J Supercomput (2015) 71:67–86
DOI 10.1007/s11227-014-1288-5

PS-Cache: an energy-efficient cache design for chip multiprocessors

Joan J. Valls · Alberto Ros · Julio Sahuquillo · Maria E. Gomez

Published online: 13 September 2014

Abstract Power consumption has become a major design concern in current high-performance chip multiprocessors, and this problem exacerbates with the number of core counts. A significant fraction of the total power budget is often consumed by on-chip caches, thus important research has focused on reducing energy consumption in these structures. To enhance performance, on-chip caches are being deployed with a high associativity degree. Consequently, accessing concurrently all the ways in the cache set is costly in terms of energy. This paper presents the PS-Cache architecture, an energy-efficient cache design that reduces the number of accessed ways without hurting the performance. The PS-Cache takes advantage of the private-shared knowledge of the referenced block to reduce energy by accessing only those ways holding the kind of block looked up. Experimental results show that, on average, the PS-Cache architecture can reduce the dynamic energy consumption of L1 and L2 caches by 22 and 40 %, respectively.

Keywords Chip multiprocessors · Cache memories · Power consumption · Multithreaded applications

J. J. Valls (✉) · J. Sahuquillo · M. E. Gómez
Department of Computing Engineering, Universitat Politècnica de València,
Valencia, Spain
e-mail: joavalmo@fiv.upv.es; joavalmo@fiv.upv.com

J. Sahuquillo
e-mail: jsahuqi@disca.upv.com

M. E. Gómez
e-mail: megomez@disca.upv.com

Alberto Ros
Department of Computer Engineering, Universidad de Murcia, Murcia, Spain
e-mail: aros@ditec.um.es

1 Introduction

As silicon resources become increasingly abundant, core counts grow rapidly in successive chip multiprocessor (CMP) generations. These CMPs usually implement the shared-memory programming model and accelerate their access to memory using one or more levels of private caches per core, which are made transparent to software by means of a cache coherence protocol. Two main types of coherence protocols are being implemented in modern CMPs: directory-based protocols and snoop-based protocols, which target different system scales.

Caches in current CMPs are organized in a two- or three-level cache hierarchy, which makes them responsible of a significant percentage of the overall CMP die area [1], as well as a large portion of the overall power budget. Concerning dynamic energy consumption, the first levels of caches are more frequently accessed than last level caches (LLCs); however, accesses to these caches are more energy consuming due to the higher number of ways in LLCs. This concern is even more important in CMPs than in monolithic processors since caches can be accessed both from the processor side and from the interconnection network side (i.e., coherence requests), increasing the number of cache accesses.

Due to performance reasons, caches are deployed with a high associativity degree in current processors (e.g., Intel Core i7 [2] and IBM POWER 7 [3]). In high-performance microprocessors, all the ways in the corresponding set are concurrently accessed on each cache operation. Therefore, the associativity degree defines the number of tags that are looked up in parallel in each cache access. Caches include one comparator per way and perform as many tag comparisons as number of ways. As a consequence, the dynamic energy dissipated per access increases with the cache associativity. In this paper, we focus on reducing the number of ways that must be looked up on each access with the aim of reducing the dynamic energy consumption of the memory hierarchy without degrading the system performance.

Many cache energy reduction approaches have focused on monolithic processors (e.g., Cache Decay [4], Drowsy Caches [5], Way Guard [6]) in the past. Some of them (e.g., [7]) were originally designed to reduce cache access time, but posterior research has proven that these schemes allow important energy savings. However, these schemes are not directly applied to CMPs. Recent research has dealt with energy savings on CMPs running parallel workloads. Parallel workloads represent an important segment for current and future CMPs mainly when many-core processors are considered. The accessed blocks in these workloads can be classified in two different categories: *private* blocks and *shared* blocks. The former are accessed only by a single core, while the latter are accessed by several cores. In this paper, we take advantage of this classification to access only a subset of the set ways on each cache access and reduce the dynamic energy consumption of the memory hierarchy.

Recent research has concentrated on proposals that take advantage of the mentioned private-shared block access behavior to enhance the performance [8–11] or the energy consumption [12–14] of different CMP components. In this paper, we present an energy-aware cache design that makes use of such behavior to address dynamic energy consumption in cache memories.

In short, this paper proposes a new cache architecture for CMPs, named as PS-Cache, with the aim of reducing dynamic energy consumed by the cache structure. The PS-Cache can be applied to any level of the cache hierarchy. Nevertheless, to focus the research, results are presented and analyzed for L1 and L2 caches, in particular, private L1 caches and shared L2 caches. The PS-Cache is based on tagging cache blocks at run-time as shared or private with a simple classification mechanism, similar to the one proposed by Cuesta et al. [9]. Upon an access to the PS-Cache, only those lines having the same type as the requested block are accessed. The PS-Cache is deployed with minimal hardware complexity and presents important design features: (i) block classification is done with negligible complexity, just a single bit must be added to each TLB entry; (ii) tags and data arrays can be accessed in parallel so no performance degradation will rise, which is a major concern in L1 caches; and (iii) unlike other approaches, such as [14], no way-alignment across cache sets must be done.

Experimental results show that the PS-Cache architecture can reduce the L1 cache dynamic energy consumption by about 22 % for both snoopy and directory protocols, and that it can reduce the L2 cache dynamic energy consumption by 40 %. These reductions in energy consumption are achieved without degrading performance. We also compare our PS-Cache against previously proposed designs showing that the PS-Cache provides the best trade-off between performance and energy when implemented in L2 caches.

The remainder of this work is organized as follows. Section 2 describes the main reasons that motivate this work. Section 3 discusses the related work. Section 4 presents the design of the PS-Cache. Section 5 describes the simulation framework. Section 6 analyzes the behavior of both types of blocks and presents the simulation results. Finally, Sect. 7 draws some concluding remarks.

2 Motivation

Two main reasons led us to deal with this work. First, memory reference instructions represent a significant percentage of the executed instructions. Second, when running multithreaded workloads, in addition to access the local cache, other caches (e.g., remote caches) can be accessed for coherence purposes.

The first reason states that cache memories are frequently accessed. We launched experiments to quantify the percentage of memory reference instructions in the studied workloads. Figure 1 shows the percentage of memory reference instructions in each individual benchmark executed on a 16-core CMP system[1]. This value is roughly the same, around 20 %, across the different benchmarks.

The second reason indicates that when running parallel workloads, the number of accesses to the cache increases with respect to monolithic processors because of coherence requests issued by other cores. In other words, the cache is not only accessed from the processor side, but also from the interconnection network (NoC) side, therefore, increasing the dynamic power consumption. In this context, the number of accesses

[1] Experimental environment, system parameters, protocols and cache hierarchy are described in Section 5.

