CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A Fine-Grained Thread-Aware Management Policy for Shared Caches

Dyer Rolán [1], Diego Andrade [2*] Basilio B. Fraguela[2] and Ramón Doallo[2]

[1]*Intel Labs Barcelona, Barcelona, Spain*
[2]*Departamento de Electrónica e Sistemas, Universidade da Coruña, A Coruña, Spain*

SUMMARY

Two of the main sources of inefficiency in current caches are the non-uniform distribution of the memory accesses across the cache sets, which causes misses due to the mapping restrictions of non fully-associative caches, and the access patterns with little locality that degrade the performance of caches under the traditional LRU replacement policy. This paper proposes a technique to tackle in a coordinated way both kinds of problems in the context of Chip Multiprocessors, whose Last Level Caches can be shared by threads with different patterns of locality. Our proposal, called Thread-Aware Mapping and Replacement Miss Reduction (TAMR2) policy, tracks the behavior of each thread in each set in order to decide the appropriate combination of policies to deal with these problems. Despite its small overhead, TAMR2 achieved in our experiments average power consumption and memory latency reductions of 10% and 12% respectively, resulting in an average throughput improvement of 5.6%, relative to a traditional cache design using 4 cores. TAMR2 also outperformed many recent related approaches in the field.
Copyright © 2013 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Thread-awareness, coordination, policies, last-level cache memories

1. INTRODUCTION

The optimization of last-level caches (LLC), which are responsible for avoiding the costly off-chip memory accesses, is a key point in the design of current computers. One of the behaviors observed in caches is the lack of uniformity in the distribution of the memory references among their sets. This way, some sets can exhibit large local miss ratios because they do not have the number of lines required by their working set, while other sets achieve good local miss ratios at the expense of a poor usage of their lines, because some or many of them are actually not needed to keep the working set. This gives place to misses, known as mapping misses [1], which happen due to a limited associativity in the cache. A number of different approaches such as victim caches [2], the adaption of the assignment of lines to sets [3] or displacements of lines between sets [4] primarily deal with this situation.

Other behaviors that stress caches are related to their temporal management, giving place to replacement misses [1]. This is the case of thrashing, which happens when a working set larger than the cache [5] evicts useful lines due to the LRU replacement policy. Mixed access patterns in which frequently accessed items are discarded from the cache due to interleaved bursts of references without temporal locality, also known as scans, also fit in this category. The proposals to deal with these problems modify in some way the replacement policy of the cache [6][7][8].

*Correspondence to: Fac. de Informática, Campus de Elviña, A Coruña, Spain. E-mail: diego.andrade@udc.es

In general, since cache management policies that mostly target either mapping or replacement misses deal with different problems, they resort to very different mechanisms both to regulate their internal behavior as well as to detect the problems they target. This makes difficult the desirable coordinated implementation of these policies in a cache. The problem of the selection of the most suitable policy and its application to the appropriate portions of the cache is further complicated in the context of Chip Multiprocessors (CMPs), as they often have LLCs shared by several cores.

This paper explores the problem of the coordinated reduction of mapping and replacement misses in shared caches. Our approach, called Thread-Aware Mapping and Replacement Miss Reduction (TAMR2), detects the degree to which each thread can suffer problems of unbalance among cache sets or inadequate temporal management in each cache set independently. Then, TAMR2 applies an appropriate combination of policies for mapping or replacement miss reduction to the lines of each thread in each set, being therefore a very fine-grain thread-aware technique. TAMR2 takes some decisions based on the joint behavior of all the applications that share the cache, while other decisions are specific to the behavior of each application. Still, these latter decisions do not consider the application in isolation, but rather take also into account how the other applications impact on the behavior of the analyzed one. As for the policies it applies, TAMR2 deals with problems of load unbalance among sets using the Set Balancing Cache [4], which displaces lines from oversubscribed sets to underutilized ones. This policy was chosen because of its reduced cost and its ability to be applied with fine granularity only in some sets and only to some threads if desired. As for replacement misses, TAMR2 has been tested with two suitable techniques that also fulfill these properties, the Bimodal Insertion Policy (BIP) [6], and the Re-Reference Interval Prediction [7].

The rest of this paper is organized as follows. The next section reviews the related work and motivates further our approach, which is described in Section 3. The environment for its evaluation is described in Section 4 while comparison with other techniques and experimental results are discussed in Section 5. Then, Section 6 gives our conclusions and future work.

2. BACKGROUND AND MOTIVATION

Contrary to the traditional 3C classification of the cache misses, which ignores the replacement policy [1, 9] in the case of conflict misses, the OPT model [1] considers it. As a result, it distinguishes two different types of conflict misses: mapping misses, which occur as a result of the restrictions imposed by the set mapping strategy in non fully associative caches, and replacement misses, which occur as a result of sub-optimal replacement inside a set.

In the past years several approaches that mostly target mapping misses have appeared. To name a few, [10] proposes an alternative indexing function that achieves a more uniform distribution of the working set on the cache than the standard one. The V-Way cache [3] adapts to the non-uniform distribution of the accesses on the cache sets by allowing different cache sets to have a different number of lines according to their demand. The Set Balancing Cache [4] tracks the degree of pressure on each set and displaces lines from stressed sets to underutilized ones. This idea is also explored in [11] using dead block prediction instead of identifying high and low pressured sets.

The problems that appear when the working set of an application is larger than the cache, or when it suffers from frequent bursts with no temporal locality (scans) that expel from the cache frequently used working sets, are qualitatively related to replacement misses. These problems have been mainly addressed by changes in the insertion and replacement policies. Some examples of this approach are the Bimodal Insertion Policy (BIP) [6], the Pseudo-LIFO policies [12] and the Dynamic Re-Reference Interval Prediction (DRRIP) replacement policy [7]. Finally, Signature-based Hit Predictor (SHiP) [8] correlates the re-reference behavior of a cache line with a unique signature so that it can apply replacement policies with a finer granularity.

In the case of shared caches management, we can distinguish between self and inter mapping and replacement misses. Self misses would be the ones that take place even if the associated application had all the cache for itself, while the inter-mapping and inter-replacement misses would be the new misses of the respective kind that appear in the presence of other applications, considering together all their working sets. This way, Adaptive Set Pinning (ASP) [13] detects inter-mapping misses

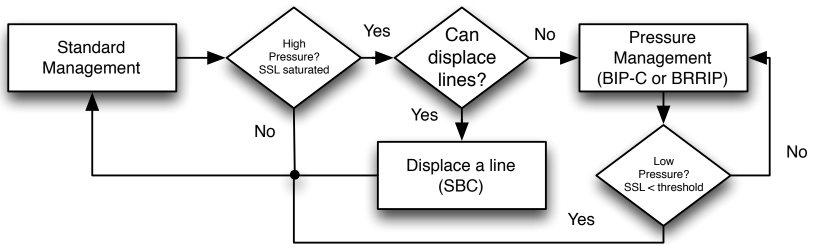
Standard Management
High Pressure? SSL saturated
Yes
Can displace lines?
No
Pressure Management (BIP-C or BRRIP)
No
Yes
Displace a line (SBC)
Low Pressure? SSL < threshold
No
Yes

Figure 1. High level view of TAMR2

in shared caches and addresses them specifically. A proposal oriented to self and inter-replacement misses is the Thread-Aware Dynamic Insertion Policy (TADIP) [14], which extends to shared caches the insertion policies introduced in [6] to deal with replacement misses in private caches. TADIP can apply these policies in isolation to each independent thread according to the benefit it can get from them, measured by means of set dueling. The thread-aware TA-DRRIP [7] also chooses the most appropriate Re-Reference Interval Prediction (RRIP) insertion policy for each application based on set dueling, RRIP being mainly focused on replacement misses. Furthermore, there are approaches like Promotion/Insertion Pseudo-Partitioning (PIPP) [15], which combines pseudo-partitioning with new insertion and promotion policies.

As we see, there are no proposals to the best of our knowledge that tackle specifically both the self and inter-mapping misses as well as the self and inter-replacement misses we identify in shared caches. Also, most of the proposals that target replacement problems in shared caches take their decisions globally, based on an average picture of the cache behavior obtained either from set dueling [6] or set sampling [16]. They also apply those decisions uniformly in all the cache sets.

3. THREAD-AWARE MAPPING AND REPLACEMENT MISS REDUCTION

As Section 1 explained, our Thread-Aware Mapping and Replacement Miss Reduction (TAMR2) proposal addresses in a coordinated way the mapping and the replacement misses commonly found in caches. This requires three components: a mechanism to track the status of the cache, a policy to tackle mapping misses, and a strategy to deal with replacement misses. We now describe at high level these components and their interaction, which is graphically depicted in Figure 1.

