IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727
PP 37-41
www.iosrjournals.org

SWIFT AND LOW POWER L2 CACHE ARCHITECTURE USING PARTIAL TAG BLOOM FILTER

Arunkumar A K [1], Boobalan R[2]
[1]*(PG Student, Dhanalakshmi Srinivasan College of Engineering, Coimbatore)*
[2]*(Assistant Professor, Dhanalakshmi Srinivasan College of Engineering, Coimbatore)*

ABSTRACT : Today's trend toward high speed as well as low power processors are the development of different level of cache for write through techniqe . To improve the execution time there for decreases the power uses , tag of cache L2 placed in L1 cache ,it reduces the power by 62%. Further decrease of power by partial tag enhanced Bloom filter to improve the accuracy of the cache miss prediction method reduce the tag comparisons of the cache hit prediction method. Here also combine both methods so that their order of application can be dynamically adjusted to adapt to changing cache access behavior, which further reduces execution time. To overcome the common limitation of multistage tag comparison methods . Experimental results showed that the new method reduces the energy consumption of tag comparison by an average of 88.40%, which translates to an average reduction of 35.34% (40.19% with low-power data access) in the total energy consumption of the L2 cache and a further reduction of 8.86% (10.07% with low-power data access) when compared with existing methods.

Keywords - Cache, Bloom filter , power consumption, partial tag, way prediction, way tag.

I. INTRODUCTION

The cache is a crucial component for managing the memory wall problem. In particular, the L2 cache has become increasingly popular in chips for high-end embedded systems such as smart phones and tablet PCs . The cache size ranges from 256 kB to 1MB and is expected to increase further to meet the ever-increasing bandwidth requirements of high-end applications. The ever-growing usage and large area of the L2 cache result in significant power consumption[1] .

L2 caches for high-end embedded systems are characterized by high switching power, particularly due to the large amount of power consumed in tag-comparison operations. It is due to two factors. First, the L2 cache is characterized by high associativity compared with the L1 cache. The main reason for adopting a high associativity is to reduce conflict misses. For instance, Ultra Sparc adopts 16-way L2 caches and a commercial L2 cache, and PL310 supports 8-way and 16- way caches . Tag comparison in highly associative caches is characterized by considerably higher power consumption than in typical 2-way or 4-way L1 caches. For instance, in the case of a 16-way, set-associative L2 cache, tag comparisons with16 tags (each 4 B in size) require that a total of 64 B of data behead from the tag array, which consumes a similar amount of energy as data access to a 64 B cache line. Second, the power consumed in tag comparison is expected to increase further because multicore processors require cache coherence . In broadcast-based snoopy coherence protocols, snoop traffic can account for a significant portion of the total inter-cache traffic. Each packet of snoop traffic requires tag comparisons to check the existence of an address in each of all the caches receiving the snoop request.

Several studies have investigated the reduction of tag comparisons within the L2 cache. Such studies can be classified into two categories based on the method used for reducing tag comparisons: cache hit prediction and cache miss prediction . The cache hit prediction methods typically use two-step tag comparison. In the first step, tags are compared for cache ways that are likely to yield a cache hit. If the prediction is successful, the power consumption otherwise needed for comparison with the other tags can be saved. However, if the prediction fails, tag comparisons with the remaining tags are performed in the next cycle, thereby resulting in an additional latency cycle. Because tag comparisons are performed even in the case of cache misses, the cache hit prediction methods may lead to high power consumption in applications with high cache miss rates. The cache miss prediction methods try to predict cache misses, e.g., based on the Bloom filter[2] . If the prediction is correct, tag comparisons are skipped, thereby saving the energy consumed in tag comparison. Improving the prediction accuracy (e.g., by using a larger Bloom filter) further reduces the amount of energy consumed in tag comparison. However, such an accurate prediction of cache misses requires more overhead of energy consumption and chip area (e.g., due to the large Bloom filter).

In this paper, use a novel tag comparison method that exploits predictions of both cache hits and misses. The contributions of work are as follows:

1) a partial tag based Bloom filter to improve the energy efficiency of cache miss predictions;
2) dynamic reordering of cache hit/miss prediction methods for multistage tag comparison that can further reduce the energy consumed in tag comparison by adapting to dynamically changing cache access behaviour;
3) performance guaranteed it between cache energy consumption and additional latency overhead, especially in the case of latency-critical programs.

