management by analyzing reuse information obtained from optimally and near-optimally managed cache structures. We also find that, while the proposed schemes outperform traditionally managed caches, their performance falls short of the performance of the near-optimal management. To compare the decisions made and the resulting performance of two

1. We use the term *multi-lateral* to refer to a level of cache that contains two or more data stores that have disjoint contents and operate in parallel. We refer to the larger of the data stores as the A cache, the smaller the B cache.

multi-lateral schemes to the near-optimal cache structure, we present three metrics: 1) prediction accuracy in determining reference locality, 2) actual usage accuracy, i.e. how likely a block in an implementable scheme and the near-optimal scheme exhibit the same reuse characteristic, and 3) tour length of a line in the cache.

This paper is organized into seven sections. Section 2 presents previous techniques proposed to reduce the average data access time. We discuss the concept of active cache management in Section 3. In Section 4 we present the two multi-lateral schemes we evaluate, NTS and PCS. Section 5 discusses the value of reuse information in data cache management and presents the experiments we performed. The results of those experiments are examined in Section 6. We conclude this study and discuss future work in Section 7.

2. Background

There are many techniques for reducing or tolerating the average data access time. Prominent among these are: 1) store buffers, used to reduce bus contention by delaying writes until idle bus cycles occur [8]; 2) non-blocking caches, which overlap multiple load misses while satisfying other requests that hit in the cache [9][10]; 3) prefetching methodologies, both hardware [11] and software [12][13], that attempt to preload data from memory to the cache before it is needed; and 4) victim caching [14], which improves the performance of direct mapped caches through the addition of a small, fully associative cache between the L1 cache and the next level in the hierarchy. While these schemes do contribute to reducing the average data access time, this paper approaches the problem from the premise that the average data access time can be reduced by exploiting reuse pattern information to actively manage the state of the L1 cache. This approach can be used with these other techniques to reduce the average data access time further.

3. Active Cache Management

Active cache management can be used to improve the performance of a given size cache structure by controlling the data placement and management in the cache to keep the active working set resident, even in the presence of transient references. Active management of caches consists of two parts: allocation of blocks within the cache structure on a demand miss[2] and replacement of blocks currently resident in the cache structure. Block allocation in today's caches is passive and straightforward: blocks that are demand fetched are placed into their corresponding set within the cache structure[3]. However, this decision does not take into consideration the block's usefulness or usage characteristics. Examining the past history of a given block is one method of aiding in the placement of the block during future allocations. Decisions can range from simply not caching the target block (bypassing) to placing the block in a particular portion of the cache structure, with the hope of making best use of the target cache block and the blocks that remain in the cache.

Simple FIFO, LRU, or random block replacement policies are typically used to choose a block for eviction in today's caches. For a multi-lateral cache, the decision may not be so straightforward: if the data stores are not all fully associative, the block chosen for eviction may be a direct consequence of the allocation of a demand-missed block. As a result, this choice is often suboptimal. By swapping blocks amongst the caches of a multi-lateral L1 cache structure, the replacement choice can be improved.

Recently, several approaches to more efficient management of the L1 data cache via block allocation decisions have emerged in the literature: C/NA [1], NTS [2], MAT [3], Dual/Selective [4], and Dynamic Exclusion [7]. However, none of these approaches makes sophisticated block replacement decisions, and instead relegates these decisions to their respective cache substructures.

4. Evaluated Multi-Lateral Schemes

Among the multi-lateral schemes performing block allocation decisions, the NTS and PCS (PC-Selective) schemes were found to perform the best [6]. We thus use NTS as the effective-address based block allocation scheme for this study. The PCS (PC-Selective) scheme is an improvement of the C/NA scheme; it makes block allocation decisions based on a memory accessing instruction's performance, as in the C/NA and Dual/Selective cache models. We review the structure and operation of the NTS and PCS schemes in the following sections.

