The Tag Filter Cache: An Energy-Efficient Approach

Joan J. Valls[*], Julio Sahuquillo[*], Alberto Ros[†], María E. Gómez[*]

[*]Department of Computer Engineering
Universitat Politècnica de València (Spain)
joavalmo@fiv.upv.es, {jsahuqui,megomez}@disca.upv.es

[†]Dept. de Ingeniería y Tecnología de Computadores
Universidad de Murcia (Spain)
aros@ditec.um.es

***Abstract*—Power consumption in current high-performance chip multiprocessors (CMPs) has become a major design concern. The current trend of increasing the core count aggravates this problem. On-chip caches consume a significant fraction of the total power budget. Most of the proposed techniques to reduce the energy consumption of these memory structures are at the cost of performance, which may become unacceptable for high-performance CMPs. On-chip caches in multi-core systems are usually deployed with a high associativity degree in order to enhance performance. Even first-level caches are currently implemented with eight ways. The concurrent access to all the ways in the cache set is costly in terms of energy.**

In this paper we propose an energy-efficient cache design, namely the Tag Filter Cache (TF-Cache) architecture, that filters some of the set ways during cache accesses, allowing to access only a subset of them without hurting the performance. Our cache for each way stores the lowest order tag bits in an auxiliary bit array and these bits are used to filter the ways that do not match those bits in the searched block tag. Experimental results show that, on average, the TF-Cache architecture reduces the dynamic power consumption up to 74.9% and 85.9% when applied to the L1 and L2 cache, respectively, for the studied applications.

I. Introduction

As silicon resources become increasingly abundant, core counts grow rapidly in successive chip-multiprocessors (CMPs) generations. These CMPs usually accelerate their memory access by using one or more levels of caches, being them responsible of a significant percentage of the overall CMP die area [4] and of an important consumption of the overall power budget. Most of that power is due to dynamic power (the switching of transistors during accesses). Some of that power is referred to as static power (current leaking even when the cache is not being accessed). Cache designers must provide a compromise among performance, cost, size, and power/energy dissipation.

Concerning dynamic energy consumption, it is dominated by the first levels of the cache hierarchy because they are more highly accessed than last level caches (LLC), e.g., L3 caches, which usually are scarcely accessed. Moreover this high consumption is also due to the fact that both tag and data are accessed in parallel, because the first levels of caches have a strong influence on the overall processor performance. 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, these caches are deployed with a high associativity degree. In high-performance microprocessors, all the ways in the target set are concurrently accessed on a cache access. Therefore, the associativity degree defines the number of tags that are looked up in parallel in each access. Caches include one comparator per way and compare as many tags as number of ways. As a consequence, the dynamic energy dissipated per access increases with the cache associativity.

Generally, the design of low-dynamic power cache focuses in minimizing the internal transistor activity during a cache access. That activity comes from reading and comparing tags in tag arrays, and from reading or writing data in data arrays. Ideally, on a cache hit, the cache would read and compare only one tag entry, and accessed one data entry without losing performance. Furthermore, on a miss, ideally the cache would have no need to access neither the tag array nor the data array. In fact, on a miss, the cache does not even have to access a complete tag entry, since just one mismatched tag bit is enough to determine a miss.

Many cache energy reduction approaches have focused on monolithic processors in the past (such as Cache Decay [10], Drowsy Caches [7] and Way Guard [9]). Some of them, *e.g.*, [6], 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 or can be improved, recent research has dealt with energy savings on CMPs when running parallel workloads.

In this paper we propose the Tag Filter Cache (TF-Cache), a cache architecture that reduces the number of tags and data blocks checked when accessing the cache hierarchy. TF-Cache can be applied to any level of the cache hierarchy with the aim of reducing dynamic power consumption in the cache structures. It is based on using the least significant bits (LSB) of the tag part of the address in order to discern which ways of a set-associative cache may contain the searched block. Only those ways that may contain the block are accessed, thus saving the energy required to access the other ways.

