Emulating Optimal Replacement with a Shepherd Cache

Kaushik Rajan† and **R. Govindarajan**† ‡
‡Department of Computer Science and Automation
†Supercomputer Education and Research Centre
Indian Institute of Science
kaushik@hpc.serc.iisc.ernet.in
govind@csa.iisc.ernet.in

Abstract

The inherent temporal locality in memory accesses is filtered out by the L1 cache. As a consequence, an L2 cache with LRU replacement incurs significantly higher misses than the optimal replacement policy (OPT). We propose to narrow this gap through a novel replacement strategy that mimics the replacement decisions of OPT. The L2 cache is logically divided into two components, a *Shepherd Cache* (SC) with a simple FIFO replacement and a *Main Cache* (MC) with an emulation of optimal replacement. The SC plays the dual role of caching lines and guiding the replacement decisions in MC. Our proposed organization can cover 40% of the gap between OPT and LRU for a 2MB cache resulting in 7% overall speedup. Comparison with the *dynamic insertion policy*, a *victim buffer*, a *V-Way cache* and an LRU based *fully associative cache* demonstrates that our scheme performs better than all these strategies.

1 Introduction

Modern processors employ a hierarchy of caches to bridge the processor-memory performance gap. The L1 cache is designed with simplicity and fast access time in mind. The L2 cache is designed to minimize expensive accesses to lower level caches (50-100 cycles) or memory (200-1000 cycles). L2 caches are therefore larger (512KB-4MB), have higher associativity (8,16 or 32) and can have a relatively more complex design [10, 6, 8]. While the size of the L2 cache is determined by the available real estate on chip, other parameters like placement (cache indexing), associativity and replacement can be fine tuned to extract more performance. The placement mechanism and associativity are simultaneously tuned in Recent proposals like the V-way cache [10] and adaptive cache compression [1] simultaneously tune the placement mechanism and associativity by decoupling the tag portion of the cache from the data portion. On the other hand, proposals like the dynamic insertion policy [11] and adaptive replacement [12] are examples of schemes that modify the replacement mechanism. While placement and associativity are critical path functionalities and have a direct impact on access time, replacement decisions are made off the critical path. Hence, a better replacement strategy that ensures fewer misses even at the expense of a slightly more complex design is desirable.

The existence of a provably optimal replacement policy, OPT [2], provides a readily available touchstone to determine the quality of a replacement policy. According to OPT, the optimal candidate for replacement is the one that is accessed farthest in the future, or in other words, the line to which an access is least imminent. Unfortunately, as OPT requires the knowledge of future accesses, it is not practically implementable. Practical replacement policies learn from the past accesses and make a replacement decision based on this history information. The effectiveness of the policy (like LRU,LFU) depends on the how well the gathered information (recency, frequency) can predict future accesses and hence approximate the optimal decision.

Due to the inherent temporal locality in L1 accesses, recency of use approximates the imminence of accesses fairly well. Hence LRU replacement is the obvious choice for L1 caches. However, as only memory accesses that miss the L1 cache go to the L2 cache, accesses with temporal locality are filtered out. As a consequence an LRU replacement policy is not very effective for L2 caches.

To highlight the ineffectiveness of LRU we compare the performance of LRU with OPT. In Figure 1 we plot the Misses Per Kilo Instruction (MPKI) incurred by four cache configurations; 512KB-lru16 - a 512KB 16 way associative cache with LRU, 512KB-lruFA - a fully associative 512KB cache with LRU, 512KB-opt16 - a 512KB 16 way associative cache with OPT, and 256KB-opt8 - a 256KB 8 way associative cache with OPT. It is to be noted that a 256KB 8 way associative cache has the

same number of sets (256) as other configurations but has only 8 ways per set. Clearly a 512KB-opt16 outperforms 512KB-lru16 achieving a reduction in MPKI of up to 83.5% (for twolf). What is more interesting to note is that even an 8 way OPT for a smaller 256KB cache (256KB-opt8) can achieve better performance than 512KB-lru16 for most benchmarks. In fact, in most cases it even out-performs 512KB-lruFA. This trend is seen at higher cache sizes also. These observations indicate that dedicating a part of the cache to guide the replacement of the remaining cache space would be beneficial.

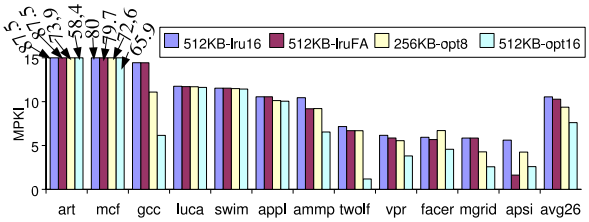


Figure 1: LRU vs OPT. Values going out of Y axis range are indicated along the bar. Only benchmarks with MPKI of at least 5 are plotted. The *avg26* is the average over all 26 SPEC2000 benchmarks.

In this paper, we propose a novel organization which facilitates the emulation of the OPT replacement policy in hardware with the aid of a *shepherd cache*. In our organization the L2 cache is logically divided into two components, the *Shepherd Cache* (SC) with a simple FIFO replacement policy and a *Main Cache* (MC) with an emulation of optimal replacement. As observed previously, OPT requires a lookahead into future accesses to be able to make an optimal decision. SC aids MC replacement by providing this lookahead window. Lines that miss in the cache are buffered in the SC before a MC replacement decision is taken. While the line is in SC it gathers information about potential replacement candidates in MC. This gathered information facilitates in making a replacement decision that approximates OPT fairly well. It is to be remembered the proposed organization only tries to emulate optimal replacement in the main cache and not the entire cache (SC+MC). Though SC and MC are logically separate, we cleverly choose the organizational parameters of the two components so that the two logical components can be fitted in as a single physical structure. By doing so, we avoid any expensive movement of lines from one component to the other.

Performance evaluation over the entire SPEC2000 suite reveals that our proposed replacement strategy closes the gap between OPT and LRU by 31% to 53% for cache sizes ranging from 512KB to 4MB. Also our proposal out-performs other contemporary schemes including the dynamic insertion policy [11], a fully associative cache with LRU replacement, the V-way cache [10] and a victim buffer [7]. The overall system performance is improved by up to 47.1% with an average speedup of 7%.

