The Tag Filter Architecture: An energy-efficient cache and directory design

Joan J. Valls[a,1], Alberto Ros[b], María E. Gómez[a], Julio Sahuquillo[a]

[a]*Universitat Politècnica de València (Spain)*
[b]*Universidad de Murcia (Spain)*

Abstract

Power consumption in current high-performance chip multiprocessors (CMPs) has become a major design concern that aggravates with the current trend of increasing the core count. A significant fraction of the total power budget is consumed by on-chip caches which are usually deployed with a high associativity degree (even L1 caches are being implemented with eight ways) to enhance the system performance. On a cache access, each way in the corresponding set is accessed in parallel, which is costly in terms of energy. On the other hand, coherence protocols also must implement efficient directory caches that scale in terms of power consumption. Most of the state-of-the-art techniques that reduce the energy consumption of directories are at the cost of performance, which may become unacceptable for high-performance CMPs.

In this paper we propose an energy-efficient architectural design that can be effectively applied to any kind of cache memories. The proposed approach, called the Tag Filter (TF) Architecture, filters the ways accessed in the target cache set, and just a few ways are searched in the tag and data arrays. This allows to reduce the dynamic energy consumption of caches without hurting their access time. For this purpose, the proposed architecture holds the X least significant bits of each tag in a small auxiliary *X-bit*-wide array. These bits are used to filter the ways where the least significant bits of the tag do not match with the bits in the *X-bit* array. Experimental results show that, on average, the TF architecture reduces the dynamic power consumption across the

Email addresses: `joavalmo@fiv.upv.es` (Joan J. Valls), `aros@ditec.um.es` (Alberto Ros), `megomez@disca.upv.es` (María E. Gómez), `jsahuqui@disca.upv.es` (Julio Sahuquillo)
[1]Corresponding author

studied applications up to 74.9%, 85.9%, and 84.5% when applied to L1 caches, L2 caches, and directory caches, respectively.

Keywords: Multicore, cache, directory, dynamic consumption

1. Introduction

As silicon resources become increasingly abundant, core counts grow rapidly in successive chip-multiprocessors (CMPs) generations. These CMPs usually improve their memory latency by using a multi-level cache hierarchy that occupies a significant percentage of the overall CMP die area [1] and consumes an important fraction of the overall power budget. Most of the cache power consumption is due to dynamic power while a small fraction is due to static power. Dynamic power consumption comes from switching activity of transistors during cache accesses while static power consumption comes from current leaking, even when the cache is not being accessed. Cache designers must reach a compromise among performance, cost, size, and power/energy dissipation.

Dynamic energy consumption is dominated by first-level (L1) caches since they are much more frequently accessed than last level caches (LLC), *e.g.* L3 caches. Moreover, the high consumption of the cache memories is due to the parallel access of tag and data arrays, a common characteristic in L1 caches since they have a strong influence on the overall processor performance. This concern is even more important in CMPs than in monolithic processors since L1 caches can be accessed both from the processor side and from the interconnection network side due to coherence requests, increasing their number of accesses.

Due to performance reasons, processor caches are being deployed with a high associativity degree. In high-performance microprocessors, ways in the target set are accessed concurrently on a cache look up. Thus, the associativity degree defines the number of accessed tags. In addition, caches include one comparator per way and compare as many tags as the number of ways. As a consequence, the dynamic energy dissipated per access increases along with the cache associativity.

Apart from the cache hierarchy, other cache structures are implemented in manycore processors. In current many-cores, like the Xeon Phi [2], directory-based coherence is the commonly preferred approach. Cache memories, referred to as directory caches or sparse directories, are used to keep track of the cached blocks. In this approach, when a directory entry is evicted, all copies of the tracked block in the processor caches are invalidated, even if the block is being used by the processor, thus rising the so-called *coverage* misses [3]. Mainly due to power reasons (*i.e.* lower associativity degree in comparison to duplicate tags directories [4, 5, 6]), sparse directories are the preferred design choice for a medium to high number of cores.

Previous research on the design of energy-efficient caches addressing dynamic power has focused on minimizing the internal transistor activity during a cache access. That activity comes from reading and comparing tags in tag arrays, and from reading and writing data in data arrays. Ideally, on a cache hit, the cache would read and compare a single tag, and access a single data entry without losing performance. Furthermore, on a miss, the cache would have no need to access neither the tag array nor the data array.

Many cache energy reduction approaches have focused on monolithic processors in the past such as Cache Decay [7], Drowsy Caches [8], and Way Guard [9]. Some of them, *e.g.* [10], were originally developed to reduce cache access time, but subsequent research has proven that these schemes provide important energy savings. However, since these schemes are not directly applied to CMPs, recent research has dealt with energy savings on CMPs when running parallel workloads.

In this work we propose the Tag Filter (TF) Architecture, a technique that can be applied to any set-associative cache in the system, ranging from processor caches that store memory blocks to directory caches holding coherence information. The TF Architecture reduces the number of tags and data entries checked when accessing a cache, which leads to reducing the dynamic power consumption. The TF Architecture is based on using the X least significant bits (LSB) of the address tag (X = 1, 2, 3, 4) to discern which ways of a cache may contain the searched block. Only those ways that may potentially contain the block are accessed, thus saving the energy required to access the other ways. This allows caches to implement high-associativity with a similar energy consumption as that of conventional low-associativity caches, while

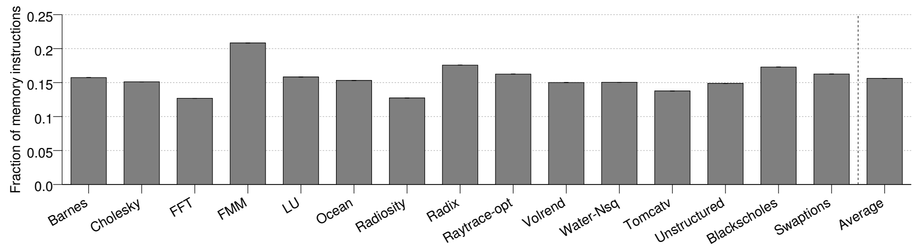


Figure 1: Fraction of memory instructions.

reducing the number of conflict misses.