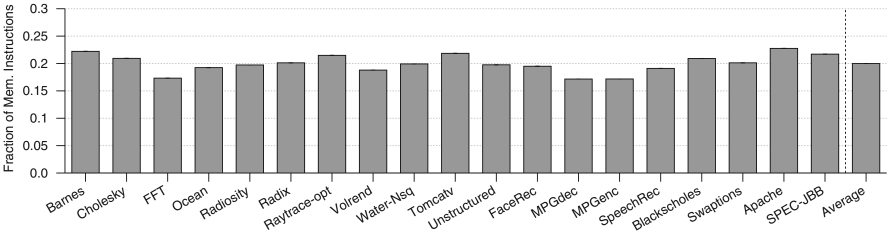


Fig. 1 Fraction of memory instructions

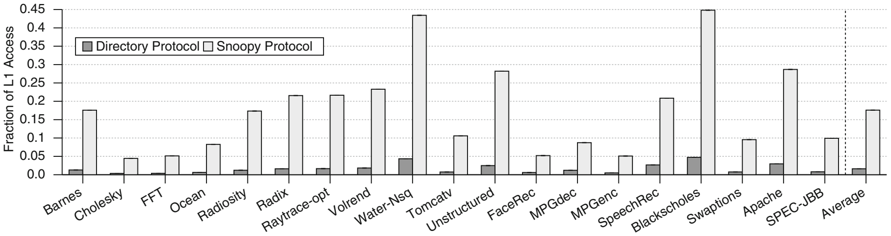


Fig. 2 L1 Coherence lookups

coming from the NoC strongly depends on the type, snoop-based or directory-based, of the underlying coherence protocol.

Snoop-based protocols are based on broadcasting coherence requests to all the cores, which require high bandwidth and energy consumption at the network and also at the caches, since all caches in the system are accessed on a coherence request. Thus, they are only appropriate for small system scales [12,15,16]. Directory-based protocols keep track of the various copies of cached blocks in a directory structure between the private and the shared cache levels [9,17,18]. This allows the processor to easily identify replicas of a block, so minimizing the coherence communication. Coherence requests are only sent to cores storing a replica, so only a subset of caches is looked up. This makes them more suitable for large-scale CMPs, since they reduce energy and bandwidth with respect to snoop-based protocols.

Figure 2 shows the fraction of cache accesses coming from the bus side in both types of protocols in our 16-core system. As observed, this value is noticeable in snoopy protocols and it represents around one fifth of the total accesses; moreover, in some workloads this value is as high as 45 %. In contrast, this value presents a scarce interest in directory-based protocols.

The previous discussion illustrates the importance of reducing dynamic power consumption in caches of CMP systems, and in snoop-based protocols as well, where more coherence requests are issued by the cache controllers. To deal with this problem, this paper proposes an architectural approach with the aim of taking advantage of the behavior exhibited by parallel workloads.

In particular, previous research has taken advantage of the private-shared block classification in parallel workloads running in CMP systems to enhance different aspects in the design of CMPs. The aim of this paper is to save energy by reducing the number of lookups on each cache access. More precisely, the proposal saves significant energy by looking up only those set ways having the same type (shared or private) as the block requested either by the processor or by the coherence access.

3 Related work

This paper presents an energy-efficient cache design that takes advantage of a private-shared detection of the blocks referenced by applications. To the best of our knowledge, there is no prior work on using private-shared classification for dynamically reducing the number of accessed ways on each cache access. Hence, this section first reviews some related work about energy-efficient cache designs, and then, it discusses some other optimizations based on a private-shared detection.

3.1 Energy-efficient cache designs

Caches consumption comes from both leakage (or static) and dynamic consumption. Regarding leakage reductions, Powell et al. [19] proposed a Gated-Vdd technique that aims to reduce leakage for instruction caches by reconfiguring them and turning off unused lines. Kaxiras et al. [4] proposed the Cache Decay, an approach that reduces the leakage power of processor caches by turning off those cache lines that are predicted to be dead, i.e., not referenced by the processor before they are evicted. Alternatively, Flautner et al. [5] exploited the fact that in a particular period of time only a subset of the cache lines is accessed to propose the Drowsy Caches. Different from the previous proposals, the voltage is reduced, but not cut off, for those cache lines that are not being accessed. In this way, the content of the cache line is not preserved.

While techniques that aim to save leakage focus on reducing (or cutting off) voltage, dynamic energy-saving techniques try to minimize the number of data read and written on every cache access. For example, Albonesi [20] proposed Selective Cache Ways, a cache design that enables only a subset of the cache ways when the cache activity is not high. The prediction of ways was previously proposed by Calder et al. [7] to reduce the access time of set-associative caches. This approach works well with L1 caches with a relatively small associativity which present very predictable access patterns; however, as we show in the evaluation, it presents poor results for lower level caches since locality is hidden by previous cache levels.

Zhang et al. [21] proposed the way-halting cache that filters lines accessed in the corresponding set by comparing during the index decoding the four least significant bits of the tag. This approach performs a fully associative search in the first comparison which negatively affects power consumption, but still it is orthogonal to our proposal and could be combined to increase the benefits of both. One possible combination could be using less bits in the first tag comparison to reduce the fully associative look-up and combining it with the PS bit to increase its effectiveness.

Ghosh et al. [6] proposed Way Guard, a mechanism for large set-associative caches that employs bloom filters to reduce dynamic energy by skipping the look-up of cache ways that do not contain the requested data according to the bloom filter. This scheme requires the addition of a large decode and a two fields per way: one segmented bloom filter, previously proposed by the same authors to filter accesses to the whole cache

[22], and another bloom filter to filter accesses to ways. This may result in excessive overhead and critical complexity. Way Guard shows improvements with respect to the way-halting cache. A quantitative comparison with Way Guard is shown in the evaluation section.

Other techniques focus on reducing both leakage and dynamic consumption, for example, by reducing the area of the cache tags, like in the TLB Index-Based Tagging [23], or by performing run-time partitioning, like in the Cooperative Caching scheme [14] or in the ReCaC scheme [24]. There are also some recent works addressing the reduction of cache energy consumption in fault tolerance systems with high reliability [25].

3.2 Private-shared optimizations

The detection of private and shared data can also be employed to optimize performance and power. Kim et al. [12] detect the sharing degree of memory pages to reduce the fraction of snoops in a token-based protocol. In this way, they can replace broadcast messages with multicast ones, thus reducing the energy consumption of the interconnection. The R-NUCA approach by Hardavellas et al. [8] detects private and read-only pages and maps them efficiently in a distributed NUCA cache. By mapping private pages to the closest NUCA bank to the core accessing them, the access latency is reduced, but also the amount of traffic generated, which impacts in the energy consumed by the network. Cuesta et al. [9] deactivate coherence for private pages, thus avoiding their tracking by directory caches. This enables to reduce the directory size up to eight times while still maintaining performance. Reductions in directory area also leads to reduce both dynamic and leakage consumption.

Some previous proposals rely on the compiler and/or memory allocator to classify memory pages in either remove coherence for private pages [26] or improve data placement [27,28]. In [27], a data ownership analysis of memory regions is performed at compilation time. This information is transferred to the page table by modifying the behavior of the memory allocator by means of hooks. This proposal is further improved in [28] by considering a new class of data, named as practically-private, which is mapped to the NUCA cache according to a first-touch policy. In [26], private data are not stored at the LLC with the aim of avoiding cache thrashing for private blocks. Different from our approach, these works mark statically data as private either by the memory allocator or at compile time, when privacy of some data cannot be guaranteed.

