Design of Low Power L2 Cache Using Multistep Tag Comparison Method

[1]Abilash.V, [2]Chidambaram.S

[1,2]Dept. of ECE, Adhiyamaan College of Engineering, Hosur, TN, India

Abstract

Tag comparison in a highly associative cache consumes a significant portion of the cache energy. Existing methods for tag comparison reduction are based on predicting either cache hits or cache misses. In this project, we propose novel ideas for both cache hit and miss predictions. In this method, a partial tag-enhanced Bloom filter is used to improve the accuracy of the cache miss prediction method and hot/cold checks that control data liveness to reduce the tag comparisons of the cache hit prediction method. This approach combine both methods so that their order of application can be dynamically adjusted to adapt to changing cache access behavior, which further reduces tag comparisons. To overcome the common limitation of multistep tag comparison methods , a new method proposed that reduces tag comparisons while meeting the given performance bound.

Keywords

Cache, Bloom filter, Tag Comparison, Multistep.

I. Introduction

Cache memory is usually part of the central processing unit, or part of a complex that includes the CPU and an adjacent chipset where memory is used to hold data and instructions that are most frequently accessed by an executing program - usually from RAM-based memory locations.CPU cache memory operates between 10 to 100 times faster than RAM, requiring only a few nanoseconds to respond to the CPU request. RAM cache, of course, is much speedier in its response time than magnetic media, which delivers I/O at rates measured in milliseconds.

A CPU cache places a small amount of memory directly on the CPU. This memory is much faster than the system RAM because it operates at the CPU's speed rather than the system bus speed. The idea behind the cache is that chip makers assume that if data has been requested once, there's a good chance it will be requested again. Placing the data on the cache makes it accessible faster. Same tag bits are used to improve error protection capability of the tag bits in the caches. When an error is detected in the tag bits, the same tag bit information is used to recover from the error in the tag bits.

II. Related Works

Hyunsun Park et al [1] discussed that the tag comparison in a highly associative cache consumes a significant portion of the cache energy. Existing methods for tag comparison reduction are based on predicting either cache hits or cache misses.In this paper, a novel idea is presented for both cache hit and miss predictions. A partial tag enhanced Bloom filter to improve the accuracy of the cache miss prediction method and hot/cold checks that control data liveliness to reduce the tag comparisons of the cache hit prediction method. Also combined both methods so that their order of application can be dynamically adjusted to adapt to changing cache access behavior, which further reduces tag comparisons.With the trend of increasing transient error rate, it is becoming important to prevent transient errors and provide a correction mechanism for hardware circuits, especially for SRAM cache memories. Caches are the largest structures in current microprocessors and, hence, are most vulnerable to the transient errors. Tag bits in cache memories are also exposed to transient errors but a few efforts have been made to reduce their vulnerability which has been proposed by Jeongkyu Hong et al [2].A new cache architecture proposed by Jianwei Dai et al [3] referred to as way tagged cache to improve the energy efficiency of write-through caches. By maintaining the way tags of L2 cache in the L1 cache during read operations, the proposed technique enables L2 cache to work in an equivalent direct-mapping manner during write hits, which account for the majority of L2 cache accesses.Soontae Kim et al [4] proposed a concept in which increasing concern about various errors, current processors adopt error protection mechanisms for their on-chip components. Especially, protecting caches in current processors incurs as much as 12.5 percent area overhead due to error-correcting codes (ECCs). Considering large L2/L3 caches employed in current high-performance processors, the area overhead is very high, consuming a large number of on-chip transistors. As an attempt to reduce that overhead, this paper proposes an area-efficient error protection architecture for large L2/L3 caches.Koustav Bhattacharya et al [5] discussed with the continuous decrease in the minimum feature size and increase in the chip density due to technology scaling, on-chip L2 caches are becoming increasingly susceptible to multi-bit soft errors. The increase in multi-bit errors could lead to higher risk of data corruption and potentially result in the crashing of application programs.

Traditionally, the L2 caches have been protected from soft errors using techniques such as:
1. Error detection/correction codes;
2. Physical interleaving of cache bit lines to convert multi-bit errors into single-bit errors;
3. Cache scrubbing.

While the first two methods incur large area overheads for multi-bit errors, identifying the time interval for scrubbing could be tricky.

III. Same Tag Information

Exploiting prevalent same tag bits to improve error protection capability of the tag bits in the caches. When data are fetched from the main memory, it is checked if adjacent cache lines have the same tag bits as those of the data fetched.This same tag bit information is stored in the caches as extra bits to be used later.