The rest of the paper is organized as follows. Section 2 summarizes the optimal replacement policy. Section 3 elaborates on our proposed shepherd cache organization. Section 4 describes the simulation methodology and reports experimental results. Section 5 describes some of the related work and we conclude in Section 6.

2 The Optimal Replacement Policy

On a miss any replacement policy could choose to replace either one of the lines present in the cache, or choose not to place (bypass) the line that caused the miss in the cache. Hence there are $A + 1$ replacement candidates for an A-way set associative cache. The optimal replacement policy (OPT) chooses for replacement, the replacement candidate that has a next access farthest in the future [2, 9]. That is, if we order the replacement candidates according to their imminence of access, OPT would replace the line that is least imminent. As a corollary, if all lines in the cache have a next access before the line that caused the miss is accessed again, then the optimal decision is to not place the miss causing line in the cache at all. Such a replacement is referred to as *self-replacement* in this paper.

We briefly illustrate the working of OPT with a simple example. We demonstrate the working with a 4 way associative cache with entries (A_1,A_2,A_3,A_4) in one of its sets initially. The access stream for the set is shown in Figure 2. When A_5 misses the cache, OPT could choose to replace either A_1,A_2,A_3,A_4 or choose not to place A_5 in the cache. To pick the optimal candidate, OPT orders these lines according to their next access in the access stream and picks the cache line that is last in this order for replacement. For A_5 the optimal order is shown in row 2 of Figure 2. The lookahead window required to make the replacement decision is circled in the figure.

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		(0		1		2	3	4)				
OPT order for A_6				(0	1	2	3				4)	

Figure 2: Working of OPT

Based on this ordering A_5 replaces A_2 and the cache now contains the lines (A_1,A_3,A_4,A_5). The next access A_1 is a hit and no changes are made to the cache. On access to A_6, a replacement candidate needs to be picked among A_1,A_3,A_4,A_5 and A_6. The optimal order is as shown in row 3 of Figure 2 . Based on this order, OPT decides not to place A_6 in the cache at all (self replace-

ment) and leaves the cache as it is. The replacements proceed on in this manner.

3 Emulating OPT with a Shepherd Cache

As is evident from the above example, OPT needs to lookahead into future accesses to be able to determine the optimal replacement candidate. We propose to emulate this lookahead through a Shepherd Cache. We split the L2 cache into 2 component caches, the Shepherd Cache (SC) and the Main Cache (MC), and try to emulate OPT in MC. New lines are initially buffered in the SC. While in the SC each line gathers information regarding the optimal ordering of replacement candidates. Finally, when the line reaches the head of the SC, a decision is made on whether it should replace an MC line or self-replace.

3.1 Overview of Shepherd Caching

Below we illustrate how the SC aids in emulating optimal replacement for MC. Subsequent sections provide a detailed description of the hardware structures required to achieve this. To emulate OPT for a 4 way associative MC we add 2 Shepherd Cache ways to each set. The two SC ways of the set are referred to as SC_1 and SC_2. To emulate OPT the imminence order[1] with respect to each SC line needs to be tracked. A column of counters is associated with each SC line for this purpose. The columns for SC_1 and SC_2 together form a Count Matrix (CM) as shown in Figure 3(a). Further, each SC line has a Next Value Counter associated with it (NVC_1 and NVC_2). The NVC's keep track of the next value to be assigned along the column. Initially we assume that the MC contains the lines (A_1, A_2, A_3, A_4) and that the SC is empty (refer to Figure 3(a)). Consider the same sequence of accesses as used previously (Figure 2 and Figure 3).

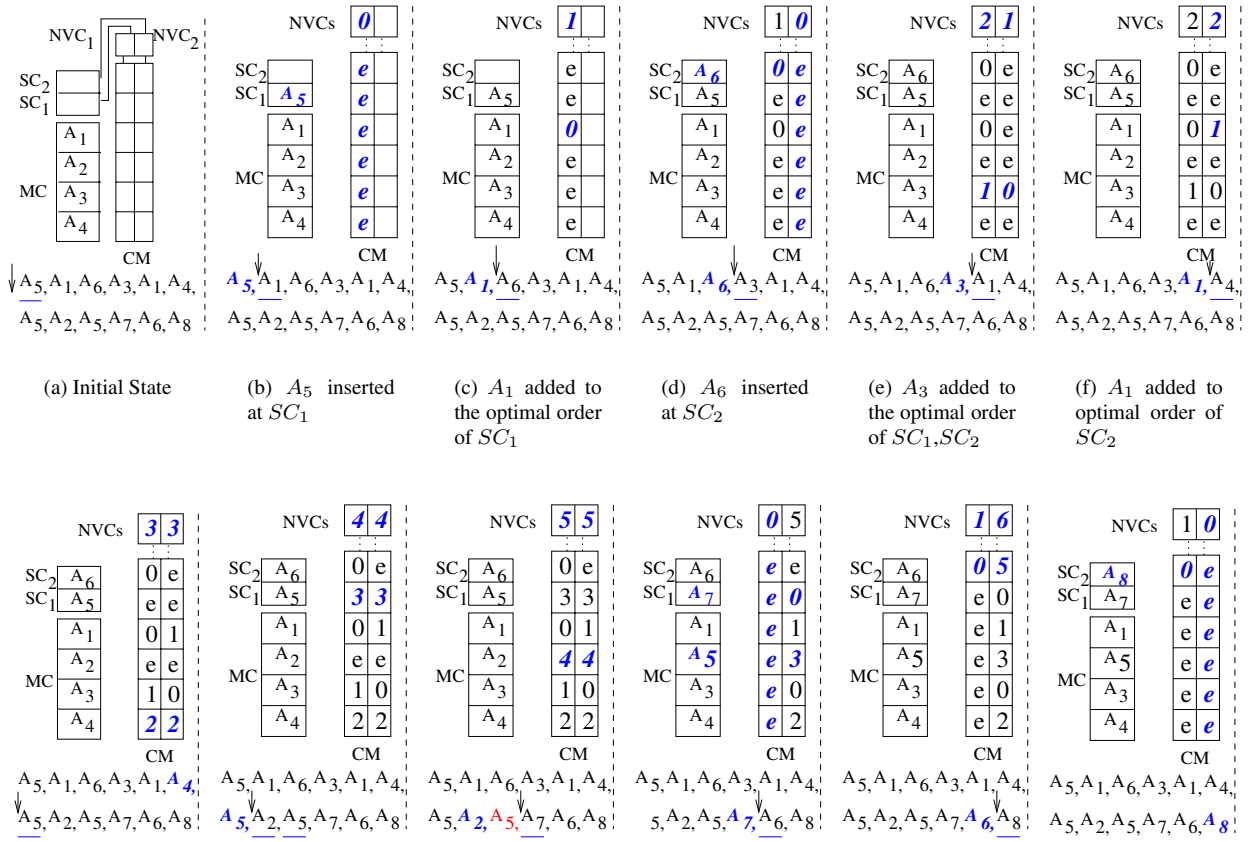
The access to A_5 misses and is buffered in SC at SC_1. The column of counters associated with SC_1 is initialized to 'e' indicating empty (Figure 3(b)) and NVC_1 is initialized to 0. The next access, A_1, is a hit in MC. The optimal order with respect to A_5 is updated to indicate the position of A_1 (Figure 3(c)). This is done by assigning the value at NVC_1 (zero) to the counter corresponding to A_1 in SC_1's column. NVC_1 is then incremented to 1. Now the column for SC_1 reflects that A_1 is the most imminent access with respect to A_5.

