Virtually Split Cache: An Efficient Mechanism to Distribute Instructions and Data

DYER ROLÁN, Intel Barcelona Research Center, Intel Labs Barcelona, Spain
BASILIO B. FRAGUELA and RAMÓN DOALLO, Universidade da Coruña, Spain

First-level caches are usually split for both instructions and data instead of unifying them in a single cache. Although that approach eases the pipeline design and provides a simple way to independently treat data and instructions, its global hit rate is usually smaller than that of a unified cache. Furthermore, unified lower-level caches usually behave and process memory requests disregarding whether they are data or instruction requests. In this article, we propose a new technique aimed to balance the amount of space devoted to instructions and data for optimizing set-associative caches: the Virtually Split Cache or VSC. Our technique combines the sharing of resources from unified approaches with the bandwidth and parallelism that split configurations provide, thus reducing power consumption while not degrading performance. Our design dynamically adjusts cache resources devoted to instructions and data depending on their particular demand. Two VSC designs are proposed in order to track the instructions and data requirements. The Shadow Tag VSC (ST-VSC) is based on shadow tags that store the last evicted line related to data and instructions in order to determine how well the cache would work with one more way per set devoted to each kind. The Global Selector VSC (GS-VSC) uses a saturation counter that is updated every time a cache miss occurs either under an instruction or data request applying a duel-like mechanism. Experiments with a variable and a fixed latency VSC show that ST-VSC and GS-VSC reduce on average the cache hierarchy power consumption by 29% and 24%, respectively, with respect to a standard baseline. As for performance, while the fixed latency designs virtually match the split baseline in a single-core system, a variable latency ST-VSC and GS-VSC increase the average IPC by 2.5% and 2%, respectively. In multicore systems, even the slower fixed latency ST-VSC and GS-VSC designs improve the baseline IPC by 3.1% and 2.5%, respectively, in a four-core system thanks to the reduction in the bandwidth demanded from the lower cache levels. This is in contrast with many techniques that trade performance degradation for power consumption reduction. VSC particularly benefits embedded processors with a single level of cache, where up to an average 9.2% IPC improvement is achieved. Interestingly, we also find that partitioning the LLC for instructions and data can improve performance around 2%.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*

General Terms: Design, Power, Performance

Additional Key Words and Phrases: Virtually split design, shadow tags, global selector, first-level cache memories

ACM Reference Format:
Rolán, D., Fraguela, B. B., and Doallo, R. 2013. Virtually split cache: An efficient mechanism to distribute instructions and data. ACM Trans. Architec. Code Optim. 10, 4, Article 27 (December 2013), 23 pages.
DOI: http://dx.doi.org/10.1145/2541228.2541234

New Article, Not an Extension of a Conference Paper.
This work is supported by the Xunta de Galicia under project *"Consolidación e Estructuración de Unidades de Investigación Competitivas"* 2010/06 and the MICINN, cofunded by the Fondo Social Europeo, under the grant with reference TIN2010-16735. The authors are also members of the HiPEAC network.
Authors' addresses: D. Rolán, Intel Labs Barcelona, Spain; email: dyer.rolan@intel.com; B. B. Fraguela, and R. Doallo, Depto. de Electrónica e Sistemas, Universidade da Coruña, Spain; email: {basilio.fraguela, ramon.doallo}@udc.es.

1. INTRODUCTION

The different levels of the cache memory hierarchy usually treat in a different way data and instructions. A split cache design for instructions and data is commonly found nowadays in the first level of modern processors. This design is the preferred one, rather than a unified approach, as it is well suited for the design of the processor pipeline so that different stages access different caches, there being no conflicts between them and achieving a better memory bandwidth than a unified cache. Also, the instruction cache design is simplified as it only requires the support of read operations, while a unified or data cache needs hardware to deal with both read and write operations. However, unified approaches, usually found in the lower levels of the memory hierarchy, provide a better use of resources by automatically sharing the cache capacity for both instructions and data, even threads of different cores in shared caches, at the expense of requiring a higher latency and design complexity and limiting the total bandwidth. Also, a first-level unified cache would usually need to be multiported, so that it could process simultaneously both instructions and data requests, which requires more complexity. Furthermore, unified approaches are not aware of the higher locality that instructions usually have compared to data. As a result they are oblivious to the different space requirements that instructions and data could demand. In this article, we propose a middle-way solution aimed to embrace the advantages that both split and unified designs provide and to alleviate their drawbacks.

The rest of this article is organized as follows. The next section describes the motivation and background of our Virtually Split Cache or VSC approach. Section 3 explains two possible implementations of our design. The environment used in our experiments is described in Section 4, and the results and analysis of our proposals are discussed in Section 5. Finally, the last section is devoted to the conclusions and future work.

2. BACKGROUND AND MOTIVATION

Nowadays, the most usual configuration for the first level of the cache memory hierarchy devotes independent caches for both instructions and data. While unified approaches usually provide higher hit rates by automatically sharing resources instead of statically partitioning them, split caches are the preferred configuration, mainly because of the following reasons:

—Different instructions can access the instruction cache, in the fetch stage of the pipeline, and the data cache, usually in the memory stage, at the same time in pipelined processors. Unified approaches would require several access ports and, thus, more complexity to provide the same advantages.
—The instruction cache design may be simpler, as it only needs to perform, ideally, read operations.
—Unified caches of the same aggregated capacity imply higher latencies.

Several designs have appeared in the last years in order to improve performance or reduce power consumption in first-level caches. The ones centered on power optimization usually offer a trade-off in which some performance is lost in exchange for the improved energy consumption. This way, Albonesi [1999] disables some cache ways of an L1 set associative cache during periods of low demand while trying not to exceed a preset performance degradation threshold. An extensive set of dynamic structures are adapted in concert by Dropsho et al. [2002] with the same purpose. The Smart Cache [Sundararajan et al. 2013] can be applied to all the caches in the chip, reconfiguring their size and associativity in order to save energy. These proposals achieve power consumption reductions of between one third and one half of the baseline in the structures they modify, with small performance degradations of around 2% to 5%.

Regarding performance improvement, many techniques focusing on the first-level **data** caches have been proposed. Besides increasing associativity, logically or virtually as pseudo-associative caches do [Agarwal and Pudar 1993; Calder et al. 1996; Zhang et al. 1997], or early evicting dead blocks in order to retain the recency stack data with higher locality [Lai et al. 2001; Liu et al. 2008], other techniques oriented to better distribute the memory references across the cache to reduce conflict misses have also appeared. For instance, the B-Cache [Zhang 2006] tries to balance the accesses to the sets of first-level direct-mapped caches by increasing the decoder length and incorporating programmable decoders and an *ad hoc* replacement policy. More recently, Ros et al. [2012] dynamically chose the best set of bits to index a direct-mapped cache among a larger set of bits. Also, other approaches try to reduce cache access latency by partitioning the first-level data cache in order to place data near those units that are more likely to use them [Racunas and Patt 2003], leveraging software techniques to increase performance [Lu et al. 2003], or merging different designs to reduce power consumption [Huang et al. 2001].

Regarding the first-level **instruction** caches, techniques to increase performance have been focused on code layout optimization [Torrellas et al. 1998] or code reorganization [Kumar and Tullsen 2002] due to the different locality properties and access patterns instructions usually have compared to data.

None of these previous approaches is aware of the underutilized space instructions or data caches may provide. This way, they are unable to balance resources in the first level of the cache memory hierarchy.

On the other hand, unified cache designs are usually found in lower levels of the memory hierarchy, where issue logic restrictions and latency constraints are more relaxed. Many approaches have been proposed in order to increase performance at these levels, both for private and for shared configurations between several cores in multicore processors (CMPs). Recent approaches are mainly oriented to reduce capacity misses [Qureshi et al. 2007], conflict misses [Rolán et al. 2009], or both [Rolán et al. 2012] [Zhan et al. 2010]. There are also techniques specifically oriented to increase performance in shared last-level caches (LLCs) of CMPs, which usually apply partitioning mechanisms [Dybdahl et al. 2006; Qureshi and Patt 2006] to limit the amount of space devoted to each core.

Apparently, there is no previous research on unified caches focused on applying different policies for instructions and data; that is, there are no unified approaches that become aware of the different localities that instructions and data may have. Even more, the design of hybrid approaches with characteristics of shared and split caches has never been considered either.