SWEL [10] is a novel hardware-based coherence protocol that uses a private-shared block classification at the directory to allocate shared read–write blocks only at the shared LLC, so removing the need of coherence maintenance for them. The main drawback of that proposal is the latency penalization of accessing shared read–write blocks, which must be served by the LLC cache. POPS [11] decouples data and coherence information in the shared LLC to reduce access latency to this information and to improve the aggregate NUCA capacity. It also employs a directory private-shared classification (this time with the help of a predictor table). Spatio-temporal Coherence Tracking [29] also classifies private and shared data at the directory, accounting for temporal private data. It tries to find large private regions to merge them in the directory to save directory space. Ros and Kaxiras [13] proposed VIPS, a complexity-efficient coherence protocol that employs write-back caches for private data for efficiency reasons and write-through caches for shared data for protocol simplicity.

In [30], the proposed L1 Collective Cache [30] uses two separate cache structures in the first-level cache to differentiate between shared and private accesses. The main goal of this approach is to reduce network communications. However, both caches are accessed in parallel so no energy savings are provided. In addition, the fixed size of each cache structure makes the scheme unable to adapt to the multiple behaviors exhibited by the different workloads. RECAP [31] focuses on reducing power consumption in the LLC. This scheme adds an access permission register or APR, a field similar to a sharer vector to each way. The register includes one bit per core to indicate whether a core has access permission to the associated way, resulting in an important area and energy overhead. It uses the information in the APR to access only those ways which the core has access permission to.

Finally, Valls et al. [32] proposed a two-level cache directory organization, named as PS-Dir. Entries keeping track of Private blocks are stored in a large but narrow cache, and an independent, much smaller and wider cache is used to keep track of shared blocks.

4 The PS-Cache

The main goal of the PS-Cache is to take advantage of the classification of private (P) and shared (S) blocks to design a power-efficient cache architecture that is able to reduce the number of ways looked up on each cache access. Instead of accessing all the ways in the corresponding set, as usually done, the PS-Cache only looks up a subset of them, in particular, those blocks whose type (private or shared) matches the requested block type.

The PS-Cache needs (i) to keep blocks tagged as private or shared in the cache, and (ii) a private-shared classification mechanism to indicate the type of the block to be looked up. Blocks are tagged in the cache using a bit (the PS bit) attached to each cache line. This bit indicates the type of the block allocated in that line. In addition, although the PS-Cache can work with any private-shared classification mechanism to find out the type of the looked-up block before accessing the data and tag arrays, this paper assumes an OS-based private-shared mechanism similar to the one proposed by Cuesta et al. [9], which keeps the information in a PS bit stored along with the TLB. Using such coarse granularity presents its advantages and shortcomings. An interesting analysis about this fact can be found in [9]. In this way, the PS-Cache only accesses those ways whose PS bit value in the entry matches with the PS bit value given by the TLB for the looked-up block.

4.1 The PS page classification mechanism

The classification mechanism is based on OS support, therefore, the classification is performed at the page granularity. This means that all the blocks of the same page

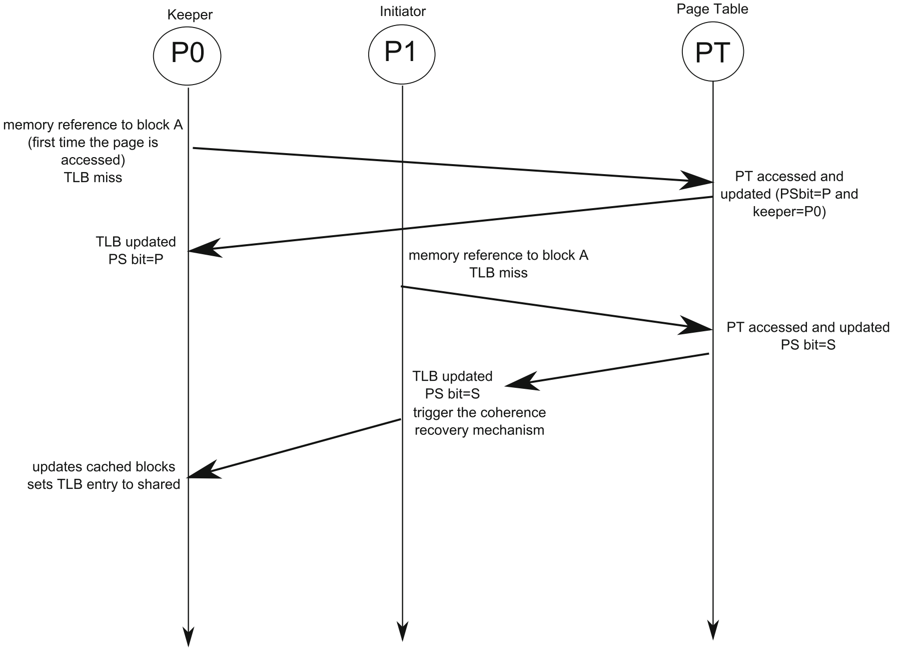

Keeper
P0
Initiator
P1
Page Table
PT
memory reference to block A (first time the page is accessed) TLB miss
PT accessed and updated (PSbit=P and keeper=P0)
TLB updated PS bit=P
memory reference to block A TLB miss
PT accessed and updated PS bit=S
TLB updated PS bit=S trigger the coherence recovery mechanism
updates cached blocks sets TLB entry to shared

Fig. 3 The PS Page classification mechanism workflow. P0 and P1 are processors and PT is the page table in main memory

are classified with the same type. The sharing information is stored both in the page table and in the TLB that holds the translation for the most recently referenced pages. The sharing information comprises the PS bit and the *keeper* id, which is the first core that requested the page translation. This information is stored in the page table, and the TLB only stores the PS bit. On a memory reference, the core obtains the block type of the reference from the TLB when it is accessed with the purpose of getting the address translation. On a TLB miss, the page table is accessed (as usually done), but the devised classification mechanism also updates the sharing information in the page table and in the core TLB.

Figure 3 depicts an overview of how the classification mechanism works. The first miss (suffered by P0 in the example) sets the page status as private and the keeper field is set to P1. The page is set to private in the P0 TLB. On subsequent misses, if the page is found as private (which occurs in the second access from P1 in the example), it is necessary to compare the keeper field (P0) with the core requesting the access (P1) to check if the page will become shared or if, otherwise, the one that requests the page translation is the same core as the first time. If the page must become shared, the page table (labeled as PT) entry is updated and a private-shared coherence recovery mechanism is triggered to maintain coherence between the page table and the keeper TLB and the PS bits in the caches.