The next access A_6 misses and is assigned a new SC entry (SC_2). As A_6 is not a replacement candidate for A_5, A_6 does not need to be ordered with respect to A_5. Remember that only A_1-A_5 are replacement candidates for A_5. To reflect this a dummy value (zero) is set along A_6's row for SC_1's column. Figure 3(d) indicates the changes made. The next accesses A_3 is again a hit and the counter values in both columns (corresponding to SC_1 and SC_2) are updated to reflect the optimal order as shown in Figure 3(e). As before the corresponding NVC's are copied into the row of A_3 and then incremented. The counter matrix now indicates that with respect to A_5 the most imminent line is A_1 and A_3 is next. At this point the replacement victim for A_5 is one among the less imminent lines - A_2,A_4 or A_5. For A_6 the CM reflects that A_3 is the most imminent access. The next few accesses $< A_1, A_4, A_5, A_2, A_5 >$ are all hits (either in SC or MC) and updations to the optimal order proceed in a similar manner. Note that A_1 does not again update its counter in SC_1's column as it already has a value indicating its position in the optimal order with respect to A_5 (Figure 3(f)). As a consequence the two columns now have different counter values for A_1, this is consistent with OPT.

After accesses $< A_1, A_4, A_5, A_2, A_5 >$, SC_1's column (Figure 3(i))reflects the complete optimal ordering for A_5, this is identical to the one in row 2 of Figure 2. When the next access A_7 misses, it needs to be placed in the SC by pushing out the oldest SC line, A_5. At this point A_5 will replace an MC line or itself. The line is chosen based on the optimal order gathered in SC_1's column. The column indicates that A_2 has the highest counter value and hence A_5 replaces A_2. This replacement decision (A_5 replacing A_2 in MC) is in tune with OPT's replacement decision. The modified state after the replacement is shown in Figure 3(j). Proceeding further in this manner (Figures 3(k) and 3(l)) it can be seen that A_6 would pick itself as the replacement candidate (again conforming to OPT).

In this example the lookahead provided by SC is sufficient to make optimal replacement choices. As long as the SC is successful in providing sufficient lookahead the 2 component structures are guaranteed to get hits that the MC alone would have achieved if it used OPT. In addition, while SC is gathering imminence information there can be a few extra hits (like the access to A_2) which an MC alone with OPT would not achieve. In comparison a 6 entry LRU would have replaced A_6 on access to A_7. As the immediately succeeding access is to A_6, such a replacement would be premature and would cause an unnecessary miss. Even if SC cannot provide the lookahead to establish a complete ordering among all the replacement candidates, it would still establish a partial ordering among the replacement candidates. This order-

[1] We use the terms imminence order and optimal order interchangeably. The imminence of an access is always with respect to a particular access in the access stream.



(a) Initial State

(b) A_5 inserted at SC_1

(c) A_1 added to the optimal order of SC_1

(d) A_6 inserted at SC_2

(e) A_3 added to the optimal order of SC_1,SC_2

(f) A_1 added to optimal order of SC_2

(g) A_4 added to optimal order of SC_1,SC_2

(h) A_5 added to optimal order of SC_1,SC_2

(i) A_2 added to optimal order of SC_1,SC_2

(j) A_5 moves from SC to MC replacing A_2

(k) A_6 added to optimal order

(l) Self Replacement (A_6 evicts itself)

Figure 3: Working of SC+MC on the access stream $< A_5, A_1, A_6, A_3, A_1, A_4, A_5, A_2, A_5, A_7, A_6, A_8 >$. The recently made modifications and recently processed accesses are shown in blue and italicized, the next set of accesses to be processed is underlined. The links shown in the first figure are implicitly assumed to be present in the remaining graphs. SC : Shepherd Cache, MC : Main Cache, NVC : Next Value Counter, CM : Count Matrix

ing would help in restricting the replacement to one of the less imminent lines. In the following subsections, we describe the Shepherd Cache organization detail.