The TF-Cache is deployed with minimal hardware complexity. Moreover, tags and data arrays can be accessed in parallel so no performance degradation rises, which is a major concern in L1 caches. Unlike other approaches such as [19] no way-alignment across sets must be done.

Experimental results show that the TF-Cache architecture can reduce dynamic power consumption up to 74.9% and 85.9% in the L1 and L2 cache, respectively, achieving better results than recent works.

The remainder of this work is organized as follows. Section II describes the main reasons that motivate us to carry out this research. Section III discusses the related work. Section IV presents the proposal. Section V describes the simulation framework. Section VI presents way accesses and energy results. Finally, Section VII draws some concluding remarks.

II. Motivation

Cache memories, especially first- and second-level caches, are frequently accessed, since memory reference instructions represent a significant percentage of the executed instructions. A significant fraction of the total power budget is often consumed by on-chip caches such as in the Niagara2 [13] processor, where 44% of the chip power is consumed by the L2 cache. Reducing dynamic power consumption in caches of CMPs is an actual problem that is being under research [12] [19]. To deal with this problem, this paper proposes an architectural approach with the aim of taking advantage of the homogeneous distribution of the least significant bits of the tag address across the ways of a set-associative cache.

We launched experiments to verify this hypothesis in the studied workloads. Figure 1 shows the average distribution of the blocks across a 8-way L1 cache and a 16-way L2 cache on a 16-core CMP system[1]. As can be seen in Figures 1(a) and 1(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. There is no need to access all ways in a set if there is some mechanism that is able to filter the access to only the subset of ways which might allocate the requested block. An homogeneous distribution as the one just mentioned (*i.e.* the tag lowest order bits) makes therefore a perfect candidate method for a filtering mechanism.

The aim of this paper is to save dynamic 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 whose least significant bits of the tag match the ones of the requested block.

III. Related Work

This paper presents an energy-efficient cache design that takes advantage of the least significant bits of the tags of the blocks referenced by the applications. Hence, this section reviews some related work about energy-efficient cache designs.

Caches consumption comes from both leakage (or static) and dynamic power consumption. Regarding leakage savings, Powell *et al.* [17] proposed a Gated-Vdd technique that aims to reduce leakage for instruction caches by reconfiguring them and turning off unused lines. Kaxiras *et al.* [10] 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.* [7] exploited the fact that in a particular period of time only a subset of the cache lines are accessed to propose Drowsy Caches. Different from the previous proposals, the voltage is reduced but not cut off for those cache lines that are not being accessed. Consequently, the content of the cache line is not preserved.

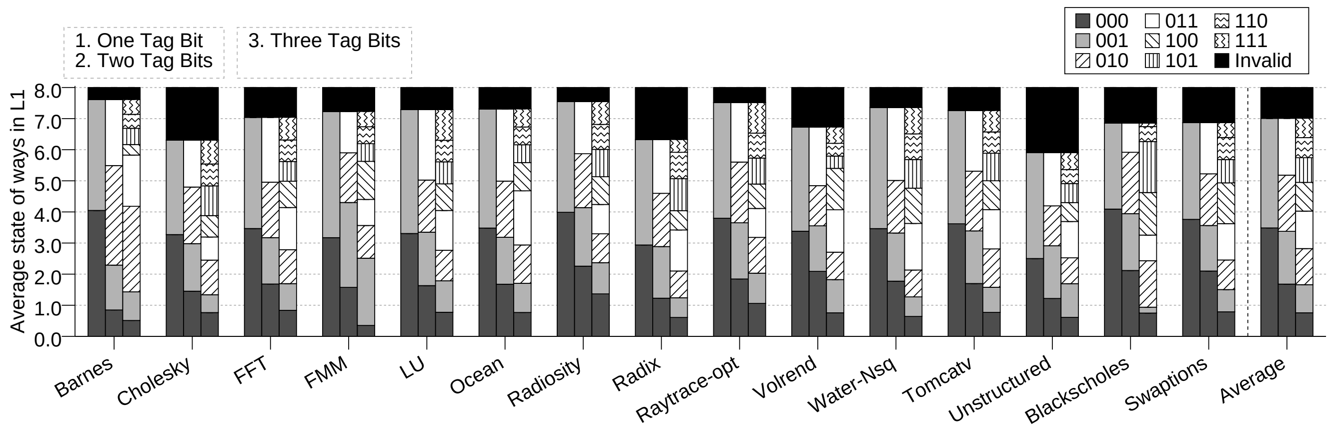
While techniques that aim to save leakage focus on reducing (or cutting off) voltage, dynamic energy saving techniques try to minimize the number of data read and written on every cache access. For example, Albonesi [3] proposed Selective Cache Ways, a cache design that enables only a subset of the cache ways when the cache activity is not high. The prediction of ways was previously proposed by Calder *et al.* [6] to reduce the access time of set-associative caches. This approach works well in L1 caches with a relatively small associativity which present very predictable access patterns; however, as we show in the evaluation, this mechanism presents poor results for lower level caches since locality is hidden by previous cache levels.