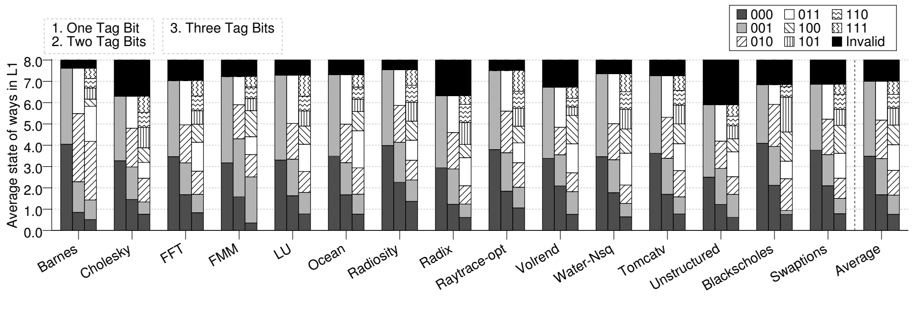
The TF Architecture is deployed with minimal hardware complexity, while allowing tag and data arrays to be accessed in parallel, so no performance degradation rises with respect to conventional caches. In addition, unlike other approaches [11], no way-alignment across sets must be done, which simplifies its implementation.

Experimental results show that the TF Architecture with $X = 4$ can reduce dynamic power consumption up to 74.9% when applied to the L1 cache and up to 85.9% when applied to the L2 cache, thus achieving better results than recent proposals. Additionally, when applied to the directory caches, the TF Architecture can reduce the dynamic energy dissipated up to 84.2% for a single-level directory cache and 84.5% for a multi-level directory cache.

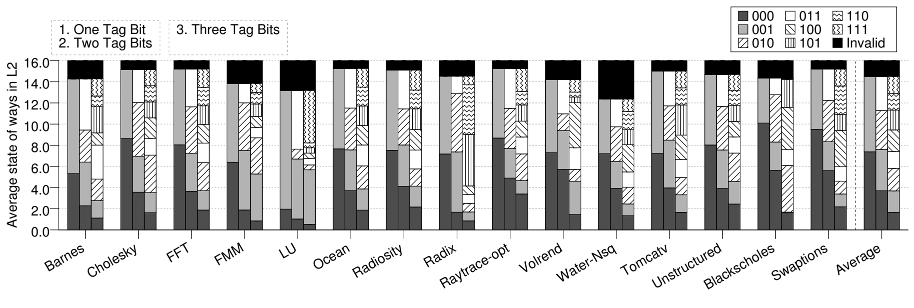
The remainder of this paper is organized as follows. Section 2 describes the main reasons that motivate us to carry out this research. Section 3 discusses the related work. Section 4 presents the proposal. Section 5 describes the simulation framework. Section 6 presents how the number of ways accessed is reduced and energy results. Finally, Section 7 draws some concluding remarks.

2. Motivation

Cache memories, especially first- and second-level caches, are frequently accessed, since memory reference instructions represent a significant percentage of the executed instructions. Figure 1 shows the fraction of memory reference instructions in each



(a) Average number of ways in a set of each type in the L1 cache



(b) Average number of ways in a set of each type in the L2 cache

Figure 2: Average number of ways in a set of each type in the cache hierarchy.

individual benchmark executed on a 16-core CMP system (experimental environment, system parameters, protocols and cache hierarchy are described in Section 5). This value is roughly the same, around 16%, across the different benchmarks.

Therefore, a significant fraction of the total power budget is often consumed by on-chip caches, as for example, in the Niagara2 processor [**?**], where 44% of the chip power is consumed by the L2 cache [12]. Reducing dynamic power consumption in caches of CMPs is an actual problem that is being under research [11, 13]. To deal with this problem, this paper proposes an architectural approach based on the hypothesis of the homogeneous distribution of the least significant bits of the address tag across the ways of a set-associative cache [14].

We launched experiments to verify this hypothesis in the considered workloads.

Figure 2 shows the average distribution of the blocks across an 8-way L1 cache and a 16-way L2 cache on a 16-core CMP system. In Figures 2(a) and 2(b) on average there are 1 and 2 ways in an invalid state, under the implemented MOESI protocol, in the L1 and L2 cache, respectively. Meanwhile, the remaining ways share a quite homogeneous distribution considering the lowest order tag bits of the allocated blocks. Therefore, there is no need to access all ways in a set if there is some mechanism able to filter accesses only looking at the subset of ways that might allocate the requested block. The homogeneous distribution of the lowest order tag bits makes our approach a perfect method for filtering accesses.

Directory caches are typically built as set-associative caches and, as experimental results will show, the least significant bits in the address tag follow a similar homogeneous distribution. A low associativity of the directory caches causes frequent evictions of directory entries that lead to extra coverage misses in the processor caches, thus as in any kind of cache, a higher associativity would improve the performance. However, their associativity is also limited due to power constraints. This power limitation can be attacked using the TF Architecture allowing the same performance as highly-associative directories with similar dynamic energy consumption as low-associative directories.

In summary, the aim of this paper is to save dynamic energy by reducing the number of lookups on each cache or directory access while keeping the performance of high-associative caches.

3. Related Work

An important amount of research has focused on reducing the energy consumption of cache memories. Most of this work has mainly focused on processor caches which was already a concern in monolithic processors. Other work has focused only on directory caches in many-core CMPs where the problem mainly lies on scalability issues. Below, we summarize these two lines of research.

3.1. Processor Caches

Cache power consumption comes from both leakage (or static) and dynamic power consumption. Regarding leakage savings, Powell *et al.* [15] proposed a Gated-Vdd technique that aims to reduce leakage for instruction caches by re-configuring them and turning off unused lines. Kaxiras *et al.* [7] proposes 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. Drowsy Caches [8] exploits the fact that in a particular period of time only a subset of the cache lines are accessed. Different from the Cache Decay proposal, in the Drowsy Caches scheme the voltage is reduced but not cut off for those cache lines that are not being accessed. In case the voltage is cut off, the content of the cache line cannot be preserved, and therefore, it has to be re-fetched, as happens in the Cache Decay proposal.

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 [16] 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.* [10] to reduce the access time of set-associative caches. This approach works well in L1 caches with a relatively small associativity which presents very predictable access patterns; however, as we show in the evaluation of this work, this mechanism presents poor results for higher level caches since locality is hidden by previously accessed cache levels.

The Way-Halting cache [17] filters lines accessed in the corresponding set by comparing the four least significant bits of the tag during the index decoding. This approach performs a fully-associative search in the first comparison which negatively affects power consumption. Unlike our proposal, the way-halting cache scheme does not work in virtually-indexed physically-tagged (VIPT) caches, since it requires the tags to be available no later than the set index. If the address tag needs to be translated by a translation look-aside buffer (TLB), then the halt array (where the low order bits are stored) look-up cannot proceed. We do not compare our approach against this scheme, since even though it filters the ways accessed in a similar manner as ours, the performance is expected to be worse and the number of bits looked in the halt array is greater than the bits looked in our proposal in order to do the filtering.