In order to analyze the behavior of instructions and data in the first level of the cache memory hierarchy, we have performed studies increasing the cache size for both instruction and data first-level caches in order to emphasize the different locality and space requirements both caches have. Figure 1 shows the evolution of the miss rate for both instruction and data caches varying their size at executing 10 benchmarks from the SPEC CPU2006 suite. Further information about the evaluation parameters can be found in Section 5. Starting with an 8KB direct-mapped cache (configuration 1), we keep increasing the size by 8KB, while keeping the number of sets constant, until we reach a 64 KB eight-way cache (configuration 8). Our 32KB four-way baseline instruction and data caches, which will be further described in Section 4, lay in between. Results show that for a first group of benchmarks such as *444.namd*, *471.omnetpp,* or *482.sphinx3*, three ways are enough to achieve a miss rate close to the one obtained using eight ways in the instruction cache; that is, allocating more than three ways in the instruction cache does not provide better performance. In fact, in some benchmarks that we can include in this category, such as *456.hmmer* and *473.astar*, a single cache

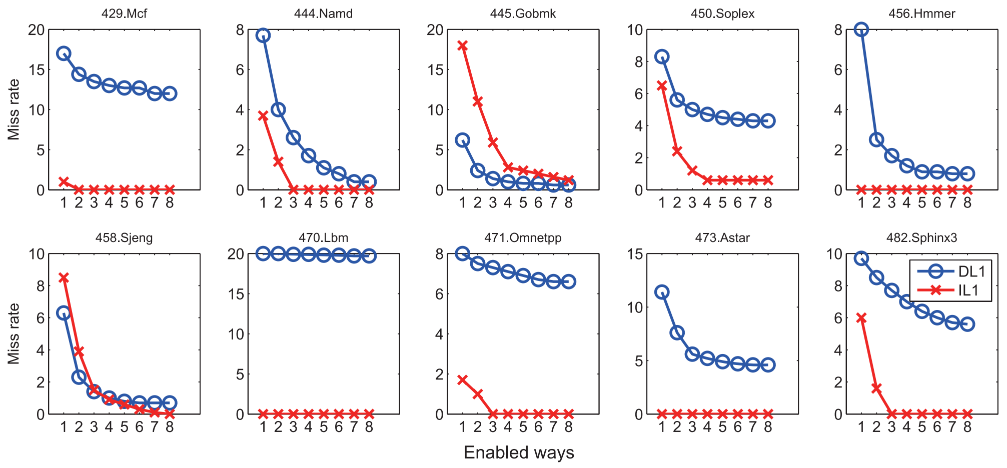

Fig. 1. Miss rate for SPEC CPU2006 benchmarks as the number of allocated ways varies for both instruction and data first-level caches. The X axis shows the number of ways allocated from an eight-way 64KB cache (the remaining ways are disabled).

way suffices for the instruction cache. On the other hand, allocating more than four ways in the data cache usually means a lower miss rate for these benchmarks. This is due to the lower locality and larger working sets that data have related to instructions. Therefore, instruction caches can provide data caches with space in order to better balance the resources of the memory hierarchy for this group of benchmarks. A second group includes those benchmarks where the opposite behavior happens. For instance, during the execution of the *445.gobmk* and *458.sjeng* benchmarks, the instruction cache can benefit from receiving underutilized ways from the data cache from enabling four ways on. Finally, a third group embraces those benchmarks, usually streaming applications, where both instruction and data caches are not hardly influenced by the number of enabled ways because of the poor locality or the great footprint size they have. An example of this behavior can be observed in benchmark *470.lbm*.

3. VIRTUALLY SPLIT CACHE

We propose a new cache design that provides the benefits of a unified approach while not bringing its inconveniences. Our design can be described as a k-way associative cache whose data-store and tag-store are partitioned so that part of them will be devoted to caching instructions while the other part will cache data, giving place to two virtual caches. Each one of these caches has its own port(s), which operate independently. Tag- and data-stores are often partitioned in banks in order to achieve power, latency, and/or bandwidth improvements [Wilton and Jouppi 1996; Sohi and Franklin 1991; Su and Despain 1995; Wada et al. 1992]. In our design both stores are partitioned in k banks each, one per cache way. This way, each bank has as many sets as the cache, but it holds the tag (in the tag-store) or the line (in the data-store) of a single way. When a traditional k-way associative cache with this design is accessed, the k banks of its tag-store are read in parallel so that their content for the selected set is compared with the requested tag. The k banks of the data-store are also read in parallel in the meantime in order to minimize the latency in case one of these comparisons results in a hit, as in that case the data from the corresponding bank will be immediately sent to the processor. Our proposal, called Virtually Split Cache or VSC, divides its k banks, or corresponding ways, in each store in two groups, one for instructions and the other for data, which operate as two independent caches. Since there is a tag-store

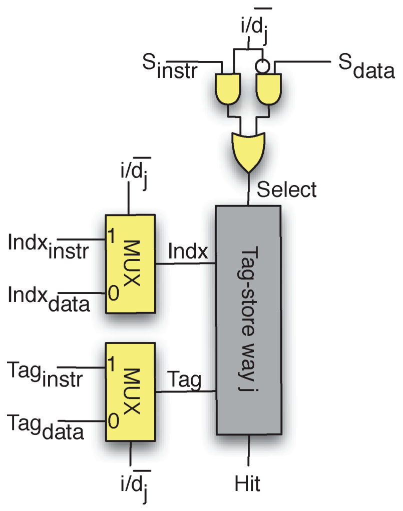

Fig. 2. Way j in the tag-store of the Virtually Split Cache.

bank and a data-store bank per cache way, the pair composed by the ith banks of both stores corresponds to the ith cache way. As a result, in the VSC these two banks form a cache way and are always assigned together to the same virtual cache. The number of ways assigned to each virtual cache is decided dynamically by our design based on an analysis of the behavior and space requirements of the instruction and data streams.

Our design needs some modifications with respect to a standard k-way cache in order to be able to dynamically assign different roles to its banks. Figure 2 shows a tag-store bank (or way) of the VSC. The line $i/\overline{d}_j$ indicates whether way j of the cache is assigned to instructions (value 1) or data (value 0). This signal controls the multiplexers that choose whether the address lines to select the set that must be indexed in the bank (*Indx*) and the tag that must be stored or compared (*Tag*) are those that come from either the instruction or data port. Similarly, it selects the appropriate selection line (S_{instr} or S_{data}) so that the bank will only perform accesses requested by the corresponding port. The modifications in the banks of the data-store of the VSC are analogous. Figure 3 shows the general structure of the VSC. Both the tag-store and the data-store, which is called *data-instr-store* in the figure to outline that it can store both kinds of information, receive all the signals needed to operate each bank, both from the instruction and data ports. They also receive the lines $i/\overline{d}_j, 0 \leq j < k$ so that each one of them controls whether the jth bank of the store is assigned to either instructions or data and operates therefore under the signals from that port. The selection boxes in the lower part of the figure take the k hit/miss lines from the tag-store, the values read from the data-instr-store, and the $i/\overline{d}$ lines that indicate whether each bank is assigned to data or instructions. With this information they can calculate in a straightforward way whether the current instruction (data) access has resulted in a hit or not, and in the first case, select the value from the associated bank and eventually provide it to the processor. Externally this design operates as two independent caches, one for instructions and the other for data, which work in parallel and are made up of ways that can be dynamically assigned to any of them. It is worthy to point out that in the case of an application that demands the same amount of cache resources for data and instructions, the VSC would ideally behave like a split design, thereby not incurring any penalty in terms of latency.

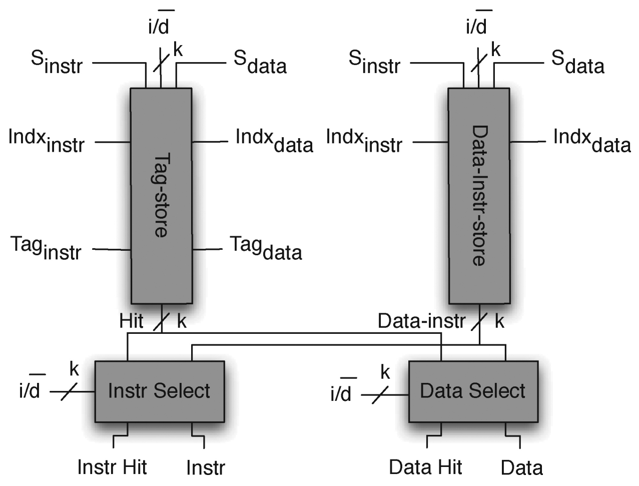

Fig. 3. Virtually Split Cache design.

We have assumed a single bank per way in each store and a single port per virtual cache in order to simplify the explanation. Nevertheless, the banks of the VSC can be further subdivided as long as all the banks associated to the same way are controlled by the signals from the same virtual cache. This way, they could be subdivided to save energy [Su and Despain 1995] or to distribute the sets among interleaved subbanks in order to support multiple ports per virtual cache [Rivers et al. 1997; Sohi and Franklin 1991]. Any of the other usual strategies to implement multiple cache ports [Rivers et al. 1997] could also be applied to the VSC.

Additionally, while in the abstract representation in Figure 3 the tag-store and the data-instr-store have been separately depicted to simplify the picture, the layout of the actual implementations must be designed in order to optimize their latency. This requires placing nearer the tag-store bank(s) and the data-instr-store bank(s) that form each cache way and ensuring that the banks that are associated to each kind of information are those placed nearer to their cache port. We propose a design in which the instruction port and the data port are on opposite sides of the array of banks, with the banks of half of the ways of the VSC being placed nearer to one of the ports with the same disposition, as in a split configuration. As we will see below, we propose that the VSC always assigns to each kind of information a set of consecutive ways. This way, the set of banks associated to instructions and the set associated to data occupy separate nonoverlapping portions on the VSC and they are always the ones nearer to the port that serves their requests, so that the wire delay of each virtual cache of the VSC is minimized. With this layout, when half or less of the ways are associated to a virtual cache, its latency does not exceed that of the equivalent split design. When more than one half of the VSC is assigned to a virtual cache, it uses further banks placed nearer to the other port, resulting in increased latency. The results of this smart design will be presented in Section 5.

The virtual data and instruction VSC caches, even if disjointed, must necessarily be placed together, as they must be able to reallocate banks between them. As a result, the VSC does not allow them to be placed in separate places inside the chip, which is a common strategy to minimize the access latency by the associated pipeline stages. However, there are state-of-the-art processors [Demerjian 2011] that place these caches very near, still providing minimal latency [Riedlinger et al. 2011], so this does not necessarily have a negative impact on performance.

Finally, note that L1 access latency can be dominated by the TLB translation time if we assume a parallel cache-TLB access design [Balasubramonian et al. 2000]. In such a case we could further reduce the latency penalty that the VSC implies in comparison with the split baseline. In this work we are not taking this into account, though.