The private-shared coherence recovery mechanism has to ensure that all the PS bits of the cached blocks of the page as well as in the TLBs keep the same type as their entry in the page table. For doing this, the requesting core issues a *recovery request* to

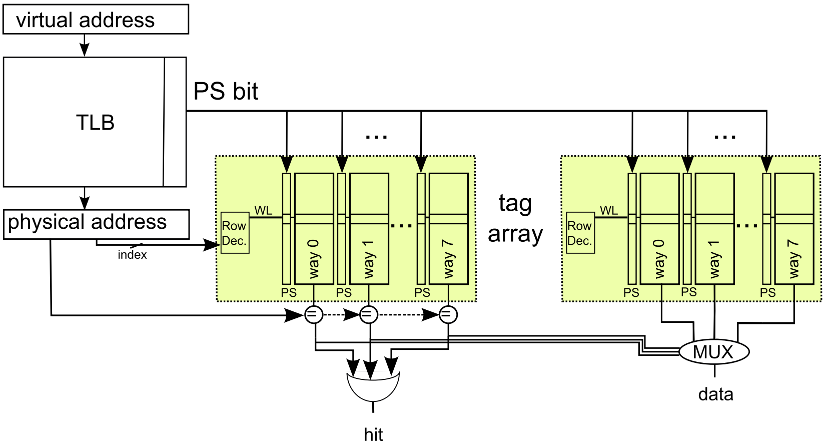


Fig. 4 The PS-Cache architecture for L1 caches

the page keeper (obtained from the page table entry). On the arrival of such a request, the keeper updates both the PS bit in the corresponding TLB entry and the PS bits of the cached blocks belonging to the given page. Notice that this recovery procedure is only required upon a Private-to-Shared transition. This way we keep the PS bit of every block in the cache coherent with the state of the page in the TLBs and in the page table. More details about the mechanism can be found in [9].

After solving the TLB miss the sharing information is in the PS bit stored along with the page translation in the TLB of the requesting core and this PS bit is used by the core to check the type of the requested block (private or shared). As commented above, the PS bit allows us to discern the group of ways in which the requested block can be found and, consequently, only these ways are accessed.

Although the private-shared classification employed in this work is performed by accessing the page table on every TLB miss [8,9], the PS-Cache can also work along with a classification mechanism that employs TLB-to-TLB transfers [33], which can improve the overall performance of the system.

4.2 The PS-Cache architecture

On the execution of a memory reference instruction, the cache controller first looks in the TLB to get the physical address[2]. As mentioned above, the TLB includes one bit per entry, the PS (Private-Shared) bit, that indicates how the page is classified. The value of this bit is read from the TLB entry jointly with the physical address. With this information, the cache controller proceeds searching the block in the cache. The TLB translation information is used to check the tags in the corresponding set, but as a novelty the proposal also uses the PS bit to avoid some of the way accesses. The mechanism can be applied to any cache level, for illustrative purposes. Figure 4 depicts an overview of the proposed cache architecture for the L1 cache.

[2] If a TLB miss occurs, after solving the miss the corresponding entry is in the TLB.

The key difference is that in the PS-Cache only those ways which share the same type as the one indicated by the TLB are accessed, thus eliminating the energy consumption caused by the look-up in the other ways. As observed, each cache line has attached a PS (Private-Shared) bit which indicates the type of each block (according to the page table and the other TLBs). The PS bit provided by the TLB is compared with the PS bits of all the ways in the set. A simple logic is included to select the wordline (WL) of those ways whose PS bit matches the value of that obtained from the TLB for the page of the current memory reference. This means that, in the tag array, only a subset of the tags is read and then compared with the tag of the physical address and, in the data array, only a subset of data blocks is read. On a hit, the mux placed in the data array would select the proper data block from the ones accessed. This allows the proposal to reduce significant dynamic energy consumption across the memory accesses since in general, as experimental results will show, only a small fraction of ways is required to be accessed in many memory operations.

The proposal also reduces dynamic energy consumption when accessing the cache from the NoC side (i.e., coherence requests). In this case, only shared ways are looked up, since the classification mechanism ensures that before arriving the coherence request, the block is shared or it has been reclassified as shared by the private-shared updating mechanism.

In addition, to reduce the static power consumption, the proposed mechanism takes also advantage of the invalid bit. The power of all ways in an invalid state is turned off and is also excluded from the process of looking the block up. This allows not only reducing the number of possible ways for the block (the lower the number, the less dynamic consumption), but also reducing the static energy consumption since power supply to these ways is cut off while they are in the invalid state.

In case of accessing to L2 or L3 caches, the PS bit of the target block (already taken from the TLB) is carried in the miss request. On the other hand, PS bits in the cache entries are updated accordingly by the cache coherence protocol, so the PS bit of a request and the PS bit of the requested block are always coherent.

Regarding hardware complexity, the proposal requires minimal complexity. On the one hand, no extra information must be added to the TLB except a single bit (the PS bit) per entry. Note that this bit can also be employed to optimize the cache coherence protocol as in [9]. On the other hand, the proposal can be easily adapted to current caches. In fact, using a single wordline for all the ways in the set presents several problems due to, among others, many transistors are connected to the row's wordlines and the column's bitlines increasing the total capacitance and resulting in an increase in delay and power dissipation. Thus, to deal with this problem, current SRAM cache designs employ the divided wordline approach (DWL), which divides the wordline into a fixed number of blocks, for instance, one WL per cache way [34]. Notice, that our proposal takes benefit of this wordline scheme already working in current caches. Due to the low overhead of our scheme the access time to the PS-Cache is not affected. In L2, the PS bit is known before accessing the cache and therefore does not affect the access time. Even regarding the first-level cache, the proposal could be integrated in most current deep pipelined processors because the access to first-level caches usually takes several stages; e.g., a L1 hit time takes 3 cycles in the AMD Opteron X4 2356 (Barcelona) [35].

The previous discussion focused on typical physically tagged and physically indexed caches. However, the proposal could be also applied to other types of caches; for instance, virtually indexed but physically tagged. In these caches the tag array is accessed in parallel with the TLB, and then the physical address is used to compare only those tags whose type matches the target one.

5 Simulation environment

The proposal has been evaluated with a full-system simulation using Virtutech Simics [36] and the Wisconsin GEMS toolset [37], which enables detailed simulation of multiprocessor systems. GARNET [38], a detailed network simulator included in the GEMS toolset, models the interconnection network.

Table 1 shows the values of the main system parameters that correspond to the assumed base system, which is a 16-tile CMP architecture. CACTI 6.5 tool [39] has been used to estimate access time, area requirements and power consumption of the different cache structures for a 32-nm technology node and high-performance transistors.