4.1. Structure and Operation of the NTS Cache

The NTS cache, using the model in [6], which was adapted from the scheme proposed in [2], actively places data within L1 based on each block's usage characteristics. In particular, blocks that have been found to exhibit temporal reuse are placed in A, while nontemporal data are sent to B. This is done in the hope of allowing temporal data to remain in the larger A cache for longer periods of time, while shorter lifetime nontemporal data can for a short while be quickly accessed from the small, but fully associative B cache.

2. Allocation decisions for blocks not loaded on a demand miss, e.g. prefetched blocks in a streaming buffer scheme as proposed in [14], are not considered here. However, schemes that make proper allocation decisions for prefetched data can further improve upon the performance of the schemes evaluated herein.

3. We consider only write-allocate caches in this paper. Write no-allocate caches follow a subset of these rules, where writes are not subject to these allocation decisions.

On a memory access, if the desired data is found in either A or B, the data is returned to the processor with no added latency, but the block remains in the cache in which it is found. On a miss, the block entering L1 is checked to see if it has an entry in a Detection Unit (DU). The DU contains temporality information about blocks recently evicted from L1. On eviction, a block is checked to see if it exhibited temporal reuse during its most recent tour in the L1 cache structure and is marked accordingly in the DU. If the new block address matches an entry in the DU, the tag of that entry is checked and the block is placed in A if it indicates temporal, and B if not. The reuse information is kept via a single bit, the T/NT bit. If no DU match is found, a new entry is created in the DU and the block is assumed to be temporal and placed in the larger A cache.

4.2. Structure and Operation of the PCS cache

The PCS cache [6] decides on data placement based on the program counter value of the memory instruction causing the current miss, rather than on the effective address of the block as in the NTS cache. Thus, the performance of individual memory accessing instructions, rather than individual data blocks, is used to determine placement of data in the PCS scheme.

The PCS cache structure modeled is similar to the NTS cache. The DU is indexed by the memory accessing instruction's program counter, but is updated in a similar manner to the NTS scheme. When a block is replaced, the temporality bit of its entry is set according to the block's reuse characteristics during its last tour of the cache. Thus, if that instruction subsequently misses, the loaded block is placed in B if the instruction's PC hits in the DU and the prediction bit indicates NT; otherwise the block is placed in A. If the instruction misses in the DU, a new DU entry is created for this instruction and the data is placed in A.

5. Using Reuse Information in Active Data Cache Management

Each of the multi-lateral schemes operate on the assumption that reuse information is a useful criterion by which to actively manage the cache. In this section, we evaluate the value of reuse information for making placement decisions in multi-lateral L1 cache structures. We then outline the experiments performed to validate the use of reuse information and compare the performance of the improved multi-lateral schemes to the performance of near-optimally managed caches of the same size.

5.1. The Value of Reuse Information

To determine the efficacy of reuse information in data cache management, we evaluated the performance of an optimal multi-lateral cache. Given an arbitrary cache A and cache B, an optimal multi-lateral configuration would require both caches to be fully associative, where blocks are free to move between the two caches as necessary in order to retain the most useful blocks in the cache structure. Based on these parameters, the optimal multi-lateral cache would reduce to a single fully associative cache of size equal to cache A plus cache B which can be managed using Belady's MIN algorithm [18], an optimal replacement algorithm for a single fully associative cache.

However, the optimal configuration has two shortcomings. First, it is not implementable, as it requires complete knowledge of future references when making its replacement decisions. While we can evaluate the performance of the optimal configuration for a given trace, it is not applicable to general use. Thus, the optimal configuration can only provide an upper-bound on the performance of a multi-lateral cache structure of a given block size and total size and arbitrary associativity. Second, the optimal configuration is inherently not a multi-lateral cache; it is a hardware-partitioned fully associative L1 cache structure that does not utilize any special methodology for block placement. We would rather evaluate a configuration that works optimally on multi-lateral structures whose caches are not both fully-associative, so we can more fairly compare its performance to that of implementable schemes.