There remains the issue of the algorithm to allocate ways to the virtual caches. Our design uses a counter I of ways that must be assigned to the instruction cache. All the lines $i/\overline{d}_j$, $0 \leq j < I$ are set to 1, and the ones for $I \leq j < k$ are set to 0. Our approach initially allocates half the ways for instructions and the other half for data. During its operation, the number of ways allocated to each virtual cache varies depending on each particular demand, provided that both caches have always one allocated bank at least. This way, the VSC provides additional benefits in comparison with a unified approach, as it is able to avoid eager behaviors from one of the two types of data. When a bank is reassigned to another virtual cache, its contents are invalidated. This implies that this new allocated line per set is likely to be selected as a victim during the next replacement operation.

We now explain in turn two practical designs of the VSC that use different algorithms to measure the demand of the virtual caches and decide the number of ways assigned to each one.

3.1. Shadow Tag VSC

This first approach tracks the space requirements for both instructions and data by estimating how well both streams would work if they acquired one additional way per set or, globally, one additional bank. Two shadow tags per set are used in order to achieve this. One shadow tag stores the last instruction tag evicted from the set and the other one stores the last data tag evicted from the same set. Each time a miss occurs in a certain set, the appropriate shadow tag, depending on the kind of request, is checked. If a hit occurs in the shadow tag and this is an instruction request, a global counter devoted to instructions is increased. Similarly, we use another global counter for data misses that hit in the data shadow tag as well. Also, as allocating an additional bank to a virtual cache implies deallocating it from the other cache, this approach needs additional structures to predict the impact of depriving a virtual cache of one bank. For this purpose, our design uses two additional counters to track how many hits take place in the least recently used (LRU) position of every set for both instructions and data. Periodically, for each 1 million cache accesses in our experiments, all four counters are checked. If the value of the shadow tag counter for data is greater than the value of the LRU counter for instructions, one bank initially devoted to instructions is allocated for data, and vice versa. The reason is that if the number of hits in shadow tags for data is higher than the number of hits in LRU positions for instructions, allocating one additional bank for data brings more benefits in terms of performance than keeping the same bank devoted to instructions. Analogous conclusions can be obtained for the opposite situation. This strategy assumes the same cost for instruction and data misses, which is not necessarily the case, since instruction misses stall the front end of the pipeline, while data misses stall the back end. For this reason, Section 5.1 studies the impact of prioritizing instruction misses over data misses in our designs. Note that the number of allocated banks for instructions and data remains unchanged if both conditions or none of them are fulfilled. After this process, the four counters are reset. Figure 4 shows the structure of this approach.

3.2. Global Selector VSC

While having a pair of shadow tags per set implies a small storage overhead (see Section 3.3), cheaper alternatives can be explored. We propose a lighter design that

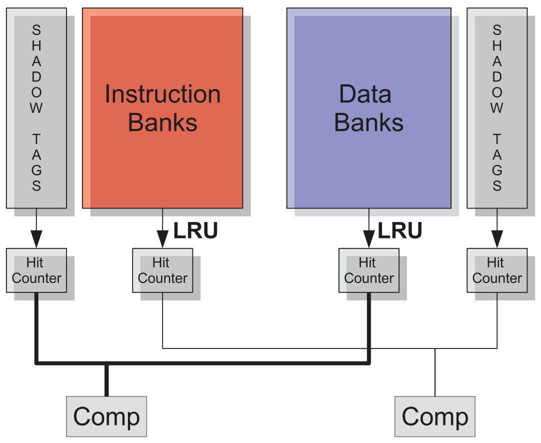

Fig. 4. Shadow Tag VSC.

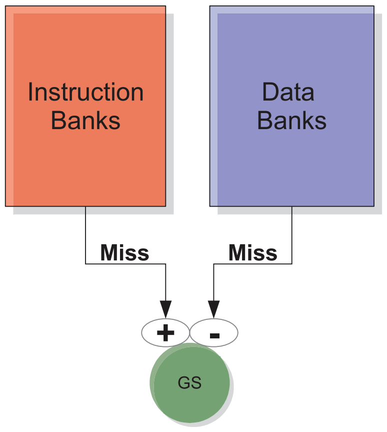

Fig. 5. Global Selector VSC.

achieves similar results using a common saturation counter, or global selector, to make instructions and data fight a duel for resources. This global selector is increased each time a miss occurs for an instruction request and decreased in case of a data miss. The counter value is checked after an update if at least 1 million cache accesses have taken place since the latest bank reallocation. If the global selector holds the maximum value, it means that instructions need more space. One additional bank, used for data hitherto, is allocated for instructions in that case. If the saturation counter has its minimum value, it means that more space for data must be provided. Our approach selects one instruction bank and allocates it for data. The global selector is then initialized with a value in the middle of its range. We have determined experimentally that a good range for the Global Selector counter is between 0 and twice the total number of ways in the cache minus one. As a result, the reset value used is the number of ways in the VSC. Figure 5 shows the structure of this design.

We tested the usage of counters with upper values from the number of ways of the whole VSC cache up to four times it. Similar results were obtained in all the tests, with lower values leading to more frequent reallocations and upper values needing more storage overhead, namely, an additional bit for the counter each time the upper limit is duplicated.

3.3. Cost

In this section we evaluate the cost of both the Shadow Tag and Global Selector Virtually Split Caches in terms of storage requirements. The analysis is based on the baseline and configurations described in Section 4; in particular, the baseline cache is described in Table II. The Global Selector VSC only needs 4 additional bits, assuming an aggregated associativity of 8, for the saturation counter. The Shadow Tag VSC needs two additional tags per set as well as two counters for tracking the hits in the shadow tags, either for instructions or data, and two additional counters for tracking the number of hits in LRU positions. According to our experiments and using the periodicity described in Section 3.1, the counter devoted to the number of hits in the LRU position for data needs 19 bits, while 16 bits are enough for the rest of the counters. Based on this, Table I calculates the storage required by both approaches of the VSC and for a four-way 32KB baseline cache with lines of 64B assuming addresses of 42 bits. Note that we have taken into account the storage cost for both instruction and data caches in the split baseline configuration, which will be further described in the next section.

Table I. Baseline (Instruction cache + Data cache), Global Selector, and Shadow Tag VSC Storage Cost in a 32KB/Four-Way/64B/LRU Cache

	Baseline	Global Selector	Shadow Tag
Tag-store entry:			
State(v+dirty+LRU)	1+1+2 bits	1+1+3 bits	1+1+3 bits
Tag ($42 - \log_2 sets - \log_2 64$)	29 bits	29 bits	29 bits
Size of tag-store entry	33 bits	34 bits	34 bits
Data-store entry:			
Set size	64*4*8 bits	64*8*8 bits	64*8*8 bits
Additional structs per set:			
Shadow Tags	-	-	2*29
Total of structs per set	-	-	58 bits
Total Counters:	-	4 bits	16*3 + 19 bits
Number of tag-store entries	512	1024	1024
Number of data-store entries	512	1024	1024
Number of sets	128	128	128
Size of the tag-store	2.0625KB	4.25KB	4.25KB
Size of the data-store	32KB	64KB	64KB
Size of additional storage	-	4 bits	936B
Total	(I+D) **2*34.0625KB**	**~68.25KB** (~0.1%)	**~69.1KB** (~1.4%)

Table II. Architecture
In the table, RT and TC stand for round trip and tag directory check, respectively.

Processor	
Frequency	4GHz
Fetch/issue	4/4
ROB entries	176
Integer/FP registers	96/80
Memory subsystem	
L1 i-cache & d-cache	32KB/four-way/64 B/LRU/WT
L1 Cache latency (cycles)	2 RT
L2 (unified, inclusive) cache	2MB/eight-way/64 B/LRU/WB
L2 Cache latency (cycles)	14 RT, 6 TC
Main memory latency	62ns

We can see that the storage cost of both approaches is small, even negligible in the case of the Global Selector VSC. We have also calculated that the storage cost of the Shadow Tag VSC could be halved, to about 0.65%, if we reduced the number of bits per tag to 10 bits applying the hash functions proposed in Ramakrishna et al. [1997], since the number of lines that a set can hold, and consequently the number of possible values that a shadow tag can have, is limited.

4. SIMULATION ENVIRONMENT

To evaluate our approach we have used the SESC simulator [Renau et al. 2005] with a baseline configuration based on a four-issue out-of-order core with a hybrid branch predictor scheme and a 16KB stride prefetcher in the L2 cache as well as two on-chip cache levels. First-level caches are single-ported and use a write-through [Butler et al. 2010; IBM 2009; Sinharoy et al. 2005; Le et al. 2007; Sinharoy 2009] policy. Both data cache levels use 32 MSHRs [Kroft 1981], while the instruction L1 uses four. This configuration is detailed in Table II, where the latency of each component of the memory hierarchy has been derived using CACTI [HP Labs 2008].

