Reducing Capacity and Conflict Misses using Set Saturation Levels

Dyer Rolán, Basilio B. Fraguela and Ramón Doallo
Grupo de Arquitectura de Computadores
Departamento de Electrónica e Sistemas
Universidade da Coruña
e-mail: {drolan, basilio.fraguela, ramon.doallo}@udc.es

***Abstract*—The well-known memory wall problem has motivated wide research in the design of caches. Last-level caches, whose misses can stall the processors for hundreds of cycles, have received particular attention. Strategies to modify adaptably the cache insertion, promotion, eviction and even placement policies have been proposed, some techniques being better at reducing different kinds of misses. For example changes in the placement policy of a cache, which are a natural option to reduce conflict misses, can do little to fight capacity misses, which depend on the relation between the working set of the application and the cache size. Nevertheless, other techniques such as the recently proposed dynamic insertion policy (DIP), whose aim is to retain a fraction of the working set in the cache when it is larger than the cache size, attack primarily capacity misses. In this paper we present a coordinated strategy to reduce both capacity and conflict misses by changing the placement and insertion policies of the cache. Our strategy takes its decisions based on the concept of the Set Saturation Level (SSL), which tries to measure to which degree a set can hold its working set. Despite requiring only less than 1% storage overhead, our proposal, called Bimodal Set Balancing Cache, reduced the average miss rate of a baseline 2MB 8-way second level cache by 16%, which translated into an average IPC improvement of 4.8% in our experiments.**

***Index Terms*—Cache; performance; adaptivity; balancing; insertion; replacement; thrashing; set saturation level**

I. Introduction

Memory hierarchy plays a key role in the performance of current computers given the memory wall problem. This has led to the current state of the art with several levels of caches, the ones nearest to the processor being primarily optimized for latency, and the last-level caches (LLC) being responsible for avoiding off-chip memory accesses, which can stall the processor for hundreds of cycles. The larger flexibility that LLCs allow with respect to their response times and the enormous importance of reducing their miss rate has led the community to propose a large number of techniques to improve their adaptability to the behavior of applications. For example, the problem of the lack of uniformity in the distribution of the memory references among the sets of set associative caches, which is a fundamental source of conflict misses, has been addressed by victim caches [1], the adaption of the assignment of lines to sets [2] or the displacement of lines from oversubscribed sets to underutilized ones [3]. Other proposals seem more adequate to reduce capacity misses. A very good example is [4], which targets memory-intensive workloads with working sets that not fit in the cache for which the traditional LRU replacement policy is counterproductive.

In this paper we investigate the possibility of combining a technique that targets primarily conflict misses with another one particularly suitable to reduce capacity misses. Our proposal is based on the set saturation level, a concept proposed in [3] to measure the degree to which a set is able to hold its working set. This indicator was used to decide when a set requires more lines than the ones available, or when its lines were underutilized. The Set Balancing Cache (SBC) they introduce associates sets with high saturation levels with sets with reduced saturation levels, allowing displacements of lines from the former to the latter. Since the SBC targets the placements of lines in the cache, it is mainly effective at reducing conflict misses, but it can do little to improve the performance when the working set of a workload is larger than the cache size.

The core idea of our proposal is to use the set saturation levels not only as indicators of unbalance among the cache sets, but also as detectors of lack of capacity of the cache to hold the working set. In order to solve the first problem, our design, which we call Bimodal Set Balancing Cache (BSBC) applies the Dynamic SBC (DSBC) introduced in [3]. If the lack of lines to hold the working sets persists after the displacement of lines from oversubscribed sets to underutilized ones, the BSBC applies a policy to address problems of capacity. Namely, the insertion policy of highly saturated sets is changed to the Bimodal Insertion Policy (BIP) [4], which often inserts lines in the LRU position instead of the MRU one. This avoids that lines that are dead on arrival expel other lines from the cache as they descend in the LRU stack. Another possible interpretation of our strategy is that both DSBC and BIP target the same problem, which is that a set may have more active blocks than ways, the first solution trying to move the blocks to an underutilized or cold set, while the other one tries to evict cold blocks within the set. Our approach then switches between the best one of the two solutions depending on the availability of underutilized sets.

Our experiments show that our coordinated approach to fight conflict and capacity misses works substantially better than simply applying simultaneously in a cache DSBC and Dynamic Insertion Policy (DIP) [4]. The latter is an insertion policy that chooses dynamically between the traditional MRU insertion policy and BIP based on a set dueling mechanism that tries to identify the one that incurs fewer misses.

The rest of this paper is organized as follows. The next section introduces the basics of the DSBC cache and the BIP and DIP insertion policies, discusses the limitations of those approaches and reasons why and in which way they can be complementary. This leads to the description of the Bimodal Set Balancing Cache we propose in Section III, which is evaluated using the environment described in Section IV. The results are discussed in Section V. The cost of this approach is examined in Section VI. Related work is discussed in Section VII. Finally, the last section is devoted to the conclusions and future work.

II. Two complementary cache management policies

As the preceding section states, the bibliography shows proposals that are more suitable to reduce conflict misses due to the oversubscription of specific sets of the cache and techniques that try to address a global problem of capacity of the cache with respect to the working set in use. We have chosen as representative techniques of both families of proposals the Dynamic Set Balancing cache [3] and the novel insertion policies proposed in [4]. Both approaches will be discussed in turn, followed by a constructive critic of their limitations and their complementarity.

A. Dynamic Set Balancing Cache

The basic idea of the Dynamic Set Balancing Cache, DSBC in what follows, is to alleviate the problems of oversubscribed cache sets by moving part of the lines originally mapped to them to other sets that have underutilized lines. This requires detecting the degree to which each cache set is able to hold its working set. The DSBC achieves this with a metric called Set Saturation Level (SSL) which is tracked separately for each set by means of a counter called saturation counter. This counter, which has saturating arithmetic, is increased each time an access to the set results in a miss, and decreased when it results in a hit. A counter in the range 0 to 2K-1 for K-way associative caches is proposed in [3].