Way Guard [9] is 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 decoder and two fields per way: one segmented bloom filter, previously proposed by the same authors to filter accesses to the whole cache [18], and another bloom filter to filter accesses to ways. This may result in excessive overhead and critical complexity. Way Guard shows performance gains with respect to the way-halting approach. A quantitative comparison with Way Guard is shown in the evaluation section.

The PS-Cache [19] is a mechanism that filters the ways looked up on each cache access by classifying each block as private or shared, according to the page table information. On a cache access, only the ways containing blocks that match with the classification of the requested block are searched. In the evaluation section, we quantitatively compare the PS cache with our TF Architecture.

Finally, other recently proposed techniques focus on reducing both leakage and dynamic consumption, for instance, by reducing the area of the cache tags like in the TLB Index-Based Tagging [13], by employing direct mapped caches along with mechanisms to remove conflict misses like in ASCIB [14**?**], or by performing run-time partitioning like in the Cooperative Caching scheme [11] or in the ReCaC scheme [20].

3.2. Directory Caches

Cache coherence is a necessity in shared memory systems where multiple cores can access the same memory block. Coherence protocols employ a coherence directory to track which processor private caches (*e.g.* L1 caches) share each block. The directory structure is accessed to carry out the required coherence actions such as sending invalidation requests to serialize write operations, or asking a copy of the block to the owner (*e.g.* the last processor that wrote it).

Cache directories [21] are organized as set-associative caches. Each cache directory entry encodes the set of sharers for the tracked block. Conventional approaches make use of a bit vector (a bit per-core cache) to encode the sharers. In this scheme, the per-core area and energy grow almost linearly with the core count but they can grow quadratically considering the aggregated directory size, since the number of directory structures increases with the number of cores. In summary, scalability issues rise in terms of energy and area.

Many proposals try to address the mentioned scalability issues by differentiating between entries with few (or even none) sharers and entries with many sharers. For instance, Coherence Deactivation [22] does not track private data (or even shared read-only data [23]) in the directory caches, since they are tracked at page granularity in the page table. SCD [24] uses different entry formats of the same length. Lines with one or a few sharers use a single directory entry while widely shared lines employ several cache lines (multi-tag format) using hierarchical bit vectors. Multi-grain directories (MGD) [25] also use different entry formats of the same length and track coherence at multiple different granularities to achieve scalability. Each MGD entry tracks either a temporarily private memory region or a single cache block with any number of sharers. PS-Dir [26] separates the directory in two structures, one storing shared entries and the other private entries. To save energy, this scheme provides low associativity for the cache that keeps track of shared blocks while private blocks are stored in a cache with more associativity. In addition, it also brings energy savings by not storing the sharer vector in the cache for private blocks. In contrast, the proposed TF Architecture reduces energy consumption by minimizing the number of lookups during a directory access.

4. Tag Filter Architecture

The Tag Filter (TF) Architecture can be applied to any set-associative cache in a CMP, such as processor or directory caches. The aim of our proposal is to achieve dynamic energy savings by filtering the number of accessed ways in the target set on each cache access. The terms Tag Filter Cache (TF-Cache) and Tag Filter Directory (TF-Directory) are used to refer to the Tag Filter Architecture when it is applied to a processor cache and to a directory cache, respectively.

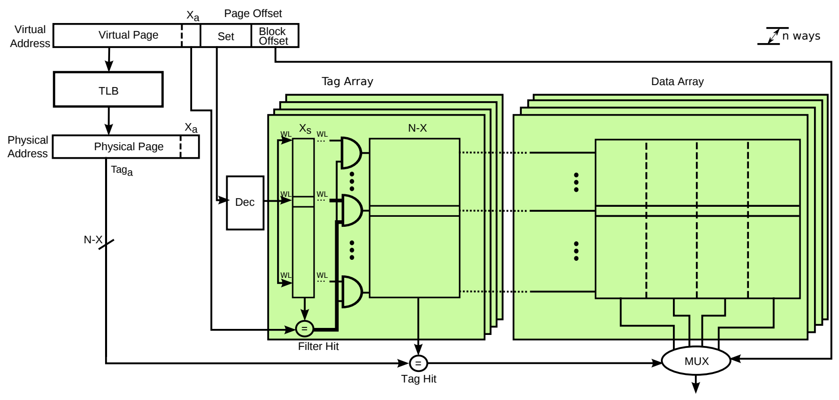


Figure 3: The TF Architecture for L1 caches.

The TF-Cache approach reduces the number of tags that are compared on each cache access and also the number of ways that are accessed in parallel in the data array. The final aim is to reduce dynamic power consumption in these cache components which represents a large percentage of the total system power consumption. In a typical cache access, to check if the searched block is in the cache, the entire tag in all the ways of the target set are compared. Although the comparison is done in all the ways, only one of them can potentially result in a hit. In first level caches, where performance is a key objective, the data array is looked up in parallel with the tag array, before knowing whether or not the target block is stored in the set.

For energy savings purposes, the proposed approach applies a first, small and fast filter to discard some of the ways accessed (both in the tag array and in the data array) to those cache ways that mismatch the small tag comparison. For this purpose, the cache memory is upgraded with minimal hardware complexity as follows. The tag array is decoupled in two main structures: one X-bit-wide and the other one $N - X$ bits wide. The TF-Cache employs the least significant bits of the tags stored in a X-bit-wide table to reduce the number of accessed ways; that is, the entries in this table act as a filter to access the tag and the data arrays as explained below. Figure 3 depicts a block diagram of the TF-Cache for a first-level cache, which is commonly virtually indexed and physically tagged (VIPT).