When an error is detected in the tag bits, the same tag bit information is used to recover from the error in the tag bits.Before accessing data from main memory to CPU it is stored in the cache memory for the processing waiting as required. During the storage in the cache memory the data can be misplaced in the corresponding address. This misplacement is known as Transient Error. Main challenges are:

- Lowering transient error in the cache memory.
- Reduced power consumption.
- Minimum area.
- Minimized delay.

IV. Counting Bloom Filter (CBF)

A Counting Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, thus a Bloom filter has a 100% recall rate. In other words, a query returns either "possibly in set" or "definitely not in set". Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter). The more elements that are added to the set, the larger the probability of false positives.

Bloom proposed the technique for applications where the amount of source data would require an impractically large amount of memory if "conventional" error-free hashing techniques were applied. He gave the example of a hyphenation algorithm for a dictionary of 500,000 words, out of which 90% follow simple hyphenation rules, but the remaining 10% require expensive disk accesses to retrieve specific hyphenation patterns. With sufficient core memory, an error-free hash could be used to eliminate all unnecessary disk accesses; on the other hand, with limited core memory, Bloom's technique uses a smaller hash area but still eliminates most unnecessary accesses. For example, a hash area only 15% of the size needed by an ideal error-free hash still eliminates 85% of the disk accesses, an 85–15 form of the Pareto principle (Bloom (1970)).

Bloom filter used to speed up answers in a key-value storage system.Values are stored on a disk which has slow access times. Bloom filter decisions are much faster. However some unnecessary disk accesses are made when the filter reports a positive (in order to weed out the false positives). Overall answer speed is better with the Bloom filter than without the Bloom filter. Use of a bloom filter reduces the power comparatively.The components of Bloom filter are probe,increment and decrement.

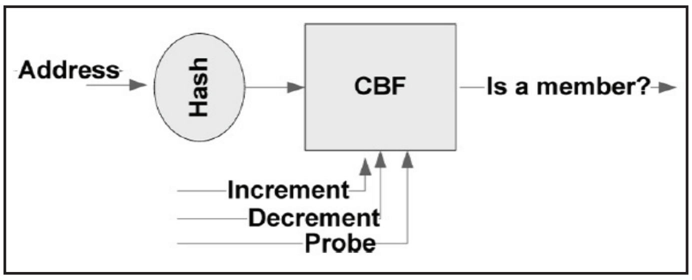

Fig. 1: Schematic Representation of CBF

The schematic representation of CBF is shown in Fig. 1. A CBF is conceptually an array of counts indexed via a hash function of the element under membership test [5]. A CBF has three operations:

1. Increment count (INC);
2. Decrement count (DEC); and
3. Test if the count is zero (PROBE).

The first two operations increment or decrement the corresponding count by one, and the third one checks if the count is zero and returns true or false (Single-bit output).

We would refer to the first two operations as updates and to the third one as a probe. A CBF is characterized by its number of entries and the width of the count per entry.

The CBF is capable of producing the desired answer to a membership test much faster and saves power on two conditions. First, accessing the CBF is significantly faster and requires much less energy than accessing the large set. Second, most membership tests are serviced by the CBF.

V. Hash Function Structure

An example of a Bloom filter, representing the set { x, y, z }. The top arrows show the positions in the bit array that each set element is mapped to. The element wise not in the set { x, y, z }, because it hashes to one bit-array position containing 0. For this structure shown in fig. 2, m = 18 and k = 3.

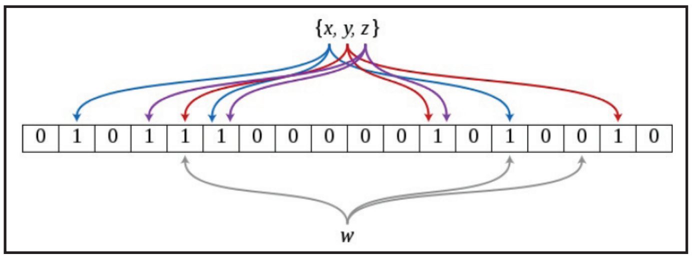

Fig. 2 : Hash Function Structure

An empty Bloom filter is a bit array of m bits, all set to 0. There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions with a uniform random distribution. To add an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.To query for an element (test whether it is in the set), feed it to each ofthe k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set – if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. In a simple Bloom filter, there is no way to distinguish between the two cases, but more advanced techniques can address this problem.Removing an element from this simple Bloom filter is impossible because false negatives are not permitted.

An element maps to k bits, and although setting any one of those k bits to zero suffices to remove the element, it also results in removing any other elements that happen to map onto that bit. Since there is no way of determining whether any other elements have been added that affect the bits for an element to be removed, clearing any of the bits would introduce the possibility for false negatives.