Since mapping misses appear due to unbalanced loads in different cache sets, the tracking mechanism needs to be implemented at the set level. Due to its low cost, fine granularity and accuracy we have chosen for this first component the Set Saturation Level (SSL) metric [4]. This is the value of a counter with saturating arithmetic that is increased each time an access to the set results in a miss, and decreased with every hit. A SSL counter works in the range 0 to $2 * K - 1$, K being the associativity of the cache henceforth. TAMR2 uses a SSL counter per set and per core that is updated only by the accesses from that core. This allows TAMR2 to take independent decisions for each thread in each cache set. A high SSL indicates the thread is stressed in the set. On the contrary, low SSLs of all the threads in a set hint the availability of space in the set. This way, when TAMR2 identifies two sets in these opposite situations, it tries to reduce mapping misses by applying the Set Balancing Cache (SBC) [4]. This technique, described in Section 3.1, balances the load of the sets by displacing lines from stressed sets to underutilized ones.

If a SSL is saturated but there are no sets with low SSLs that are not already helping to reduce the pressure of stressed sets, TAMR2 cannot apply SBC. Since the cache is under stress and SBC does not suffice to alleviate it, TAMR2 assumes that at least part of the problem must be related to the cache temporal management. Thus, when this happens, a strategy to reduce replacement misses is applied to the lines of the thread whose SSL got saturated. In this situation the thread changes in the set from the standard management (SM) mode, which follows the usual insertion and replacement policies, to pressure management (PM), which applies policies to reduce replacement misses. The

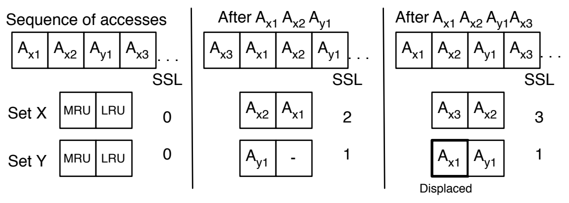
Sequence of accesses
After A_{x1} A_{x2} A_{y1}
After A_{x1} A_{x2} A_{y1} A_{x3}
A_{x1} A_{x2} A_{y1} A_{x3} . . .
A_{x3} A_{x1} A_{x2} A_{y1} . . .
SSL
Set X
Set Y
MRU
LRU
0
2
3
1
Displaced

Figure 2. Set Balancing Cache example in a 2-way cache.

thread returns to standard management (SM) when its SSL falls below K, as this indicates the thread is no longer stressed in the set. Section 3.2 describes the temporal management strategies used by TAMR2, followed by a discussion on the coordination and interaction of these policies with SBC under TAMR2 in Section 3.3.

3.1. Unbalances Among Sets: Mapping Misses

The Set Balancing Cache (SBC) [4] associates sets with high SSL with sets with low SSL that will be used to store part of the working set of the former ones. Concretely, when the SSL of a set is saturated, SBC associates it with the set with the lowest SSL that is not currently involved in any association, provided that its SSL is smaller than K. This latter condition avoids associating sets without enough unbalance between them. All subsequently evicted lines from the set with the high SSL of an association (source set) are displaced to the set with the low SSL (destination set) rather than to memory, as long as the SSL of the source set has its maximum value when the eviction takes place. That is, if the SSL of the source set is $2 * K - 1$, its LRU line is displaced to the destination set, where it is inserted in the MRU position; otherwise the line is evicted from the cache. Misses in source sets of associations lead to searches in their destination set, giving place to secondary hits and misses. Thus, this technique needs an additional bit in every tag-store entry to identify displaced lines. Finally, associations are broken when the destination set evicts all the displaced lines.

Figure 2 illustrates how SBC works in a 2-way associative cache in which sets X and Y are depicted. The cache cyclically receives the sequence of accesses to different lines $A_{x1}A_{x2}A_{y1}A_{x3}$ where the A_x ones are mapped to set X an the A_y ones to set Y. Since the sequence has 3 lines mapped to the same set and there are 2 ways, in a standard cache all the A_x accesses result in misses, while set Y keeps a single line. In a SBC the first three accesses operate in the standard way, just increasing the SSLs, resulting in the middle illustration. The access to A_{x3} saturates the SSL of set X, which becomes associated to set Y thanks to its low SSL, smaller than the associativity. The stressed set then displaces its LRU line, A_{x1} to the MRU position of set Y, freeing up the line for A_{x3}, which becomes the MRU line in set X. In the resulting cache, shown in the last illustration in Figure 2, all the subsequent cycles of access always result in hits. In particular, all accesses are first hits except A_{x1}, which is found in a second access in set Y after the initial search in set X fails.

SBC is not thread-aware and thus it uses a single SSL per set. In TAMR2 a SSL per core and per set determines the state of each thread in each set and the saturation of any SSL indicates the need to perform displacements. However, a set should only receive displaced lines if none of the threads is stressed in the set. This way in TAMR2 the fact whether a set can be a destination set or not is based on a single SSL value per set called Global SSL (GSSL), which is computed every time a SSL is updated. The GSSL does not need to be stored in the set; it is just supplied to the Destination Set Selector (DSS) [4], a low-cost structure that tracks the sets with the lowest SSL in the cache that are not involved in any association in order to provide the best one when an association request is issued. The GSSL and the corresponding set index are stored in the DSS only if the GSSL is smaller than the maximum value in the selector, and it is also below the saturation limit K (the cache associativity) allowed for a set to be candidate to become a destination set.

The GSSL can be computed by simply adding the SSLs and saturating the outcome to the maximum value of a single SSL. The GSSL can be also tuned taking into account the state of each application. Applications in PM mode always have a SSL$\geq K$, since they return to SM when the SSL falls below K, as this indicates that the working set fits in the set. This way, if their SSLs

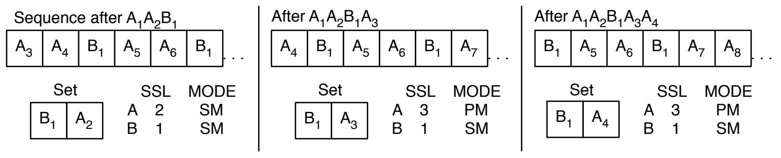

Figure 3. TAMR2 management of replacement misses in a 2-way cache shared by two cores using BIP.

are added up directly, these applications preclude GSSLs smaller than K, making impossible for the corresponding sets to become destination sets. Still, the high SSL of an application in PM mode does not mean that it is effectively using many lines in the set. Rather, it means that there is high recent miss rate, probably because the application is memory-intensive. As as result it is likely that many of its lines are of little use and it would be better to give them to another application. Of course this can also happen with applications in SM mode, but the chances are higher among those in PM mode. Thus, in our experiments the SSL of applications in PM mode is scaled down for the sake of the computation of the GSSL by dividing them by 2, since this can be implemented in hardware with a simple shift of one bit. This minor change improved the throughput in our tests around 0.25%.

3.2. *Suboptimal temporal management inside cache sets: Replacement Misses*

As pointed out in Section 2, there are several proposals that target replacement misses. We have evaluated TAMR2 with two simple and effective techniques. The first one is the Bimodal Insertion Policy (BIP) [6], which is applied in caches with LRU replacement policy. BIP inserts most lines in the LRU position of the recency stack and, only with a small probability, ε, in the MRU position. This way, most lines require a second access to be promoted to the MRU position. As a result, lines that are dead on fill cannot trash the cache, and are instead evicted when a new line is inserted in the set. Therefore, the number of replacement misses is reduced by keeping those lines with more locality in the cache set while discarding temporary data as soon as possible. Nevertheless, a plain BIP policy may lead to some performance degradation in shared caches, so we have used a modified version that will be further described in the next Section. The complementary technique to BIP in TAMR2, that is, the one to apply when the thread is in standard management (SM) mode, is the standard MRU insertion policy.

The second policy evaluated is Bimodal Re-Reference Interval Prediction (BRRIP) [7]. BRRIP works similarly to BIP in a cache that applies Re-reference Interval Prediction (RRIP), a replacement policy in which each line has an associated Re-reference Prediction Value (RRPV), so that under a miss, a line with the longest predicted re-reference interval is evicted. While Static RRIP inserts all lines with some intermediate re-reference interval prediction, Bimodal RRIP predicts a long re-reference interval for most lines, and an intermediate interval only for a small number of the insertions. This way, in the context of TAMR2, threads in SM mode apply SRRIP, while sets under pressure management (PM) apply BRRIP.

Figure 3 illustrates the temporal management or TAMR2 in a set of a 2-way cache shared by two cores, A and B, using BIP. In this example thread A has poor locality while thread B periodically accesses a single line B_1 in the set, interleaved with the accesses of A. This gives place to a sequence of the form $A_1A_2B_1A_3A_4B_1\ldots$. The first illustration in the figure shows the situation after the first three accesses assuming that the SSLs were initially 0. The miss on A_3 saturates A's SSL. Thus if TAMR2 cannot find an underutilized set to which to displace lines from this set, A enters pressure management in the set, which results in A_3 being inserted in the LRU position of the set under BIP. The illustration in the middle shows the outcome. The access to A_4 is another miss, but A's SSL is already saturated and we continue applying BIP, giving place to the last state shown in Figure 3. We can see that thanks to the protection that this management has provided to B_1, its upcoming access will hit in the cache. In a standard cache, however, B_1 would not be protected and its accesses would systematically result in misses.