A configuration that conforms to the purposes of a multi-lateral design is one that is managed similar to Belady's MIN algorithm but contains A and B caches of different associativity. As in the optimal configuration, free movement of blocks between A and B is allowed, provided that no block simultaneously resides in both A and B. Management is adapted from Belady's MIN algorithm, as follows. On a miss, we fill an empty block in the corresponding set of cache A or cache B, if one exists. If not, then for each set of cache A define an extended set of blocks consisting of all blocks in cache A and cache B that map to this set in cache A. If the extended set includes any blocks that currently reside in cache B, i.e., the extended set is larger than the associativity of cache A, then find block γ of the extended set whose next reference is farthest in the future. If block γ does not currently reside in cache B, then swap it with one of the cache B blocks of this extended set. Now, among the blocks in cache B that map to the same cache B set as the new access, the block that is next referenced farthest in the future is replaced. This choice is not optimal in all cases.

For the example reference pattern in Figure 1, consider a design with a direct-mapped cache A of size 2 blocks (2 sets) and a one block cache B. The references shown in the figure are block addresses. Capital letters map to set 0 and small letters map to set 1 of cache A. The figure shows the contents of set 0 and set 1 in cache A and cache B after each memory access. Using our adapted MIN replacement policy we incur 7 misses, whereas the minimum possible number of misses is 6. For example, at time 4, if we

Ref:	b	c	F	D	b	E	D	F	c
Time	1	2	3	4	5	6	7	8	9
Set 0	-	-	F	D	D	E	D	D	D
Set 1	b	b	b	b	b	b	b	b	c
Cache B		c	c	F	F	D	E	F	F
Hit/Miss	M	M	M	M	H	M	H	M	M

Figure 1: An example showing why *pseudo-opt* is suboptimal.

replace block **F** (in set 0 of cache A) instead of block **c** (in cache B), we incur 6 misses instead of 7. In cases where the A and B caches are fully associative, the adapted MIN algorithm reduces to MIN. However, in all other cases, the management is sub-optimal; we refer to this management as *pseudo-opt* [19].

Although *pseudo-opt* is also not an implementable policy, its performance, as seen in Table 1, is close to that of *opt* and can be used as a standard against which we compare the performance of the implementable multi-lateral cache schemes. First, we show how optimal for the enlarged (combined A and B) cache (herein referred to as *opt*) and *pseudo-opt* for the A/B multi-lateral cache perform for a specified size L1 cache structure. These results show the effect of limiting the associativity of the A cache. We see in Table 1 that, based on miss ratio, *pseudo-opt* actually performs quite well compared to *opt*, despite the limited associativity of the configuration and its sub-optimal replacement policy. Since *pseudo-opt* performs close to *opt* and honors more restrictive associativity constraints, we compare the performance of the implementable schemes (NTS and PCS) to that of *pseudo-opt*. The differences in placement and replacement decisions seen in the implementable schemes provide insights into their performance relative to a near-optimal scheme.

5.2. The Block Tour and Reuse Concept

In addition to miss ratio, 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. A tour of a cache block is the time interval between an allocation of the block in L1 and its subsequent eviction. A given memory block can have many tours through the cache.

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 one word was used. This allows us to classify all tours into four data reuse 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 blocks making TS tours; NTNS and NTS tours can be problematic, as a majority of references to such tours have a high likelihood of causing excessive cache pollution (as they may displace TS tours). To minimize the impact of bad tours, a good multi-lateral cache management scheme should utilize an accurate block behavior prediction mechanism for data placement decisions.

5.3. Simulation Environment

Due to the nature of the *opt* and *pseudo-opt* schemes, we collected memory reference traces generated from the *SimpleScalar* processor environment [15]. Each trace entry contains the (effective) address accessed, the type of access (load or store), and the program counter of the instruction responsible for the access.