One-time removal of an element from a Bloom filter can be simulated by having a second Bloom filter that contains items that have been removed.However, false positives in the second filter become false negatives in the composite filter, which may be undesirable. In this approach re-adding a previously removed item is not possible, as one would have to remove it from the "removed" filter. It is often the case that all the keys are available but are expensive to enumerate (for example, requiring many disk reads). When the false positive rate gets too high, the filter can be regenerated; this should be a relatively rare event.

VI. Partial Tag Bloom Filter(pBF)

In a Counting Bloom filter, the component 'Prob' will mention as "True Positive", "False Negative" and "False Positive". In which true positive means data is present and false negative means data is not present.

"False Positive" is the multi detected address denotation, which simply means "data can be considered. The probability of the occurrence of a false positive need to be reduced to increase the energy gain enabled by the Bloom filter. In this work, it is aimed to reduce the probability of a false positive by equipping the Bloom filter with partial tags.

VII. Data Liveness Aware Tag Comparison

At time t1, a cache line is first referenced after being fetched from the main memory. During the period from t1 to t2, there are frequent accesses to the cache line. We call such a period a hot period. We also call the cache line in a hot period a hot cache line. During the period from t2 to t3, there is no access, and the cache line is evicted at time t3. During such a cold period, the cache line is called a cold cache line.

VIII. Dynamic Multistep Tag Comparison

In this proposed method, tag comparison in a highly associative cache consumes a significant portion of the cache energy. In existing methods for tag comparison reduction are based on predicting either cache hits or cache misses. A novel ideas for both cache hit and miss predictions are obtained.

This method of dynamic multistep tag comparison combines both cache hit prediction (hot and cold cache line checks) and cache miss prediction (partial tag enhanced Bloom filter) methods. A partial tag enhanced Bloom filter (pBF) is presented to improve the accuracy of the cache miss prediction method and hot/cold checks that control data liveliness to reduce the tag comparisons of the cache hit prediction method. We also combine both methods so that their order of application can be dynamically adjusted to adapt to changing cache access behavior, which further reduces tag comparisons. To overcome the common limitation of multistep tag comparison methods, it is proposed by a method that reduces tag comparisons while meeting the given performance bound. It is dynamic because it dynamically adjusts the order of tag comparison steps to maximize the efficiency of each of the cache hit and miss prediction methods.

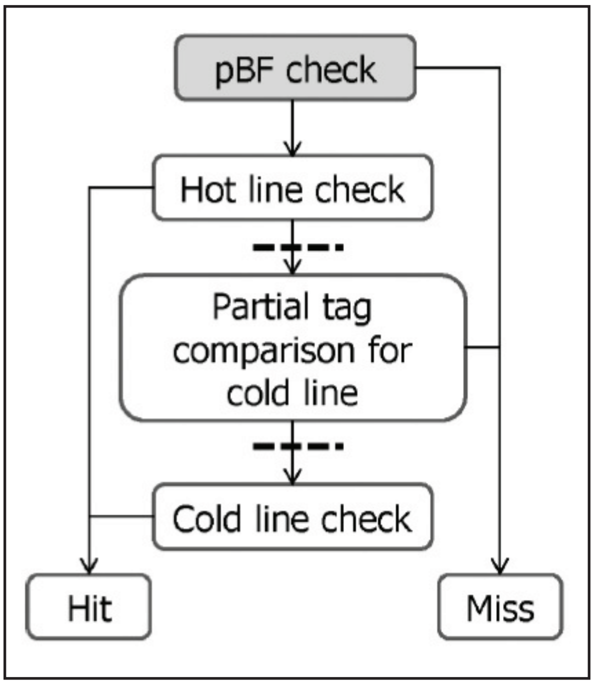

Fig. 3 : Configuration of Multistep Comparison at Low or Medium Hot Cache Hit

At low or medium hot hit rates, as Fig. 3, shows when applying a partial tag-enhanced Bloom Filter (pBF) first because the number of cache accesses filtered by the Bloom filter (= #total accesses × cache miss rate × cache miss prediction accuracy) increases as the hot hit rate decreases (i.e., cache miss rate increases). This reduces tag comparisons otherwise required only to give cache misses as the results while wasting energy.