I. WAY TAG CACHE

Low-power cache architectures are classified into three categories depending on the targeted potion of cache power consumption: tag comparison, data access, and leakage presented a way prediction method that predicts the most recently used (MRU) [4]way as the candidate likely to be hit and performs tag comparison with the MRU way for a fast hit. If the prediction fails, the tags in the other ways are compared for a slow hit in the next cycle. Powell utilized selective direct mapping, which tries to maximize direct mapping) to reduce tag comparison energy consumption. Their method combines the above-mentioned way prediction and selective direct mapping for further energy reduction in the L1 cache. A way prediction method called multiple MRU is presented that predicts an MRU way per partial tag to improve the way prediction accuracy. Dai and Wang presented a way-tagged cache to reduce L2 cache tag accesses in case of a write-through L1 cache. The L1 cache maintains the way tags of the L2 cache and allows the request to access only the required way in the L2 cache during write operations.

Many high-performance microprocessors employ cache write-through technique for performance improvement and at the same time to avoid soft errors in caches. Write-through policy has a large energy overhead due to the increased accesses to caches at the lower level during write operations way-tagged cache to improve the energy efficiency of write-through cache technique . By maintain the way tags of L2 cache in the L1 cache during read operations, this technique enables[5] , check in L1 cache after that the way tag of L2 in L1 is checked , so no need of L2 if data is not in way tag. This tends to reduce energy for working and no performance decrease .It shown in figure 1. Simulation results on the SPEC benchmarks gives that this technique achieves 65.6% energy savings in L2 caches on average with only 0.02% area overhead and no degradation in performance. Same reduction in energy are also obtained under different L1 and L2 cache configurations , hence decreases execution time . The way tagging can be applied to today’s low-power cache design techniques to further improve energy efficiency.

The cache hit prediction methods described above suffer from a high penalty in case of cache misses because they then compare all tags in the set. The cache miss prediction methods try to overcome this limitation. Zhang *et al.* presented a method called way halting, where tag comparison at each cache way is performed in two steps. In the first step, the partial tag (e.g., four LSB bits of tag) is compared with that of the incoming address. If the comparison does not yield a match, the remainder of the tag does not need to be compared; this saves energy because the corresponding cache way does not include the incoming address[6]. If the partial tag comparison yields a match, then the remainder of the tag is compared. A Bloom filter is used to predict cache misses. The Bloom filter is a non membership function that tries to predict nonexistence . Upon cache access, the incoming address is looked up in the Bloom filter on each cache way. Bloom filter predicts the cache way miss, ie not found the tag of the corresponding way is not accessed, which saves the time of execution. If the Bloom filter cannot predict a cache way miss, the tags are compared for the corresponding way; this tag-comparison process consumes energy. a larger Bloom filter provides higher prediction accuracy, which further reduces tag access energy consumption. However, the larger size of the Bloom filter itself results in higher power and area overheads.

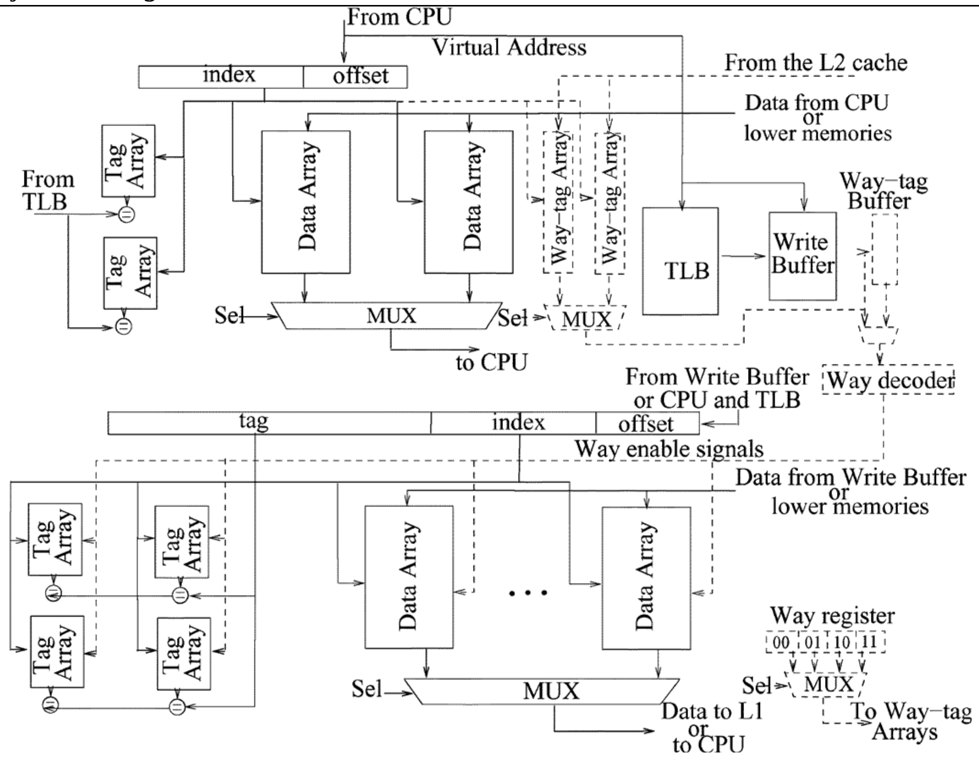

Fig.1 Way-tagged cache[1]