Tracking the status of each thread in each set by means of a SSL allows TAMR2 to decide with a granularity of thread and set whether to apply the standard or the pressure management.

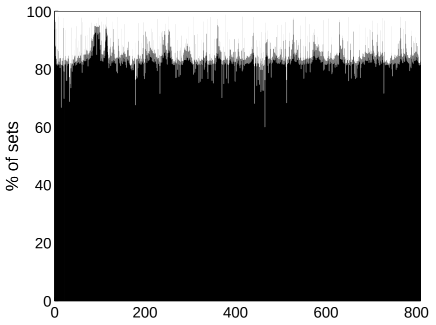

Figure 4. Distribution of the sets with a high (black), medium (gray) and low (white) SSL for *433.milc* during its simultaneous execution with *482.sphinx3* in a 8-way 2MB cache. Samples each $5 * 10^5$K accesses.

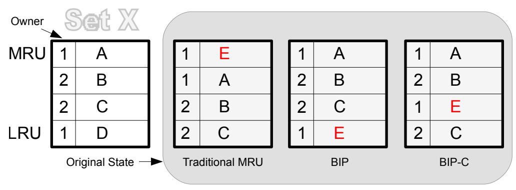

Figure 5. Behavior of different insertion policies after inserting new line *E* in set *X* of a 4-way LLC shared by *2* cores.

The importance of thread-awareness for the policies that deal with replacement misses is already known [14][15][7], as applications with good locality will be hurt by the policies applied under pressure management. However, while all the works we know of uniformly apply the same policy to all the lines of each given application in all the cache sets, TAMR2 can apply different policies to the same thread in different sets. This finer granularity is useful because some sets may require policies to avoid replacement misses while others do not, being thus counterproductive to uniformly apply the same policy in all the sets.

This latter observation is exemplified in Figure 4 with a situation we have seen in many parallel executions. It classifies in three categories the sets of an 8-way 2MB L2 cache shared by SPEC CPU 2006 applications 433.milc and 482.sphinx3, during the execution of 10 billion instructions of 433.milc and taking samples every $5 * 10^5$ accesses to the cache. The sets are classified according to the value of a SSL which is only updated by the accesses by benchmark 433.milc to the set. Concretely, each SSL is classified as low (0-5), medium (6-10) or high (11-15). Another SSL updated in each set only by 482.sphinx3, not shown, is uniformly saturated in all the sets along this simultaneous execution. We see there is a representative and sustained 20% to 25% of sets in which 433.milc exhibits a low SSL, pointing to a good locality that should benefit from a standard management. This variability of the pressure on the lines of the same application in different sets is even larger in the context of TAMR2 since, as it also applies SBC policies, some sets that originally had problems to retain their working set in the cache could solve them thanks to displacements.

3.2.1. BIP-C, A New Insertion Policy We propose a modified BIP to apply under the PM mode, called BIP-C, which inserts the incoming lines $min\{C-1, \lfloor K/2 \rfloor - 1\}$ positions away from the LRU position of the recency stack, where C is the number of cores sharing the cache and K the associativity. The rationale for this is that while in a private cache it is only up to the owner application to evict the line with a subsequent miss, or reuse it and bring it to the MRU position, in a shared cache any other application can evict the line before the owner has a chance to reuse it. The situation is even more challenging when the cache works under TAMR2, as lines near the bottom of the LRU stack can be replaced not only by incoming lines mapped to the set, but from lines that come from displacements as a result of applying SBC. BIP-C inserts lines near the bottom of the recency stack in typical configurations (K $\gg$ C) while allowing a few misses in the set from the other running threads before the line is evicted, so that the owner has more chances to prove the merit of the line. Figure 5 shows an example of its operation in a LLC shared by 2 cores and assuming a replacement operation driven by core number 1.

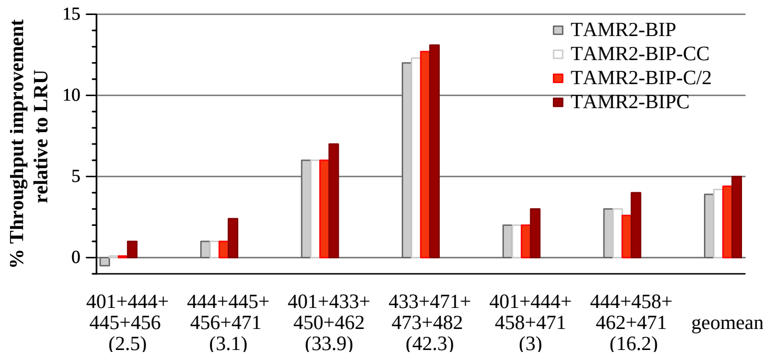

Figure 6. Throughput improvement over a 4MB 16 ways baseline cache shared by 4 cores using TAMR2 with different versions of BIP. MPKI (misses per kilo instruction) of each workload in parenthesis under its name.

Figure 6 shows the throughput improvement of TAMR2 with respect to a baseline 4MB 16-way cache shared by four cores when using four BIP variations. The variations are the original BIP, BIP-CC, which inserts new lines as many positions away from the LRU one as different cores have lines in the set; BIP-C/2, which inserts $min\{\lfloor C/2 \rfloor, \lfloor K/2 \rfloor - 1\}$ positions away from the LRU one, and, finally, BIP-C. The workloads consist of SPEC CPU 2006 benchmarks that will be presented in Section 4 with all the other parameters of the simulations. BIP-C outperforms clearly the other approaches. Altogether, TAMR2 using BIP-C provides 1.1% better throughput than a modified TAMR2 using BIP due to the reasons explained above.

3.3. Coordination and Interaction between the Policies to Reduce Mapping and Replacement Misses

As we have explained, TAMR2 changes to PM mode applications whose SSL reaches the maximum value and which cannot find a destination set to solve a potential problem of mapping misses. The reason is that if there are no candidates, the cache must be under pressure globally, and we cannot provide the sets with more space, so a policy to deal with replacement misses must be applied. Now, if an application experiences this problem, the associations initially established will become useless. In fact, they will be counterproductive: dead lines will be moved to destination sets, giving later place to second searches that will be useless most of the time, delaying the resolution of misses and potentially increasing the miss rate in destination sets. Also, there must be a way to enable an application to change to PM in the sets that participate in an association, and eventually break it if the high SSLs continue. TAMR2 does this by changing the state of a thread to PM also for (1) applications in destination sets in which their SSL reach the highest value, $2 * K - 1$, and (2) applications in source sets that try to displace a line to their destination set and find that the application is in PM mode there. Notice that the second situation implies the SSL of the application is the highest one in the source set as well. This way, if an application suffers replacement misses in an association, this is first detected in the destination set, which is the one that suffers more pressure because of the displacements, and then it is propagated to the source set. Relatedly, in TAMR2 if an application is in PM mode in a source set, it stops displacing lines to the destination set, since the displacements are not helping. This avoids the counterproductive situation described above. It also avoids the perverse effect that lines recently inserted in the destination set with a high priority of eviction (due to the pressure management) are evicted due to displacements from the source set before having a chance to be reused. Further, this strategy facilitates breaking the association, since this happens when the destination set evicts all the lines from the source set. Misses in the source set always lead to searches in the destination set while the association lasts.

While it is clear that insertions are ruled by the application that requests the line to be inserted, a clarification is needed on the rules for displacements. TAMR2, as the SBC, displaces the line that is selected by the replacement policy in the source set. Nevertheless, its displacement must not be

```
access(Core i, Address addr) {
 if hit(addr) {
   SSL[i]−−;
   update priority of line in replacement algorithm;
   if lowlySaturated(SSL[i])
     mode[i] = SM;
 }
 else {
   SSL[i]++;
   if (this is a secondary search)
     return;
   if (set is source set of an association)
     if (access(i, addr) is successful in destinationSet)
       return;
   request addr line to memory;
   let line_to_evict = line to evict according to replacement algorithm;
   if (set is not associated) {
     if (exists candidate destination set in DSS) {
       if mode[line_to_evict.owner] == SM && saturated(SSL[line_to_evict.owner])
         associate this set to candidate destination set;
     } else if saturated(SSL[i])
         mode[i] = PM;
   }
   if (set is source set of an association && mode[line_to_evict.owner] == SM && saturated(SSL[line_to_evict.owner])
     if destinationSet.mode[line_to_evict.owner] == PM {
       mode[line_to_evict.owner] = PM;
       evict line_to_evict from cache;
     } else
       displace line_to_evict to position with highest locality in destination set;
   else
     evict line_to_evict from cache;
   if (set is the destination set of an association) {
     if saturated(SSL[i])
       mode[i] = PM;
     if (there are no lines from the source set)
       break association;
   }
 }
}
```

Figure 7. TAMR2 operation under a cache access.

ruled by the application that generates the eviction, but by the owner of the line to be potentially displaced. Thus TAMR2 requires storing the owner of each line ($\log_2$C bits, where C is the number of cores) in the tag-store. Under a miss, the field is examined for the line to be evicted in order to decide which is the policy to apply based on the mode and SSL of its owner.