A cache set is deemed as highly saturated, and thus unable to hold its working set, when its SSL, which is the value of its saturation counter, reaches the maximum (2K-1). At that point, the DSBC tries to associate this set with another set with a low SSL that will be used to store part of the working set of the saturated set. The DSBC chooses for this purpose the set with the lowest SSL in the cache that is not yet associated to another set, provided that its SSL is smaller than K. The rationale for this latter limitation is that it does not seem useful to associate two sets with high SSLs in order to lend lines from one of them to the other one.

Once an association is established, the less saturated set becomes the destination set for displacements of lines from the highly saturated set, which becomes the source set of the association. The displacements take place when a line is evicted from the source set and its SSL adopts the maximum value. In this situation the line evicted is moved to the destination set of the association rather than to memory. Displaced lines are marked with a bit that allows to distinguish them from the native lines of the destination set. Insertions of lines in sets always take place in the MRU position of the recency stack and the traditional LRU replacement policy is applied. Another consequence of the association is that while it lasts, accesses that result in misses in the source set of the association make a second attempt to find the requested data in the destination set. This gives place to secondary hits and misses. An association is broken when the destination set evicts all the lines it has received from the source set.

B. Adaptive Insertion Policies

Most caches nowadays use a LRU replacement policy in which lines are inserted in the MRU position in the recency stack. The lines must then descend this stack position by position until reaching the LRU position before being evicted. Although this policy works very well for many workloads, in [4] it was observed that it often leads memory-intensive workloads with working sets larger than the cache size to thrashing. As a result, they proposed an LRU Insertion Policy (LIP) which always inserts lines in the LRU position of the recency stack. If the inserted line is reutilized, it is moved to the MRU position, as in any cache. If the line is not reused before the next miss in the set, the next line inserted replaces it. At this point it is very important to remember that [4], just as [3] and this paper, deal with non first level caches. LIP exploits the fact that in these caches many lines are dead on arrival (i.e. they are not reused in the cache before their eviction) because all their potential short term temporal or spatial locality is exploited in upper level caches.

While LIP works well for some workloads, it may tend to retain in the non-LRU positions of the recency stack lines which are actually not useful, that is, that do not belong to the current working set. Thus a Bimodal Insertion Policy (BIP) that tries to adapt the contents of the set to the active working set of the application was also proposed in [4]. BIP achieves this by inserting with a low probability ε the incoming lines in the MRU position of the recency stack, operating like LIP all the other times.

There is not an absolute winner between BIP and the traditional MRU insertion policy, BIP being better suited for some applications and the traditional policy for others. Thus [4] proposed the Dynamic Insertion Policy (DIP), which uses set dueling to track the behavior of both insertion policies in order to apply the best one to the remaining cache sets, called follower sets. Set dueling requires dedicating a fraction of the cache sets to operate always under BIP and another fraction to apply always MRU insertion. A group of 32 sets dedicated to each policy is found to be good in [4]. The misses in one of the groups of sets increase the value of a global saturating counter while the misses in the other group decrease it. The most significant bit of the counter indicates then which is the best policy in each moment. All the follower sets apply the policy indicated by the counter.

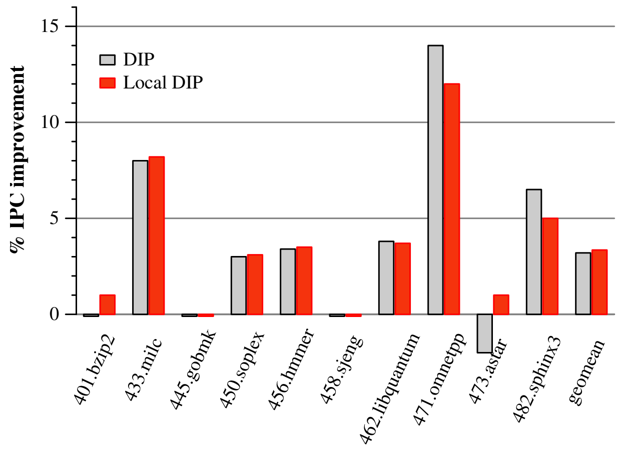

Figure 1. Percentage of IPC improvement over a 2MB 8-ways and 64 bytes line size baseline cache using DIP and Local DIP.

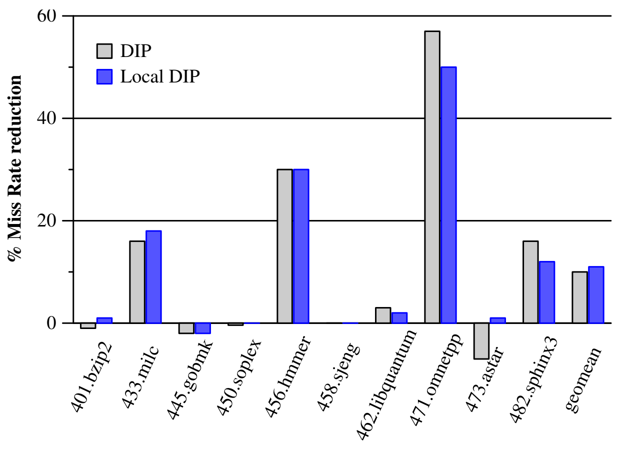

Figure 2. Miss rate reduction related to a 2MB 8-ways and 64 bytes line size baseline cache using DIP and Local DIP.