We skipped the first 100 million instructions (to avoid initialization effects) and, due to space and computation constraints, analyzed the subsequent 25 million memory references. Our experiments use 4 SPEC95 integer benchmarks -- **compress**, **gcc**, **go**, and **li,** since sampling such a small portion of a floating point program will likely generate references that form part of a regular loop, resulting in very low miss rates. The sampled traces for the integer programs, however, reasonably mirror the actual memory reference behavior of the complete program execution.

The traces were annotated to include the information necessary to perform the *opt* and *pseudo-opt* replacement decisions and counters for many useful statistics. These included the outcome of an access (hit or miss) as it would have occurred in an *opt*/*pseudo-opt* configuration, the usage information for each accessed block during a tour (as seen for each scheme), and the number of blocks in each reuse category that are in the cache at each instant in time (i.e. the number of NTNS, NTS, TNS, and TS blocks resident in the cache). From these statistics, we gathered information regarding the performance of the *opt* and *pseudo-opt* management schemes and compared the performance of the implementable schemes to that of the optimal schemes on an access-by-access basis.

To mirror the previously proposed implementable configurations, we chose a direct-mapped, 8KB A cache and a fully associative, 1KB B cache, both with a 32B blocksize, for the *pseudo-opt* configuration. (The *opt* configuration is simply a 9K fully associative cache.)

6. Results

6.1. Analysis of *opt* vs. *pseudo-opt*

We analyzed the annotated traces produced from the *opt* and *pseudo-opt* runs to determine their relative performance based on miss ratio measurement and block usage information. In particular, we counted the number of tours

	Compress	Gcc	Go	Li
opt	0.042	0.013	0.008	0.008
pseudo-opt	0.048	0.016	0.023	0.010
NTS	0.063	0.029	0.054	0.018
PCS	0.067	0.032	0.060	0.019
DM:16K	0.063	0.029	0.050	0.020

Table 1: Miss ratios for the four (8+1)KB and the 16KB cache configurations on the trace inputs.

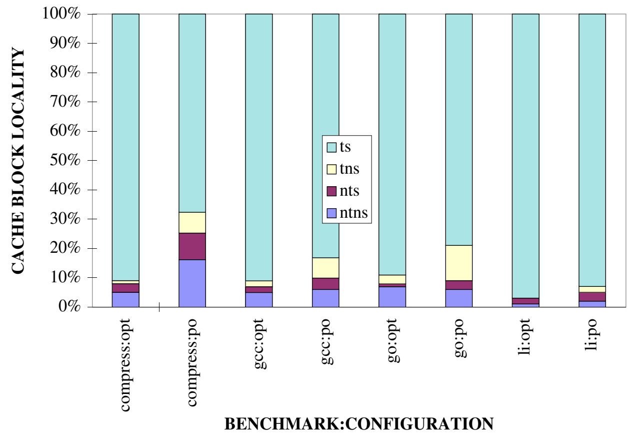

Figure 2: Dynamic cache block occupation in *opt* and *pseudo-opt (*denoted *po*), grouped into NTNS, NTS, TNS, and TS usage patterns.

that show each reuse pattern, the number of each type of block resident in the cache at any given instant, and how often a block with a prior reuse characteristic changes its usage pattern in a subsequent tour.

6.2. Miss Ratio

As the miss ratios in Table 1 show, the performance of *pseudo-opt* is relatively close to that of *opt*, except for **go**. In **go**, this disparity is due to the limitations on the associativity of the A cache and the suboptimal replacement policy of *pseudo-opt*. On a replacement, any number of swaps can be done in *opt* to rearrange the cache contents so that the least desired cache block is evicted. However, in *pseudo-opt*, the movement choices are limited by the mapping requirements of the direct-mapped A cache. As a result, there are times when a block is brought in and a sub-optimal block is evicted from L1, as the block maps to a set in A that contains data that will be used sooner than data in another set. The difference in performance between *pseudo-opt* and *opt* is, however, much smaller than the difference between the implementable configurations and *pseudo-opt*. (The performance of the implementable schemes is discussed in Section 6.5.) As a result, we use *pseudo-opt* as our basis for comparison to the implementable schemes to attribute the differences to allocation and replacement policies and not to decreased associativity.