3.2 Cache Organization

In order to keep the physical organization simple we propose to split the cache such that a fixed number of lines of each set are reserved for SC and the remaining are reserved for MC. The lines belonging to the shepherd cache constitute the *SC-associativity* and the remaining lines of the set constitute the *MC-associativity*. For example, a 512KB cache with 128B line size, 12 way MC-associativity and 4 way SC-associativity will contain 256 sets, with each set having 4 lines for SC and the remaining 12 for MC. Each logical structure is assumed to have support for an inherent baseline replacement strategy. We restrict SC to have a simple FIFO re-

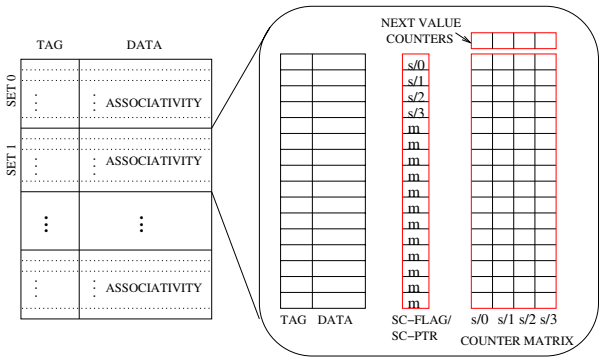


Figure 4: Logical Cache Organization (SC-4, MC-12)

placement, while for the MC we experiment with LRU, LFU and random baseline replacement strategies. It is possible that the SC is unable to provide sufficient lookahead to find the least imminent line but only narrow down the candidates to the less imminent lines. In this

case, the baseline MC replacement strategy is used to choose among the less imminent lines. The overall logical organization of the cache and internal structures within a set are shown in Figure 4. The term *associativity* refers to the total associativity of the set unless explicitly qualified as SC-associativity or MC-associativity.

In order to track the imminence order of accesses a Count Matrix (CM) is associated with each set. The Count Matrix has one column for each SC line to track the optimal order with respect to it, and a row per cache line (SC+MC) as each line can have a different positions in the optimal order with respect to the different SC lines. Each entry in CM is $log_2(associativity) + 1$ bits wide with one bit in each counter used as an empty flag to indicate whether it contains a value or not. In addition to the CM, a single row of Next Value Counters is used to track the next value to be assigned along each column. There has are SC-associativity such entries. These structures were briefly introduced in the previous section.

Further, each line has an *SC-flag* and *SC-ptr* associated with it. The SC flag is used to indicate whether the line belongs to SC or MC. The *SC-ptr* keeps track of the corresponding column for SC lines using a $log_2(SC\text{-}associativity)$ bits. The *SC-flag* and *SC-ptr* together prevent the need for physically moving lines from one component to the other. This would simplify the changes illustrated in Figure 3(j) where A_5 was physically moved to MC. Overall for an SC-associativity of 4 and an MC-associativity of 12 the auxiliary structures (*SC-Flag/ptr*, CM and NVA) amount to 48B per set and 12KB totally for a 512MB cache. This amounts to only 2.3% of the size of the cache. Further, the access to these structures is off the critical path as they only aid in L2 replacements.

Having described the organization of the cache we now explain the working of the cache. In the following discussion a line that hits the cache (SC or MC) is denoted by H, a line that misses the cache (SC+MC) is represented by M, the oldest SC entry by FIFO order is denoted by O and the replacement victim chosen by O based on the order collected is referred to by R.

3.3 Modifications on a Cache Hit

On every cache hit the counters of the row corresponding to the line H are updated based on the *next value counters*. A new value is assigned only if the empty flag is set. The corresponding *next value counters* are incremented. Note that only columns which do not already contain a value are given a new value to ensure that only the first access to a line is tracked. This is required for accurately mimicking OPT as OPT tracks only the immediate next accesses of the lines. Figure 5 illustrates the changes to the structures on a cache hit.

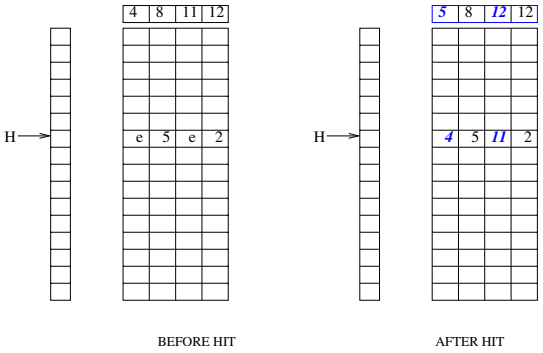


Figure 5: Update on a cache hit. Only relevant CM values are shown, other entries may also be populated. A value other than e for a CM entry implicitly means that the empty flag is reset. Modified entries are in blue and italicized.

3.4 Modifications on a Cache Miss

On every cache miss the incoming line, M, needs to be allocated an entry in the SC and a new column of counters associated with it. Logically two sets of replacements take place: the oldest SC entry O replaces a line R chosen based on the gathered optimal order (explained in the next subsection), and M replaces O in the shepherd cache. Physically this is accomplished by combining these replacements into one by replacing the line R with M and adjusting the auxiliary structures to indicate that (1) if O is not self-replaced, it now belongs to MC, (2) M belongs to SC and (3) *SC-ptr* of M points to the column (in CM) previously used by O.

In addition, the following modifications are made to the CM. The empty flag of all entries along M's newly assigned column are set. All values along M's row are set to the dummy value (0) and their empty flags are reset. By doing so the new SC line is artificially made the most imminent line with respect to the older SC lines. Note that now there may be more than one line with value 0, but no information is lost with regard to the least imminent line[2]. Figure 6 illustrates these changes that take place on a miss.

Further, if O has moved into the MC (not self-replaced), it needs to be tracked by the baseline MC replacement policy. It is to be noted that O has already been through an SC and is not necessarily moved into the MC on an access to it. Therefore, if the baseline replacement policy is LRU, O is conservatively assumed to be the Least Recently Used with respect to previously present MC lines. If the baseline MC replacement policy is LFU, O

[2]The least imminent line has either empty flag set or a non-zero value as long as MC-associativity>1

is assumed to start with a frequency count of zero.

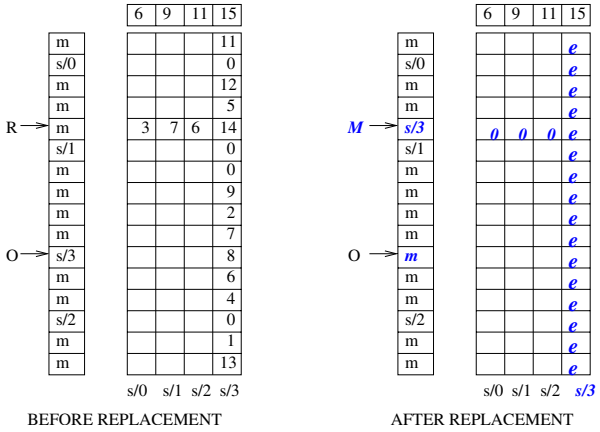


Figure 6: Update on a cache miss. Only relevant CM values are shown. m and s indicates MC and SC respectively. Modified entries are in blue and italicized.