Finally, the association algorithm changes slightly with respect to the one in SBC. Association attempts are triggered when under a miss the owner of the line to be evicted is in SM mode and its SSL is the highest one. This is sensible since this is a precondition for a displacement to take place. The association takes place if the Destination Set Selector can provide a suitable destination set, that is, one with a GSSL smaller than K.

Figure 7 shows a C-like pseudocode of the actions taken by TAMR2 under an access to a set. The computation of the GSSL and corresponding update of the DSS, which take place when an SSL is updated, are elided. The insertion of incoming lines, which simply follows the insertion policy of the owner application in the set, is not reflected either because it takes place when they arrive.

Figure 8 illustrates how TAMR2 coordinates SBC and BIP in a 2-way cache shared by two cores, A and B. We use the sets X and Y from Figure 2, assuming that they remain associated under SBC and with the same contents as in the last illustration of that figure. The only difference is that the initial SSLs are assumed to be 0. At this point, a sequence of accesses $A_{x1}B_{x1}B_{x2}A_{x2}B_{x3}B_{x4}A_{x1}B_{x5}B_{x6}A_{x2}\ldots$ reaches the cache. Since the accesses from B have no locality, they should be inserted with low priority, while A would benefit from keeping the two lines it reuses in the cache. The first illustration in Figure 8 shows the status of the cache after the accesses

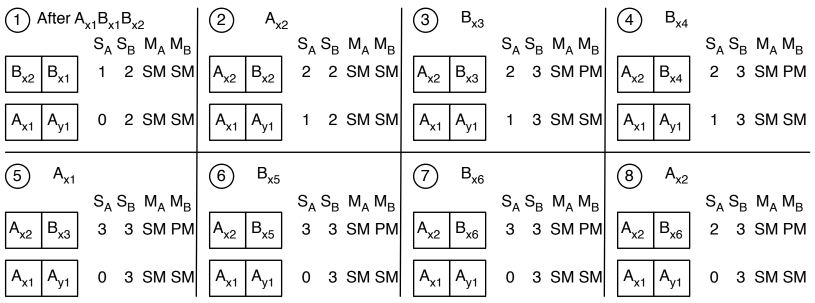

Figure 8. TAMR2 management in a 2-way cache shared by two cores using BIP.

$A_{x1}B_{x1}B_{x2}$, where S_A and S_B are the SSLs and M_A and M_B the management for the threads. In set X $S_A = 1$ because A_{x1} misses there, while it hits in set Y. In both sets $S_B = 2$ because due to the association, B_{x1} and B_{x2} miss in both sets. Each one of the subsequent illustrations update the cache after another access from the sequence. This way, A_{x2} misses in both sets and is inserted under SM in set X. Then, B_{x3} misses in both sets, triggering PM in set X. This results in its insertion in the LRU position under BIP. The same happens in B_{x4}. A_{x1} misses again in set X but is found in set Y. The figure continues showing the updates for B_{x5}, B_{x6} and A_{x2}. Altogether, the facts that B is managed under PM, and that the sets association under SBC allows to balance the load for applications in SM, allows A_{x1} and A_{x2} to remain in the cache, thus providing an optimal miss rate.

Finally, we must mention that real environments present situations of context switching and thread migration. Since TAMR2 decisions are managed by counters of a small size that are updated with each access to the L2, our design can very quickly adjust its policies in each set to new circumstances. This way, when this kind of changes takes place, TAMR2 may take wrong decisions just for a very short period, until the SSLs reflect the new situation.

4. SIMULATION ENVIRONMENT

We use the SESC simulator [17] to evaluate our proposal. The baseline system consists of two cores with private L1 caches and a shared L2 cache. Table I shows the main parameters of the architecture. The tag check delay TC and the total round trip RT time are provided for the L2 to help evaluate the cost of second searches when there are associations of sets. As in [4] and others, the tag and the data arrays are accessed sequentially in the L2 cache. This reduces the power dissipation of large cache arrays and limits the additional delay of second searches to the tag check delay. The power analyses are based on on estimations of the dynamic energy per access in the different levels of the memory hierarchy calculated with CACTI assuming a 32 nm technology and disregarding idle time in DRAM, as the simulator does not model DRAM power modes. As our approach reduces the number of accesses to main memory it is pretty likely that the power reductions we have calculated are smaller than the ones that would be obtained considering DRAM power modes.

As for the parameters that are specific to the different approaches evaluated in this study, TADIP (specifically TADIP-Feedback) [14] and TADRRIP (using *Hit Priority*) [7] use 32 sets dedicated to each policy for each core to decide between BIP and MRU insertion in the former case, and between BRRIP and SRRIP in the latter. These bimodal policies, BIP and BRRIP, as well as the ones triggered by TAMR2, use a probability $\varepsilon = 1/32$ that a new line is inserted in the MRU position of the recency stack or with a long re-reference interval prediction in the BRRIP case. SHiP uses as signature the program counter, as it is the one with the best results in [8], and uses a table of 16K entries with counters of three bits and no set sampling. The RRIP policies in TADRRIP, TAMR2-RRIP and SHiP use two bits to characterize the re-reference prediction and Hit Priority update policy [7]. Finally, the Destination Set Selector of BSBC [19] and TAMR2 has four entries.

Table I. Architecture

Processor	
Frequency	4GHz
Fetch/Issue	6/4
Inst. window size	80 int+mem, 40 FP
ROB entries	152
Integer/FP registers	104/80
Integer FU	3 ALU, Mult. and Div.
FP FU	2 ALU, Mult. and Div.
Branch predictor	Hybrid [18]
Memory subsystem	
TLB entries	64 for instructions + 64 data
L1 i-cache & d-cache	32kB/8-ways/64B/LRU/2 ports
L1 latency (cycles)	4 RT
L2 (unified, inclusive, shared)	4MB/16-ways/64B/LRU/1 port
L2 latency (cycles)	14 RT, 6 TC
System bus bandwidth	10GB/s
Memory latency	125ns

RT, round trip; TC, tag directory check.

4.1. Multiprogrammed workloads

Twelve benchmarks from the SPEC CPU 2006 suite have been selected to make 16 multiprogrammed workloads with at least an MPKI (misses per kilo instruction) of 1. They have been simulated using the reference input set (*ref*), during the execution of their first 10 billion instructions after the initialization. When the first core reaches this number of instructions it continues its execution until the second core finishes, in order to keep both cores competing for the cache shared resources. The statistics for each application are only recorded during the execution of its first 10 billion instructions. These long simulations, in which applications can go through several stages, lead to lower IPC improvements over the baseline for all the policies in our experiments than those observed in other studies. We feel the results achieved in this way are very stable and representative of real behaviors. Table II characterizes the multiprogrammed workloads with their miss rate (MR) and MPKI in a 4MB 16-ways L2 cache.

5. TAMR2 RESULTS AND ANALYSIS

Figure 9 shows the throughput improvement of all the approaches relative to the LRU traditional policy for the 16 workloads described above. In this figure and all the subsequent ones the last group

Table II. Multiprogrammed workloads characterization.

Name	Benchmarks	L2 miss rate(%)	MPKI
MW1	471.omnetpp + 473.astar	7.8	6
MW2	433.milc + 482.sphinx3	65.1	26
MW3	401.bzip2 + 462.libquantum	32.3	13.2
MW4	462.libquantum + 470.lbm	36.1	45.6
MW5	401.bzip2 + 429.mcf	17.5	24
MW6	429.mcf + 433.milc	49	72
MW7	444.namd + 471.omnetpp	1.6	1.1
MW8	456.hmmer + 482.sphinx3	52.4	10.4
MW9	445.gobmk + 473.astar	12.9	4.7
MW10	462.libquantum + 471.omnetpp	24.5	13.4
MW11	433.milc + 473.astar	42.1	19.2
MW12	458.sjeng + 482.sphinx3	52.4	9
MW13	429.mcf + 444.namd	15.4	18.2
MW14	429.mcf + 445.gobmk	18.3	21
MW15	433.milc + 470.lbm	37.5	56
MW16	401.bzip2 + 458.sjeng	6.5	1.2

MPKI, mises per kilo instruction

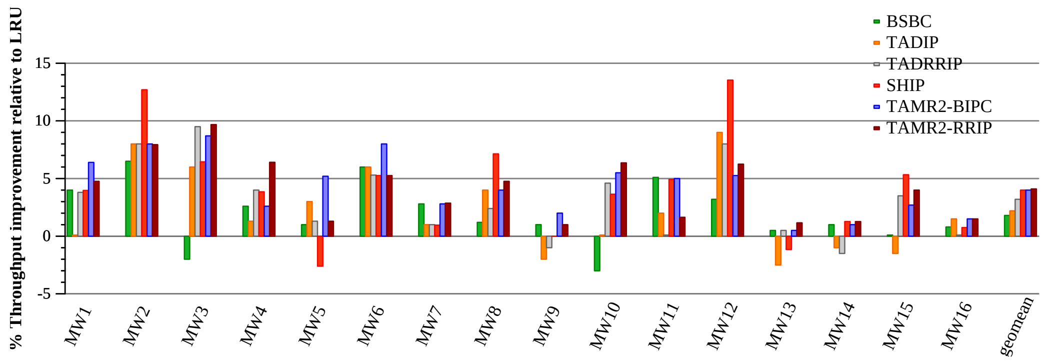

Figure 9. Throughput improvement over the baseline configuration using several policies.