C. Discussion

The DSBC relies on the unbalance of the working set of different sets to trigger the mechanisms that make it different from a standard cache. As a result it is oriented to reduce conflict misses rather than capacity misses. Another consequence of this approach is that its metrics must be defined at the cache set level in order to find these unbalances, which is indeed the case of the SSL.

The adaptive insertion policies proposed in [4] alleviate the lack of capacity of the cache to hold the data set manipulated by an application. Since this is a global problem for all the cache, these policies are applied to all the sets at once. For the same reason, DIP, the one that can adapt dynamically to the characteristics of the workloads, relies on a global metric gathered on the behavior of sets spread along the cache.

This way, it looks straightforward that DSBC and DIP should be complementary and that implementing them simultaneously in a cache will offer a higher level of protection against misses than using only one of them. This is also very feasible given their reduced hardware overhead. Nevertheless, the simultaneous implementation of DSBC and DIP, which we call DSBC+DIP, yields results very similar or even worse than the ones achieved with any of them independently, as we will see in Section V. The main problems happen when DIP chooses BIP for the followers. DSBC displaces the LRU line of the source sets, as it seems the natural option. This means that the line that DIP exposes to be evicted on the next miss in the source set of an association is actually saved by DSBC, which moves it to the destination set. Since these lines are not actually useful, their existence in the destination set gives place to unsuccessful second searches, which delay the resolution of the miss by the time taken to make a new access to the tag array. Even worse, they may expel an actually useful line just inserted in the LRU position of the destination set before it gets a chance to be reused.

When DIP chooses the traditional insertion policy for the followers, a DSCB+DIP cache behaves very much as a DSBC with two penalties. The first one, which is inherent to DIP, is the existence of sets (32 in our implementation, as advised in [4]) that are forced to apply BIP for the sake of the set dueling even when it is not performing well. The other is that when these sets are involved in an association they generate the problems discussed above.

As a result, while these policies seem complementary, they require a coordinated approach to work properly together. Our proposal is presented in the next section.

III. Bimodal Set Balancing Cache

A first issue we explore in the attempt to exploit jointly DSBC and the adaptive insertion policies is the possibility of using the same metric to control them. This will ease their coordination and it can even simplify the hardware with respect to the one required for implementing them separately, thanks to the reuse of the hardware that computes the metric. A metric per set like the one that DSBC requires cannot be obtained from the global counter used by DIP. Thus we checked whether the decision that DIP takes based on the set dueling can be made instead based on the SSL provided by the DSBC. This would not only simplify the design of the cache, but also avoid having always a fraction of the cache sets working with a wrong policy, even if this fraction is small. A way to achieve this is to use the SSL of each set to decide whether the traditional insertion policy of BIP is better suited for that specific set. Our proposal is to change the insertion policy of a set to BIP only if it gets saturated (SSL=2K-1 for a saturation counter in the range 0 to 2K-1), and revert to MRU insertion when it reduces clearly its SSL. An SSL below K has been chosen to trigger the change to MRU insertion. This proposal, which we call Local DIP, only involves a saturation counter and one additional bit per set, called *insertion policy bit*, that indicates the insertion policy of the set. Local DIP is compared with DIP in terms of IPC improvement and miss rate reduction over a baseline 8-way second level cache of 2MB with lines of 64 bytes in Figures 1 and 2, respectively. BIP uses $\varepsilon = 1/32$ as the probability a new line is inserted in the MRU position of the recency stack instead of in the LRU one in both implementations. The simulation environment and the benchmarks are explained in Section IV. The results are similar, Local DIP being slightly better than DIP on average. Thus we dropped set dueling in favor of a local per-set decision based on its SSL.

Let us consider now the nature of the SSL. A high SSL indicates that the set cannot hold its working set, but it is difficult to know only with this value whether this is a problem specific to the set, which means other sets have no problems with their working sets, or a global problem of capacity of the cache. The answer lies in the comparison with the SSL of the other cache sets. If the cache has enough capacity to hold its working set, the DSBC mechanism should be able to find suitable sets to be associated to the problematic one, allowing it to displace part of the lines of its working set to a destination set with underutilized lines. If the DSBC cannot associate the set, that is because there are no sets with a SSL low enough to deem them good candidates to receive lines from other sets. This then points to a potential problem of capacity of the cache, which can be dealt with adopting BIP. Since DSBC only seeks to initiate associations when a set is saturated, a good strategy is to first try to associate the set to a destination set, and if no good candidate is found, change the set insertion policy to BIP. Altogether this strategy equates to first trying to consider the high SSL in the set as a local problem, that is, conflict misses due to the oversubscription of this specific set, and if this fails, consider that there may be a global problem of capacity which requires turning to BIP. If the cache has a capacity problem, it is very likely that sets that were chosen previously as destinations of an association become saturated too. Thus we propose that destination sets that become saturated change their insertion policy to BIP. Relatedly, it is logical that if the source set of an association gets saturated and its destination set is applying BIP, the source set changes to BIP too. This acknowledges that a capacity problem rather than a local conflict problem is been faced. Finally, just as in our Local DIP, if the SSL of a set in BIP mode drops below K, its insertion policy changes to MRU insertion, since the capacity problems seem to have disappeared.

The eviction of recently inserted lines in destination sets that operate under BIP by lines displaced from their source set was identified as one of the problems of the DSBC+DIP approach in Section II-C. BIP puts useful lines in destination sets in a dangerous situation because, since they are inserted in the LRU position, any displacement before their reuse evicts them from the cache. Our design avoids this enforcing that destination sets in BIP mode are in read-only mode. This means that misses in their source set will lead to searches in them, but no displacements of lines from the source set will be allowed. This is consistent with our view that BIP is triggered to fight capacity problems rather than conflicts. Another positive side-effect of this policy is that since no displacements are allowed in BIP mode, it is easier to break the association, which is in fact not helpful when there is a capacity problem. Let us recall that the association is broken when during its operation the destination set evicts all the lines it received from the source set. Finally, when the SSL of a destination set goes below K, besides reverting to MRU insertion, it also enables again the displacement of lines from its source set.