Table III. Benchmark Characterization

Benchmark	DL1 miss rate	IL1 miss rate	Combined miss rate	CPI
401.bzip2	2.7%	0.001%	1.8%	1.5
429.mcf	13%	0.001	11%	6.5
433.milc	5.7%	0.001%	3.7%	2.6
444.namd	1.7%	0.01%	1%	0.96
445.gobmk	1%	3%	1.7%	1.44
450.soplex	5%	1%	2.7%	2.32
456.hmmer	1.2%	0.003%	1%	2.07
458.sjeng	1%	1%	1%	1.6
462.libquantum	8%	0.0001%	3.1%	2
470.lbm	20%	0.0001%	9.6%	1.51
471.omnetpp	7%	1%	6%	1.4
473.astar	5.2%	0.001%	3.1%	3.1
482.sphinx3	7%	0.01%	2%	1.8

4.1. Benchmarks

We have performed experiments running 13 benchmarks from the SPEC CPU 2006 suite, specifically those with a combined L1 miss rate (considering both data and instruction accesses) of at least 1%. They have been executed using the reference input set (*ref*) during 10 billion instructions after the initialization. Table III characterizes them providing the miss rate for both instruction and data L1 caches, the combined miss rate regarding the number of accesses and misses both caches have altogether, and the CPI obtained with the baseline.

5. PERFORMANCE EVALUATION

We have applied the VSC in the first level of the cache memory hierarchy to evaluate its performance and power consumption. We have compared both versions of the VSC as well as a true dual-ported unified cache (with one port for instructions and another one for data, like the baseline and the VSC) with aggregated capacity of 64 KB and eight ways, with the baseline. The hit latency, calculated with CACTI [HP Labs 2008], for this unified approach is three cycles. As for the VSC, as we explained in Section 3, while it is feasible to get the same latency in cycles as the baseline when a virtual cache has the same or fewer ways than the equivalent portion of a split cache, when more ways are in use, it is fair to assume that a longer latency will be needed. We have conservatively estimated that whenever a VSC virtual cache has even a single more way than the baseline, its latency grows to three cycles. This estimation is conservative because this is the latency that CACTI computes for the dual-ported unified cache, which in fact should tend to be always slower than the VSC for two reasons. First, the VSC and split cache banks are single ported, as they only support a single access per cycle, while the unified cache banks need to be truly dual ported in order to provide a fixed latency for the simultaneous instruction and data accesses, which involve all the banks. This considerably increases the size of these banks [Tatsumi and Mattausch 1999]. The other reason is that the split and VSC cache ports are connected to four and to between one and seven banks, respectively, versus the eight banks the unified cache has always in use. Also, note that the extra hardware the VSC uses to decide its configuration (shadow tags and counters) is small, and more importantly, it is not in its critical path, so it can be placed isolated from the VSC core and it can operate in later stages, particularly as it is only used on misses, or hits in the LRU way, thus not affecting the VSC latency.

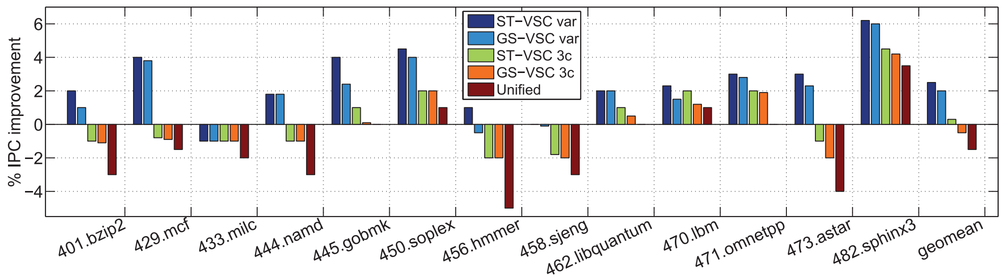

Fig. 6. Percentage of IPC improvement relative to the baseline cache achieved by several VSC designs.

Since VSC could satisfy hits with different latencies depending on the number of ways allocated, we evaluate two versions of each one of the designs proposed in Section 3. The first version is a VSC with variable latency. Using CACTI we have estimated a latency for each one of the two virtual caches provided by the VSC of one cycle when it has one or two allocated banks, two cycles for three or four allocated banks, and three cycles for between five and seven banks. This version requires a pipeline that adjusts to this latency when executing the instructions. There are several research articles [Agarwal and Pudar 1993; Calder et al. 1996; Zhang et al. 1997; Balasubramonian et al. 2000] as well as actual well-known architectures [Tendler et al. 2002] that have different latencies in the L1 cache despite the issue logic restrictions. In these designs, the wake-up logic in the pipeline accurately handles the variable return times for the L1, so that different hit times are supported, in our case depending on the number of banks a virtual cache uses at a given time. Note that for VSC, each virtual cache, and thus its latency, is fixed for millions of cycles (on average, 4.4 million and 9 million for Shadow Tag VSC and Glogal Selector VSC in our tests in Section 5.6, respectively). Thus, the pipeline and the specific wake-up logic only need to be adjusted to a new latency for instruction and data accesses very seldomly, with a warranted lower bound of 1 million accesses, or the minimum policy revision time chosen for the VSC implementation, working the vast majority of the time as a system with fixed cache latency. This way, the variable latency VSC is much more feasible to support than other variable latency approaches in which the hit latency can vary for each access. In our simulations with the variable latency VSC cache reconfigurations are coupled with a restart of the processor pipeline with the same cost as a branch misprediction to account for the reconfiguration cost. The impact on performance is negligible. The second version of the VSC we test is a design with a fixed latency of three cycles, no matter the current internal configuration.

Figure 6 shows the percentage of IPC improvement for the VSC designs and the unified cache relative to the split baseline configuration. The last column shows the geometric mean of the values normalized to 1. The unified cache degrades the IPC relative to the baseline by 1.5% due to its higher latency, even if it has a better miss rate than the baseline, as we will see later. Using variable latencies, our Global Selector VSC (GS-VSC var) improves IPC by 2%; when using the Shadow Tags VSC (ST-VSC var) this percentage increases to 2.5%. With a fixed three-cycle latency none of the VSC approaches noticeably improve over the split baseline design, but they still outperform the unified approach.

Figure 7 shows the miss rates of the unified and VSC designs relative to the combined miss rate of the baseline. The unified cache gets a 6% miss rate reduction, ST-VSC 13%, and finally GS-VSC 11%. The further miss rate reduction of the VSC designs with respect to the unified cache explains why they work better even when they are evaluated with the same three-cycle latency. The reason for the better hit rate is that

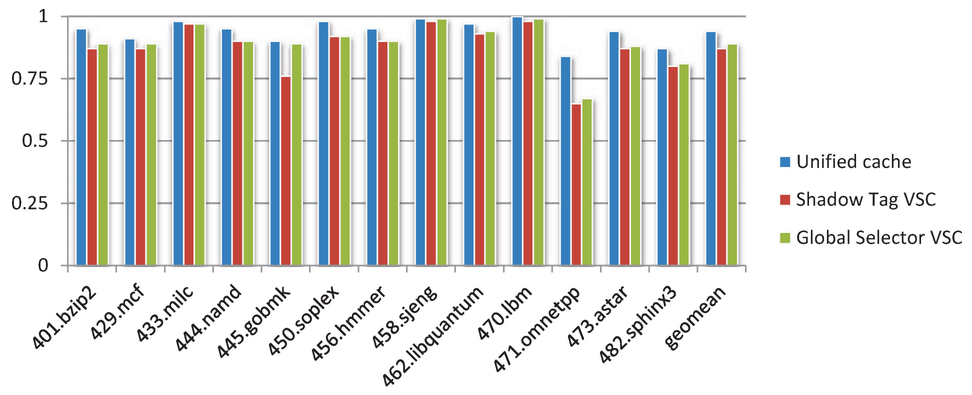

Fig. 7. Miss rate of the unified and VSC designs relative to the baseline cache.

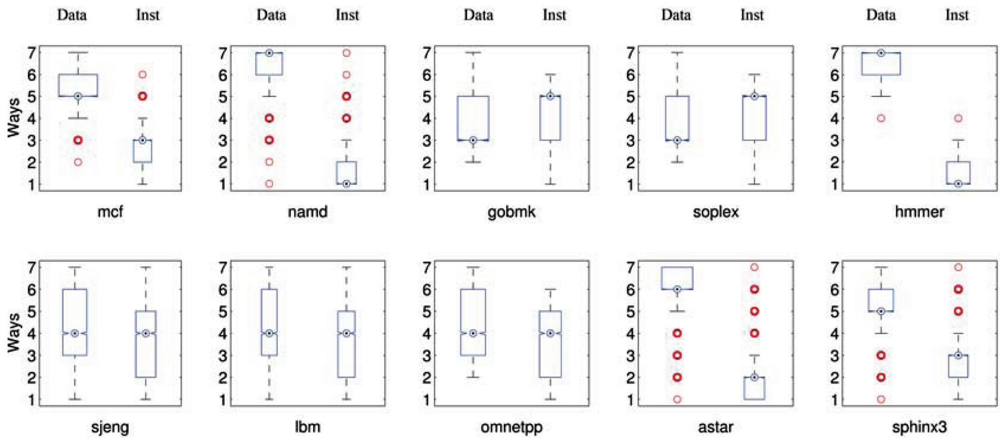

Fig. 8. Distribution of the number of banks allocated for instructions and data using the Shadow Tag VSC (data cache allocation on the left and instruction cache allocation on the right).

VSC, contrary to a unified cache, guarantees a minimum of one way per set both for instructions and for data. We can conclude that our VSC combines the lower latency of split approaches with the higher hit rate of unified caches.