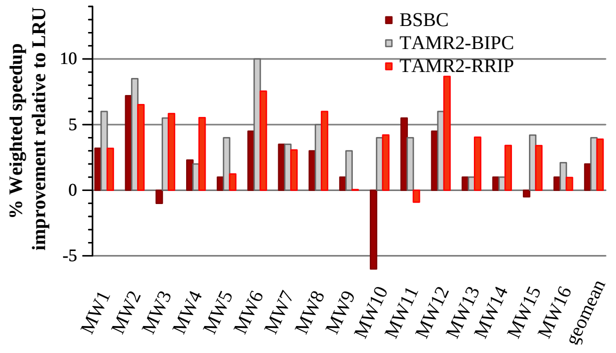

Figure 10. Weighted speedup improvement over the baseline configuration using BSBC and TAMR2.

of columns corresponds to the geometric mean for the values achieved for all the workloads. The mean improvement achieved by each technique is: BSBC 1.8%, TADIP 2.2%, TADRRIP 3.1%, SHiP 4%, 4% for TAMR2 with BIP-C and 4.1% for TAMR2 with RRIP. These results show two trends that appear consistently in all the evaluation. The first one is that, non surprisingly, thread-aware approaches, tend to outperform in general the non-thread-aware BSBC. For example, comparing BSBC and TAMR2-BIPC: out of the 16 workloads, the thread-aware TAMR2 technique proposed in this paper outperforms the thread-oblivious BSBC, which also reduces replacement as well as mapping misses, in 11 workloads, while both yield very similar results in the other 5 workloads. Further, TAMR2-BIPC improvements over the baseline are on average 120% larger than those of BSBC. The second trend observed is that those approaches that tackle both mapping and replacement issues tend to achieve the best results in their category. The other techniques can reduce replacement misses. Some also improve the detection and eviction of dead blocks in each independent set, which can reduce conflicts, but they cannot exploit the load unbalance among sets, therefore it is more difficult for them to make effective use of large portions of dead cache lines [11].

The other performance metrics analyzed show very similar trends to those observed in Figure 9 for the throughput improvement. Therefore, Figures 10 and 11, devoted to the weighted speedup [20] and the harmonic mean of weighted speedups [21], respectively, only show the values for BSBC and TAMR2 in order to emphasize the value of thread-awareness. This way, under the weighted speedup metric the relative advantage of TAMR2-BIPC over the baseline is about 100% larger than that of BSBC, since this technique is able to apply the best policies to each thread depending on its behavior. This helps avoid slowing down threads at the expense of others. When TAMR2 is applied with the RRIP policies, a 50% of additional improvement with respect to BSBC is achieved.

Our fairness metric also clearly reflects the positive properties of TAMR2 in this regard in Figure 11. On average, BSBC gets 2.3% improvement over the baseline, which is largely improved by TAMR2-BIPC with 4.4% and TAMR2-RRIP with 4.8%. The average values for the other techniques evaluated not shown in the graph are: TADIP 2.6%, TADRRIP 3.4% and SHiP 4%.

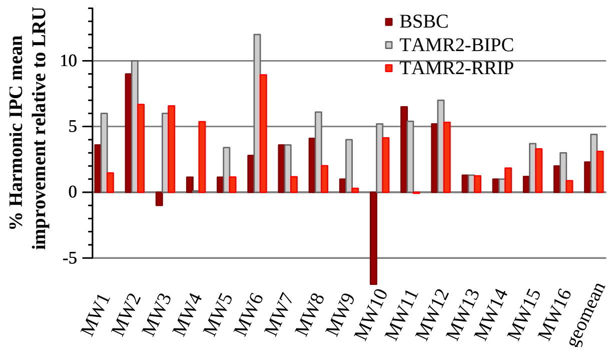

Figure 11. Harmonic IPC improvement over the baseline configuration using BSBC and TAMR2.

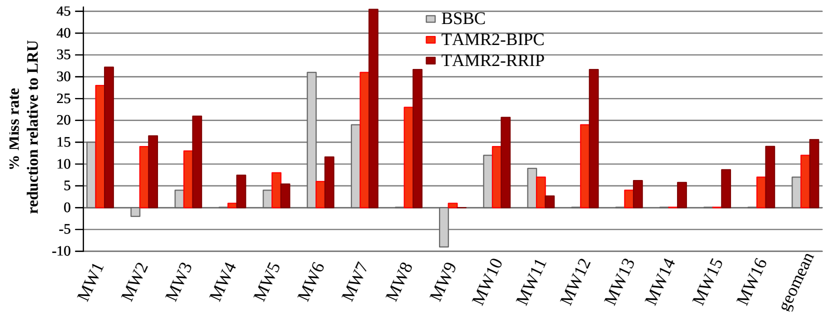

Figure 12. Miss rate reduction over the baseline configuration using BSBC and TAMR2.

As expected, thread-aware policies tend to show more potential for fairness. Among them, TAMR2 has the best behavior because while the other techniques are restricted to managing the working set associated to each cache set, TAMR2 can better promote a fair usage of the cache resources thanks to the incorporation of a displacement policy among sets based on their state.

It is interesting to notice that TAMR2 is the only proposal that does not slow down any IPC metric for absolutely any workload in any configuration with respect to its baseline. This is in contrast with other strategies, which can reduce the metrics up to 10% (15% in the case of fairness) in some experiments.

Another metric we evaluate is the global MR, which considers, as a whole, the accesses of all the applications during their simultaneous execution until the slowest one completes its 10 billion instructions. These rates give an idea of the reduction of bandwidth to memory provided by each approach, a resource which becomes more critical as the number of cores in CMPs increases. Since accesses to memory require much more power than cache hits, they are also a good indicator of the energy savings that can be achieved by the different techniques. Figure 12 shows this MR reduction relative to the one observed in the baseline for the 4MB L2 caches considered. The (geometric) average relative reduction of the MR achieved by each policy is BSBC 7%, TAMR2-BIPC 12% and TAMR2-RRIP with 15.6%. The qualitative tendency in these figures is the same as in the IPC metrics. When the benefit of thread-aware policies is added, the good MR reduction achieved by BSBC can still be almost duplicated by TAMR2-BIPC. In fact, this technique, with only a 0.64% overhead as we will see in section 5.4, provides 40% of the MR reduction achieved by doubling the size of our baseline cache to 8MB. Still more impressive is TAMR2-RRIP, which reaches more than 50% of that reduction with an overhead below 0.3%.

The percentage that second hits represent in terms of power consumption and latency has been counted in Figure 13, which shows the percentage of power consumption and latency reduction achieved by TAMR2-RRIP, the best TAMR2 implementation. Each bar is broken down in the percentage of hits that are satisfied in the second level of the memory hierarchy or in the main memory. The percentage due to the L1 hits is not shown because it is very similar for all the

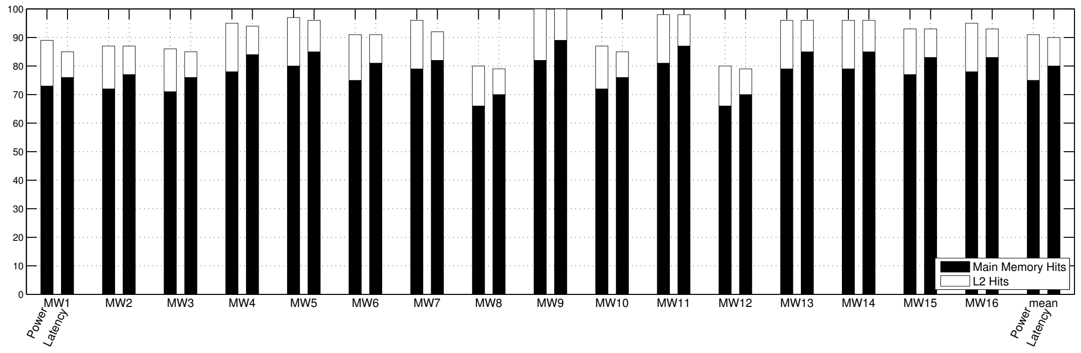

Figure 13. Average power consumption and memory latency reduction achieved by TAMR2-RRIP relative to the baseline using **2** applications. Each bar shows a breakdown of the accesses, either satisfied in the L2 cache or in main memory.