Altogether, our proposal, which we call Bimodal Set Balancing Cache (BSBC) because it is a Set Balancing Cache with an integrated BIP, has almost the same hardware overhead as a DSBC. Only one additional bit is required per set in order to store its current insertion policy. As for the time required to apply the BSBC algorithms, just as in the DSBC and DIP, they are triggered by misses and can be thus overlapped with their resolution. The contention in the tag array due to second searches has been considered in our evaluation.

Table I
ARCHITECTURE. IN THE TABLE RT, TC AND MSHR STAND FOR ROUND TRIP, TAG DIRECTORY CHECK AND MISS STATUS HOLDING REGISTERS, RESPECTIVELY.

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.
Common memory subsystem	
L1 i-cache & d-cache	32kB/8-way/64B/LRU
L1 Cache ports	2 i/ 2 d
L1 Cache latency (cycles)	4 RT
L1 MSHRs	4 i / 32 d
System bus bandwidth	10GB/s
Memory latency	125ns
Two levels specific memory subsystem	
L2 (unified, inclusive) cache	2MB/8-way/64B/LRU
L2 Cache ports	1
L2 Cache latency (cycles)	14 RT, 6 TC
L2 MSHR	32
Three levels specific memory subsystem	
L2 (unified, inclusive) cache	256kB/8-way/64B/LRU
L3 (unified, inclusive) cache	2MB/16-way/64B/LRU
Cache ports	1 L2, 1 L3
Cache latency (cycles)	11 RT L2, 4 TC L2, 39 RT L3, 11 TC L3
MSHR	32 L2, 32 L3

IV. Simulation Environment

Our evaluation is based on simulations performed using the SESC environment [5]. The baseline system has a four-issue out-of-order CPU clocked at 4GHz. Two memory hierarchies, one with two levels of cache, and another one with three levels, have been studied. When a modification of the design of the caches of the baseline system is evaluated, it is applied in the L2 cache of the hierarchy with two levels of cache, and in the L2 and L3 caches of the hierarchy with three levels. Table I displays the simulation parameters, whose timings assume a 45nm technology process.

The three-level hierarchy is inspired in the Core i7 [6], the L3 being proportionally smaller to account for the fact that only one core is used in our experiments. Both configurations allow an aggressive parallelization of misses, as there are

Table II
BENCHMARKS CHARACTERIZATION.

Benchmark	# L2 Accesses	2MB L2 Miss rate	256kB L2 Miss rate	Component
bzip2	125M	9%	41%	INT
milc	255M	71%	75%	FP
gobmk	77M	5%	10%	INT
soplex	105M	8%	15%	FP
hmmer	55M	10%	41%	INT
sjeng	32M	26%	27%	INT
libquantum	156M	74%	74%	INT
omnetpp	100M	28%	91%	INT
astar	192M	23%	48%	INT
sphinx3	122M	68%	76%	FP

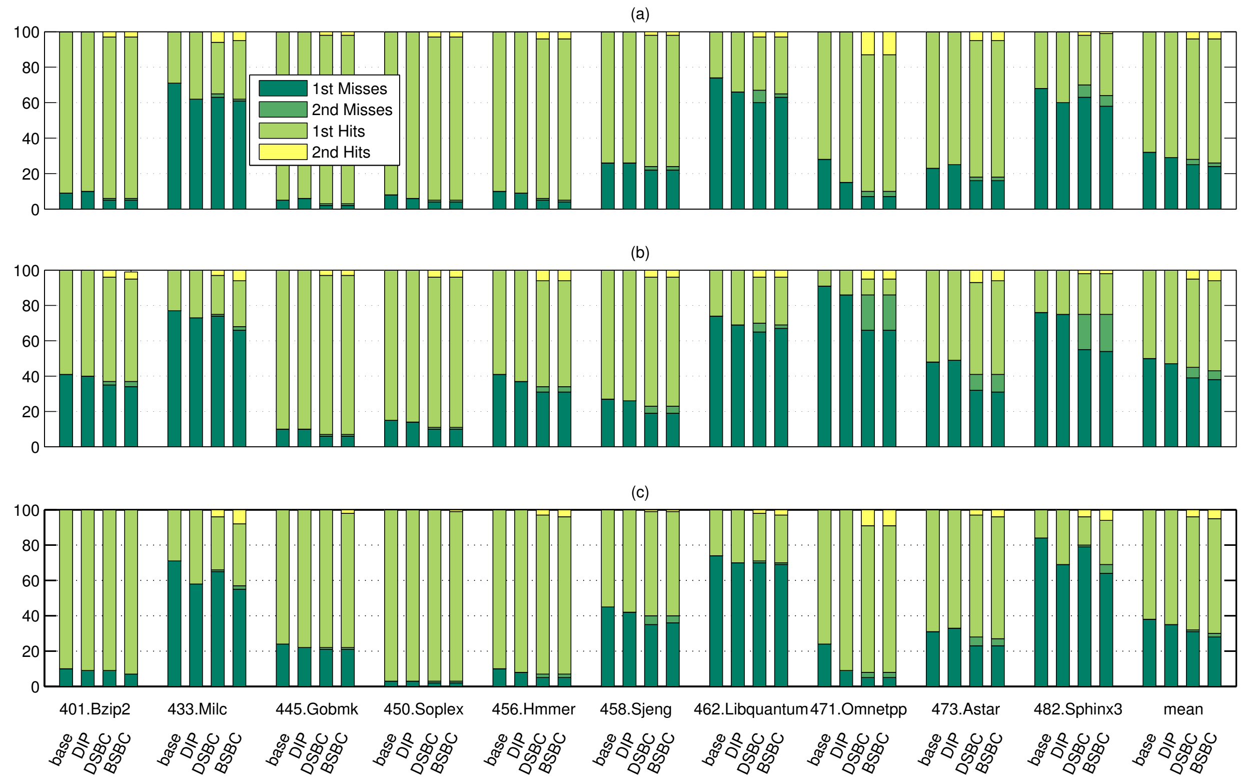

Figure 3. Primary miss, secondary miss, primary hit and secondary hit rates for the baseline system, a DIP variation, a DSBC variation and a BSBC one for (a) L2 cache in the two-level configuration, (b) L2 cache in the three-level configuration, and (c) L3 cache in the three-level configuration.

