Allocation By Conflict: A Simple, Effective Multilateral Cache Management Scheme

Edward S. Tam[†], Stevan A. Vlaovic, Gary S. Tyson, and Edward S. Davidson
[†]*Apple Computer, Inc.*
Cupertino, CA 95051
etam@apple.com

Advanced Computer Architecture Laboratory
The University of Michigan
Ann Arbor,MI 48109-2122
{vlaovic,tyson,davidson}@eecs.umich.edu

Abstract

Several schemes have been proposed that incorporate an auxiliary buffer to improve the performance of a given size cache. Victim caching, aims to reduce the impact of conflict misses in direct-mapped caches. Victim offers competitive performance benefits, but requires a costly data path for swaps and saves between the main cache and the added buffer.

Several multilateral schemes (e.g. NTS, PCS) offer competitive performance with Victim across a wide range of associativities, but require no swap/save data path. While these schemes perform well overall, their overall performance lags that of Victim when the main cache is direct-mapped. Furthermore, they also require costly hardware support, but in the form of history tables for maintaining allocation decision information.

This paper introduces a new multilateral cache management scheme, Allocation By Conflict (ABC), which generally outperforms Victim, NTS, and PCS. Furthermore, ABC has the lowest hardware requirements of any multilateral scheme — only a single additional bit per block in the main cache is required to maintain usage information for the allocation decision process, and no swap/save data path is needed.

1 Introduction

The Victim cache [6] was proposed as a method to reduce the impact of conflict misses in direct-mapped cache structures. While the Victim scheme performs well, it requires an auxiliary cache (buffer) to hold victims and a costly data path for swaps and saves between the two caches.

Several other schemes have been proposed which, like Victim, make use of a second (small) cache placed in parallel with the main cache, but unlike Victim, attempt to achieve their performance gains without using a data path between the two caches. The best performing of these *multilateral* [9] schemes are NTS [8][10] and PCS [10], which make their allocation decisions between the two caches based on the predicted usage of an incoming block, based on previous past tours[1]. A multilateral cache, as shown in Figure 1, consists of two unconnected data stores with disjoint contents; the A cache is the larger data store (or the “main” cache), while B refers to the smaller data store (or the “buffer”). Instead of using an A to B data path, these multilateral schemes make use of a history table to contain the information necessary to make informed allocation decisions so as to avoid swapping and saving blocks between A and B. Each of NTS, PCS and Victim have certain applications and system configurations where they perform best, but their average performance gains are comparable.

We have developed a new multilateral allocation scheme, Allocation By Conflict (ABC), that generally outperforms Victim, NTS, and PCS, and has reduced hardware complexity. ABC decides where to allocate blocks (into A or B) based on the current tour usage of the block it might replace in A, rather than on the past tour usage of the incoming block. ABC allocates a block to the A cache if the LRU element of its set in A has not been reaccessed since the last miss reference to that set that did not replace a block in A; otherwise, the block is allocated to the B cache. ABC’s performance gains derive from its ability to reduce the impact of conflict misses by extending the tour length of an LRU block in A that is still actively being referenced when a miss occurs to its set, thereby eliminating the need for swaps as in Victim. ABC’s management decisions require only a single additional bit per block in cache A, rather than a costly table as

1. "Tour" refers to the time between an allocation of a block in the level 1 data cache and its subsequent eviction. A cache block may have many tours through the cache during program execution.

in NTS and PCS.

In this paper we discuss the ABC scheme operation and compare its performance to Victim [6], NTS [8], and PCS [10]. For comparison, we also evaluate the performance of a random allocation scheme. We find that ABC generally outperforms NTS, PCS, and Victim in all but the floating point benchmarks (CFP2000). Surprisingly, despite its lack of “intelligent” decision making, Random’s performance is often comparable to NTS, PCS, and Victim.

Section 2 reviews previous cache allocation schemes and Section 3 describes the new Allocation By Conflict scheme, as well as Random allocation. Our simulation environment and experimental results are described and analyzed in Section 4. We conclude in Section 5.

2 Previously proposed allocation schemes

Several schemes have been proposed for managing multilateral cache structures, in particular, Dual [4], NTS [8][10], MAT [5], and PCS [10]. Of these, NTS and PCS were found to perform best overall [10]. We compare the performance of ABC to NTS, PCS, and to Victim [6] which would effectively be a multilateral cache structure if swap penalties are ignored (set to 0, i.e. a 1 cycle latency for either an A cache hit or for an A miss that hits in B). As Victim is usually assigned a 2 to 4 cycle swap penalty (a 0 swap penalty) as in our simulation model, it provides an upper bound on the performance of an implementable Victim scheme. In this section we describe the operation of the Victim, NTS, and PCS schemes. We focus our discussion on multilateral first level (L1) caches for the following evaluations.