3.5 Making a Replacement Decision

In order to accommodate a new entry M, a replacement decision needs to be made based on the oldest SC candidate O (O can be identified using the baseline FIFO of SC). The column corresponding to O is traversed to pick out a replacement victim. If the empty flag is set for some lines in the column, one among them is chosen based on the baseline replacement policy as the replacement victim R. Otherwise the line with the maximum counter value is chosen for replacement. Note that the replacement victim is implicitly restricted to be one among the lines in MC or O itself as the remaining newer SC entries would have the lowest possible counter value (zero) and empty flag reset. The replacement choices are therefore restricted to the choices that OPT would consider. Figure 6 illustrates the changes.

3.6 Replacement Complexity

L2 misses have high miss penalties because the time for memory to return the missing data is in the order of 400 cycles. Replacement decisions are hence off the processors' critical path and can tolerate some complexity Further, we expect that the complexity of our replacement scheme to be not very different from the commonly used LRU replacement policy. To justify this claim, we estimate replacement complexity in terms of the number of bits that need to be accessed to make a replacement decision. The estimates are made for an (SC-4,MC-12) cache assuming simple naive implementations.

For each miss the following needs to be done. (1) Find the oldest SC entry. This conservatively requires access to four 2 bit values ($4 \times 2 = 8$ bits). (2) Find entries along the corresponding CM column with their empty flag set, if any. This requires access to 16 empty flags. (3a) If empty entries exist, apply the baseline replacement policy and pick a replacement victim among lines corresponding to these empty entries. In the worst case, this requires access to counter values of all 12 MC lines ($12 \times 4 = 48$ bits). or (3b) If there is no entry with empty flag set, choose the entry with the largest value for replacement. This requires access to $16 \times 4 = 64$ bits. Overall this amounts to access of 72 to 88 bits. In comparison a naive implementation of LRU-16 would require access to $16 \times 4 = 64$ bits.

4 Performance Evaluation

4.1 Methodology and Configurations

We report the performance results both in terms of MPKI and speedup in CPI. The evaluation is done for *ref* inputs of the entire SPEC2000 benchmark suite. The performance is evaluated over 3 billion instructions after forwarding the first 2 billion instructions. A 3 billion instruction interval is chosen so that all benchmarks incur at least 10 million L2 accesses. The MPKI is measured using trace driven simulation while CPI is measured using simplescalar-3.0 [3]. The processor configuration used in our simulations is indicated in Table 1.

Table 1: processor configuration

Parameter	Value
CPU	8 wide decode/commit, out-of-order 8 wide issue, speculative execution
ROB, LSQ	64 entry ROB, 32 entry LSQ
Branch Predictor	12KB combining Branch predictor, bimodal + Gshare, 10 cycle penalty
L2 cache	unified 512KB,1MB,2MB or 4MB 128B Block, 16 way, 8 cycle latency
Write buffer	Infinite write buffer
L1-I and L1 D cache	16KB 2 way, 32B Block, 2 cycle latency
Memory	Infinite size, 400 cycles

Table 2: Configurations for alternative schemes

Scheme	Parameters
DIP	16/32/64 dedicated sets, epsilon $1/32$
V-way cache	32 tags per set, tdr 2 global reuse distance replacement
Victim Buffer	MC+VB, VB same assoc as SC
fa cache	fully associative with LRU
OPT-16	16 way associative OPT replacement

We evaluate performance for various SC+MC configurations by keeping the total associativity of the set to

16 and varying the SC-associativity from 2 to 14. MC-associativity varies correspondingly from 14 to 2. These schemes are referred to by their SC-associativity as SC-2 to SC-14 respectively. For example, SC-4 refers to an organization where in each set SC-associativity is 4 and MC-associativity is 12. Note that the amount of lookahead provided by the different schemes increases with increase in SC-associativity. At the same time the size of the main cache for which OPT is emulated decreases with increase in SC-associativity.

In addition to LRU, we compare the performance (in terms of MPKI) with two recent L2 replacement strategies, the Dynamic Insertion Policy (DIP) [11] and the V-way cache [10]. We also compare the performance of a Victim Buffer with an identical configuration as our Shepherd Cache. Further, we compare performance with two idealistic caching alternatives, a fully associative cache with LRU replacement and a 16 way associative cache with OPT replacement. The configurations of all the above schemes are indicated in Table 2. For DIP we vary the number of dedicated sets and report the performance at the configuration that gives the best MPKI reduction for a given cache size.

4.2 SC-associativity vs MC-associativity

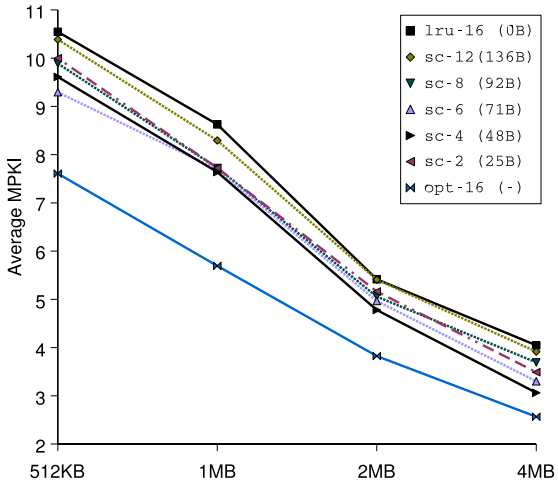


Figure 7: SC-assoc vs MC-assoc. In the legend we indicate the number of additional bytes per set for the auxiliary structures (CM,NVA,SC-flag,SC-ptr)