II. BLOOM FILTER AND PARTIAL TAG

The Bloom filter is utilized to check the approximate nonmembership of a set. When applied to reducing tag comparisons, each cache way is equipped with a Bloom filter. A query to the Bloom filter gives either of two results: negative that means definite nonexistence and positive that means likely existence. A negative result from the Bloom filter guarantees nonexistence, a cache way miss. Before the tag structure in each cache is accessed, the Bloom filter per cache way is searched . If the Bloom filter indicates nonexistence, then tag comparison for the cache way is avoided, thereby saving the energy that would have been consumed in tag comparison.
Both: 1) the smaller energy consumed to access the Bloom filter rather than the tag; and 2) the high prediction accuracy for cache way misses reduce the energy consumed in tag comparison. For instance, in the case of a cache way miss, using the Bloom filter produces a net energy gain as long as the following relationship holds.

$$E_b < p\times E_t$$

where *Eb* and *Et* represent the energy consumed while accessing the Bloom filter and the tag structure, respectively,2 and *p* is the cache miss prediction accuracy of the Bloom filter that equal to number of negative results/number of total cache way misses. Bloom filter is described as. Assume a set *p* = {*a*1, *a*2, . . . , *aN*} of *N* elements (*p* corresponds to a cache way and *N* is the number of tags in the cache way) and *k* distinct hash functions, each of which takes *ai* as the input and outputs an index of log2*M* bits as the result. The binary .Bloom filter is defined as follows.

Binary Bloom filter is an *m*-bit vector, with initially set to 0. There are two actions available: programming and query. When programming an element *ai* in set *p*, the *k* bit positions in the *m*-bit vector indexed by the *k* hash functions are set to 1. On a query on an element *aj* , if all of the bits in the *m*-bit vector indexed by the *k* hash functions have values of 1, then the query result is positive likely existence. If not, the result is negative definite nonexistence.

Increasing number of architectural techniques have relied on hardware *counting bloom filters* ie CBFs to improve upon the power, delay, and complexity of various processor structures. For example, CBFs have been used to improve performance and power in snoop-coherent multiprocessor or multi-core systems . CBFs have been also utilized to improve the scalability of load/store scheduling queues and to reduce instruction replays by assisting in early miss determination at the L1 data cache . Shown in figure 2. In these applications, CBFs help eliminate broadcasts over the interconnection network in multiprocessor systems ; CBFs also help reduce accesses to much larger and thus much slower and power-hungry content addressable memories , or cache tag arrays

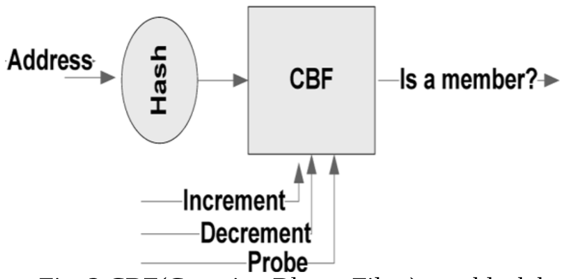

Fig. 2.CBF(Counting Bloom Filter) as a black box.[3]

Programming of a binary Bloom filter corresponds to the cache line-fill. When a new cache line starts to reside in a cache way, the Bloom filter associated with the cache way is programmed with the tag of the new cache line. Query corresponds to a cache access for reading or writing. The Bloom filter can give a positive result, i.e., likely existence, even in the case of a cache miss, which is called a *false positive*. When there is a false positive, tag comparison needs to be performed to generate a cache way miss result, which leads to wastage of energy. The probability of the occurrence of a false positive needs to be reduced to increase the energy gain enabled by the Bloom filter. In our work, we aim to reduce the probability of a false positive by equipping the Bloom filter with partial tags, which is explained later in this subsection.

Binary Bloom filter has a pollution problem because we can add new elements corresponding to line-fills into the Bloom filter, but cannot remove them ,corresponding to cache line eviction. When applying the Bloom filter for reducing tag comparisons, need a counting Bloom filter with an array of *m* counters instead of the *m* bit vector. The counting Bloom filter is programmed such that it increments the corresponding counter by one. In the counting Bloom filter, we need a new action called deprogramming, which removes an element from the counting Bloom filter. It decrements the corresponding counter by one. Thus, if a cache line enters and then exits a cache way, the corresponding counter in the counting Bloom filter is incremented ,when it enters the cache way and decremented ,when it is evicted from the cache way.