Zhang *et al.* [22] proposed the way-halting cache that 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. Unless 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 tag address needs to be translated by a translation look-aside buffer (TLB), then the halt array, where the low order bits are stored, lookup cannot proceed. We do not compare against this scheme. 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.

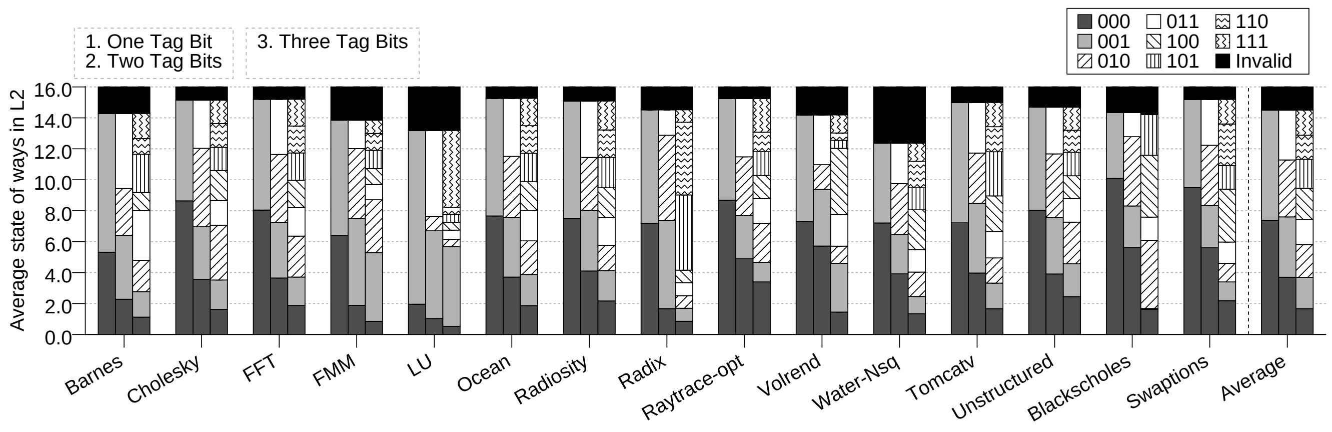
Ghosh *et al.* [9] proposed Way Guard, a mechanism for large set-associative caches that employs bloom filters to reduce dynamic energy by skipping the look up of cache ways that do not contain the requested data according to the bloom filter. This scheme requires the addition of a large decoder and a two fields per way: one segmented bloom filter, previously proposed by the same authors to filter accesses to the whole cache [8], 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 cache. A quantitative comparison with Way Guard is shown in the evaluation section.