The average MPKI over all 26 benchmarks for 5 representative SC based schemes and their variation with changing cache sizes is plotted in Figure 7. For the sake of clarity we only plot the results for SC-2, SC-4, SC-6, SC-8 and SC-12. It can be seen that all the SC based schemes perform better than a 16 way associative LRU. For configurations with SC associativity greater than 8 the performance improvement over LRU is marginal (we plot the results of SC-12 as representative of these configurations). This is expected as for these configurations the MC for which OPT is emulated is of less than half the cache size. In general the performance improves gradually from SC-12 to SC-6. From SC-6 to SC-4 the performance improvement is marginal. Performance then degrades as we move further towards SC-2. The above trend indicates the trade-off between the amount of lookahead and size of the MC for which OPT is emulated. From Figure 7 it can be inferred that between half to three quarters of the cache should be dedicated to MC and the remaining should be used for lookahead. It can also observed that the hardware overhead of the SC schemes (shown along the legend) increases with increase in SC-associativity. However, the overhead is marginal (less than 4.5%) for SC-associativities of interest (SC-8 and lower).

4.3 Bridging the gap between LRU and OPT

Figure 7 also plots the MPKI achieved by 16 way LRU and OPT for various cache sizes. It can be seen that SC-4 successfully bridges the gap between LRU and OPT by a significant fraction. With increase in cache size SC-4 comes closer to OPT-16. SC-4 narrows down the gap by 31.8% at 512KB, 33.7% at 1MB, 40% at 2MB and 51.9% at 4MB. SC-6 performs comparable to SC-4. SC-2 and SC-8 narrow the gap by 19% to 38%.

4.4 Comparison with alternate proposals

We compare the performance of SC based schemes with a baseline set associative LRU cache and 5 other schemes listed in Table 2. Among these the base LRU cache, VB and DIP have very little or no additional hardware overheads. In order to make a fair comparison between SC based schemes and these low overhead schemes we add one additional line (128B) per set for these schemes[3]. We refer to these schemes as LRU-17, VB-best-17 and DIP-17 respectively. We do not add any additional lines to the V-way cache as it also has comparable overheads - maintaining forward and reverse pointers between the tag array and the data array. For the main cache with victim buffer configuration, we evaluated performance at three different VB associativities 4,8, and 12, keeping the total associativity constant at 17. We report the best performing among these configurations (VB-best-17).

The MPKI achieved by different schemes is measured for cache size corresponding to SC+MC at 512KB, 1MB, 2MB and 4MB. The results for two sample cache sizes

[3]For example, a 16 way associative, 512KB, SC+MC is compared with a Dynamic Insertion Policy based cache having the same number of sets (256) but with one additional line per set (total of 544KB).

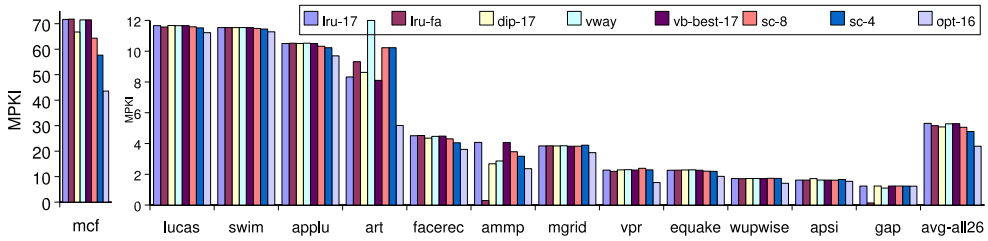


Figure 8: MPKI for a 2MB cache. Benchmarks with MPKI < 1 omitted, avg-all26 is over all 26 benchmarks.

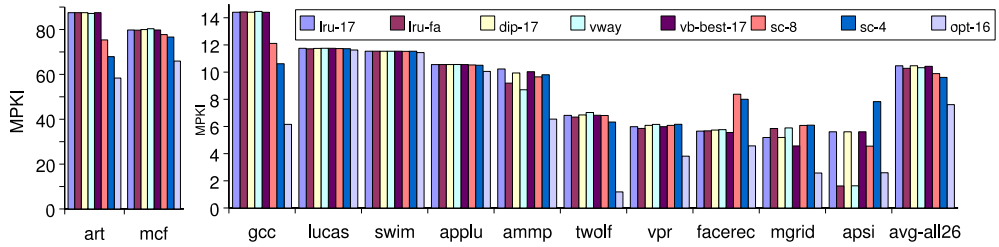


Figure 9: MPKI for a 512KB cache. High MPKI benchmarks, avg-all26 is over all 26 benchmarks.

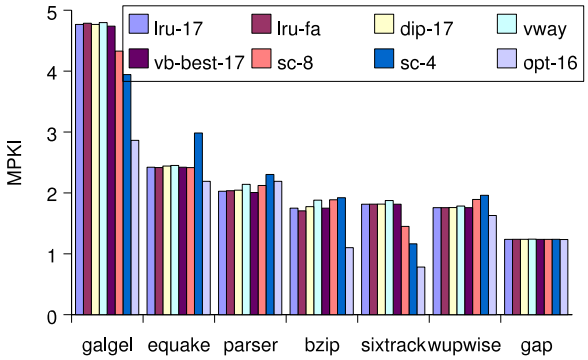


Figure 10: MPKI for a 512KB cache. Low MPKI benchmarks, benchmarks with MPKI < 1 omitted.

namely 512KB and 2MB for 2 sample SC schemes (SC-4 and SC-8) are reported in Figures 8-10. The benchmarks are categorized into two or three MPKI ranges and each range is plotted separately. Benchmarks with very low MPKI (< 1) are omitted. The avg-all26 bar indicates the MPKI average over all 26 benchmarks and not any single category. It can be observed that both SC-4 and SC-8 perform better than all other schemes for most of the benchmarks. On average a 2MB SC-4 reduces MPKI over LRU-17 by 10%. The corresponding improvements over VB-best-17, DIP-17 and vway are 9.3%, 5.9% and 9.6% respectively. For SC-8 the performance difference with respect to the other schemes ranges from 1% to 4.6%. It is interesting to note that LRU-fa, though performs exceptionally well for a few configurations (ammp, apsi at 512KB and ammp, gap at 2MB), performs worse than SC based schemes for most benchmarks. A V-way cache tries to emulate a fully associative LRU cache and it is not surprising that a V-way cache does not perform comparable to DIP and SC based proposals.