In the **Victim cache**[6] the B cache is a victim buffer that is used to hold recently evicted blocks (victims) from the A cache in the hope of keeping them in B until their next access. Unlike the multilateral cache in Figure 1, there is no direct path between B and the processor or between A and the secondary cache, but there is a bidirectional path between A and B. A block evicted from A as the result of a new block’s arrival is placed (saved) in B, a block evicted from B returns to the next level of memory, and a block hit in B is swapped with the LRU block of the corresponding set in A. The A to B data path is costly since it is bidirectional and lies on the critical path when there is a hit in B.

Many previous evaluations have been performed on the Victim cache scheme, and several improvements to the basic Victim scheme have been proposed, including Selective Victim Caching [12] to reduce saves and swaps and selectively maintain blocks in L1, and Prediction Caches [1] to combine streaming buffers [6] with Victim buffers and also selectively hold blocks in L1. Each of these schemes improve the performance, but each requires additional hardware support. Victim caching is combined with several other strategies in [3].

NTS (nontemporal streaming) [8][10] uses hardware to dynamically partition cache blocks into temporal (T) and nontemporal (NT) blocks, based on their reuse behavior during a past tour. A block is classified NT if during a tour in L1, no word in that block is reused. In subsequent tours, NT blocks are allocated in the B cache; all other blocks (those marked T and those for which no prior information is available) are allocated in the A cache. A T/NT bit is associated with the effective address of each block and kept in a lookup buffer that is associatively accessed when a miss occurs.

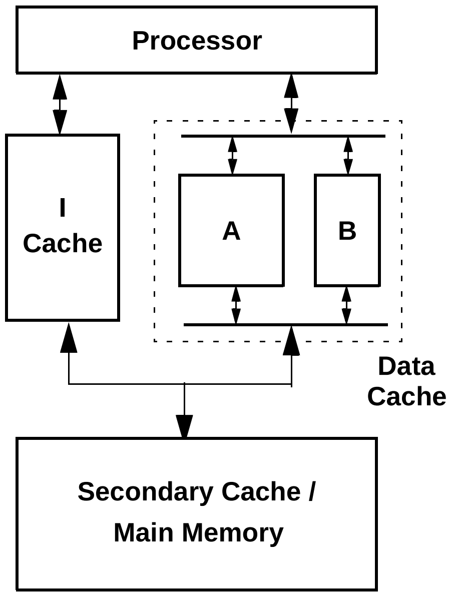
PCS (program counter selective) [10] is a multilateral cache design that evolved from the CNA cache scheme [16]. PCS determines block placement based on the program counter value of the memory instruction causing the current miss, rather than on the effective address of the miss block as in NTS. Thus, in PCS, recent tour characteristics among all blocks recently brought to cache by this memory accessing instruction, rather than that of only the current block being allocated as in NTS, is used to determine the placement of this block[10]. PCS performance is best in applications (like **twolf** and **ammp**) where the reference behavior of a given datum is well-correlated with the memory-referencing instruction that brings the block to cache.

3 New allocation schemes

3.1 Allocation By Conflict (ABC)

NTS and PCS focus on placement decisions by asking whether an incoming (miss) block is deemed to have temporal or nontemporal reference behavior, and placing it in A or B accordingly. The Allocation By Conflict scheme (ABC) focuses instead on the replacement decision by asking whether the *conflict block* (the block that would be replaced in A if the incoming block were allocated to A) is still being actively referenced, i.e. whether it is deemed desirable to prolong the tour length of the conflict block. If so, the incoming block is placed in B, replacing some other block; otherwise it is placed in A and replaces the conflict block.

How is ABC implemented? In particular how does it decide whether the conflict block is still being actively referenced? For conciseness, we refer to the set of A that the miss block maps to as the *A-set*, and the corresponding set of B as the *B-set*. Whenever a miss block is allocated to B we say that a *CNR* (Conflict with No Replacement) has occurred in its A-set. Each block in A has an associated C bit that indicates whether a CNR to its A-set has occurred since its last reference. In particular, when a block begins a tour in A, its C-bit is set to 0. Whenever a CNR occurs, C is set to 1 for all the blocks in that A-set.