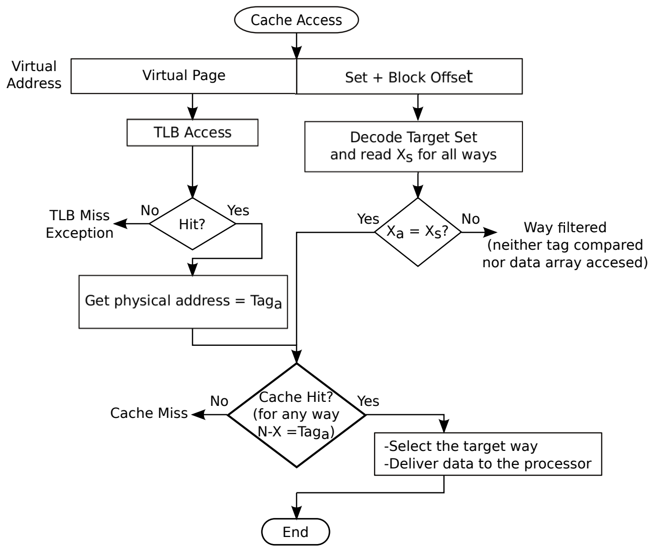

Cache Access
Virtual Address
Virtual Page
Set + Block Offset
TLB Access
Decode Target Set and read Xs for all ways
TLB Miss Exception
No
Hit?
Yes
Yes
Xa = Xs?
No
Way filtered (neither tag compared nor data array accesed)
Get physical address = Taga
Cache Miss
No
Cache Hit? (for any way N-X =Taga)
Yes
-Select the target way
-Deliver data to the processor
End

Figure 4: The TF Architecture working flow for L1 caches.

To allow the mechanism to work in current VIPT caches, like those of Intel processors, the first comparison must start before the TLB output is known. For this purpose, we assume that the operating system (OS) is responsible to ensure that the X least significant bits of the virtual address are the same as those of the physical address. This assumption is reasonable since i) a uniform page address distribution is expected and ii) main memory capacities are by four orders of magnitude bigger than page sizes (*e.g.* a 32GB main memory [27] and a 4KB page size), which allows the OS to have some allocation flexibility, and so, being able to tolerate this small restriction.

Unlike a typical cache that performs a single N-bit tag comparison, the proposed mechanism performs two tag comparisons, one of X-bit and the other of N-X-bit as depicted in the flowchart depicted in Figure 4 and illustrated in the block diagram of Figure 3. Under the aforementioned assumption, the first comparison can be done once the output of the set decoder is provided, while the address translation in the TLB is being performed. In this step, only the X least significant bits of the virtual page (namely X_a in Figure 3) are looked up in all the ways of the target set (namely X_s). The few number of bits (from 1 to 4) used in this comparison, allows it to be fast and effective (as experimental results will show), thus introducing negligible time penalty and important energy savings.

In case that the first comparison fails in all the cache ways, the L2 cache is accessed. Otherwise, two main actions are performed in parallel. On one hand, once the TLB provides the *N-X* bits of the physical address (*i.e.* Tag_a), the remaining *N-X* bits of the tag array are compared to those provided by the TLB. Notice, that this comparison involves a much larger number of bits, however, it is only performed in those cache ways that succeed the first comparison. Similarly, those entries of the data array corresponding to those ways that matched the first tag comparison are accessed.

From a complexity perspective, the proposal requires minimal hardware requirements to be applied to current caches: an additional AND gate per set and way, plus an additional X-bit small comparator per way (the one showed in the box representing the tag array in Figure 3). For instance, in the evaluated caches (see Section 5) this gives a simple circuitry consisting of 1K AND gates and 8 X-bit comparators, where X varies from 1 to 4 depending on the evaluated configuration. As mentioned above, the tag array is decoupled in two independent structures. A simple logic is required to drive the wordline (WL) signal to both the $N - X$ tag structure and the data array. As observed in Figure 3, the wordline is allowed to drive both the wide tag structure and the data array for a given way, but only in case the first comparison for that way succeeds. Notice that the AND gates do not remove the power supply since this would not preserve the data contents.

Regarding the TF-Directory, on a directory access the working mechanism works very close to that of the TF-Cache. The tag array is decoupled in two, a X-bit and a $N - X$ bits wide, tables and the access is split in two sequential steps. In this manner, the tags whose least significant bits do not match with the ones of the searched block are filtered in a similar way as explained in the L1 data cache.

This design allows the mechanism to significantly reduce dynamic energy consumption across the cache accesses since, in general, only a small fraction of ways is compared in most of the accesses.

Finally, we would like to remark that the benefits of our proposal mainly vary with the cache associativity. The higher the associativity the larger the energy savings achieved by the TF Architecture. However, since our focus is on dynamic energy, varying the number of sets or the cache size would have a minimal impact on performance other than changing the capacity misses so incurring in less or more cache accesses. Anyway, it is expected that varying the number of accesses would maintain approximately the same percentage of filtered ways hence keeping comparable energy savings. A similar rationale could be applied for an increasing core count.

5. Simulation Environment

We evaluate our proposal with a full-system simulation using Virtutech Simics [28] and the Wisconsin GEMS toolset [29], which enables detailed simulation of multiprocessor systems. GARNET [30], a detailed network simulator included in the GEMS toolset, models the interconnection network.

Table 5 shows the values of the main system parameters that correspond to our base system, which is a 16-tile CMP architecture. We use the CACTI 6.5 tool [31] to estimate access time, area requirements, and power consumption of the different cache structures for a 32nm technology node and high performance transistors.

Our evaluation analyzes a 16-core CMP comprised of a cache hierarchy with private L1 caches and a shared L2 NUCA distributed among all tiles. A MOESI directory-based cache coherence protocol keeps coherence for the data within the private caches. The directory scheme employed is an on-chip sparse directory that employs a bit-vector sharing code.

The proposal has been evaluated with a wide range of scientific applications. *Barnes* (16K particles), *Cholesky* (tk15), *FFT* (64K complex doubles), *FMM* (16K particles), *LU* (512×512 matrix), *Ocean* (514×514 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Radix* (512K keys, 1024 radix), *Raytrace* (teapot –optimized by removing locks for unused ray ids–), *Volrend* (head), and *Water-Nsq* (512 molecules) are from the SPLASH-2 benchmark suite. *Tomcatv* (256 points) and *Unstructured* (Mesh.2K) are two scientific benchmarks. *Blackscholes* (simmedium) and *Swaptions* (simmedium) belong to the PARSEC suite. The experimental results reported in this work correspond to the parallel phase of the evaluated benchmarks. Details about traffic patterns

Table 1: System parameters

Legend– BW: buffer writing, RC: route computation, VA: VC allocation, SA: switch allocation, ST: switch traversal, and LT: link traversal.

Memory Parameters	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	64KB, 8-way
L1 cache access time	2 cycles
Shared single L2 cache	512KB/tile, 16-way
L2 cache access time	6 cycles (2 if only tag accessed)
Directory cache	256 sets, 4 ways (same as L1)
Directory cache hit time	2 cycles
Memory access time	160 cycles

Network Parameters	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Buffer size	6 flits
Data and control message size	5 flits and 1 flit
Stages of the pipelined router	BW, RC, VA, SA, ST and LT

of SPLASH-2 and PARSEC benchmarks can be found in [32] and [33] respectively.