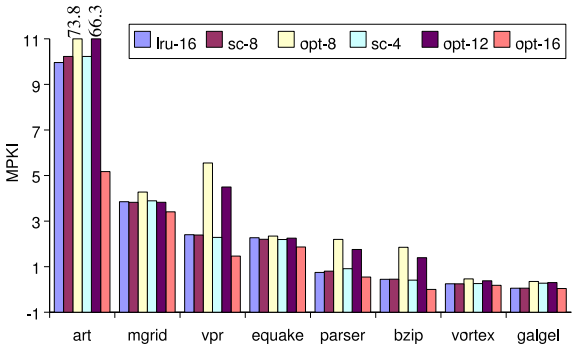
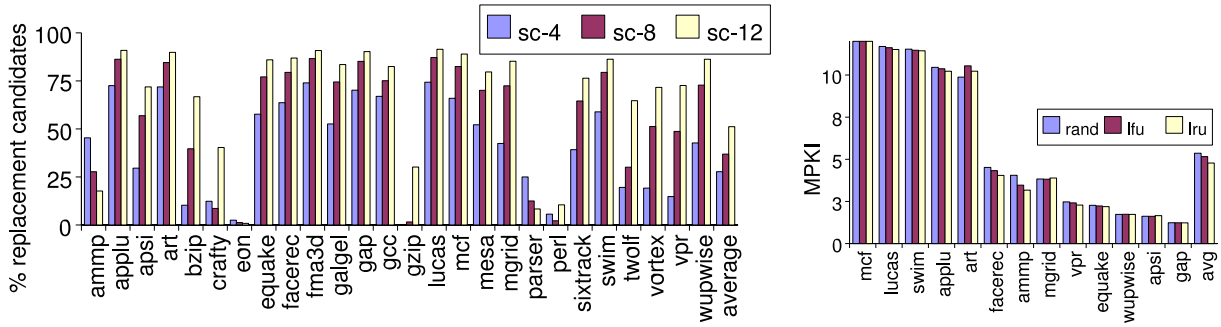


Figure 11: Correlating MKPI drop with OPT

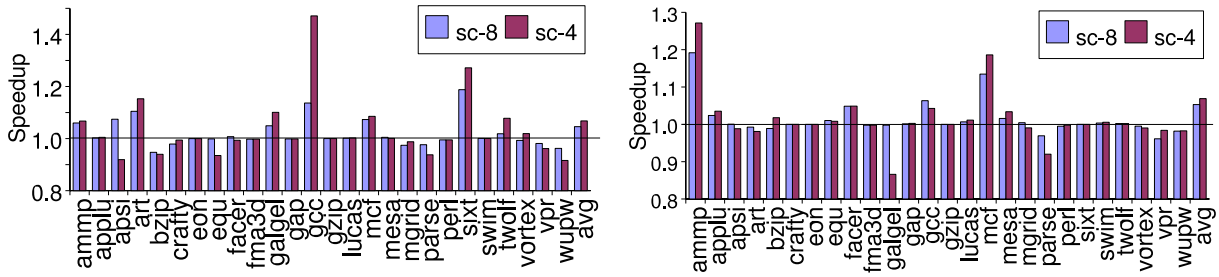
Finally, it can also be noted from that although for a majority of the benchmarks SC performs well, there are some benchmarks for which SC performs poorly. To understand the cause for this, we analyze the benchmarks for which SC-8 or SC-4 degrade performance. Specifically, we examine the performance of OPT at sizes and associativities that these schemes try to emulate (refer to Figure 11). All these configurations have the same number of sets (1024) but their associativities are different (varied from 8 to 16). It is to be remembered that SC-8 emulates OPT-8 and SC-4 emulates OPT-12. As can be seen from Figure 11, there is a strong correlation between SC's performance with the corresponding OPTs performance. Whenever, SC performs poorly the corresponding OPT also performs poorly. However, because of the additional hits obtained while the optimal order is established, SC mitigates the performance loss to some extent. This is evident in *art, mgrid, vpr, parser and bzip*.



(a) Percentage of replacement candidates for baseline MC replacement

(b) Comparing MC replacement policies

Figure 12: Impact of baseline MC replacement policy



(a) 512KB SC compared to LRU-17

(b) 2MB SC compared to LRU-17

Figure 13: Speedup over LRU-17

4.5 Effect of baseline MC replacement

The baseline MC replacement policy is applied only on the fraction of MC lines that are not covered by the lookahead window. Its influence therefore depends on the lookahead window provided by the SC. In Figure 12(a) we plot the fraction of the MC candidates that do not get covered by the lookahead for a 2MB cache with SC-4, SC-8 and SC-12. As expected higher the SC associativity, fewer are the replacement candidates left for the baseline MC policy. On average SC-12 leaves only 0.96 (out of 4) candidates for MC replacement, SC-8 leaves 3.62 (out of 8) and SC-4 6.81 (out of 12).

We compare 3 replacement strategies for MC; LRU, LFU and random. The performance of these three policies is compared in Figure 12(b) for SC-4. It can be seen that LRU performs better than LFU and random replacement policies. With the exception of *art, galgel, parser* LRU consistently outperforms the other policies.

4.6 Impact on System Performance

Figure 13 plots the speedup obtained when using SC-4 and SC-8 instead of LRU-17. The speedup is calculated as the ratio of the CPI of the LRU-17 to the CPI of the SC scheme in consideration. The average bar indicates the speedup in average CPI. SC-4 obtains a speedup of up to 47.1% (gcc 512KB) and an average speedup of 7% for a 512KB cache. In comparison SC-8 obtains an average speedup of 5.1%. The corresponding numbers for a 2MB cache are 7.1% and 5.3% respectively.