6.3. Cache Block Locality Analysis

We examined the locality of cache blocks that are resident in the cache structure at any given time by counting the number of blocks in each category at the time of each miss and taking an average over the duration of the program. This breakdown of blocks for the *opt* and *pseudo-opt* configurations is shown in Figure 2.

For *opt* and *pseudo-opt* managed caches, our earlier observations hold true, i.e. data that exhibits temporal reuse occupies a large portion of the cache space. Furthermore, most of the blocks are both temporal *and* spatial. Though the blocks are relatively small in size (32 bytes), we find that spatial reuse can be exploited well if the cache is managed properly. As expected, the *pseudo-opt* configurations (vs. the *opt* configurations) hold fewer TS blocks in the cache, due to the *pseudo-opt* configuration's limited placement options and sub-optimal replacement decisions.

Nontemporal data occupies a relatively small portion of the cache (at most 23%, for **compress** under *pseudo-opt*), but is responsible for much of the pollution in the cache because of the interleaved nature of references to such data with other useful data. As we will see later in Table 3, the average tour length of NT tours is much shorter than that of T tours. As such, NT blocks are less resident, and thus less useful, than T blocks they may replace. Nonetheless, it is still necessary to cache such data for different portions of program execution. Thus, if we can manage the state of the cache properly by considering reuse information, we can improve the performance of a given cache structure by keeping a large number of TS blocks in the cache at any given time, so as to approach the optimal performance.

6.4. Block Usage Persistence

While the presence of certain types of blocks in the cache shows the potential benefit of managing the cache with reuse information, management based on this information is not straightforward. Blocks can have different usage characteristics during different portions of program execution, making block usage hard to predict. The prediction of a particular block's usage pattern is similar to the branch prediction problem. However, branch outcomes are easier to predict than optimal block usage characteristics in aiding placement decisions.

To assess the value of reuse information, we examined the *persistence* of cache block behavior in the *opt* and *pseudo-opt* schemes, i.e. once a block exhibits a given usage characteristic in a current tour, how likely is it to maintain that characteristic in its next tour. If there is a high correlation between past and future use (i.e. the block's usage characteristic is persistent), prediction of future usage behavior will be easier. Block *persistence* is therefore analogous to the *taken* or *not-taken* terminology

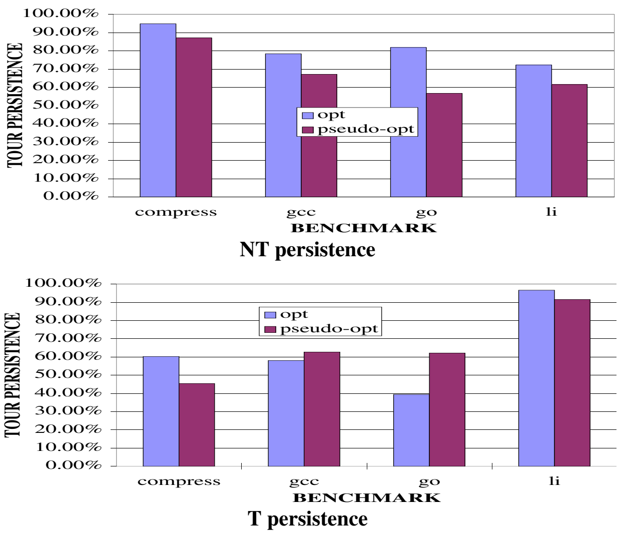

Figure 3: Block usage persistence within *opt* and *pseudo-opt*.