Valls *et al.* [20] proposed PS-Cache, 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

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



(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

Fig. 1. Average number of ways in a set of each type in the cache hierarchy.

are searched. The PS-Cache has been also implemented and compared quantitatively against our TF-Cache.

Finally, other recently proposed 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 [12], by employing direct mapped caches along with mechanisms to remove conflict misses, like in ASCIB [18], or by performing run-time partitioning, like in the Cooperative Caching scheme [19] or in the ReCaC scheme [11].

IV. The Tag Filter Cache

The main goal of the Tag Filter Cache is to reduce 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 aim is to reduce dynamic power consumption in these structures. In a typical cache access, to check if the searched block is in cache, the entire tags in the target set are compared and, at most, one of those tags will match, while the other ones will mismatch. In first level caches, all the ways of the set in the data array are looked up at the same time, before knowing whether or not the target block is in the set. This paper proposes filtering the access to those ways (tag and data) that are expected to mismatch the tag comparison.

Figure 2 depicts a block diagram of the proposed approach for a L1 cache. The filter consists in comparing only a subset of bits of the tags, say X. For this purpose the tag array is decoupled in two 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 the X-bit wide structure to reduce the number of accessed ways.

The proposed mechanism performs the tag comparison in two stages. At the first stage, only the X least significant bits are checked in all the ways of the target set. The few number of least significant bits used in the mechanism, allows the first comparison to be done faster, introducing negligible time penalty. At the second stage, the remaining bits of the tag are compared to the corresponding tag bits of the virtual address

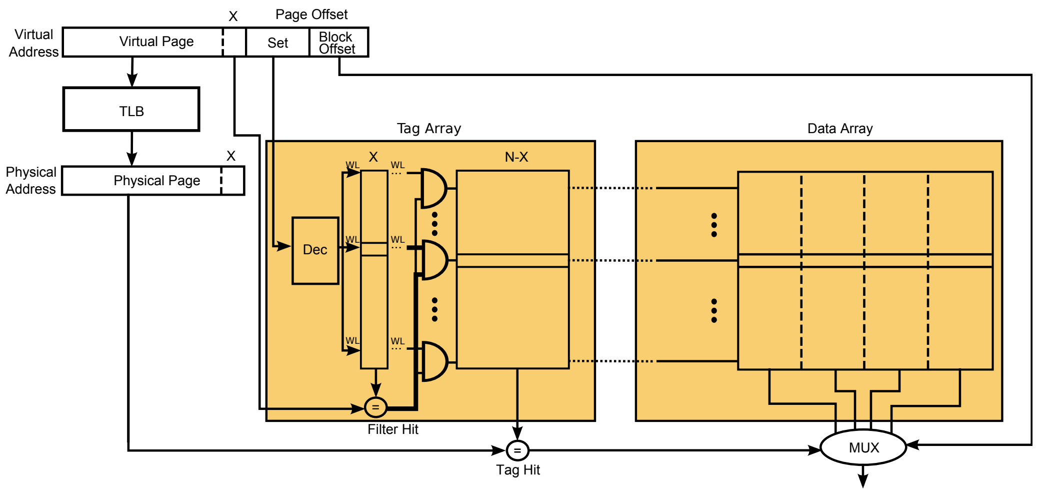


Fig. 2. The TF-Cache architecture for L1 caches.

of the block that is being searched. This second comparison, which implies more bits, is only performed in the ways that succeed the first comparison.

To allow the mechanism to work in current VIPT (virtually indexed physically-tagged) 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 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 a uniform page address distribution is expected and main memory capacities are by four orders of magnitude bigger than page sizes (e.g. a 32GB main memory [1] and a 4KB page size), which allows the OS to have some allocation flexibility. Under this assumption, the first comparison can be done once the output of the set decoder is provided, while the address translation in the TLB is still pending. Once the TLB provides the result, then the remaining $N-X$ bits of the tag array are compared with the $N-X$ provided by the TLB in a second comparison, but only for those ways not filtered in the first comparison.