Table III. Percentage of throughput, weighted speedup, harmonic IPC improvement and MR reduction of TAMR2-BIPC over the baseline, varying the cache size.

Cache size (MB)	Throughput improvement (%)	Weighted Speedup (%)	Harmonic IPC improvement (%)	Miss rate reduction (%)
1	4.5	6.3	6.3	11.0
2	5.1	4.2	4.3	14.3
4	4.0	4.0	4.4	12.0
8	3.0	3.2	3.5	11.0

approaches. As the tag check delay means only a 3% of the total power consumption per read/write access in the cache, our approach has a negligible power consumption overhead. On average, a 9% and 10% of power consumption and memory latency savings are achieved.

5.1. Scalability Analysis

In this section the scalability of TAMR2 with respect to the cache size and the number of cores are evaluated. This way, Table III shows the evolution of the metrics previously considered to evaluate TAMR2 with respect to a baseline 16 ways cache shared by two cores for different cache sizes when it is used in conjunction with BIP-C. These values are geometric means obtained on the 16 multiprogrammed workloads in Table II. As expected the values tend to diminish with the cache size, but the reduction is not continuous or pronounced.

Figure 14 shows the throughput improvement with respect to the baseline 4MB 16-way cache using four cores. All the workloads used, identified by the benchmark numbers, have also an MPKI (in parenthesis) greater than 1. On average, BSBC gets an improvement of 1.8%, TADIP 3.5%, TADRRIP 3.4%, SHiP 3.6%, TAMR2-BIPC 5% and TAMR2-RRIP 5.6%. The increase in the number of cores that share the cache leads thread-aware techniques to more clearly outperform the non-thread-aware BSBC. As for TAMR2, the variety of policies it uses, coupled with the accuracy and fine grain with which it applies them, allows it to adapt very well to an increasing number of cores, showing the best potential for scalability of results. Its fine granularity and accuracy are also reflected in the fact that it does not slow down any workload in this environment either. Regarding the TAMR2 variations evaluated, the advantage of TAMR2-RRIP over TAMR2-BIPC observed in the 2-core experiments grows in this environment. Finally, Figure 14 also compares a CMP to a private LLC (16 ways, 1MB per core, 8 cycles for the round trip latency and 4 for the tag check delay) per core. We can see the importance of sharing resources, as this configuration clearly performs worse than our baseline shared LLC.

Figure 15 completes our scalability study with the power consumption and latency reduction achieved by TAMR2-RRIP in our 4-core experiments. The results also indicate a growth in the

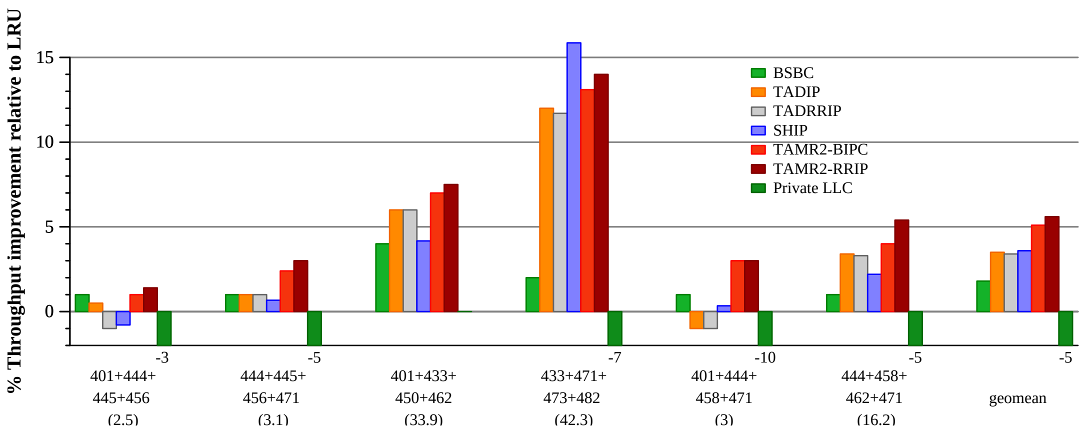

Figure 14. Throughput improvement over the 4MB 16 ways baseline cache shared by **4** cores using several policies. MPKI of each workload in parenthesis under its name.

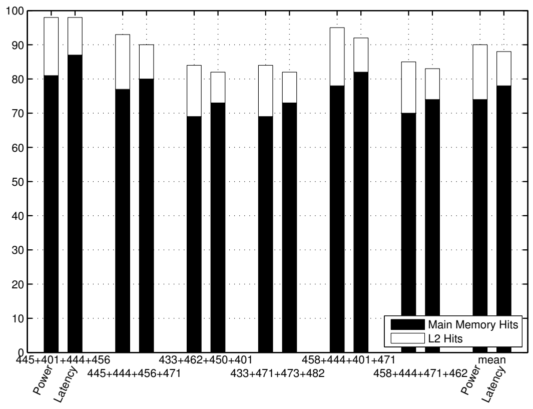

Figure 15. Average power consumption and memory latency reduction achieved by TAMR2-RRIP relative to the baseline using **4** applications. Each bar shows a breakdown of the accesses, either satisfied in the L2 cache or in main memory.

improvements achieved by TAMR2 in this environment. This way, TAMR2-RRIP reduces power consumption and memory latency by 10% an 12%, respectively.

5.2. *Experiments with multithreaded applications*

We performed experiments with multithreaded applications in order to evaluate TAMR2 in environments where each core tends to exert the same pressure on the shared cache. For these experiments benchmarks from SPLASH2 and PARSEC were run on 4 cores for 10 billion instructions (most of them until completion) using the large input set for PARSEC and the appropriate input set for SPLASH2. Most of these benchmarks are not hard memory demanding so the L2 capacity was reduced to 512 Kb to get more meaningful results. Figure 16 shows the reduction in execution time over the baseline using 4 threads for several techniques. As expected, the execution time reductions achieved are smaller than in multiprogrammed workloads. This way, the average performance improvement is 3% for TAMR2-BIPC and 4.5% for TAMR2-RRIP. In a multithreaded application it makes sense that the advantage of RRIP over BIP-C is larger than the observed in a multiprogrammed workload. The reason is that BIP-C provides more opportunities for a line to be reused before it is evicted from a set than RRIP in order to take into account that the cache is shared with other cores. Concretely, when there are replacement problems in a set, a single miss will evict a line inserted by RRIP with the longest re-reference prediction interval, while C-1 misses will be required by a line inserted by BIP-C, where C is the number of cores sharing the cache. If a

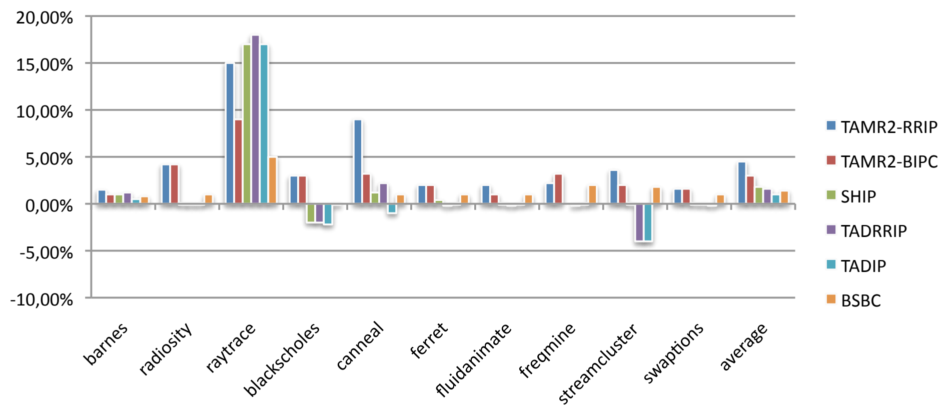

Figure 16. Reduction in execution time over the baseline LRU cache for several policies when running multithreaded applications with four threads.

single application is running in the C cores and BIP-C or RRIP is activated by one of its threads in a set, most chances are that all the threads are experiencing replacement misses (for example, due to thrashing). In this situation, while RRIP only wastes one way of the cache to store the data without locality, BIP-C subjects C-1 ways to thrashing. This hurts multithreaded applications under BIP-C when more than 2 cores share the cache. In any case, both TAMR2 variants clearly outperform execution time reductions of 1.8% of SHiP, 1.6% of TADRRIP, 1% of TADIP and 1.4% of BSBC.