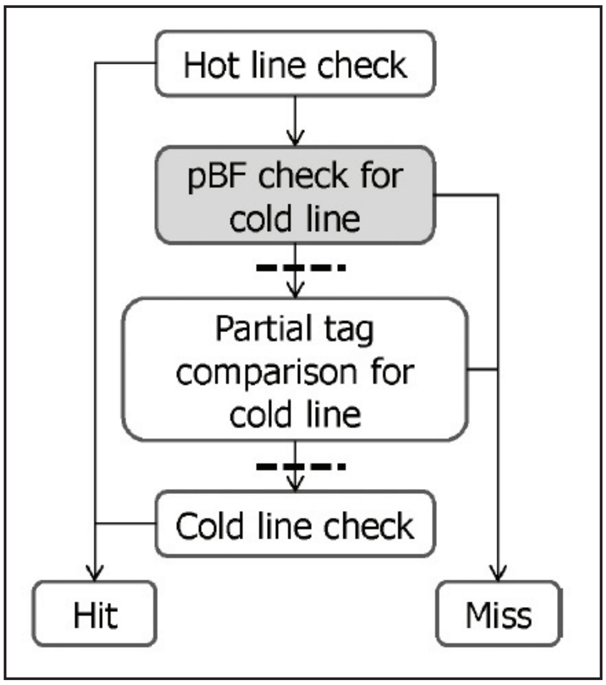

Fig. 4 : Configuration of Multistep Comparison at High Hot Cache Hit

At high hot hit rates, as Fig. 4 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.

IX. Results and Discussion

The simulation result of L2 cache with counting Bloom filter method and dynamic multistep tag comparison method is obtained with XILINX ISE 14.1 using VHDL language. The area, delay and power consumed have been reported in Table 1.

Table 1: Simulation Results

Simulation Parameters	Counting Bloom Filter Method	Dynamic Multistep Tag Comparison Method
Area	71299 (Gate Count)	35656 (Gate Count)
Power Consumed	675 mW	194 mW
Delay	8.732 ns	4.073 ns

X. Conclusion

We summarized the multistep tag comparison method to reduce the energy consumed in tag comparison within highly associative L2 caches. We presented a partial tag-enhanced Bloom filter to improve the accuracy of cache miss prediction. We also explained hot/cold checks (with dynamic timeout tracking) as a cache hit prediction method. To further reduce tag comparisons, we presented a partial tag comparison that takes place during cold checks. When data are fetched from the main memory, it is checked if adjacent cache lines have the same tag bits as those of the data fetched. The modification done by using Counting Bloom Filter (CBF) scheme, which makes output only with comparatively reduced power.The estimated area and delay is not much reduced.The simulation result shows that there is a significant improvement in terms of area , power and time delay of dynamic multistep tag comparison method.

References

[1] Hyunsun Park, Sungjoo Yoo, 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 2015.

[2] Jeongkyu Hong, Jesung Kim, Soontae Kim,"Exploiting Same Tag Bits to Improve the Reliability of the Cache Memories," IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 23, No. 2, February 2015.

[3] Jianwei Dai, Lei Wang,"An Energy-Efficient L2 Cache Architecture Using Way Tag Information Under Write-Through Policy," IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 21, No. 1, January 2013.

[4] Soontae Kim,"Reducing Area Overhead for Error-Protecting Large L2/L3 Caches," IEEE Transactions On Computers, Vol. 58, No. 3, March 2009.

[5] Koustav Bhattacharya, Nagarajan Ranganathan, Soontae Kim,"A Framework for Correction of Multi-Bit Soft Errors in L2 Caches Based on Redundancy," IEEE Transactions On Very Large Scale Integration (VLSI) Systems, Vol. 17, No. 2, February 2009.

[6] Zhigang Hu, Stefanos Kaxiras, Margaret Martonosi, "Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior," IEEE Transactions on Dependable and Secure Computing, Vol. 3, No. 3, July-September 2006.

[7] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, Norman P. Jouppi,"McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," IEEE Transactions On Computers, Vol. 54, No. 12, December 2005.

[8] Wei Zhang,"Replication Cache: A Small Fully Associative Cache to Improve Data Cache Reliability," IEEE Transactions On Computers, Vol. 54, No. 12, November 2005.

[9] Patrick J. Meaney, Scott B. S., Pia N. Sanda, Lisa Spainhower,"IBM z990 Soft Error Detection and Recovery" IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, September 2005.

[10] Charles W. Slayman,"Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations", IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, September 2005.

V.Abilash is pursuing M.E in VLSI Design in the Department of Electronics and Communication Engineering at Adhiyamaan College of Engineering, India. He received his B.E degree in Electronics and Communication Engineering from Anna University, India in 2015. His research interests are in the domain of VLSI signal processing, Image processing and Embedded Systems.

S.Chidambaram is working as Associate Professor in the Department of Electronics and Communication Engineering at Adhiyamaan College of Engineering, India, where he leads the Image and Signal Processing Lab. He received his B.E degree in Electronics and Communication Engineering from Bharathidasan University, India in 2002, M.E degree in Power Electronics and Drives from Anna University, India in 2005 and currently he is working toward his Ph.D in the domain of hyperspectral Image Processing. His research interests lie in the field of Digital Signal Processing, Hyperspectral Image processing, Satellite Image Processing such as enhancement, classification and compression algorithms, Statistical Signal Processing, Sparse Representations. He is a member of IEEE and ISTE.