TF-Cache only accesses those ways in the data array that succeed the first comparison. Notice that in a conventional cache, all the tag bits are compared in parallel in all the set ways and all the data ways are accessed. As proven in the evaluation section, the mechanism may provide important energy benefits.

Regarding complexity, the proposal requires minimal hardware to be adapted to current caches. As mentioned above the tag array is decoupled in two independent structures. Then 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 2, the wordline is allowed to drive both the wide tag structure and the data array for a given way, 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. This design allows significantly reduce dynamic energy consumption across the cache accesses since in general, as experimental results will show, only a small fraction of ways is compared in most of the accesses.

TABLE I
SYSTEM PARAMETERS

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
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

V. Simulation Environment

We evaluate our proposal with a full-system simulation using Virtutech Simics [14] and the Wisconsin GEMS toolset [15], which enables detailed simulation of multiprocessor systems. GARNET [2], a detailed network simulator included in the GEMS toolset, models the interconnection network.

Table I 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 [16] 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 cache hierarchy with private L1 caches and a shared L2 NUCA distributed among all tiles. A directory-based cache coherence protocol keeps coherence for the data within the private caches.

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 [21]. *Tomcatv* (256 points) and *Unstructured* (Mesh.2K) are two scientific benchmarks. *Blackscholes* (simmedium) and *Swaptions* (simmedium) belong to PARSEC suite [5]. The experimental results reported in this work correspond to the parallel phase of the evaluated benchmarks.

VI. Experimental Evaluation

This section briefly explains the schemes used for a comparative evaluation and analyzes the experimental results obtained.

A. Compared Schemes

We compare the proposed scheme 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 and two recent approaches: Way Guard and PS-Cache.

Way Prediction techniques [3], [6] 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 up the target block. This means that on missprediction, 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 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.

The PS-Cache [20] tags cache blocks at run-time as shared or private according to a simple classification mechanism based on the page table information. Upon an access to the PS-Cache, only those ways having the same type as the requested block are accessed.

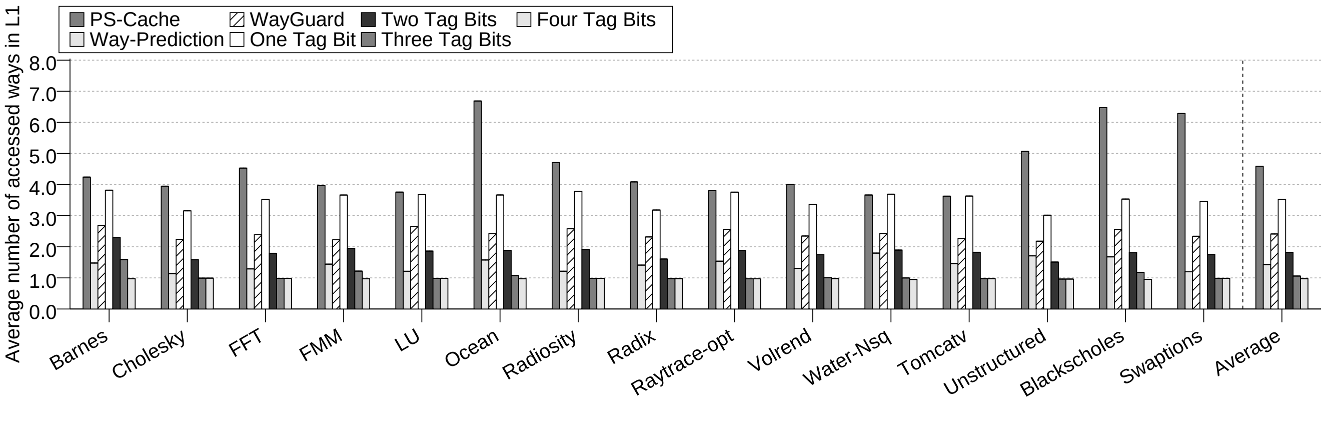
B. Experimental results

Benefits of the proposal depend on the average number of ways that are looked up by the cache accesses. This number mainly changes depending on the accessed cache (L1 or L2) and on the applications behavior.