Figure 8 shows a box plot with the distribution of the number of banks allocated by the ST-VSC for both data (column on the left) and instructions (column on the right) during the execution of the benchmarks considered in Figure 1. The number of banks devoted to both instructions and data over time has been sampled each 1,000,000 accesses to the cache, that is, each time a new adjustment is performed in the ST-VSC. The height of each box indicates the degree of dispersion of the data between the 25th percentile or Q1, the bottom of the box, and the 75th percentile or Q3, the top border of the box. The width of each box indicates the size of the sampled population for both instructions and data (e.g., we gathered more samples, since there are more accesses to the cache, in *mcf* than in *sjeng*). The median, or second quartile (Q2), of the population is marked with a black dot inside a circle. It often overlaps the first or the third quartile, meaning that the degree of dispersion is low. Whiskers show both the minimum and maximum samples not considered outliers. Finally, circles represent outliers, which are the observations with a value outside the range $[Q_1 - w(Q_3 - Q_1), Q_3 + w(Q_3 - Q_1)]$ with $w = 1.5$. The most frequent ones are marked with thicker circles. We can observe how the often

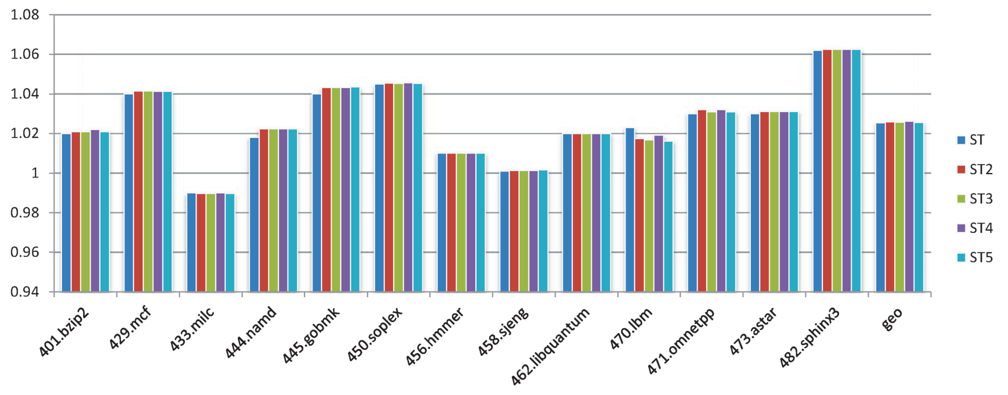

Fig. 9. IPC achieved for different priority levels for instructions in the Shadow Tag VSC relative to the split baseline.

poorer locality and larger working set of data compared to instructions translates into all benchmarks allocating, in a certain moment of the execution, the maximum number of banks allowed by our approach (the associativity minus one) for data. On the other hand, half the benchmarks allocate the maximum number of allowed banks for instructions in a certain moment of time, but most of them are outliers, like when four or more ways are allocated for instructions. As a result, all benchmark executions usually have three or more banks allocated for data, although there are samples with only one or two ways allocated for data in all the benchmarks but *456.hmmer*. These samples are very rare (outliers) in *429.mcf*, *444.namd*, *473.astar,* and *482.sphinx3*, and a little bit more frequent in benchmarks like *458.sjeng* or *470.lbm*. The minimum number of allocated banks for instructions is only one, with two or three in the most common case. Only those benchmarks where the data working set is smaller than the instruction one, like *445.gobmk* or *450.soplex*, or where resources are fairly balanced between data and instructions, like *458.sjeng* or *470.lbm*, do not need to devote more banks for data than for instructions.

5.1. Prioritizing Instruction Hits

As instruction misses have a greater impact on performance than data ones due to the pipeline stalls generated by front-end misses, we have made some experiments adapting our VSC designs in order to favor the reduction of instruction misses over data misses. Namely, we have modified the Shadow Tag and Global Selector VSC to increase the instruction counters (the Shadow Tag counters for instructions in the former case, and the increasing value for the Global Selector after an instruction miss in the latter) with a value two, three, four, and five times greater than in the case of data misses, thus giving more weight to instruction misses. Figures 9 and 10 show the IPC improvement over the split baseline of our original design and the versions with increased instruction miss weight for the Shadow Tag and Global Selector VSC, respectively. We can see that the design that brought the best results for the ST-VSC was the one in which the update values for the instructions counters are four times greater than for the data counters, which outperformed the split baseline design by 2.6% on average. We did not select this option for our ST-VSC baseline as it can mean a higher storage overhead for instruction counters (assuming the reallocation frequency described in Section 3.1). In the GS-VSC, the best design was the one that gave five times more weight to instructions than to data, which achieved a 2.1% improvement but also led to much more often counter saturations, and thus frequent reallocations.

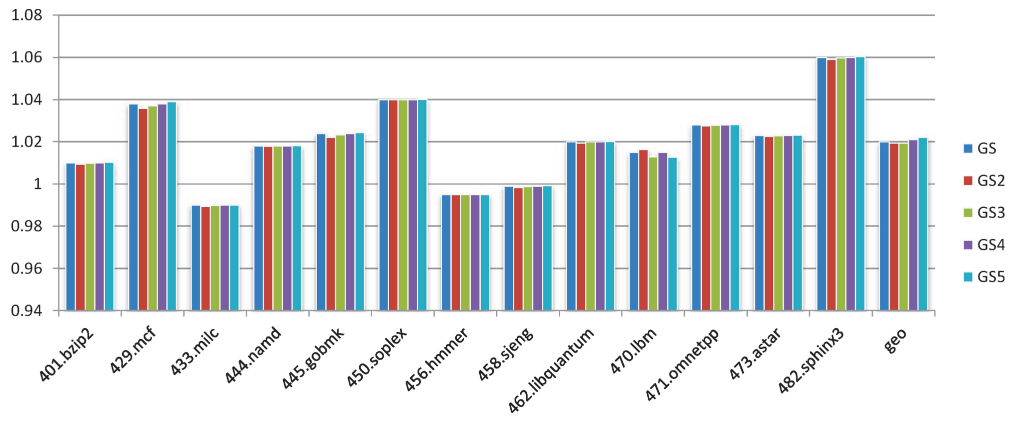

Fig. 10. IPC achieved for different priority levels for instructions in the Global Selector VSC relative to the split baseline.

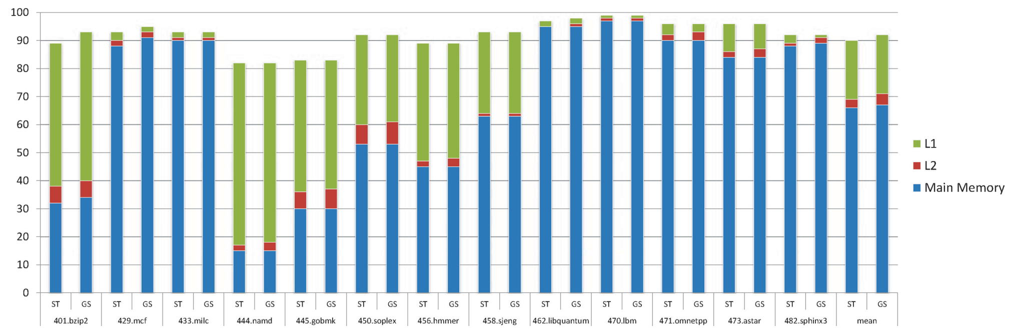

Fig. 11. Energy consumption of the Shadow Tag and Global Selector VSC relative to the split baseline. The total height of each bar indicates the energy consumption of the complete memory hierarchy when using the associated VSC design, relative to that of the baseline (100). The extent of each color inside each bar indicates the relative portion of that energy consumption that took place in the L1 cache, the L2 cache, and the main memory.

5.2. Memory Hierarchy Energy Consumption Analysis

We have analyzed the energy consumption of our approaches relative to the baseline configuration in the memory system. The energy consumption per access in the different levels of the hierarchy has been estimated using CACTI [HP Labs 2008]. Although no access is required for the eight VSC banks, we have conservatively estimated the cost of each VSC access to be the same one as an access to a 64KB eight-way cache. That is, our estimation is the same as if every access to any of the virtual caches of the VSC accessed all its banks, when on average, only half of them are required. Note the fewer banks the VSC needs to access the lower energy consumption. The previous estimation accounts also for the minimal cost of the counters and the eventual shadow tag accesses of the ST-VSC, which only happen under a miss. This costs a total of 0.06nJ versus the 0.04nJ required by an access to one of the four-way 32KB caches of the baseline. Figure 11 shows the relative energy consumption of the full memory hierarchy when using the Shadow Tag and Global Selector VSC relative to the one measured when the baseline split cache is used. Each bar is broken down into the percentage of energy consumption due to hits that are satisfied in the first or second level of the memory hierarchy or in the main memory, out of the total amount of energy consumption. The last column shows the arithmetic mean of the reduction as well as the distribution of

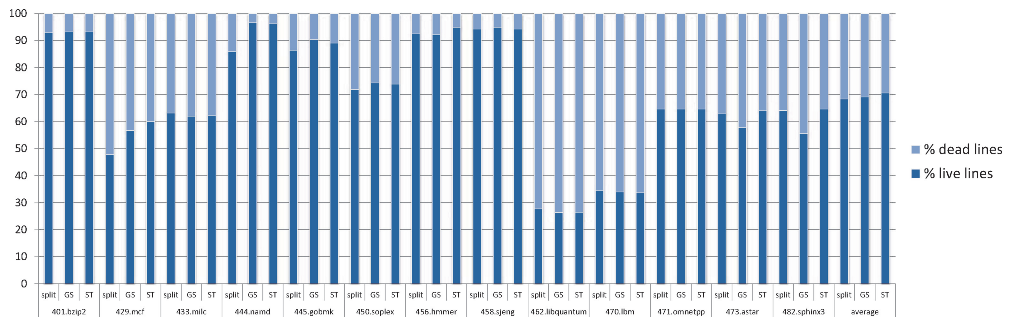

Fig. 12. Average ratio of live and dead lines for Shadow Tag and Global Selector VSC as well as the split baseline.

the accesses. The Shadow Tag VSC achieves 10% energy consumption reduction, while the Global Selector VSC achieves 8%. The reason for the larger reduction achieved by the Shadow Tag VSC is the larger fraction of accesses to the second-level cache that it is able to avoid. If we compute the energy reduction only on the cache hierarchy, as most related works that deal with the reduction of power consumption in caches do [Albonesi 1999; Dropsho et al. 2002; Sundararajan et al. 2013], that is, disregarding the main memory, the savings are 29.4% for ST-VSC and 24.2% for GS-VSC. This way, VSC is always an interesting design point, since even if the fixed latency VSC is used, which offers no performance advantages, power savings on the order of those found in the bibliography are achieved, but with smaller or even no performance degradation observed.

