Technique to Mitigate Soft Errors in Caches with CAM-based Tags

Luong D. HUNG†, Masahiro GOSHIMA†, and Shuichi SAKAI†

† Department of Information and Communication Engineering, the University of Tokyo  Hongo 7–3–1, Bunkyo-ku, Tokyo, 113–8656 Japan

Abstract  Content Addressable Memories (CAM) are widely used for the tag portions in highly associative caches. Since data are not explicitly read out of tag array in CAM search, the detection of false misses caused by soft errors for such caches, is difficult. This paper presents a novel technique to detect the false miss in highly associative cache with CAM-based tag. The technique involves subdividing the tags and providing backup checking for cases the tags are partially matched. An original tag encoding scheme is also proposed to reduce the frequency of back-up checking. Modifications to support the technique do not increase the cache access latency. The performance degradation incurred by additional cycles for false miss checking is very low.

Key words  Soft Error, Content Addressable Memory, Cache

1. Introduction

In typical implementation of a cache, the data portion is implemented using RAM (Random Access Memory), while the tag portion can be implemented using either RAM or CAM (Content Addressable Memory). In this paper, we refer to a cache with RAM-based tag as RAM-RAM cache, and a cache with CAM-based tag as CAM-RAM cache.

The CAM-RAM structure allows realization of highly associative caches. Highly associative CAM-RAM caches offer low cache miss rate, fine granularity of cache lock down [6]. Cache lock-down reserves a portion of cache for exclusive use of critical interrupts. Intel XScale processor [1] and other ARM processors [6] use highly associative (typically 32-way) CAM-RAM caches for their instruction caches and data caches. CAM-RAM caches are also claimed to be more power-efficient than RAM-RAM caches [7] [11] [12].

Soft error [2] in a tag raises the possibilities of false hit, and false miss. A false hit (false miss) refers to a cache hit (miss) that is actually a miss (hit) in the absence of the soft error. The false hit leads the processor to use the data of the wrong location. False miss is a data integrity problem if the corrupted tag refers to dirty data. False miss in this case may trigger fetch and use of stale data from the lower level cache.

In the case of RAM-RAM cache, the false hit and false miss can be detected with conventional coding techniques [3] [9]. These techniques store check bit(s) for each tag entry. Since tags are always read from the RAM tag array and compared with the input for hit/miss judgment, recomputing the check bits from these tags and comparing with previously stored check bits can detect the soft error.

These same techniques can be used to detect the false hit in CAM-RAM cache. Here, the tag check bits can be stored in the data portion of the cache and recomputation of the check bits are based on the input. However, the false miss in CAM-RAM cache cannot be detected with these conventional coding techniques. Since the bit comparison function is built into the CAM cells, CAM search is accomplished without explicitly reading the data out of tag array. The unavailability of tag data in CAM access makes detection of the false miss challenging.

False miss in CAM tags can be prevented by using write-through caches, or be detected through tag replication, or be reduced through cache scrubbing [8]. However, there are limitations with those approaches. Write-through caches increase the traffic to lower level caches and may impact performance. Replicating the tag portion and comparing the tag search results incur large overheads. False miss is still possible with cache scrubbing since the data may be corrupted and accessed between scrubbing intervals, especially for frequently accessed level-1 (L1) caches.

This paper presents a novel technique to mitigate false miss in highly associative CAM-RAM caches. The technique involves subdividing the CAM and providing backup check for cases the tags are partially matched. An original tag encoding scheme is also proposed to reduce the frequency of back-up checking. Modifications to support the technique do not increase the cache access latency. The performance degradation incurred by additional cycles for false miss checking is very low.

Section 2. discusses the highly associative CAM-RAM
caches. Section 3. discusses the data integrity problems caused by soft errors in CAM tag. Section 4. explains the structure and the access algorithm of our modified CAM-RAM cache. Section 5. describes the original tag encoding scheme. Section 6. evaluates the overheads of the technique. Finally, Section 7. concludes the paper.