Two cache coherence protocols, a directory-based protocol and a snoop-based protocol, have been implemented and evaluated. Both protocols store the blocks in the private caches considering MOESI states, and implement a non-inclusive LLC (L2 in our study) cache. Also, invalidation acknowledgments are directly sent to the requester. The main difference between the two protocols is the trade-off between area and power. The directory protocol implements an on-chip directory cache, which increases its area overhead, while the snoopy protocol performs a broadcast on every write, and on every load in case the data is not found in the LLC. As analyzed in Sect. 2, snoopy protocols induce a higher number of coherence requests to the L1 caches, therefore, a reduction in the average number of accessed ways results in higher energy savings than in directory-based protocols. Energy results account for any access to the cache, includ-

Table 1 System parameters

Memory parameters	
L1 I & L1 D caches	64KB, 8-way
L1 cache access time	2 cycles
Cache block size	64 bytes
Shared single L2 cache	Non-inclusive, 512KB/tile, 16-way
L2 cache access time	6 cycles (2 if only tag accessed)
Memory access time	160 cycles
Network parameters	
Topology	2-dimensional mesh (4 × 4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

Table 2 Simulated applications

Name	Parameters
SPLASH-2 benchmark suite	
Barnes	16K particles
Cholesky	tk15
FFT	64K complex doubles
Ocean	514 × 514 ocean
Radiosity	Room, -ae 5,000.0 -en 0.050 -bf 0.10
Radix	512K keys, 1,024 radix
Raytrace	Teapot—optimized by removing locks for unused ray ids
Volrend	Head
Water-Nsq	512 molecules
PARSEC suite	
Blackscholes	Simmedium
Swaptions	Simmedium
ALPBench suite	
FaceRec	Script
MPGdec	525_tens_040.m2v
MPGenc	Output of MPGdec
SpeechRec	Script
Scientific benchmarks	
Tomcatv	256 points
Unstructured	Mesh.2K
Commercial workloads	
Apache	4,000 transactions
SPEC-JBB	4,000 transactions

ing those that come as a consequence of the private-shared classification mechanism of page tables.

The proposal has been evaluated using a wide range of scientific applications from the SPLASH-2 benchmark suite [40], the ALPBench suite [41], the PARSEC suite [42], scientific applications and commercial workloads. Table 2 shows the list of applications considered in the study. Experimental results reported in this work correspond to the parallel phase of the evaluated benchmarks.

6 Experimental evaluation

6.1 Private-shared blocks behavior analysis

Energy benefits of the proposal depend on the average number of ways that are looked up by the cache accesses. This number would mainly change depending on which cache we are accessing to (L1 or L2), and on the type of block we are looking for.

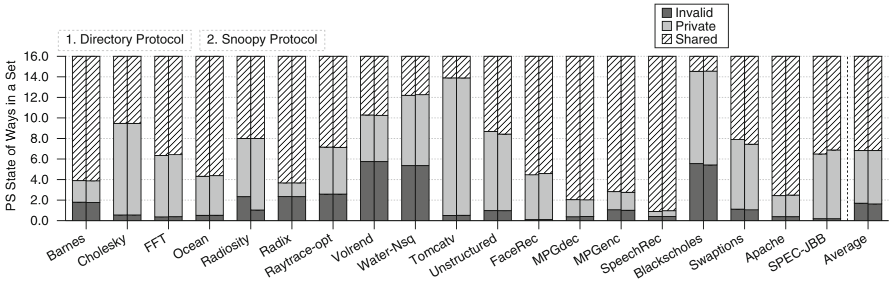


Fig. 5 Average number of ways in a set of each type in the L2 cache for both studied protocols

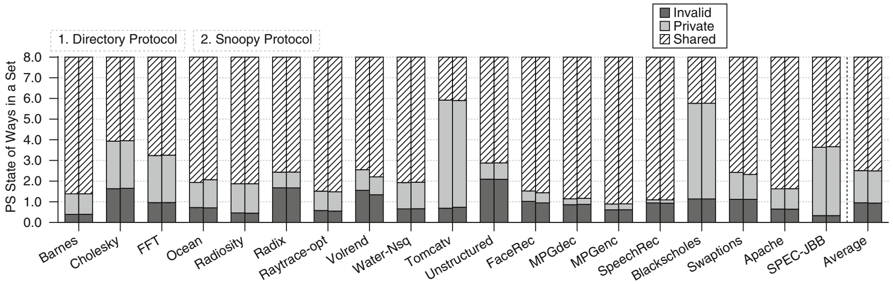


Fig. 6 Average number of ways in a set of each type in the L1 cache for both studied protocols

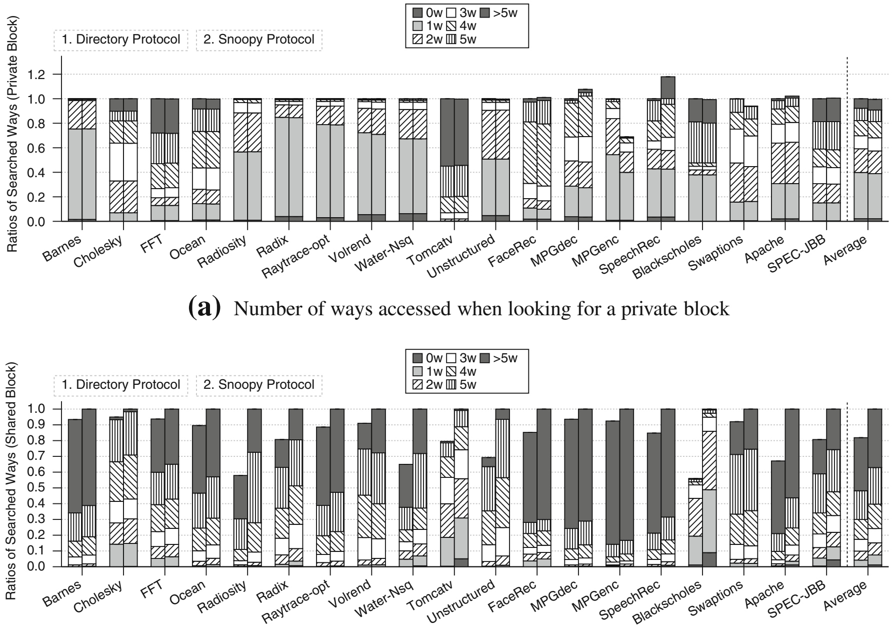
The study starts with the L2 cache since it implements a higher number of ways, thus the proposal can potentially achieve higher energy savings in this cache.

Figure 5 depicts the average number of blocks of each type in the 16-way set-associative L2 cache. Results are shown for the snoopy and directory MOESI-based protocols considered in this paper. As observed, on average, there are around five private blocks in a set, whose access would result in important energy savings. Nevertheless this number strongly depends on the application. There are some few applications with more than twelve private blocks per set on average (e.g., *tomcatv*), but as can be seen most of the applications store shared blocks in most of the ways.

Figure 6 shows the average number of blocks of each type in the 8-way set-associative L1 cache. Unlike L2 caches, the difference in the amount between private and shared ways is higher, around two ways storing private blocks and five storing shared blocks.

Results confirm that the final number of accessed ways vary according to the type of block we are looking for and the application behavior. That is, the average number of blocks of each type seen on the arrival of a request to a private block can widely differ from that seen on the arrival of a request to a shared block.