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 instruction 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
Functional Unit 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	L1 (A and B) write-back, write-allocate, 32 byte lines, 1 cycle hit latency (0 swap penalty for Victim caches), 18 cycle miss latency, non-blocking, 8 memory ports

Figure 1: Processor and memory subsystem characteristics.

TABLE 1: Parameter value of the configurations studied.

	Single	**Victim**		**ABC/NTS/PCS/Random**	
Cache	A	A	B	A	B
Size	32 / 64K	32K	4K	32K	4K
Associativity	1,2,4,8 / 1	1,2,4	32	1,2,4	32

The C-bit of a block is reset each time the block is referenced. Thus C=1 indicates that at least one CNR has occurred in this A-set during the current tour of this block and this block has not been referenced since the last CNR to this A-set. For completeness, empty or invalid blocks have C=1 by default. The conflict block is deemed "being actively referenced" if its C bit is 0.

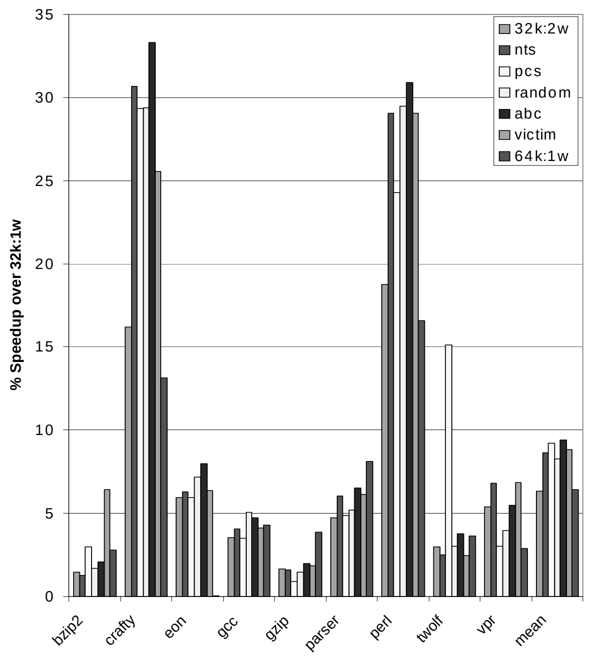
ABC simply allocates a miss block to A if C=1 for the conflict block, and to B if C=0; both A and B use LRU replacement. A block in A thus continues to remain in A until a miss occurs while it is the LRU block of its A-set *and* at least one CNR has occurred to its A-set during its current tour *and* this block has not been referenced since the most recent such CNR, i.e. until it is the conflict block of a miss *and* is deemed "not being actively referenced." The B-cache must simply accommodate all block tours that begin when the conflict block (in A) is deemed "being actively referenced."

What are some consequences of the ABC policy? If a block is accessed at least once between successive CNRs to its A-set, it can remain in A indefinitely. Note that if a block has not been accessed since the last CNR to its A-set, then it could have been replaced at that last CNR without being missed from then until now; it is therefore considered to be a good candidate for replacement now. Thus, ABC makes active *replacement* decisions based on whether to retain or replace the conflict block in A, which in turn is based only on the current tour behavior of the conflict block since the last CNR to its set. By contrast, NTS and PCS make active *placement* decisions based only on past tour characteristics of the miss block or miss instruction, respectively; Victim makes no active decisions and simply uses B to buffer recently replaced blocks from A. ABC, like NTS and PCS, but unlike Victim, requires no data path between the A and B caches.

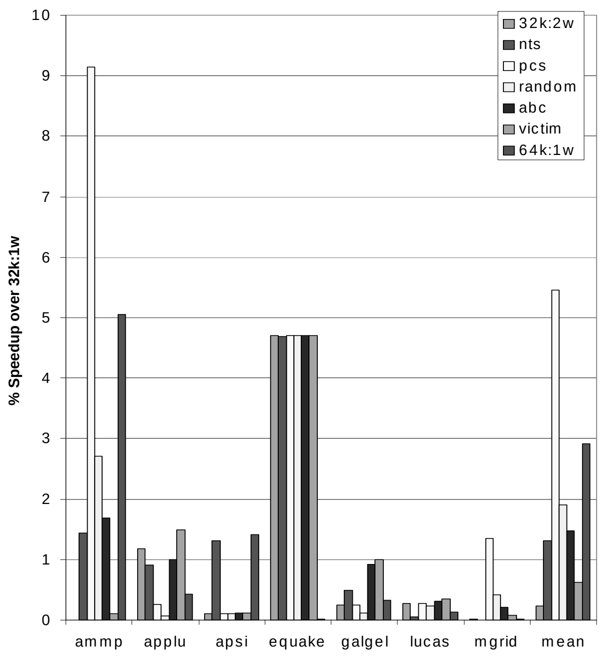
We have also evaluated two ABC variants: a) when a CNR occurs, rather than setting C=1 for all blocks in the A-set, we set C=1 for only the LRU block and b) each C bit is replaced with a 2-bit saturating counter [13]. However, neither of these schemes performed as well as the ABC scheme presented here.