The counting Bloom filter, if all of the counters indexed by the hash functions are positive, then the query result is positive. When there is any counter with a zero value, the query result is negative. To improve the cache miss prediction accuracy of the Bloom filter, a typical solution is to increase the size of the Bloom filter, *m*. However, it also increases the overhead of the Bloom filter in terms of energy and area.

Fig. 3 shows two possible configurations used in our dynamic multistep comparison. At low or medium hot hit rates, as Fig. 3 shows, here apply a partial tag-enhanced Bloom filter ie. pBF first because the number of cache accesses filtered by the Bloom filter equal to total accesses × cache miss rate × cache miss prediction accuracy increases as the hot hit rate decreases, cache miss rate increases. This reduces tag comparisons otherwise required only to give cache misses as the results while wasting energy , at high hot hit rates, As Fig. 4(b) shows, the hot line check is performed before the partial tag-enhanced Bloom filter check. This order is adopted because the hot line check is more likely to give cache hits, which allows both subsequent Bloom filter checks and tag comparisons for cold lines to be skipped thereby reducing energy consumption.

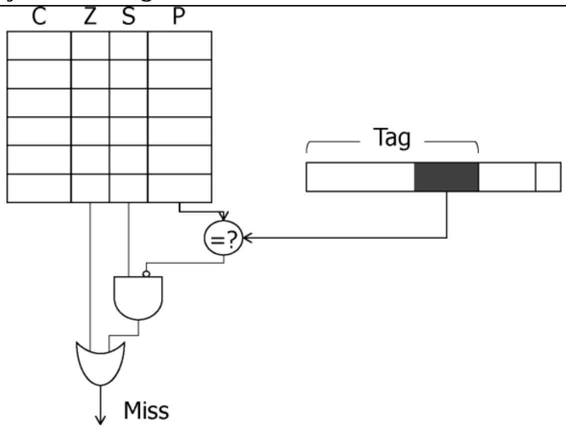

Fig. 3. Bloom filter with a partial tag.[2]

III. CONCLUSION

A partial tag with bloom filter method reduce the energy consumed in tag during data access, hence speed up the process. Here presented a partial tag-enhanced Bloom filter to improve the accuracy of cache miss prediction. To further reduce tag comparisons, Here presented a partial tag comparison that takes place during cold checks. A dynamic reordering of tag comparison steps to adapt to dynamically changing cache access behavior, and thus, further reduce tag comparisons. Finally, a method to determine the tradeoff between energy consumption and performance, which will be particularly useful for latencycritical programs. This method reduces the total cache energy consumption by 88.4% as compared to existing methods and by 10.07% when zero awareness is applied. In future, there will investigate the effectiveness of the this method in multicore environments.

IV. ACKNOWLEDGMENT

I, ARUNKUMAR A K, student of M.E APPLIED ELECTRONICS, Dept. of ECE, Dhanalakshmi Srinivasan College of Engineering, Coimbatore would like to express thanks to Asst. Prof. Mr.BOOBALAN R for his encouragement and constant co-operation throughout the completion of the paper. I deeply express my gratitude to all the ECE department staffs for their valuable advice and co-operation.

REFERENCES

[1]. Dai.J and Wang.L(2009), "Way-tagged cache: An energy efficient L2 cache architecture under write through policy," in Proc. Int. Symp. Low Power Electron. Design.
[2]. Hyunsun Park , Sungjoo Yoo , and Sunggu Lee "A Multistep Tag Comparison Method for a Low-Power L2 Cache" Ieee Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 31, No. 4, April 2012
[3]. *Elham Safi, Andreas Moshovos Andreas Veneris "L-CBF: A Low-Power, Fast Counting Bloom Filter Architecture"* IEEE Transactions On Very Large Scale Integration (Vlsi) Systems, Vol. 16, No. 6, June 2008
[4]. Min.R, Jone.W, and Hu.Y(2004), "Location cache: A low-power L2 cache system," in Proc. Int. Symp. Low Power Electron. Design, pp. 120–125.
[5]. Vijaykumar.T.N(2011), "Reactive-associative caches," in Proc. Int. Conf. Parallel Arch. Compiler Tech, p. 4961
[6]. Zhang.C, Vahid.F, and Najjar.W(2003), "A highly-configurable cache architecture for embedded systems," in Proc. Int. Symp. Comput. Arch, pp. 136–146.