6. Experimental Evaluation

This section briefly explains the schemes that are considered with comparison purposes against the TF Architecture. Then, experimental results are presented and analyzed for both processor caches and directory caches. The benefits of the TF Architecture depend on the average number of ways that are looked up by the memory accesses. This number mainly changes depending on the accessed structure (first or second cache level or directory cache) and on the application behavior.

6.1. Compared Schemes

First, to evaluate the TF-Cache against other proposals that also reduce dynamic energy consumption by accessing a subset of the cache ways instead of all of them. These schemes are Way Prediction [10, 16] and two recent approaches: Way Guard [9] and PS-Cache [19].

Way Prediction techniques [10, 16] 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, after performing the comparison of the MRU complete tag, all the remaining ways are accessed at a second phase to look up the target block. This means that on misprediction, both energy wasting rises and latency increases, since additional cycles are required to solve the memory request.

Way Guard [9] has been proven to work efficiently in highly-associative caches. The mechanism implements a counting bloom filter associated to 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 searched. 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.

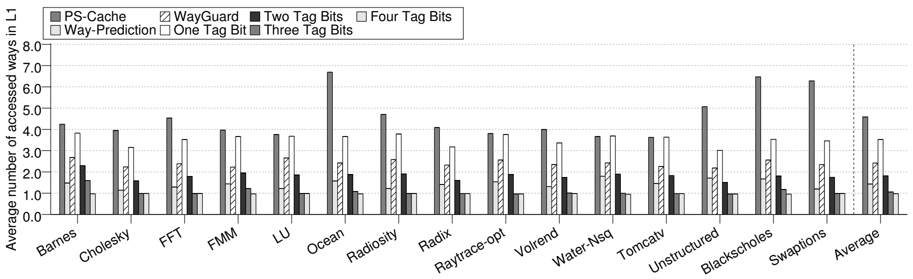
In the original paper, results are shown for m equal to $4\times$ the number of blocks in a cache. We will refer to this configuration as $WayGuard-4\times$. This approach requires a decoder with $4\times$ more outputs than the already implemented in the cache to index the target set.

The PS-Cache [19] tags cache blocks at run-time as shared or private according to a simple classification mechanism based on the OS. Upon an access to the PS-Cache, only those ways having the same type as the requested block are accessed.

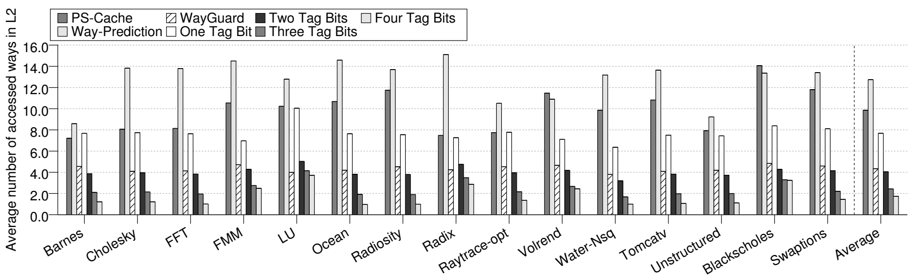
To evaluate the TF-Directory we implemented the Tag Filter Architecture in two directory schemes: in a convention single-level directory cache [21] and in the PS-Dir [26]. The latter approach, separates the directory in two different structures referred to as the Shared Cache and the Private Cache. The former is smaller and stores those entries keeping track of shared blocks, whereas the latter is bigger, lacks the sharer vector, and keeps track of private blocks. The PS-Dir relies on the fact that most of the directory hits are in shared entries, which we want to solve as quickly as possible, while the number of private entries is much higher than the number of shared ones. By having two different structures the necessities of each entry type can be addressed more adequately. Results will show how the Tag Filter mechanism affects directory caches with one or more levels.

6.2. TF Architecture in Processor Caches

Figure 5(a) shows the average number of searched ways in the 8-way L1 cache for the different studied techniques. As can be seen, the more bits (from 1 to 4) we use in the bit-array for filtering the ways, the less ways are accessed. On average, the number of accessed ways in a 8-way cache for 1-,2-,3-, and 4-bit tag filter is 3.53, 1.82, 1.06, and 0.98, respectively. This means that accesses follow a uniform distribution when considering the least significant bits. As expected using three bits suffices to limit the number of ways needed to be looked up to only a single one, since our first-level cache has 8 ways, therefore allowing the consumption of a set-associative cache that uses this mechanism to be similar as that of a direct-mapped cache. There is no high difference in the obtained results across the different applications for a given number of tag bits. Notice that, it is possible to have an average number of ways accessed lower than 1. It



(a) Average number of ways accessed in the 8-way L1 cache.



(b) Average number of ways accessed in the 16-way L2 cache.

Figure 5: Average number of ways accessed in the cache hierarchy.

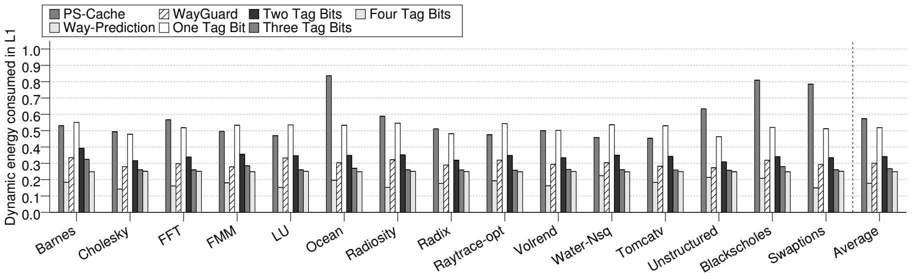
might happen that the least significant bits of the tag address have no match in the tag array. In this case, no way has to be accessed and the cache miss is triggered earlier than it would be without filtering. This explains the results obtained for four bits.