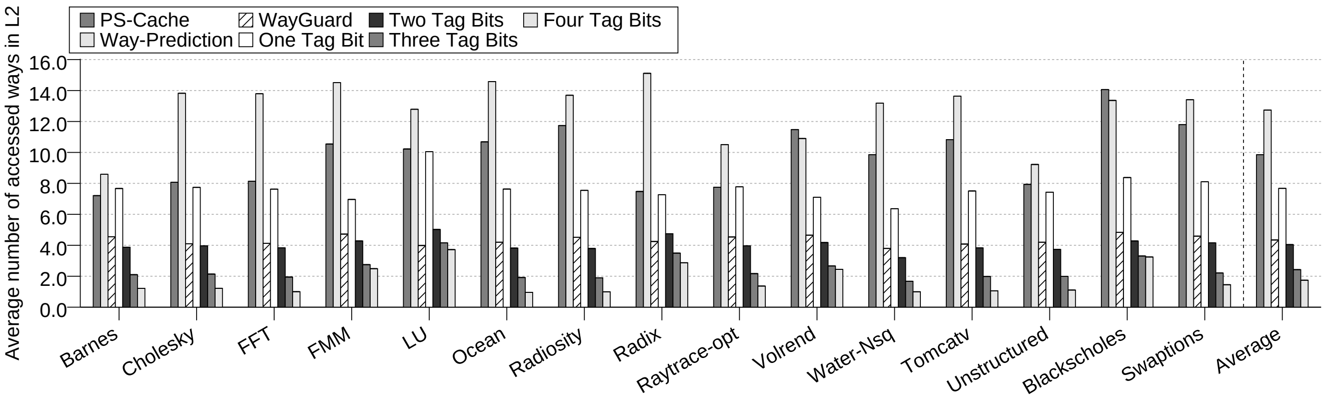
Figure 3(a) shows the average number of searched ways in the 8-way L1 cache for the different studied techniques. As can be seen, the more bits (from 1 to 4) we use in the bit-array for filtering the ways, the less ways are accessed. On average, for a 8-way cache when using a single tag bit 3.53 ways are accessed, 1.82 with two bits, 1.06 with three bits and 0.98 with four. This means that accesses follow a uniform distribution when considering the less 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 to that of a direct-mapped cache. There is no high difference in the results obtained between the different applications for a same number of tag bits. Notice that, it is possible to have an average number of ways accessed lesser than 1. It 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 a bit earlier than it would be. 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, that is, X equal to 1 bit. The PS-Cache accesses on average to 4.6 ways and the results greatly 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 more or less 50%. The private-shared access pattern varies between the applications greatly, hence these results. 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 3(b) shows the average number of searched ways in the 16-way L2 cache. The number of ways accessed on average in this level of the cache hierarchy is 7.68, 4.04, 2.43 and 1.74 for X equal to one, two, three and four bits respectively. As happened in the first-level 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 respectively. Way-Prediction, which works really well for the L1 cache, performs poorly in lower levels of the cache hierarchy, since L1 filters many of the processor accesses, thus application locality is much poorer. When the prediction hits, only a way is accessed and when it misses the remaining ways