2. Highly Associative CAM-RAM Cache

Fig. 1 shows the typical structure of an highly associative CAM-RAM cache. The cache consists of several CAM-RAM macros. Each CAM-RAM macro corresponds to a cache set. Each row of CAM-RAM macro holds the tag and data of a cache line.

For each cache access, some low order bits of the input address are used to select a CAM-RAM macro. The high order bits are used as the input tag. The match lines in the CAM are initially precharged high. The input tag is fed into the bit lines of the CAM. Any mismatch between the bits inputted to the bit lines and the bits stored in CAM cells in a row discharges the match line for that row. The match line remaining high after the search indicates a hit. The hit match line then drives the corresponding word line of RAM, from which the relevant data word is selected. Since the bit comparison function is built into the CAM cells, CAM search is accomplished without explicitly reading the CAM row data from tag array—only hit/miss information indicated by the match lines is available after the search.

There are several advantages with such highly associative CAM-RAM caches. First, highly associative cache offers high hit rate and improves processor performance. Second, fine granularity of cache lock-down is possible [6]. In cache lock-down, a portion of cache is reserved for exclusive use of critical interrupts. With a 4-way set-associative cache, for example, it is practical to lock down a quarter of cache (1-way), half of the cache (2-way), or three-quarters of the cache (3-way). This is a coarse granularity and is inefficient if, for example, all that need be lock down is memory to hold data for a small interrupt handler. The CAM-RAM caches typically offer 32-way associativity, enabling the cache to be locked-down in units of 1/32 of the cache, which is a much finer granularity. Third, since only one CAM-RAM macro is activated for each access and other macros are clock gated, CAM-RAM cache can offer power-efficient [4][7][11]. These advantages make CAM-RAM cache a valuable design choice.

3. Data integrity problems caused by soft errors in CAM Tag

Soft error in a CAM tag raises the possibilities of false hit and false miss. Fig. 2 illustrates examples of false hit and false miss. A false hit refers to a hit that is actually a miss in the absence of the soft error. The false hit in CAM may lead the processor to use the incorrectly matched data. However, the false hit can be detected by storing the check bits for each row of the CAM in the RAM. When a access for a given input results in a hit, the corresponding check bits of the hit CAM row are read out from RAM and compared with those check bits recomputed from the input tag. Mismatch in the check bits signifies that the data has been corrupted and the hit is indeed a false hit [3][9].

A false miss refers to a miss that is actually a hit in the absence of the soft error. False miss may cause a data integrity problem in write-back data cache. Dirty data may be hold in the write-back cache. A false miss on dirty data triggers fetch and use of stale data from lower level cache.

Soft error may also result in a multi-hit—a search hits in multiple entries. The same method used to detect false hit can be applied to identify among the hit entries which one is true hit and which one is false hit. We treat multi-hit as a special case of false hit in this paper.

The false miss problem in CAM tag can be prevented by using write-through cache, or be detected through tag replication, or be reduced through cache scrubbing [8]. However, write-through cache increases the traffic to lower level cache. Replicating the tag portion and comparing the search results increase the area and power consumption (cf. a CAM cell
occupies an area approximately four times that of a RAM cell [7]). Cache scrubbing involves reading periodically the contents of the cache and correcting all the correctable errors. Data integrity violation is still possible in cache scrubbing if the data are corrupted and accessed in the scrubbing interval, especially for frequently accessed L1 caches [8]. To our knowledge, there has been no cost-effective technique to deal with the false miss problem in highly associative CAM-RAM cache so far. Our technique provides such one.

4. Modified CAM-RAM Caches

4.1 Cache Structure

Our technique divides the CAM in each cache macro into two sub-CAMs, as shown in Fig. 3. A tag is divided into two sub-tags and each sub-tag is stored in a sub-CAM. The pair of match lines in the sub-CAMs (local match lines) are ANDed to provide the global match line. The OR logic of the local match lines, called close hit line, is also provided. When a tag is stored in the CAM, the check bits computed from the tag value are stored together with the corresponding line data in the RAM.