Compared to the PS-Cache, the proposal always achieves better results even with just a single tag bit (*i.e.* X equal to 1 bit). The PS-Cache accesses on average to 4.6 ways and the results widely vary from one application to another. In some cases like *Ocean* there is almost no access reduction, whereas in others (*i.e. Tomcatv*) it can reduce it by about 50%. This variability in the results is due to the high variation in the private-shared access pattern across the applications. WayGuard and Way-Prediction access on average 2.41 and 1.43 ways, which remains mostly constant along all the studied applications. Thus, they perform better than the proposal with a single bit. Two bits are enough to surpass WayGuard and a third one in order to surpass Way-

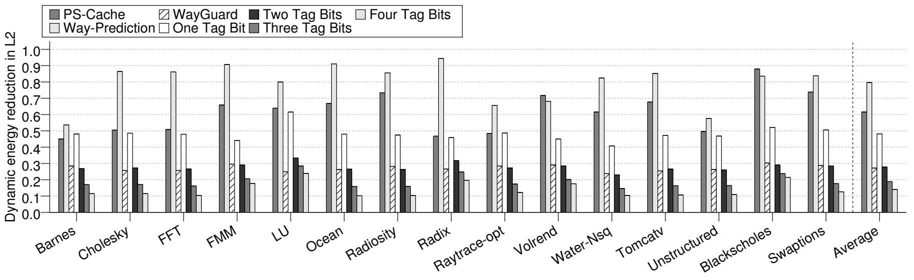
Prediction. Using the MRU way as a prediction does prove to be good enough for first-level caches providing a good hit ratio.

Figure 5(b) shows the average number of searched ways in the 16-way L2 cache. The number of ways accessed on average is by 7.68, 4.04, 2.43 and 1.74 for X equal to one, two, three, and four bits, respectively. As in the L1 cache, which was previously discussed, the reduction distributes evenly across all the studied applications. The trend shows that there is still room for improvement, but there has to be a limit to how big the tag array can be. In comparison, the PS-Cache, Way-Prediction and WayGuard access 9.85, 12.7 and 4.34 ways, respectively. Way-Prediction, which works really well for the L1 cache, performs poorly in lower levels of the cache hierarchy, since L1 caches filter many of the processor accesses, thus application locality is much poorer. When the prediction hits, only a way is accessed, but when it misses the remaining ways have to be accessed. Therefore, the figure shows a poor hit ratio in the LLC. Also it is worth to note, that a failed prediction also means a penalty in the access time since additional cycles are required in order to get the data information. That is, Way-Prediction is a hindrance for performance when applied in this level. Both Way-Prediction and the PS-Cache perform worse than the TF Architecture even with one bit, whereas WayGuard performs almost as well as the proposal when employing a two-bit tag array.

As a consequence of reducing the number of accessed ways in the data caches, the dynamic energy consumption is also reduced. Figure 6(a) shows the dynamic energy consumed by the first-level cache. Results have been normalized to those of a set-associative cache in which all ways are accessed, which also include the power overhead incurred by the extra comparators. The Tag Filter Cache is able to reduce the dynamic energy consumed by 48.1%, 65.8%, 73.2%, and 74.9% for a tag filter with one, two, three, and four bits, respectively. It can be seen that the marginal benefits of adding additional bits to the filter are fewer with each additional step. We can assume that results for a five-bit filter will not differ much from the ones shown for a four-bit one. As expected from the previous results, Way-Prediction shows the best results, being able to reduce dynamic energy consumption up to 82.1% in the Ocean application. The PS-Cache obtains the worst results, since it is also the scheme that accesses more ways. Analogously, Figure 6(b) depicts the same results, but for the L2 cache. The Tag



(a) Dynamic energy consumed in the 8-way L1 cache normalized to a conventional cache.



(b) Dynamic energy consumed in the 16-way L2 cache normalized to a conventional cache.

Figure 6: Dynamic energy consumed in the cache hierarchy.

Filter Cache is able to reduce consumption by 51.8%, 72.2%, 81.1%, and 85.9% for a tag filter with one, two, three, and four bits, respectively. Again one can see the diminishing benefits of further increasing the tag filter size. WayGuard achieves reductions similar as a 2-bit TF-Cache, whereas PS-Cache and Way-Prediction display no such improvements in comparison to the proposed architecture, reducing energy consumed only by 38.4% and 20.4%, respectively.

No performance evaluation is shown, since the mechanism introduces no access time penalty.

6.3. *TF Architecture in Directory Caches*

This section evaluates the TF Architecture implemented both, in a conventional single-level directory cache and in the recently proposed PS-Dir [26] approach. Experimental results assume an 8-way conventional directory cache and a PS-Dir with a

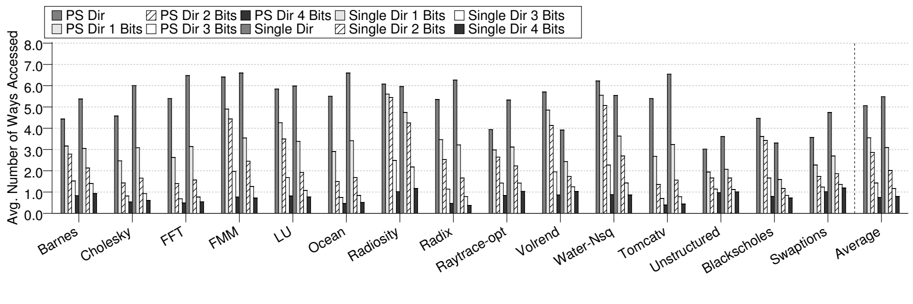


Figure 7: Average number of ways accessed in the directory per directory access.

2-way Shared cache and a 6-way Private cache (the configuration assumed in the original work). For each of them we evaluated the proposal ranging the filter size from 1 to 4 bits in 1-bit steps.

Figure 7 shows the average number of accessed ways on a cache access in the studied schemes. As can be seen, the base directory schemes access an average of 5.5 and 5.1 ways on each memory access, for the directory cache and the PS-Dir, respectively. Unless some few exceptions, PS-Dir always accesses fewer ways than the conventional directory cache. The TF Architecture further improves these numbers. As happened in the TF-Cache, the more bits are used in the filtering, the less ways are accessed. Just one bit is enough to reduce to 3.1 and 3.6 the average accessed ways. The TF-Directory is able to reduce as much as 0.8 and 0.7 accessed ways, for directory cache and PS-Dir respectively, when employing 4 bits. The tag filtering behaves almost identically in both directory protocols which means that it is applicable to any other cache directory scheme. Also, as happened in the TF-Cache, directory misses could be detected earlier with this architecture if there is no way match in the least significant bits.