5.3. *Interaction with Prefetching*

We have performed experiments adding a 16KB stride prefetcher to the L2 shared cache for the TAMR2 and the baseline configurations. The inclusion of this prefetcher made the baseline performance improve 5% on average. We briefly comment some of these results in terms of IPC improvement. For the 2-core experiments TAMR2-BIPC got an average IPC improvement of 3.97% over the baseline, quite similar to the 4% obtained without prefetching. The IPC improvement achieved by TAMR2-RRIP grew to 4.6%, versus the 4.1% achieved without prefetching. As for the 4-core applications, TAMR2-BIPC achieved a 6.1% improvement, which is 22% greater than the 5% reached in the plain experiments. TAMR2-RRIP reached a 8.3% average IPC improvement, which nearly duplicates the 4.4% improvement achieved without prefetching. This way, we can make three important observations from these results. First, in the presence of a prefetcher, the advantages derived from the application of TAMR2 stay similar or grow. This makes sense since, particularly in the case of shared caches, a prefetcher further stresses the bandwidth requirements of the memory. Therefore, any policy that saves memory bandwidth, such as TAMR2, has a potential for a greater impact than in an environment without prefetcher. Even more, as TAMR2 keeps the lines with more locality in the cache, although still giving a chance to those without locality, the prefetcher is able to perform better predictions, therefore achieving better results. The second observation is that TAMR2 advantages grow, both in absolute and relative terms, as the number of applications sharing the cache increases. This is due both to the larger bandwidth demand as we increase the number of cores sharing the cache and to the nature of TAMR2. Namely, the particular management of each core that TAMR2 proposes allows the prefetcher to obtain a more accurate prediction, achieving higher hit rates than the prefetcher applied to the baseline configuration. For example, as the BIP-C policy may prevent recently inserted lines from being evicted by a different core from that one that brought the line to the cache, the stream of references that the prefetcher receives has fewer interferences [22] between cores than the prefetcher used in the baseline configuration. Note that we are using a simple prefetcher, without regarding the interaction between several cores in the shared cache. Our last observation is that of the two TAMR2 variations evaluated, TAMR2-RRIP is the one in which these effects are stronger. The reason is that the advantages of RRIP over BIP and

Table IV. Baseline and TAMR2 storage cost in a 4MB/16-way/64B/LRU cache shared between **2** cores.

	Baseline	TAMR2-BIPC	TAMR2-BRRIP
Tag-store entry:			
State(v+dirty+Repl+[d]+[c])	6 bits	8 bits	6 bits
Tag ($42 - \log_2 sets - \log_2 64$)	24 bits	24 bits	24 bits
Size of tag-store entry	30 bits	32 bits	30 bits
Data-store entry:			
Set size	64*16*8 bits	64*16*8 bits	64*16*8 bits
Additional structs per set:			
Saturation Counters	-	2*5 bits	2*5 bits
Insertion policy bit	-	2*1 bits	2*1 bits
Association Table	-	12+1 bits	12+1 bits
Total of structs per set	-	25 bits	25 bits
DSS (entries+registers)	-	11B	11B
Number of tag-store entries	65536	65536	65536
Number of data-store entries	65536	65536	65536
Number of sets	4096	4096	4096
Size of the tag-store	245760B	262144B	245760B
Size of the data-store	4096kB	4096kB	4096kB
Size of additional structs	-	12811B	12811B
Total	**4336kB**	**4364kB**	**4348.5kB**

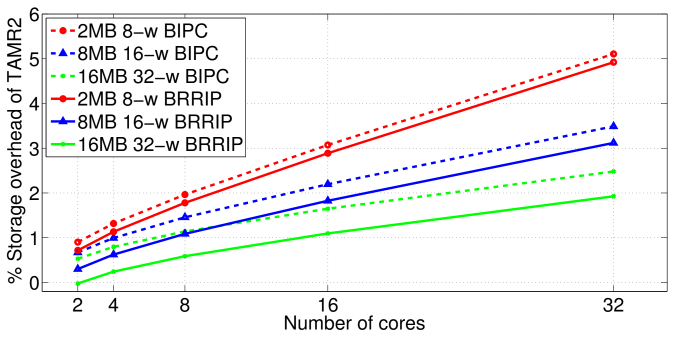

Figure 17. Storage overhead of TAMR2 as a function of the number of cores for several cache configurations.

even BIP-C are more noticeable as the demand on the cache increases and the prefetcher removes misses that no replacement policy can avoid.

5.4. *Cost*

We consider here the costs of TAMR2 in terms of storage. Table IV calculates the storage required for a 4MB 16-way baseline cache with lines of 64B assuming addresses of 42 bits. The TAMR2 using BIP-C requires the following additional hardware with respect to a standard cache: a saturation counter per set to compute the SSL for each core; an additional bit per entry in the tag-array to identify displaced lines (d bit in Table IV); an Association Table with one entry per set that stores a bit to specify whether the set is the source or the destination of the association; and the index of the set it is associated to, and, finally, a Destination Set Selector (DSS) to choose the best set for an association. TAMR2 also needs one bit per set to indicate the management mode for each core. Based on this, the TAMR2 storage overhead using BIP-C is about 0.64%. The overhead is reduced to 0.29% when TAMR2 applies SRRIP/BRRIP thanks to the reduction from 4 to 2 bits per line needed for the replacement policy. In comparison, the overhead of SHiP, the alternative with the most similar performance, is around 2.54% for this configuration.

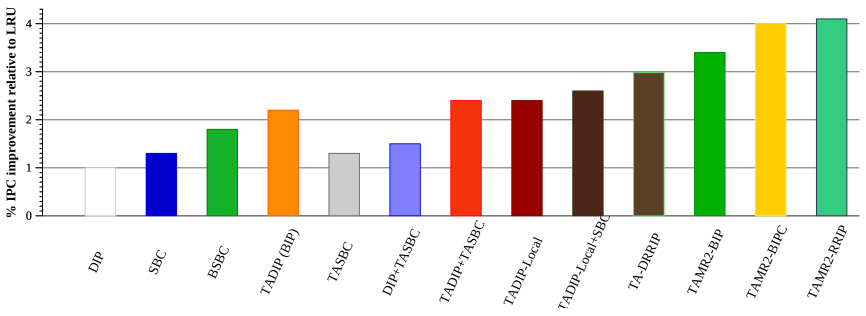

Figure 18. Throughput improvement over a 4MB 16 ways baseline cache shared by two cores under several policies.

Figure 17 shows the hardware overhead scalability of our design when varying the number of cores for three cache configurations. We can see that for typical realistic configurations the cost stays around 1% or below. In fact, very unrepresentative configurations, such as a 2MB cache shared by 8 cores or a 8MB cache shared by 16, are needed to reach a 2% cost, which is still reasonable. As the cache size increases, the overhead is reduced. Also, its growth is sublinear with respect to the number of cores that share the cache. Altogether, while the cost of other techniques designed for shared caches is even smaller, the large performance benefits of TAMR2 coupled with its small cost make it a very interesting design point, particularly given the large availability of transistors and the critical role of cache performance in current systems. Additionally, the cost of the structures per set can be reduced by applying the same policies and SSL counters to a number of consecutive sets instead of a single one. This ideas has been successfully validated in [23], proving that it can even improve the performance achieved. We do not evaluate this here due to space limitations and because it is not needed in the studied environments.

5.5. Contribution of each Policy to TAMR2 Performance

Our evaluation finishes analyzing by means of Figure 18 the contribution of each portion of TAMR2 to its global behavior by comparing the average throughput improvement that several policies achieve over the 4MB 16-ways baseline cache shared by two cores described in Section 4 when running the workloads in Table II. From left to right, a non-thread aware Dynamic Insertion Policy (DIP) [6], which selects between BIP or LRU insertion depending on which one is working better in the cache, provides less performance than a non-thread aware SBC. The Bimodal Set Balancing Cache (BSBC) [19], which combines BIP and SBC in private caches, but which, unlike TAMR2, is unaware of the existence and the behavior of the different applications, performs much better by coordinating efficiently placement and insertion policies to reduce mapping and replacement misses. TADIP thread-awareness applied to insertion policy management brings large advantages in shared caches, as seen in [14].

Nevertheless, a thread-aware SBC (TASBC), that is a cache that uses an SSL per core in each set in order to apply SBC to the accesses of each code depending on its specific SSL value, has no advantages over SBC. The reason is that the problem solved by SBC, i.e., the unbalance of the load of different cache sets, is not specific to the behavior of a thread, but to the combined behavior of all of them in each set. The importance of adequately coordinating the insertion and placement policies can be seen by comparing BSBC, which is totally thread-oblivious but coordinates the insertion policy and SBC, with DIP+TASBC, which applies independently DIP and the TASBC policy described above, being each one of them totally unaware of the behavior of the other one.