3.2 Random allocation

We also evaluate the Random allocation scheme to compare its performance against multilateral caches of a given size and configuration. Although Random uses no intelligence in its allocation decisions, it provides surprisingly good performance. The Random allocation scheme



i) CINT2000



ii) CFP2000

Figure 2: Percent speedup for each application and each cache (direct mapped A configurations).

has no data path between the A and B caches, each of which uses LRU replacement. A virtual coin-flip decides whether a miss block is allocated to A or B. A versus B allocations are equally likely in this paper; biased probabilities of allocation were also evaluated in [13], but showed only similar or degraded performance.

Table 2: IPC for the baseline (32k:1w) system

CINT2000	IPC	CFP2000	IPC
bzip2	6.2673	ammp	0.8568
crafty	6.4367	applu	7.0527
eon	6.7306	apsi	9.0862
gcc	5.9598	equake	6.7031
gzip	5.3890	galgel	4.3838
parser	3.5620	lucas	7.2748
perl	4.6181	mgrid	6.4978
twolf	3.9622		
vpr	4.6796		

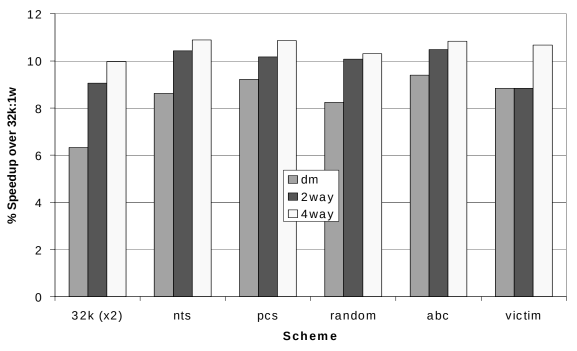
4 Experimental results

In this section, we present the performance of 4 multilateral caches embedded in the highly concurrent system sketched in Figure 1 (NTS, PCS, ABC, and Random) and Victim caches (which include an A to B swap/save path and use the B cache as a victim buffer). All performance is shown as percent speedup relative to a conventional (no B cache) 32 KB direct-mapped (32k:1w) baseline cache configuration. We also include 32k:2w and 64k:1w conventional caches for comparison. Parameters of the specific configurations studied range over the values shown in Table 1. The experimental evaluations were carried out on nine CINT2000 and seven CFP2000 benchmarks [11], run for 2 billion instructions each, using the SimpleScalar toolset developed by Austin and Burger [2]. For benchmarks with multiple data sets provided, the harmonic mean performance across these sets is used. The performance of the baseline system, in instructions per cycle (IPC), is shown in Table 2 for each of these benchmarks. A preliminary evaluation for some Windows NT applications is presented at the end of this section.

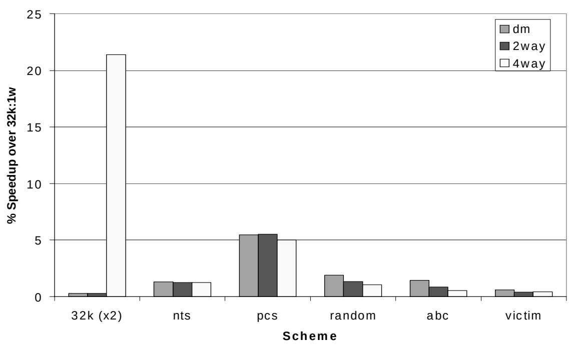
4.1 Performance with direct-mapped A caches

The percent speedup achieved by each cache scheme for each program is shown in Figure 2, relative to the baseline performance shown in Table 2. The speedup in harmonic mean performance over the CINT2000 and over the CFP2000 benchmarks is also shown (under "mean"). Overall, the speedup obtained by using the multilateral[2]