between 16 and 32 Miss Status Holding Registers per cache. The tag check delay and the total round trip access are provided for the L2 and L3 to help evaluate the cost of second searches when there are associations. As in several existing processors [7][8], and works in the bibliography [9][10][2][3], the accesses to non-first level caches access sequentially the tag and the data arrays. This reduces the power dissipation of large cache arrays and limits the additional delay of second searches to the tag check delay.

As for the parameters that are specific to the different approaches used in this study, DIP uses 32 sets dedicated to each policy to decide between BIP and MRU insertion. This BIP, as well as the one triggered by the BSBC, uses a probability $\varepsilon = 1/32$ that a new line is inserted in the MRU position of the recency stack. The DSBC and the BSBC use a DSS, or Destination Set Selector, of four entries like the one used in [3]. This is the structure that stores the potential sets for an association in order to provide quickly a candidate when it is requested.

A. Benchmarks

Ten benchmarks from the SPEC CPU 2006 suite have been used to evaluate our approach. They have been simulated using the reference input set (*ref*), during 10 billion instructions after the initialization. Table II characterizes them with the number of L2 accesses during the simulation, the miss rate in the L2 cache both in the two-level (2MB L2) and the three-level (256kB) configurations, and whether they belong to the INT or FP set of the suite. As we can see, the benchmarks in this mix vary largely both in number of accesses that reach the caches under the first level and in the miss ratios.

V. Results and comparison with other approaches

The behavior of DIP, DSBC and BSBC is compared in the second level cache of the hierarchy with two levels of caches and the two lower levels of the hierarchy with three levels in Figure 3. It shows the rate of accesses that result in misses after a single access to the tag-array (primary misses) or two (secondary misses), and accesses that hit in the cache in the first check of the tag-array (primary hits) or after the second one (secondary hits). Only the DSBC and the BSBC present secondary accesses, which take place when an access misses in the source set of an association. The last group of columns (mean), represents the arithmetic mean of the rates observed in each cache. For example in the L2 of the two-level configuration the BSBC gets an average miss rate (considering both kinds of misses) of 27% compared to the 32% of the baseline configuration. This is a relative reduction in the miss rate of 15.7%. In this cache DIP achieves a miss rate reduction of 10% and DSBC of 11.5%. So we can see that our design allows the two policies to work coordinately getting the best of each one of them. The ratio of all the cache accesses that result in secondary misses is 2% and 3% for the BSBC and the DSBC, respectively. The additional delay of a secondary miss with respect to a primary miss is small, but still it is good that BSBC not only generates more hits than DSBC but also reduces by 1/3 the number of secondary misses. The reduction is not surprising if we realize that in applications with capacity problems, the saturation of the destination sets will avoid displacements of lines that are actually not useful. This is will also enable these sets to break the association before. Let us remember that an association is broken when the destination set evicts all the lines received from the source set. Altogether this leads to fewer unsuccessful secondary searches than in the DSBC. DSBC and BSBC present the same rate of accesses that result in secondary hits, about 4%.

Looking at individual benchmarks we can appreciate how BSBC adapts to the different types of applications, often performing better than both DIP and DSBC. For example, in 433.milc, 462.libquantum and 482.sphinx3, which are more suited to DIP than to DSBC, BSBC achieves similar results to DIP (somewhat worse in 482.sphinx3), and better than DSBC.

The BSBC is also able to adapt to those applications that benefit more from DSBC because of imbalances in the working sets sizes for different cache sets. This happens, for example, in 401.bzip2, 471.omnetpp and 473.astar, where BSBC and DSBC work better than DIP. Therefore, the BSBC works largely as DIP for streaming applications using BIP, and mostly as the DSBC when the application presents imbalances among the working sets of sets. It is often the case that the BSBC even improves over both approaches by combining both behaviors.

The effects just discussed can be seen more clearly in Figure 4, which shows the relative miss rate reduction with respect to the baseline L2 cache of the configuration with two levels of cache for five cache designs. The approaches compared are DIP, DSBC, their simultaneous implementation

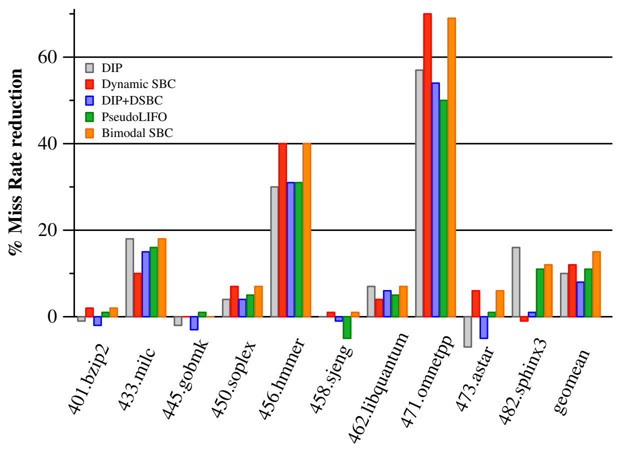

Figure 4. Miss rate reduction over the baseline in the two-level configuration using DIP, DSBC, their simultaneous implementation DIP+DSBC, pseudoLIFO and the BSBC.