To provide further insights on how much energy savings the proposal is able to bring, Fig. 7a, b shows the distribution of the number of accessed ways on each access on the arrival of a private or shared request, respectively, in the L1 cache. The data in the first figure is normalized to the number of memory accesses when employing a directory protocol with PS-Cache, whereas the second figure is normalized to the number of memory accesses when employing a snoopy protocol with PS-Cache. As



(a) Number of ways accessed when looking for a private block

(b) Number of ways accessed when looking for a shared block

Fig. 7 Distribution of the number of ways accessed in the L1 cache normalized with respect to the snoopy protocol

expected from Fig. 6, most applications look up more ways when accessing shared blocks than when accessing private blocks, with only few exceptions such as *Tomcatv* and *Blackscholes*. An interesting observation is that when looking for a private block, most of the times (over 60 % of the accesses) just one or two ways are looked up. Benefits, are lower when looking for a shared block, but even in this case, around 60 % of times, five or less ways are looked up, which will also bring important energy savings. Regarding protocols, two main observations can be drawn. First, it can be appreciated that the rate of shared to private blocks accessed is quite similar in both protocols regardless of whether a shared or private block is requested. Second, major differences among protocols mostly appear when looking for a shared block, with the exception of *SpeechRec* when looking for a private block. In this case, a directory-based protocol reduces the number of lookups on average around 20 % with respect to the snoopy protocol. Moreover, this reduction can be as high as 70 % in *Water-Nsq* when looking a shared block.

Figure 8 shows the average number of accessed ways per access in the L2. On average a second-level cache implementing the PS-Cache architecture needs only to look 10 of its 16 ways up, although there are some cases (i.e., *BlackScholes*) in which this number can be as low as 2 ways per access.

As mentioned above, the proposal can be applied to any level of the cache hierarchy, thus this section also explores the benefits on the L1 cache in the studied system.

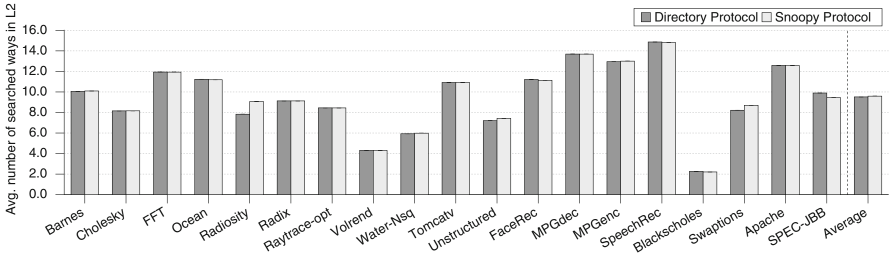


Fig. 8 Average number of ways accessed in the L2 cache for the studied protocols

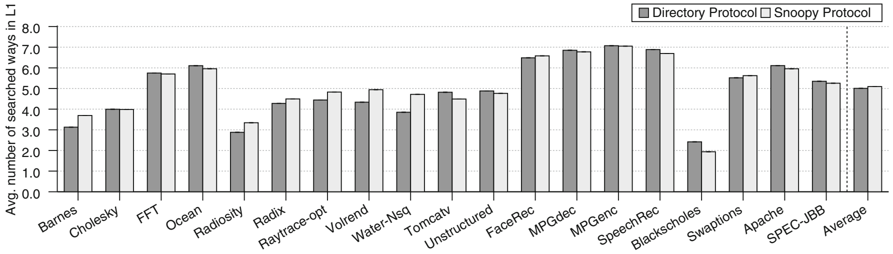


Fig. 9 Average number of ways accessed in the L1 cache for the studied protocols

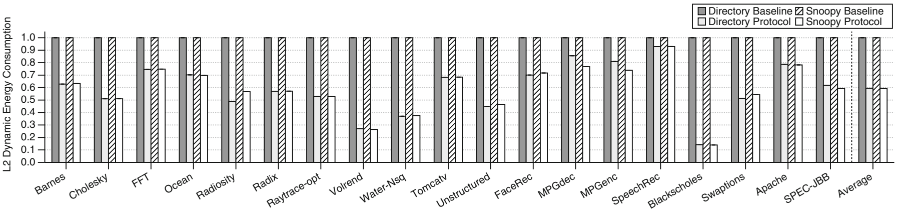


Fig. 10 Reduction of the dynamic energy consumption in L2

Figure 9 shows how many ways are looked up on average across all the benchmarks in the proposed 8-way first-level cache. Results show scarce difference between both types of coherence protocols, both of them accessing 5 ways on average. In some applications the PS-Cache greatly reduces the number of ways to be looked up (e.g., only 2 in *Blackscholes*), while in others like *MPGenc* that presents a large number of shared ways, the impact is not so high.

6.2 Energy consumption

This section analyzes the impact of the proposal on the energy consumption of the caches.

Figure 10 shows the dynamic energy consumed by the L2 cache for the directory and a snoopy protocols considered in this paper. Conventional protocols and caches (labeled as baseline) have been included for comparison purposes. Results of the PS approach have been labeled with the name of the protocol implemented in the system.

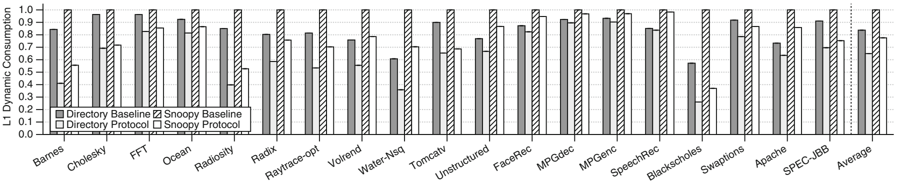


Fig. 11 Reduction of the dynamic energy consumption in L1

In this cache, there is not much difference between both coherence protocols. As observed, energy savings widely differ across benchmarks. On average, the PS-Cache achieves by 40 % of energy reduction in both coherence protocols. However, notice that in some cases, energy consumption of the PS-Cache is only by 12 % that of the conventional system (*BlackScholes*), and even in the worst case, these benefits always exceed 8 %.

Figure 11 shows the results for the L1 cache. On average, similar energy savings (i.e., by 22 %) in percentage are brought by both snoopy and directory protocols in the L1 cache. An interesting remark is that a snoopy protocol with the PS-Cache architecture can consume less than a conventional directory one. This is simply achieved through the selective look-up within the different ways of a set provided by the PS bit.

As suggested in the previous section, applications with a large number of private-block lookups, obtain higher energy reductions. For example, *Barnes* reduces dynamic power consumption by 44 and 51 %, for snoopy and directory protocols, respectively, and *Radiosity* by 47 and 53 %, respectively. On the other hand, applications with a low number of private-block lookups offer no such benefits. Best example of this scenario is the *SpeechRec* benchmark, which only reduces the power consumption by 3 %.

Results show that the dynamic energy consumption reduction is higher in L2 caches than in L1 caches, even more so if we considered virtually-indexed, physically-tagged L1 caches instead of physically-indexed, physically-tagged ones. Hence, it can be concluded that the lower the cache level the higher the benefits provided by this technique.

6.3 Comparing PS-Cache versus way prediction and way guard

This section compares the PS-Cache with previous proposals that also reduce dynamic energy consumption by accessing a subset of the cache ways instead of all of them, i.e., Way Prediction and Way Guard. The following subsections provide a brief summary of these schemes to make the paper self-contained. Finally, a comparison of both of them against our PS-Cache scheme is shown. For the sake of fairness, the invalid bit technique has also been implemented in all the compared schemes.