2. From here on we use the term multilateral informally to include Victim as well as NTS, PCS, ABC, and Random.



i) CINT2000



ii) CFP2000

Figure 3: Percent speedups in harmonic mean performance of each cache scheme as A associativity increases. The effects of increasing associativity on a single structure cache are also shown (labeled 32k (x2) for which the *dm* bar shows 32k:2w performance, *2way* shows 32k:4w performance, and *4way* shows 32k:8w performance).

cache schemes ranges from virtually none in **lucas** to ~33% in **crafty** with ABC. There are also some singularity points like PCS on **ammp** which does sufficiently well on this very low IPC application to dominate the CFP2000 mean speedup. Clearly, some of the benchmarks tested do not benefit much from any of the improvements offered by the cache schemes evaluated, i.e. better management by the multilateral schemes of the L1 data stores with B buffers, increased associativity of a single cache (32k:2w), or a larger cache (64k:1w). In benchmarks where there is appreciable performance gain over the base cache, the multilateral schemes often perform as well as or better than a more associative single cache or a larger direct-mapped cache.

Among the multilateral schemes, we see several trends that were exposed in earlier studies [9][10][13][14][15]. With a direct-mapped A cache, NTS usually outperforms PCS (in all benchmarks except **bzip2, twolf, ammp, lucas,** and **mgrid**), and Victim usually outperforms both NTS and PCS (the exceptions being **crafty, gcc, twolf, ammp, apsi,** and **mgrid**). Victim has the advantage of a 0 swap latency, but even when a 2 or 4 cycle swap latency is incorporated for a B hit that misses in A, Victim still generally outperforms NTS and PCS[13].

Surprisingly, the Random allocation scheme also performs very well, equaling or beating both NTS and PCS in some of the benchmarks (**eon**, **gcc**, **perl**, and **equake**). Random's performance is intriguing since, unlike NTS and PCS, it does not take any reuse information into account when making its allocation decisions.

ABC achieves the highest mean speedup over CINT2000, even outperforming Victim in all of these integer benchmarks except **bzip2** and **vpr**. In the floating point benchmarks, Victim outperforms ABC in **ammp, apsi,** and **mgrid**, PCS dominates the mean speedup due to its high speedup on **ammp**.

Both ABC and Victim outperform an associative cache of similar size (32k:2w) and a single structure cache nearly twice as large (64k:1w) in the CINT2000 harmonic mean speedup. However, this result does not hold for CFP2000 due to the high 64k:1w cache performance on **ammp** and **apsi**. The best harmonic mean speedups over the baseline 32k:1w cache are 9.39% (ABC) over the integer benchmarks and a more modest 5.51% (PCS) over the floating point benchmarks.

4.2 Performance with associative A caches

The speedups in harmonic mean performance achieved by each scheme as the associativity of the A cache is increased to 2- and 4-way are shown in Figure 3. The base configuration is the 32k:1w cache as above. For comparison 32k:2w, 4w, and 8w conventional caches are shown in the bars labeled 32k (x2).

We see, from the CINT2000 data in Figure 3, that as the associativity of the A cache increases the speedups increase for each scheme except Victim which anomalously shows slightly lower performance for 2-way, concurring with the results in [10][13]. All the multilateral caches show similar CINT2000 performance with a 4-way associative A cache, and they all outperform an 8-way cache of the same size (32k:8w). Random's speedup falls off somewhat and is slightly lower than any of the "intelligent" multilateral cache schemes.

Performance over the CFP2000 floating point benchmarks is much more erratic. PCS offers the best multilateral cache performance, due primarily to its 8% to 9% speedup on **ammp**; a 43% speedup on **ammp** is also responsible for the high mean of the conventional 32k:8w cache.

Over CINT2000, ABC consistently offers the highest performance, with almost an 11% improvement in the 4-

way case. NTS is generally close to ABC, and beats or equals PCS.

4.3 Random allocation performance

The Random allocation scheme might serve as a baseline for comparing multilateral structures of a given size and configuration. However, we found that in some cases the Random scheme performed better than either NTS or PCS, particularly on CINT2000 with a direct-mapped A cache.

Random's high performance is possible for several reasons: 1) Random allocates blocks to A and B with a 50/50 ratio, thereby distributing tours equally among both caches. While a given block may in fact benefit, i.e. experience longer tours, when allocated to the more associative B cache or the larger size A cache, when the block is evicted from a sub-optimally allocated tour, it has a 50/50 chance of being allocated to its preferred cache structure for its next tour. 2) Each of the caches is managed via LRU replacement, so blocks that are recently accessed have a high likelihood of remaining in the cache they are allocated to. Thus, despite not making any "intelligent" allocation decision for a miss block, the use of LRU and the even allocation split between the two caches gives a block a reasonably good chance of spending most of its total L1 tour time in its preferred structure.