5.3. Cache Utilization and Usefulness

Since the VSC performs invalidations during the reallocation process, a natural question is whether this leads to a lower utilization of the cache than the baseline. A rough indicator of the utilization is the ratio of cache lines that are not invalid; however, a more accurate one is the ratio of live lines. A line is live if it holds information that is going to be reused by the processor before the line is evicted. A line is, however, dead since its last reference until it is evicted (or invalidated) [Lai et al. 2001]. Figure 12 shows the average percentage of live and dead lines for all benchmarks and designs considered during the simulation. The split baseline achieves 67% of live lines, while the Global Selector VSC obtains 69% and Shadow Tag VSC 71%. An important reason for the lower utilization of the baseline is the invalidation of instruction cache lines when manipulating working sets (data plus instructions) that do not fit in the L2 cache of our inclusive memory hierarchy. This way, the VSC keeps more useful lines in the cache than the baseline despite the periodic (but seldom) invalidations it performs in the bank reallocations, which disregard the usefulness of each individual line installed on them. We can also see that the live line ratios correlate with the performance observed in the preceding sections. In conclusion, the VSC is able to better utilize the resources, also keeping more useful lines.

5.4. Embedded System Analysis

Several models of embedded system processors include a single level of cache, which is usually split, and as a result their L1 cache misses go directly to main memory [Freescale Semiconductor 2011; ARM 2010; Freescale Semiconductor 2007]. In this section, we evaluate the benefits of the VSC in these systems. Figure 13 shows a study of the VSC applied in such a baseline architecture by removing the L2 cache in our baseline configuration. In this environment, the variable latency Shadow Tag

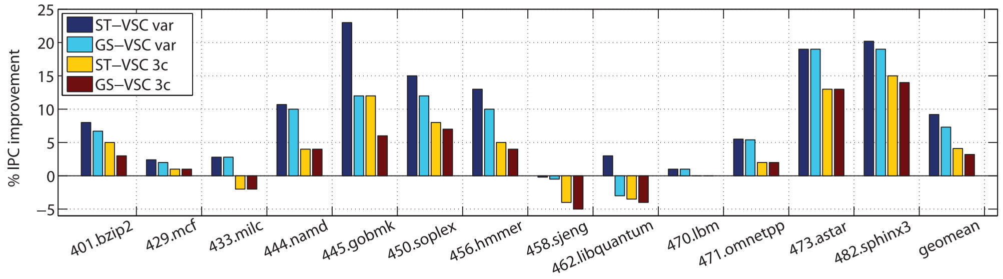

Fig. 13. IPC improvement relative to a baseline architecture without L2 cache.

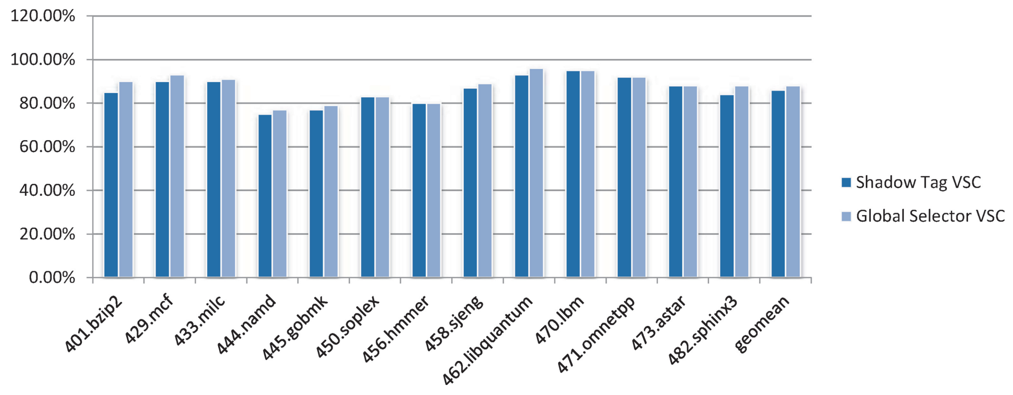

Fig. 14. Power consumption reduction relative to a baseline architecture without L2 cache.

VSC outperforms the baseline by 9.2% on average, while the Global Selector VSC achieved 7.3% improvement. With a three-cycle fixed latency, the average speedups are reduced to a still meaningful 4.1% and 3.2%, respectively. Figure 14 shows the power consumption reduction in this environment for the cache plus the main memory for both VSC designs. The Shadow Tag VSC reduced the global memory hierarchy power consumption by 14% and the Global Selector VSC by 12%, on average. This way the power reduction is about 50% larger than in systems with multiple levels of cache.

5.5. Lower-Level Cache Analysis

The VSC was initially designed for the first level of the cache memory hierarchy, since it is at this level that we can easily identify instructions and data accesses and associate them to different caches with their respective ports. For this reason, it is usually the only level of the memory hierarchy where split cache designs are found, which wastes resources when the requirements for instructions and data memory blocks are unbalanced; this is the main problem the VSC addresses. However, one may wonder what is the effect of applying the ideas of the VSC in lower cache levels, which are always unified caches and thus already allow maximum flexibility for sharing their resources. Although many approaches have been proposed to distribute resources in lower-level caches between different threads [Qureshi and Patt 2006; Xie and Loh 2009], this is not the case for distributing instructions and data.

We applied the VSC approaches described in Sections 3.1 and 3.2 in order to apply a way-partitioning approach for instructions and data in a unified L2 cache. The system configuration was the same used in the preceding experiments, with the only change that the memory hierarchy was made noninclusive in order to simplify the design and

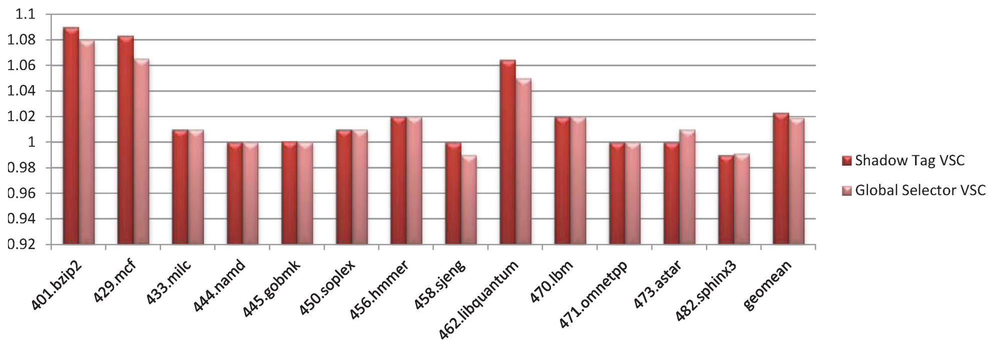

Fig. 15. IPC achieved using a L2 that partitions its resources in each set between instructions and data using the VSC strategies relative to the unmanaged unified baseline.

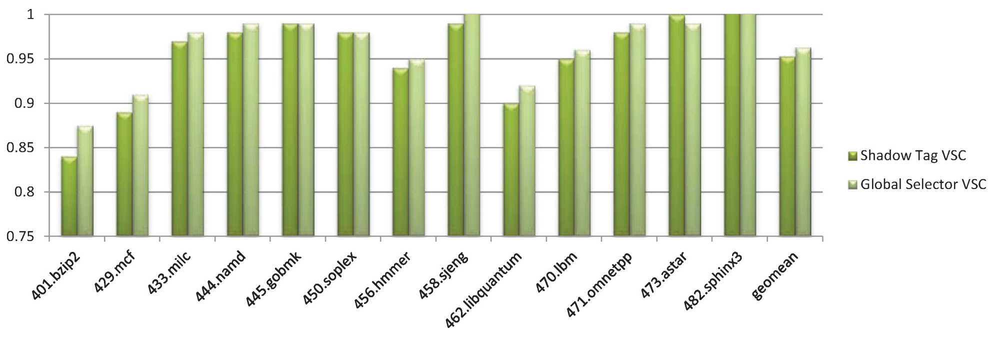

Fig. 16. Miss rate for both VSC designs applied in the LLC relative to the unmanaged unified baseline.

to increase the aggregated capacity of the memory system. Also, the same split L1 cache configuration was used with both the regularly managed L2 and the one that partitions its resources between instruction and data applying the VSC strategies. Figure 15 shows the relative IPC achieved by the modified L2 designs with respect to the baseline. The L2 managed under the Shadow Tag VSC outperforms the baseline by 2.3%, while the one that follows the Global Selector VSC does so by 1.9%. Figure 16 shows the relative miss rate achieved by this partitioning of the L2 with respect to the baseline. The average relative miss rate reduction is about 5% in the case of the Shadow Tag VSC and close to 4% in the case of the Global Selector VSC. As expected, the improvement is smaller than the one achieved in the first level, but still, it outperforms the baseline design, thanks to the fact that our approach is able to better distribute space for instructions and data and to avoid one type of data being deprived of space by the other one. In this environment, the Global Selector VSC, which only needs a counter, seems a very cheap and attractive alternative with respect to the Shadow Tag approach, which needs two shadow tags per set. However, sampling strategies (using shadow tags only in some sets) such as those proposed in Qureshi and Patt [2006] would strongly reduce ST-VSC cost.