Applying TADIP with TASBC in an uncoordinated way brings similar performance advantages to those of discovering and applying the best insertion policy to each thread in each set independently, a technique we have called TADIP-Local. This technique uses an SSL per core in each set to choose between BIP, which is activated when the SSL is saturated, and the usual MRU insertion policy, which is goes into effect when the SSL falls below K, for the accesses of the core associated to the SSL. This way, TADIP-Local acts like TAMR2 without its SBC component, and using BIP instead of BIP-C or RRIP. If TADIP-Local is combined with SBC (which, let us remember, has very similar performance to TASBC) in an uncoordinated way, a small additional performance gain is achieved. Let us recall that all these approaches that combine in a non-coordinated way a technique that tries to reduce replacement misses, like DIP, with another one that mainly tackles mapping misses, like SBC, can generate harmful behaviors like displacing recently inserted lines in source sets working under BIP, giving raise to unsuccessful second searches, or like evicting just inserted lines in destination sets due to displacements, depriving them of a chance to be reused before being evicted. As a result, this technique achieves worse results than the simple TA-DRRIP policy [7]. The situation changes noticeably when TADIP-Local+SBC is improved by means the coordination provided by the TAMR2, giving place to TAMR2 (BIP), which uses BIP for those sets in PM mode, just like TADIP and TADIP-Local. Finally, when TAMR2 applies the BIP-C policy proposed in this paper, performance improves to levels very similar to those of TAMR2 operating with the SRRIP/BRRIP policies, as the last two bars show.

The relative contributions of each policy vary of course with the environment. For example thread-awareness relevance grows with the number of cores sharing the cache, while SBC becomes more important for workloads whose working sets fit relatively well in the cache, for which MRU insertion works well.

6. CONCLUSIONS

The traditional problems due to the load unbalance among cache sets and the suboptimal replacement algorithms present in single core environments, which the Set Saturation Level (SSL) metric has been successfully proved to detect [4] [19], are found in shared caches as well. In fact, new misses of both kinds appear in shared caches due to the effects of the joint working set of the applications sharing them. From this analysis, this paper proposes, in a reasoned way, a coordinated thread-aware strategy to alleviate both problems. This technique, called Thread-Aware Mapping and Replacement Miss Reduction (TAMR2) policy measures the degree of pressure that each application applies to each cache using the Set Saturation Level. When TAMR2 estimates an application is experiencing problems in a set, it first tries to displace lines of the problematic application to underutilized sets applying the Set Balancing Cache [4] techniques. When this fails or is not possible, it resorts to policies specifically designed to deal with thrashing such as a modified Bimodal Insertion Policy, called BIP-C, or BRRIP.

Simulations using a wide range of workloads indicate five things. First, thread-awareness is a very desirable property for policies oriented to shared caches and can be successfully managed by the SSL as well. Second, among them, TAMR2 consistently achieves the best results overall. This is due to two key characteristics that clearly distinguish it from all the other thread-aware techniques we are aware of. The first one is the small granularity with which it can take and apply decisions, as opposed to the global decisions of other approaches. The second one is its coordinated approach to reduce mapping and replacement misses that can balance load among sets. This latter issue is largely ignored by the other proposals specifically designed for shared caches, which only focus on the workload inside each set. We think that it is of the upmost importance to identify the dead blocks in the cache and make the best usage of them. This requires changes to the placement policy like the ones explored here. The third conclusion is that, according to our experiments, TAMR2 exhibits a very good scalability potential. The fourth observation is that despite its fine-grained nature both in terms of measurement and modification of the cache behavior, and the variety of policies that it can apply, TAMR2 cost is very reasonable; well below 1% in representative configurations. Finally, it is worth to point out that the proposed BIP-C insertion policy implies a notable improvement with respect to the original BIP [6] in shared caches, as it protects lines from early displacements due to other threads.

In this paper we have evaluated TAMR2 considering one thread per core. In a SMT system, the number of threads to manage equals the number of cores multiplied by the number of threads per core. Each one of these hardware threads would be treated as a (virtual) core from the point of view of TAMR2.

As future work, the cooperative implementation of other policies to reduce misses can be explored. Reductions of the hardware required by TAMR2 can also be studied, for example grouping the observation and management of nearby sets. A combined replacement algorithm that takes into account the core making the request can also be explored.

ACKNOWLEDGEMENTS

This work has been supported by the Galician Government under projects Consolidation of Competitive Research Groups (ref 2010/6), INCITE08PXIB105161PR and UDC/GI-000265, and the Ministry of Education and Science of Spain, FEDER funds of the European Union (Project TIN2010-16735). We also thank the anonymous reviewers for their suggestions, which helped improve the paper. The authors are members of the HIPEAC network.

REFERENCES

1. Sugumar RA, Abraham SG. Efficient simulation of caches under optimal replacement with applications to miss characterization. *Proceedings of the ACM Sigmetrics Conference on Measurement and Modling of Computer Systems*, SIGMETRICS '93, 1993; 24–35.
2. Jouppi NP. Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers. *Proceedings of the 17th International Symposium on Computer Architecture*, 1990; 364–373.
3. Qureshi MK, Thompson D, Patt YN. The V-Way cache: Demand-based associativity via global replacement. *Proceedings of the 32st International Symposium on Computer Architecture*, 2005; 544–555.
4. Rolán D, Fraguela BB, Doallo R. Adaptive line placement with the Set Balancing Cache. *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture*, 2009; 529–540.
5. Bershad BN, Lee D, Romer TH, Chen JB. Avoiding conflict misses dynamically in large direct-mapped caches. *Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*, ASPLOS-VI, 1994; 158–170.
6. Qureshi MK, Jaleel A, Patt YN, Steely Jr SC, Emer JS. Adaptive insertion policies for high performance caching. *Proceedings of the 34th International Symposium on Computer Architecture*, 2007; 381–391.
7. Jaleel A, Theobald KB, Jr SCS, Emer JS. High performance cache replacement using re-reference interval prediction (RRIP). *Proceedings of the 37th International Symposium on Computer Architecture*, 2010; 60–71.
8. Wu CJ, Jaleel A, Hasenplaugh W, Martonosi M, Jr SCS, Emer JS. Ship: signature-based hit predictor for high performance caching. *Proceedings of the 44th IEEE/ACM International Symposium on Microarchitecture*, 2011; 430–441.
9. Hennessy JL, Patterson DA. *Computer Architecture: A Quantitative Approach*. 4th edn., Morgan Kaufmann, 2006.
10. Kharbutli M, Irwin K, Solihin Y, Lee J. Using prime numbers for cache indexing to eliminate conflict misses. *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, 2004; 288–299.
11. Khan SM, Jiménez DA, Burger D, Falsafi B. Using dead blocks as a virtual victim cache. *Proceedings of the 19th International Conferene on Parallel Architecture and Compilation Techniques*, 2010; 489–500.
12. Chaudhuri M. Pseudo-LIFO: The foundation of a new family of replacement policies for last-level caches. *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009; 401–412.
13. Srikantaiah S, Kandemir MT, Irwin MJ. Adaptive set pinning: managing shared caches in chip multiprocessors. *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2008; 135–144.
14. Jaleel A, Hasenplaugh W, Qureshi MK, Sebot J, Jr SCS, Emer JS. Adaptive insertion policies for managing shared caches. *Proceedings of the 17th International Conference on Parallel Architecture and Compilation Techniques*, 2008; 208–219.
15. Xie Y, Loh GH. PIPP: promotion/insertion pseudo-partitioning of multi-core shared caches. *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2009; 174–183.
16. Qureshi MK, Patt YN. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006; 423–432.
17. Renau J, Fraguela B, Tuck J, Liu W, Prvulovic M, Ceze L, Sarangi S, Sack P, Strauss K, Montesinos P. SESC simulator January 2005. Http://sesc.sourceforge.net.
18. McFarling S. Combining branch predictors 1993.
19. Rolán D, Fraguela BB, Doallo R. Reducing capacity and conflict misses using Set Saturation Levels. *Proceedings of the 2010 International Conference on High Performance Computing*, 2010.
20. Snavely A, Tullsen DM. Symbiotic jobscheduling for a simultaneous multithreading processor. *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000; 234–244.
21. Luo K, Gummaraju J, Franklin M. Balancing throughput and fairness in SMT processors. *Proceedings of the 2001 IEEE International Symposium on Performance Analysis of Systems and Software*, 2001; 164–171.

22. Wu CJ, Martonosi M. Characterization and dynamic mitigation of intra-application cache interference. *Proceedings of the 2011 IEEE International Symposium on Performance Analysis of Systems and Software*, 2011; 2–11.
23. Rolán D, Fraguela BB, Doallo R. Adaptive set-granular cooperative caching. *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture*, 2012; 213–224.