While the Random scheme's performance is reasonable, its behavior is not easily predictable across different benchmarks. Schemes whose allocation decisions are based on some current or past reuse information, like the ABC, NTS, and PCS schemes, appear to be more rational and do indeed perform better overall as applications and cache configurations are varied. Nevertheless, Random's good performance does cast some doubt on the benefits of relying on past tour reuse information for making allocation decisions.

4.4 ABC vs. NTS and PCS

While NTS and PCS make their block allocation predictions based on past usage of miss blocks (NTS) and other blocks associated with them (PCS), ABC determines whether to allocate an incoming miss block into A or B by looking only at the *current* usage of the block *in cache A* that conflicts with the miss block. Using the current state of the cache to make placement decisions usually results in better performance improvement than using past tour usage behavior, as tour usage behavior tends not to have very high persistence [15].

ABC thus makes its allocation decision based only on whether to replace the conflict block in A (based on whether it has been referenced since the last CNR to its A-set during its current tour), while NTS and PCS look only at the miss block's (NTS) or instruction's (PCS) temporality of usage in previous tours. Furthermore, ABC stores only a single bit (C bit) per A-cache block and makes its allocation decision simply by checking the C bit of the conflict block. NTS and PCS allocation decisions are made via a lookup and associative compare using a large table that stores temporality information about past tours.

When the associativity of the A cache is low, too many conflicts occur and many tours end prematurely before temporal reuse occurs, often leading to misclassification of blocks as nontemporal. As the associativity of the A cache increases, fewer conflicts occur, and blocks that are evicted from A that show nontemporal reuse are more likely to actually be nontemporal blocks. The longer a block stays in the cache, the more likely its reuse behavior is to reflect its optimal reuse characteristics [15]. As a result, the performance of NTS and PCS improves relative to Victim and Random as A cache associativity increases [13]. ABC has no such misclassification problem.

4.5 ABC vs. Victim

Inherently, the way both ABC and Victim attain their performance gains is similar: both reduce the impact of hot sets by providing dynamically-sized cache sets, and both achieve this by using the B cache as a buffer that may contain several blocks that map to the same set of the A cache. In both ABC and Victim, the allocation of some block to B occurs when a conflict occurs in cache A. However, beyond that, their approach to providing these dynamically-sized sets differs. Victim in effect may create a larger size cache set whenever a conflict occurs in a set of A — the new miss block is allocated to A and the replaced block, the victim of that allocation, is saved in the B cache. If the element replaced from B due to the victim's allocation maps to the same set as the new miss block, the dynamic size of that set remains constant. If the replaced block and the victim are from different sets, the dynamic size of the victim's set increases, and that of the replaced block's set decreases by one.

ABC, on the other hand, may only create a larger size cache set when a conflict is found in A and the conflict block has yet to experience a CNR, or the conflict block has been accessed since the last CNR to that A-set. In this case, the conflict block is retained in A and the miss block is allocated into B; if the miss block and the replaced block map to different sets in A, the dynamic size of the miss block's set increases by one block and the dynamic size of the replaced block's set decreases by one. In all other cases, the miss block is allocated into A and all set sizes remain the same.

When the A cache is direct-mapped, the ABC scheme outperforms the Victim cache in all but two CINT2000

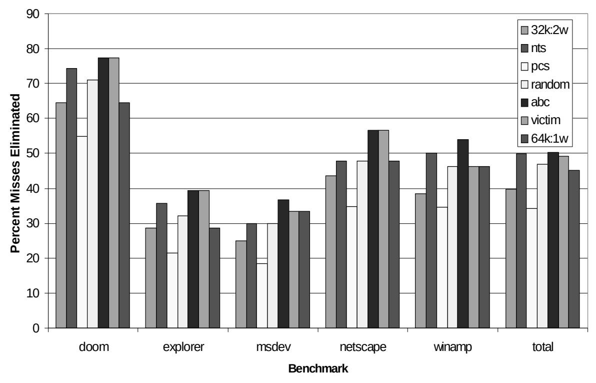


Figure 4: Percentage of misses eliminated from the (32k:1w) baseline for Windows NT applications.

benchmarks (**bzip2** and **vpr**). In general, Victim saves too many useless elements in the B cache and does not use the B cache as effectively as ABC. ABC's better usage of the B cache is due to its ability to allow elements to reside in B longer, as it is possible to evict items directly from A; in Victim, all evicted items from A must first travel through the B cache before they can leave the L1 cache structure. However, in the CFP2000 benchmarks, this additional time in L1 for each evicted A cache element benefits Victim cache performance, which equals or beats ABC's performance in all the benchmarks except **ammp** and **mgrid**. Although ABC's mean speedup beats Victim on CFP2000, neither scheme has a speedup of more than 1.5% over the base cache.