5.6. Sensitivity Analysis Using Write-Back Policy

As explained in Section 3, our approach simplifies the reallocation process by invalidating lines in the new acquired bank. Until now, we have considered a write-through policy for the first level. This design is used in many single-core and multicore processors [Butler et al. 2010; IBM 2009; Sinharoy et al. 2005; Le et al. 2007; Sinharoy 2009],

Table IV. Shadow Tag VSC Statistics When Applied in a Write-Back L1 Cache

Benchmark	Average number of dirty lines per reallocation	Percentage of dirty lines in the bank	Average number of cycles between reallocations (thousands)	% IPC improvement over split design
401.bzip2	1.6	1%	2900	1.5%
429.mcf	4	3.1%	9500	4%
433.milc	18	14%	4900	−1%
444.namd	0.78	0.006%	2100	1.5%
445.gobmk	38	29.6%	2800	3.6%
450.soplex	7	5.4%	4700	4.5%
456.hmmer	18	14%	3400	2%
458.sjeng	35	27.3%	3300	−1%
462.libquantum	0.5	0.003%	5100	2.2%
470.lbm	15	11.7%	5100	2.5%
471.omnetpp	11	8.5%	2700	2.5%
473.astar	9	7%	6000	3%
482.sphinx3	3	2.3%	3900	6%
Average	**12**	**9%**	**4400**	**2.4%**

Table V. Global Selector VSC Statistics When Applied in a Write-Back L1 Cache

Benchmark	Average number of dirty lines per reallocation	Percentage of dirty lines in the bank	Average number of cycles between reallocations (thousands)	% IPC improvement over split design
401.bzip2	46	36%	5000	1%
429.mcf	17	13%	2000	3.8%
433.milc	30	23%	3000	−1%
444.namd	98	76%	8000	1.7%
445.gobmk	50	39%	1200	2%
450.soplex	39	31%	1200	3.8%
456.hmmer	51	39%	8200	−0.5%
458.sjeng	39	31%	1300	−0.1%
462.libquantum	47	36%	42000	2.2%
470.lbm	54	42%	11000	1.5%
471.omnetpp	20	16%	1200	2.8%
473.astar	59	46%	30000	2.3%
482.sphinx3	46	36%	3300	6%
Average	**45**	**35%**	**9000**	**1.9%**

as it simplifies the coherence both in the chip, in the case of multicore ones, and with other chips. However, write-back L1 caches are also used in many processors in order to avoid the bandwidth contention produced by write-through policies in the lower cache. In these systems, coherence is usually handled by snooping mechanisms.

In this section we explore the impact of using an L1 cache with write-back policy for the VSC. If we consider an L1 cache with the same parameters as in the preceding sections (64KB of capacity, eight ways, and lines of 64B), each time the VSC performs a reallocation, the 128 lines kept in a bank, one per set, are switched. Tables IV and V show the average number of dirty lines that need to be pushed to the lower level during a reallocation process, and the percentage of the 128 lines in the bank they represent, when write-back Shadow Tag and Global Selector VSCs are used, respectively. The tables also include the average number of cycles between reallocations for each benchmark and the IPC improvement obtained. Notice that the reallocation frequency is independent of whether the cache is write-through or write-back, so this is also the average time between reallocations in the experiments using write-through caches. We can see that the average number of lines to be pushed in the Shadow Tag VSC, 12 out of 128, is small enough to not increase the contention in the lower level. This is particularly true given that, as we can see in the table, reallocations only happen each 4.4 million cycles on average. Thus, the average performance improvement is about 2.4%, quite similar to the 2.5% obtained in the case of the write-through policy. The improvement is slightly smaller because those benchmarks with the highest number of pushed lines per reallocation, like *445.gobmk* or *458.sjeng*, get smaller improvements than in the case of using a write-through policy due to the latency penalty derived from performing these write-backs. The average number of dirty evictions per reallocation is higher in the Global Selector VSC. Still, this does not degrade performance much because the average number of cycles to perform reallocations is more than twice greater than in the case of the Shadow Tag VSC. This lower reallocation frequency is of course directly related to the increase in the number of dirty lines to write back. The performance improvement, 1.9%, is again slightly smaller than the 2% observed in the write-through cache. Note that these results do not mean that the write-back policy performs worse than the write-through one, as we are showing relative numbers to the baseline. In fact, the write-back policy outperforms the write-through policy in our experiments by 0.8%, on average.

5.7. Multicore Experiments

Improving the cache behavior in the first level of the hierarchy, which is the main goal of the VSC, usually brings benefits into the overall system, as lower levels experience less contention. This is particularly significant in multicore processors with a shared last-level cache, where the bandwidth that this latter device can provide diminishes as the number of demanding cores increases. In order to evaluate the overall impact of the VSC in this scenario, we have performed experiments in two different multicore environments, namely, dual and quad-core, configured with a 16-way shared L2 cache of 4 MB (30-cycle hit latency) in case of the dual-core system and 8 MB (40-cycle hit latency) for the quad-core one. We use the same baseline configuration described in Table IV for the cores and L1. We have used the benchmarks listed in Table III to create 16 multiprogrammed workloads of two applications and six of four applications. All these workloads have at least one miss per kilo-instructions (MPKI). Each benchmark is executed until it commits 10 billion instructions after the initialization, with the same reference input set as in the previous experiments. When each core reaches this number of instructions, it continues its execution until the slowest core finishes, in order to keep competing for the cache resources. Figure 17 shows the percentage of IPC improvement relative to the baseline for both the Shadow Tag and Global Selector VSC in the experiments using two cores evaluating both the variable and the fixed latency implementations. Each group of four columns along the X axis is labeled with the number of the two benchmarks that were run in parallel (the correspondence to the benchmark name can be seen in Table III) separated by a plus sign. The last column indicates the geometric mean. The Shadow Tag VSC has a 4% IPC improvement, while the Global Selector VSC averages 3.2% when variable latencies are supported; otherwise, the improvements are 2.2% and 1.6%, respectively.

The experiments with four simultaneous applications are shown in Figure 18 using the same naming scheme for the workloads along the X axis. The variable latency Shadow Tag and Global Selector VSC obtain 4.8% and 3.9% IPC improvement over the

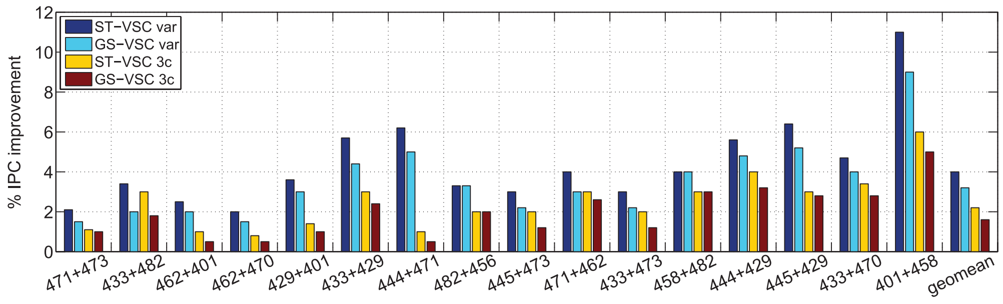

Fig. 17. Percentage performance improvement over the split baseline executing two applications using the Shadow Tag and Global Selector VSC.

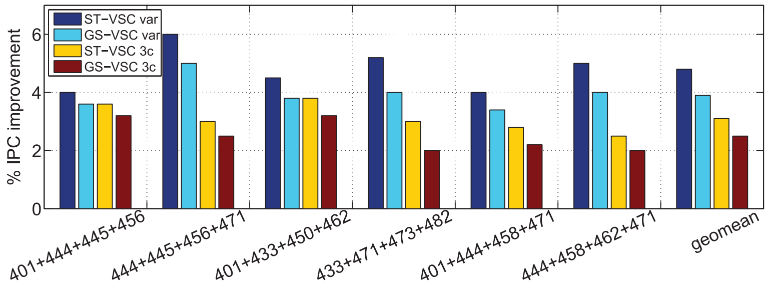

Fig. 18. Percentage performance improvement over the split baseline executing four applications using the Shadow Tag and Global Selector VSC.

baseline, respectively. The fixed three-cycle latency variations provide a 3.1% and 2.5% speedup, respectively.

From these results we can infer that the benefits of VSC grow as the number of cores applying it and sharing the lower level increases. The reason is the increased availability of the shared cache thanks to the smaller number of accesses that it must handle as a result of the usage of the VSC in the L1 of the cores. As a result, in multicore systems even the VSC design with the fixed worst-case latency provides not only power savings but also performance improvements.

6. CONCLUSIONS

We have proposed the Virtually Split Cache (VSC), a new design that provides higher hit rates than split configurations and even unified approaches by balancing resources between the two kinds of information found in the cache memory hierarchy: instructions and data. It can also provide lower latencies than a unified cache by (1) specifically devoting each cache bank only either to data or to instructions, which eliminates the need for truly multiporting them in order to support simultaneous data and instruction accesses, and thus reduces their size; (2) limiting the maximum number of cache banks that can be devoted to each kind of information; and (3) assigning to each kind of information only the banks that are nearer to the corresponding port of access. As far as we know this is the first approach to look at the varying capacity needs for instructions and data within an application and to allocate resources for both depending on their demand.