Figure 8 shows the normalized dynamic energy consumed by the different directory configurations. The proposal is able to achieve energy reductions by 30.2%, 43.6%, 71.4% and 84.5% with 1, 2, 3, and 4 bits, respectively, in the PS-Dir. Analogously, reductions by 43.5%, 62.5%, 77.7%, and 84.2% are achieved in the directory cache. Single-level directories seem to profit more of the tag filtering than many-level ones

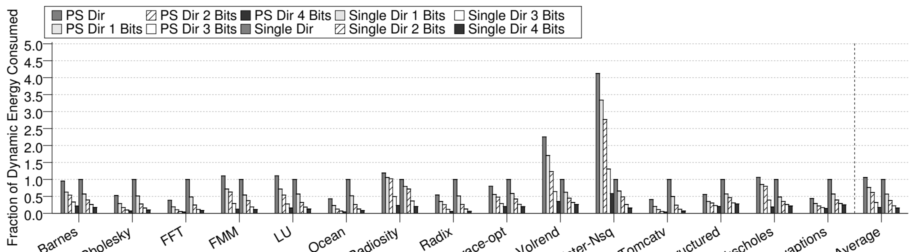


Figure 8: Normalized dynamic energy consumed by the directory.

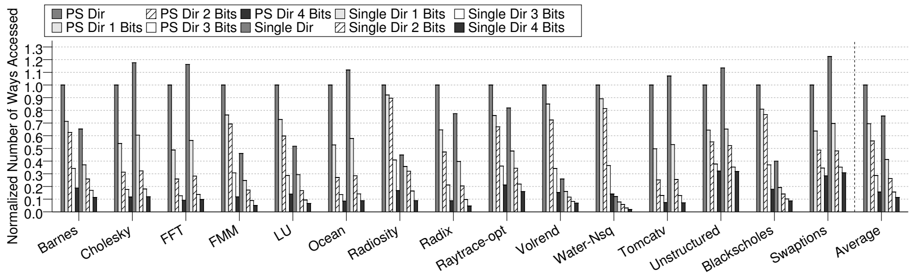


Figure 9: Normalized total number of accessed ways in the directory during execution time.

when few bits are selected in the filtering process. Nonetheless, results in energy consumed seem to converge as we increase this number of bits. Comparing this figure with Figure 7, it can be appreciated that there is no direct correlation between energy consumption and average number of accessed ways. For this purpose we should account for the total number of accesses, which varies among the studied schemes since they produce a different number of invalidations due to evictions of entries in the directory cache. Below we present these results.

Figure 9 shows the normalized total number of ways accessed in the directory cache along the complete execution of the applications. These results effectively confirm that there is a direct correlation between the total number of accessed ways and dynamic energy consumed by the directory. As such, the lower this number, the more energy can be saved. Remember that an access to the directory is triggered on a miss in the L1 processor cache for coherence maintenance. Even though the PS-Dir looks up a lower average number of ways per access than the directory cache, it looks up on overall a higher number of ways because it performs more accesses to the directory. This is due to a higher number of the so called coverage misses, *i.e.* misses that occur due to space constraints in the directory. The tag filter is able to decrease the number of ways accessed from 69.4% with one single bit up to 15.7% with 4 bits in the PS-Dir and from 56.5% to 15.8% in the conventional directory cache. Although there seems to be a convergence in the total number of accessed ways as the number of bits of the tag filter increases, better results are achieved when the TF Architecture is applied to the conventional directory cache in comparison to the PS-Dir. The reason is the higher number of ways assumed in the conventional directory.

In summary, the Tag Filter Cache and the Tag Filter Directory achieve significant energy gains, the more bits we use in the tag filter the more it surpasses recent approaches like WayGuard, Way-Prediction, PS-Cache and PS-Dir. Furthermore, this idea can be applied to any level of the cache hierarchy without inquiring in any performance degradation, which is the case of Way-Prediction in lower level caches.

7. Conclusions

Power consumption is a major design concern in current high-performance chip multiprocessors, which increases with the core count. On-chip caches often consume a significant fraction of the total power budget, and important research has focused on reducing energy consumption in these memory structures frequently at the cost of performance.

This work proposes the TF Architecture (TF-Cache and TF-Directory) to reduce dynamic power consumption in set-associative caches. The proposed mechanism saves a significant amount of energy by effectively reducing the number of searched ways. In this paper we have implemented and evaluated the proposed architecture both in processor caches (L1 and L2) and in the directory cache.

Results show that the TF-Cache can reduce up to 87.75% and 89.13% the average number of ways that are looked up on every access to the L1 and L2 caches, respectively; which translates in energy savings by 74.9% and 85.9%. Compared to other state-of-the-art schemes, TF-Cache achieves better results than the compared architectures, with the only exception of Way-Prediction in first-level caches by a small margin. Way-Prediction has proven to be ineffective when applied to other levels of the cache hierarchy, whereas our TF Architecture works better at any level.

The TF-Directory has been implemented and evaluated on a conventional single-cache directory and the PS-Dir (a state-of-the-art approach). Results show that up to 84% of the ways accessed by the conventional cache are filtered in both approaches, which translates in roughly the same percentage of energy savings.

All in all, we achieve significant energy benefits with minimal hardware complexity and without impacting applications performance.

Acknowledgments

This work has been jointly supported by MINECO and European Commission (FEDER funds) under the project TIN2015-66972-C5-1-R/3-R and by *Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia* under the project *Jóvenes Líderes en Investigación* 18956/JLI/13.

Vitae

Joan J. Valls received the MS degree in computer science from the UPV, Spain, in 2013. He is currently a PhD student at the Parallel Architecture Group (GAP) of the UPV with a fellowship from the UPV. His research interests include cache coherence protocols, and chip multiprocessor architectures.

Alberto Ros received the MS and PhD degree in computer science from the University of Murcia, Spain, in 2004 and 2009, respectively. In 2005, he joined the Computer Engineering Department at the same university as a PhD student with a fellowship from the Spanish government. He has been working as a postdoctoral researcher at the Universitat Politècnica de València and at Uppsala University. Currently, he is Associate Professor at the University of Murcia. His research interests include cache coherence protocols memory hierarchy designs, and memory consistency for manycore architectures.

María-Engracia Gómez obtained her MS and PhD degrees in Computer Science from the UPV, Spain, in 1996 and 2000, respectively. She joined the Department of Computer Engineering (DISCA) at UPV in 1996 where she is currently an Associate Professor of Computer Architecture and Technology. Her research interests are in the field of interconnection networks, network-on-chips and cache coherence protocols.