The number of useful elements found in the B cache of a Victim scheme decreases greatly as A cache associativity increases. As a result, the performance benefit of the B cache in a Victim scheme degrades as A cache associativity increases [13]. In ABC, however, the B cache continues to perform well since it is used to accommodate new block tours when the conflict block's tour length is extended in A; furthermore when an inactive block is evicted from A, it returns to the next level of memory thereby allowing the B cache elements to remain and potentially be reaccessed before their eviction.

4.6 A preliminary evaluation for some Windows NT applications

Additionally, we looked at the impact of these multilateral cache organizations on a set of Windows NT 4.0 applications that were introduced in [17]. Table 3 shows some characteristics of the applications. These benchmark measurements include all relevant Windows NT 4.0 system code.

We measured the miss rate of the data cache for the baseline cache and the seven caches shown in Figure 4. Since, as Table 4 shows, most of the miss rates for these applications are rather low for our 32k:1w data cache baseline, a more useful impression of the significance of the performance gain is given by Figure 4 which shows the percentage of the misses of the baseline cache that are eliminated by the cache under evaluation. The bars labeled "total" reflect the total number of misses in a given scheme over all these applications relative to the total for the baseline. To the extent that the data cache miss penalty is the performance bottleneck, the performance of a system with a sufficiently powerful processor is inversely proportional to the percentage of misses remaining.

Table 3: Windows NT Application Characteristics

Application	Instructions ($x10^6$)	Data Refs ($x10^6$)	Avg. Basic Block Size
doom	761	510	5.87
explorer	408	247	5.40
msdev	697	432	4.14
netscape	865	500	4.90
winamp	935	772	6.64

As Figure 4 shows, on each of these Windows applications ABC eliminates at least as many misses from the baseline cache as Victim. In total, ABC eliminates 1% more misses than Victim, 5% more than a 64k:1w, and 10% more than 32k:2w.

Miss rates are shown in Table 4 for each application and cache scheme, with direct-mapped A caches. Although only miss rates are shown in Table 4, our simulations also model delayed hits (accesses to a cache line that is currently being fetched). The access latency for delayed hits vary all the way from a cache hit to a cache miss latency. For example, in **apsi**, all of the schemes have a miss rate of 22%. However, the delayed hit rates for 32k:1w and NTS are 78.3% and 74.9%, respectively. Therefore NTS has a higher speedup since more accesses are hits rather than delayed hits. The problem with using delayed hits for comparison is their variable length latency; hence using the percent speedup is a more useful metric.

5 Conclusions

In this paper we have introduced a new cache management scheme, Allocation By Conflict (ABC), and compared its performance to the Victim cache, two previously proposed reuse-based multilateral schemes, NTS and PCS, and to Random multilateral allocation. As shown in earlier work, the performance of each of the multilateral schemes is better than conventional single structure caches of the same size, and comparable or better than

Table 4: Miss rate per application with 32k:1w A caches (best cache scheme is bold)