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



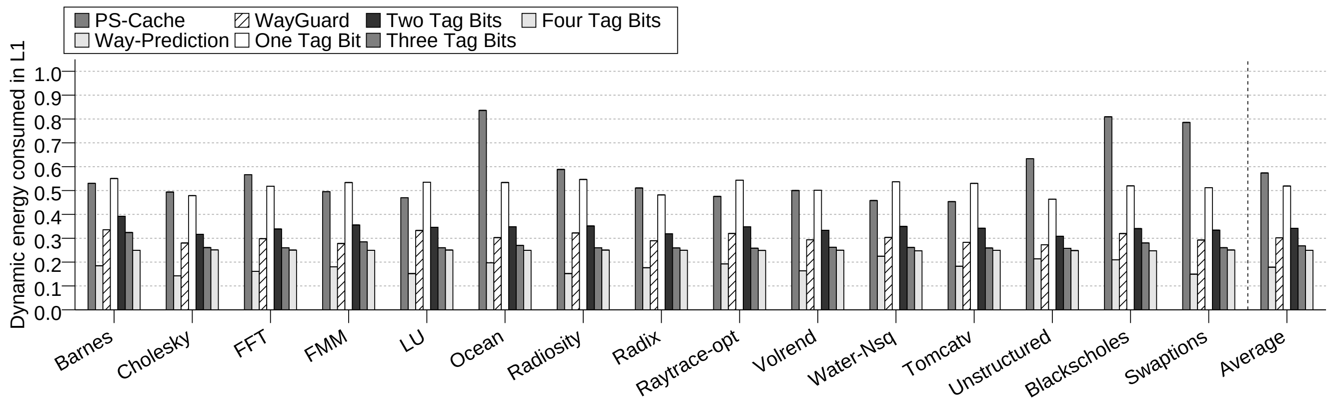
(b) Average number of ways accessed in the 16-way L2 cache.

Fig. 3. Average number of ways accessed in the cache hierarchy.

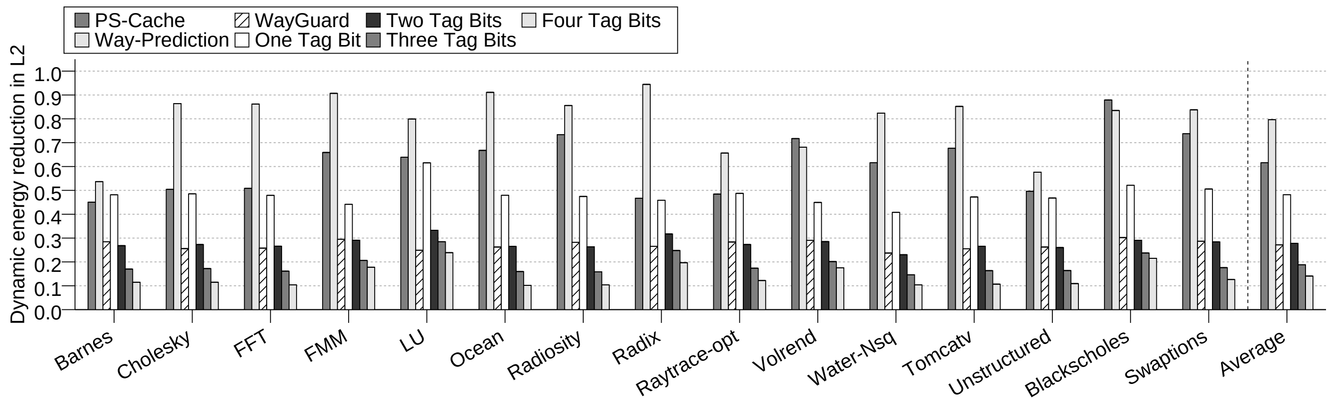
have to be accessed. The figure then shows a poor hit ratio in the LLC. Also it is worth to note, that a failed prediction also means additional cycles in order to get the data information. Way-Prediction then is a hindrance for performance when applied in this level. Both Way-Prediction and the PS-Cache perform worse than our proposal even with one bit, whereas WayGuard performs almost as well as the proposal when employing a two-bit tag array.