		Tour prediction accuracy		Actual usage accuracy	
		NTS	**PCS**	**NTS**	**PCS**
Compress	**NT**	0.483	0.703	0.859	0.828
	T	0.240	0.210	0.402	0.412
	Total	0.247	0.495	0.725	0.725
Gcc	**NT**	0.322	0.589	0.684	0.617
	T	0.539	0.596	0.703	0.704
	Total	0.527	0.595	0.696	0.675
Go	**NT**	0.337	0.454	0.573	0.519
	T	0.588	0.621	0.646	0.652
	Total	0.561	0.596	0.624	0.609
Li	**NT**	0.225	0.243	0.390	0.363
	T	0.815	0.815	0.873	0.866
	Total	0.786	0.723	0.787	0.758

Table 2: Tour prediction and actual usage accuracy for NTS and PCS, broken into NT, T, and overall (total) accuracy. Prediction accuracy is relative to actual block usage in *pseudo-opt*.

used in branch prediction studies [20] to predict the path a specific branch will take given behavior history. To make our placement decisions easier, we looked at the block usage at a higher level of granularity. Instead of determining block persistence in terms of the four usage patterns NTNS, NTS, TNS, and TS, we decided to examine only the persistence of T and NT patterns. This larger grouping is better suited for comparison to the proposed 2-unit cache structures.

Figure 3 presents data for block tour persistence. In general, blocks that exhibit NT usage behavior in prior tours have a strong likelihood of exhibiting NT behavior again in future tours. In the *opt* scheme, this likelihood ranges from 72% to 95% for the evaluated benchmarks. The *pseudo-opt* scheme shows more variability, however, with NT blocks persisting from 56% (**li**) to 87% (**compress**) of the time. Nevertheless, the likelihood of NT blocks persisting in both *opt* and *pseudo-opt* schemes is well over 50%. Blocks that exhibit T usage behavior are less persistent, and thus, less predictable for the future. For T blocks in the *opt* scheme, 40% (**go**) to 97% (**li**) of T blocks will exhibit temporal reuse in their next tour. The numbers are similar for the *pseudo-opt* scheme: 45% (**compress**) to 92% (**li**) of the temporal blocks will show the same usage during their next tour.

Examining the trace for **li**, we found that only ~1300 unique blocks were accessed throughout the trace. Given that there are 288 blocks in our cache configurations, a high proportion of those blocks is likely to be resident in the cache at any given time, resulting in high temporal reuse. Removing **li** from the temporal block numbers, we find that blocks that are marked T exhibit the same behavior 40% (**go**) to 60% (**compress**) of the time for the *opt* scheme and 45% (**compress**) to 63% (**gcc**) of the time in the *pseudo-opt* scheme. As **compress**, **gcc**, and **go** represent a wider range of program execution than **li** alone, we see that future tours of temporal blocks are harder to predict, i.e. blocks that show temporal reuse are *weakly biased* in exhibiting T or NT usage patterns in subsequent tours.

6.5. Multi-Lateral Scheme Performance

6.5.1. Miss Ratio Performance of NTS and PCS

NTS and PCS are as similar as possible, differing only in their data placement criteria. To compare with the *pseudo-opt* configuration used, we configured these structures to have an 8KB direct-mapped A cache, a 1KB fully associative, LRU-managed B cache, 32B block size, and a 32-entry detection unit (DU).

The miss ratio performance of the two configurations is shown in Table 1 along with the performance of *opt*, *pseudo-opt*, and a direct-mapped 16KB single structure cache. Concurring with results of earlier analyses of these multi-lateral configurations, the NTS and PCS caches both perform nearly as well as the direct-mapped cache, which is nearly twice the size.

6.5.2. Prediction Accuracy

NTS makes placement decisions based on a given block's usage during the most recent past tour – if the block exhibits temporal reuse in some tour, the block is placed in the larger A cache for its next tour, while blocks exhibiting nontemporal reuse are next placed in the B cache. PCS makes placement decisions based on the reuse of blocks loaded by the memory accessing instruction. If a block loaded by a particular PC exhibits temporal reuse, subsequent blocks loaded by that PC are predicted to do the same and are placed in the larger A cache. If a block loaded by a particular PC exhibits nontemporal reuse, subsequent blocks loaded by that PC are placed in the B cache. In both schemes, accesses without a corresponding entry in the DU are assumed to exhibit temporal reuse and are placed in the A cache.