	32k:1 w	32k:2 w	nts	pcs	ran	abc	vic	64k:1 w
bzip2	2.8	2.7	2.7	2.8	2.7	**2.6**	2.9	**2.6**
crafty	4.0	1.8	0.9	0.9	1.0	**0.7**	0.8	1.7
eon	1.5	0.6	0.3	0.4	0.3	**0.2**	1.5	0.9
gcc	7.7	7.2	7.0	**6.8**	**6.8**	**6.8**	**6.8**	6.7
gzip	3.0	2.8	2.8	2.8	2.8	**2.7**	3.9	2.4
parser	4.9	4.0	3.9	4.1	4.0	3.8	3.9	**3.7**
perl	2.6	0.9	**0.3**	0.7	**0.3**	**0.3**	**0.3**	1.0
twolf	5.9	5.3	5.3	**0.2**	5.4	5.2	5.3	5.2
vpr	5.1	4.4	**4.2**	4.7	4.5	4.3	**4.2**	4.4
ammp	29.9	30.1	29.2	**26.1**	28.7	29.2	29.9	27.8
applu	5.8	**5.4**	**5.4**	5.9	5.7	5.5	5.8	5.5
apsi	**22.0**	**22.0**	**22.0**	**22.0**	**22.0**	**22.0**	**22.0**	**22.0**
equake	0.8	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	0.8	0.8
galgel	11.2	9.9	9.1	22.6	13.0	11.5	11.2	**9.1**
lucas	3.3	**3.2**	3.3	**3.2**	**3.2**	**3.2**	3.3	**3.2**
mgrid	**2.4**	**2.4**	**2.4**	4.1	2.9	2.7	**2.4**	**2.4**
doom	3.1	1.1	0.8	1.4	0.9	**0.7**	**0.7**	1.1
explorer	2.8	2.0	1.8	2.2	1.9	**1.7**	**1.7**	2.0
msdev	6.0	5.5	4.2	4.9	4.8	**3.8**	4.0	4.0
netscape	2.3	1.3	1.2	1.5	1.2	**1.0**	**1.0**	1.2
winamp	2.6	1.6	1.3	1.7	1.4	**1.2**	1.4	1.4

conventional caches with the same A cache associativity of nearly twice the size.

We have shown that ABC has the highest overall performance of the reuse-based allocation schemes because it makes its allocation decisions based on the current usage of blocks in the cache, rather than on the past tour usage of incoming miss blocks as in NTS and PCS. Furthermore, ABC outperforms Victim over each of the application sets: CINT2000, CFP2000, and Windows NT. Surprisingly, Random allocation also performs quite well; however, its performance is less predictable as the applications and cache configurations are varied.

ABC has much lower hardware requirements than NTS or PCS, requiring only a single additional bit per block in the A cache. NTS and PCS require more bits, and a Detection Unit for storing reuse information about past tours and an associative lookup for making allocation decisions. Unlike ABC, Victim requires a costly bidirectional, time-critical data path between the caches for swaps and saves. Despite its low cost, ABC has the overall best performance.

6 References

[1] J. E. Bennett and M. J. Flynn, “Prediction Caches for Superscalar Processors,” *Proceedings of ISCA-30*, pp. 81–90, December 1997.

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

[3] J. D. Collins and D.M. Tullsen, “Hardware Indentification of Cache Conflict Misses” *Proceedings of MICRO-32*, November 1999.

[4] A. Gonzalez, C. Aliagas, and M. Valero, “A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality,” *Supercomputing ’95*, pp. 338–347, July 1995.

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

[6] 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.

[7] 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.

[8] J. A. Rivers and E. S. Davidson, “Reducing Conflicts in Direct-Mapped Caches with a Temporality-Based Design,” *Proceedings of the ICPP*, vol. I., pp. 151 - 160, August 1996.

[9] J. A. Rivers, E. S. Tam, and E. S. Davidson, “On Effective Data Supply for Multi-Issue Processors,” *Proceedings of the 1997 ICCD*, pp. 519-528, October 1997.

[10] 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 ICS*, pp. 449 - 456, July 1998.

[11] Standard Performance Evaluation Corporation, CPU2000 documentation. *http://www.spec.org/osg/cpu2000/docs*

[12] D. Stiliadis and A. Varma, “Selective Victim Caching: A Method to Improve the Performance of Direct-Mapped Caches,” *IEEE Transactions on Computers*, vol.46, no.5, pp. 603–610, May 1997.

[13] E. S. Tam, “Improving Cache Performance Via Active Management,” Ph.D. Dissertation, Dept. of EECS, University of Michigan, Ann Arbor, 1999.

[14] E. S. Tam. J. A. Rivers, G. S. Tyson, and E. S Davidson, “*mlcache:* A Flexible multilateral Cache Simulator,” *Proceedings of MASCOTS’98*, pp. 19–26, July 1998.

[15] E. S. Tam, J. A. Rivers, V. Srinivasan, G. S. Tyson, and E. S. Davidson, “Evaluating the Performance of Active Cache Management Schemes,” *Proceedings of the ICCD*, pp. 368–375, October 1998.

[16] 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.

[17] S.Vlaovic, E.S. Davidson, and G.S. Tyson. "Improving BTB Performance in the Presence of DLLs," *Proceedings of MICRO-33*, pp. 77-86, December 2000