6.3.1 Way prediction

Way Prediction techniques [7,20] predict the way to be accessed in advance (typically the way containing the MRU block) and only that way is accessed first. The problem lies when the prediction fails; in such a case, all the remaining ways are accessed at a second phase to look for the target block. This means that on misprediction, energy wasting rises and latency increases, since additional cycles are required to solve the memory request.

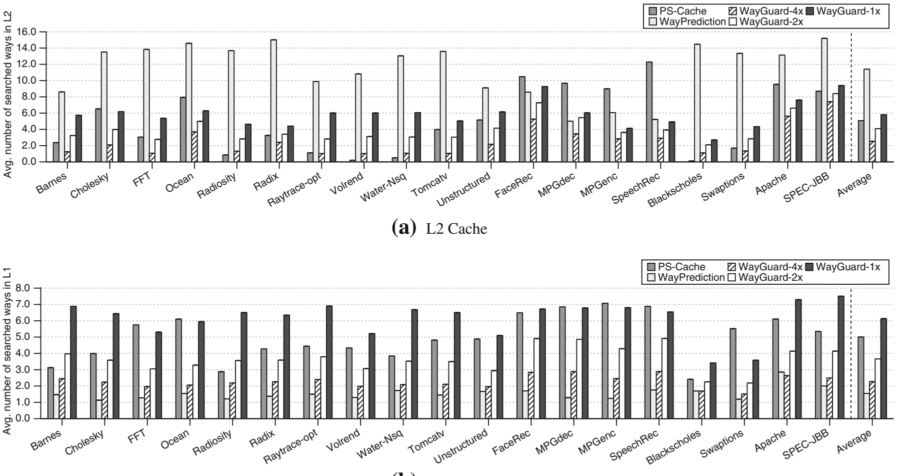
6.3.2 Way Guard

Way Guard [6] has been proven to work efficiently in highly associative caches. The mechanism implements a counting bloom filter associated with each cache way. Way Guard works as follows. First, a hash function is applied to a subset of bits of the address of the target block. The output of the hash is a m-bit index that is decoded to access the $2^m - 1$ entry bloom filter vector. If the bit is set to 1 then the associated cache way is accessed (both tags and data arrays), otherwise that way is not accessed. Each entry of the bloom filter has associated an up/down counter (e.g., 3-bit in the original work), that is decremented each time a cache line whose address maps to that position is evicted from the cache and increased when the block is written in the cache. In the original paper, results are shown for m equal to four times the number of blocks in a cache. We will refer to this configuration as *WayGuard-4*×. This approach requires a decoder with 4 times more outputs than the already implemented in the cache to index the target set. Therefore, to perform a fair comparison in terms of area and complexity we evaluate two additional configurations: *WayGuard-2*× and *WayGuard-1*×, both of them still having more memory requirements than the PS-Cache. The former uses a decoder which doubles the outputs of the cache set decoder and the latter can share the same decoder. Note that *WayGuard-1*× includes a 3-bit counter and one extra way-selection bit per cache entry, while the PS-Cache only adds one bit per entry.

6.3.3 Quantitative comparison

Figure 12a, b shows the average number of ways accessed in the PS-Cache, Way Prediction and three Way Guard variants in the L2 cache and the L1 cache, respectively.

In the L2 cache, the proposed PS-Cache is the design choice that provides the best performance-complexity trade-off. The PS-Cache scheme works really well in some applications where the number of accessed ways is below 6, accessing on average around 8 ways, which is the expected value having only one bit to filter ways in a 16-way cache. As mentioned above, Way Prediction is not an adequate design choice for the lower level caches since they are much less predictable than L1 caches. This is because the L1 cache filters the high-locality requests. Results show that on average 12 ways are accessed for the Way Prediction in the L2 cache, while only 4 ways are accessed by the PS-Cache. There are only a few applications in which Way Prediction achieves better results. Comparing Way Guard, the PS-Cache approach achieves even higher reductions than *WayGuard-2*× with much less complexity. Remember that this configuration has eight bits per line (two bloom filter bits—the decoder is twice as large—and two 3-bit counters). Regarding, *WayGuard-4*×, it achieves slightly better results but with much more hardware complexity.



(a) L2 Cache

(b) L1 Cache

Fig. 12 Average number of ways accessed comparison

An interesting remark is that the Way Guard approach puts the look-up to the bloom filter bit on the critical path regarding the accessed cache level which might impact on the access time. That is, the cache cannot be accessed until the bloom filter vector is known. In contrast, the PS-Cache has no extra delay when accessing the L2 cache. Also, energy consumption due to a bigger decoder and logic of the up/down counter might make energy benefits of the Way Guard to be slightly mitigated by this additional circuitry.

In the L1 cache, Way Prediction is the best design choice in L1 caches, since the MRU hit ratio is really good at this cache level. Compared to Way Guard, the PS-Cache approach, with less hardware complexity (it does not require a 3-bit up/down counter per cache entry), improves *WayGuard-1*× but it is worse than the remaining Way Guard approaches which have a higher complexity. Hence, since they employ more bits to filter ways, the expected results are better.

In summary, one can conclude that the optimal solution from performance and energy points of views would be to employ Way Prediction in first-level caches and PS-Cache in lower level ones, obtaining in this way both the most energy-saving and the shorter execution time.

7 Conclusions

One of the major design concerns in current high-performance CMPs is the power consumption, which increases as the number of core counts grows. On-chip caches consume a significant fraction of the total power budget, and important research has focused on reducing energy consumption in these memory structures, sometimes at the cost of performance.

This work has proposed the PS-Cache, an energy-efficient cache design that only accesses a subset of the set ways, without hurting the performance. The PS-Cache assumes that blocks are classified at page level as shared or private, according to the TLB information. It also adds a single bit attached to each cache entry, which only activates the word line if the block type matches the one provided by the TLB. In this way, dynamic energy consumption is largely saved. On the other hand, coherence requests to remote private caches only access the subset of ways that has blocks with the shared type.

Results have shown that in CMPs, implementing either directory-based or snoop-based protocols, the PS-Cache can bring important energy savings; and energy savings (quantified in percentage) are roughly the same for all tested coherence protocols.

The proposal has been evaluated in both L1 and L2 caches. Different cache schemes has been compared. Results show that the well-known Way Prediction is the best performing approach for L1 caches. However, for L2 caches the proposal reduces more than 2× the number of searched ways by the Way Prediction Scheme, which brings energy savings by 40 % with respect to a conventional scheme. Moreover, it reduces more energy with much less hardware complexity than Way Guard, an state-of-the-art energy-aware cache approach.

Acknowledgments This work has been jointly supported by the MINECO and European Commission (FEDER funds) under the project TIN2012-38341-C04-01 and the Fundaci'on Seneca-Agencia de Ciencia y Tecnolo'ia de la Regi'on de Murcia under the project J'ovenes L'ideres en Investigaci'on 18956/JLI/13.