DIP+DSBC, pseudoLIFO and the BSBC. PseudoLIFO stands for the probabilistic escape LIFO, which has been simulated approximating the hit counts by the next power of two for escape probabilities. This is the best policy among the family of pseudoLIFO replacement policies proposed in [11] according to its evaluation. These policies will be reviewed briefly in Section VII.

The last group of columns in Figure 4 is the geometric mean of the miss rate reduction for each approach. Altogether, BSBC achieves a relative miss rate reduction of 16% compared to the 12% of the DSBC, the 11% of pseudoLIFO and the 10% of DIP. The worst performer is DIP+DSBC, which only reduces the miss rate by 8.3% despite its increased complexity with respect to DSBC or DIP taken separately. So it is very interesting that in fact without the contributions of this paper, the combination of DIP and DSBC achieves the worst performance, while following the approach we propose, their coordinated application achieves the best result by far.

Figures 5 and 6 show the performance improvement in terms of instructions per cycle (IPC) for each benchmark with respect to the baseline system in the two level and the three level memory hierarchies tested, respectively. The cache designs considered are the same as in Figure 4. In the two-level configuration the BSBC always has a positive or, at worst, in the case of 445.gobmk benchmark, a negligible negative effect on performance smaller than 1%. The geometric mean of the relative IPC improvement for DSBC with respect to the baseline configuration is 4.8% in the two-level configuration. DIP, DSBC, DIP+DSBC and pseudoLIFO achieve 3.2%, 3.6%, 3% and 3.4%, respectively. The analysis based on IPC points again to the importance of the contributions of this paper. A coordinated effort of DIP and DSBC guided by the SSL allows to go from the worst results with a straightforward DIP+DBSC to the best performance with BSBC. The situation is very similar in the configuration with three levels of cache. Here BSBC improves 6% on average the IPC with respect

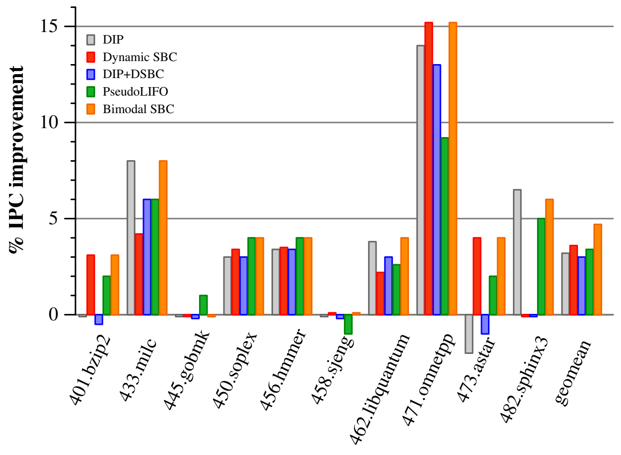

Figure 5. Percentage of IPC improvement over the baseline in the two-level configuration using DIP, the DSBC, a combination of both previous approaches, pseudoLIFO or the BSBC.

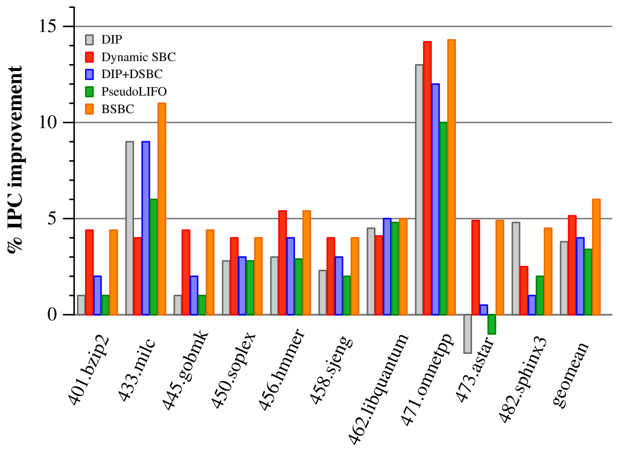

Figure 6. Percentage of IPC improvement over the baseline in the three-level configuration using DIP, the DSBC, a combination of both previous approaches, pseudoLIFO or the BSBC.

Table III
BASELINE, DSBC AND BSBC STORAGE COST IN A 2MB/8-WAY/64B/LRU CACHE.

	Baseline	DSBC	BSBC
Tag-store entry:			
State(v+dirty+LRU+[d])	5 bits	6 bits	6 bits
Tag $(42 - \log_2 sets - \log_2 64)$	24 bits	24 bits	24 bits
Size of tag-store entry	29 bits	30 bits	30 bits
Data-store entry:			
Set size	64*8*8 bits	64*8*8 bits	64*8*8 bits
Additional structs per set:			
Saturation Counters	-	4 bits	4 bits
Insertion policy bit	-	-	1 bit
Association Table	-	12+1 bits	12+1 bits
Total of structs per set	-	17 bits	18 bits
DSS (entries+registers)	-	4*(1+12+4)+2*(2+4) bits	4*(1+12+4)+2*(2+4) bits
Number of tag-store entries	32768	32768	32768
Number of data-store entries	32768	32768	32768
Number of Sets	4096	4096	4096
Size of the tag-store	118.7kB	122.8kB	122.8kB
Size of the data-store	2MB	2MB	2MB
Size of additional structs	-	8714B	9226B
Total	**2215kB**	**2228kB** ($< 1\%$)	**2229kB** ($< 1\%$)

to the baseline system, while DIP, DSBC, DIP+DSBC and pseudoLIFO achieve increases of 3.8%, 5%, 4% and 3.5%, respectively.