5 Related Work

The observation that a large number of L2 lines lack temporal locality has motivated several research endeavors. Modifications to the LRU replacement policy [14] have been proposed to ensure that lines that do not have temporal locality are not brought into the cache at all. A new reuse distance based global replacement policy is proposed for the V-Way cache [10] based on the observation that a large number of lines brought into the L2 cache are never accessed again. Further to facilitate variable set associativity the tag and data portion are decoupled. Quantitative comparison reveals that SC-4 reduces the MPKI obtained by a 2MB 32 way associative, tdr 2, V-way cache by 9.6%. Like th V-way cache, a few other schemes [4, 1, 6] also propose decoupling of tag

and data entries to leverage performance.

The dynamic insertion policy [11] tries to prevent lines that do not exhibit temporal locality from staying in the cache for long. This is done by dynamically adapting the position in the LRU stack at which a line is inserted. Quantitative comparison with DIP reveals that a 2MB SC-4 reduces MPKI by 5.9% over a comparable sized DIP. A victim buffer [7] originally proposed to assist direct mapped L1 caches, can be effective for an L2 cache too as it provides a second chance to replacement victims prematurely evicted due to a bad replacement decision. Conceptually a victim buffer is like an SC placed after the L2 cache instead of before it. However, a victim buffer differs from an SC as it allows the base replacement scheme to make a potentially bad replacement choice. It then reactively pushes back the poorly chosen replacement victims into the main cache. In comparison, an SC is proactive and directly aids in taking replacement decisions. Performance comparison with a victim buffer reveals that SC-4 at 2MB achieves 9.3% lesser MPKI than a cache with a victim buffer. The recently proposed adaptive replacement policy [12] provides support for multiple replacement policies and dynamically picks the best performing policy on a per set basis. They conclude that a combination of LRU and LFU performs fairly well for an L2 cache. Such a mechanism could be employed to adapt the lookahead dynamically and prevent the loss of performance that SC based schemes suffer for a few benchmarks. We plan to evaluate this as future work. Finally, an interesting analytical model is proposed [5] that can analyze different LRU stack based replacement policies.

Our proposal has some similarity with the way OPT replacement is simulated [13] by an offline trace driven simulator. The simulator typically uses a lookahead window to observe future accesses and pick out the optimal replacement candidate. An approximate decision is made if the optimal candidate is beyond the lookahead window and the decision is corrected later on if required. Further, the simulation of multiple OPT configurations makes use of the fact that OPT obeys the stack property [9]. That is, caches with the same number of sets but higher associativity are guaranteed to contain all the lines that a smaller associative cache would contain at any point in time. This stack property of OPT partly explains why emulating an MC at higher associativity is almost always more beneficial. This property also implies that in theory if the SC can provide sufficient lookahead to always pick out the optimal replacement candidate then emulating OPT for a larger associativity is guaranteed to provide more benefits.

6 Conclusions

As the inherent temporal locality in memory accesses is filtered out by the L1 cache, an L2 cache with LRU incurs significantly higher misses than the optimal replacement policy (OPT). In this paper we close down this gap through a novel replacement strategy that mimics the replacement decisions of OPT. Our proposed organization can bridge 40% of the gap between OPT and LRU for a 2MB cache leading to a 7% overall speedup. Comparison with the *dynamic insertion policy*, *victim buffer*, *V-Way cache* and an LRU based *fully associative cache* demonstrates that our scheme performs better than all these strategies.

Acknowledgments

We would like to thank Prof S. Kaxiras for shepherding our paper. We also thank the anonymous reviewers and members of the laboratory of high performance computing for their useful comments and suggestions. The first author acknowledges the scholarship and travel grant from Microsoft Research India.

References

[1] A. Alameldeen and D. Wood. Adaptive Cache Compression for High-Performance Processors. In *Proc. of Int. Symp. Computer Architecture*, ISCA 2004.

[2] L Belady. A study of Replacement Algorithms for a Virtual-storage Computer. *IBM Systems Journal*, 1966.

[3] D. Burger and T. M. Austin. The SimpleScalar tool set:Version 2.0, *Tech. Report, Dept. of CS, Univ. of Wisconsin-Madison, June 1997 and documentation for all Simplescalar releases (through version 3.0).*

[4] Z. Chishti, M.D. Powell and T.N.Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. In *Proc. of Intl. Symp. on Microarchitecture*, Micro 2003.

[5] F. Guo and Y. Solihin. An Analytical Model for Cahce Replacement Policy Performance.In *Proc. of Intl. Conf. on Measurement and Modeling of Computer Systems*, SIGMETRICS 2006.

[6] E. Hallnor and S. Reinhardt. A compressed memory hierarchy using an indirect index cache. *Proceedings of the 3rd workshop on Memory performance issues: in conjunction with the 31st Int. Symp. on computer architecture*, 2004.

[7] N.P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of Intl. Symp. on Computer Architecture*, ISCA 1990.

[8] M.Kharbutli, K.Irwin, Y.Solilin, J.Lee. Using prime numbers for cache indexing to eliminate conflict misses. In *Proc. of IEEE Int. Symp. on High Performance Computer Architecture*, HPCA 2004.

[9] R Mattson et. al. Evaluation Techniques for Storage Heirarchies. *IBM Systems Journal* 1970.

[10] M.K. Qureshi, D. Thompson, Y.N. Patt. The V-Way Cache: Demand Based Associativity via Global Replacement. In *Proc. of Int. Symp. Computer Architecture*, ISCA 2005.

[11] M.K. Qureshi et. al. Adaptive Insertion Policies for High Performance Computing In *Proc. of Int. Symp. Computer Architecture*, ISCA 2007.

[12] R. Subramanian, Y. Smaragdakis and G. Loh. Adaptive Caches: Effective Shaping of Cache Behavior to Workloads. In *Proc. of Intl. Symp. on Microarchitecture*, Micro 2006.

[13] R. Sugumar and S. Abraham. Efficient simulation of caches under optimal replacement with applications to miss characterization. In *Proc. of Intl. Conf. on Measurement and Modeling of Computer Systems*, SIGMETRICS 1993.

[14] Wayne A. Wong and Jean-Loup Baer: Modified LRU Policies for Improving Second-Level Cache Behavior. In *Proc. of Int. Symp. High Performance Comp. Arch.*, HPCA 2000.