Figure 4(a) shows the dynamic energy consumed by the first-level cache evaluated in this work. Results have been normalized to those of a set-associative cache in which all ways are accessed. 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 was expected due to the results previously shown, Way-Prediction achieves the best results, being able to reduce energy consumption up to 82.1%. Meanwhile, the PS-Cache obtains the worst results, since it is also the scheme that accesses more ways. Analogously, Figure 4(b) depicts the same results, but for the L2 cache. The Tag 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. WayGuard achieves reductions similar to a two 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.

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



(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.

Fig. 4. Dynamic energy consumed in the cache hierarchy.

VII. Conclusions

One of the major design concerns in current high-performance chip multiprocessors is power consumption, which increases as the core count grows. 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 at the cost of performance.

In this work, we have proposed TF-Cache, an energy-efficient cache design which only accesses a subset of the set ways without hurting performance. The proposal divides the tag array in two different segments. One of them, with few of the least significant bits and the other with the rest of bits of the tag. In order to filter the set ways, two comparisons are performed. In the first comparison the least significant bits in the tag of the searched block are compared to the least significant bits stored in all the ways of the set. This comparison is performed very fast without waiting for the TLB output. Once we have the result of this comparison, the second is performed, comparing the rest of bits in the tag, but only for those ways that succeed the first comparison. If the data array is accessed in parallel with the tag array, then only those ways matching the first comparison are accessed in the data array. This way filters cache accesses and allows achieving important dynamic energy reductions. This filter choice is appropriate since, as results show, there is rather homogeneous distribution of the bit array field contents across the various ways of a set. This cache design can be implemented in any cache level of the cache hierarchy, although the most frequently accessed one will achieve the best energy savings. Also, the higher the associativity a cache implements, the more potential benefits TF-Cache can bring.

Results have shown that TF-Cache can reduce up to 87.75% and 89.13% the average number of looked up ways when applied to the L1 and L2 cache, respectively. This translates in energy savings, more specifically, the TF-Cache scheme reduces dynamic consumption by 74.9% and 85.9% when applied to the L1 and L2 cache, respectively. 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. Regretfully, Way-Prediction has proven to be ineffective when applied to other levels of the cache hierarchy, whereas the proposal works best at any level.

Acknowledgments

This work has been jointly supported by MINECO and European Commission (FEDER funds) under the project TIN2012-38341-C04-01/03 and by *Fundación Seneca-Agencia de Ciencia y Tecnología de la Región de Murcia* under the project *Jóvenes Líderes en Investigación* 18956/JLI/13.

References

[1] 27-inch imac, technical specifications, available online (nov, 2014) at http://www.apple.com/imac/specs/.

[2] N. Agarwal, T. Krishna, L.-S. Peh, and 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)*, Apr. 2009, pp. 33–42.

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

[4] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *Multi-Core Cache Hierarchies*, ser. Synthesis Lectures on Computer Architecture, M. D. Hill, Ed. Morgan & Claypool Publishers, 2011.

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

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

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

[8] M. Ghosh, E. Özer, S. Biles, and 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)*, Mar. 2006, pp. 283–297.

[9] M. Ghosh, E. Özer, S. Ford, S. Biles, and 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)*, Aug. 2009, pp. 165–170.

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

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

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

[13] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and 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)*, Dec. 2009, pp. 469–480.

[14] P. S. Magnusson, M. Christensson, and J. Eskilson, et al, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[15] M. M. Martin, D. J. Sorin, and B. M. Beckmann, et al, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[16] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0," HP Labs, Tech. Rep. HPL-2009-85, Apr. 2009.

[17] M. Powell, S. hyun Yang, B. Falsafi, K. Roy, and 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)*, Jul. 2000, pp. 90–95.

[18] A. Ros, P. Xekalakis, M. Cintra, M. E. Acacio, and 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)*, Jul. 2012, pp. 51–56.

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

[20] J. J. Valls, A. Ros, J. Sahuquillo, and 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)*, Sep. 2013, pp. 407–408.

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

[22] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, pp. 34–54, Mar. 2005.