We can observe that pseudoLIFO outperformed DIP and DIP+DSBC in the two-level configuration. Nevertheless the opposite happened in the three-level configuration, and by somewhat larger margins. This, coupled with the negligible hardware cost and simple algorithm of DIP were the main reasons to choose DIP as the preferred approach to deal with capacity misses in the BSBC design.

VI. COST

We consider here the costs of the BSBC in terms of storage and area. The latter has been modeled with CACTI 6.5 [12] assuming a 45 nm technology process. The costs of the DSBC cache are also computed for comparison purposes, the cost of DIP being negligible (just 10 bits for the global counter).

The BSBC, like the DSBC, requires the following additional hardware with respect to a standard cache: a saturation counter per set to compute the SSL, an additional bit per entry in the tag-array to identify displaced lines (d bit), 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. A 4-entry DSS has been used in our evaluation. The BSBC needs also one bit per set to indicate the set insertion policy. Based on this, Table III calculates the storage required for a baseline 8-way 2 MB cache with lines of 64B assuming addresses of 42 bits. The corresponding area overhead calculated by CACTI is shown in Table IV.

Table IV
BASELINE, DSBC AND BSBC AREA.

Config.	Components	Details	Subtotal	Total
Baseline	Data + Tag	2MB 8-way 64B linesize + tag-store	4.3 mm^2	**4.3** mm^2
Dynamic SBC	Data + Tag	2MB 8-way 64B linesize + 1 additional displaced bit in tag-store	4.31 mm^2	**4.33** mm^2 ($< 1\%$)
	Counters	4096*4 bits	< 0.01 mm^2	
	Association Table	4096*12 bits	< 0.01 mm^2	
	DSS (entries + registers)	4*(1+12+4)+2*(2+4) bits	< 0.01 mm^2	
Bimodal SBC	Data + Tag	2MB 8-way 64B linesize + 1 additional displaced bit in tag-store	4.31 mm^2	**4.34** mm^2 ($< 1\%$)
	Counters	4096*4 bits	< 0.01 mm^2	
	Insertion Policy	4096 bits	< 0.01 mm^2	
	Association Table	4096*12 bits	< 0.01 mm^2	
	DSS (entries + registers)	4*(1+12+4)+2*(2+4) bits	< 0.01 mm^2	

VII. RELATED WORK

There have been many proposals to improve the performance of caches. We will review here briefly the ones that are more related to ours. The impossibility for some cache sets to hold their working set has been addressed by victim caches [1], which simply store the latest lines evicted from the cache. This idea has been later refined with heuristics to decide which lines to store in the victim cache. For example, [13] takes its decisions on reload intervals, while [14] considers the frequency with which each line appears in the miss stream.

The unbalance in the working set of different cache sets has also been tackled with a large variety of approaches. This way, cache indexing functions [15][16] that distribute accesses more uniformly than the standard indexing have been proposed. Another alternative are pseudo-associative caches, which increase the flexibility of the placement of lines in the cache considering two [17][18] or more alternative locations [19] for each line. The pseudo-associative caches, contrary to the BSBC, do not use any mechanism such as the SSL to decide when it is better to displace a line. Besides they perform searches line by line rather than set by set. Something similar happens with the adaptive group-associative cache (AGAC) [20], a proposal for first level direct-mapped caches that stores lines that would have been otherwise evicted in underutilized cache frames. AGAC takes decisions on a line basis, based by recency of use and needs multiple banks to aid swapping.

The Indirect Index Cache (IIC) [9] offers full associativity because its tag-store entries keep pointers that allow to associate them to any data-entry. The IIC is accessed through a hash table with chaining in a singly-linked collision table and its management is much more complex than that of the BSBC. For example, its generational replacement is run by software.

The NuRAPID cache [10] relocates data-entries inside the data array in order to reduce the average access latency. This requires the usage of pointers between tag-entries and data-entries. Nevertheless its tag-array is managed in a standard way both in terms of mapping and replacement policies. As a result, it has the same miss rate and workload imbalance problems among sets as a standard cache.

The V-Way cache [2] duplicates the number of sets and tag-store entries while keeping the same associativity and number of data lines. Data lines are assigned dynamically to sets depending on the demand that each set experiences and a global replacement algorithm on the data lines based on their reuse frequency. This allows different sets to end up having different numbers of lines. Forward pointer between the tag-store and the data-store entries allow any data line to be assigned to any tag-entry. Reverse pointers allow the replacement algorithm on the data lines to reassign them from one tag-store entry to another one. According to their studies, the V-Way cache offers similar miss rate reductions as IIC and outperforms AGAC.

The Dynamic Set Balancing Cache (DSBC) [3] has already been explained in detail in Section II-A. That paper also presented a Static SBC (SSBC), which restricts each set to be associated only with the farthest set in the cache, that is, the one whose index is obtained complementing the most significant bit of the set index. We have used DSBC because it outperformed SSBC consistently in their experiments thanks to its flexibility, while the extra implementation cost is small. It also achieved better miss rates than the V-way cache and DIP.

The proposals we have just discussed emphasize the flexibility of placement of lines in the cache to improve miss rates or access time. Other researchers have focused on the modification of the insertion and replacement policies in order to keep in the set where they belong the most useful lines. Examples of this kind of works are the adaptive insertion policies in [4] that were detailed in Section II-B and the pseudo-LIFO replacement policies [11]. Pseudo-LIFO policies evict blocks from the upper part of the fill stack, i.e., among the most recently inserted lines of the set. This contributes to retain a large fraction of the working set in the cache. The probabilistic escape LIFO policy used in our evaluation, which dynamically learns the most preferred eviction positions within the fill stack and prioritizes the ones close to the top of the stack, belongs to this family. It has been included in our comparative evaluation because it outperformed many recent proposals on a set of single-threaded, multiprogrammed, and multithreaded workloads in [11].