The accuracy of these predictions can be determined based on the actual usage of the blocks in the *pseudo-opt* scheme. As noted in Section 5.3, the annotated traces fed to our trace-driven cache simulator, a modified version of *mlcache* [21], contain the actual block usage information for the tours seen in the *pseudo-opt* scheme. The simulator provides information on a scheme's block prediction and actual usage accuracy. Given this information, it is easy to determine how well each of the configurations predict a block's usage, and consequently, whether it is properly placed within the L1 cache structure.

Table 2 shows the prediction accuracy of NTS and PCS for the benchmarks examined. In general, the prediction accuracy is relatively low, ranging from 25% (**compress**) to 60% (**go**) (**li** is ignored due to its excessively high temporal reuse), despite the larger granularity of block typing (two categories T and NT versus the complete four category breakdown). Both the NTS and PCS schemes show poor prediction accuracies, directly impacting their block allocation decisions and their resulting overall performance. Thus, if the block usage prediction for these schemes is improved, we might better place blocks in the cache structure for higher performance.

6.5.3. Actual Usage Accuracy

Regardless of the prediction accuracy, a given block will exhibit reuse characteristics based on the duration and time of its tour through the cache. We examined the actual usage of each block as it was evicted from each of the caches in the implementable schemes to see how that usage compared to the same block's usage in the *pseudo-opt* scheme. This comparison sheds some light on the effect of eliminating the movement of blocks between the caches during an L1 tour.

We see in Table 2 that the actual usage of the blocks in the cache is closer to the actual usage of the blocks in *pseudo-opt*, despite the low prediction accuracy we saw in Section 6.5.2. Thus, despite our placement decisions, some blocks still exhibited reuse behavior akin to that seen in a *pseudo-opt* configuration.

While this is interesting, the block tour lengths in the implementable schemes are not directly related to the tour lengths in the *pseudo-opt* scheme, as *pseudo-opt* has a more elaborate replacement policy. The disparate tour lengths can affect these comparisons in two ways. First, for tours that are shorter in the *pseudo-opt* configuration than in the implementable schemes, the implementable schemes may keep a block around longer than necessary and permit it to exhibit seemingly more beneficial usage patterns (i.e. TNS or TS). While this makes a particular block seem more useful, the presence of that block in the implementable schemes may preclude the optimal placement of a subsequent block in the cache structure.

	Compress		**Gcc**		**Go**		**Li**	
	NT	**T**	**NT**	**T**	**NT**	**T**	**NT**	**T**
p-o	1293.3	26695	1731.2	35946	1177.8	22297.3	4400.4	34221.5
NTS	3.9	41.5	1.8	44.8	1.2	22.0	2.7	57.4
PCS	3.7	48.9	2.1	49.8	1.3	24.1	3.2	66.1

Table 3: Average tour lengths for *pseudo-opt (p-o)*, NTS, and PCS. Tour lengths are measured by the number of accesses handled while the block is in cache. Tours are broken into NT and T.

More likely, where tours are longer in the *pseudo-opt* configuration than in the implementable schemes, the blocks in the implementable schemes are evicted before they can exhibit their optimal usage characteristics, causing the correlation to be lower.

Table 3 shows the average tour length for blocks showing T or NT usage characteristics. As expected, tours in *pseudo-opt* are much longer than tours in either NTS or PCS, due to its greater flexibility in management of data once it has entered the L1 cache structure. In NTS and PCS, once a block is placed in L1, it remains in the same cache until it is evicted, and is thus subject to the replacement policy inherent in the target cache. For instance, if block α is deemed to be T in either NTS or PCS, it is placed in the direct-mapped A cache. If a subsequent block β maps to the same set in A and is also marked T, the earlier T block, α, will be evicted, likely before it can exhibit its optimum usage characteristics. In *pseudo-opt,* block α would simply be moved to the B cache, where it could remain until it is no longer useful. As a result, the tour length of block α will be much longer under *pseudo-opt* than under NTS or PCS.