References

1. Balasubramonian R, Jouppi NP, Muralimanohar N (2011) Multi-core cache hierarchies. In: Synthesis lectures on computer architecture. Morgan & Claypool Publishers, San Rafael
2. Hennessy JL, Patterson DA (2011) Computer architecture, fifth edition: a quantitative approach, 5th edn. Morgan Kaufmann Publishers Inc., San Francisco
3. Sinharoy B, Kalla R, Starke WJ, Le HQ, Cargnoni R, Van Norstrand JA, Ronchetti BJ, Stuecheli J, Leenstra J, Guthrie GL, Nguyen DQ, Blaner B, Marino CF, Retter E, Williams P (2011) IBM POWER7 multicore server processor. IBM J Res Dev 5(3):1:1-1:29 doi:10.1147/JRD.2011.2127330
4. Kaxiras S, Hu Z, Martonosi M (2011) 28th International symposium on computer architecture (ISCA), pp 240–251
5. Flautner K, Kim NS, Martin S, Blaauw D, Kaxiras TM, Hu Z, Martonosi M (2002) 29th International symposium on computer architecture (ISCA), pp 148–157
6. Ghosh M, Özer E, Ford S, Biles S, Lee HHS (2009) International symposium on low power electronics and design (ISLPED), pp 165–170
7. Calder B, Grunwald D (1996) 2nd international symposium on high-performance computer architecture (HPCA) (1996), pp 244–253
8. Hardavellas N, Ferdman M, Falsafi B, Ailamaki A (2009) 36th international symposium on computer architecture (ISCA), pp 184–195
9. Cuesta B, Ros A, Gómez ME, Robles A, Duato J (2011) 38th international symposium on computer architecture (ISCA), pp 93–103
10. Pugsley SH, Spjut JB, Nellans DW, Balasubramonian R (2010) 19th international conference on parallel architectures and compilation techniques (PACT), pp 465–476
11. Hossain H, Dwarkadas S, Huang MC (2011) 20th international conference on parallel architectures and compilation techniques (PACT), pp 45–55
12. Kim D, Kim JAJ, Huh J (2010) 19th international conference on parallel architectures and compilation techniques (PACT), pp 111–122
13. Ros A, Kaxiras S (2012) 21st international conference on parallel architectures and compilation techniques (PACT), pp 241–252
14. Sundararajan KT, Porpodas V, Jones TM, Topham NP, Franke B (2012) 18th international symposium on high-performance computer architecture (HPCA), pp 311–322

15. Agarwal N, Peh LS, Jha NK (2009) 15th international symposium on high-performance computer architecture (HPCA), pp 67–78
16. Cantin JF, Smith JE, Lipasti MH, Moshovos A, Falsafi B (2006) Coarse-grain coherence tracking: regiόn scout and region coherence arrays. IEEE Micro 26(1):70–95
17. Ferdman M, Lotfi-Kamran P, Balet K, Falsafi B (2011) 17th international symposium on high-performance computer architecture (HPCA), pp 169–180
18. Zebchuk J, Srinivasan V, Qureshi MK, Moshovos A (2009) 42nd IEEE/ACM international symposium on microarchitecture (MICRO), pp 423–434
19. Powell M, Hyun Yang S, Falsafi B, Roy K, Vijaykumar TN (2000) International symposium on low power electronics and design (ISLPED), pp 90–95
20. Albonesi DH (1999) 32nd IEEE/ACM international symposium on microarchitecture (MICRO), pp 248–259
21. Zhang C, Vahid F, Yang J, Najjar W (2005) A way-halting cache for low-energy high-performance systems. ACM Transactions on Architecture and Code Optimization. 2(1):34–54
22. Ghosh M, Özer E, Biles S, Lee HHS (2006) 19th international conference on architecture of computing systems (ARCS), pp 283–297
23. Lee J, Hong S, Kim S (2011) 17th international symposium on low power electronics and design (ISLPED), pp 85–90
24. Kedzierski K, Cazorla FJ, Gioiosa R, Buyuktosunoglu A, Valero M (2010) 2nd international forum on next-generation multicore/manycore technologies, pp 1–12
25. Alouani I, Niar S, Kurdahi F, Abid M (2012) 23rd IEEE international symposium on rapid system prototyping (RSP), pp 44–48
26. Meng J, Skadron K (2009) International conference on computer design (ICCD), pp 282–288
27. Li Y, Abousamra A, Melhem R, Jones AK (2010) 19th international conference on parallel architectures and compilation techniques (PACT), pp 501–512
28. Li Y, Melhem RG, Jones AK (2012) 21st international conference on parallel architectures and compilation techniques (PACT), pp 231–240
29. Alisafaee M (2012) 45th IEEE/ACM international symposium on microarchitecture (MICRO), pp 341–350
30. Jiang G, Fen D, Tong L, Xiang L, Wang C, Chen T (2009) 8th international symposium on advanced parallel processing technologies. Springer, Berlin, pp 123–133
31. Sundararajan K, Jones T, Topham N (2013) IEEE 31st international conference on computer design (ICCD), pp 294–301
32. Valls JJ, Ros A, Sahuquillo J, Gómez ME, Duato J (2012) 21st international conference on parallel architectures and compilation techniques (PACT), pp 451–452
33. Ros A, Cuesta B, Gómez ME, Robles A, Duato J (2013) 42nd international conference on parallel processing (ICPP), pp 562–571
34. Jacob B, Ng S, Wang D (2007) Memory systems: cache, DRAM, disk, 4th edn. Morgan Kaufmann Publishers Inc., San Francisco
35. Patterson DA, Hennessy JL (2008) Computer organization and design: the hardware/software interface. The Morgan Kaufmann Series in Computer Architecture and Design, 4th edn. Morgan Kaufmann Publishers Inc., San Francisco
36. Magnusson PS, Christensson M, Eskilson J, Forsgren D, Hallberg G, Hogberg J, Larsson F, Moestedt A, Werner B (2002) Simics: a full system simulation platform. IEEE Comput 35(2):50–58
37. Martin MM, Sorin DJ, Beckmann BM, Marty MR, Xu M, Alameldeen AR, Moore KE, Hill MD, Wood DA (2005) Multifacet's general execution-driven multiprocessor simulator GEMS toolset. Comput Archit News 33(4):92–99
38. Agarwal N, Krishna T, Peh LS, Jha NK (2009) IEEE international symposium on performance analysis of systems and software (ISPASS), pp 33–42
39. Muralimanohar N, Balasubramonian R, Jouppi NP (2009) Cacti 6.0. Tech. Rep. HPL-2009-85, HP Labs
40. Woo SC, Ohara M, Torrie E, Singh JP, Gupta A (1995) 22nd international symposium on computer architecture (ISCA), pp 24–36
41. Li ML, Sasanka R, Adve SV, Chen YK, Debes E (2005) International symposium on workload characterization, pp 34–45
42. Bienia C, Kumar S, Singh JP, Li K (2008) 17th international conference on parallel architectures and compilation techniques (PACT), pp 72–81