VIII. CONCLUSIONS

There has been extensive research to improve the behavior of caches, particularly of non-first level ones. Different approaches are sometimes better suited to reduce different kinds of misses. For example, DSBC [3] and DIP [4] target somewhat different problems. If implemented jointly in the cache, complementary techniques like these ones should have the potential to achieve better performance than any of them isolated. Nevertheless, as this paper shows with the case of DSBC and DIP, implementing them at the same time is not enough to exploit their advantages. In fact, the direct simultaneous application of these techniques often yields worse results than the usage of only one of them. We have analyzed the reasons for this behavior and proposed in a reasoned way an integrated design of these policies that allows them to cooperate effectively. As part of this design we demonstrate the usefulness of the Set Saturation Level (SSL) metric to detect problems of capacity in the cache. Since this metric already proved successful at detecting problems of unbalance among the working set of different sets in [3], it was natural to turn it into the arbiter of our coordinated approach to fight conflict and capacity misses.

Simulations using benchmarks with varying characteristics show that, when properly integrated with our Bimodal Set Balancing Cache (BSBC) design, the joint application of the DSBC and BIP policies goes from being often one of the worst approaches to being the best one. For example in a 2MB, 8-way second level cache DIP+DSBC jointly reduces the miss rate by 8.3% in relative terms, while DSBC and DIP reduce it by 12% and 10% respectively. With BSBC the relative miss rate reduction almost doubles to 16%. This leads also the BSBC to get the largest IPC improvement, 4.8% on average for this configuration, compared to the 3% that a straight DSBC+DIP implementation provides. The other policies tested, DSBC, DIP and probabilistic escape LIFO lay in between.

Future work includes evaluating and adapting the ideas developed in this paper in the context of shared caches.

Acknowledgments

This work was supported by the Xunta de Galicia under project INCITE08PXIB105161PR and the Ministry of Science and Innovation, cofunded by the FEDER funds of the European Union, under the grant TIN2007-67537-C03-02. Also, the authors are members of the HiPEAC network. Finally, we want to acknowledge Mainak Chaudhuri for assistance with the pseudo-LIFO cache.

References

[1] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers," in *Proc. 17th Intl. Symp. on Computer Architecture*, June 1990, pp. 364–373.

[2] M. K. Qureshi, D. Thompson, and Y. N. Patt, "The V-Way Cache: Demand-Based Associativity via Global Replacement," in *Proc. 32st Intl. Symp. on Computer Architecture*, June 2005, pp. 544–555.

[3] D. Rolán, B. B. Fraguela, and R. Doallo, "Adaptive line placement with the set balancing cache," in *Proc. 42nd IEEE/ACM Intl. Symp. on Microarchitecture*, December 2009, pp. 529–540.

[4] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., and J. S. Emer, "Adaptive insertion policies for high performance caching," in *Proc. 34th Intl. Symp. on Computer Architecture*, June 2007, pp. 381–391.

[5] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," January 2005, http://sesc.sourceforge.net.

[6] Intel Corporation, "Intel core i7 processor extreme edition and intel core i7 processor datasheet," 2008.

[7] Digital Equipment Corporation, "Digital semiconductor 21164 alpha microprocessor product brief," March 1997.

[8] D. Weiss, J. Wuu, and V. Chin, "The on-chip 3-mb subarray-based third-level cache on an itanium microprocessor," *IEEE journal of Solid State Circuits*, vol. 37, no. 11, pp. 1523–1529, November 2002.

[9] E. G. Hallnor and S. K. Reinhardt, "A fully associative software-managed cache design," in *Proc. 27th annual Intl. Symp. on Computer architecture*, June 2000, pp. 107–116.

[10] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures," in *Proc. of the 36th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, December 2003, pp. 55–66.

[11] M. Chaudhuri, "Pseudo-lifo: the foundation of a new family of replacement policies for last-level caches," in *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 401–412.

[12] HP Labs, "CACTI 6.5," cacti65.tgz, retrieved in May, 2010, from http://www.hpl.hp.com/research/cacti/.

[13] Z. Hu, S. Kaxiras, and M. Martonosi, "Timekeeping in the memory system: predicting and optimizing memory behavior," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 209–220, 2002.

[14] A. Basu, N. Kirman, M. Kirman, M. Chaudhuri, and J. F. Martínez, "Scavenger: A new last level cache architecture with global block priority," in *40th Annual IEEE/ACM Intl. Symp. on Microarchitecture*, December 2007, pp. 421–432.

[15] A. Seznec, "A case for two-way skewed-associative caches," in *Proc. 20th Annual Intl. Symp. on Computer Architecture*, May 1993, pp. 169–178.

[16] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee, "Using prime numbers for cache indexing to eliminate conflict misses," in *Proc. 10th Intl. Symp. on High Performance Computer Architecture*, February 2004, pp. 288–299.

[17] A. Agarwal and S. D. Pudar, "Column-associative caches: A technique for reducing the miss rate of direct-mapped caches," in *Proc. 20th Annual Intl. Symp. on Computer Architecture*, May 1993, pp. 179–190.

[18] B. Calder, D. Grunwald, and J. S. Emer, "Predictive sequential associative cache," in *Proc. of the Second Intl. Symp. on High-Performance Computer Architecture*, February 1996, pp. 244–253.

[19] C. Zhang, X. Zhang, and Y. Yan, "Two fast and high-associativity cache schemes," *IEEE MICRO*, vol. 17, pp. 40–49, 1997.

[20] J. Peir, Y. Lee, and W. W. Hsu, "Capturing dynamic memory reference behavior with adaptative cache topology," in *Proc. of the 8th Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998, pp. 240–250.