Two alternative designs to track the different requirements that instructions and data demand have been proposed, the Shadow Tag VSC (ST-VSC), which uses shadow tags to decide whether assigning one more bank for instructions or data increases performance, and the Global Selector VSC (GS-VSC), which uses a common saturation counter to make instructions and data fight a duel for resources. Both designs have a very small hardware overhead, and they have been evaluated both using variable latencies depending on the number of allocated ways and using a fixed latency identical to that of a dual-ported unified cache of the same global size.

Our experiments indicate that the VSC can reduce the power consumption in the cache hierarchy on average about 29% for the ST-VSC and 24% for the GS-VSC. Contrary to other techniques that provide energy savings, this does not translate into performance degradation. In fact, while the fixed latency VSCs match the performance of the baseline, the variable latency ST-VSC and GS-VSC improve the IPC around 2.5% and 2% on average, respectively. Prioritizing instruction misses over data misses, given their higher impact of performance, slightly improves the performance of the basic VSC designs evaluated. Furthermore, the advantages of the VSC grow with the number of cores that share the LLC thanks to the reduction in the contention of the shared cache levels it achieves. This way, the ST-VSC outperformed a four-core baseline configuration by 4.8% on average in terms of throughput when using variable latency and by 3.1% with fixed latency, the corresponding values for GS-VSC being 3.9% and 2.5%, respectively. Not surprisingly, embedded systems with a single level of cache particularly benefit from the VSC, achieving 50% more power savings and average performance improvements between 3.2% and 9.2%. Finally, we found that reserving a specific number of lines for instructions and for data in each set of a lower-level cache using the policies we have proposed to manage the VSC also has a small positive impact on performance.

Future directions for research include studying the behavior in terms of performance at a set level, or even finer granularities like line level, as well as exploring other metrics and mechanisms to track the instructions and data requirements.

REFERENCES

Agarwal, A. and Pudar, S. D. 1993. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA'93)*. 179–190.

Albonesi, D. H. 1999. Selective cache ways: On-demand cache resource allocation. In *Proceedings of the 32nd Annual International Symposium on Microarchitecture (MICRO'99)*. 248–259.

ARM. 2010. Cortex A5: Technical Reference Manual.

Balasubramonian, R., Albones, D., Buyuktosunoglu, A., and Dwarkadas, S. 2000. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-33)*. 245–257.

Butler, M., Barnes, L., Sarma, D. D., and Gelinas, B. 2010. Bulldozer: A new approach to multithreaded compute performance. In *Proceedings of the Symposium on High Performance Chips (Hot Chips 22)*.

Calder, B., Grunwald, D., and Emer, J. S. 1996. Predictive sequential associative cache. In *Proceedings of the 2nd International Symposium on High-Performance Computer Architecture*. 244–253.

Demerjian, C. 2011. Intel updates Itanium architecture with Poulson. http://semiaccurate.com/2011/02/21/intel-updates-itanuim-architecture-poulson/. Accessed: 12 Feb 2013.

Dropsho, S., Buyuktosunoglu, A., Balasubramonian, R., Albonesi, D. H., Dwarkadas, S., Semeraro, G., Magklis, G., and Scott, M. L. 2002. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques (PACT'02)*. 141–152.

Dybdahl, H., Stenström, P., and Natvig, L. 2006. A cache-partitioning aware replacement policy for chip multiprocessors. In *Proceedings of the 13th International Conference on High Performance Computing (HiPC'06)*. 22–34.

Freescale Semiconductor. 2007. MCF548x ColdFire Microprocessor.

Freescale Semiconductor. 2011. PowerPC 603e RISC Microprocessor Family: PID7t-603e Hardware Specifications.

HP Labs. 2008. CACTI 6.5. http://www.hpl.hp.com/research/cacti/.

Huang, M. C., Renau, J., Yoo, S.-M., and Torrellas, J. 2001. L1 data cache decomposition for energy efficiency. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design (ISLPED'01)*. 10–15.

IBM. 2009. IBM System z10 Enterprise Class Technical Guide.

Kroft, D. 1981. Lockup-Free Instruction Fetch/Prefetch Cache Organization. In *Proceedings of the 8th Annual Symposium on Computer Architecture (ISCA'81)*. 81–88.

Kumar, R. and Tullsen, D. M. 2002. Compiling for instruction cache performance on a multithreaded architecture. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'02)*. 419–429.

Lai, A.-C., Fide, C., and Falsafi, B. 2001. Dead-block prediction & dead-block correlating prefetchers. *SIGARCH Comput. Archit. News 29*, 2, 144–154.

Le, H. Q., Starke, W. J., Fields, J. S., O'Connell, F. P., Nguyen, D. Q., Ronchetti, B. J., Sauer, W. M., Schwarz, E. M., and Vaden, M. T. 2007. IBM POWER6 microarchitecture. *IBM J. Res. Dev. 51*, 6, 639–662.

Liu, H., Ferdman, M., Huh, J., and Burger, D. 2008. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'08)*. 222–233.

Lu, J., Chen, H., Fu, R., Hsu, W.-C., Othmer, B., Yew, P.-C., and Chen, D. y. 2003. The performance of runtime data cache prefetching in a dynamic optimization system. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecutre (MICRO'03)*. 180–190.

Qureshi, M. K., Jaleel, A., Patt, Y. N., Steely Jr, S. C., and Emer, J. S. 2007. Adaptive insertion policies for high performance caching. In *Proceedings 34th International Symposium on Computer Architecture (ISCA'07)*. 381–391.

Qureshi, M. K. and Patt, Y. N. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 423–432.

Racunas, P. and Patt, Y. N. 2003. Partitioned first-level cache design for clustered microarchitectures. In *Proceedings of the 17th International Conference on Supercomputing (ICS '03)*. 22–31.

Ramakrishna, M. V., Fu, E., and Bahcekapili, E. 1997. Efficient hardware hashing functions for high performance computers. *IEEE Trans. Comput. 46*, 12, 1378–1381.

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

Riedlinger, R. J., Bhatia, R., Biro, L., Bowhill, B., Fetzer, E., Gronowski, P., and Grutkowski, T. 2011. A 32nm 3.1 billion transistor 12-wide-issue Itanium processor for mission-critical servers. In *2011 IEEE International Solid-State Circuits Conference (ISSCC'11)*. 84–86.

Rivers, J. A., Tyson, G. S., Davidson, E. S., and Austin, T. M. 1997. On high-bandwidth data cache design for multi-issue processors. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'97)*. 46–56.

Rolán, D., Fraguela, B. B., and Doallo, R. 2009. Adaptive line placement with the Set Balancing Cache. In *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 529–540.

Rolán, D., Fraguela, B. B., and Doallo, R. 2012. Adaptive set-granular cooperative caching. In *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA'12)*. 213–224.

Ros, A., Xekalakis, P., Cintra, M., Acacio, M. E., and García, J. M. 2012. ASCIB: Adaptive selection of cache indexing bits for removing conflict misses. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*. 51–56.

Sinharoy, B. 2009. POWER7 multi-core processor design. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 1–1.

Sinharoy, B., Kalla, R. N., Tendler, J. M., Eickemeyer, R. J., and Joyner, J. B. 2005. POWER5 system microarchitecture. *IBM J. Res. Dev. 49*, 4, 505–522.

Sohi, G. S. and Franklin, M. 1991. High-bandwidth data memory systems for superscalar processors. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'91)*. 53–62.

Su, C.-L. and Despain, A. M. 1995. Cache design trade-offs for power and performance optimization: A case study. In *Proceedings of the 1995 International Symposium on Low Power Design (ISLPED'95)*. 63–68.

Sundararajan, K. T., Jones, T. M., and Topham, N. P. 2013. The Smart Cache: An energy-efficient cache architecture through dynamic adaptation. *Int. J. Parallel Program. 41*, 2, 305–330.

Tatsumi, Y. and Mattausch, H. J. 1999. Fast quadratic increase of multiport-storage-cell area with port number. *Electron. Lett. 35*, 25, 2185–2187.

Tendler, J. M., Dodson, J. S., Fields, Jr., J. S., Le, H., and Sinharoy, B. 2002. POWER4 system microarchitecture. *IBM J. Res. Dev. 41*, 1, 5–25.

Torrellas, J., Xia, C., and Daigle, R. L. 1998. Optimizing the instruction cache performance of the operating system. *IEEE Trans. Comput. 47*, 12, 1363–1381.

Wada, T., Rajan, S., and Przybylski, S. A. 1992. An analytical access time model for on-chip cache memories. *IEEE J. Solid-State Circuits 27*, 8, 1147–1156.

Wilton, N. and Jouppi, N. 1996. CACTI: An enhanced cache access and cycle time model. *IEEE J. Solid-State Circuits 31*, 5, 677–688.

Xie, Y. and Loh, G. H. 2009. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA'09)*. 174–183.

Zhan, D., Jiang, H., and Seth, S. C. 2010. STEM: Spatiotemporal management of capacity for intra-core last level caches. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'10)*. 163–174.

Zhang, C. 2006. Balanced cache: Reducing conflict misses of direct-mapped caches. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06)*. 155–166.

Zhang, C., Zhang, X., and Yan, Y. 1997. Two fast and high-associativity cache schemes. *IEEE MICRO 17*, 40–49.

Received March 2012; revised June 2013; accepted July 2013