Julio Sahuquillo received his BS, MS, and PhD degrees in Computer Engineering from the UPV, Spain. Since 2002 he is an associate professor at the DISCA department at the UPV. He has published more than 100 refereed conference and journal papers. His current research topics include multi- and manycore proces-

sors, memory hierarchy design, photonic interconnects, and power dissipation.

Bibliography

[1] R. Balasubramonian, N. P. Jouppi, N. Muralimanohar, Multi-Core Cache Hierarchies, Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2011.

[2] Intel Xeon Phi Coprocessor, `http://software.intel.com/en-us/mic-developer` (Apr. 2013).
URL `http://software.intel.com/en-us/mic-developer`

[3] A. Ros, B. Cuesta, R. Fernández-Pascual, M. E. Gómez, M. E. Acacio, A. Robles, J. M. García, J. Duato, EMC^2: Extending magny-cours coherence for large-scale servers, in: 17th Int'l Conf. on High Performance Computing (HiPC), 2010, pp. 1–10.

[4] A. K. Nanda, A.-T. Nguyen, M. M. Michael, D. J. Joseph, High-throughput coherence control and hardware messaging in Everest, IBM Journal of Research and Development 45 (2) (2001) 229–244.

[5] A. Ros, M. E. Acacio, J. M. García, Scalable directory organization for tiled CMP architectures, in: Int'l Conference on Computer Design (CDES), 2008, pp. 112–118.

[6] A. Ros, M. E. Acacio, J. M. García, A scalable organization for distributed directories, Journal of Systems Architecture (JSA) 56 (2-3) (2010) 77–87.

[7] S. Kaxiras, Z. Hu, M. Martonosi, Cache decay: Exploiting generational behavior to reduce cache leakage power, in: 28th Int'l Symp. on Computer Architecture (ISCA), 2001, pp. 240–251.

[8] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, T. Mudge, S. Kaxiras, Z. Hu, M. Martonosi, Drowsy caches: Simple techniques for reducing leakage power, in: 29th Int'l Symp. on Computer Architecture (ISCA), 2002, pp. 148–157.

[9] M. Ghosh, E. Özer, S. Ford, S. Biles, H.-H. S. Lee, Way guard: A segmented counting bloom filter approach to reducing energy for set-associative caches, in: Int'l Symp. on Low Power Electronics and Design (ISLPED), 2009, pp. 165–170.

[10] B. Calder, D. Grunwald, Predictive sequential associative cache, in: 2nd Int'l Symp. on High-Performance Computer Architecture (HPCA), 1996, pp. 244–253.

[11] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, B. Franke, Cooperative partitioning: Energy-efficient cache partitioning for high-performance cmps, in: 18th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2012, pp. 311–322.

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, N. P. Jouppi, Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures, in: 42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2009, pp. 469–480.

[13] J. Lee, S. Hong, S. Kim, Tlb index-based tagging for cache energy reduction, in: 17th Int'l Symp. on Low Power Electronics and Design (ISLPED), 2011, pp. 85–90.

[14] A. Ros, P. Xekalakis, M. Cintra, M. E. Acacio, J. M. García, Ascib: Adaptive selection of cache indexing bits for reducing conflict misses, in: Int'l Symp. on Low Power Electronics and Design (ISLPED), 2012, pp. 51–56.

[15] M. Powell, S. hyun Yang, B. Falsafi, K. Roy, T. N. Vijaykumar, Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories, in: Int'l Symp. on Low Power Electronics and Design (ISLPED), 2000, pp. 90–95.

[16] D. H. Albonesi, Selective cache ways: On-demand cache resource allocation, in: 32nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 1999, pp. 248–259.

[17] C. Zhang, F. Vahid, J. Yang, W. Najjar, A way-halting cache for low-energy high-performance systems, ACM Trans. Archit. Code Optim. 2 (1) (2005) 34–54.

[18] M. Ghosh, E. Özer, S. Biles, H.-H. S. Lee, Efficient system-on-chip energy management with a segmented bloom filter, in: 19th Int'l Conf. on Architecture of Computing Systems (ARCS), 2006, pp. 283–297.

[19] J. J. Valls, A. Ros, J. Sahuquillo, M. E. Gómez, PS-cache: An energy-efficient cache design for chip multiprocessors, in: 22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2013, pp. 407–408.

[20] K. Kedzierski, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, M. Valero, Power and performance aware reconfigurable cache for cmps, in: 2nd Int'l Forum on Next-Generation Multicore/Manycore Technologies, 2010, pp. 1–12.

[21] A. Gupta, W.-D. Weber, T. C. Mowry, Reducing memory traffic requirements for scalable directory-based cache coherence schemes, in: Int'l Conf. on Parallel Processing (ICPP), 1990, pp. 312–321.

[22] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, J. Duato, Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks, in: 38th Int'l Symp. on Computer Architecture (ISCA), 2011, pp. 93–103.

[23] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, J. Duato, Increasing the effectiveness of directory caches by avoiding the tracking of non-coherent memory blocks, IEEE Transactions on Computers (TC) 62 (3) (2013) 482–495.

[24] D. Sanchez, C. Kozyrakis, SCD: A scalable coherence directory with flexible sharer set encoding, in: 18th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2012, pp. 129–140.

[25] J. Zebchuk, B. Falsafi, A. Moshovos, Multi-grain coherence directories, in: 46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2013, pp. 359–370.

[26] J. J. Valls, A. Ros, J. Sahuquillo, M. E. Gómez, J. Duato, PS-Dir: A scalable two-level directory cache, in: 21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 451–452.

[27] 27-inch imac, technical specifications, available online (nov, 2014) at http://www.apple.com/imac/specs/.

[28] P. S. Magnusson, M. Christensson, J. Eskilson, et al, Simics: A full system simulation platform, IEEE Computer 35 (2) (2002) 50–58.

[29] M. M. Martin, D. J. Sorin, B. M. Beckmann, et al, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, Computer Architecture News 33 (4) (2005) 92–99.

[30] N. Agarwal, T. Krishna, L.-S. Peh, N. K. Jha, GARNET: A detailed on-chip network model inside a full-system simulator, in: IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS), 2009, pp. 33–42.

[31] N. Muralimanohar, R. Balasubramonian, N. P. Jouppi, Cacti 6.0, Tech. Rep. HPL-2009-85, HP Labs (Apr. 2009).

[32] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: 22nd Int'l Symp. on Computer Architecture (ISCA), 1995, pp. 24–36.

[33] C. Bienia, S. Kumar, J. P. Singh, K. Li, The PARSEC benchmark suite: Characterization and architectural implications, in: 17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2008, pp. 72–81.