We see that there is a clear gap between the usage of blocks in the *pseudo-opt* and in the implementable schemes. By improving upon the prediction of the usage characteristics of a block and the management of those blocks once they are placed in the L1 cache structure, we can improve the performance of the NTS and PCS schemes to reduce the performance gap that exists between these schemes and *pseudo-opt*.

7. Conclusions and Future Work

Previous studies have shown the performance benefit of using multi-lateral caches, such as NTS and PCS, over using traditional cache structures. In this work we have explored the potential for active management of cache resources. In this limits study, we show that multi-lateral structures can obtain even greater performance gains over those schemes proposed to date.

We introduced three metrics by which a multi-lateral scheme, which uses reuse information to guide block allocation, can be evaluated and compared to a near-optimal (*pseudo-opt* managed) cache structure: 1) prediction accuracy, indicating how well a given scheme predicts the usage of demand-missed blocks, 2) actual usage accuracy, indicating how similarly blocks in the proposed scheme are actually used compared to the block's usage in *pseudo-opt*, and 3) tour length, indicating the scheme's ability to maintain useful blocks (and similarly evict non-useful blocks) in the cache structure. Our results show that the performance difference between the implementable schemes and *pseudo-opt* is great, due largely to the simple prediction strategies used in these schemes along with their limited management of blocks once they are placed in the cache structure.

We feel that making block allocation decisions by exploiting reuse information is a viable way to improve the performance of a cache structure. In the future, we will explore new active cache management schemes that incorporate active block replacement and reduce misprediction rates. These schemes may also make use of different block allocation schemes. We also intend to evaluate how well increased associativity approaches (near-)optimal performance for a given-sized multi-lateral cache structure. The tractability of specific schemes will likely lead to hybrid schemes which can better adapt to program behavior than the individual schemes alone.

8. Acknowledgments

This research was supported by the National Science Foundation grant MIP 9734023 and by a gift from IBM. The simulation facility was provided through an Intel Technology for Education 2000 grant.

References

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

[2] 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.

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

[4] A. Gonzalez, C. Aliagas, and M. Valero, "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," *Proceedings of the 1995 ICS*, pp. 338-347, 1995.

[5] J. A. Rivers, E. S. Tam, and E. S. Davidson, "On Effective Data Supply for Multi-Issue Processors," *Proceedings of the 1997 ICCD*, October 1997.

[6] J. A. Rivers, E. S. Tam, G. S. Tyson, E. S. Davidson, and M. Farrens, "Utilizing Reuse Information in Data Cache Management," *Proceedings of the 1998 ICS*, July 1998.

[7] S. McFarling, "Cache Replacement with Dynamic Exclusion," Proceedings of ISCA-19, pp. 191-200, May, 1992.

[8] N. P. Jouppi, "Cache Write Policies and Performance," *Proceedings of ISCA-20*, pp. 191-201, May 1993.

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

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

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

[12] J.-L. Baer and T.-F. Chen, "An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty," *Proceedings of Supercomputing'91*, pp. 176-186, November 1991.

[13] 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.

[14] 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.

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

[16] G. Kurpanchek et al, "PA-7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface." *COMPCON Digest of Papers*, February 1994.

[17] J-L. Baer and W-H. Wang, "On the Inclusion Properties for Multi-Level Cache Hierarchies," *Proceedings of ISCA-15*, May 1988.

[18] L. A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Systems Journal, Vol. 5,* pp. 78-101, 1966.

[19] Vijayalakshmi Srinivasan and E. S. Davidson, "Improving Performance of an L1 Cache with an Associated Buffer," *Technical Report CSE-TR-361-98*, University of Michigan, March, 1998.

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

[21] E. S. Tam. J. A. Rivers, G. S. Tyson, and E. S Davidson, "mlcache: A Flexible Multi-Lateral Cache Simulator," *Proceedings of MASCOTS'98*, pp. 19-26, July, 1998.