4.2 Cache Access Algorithm

Given an input address, the search process first divides the tag portion of the address into sub-tags and feeds each sub-tag into a sub-CAM of the selected macro. The sub-CAMs then perform associative search in parallel. A global match line staying asserted after the search indicates a cache hit. In this case, the check bits of the hit tag are also read out from RAM with the data word. Check is performed to verify whether the hit is really a true hit, or a false hit.

We focus on the case of cache miss. There is no global match line asserted in this case. If there is any close hit line asserted, then the case is called close hit. Since soft error is a rare event and affects only a few bits located closely, the separation of the sub-CAMs makes the probability that both halves of a tag stored inside the sub-CAMs be corrupted, very low. A false miss, therefore, will show up as a close hit. When a close hit is encountered, the tag with close hit line asserted is explicitly read from the CAM. The check bits are computed from the present value of the tag and compared with the check bits previously stored in RAM to verify whether the tag has been corrupted or not. The close hit is a false miss if the tag is found out to be corrupted. Otherwise, it is considered a true miss.

It is possible that a tag search results in hits or close hits of multiple CAM entries. In this case, the checking routine is repeated for each of those entries.

While algorithm described so far concerns only the detection of false hit and false miss (e.g., using parity bits as the check bits), the correction of the corrupted tags, if required, can also be supported by using appropriate error correcting code as the check bits (e.g., Hamming code).

5. Tag Encoding for Close Hit Reduction

In the common case of error-free cache access, a close hit is verified to be a true miss. Since verifying if the close hit is a false miss or true miss requires the tag to be explicitly read from the sub-CAMs, overhead in term of access latency and power consumption incurs. This section first discusses how often the close hits happen. An encoding scheme for reducing the frequency of close hits is then proposed.

5.1 Close Hit Rate

If we assume that the target addresses of cache accesses
hit rate, $R_{ch}$, can be roughly estimated by Equation 1.

$$R_{ch} = R_m \times \frac{1}{2^{T/2}} \times W \times 2$$  \hspace{1cm} (1)$$

Here, $R_m$ is the cache miss rate, $W$ is the cache associativity, and $T$ is the length of tag.

Close hit case is considered only when the cache is missed (no global match line asserted), hence the multiplication of $R_m$ in the right hand side (RHS) of the equation. The second term in RHS is the probability that a two randomly chosen sub-tags are coincident. A input sub-tag needs to be compared with $W$ sub-tags on the same set, hence the multiplication of $W$. Finally, close hit can be triggered by matching of either of the two input sub-tags, hence the last term in the RHS.

Let us consider a concrete example: the 32KB, 32-way, 32B line cache similar to the data cache found in Intel XScale processor [1]. With 32-bit address, five bits are used to index bytes inside a line, another five bits are used for set indexing (32 sets), leaving the remaining 22 bits used for tag. Substituting these values ($W=32$, $T=22$) into Equation 1, we have $R_{ch} = R_m / 32$. That is, a close hit is estimated to occur for 32 cache misses, on average. With such low level of close hit rate, the overhead of checking for false miss can be tolerable.

Fig. 4 shows the miss rate, close hit rate of SPEC2000 benchmarks for the 32KB data cache discussed above. The data cache is virtually-indexed, virtually-tagged. The cache simulation is carried out using the cache simulator in SimpleScalar toolset [5]. The ARM binaries are generated using GCC compiler (version 2.95). Each benchmark is run for 20 billion instructions. A tag is divided based on high, and low order bits.

Contradict to our initial estimation ($R_{ch} = R_m / 32$), the close hit rate is prohibitively high, particularly for ammp, art, gcc, and mcf. The close hit rates many time higher than the miss rates are possible since multiple close hits may happen in a single miss.

5.2 Distribution of Access Addresses

The cause of high close hit rate can be understood by considering the distribution of the target addresses of the accesses to the data cache. In Fig. 5, the distributions of target addresses over the 32-bit address space are plotted. The 4GB address space is divided into 32MB regions, as shown on the Y axis. The darkness of a region represents the access rate to the region.

Accesses cluster in a very few regions in the address space for all benchmarks. The cluster of accesses results in low diversity of the high order bits of the target addresses. If we simply divide the tags to sub-tags containing respectively the high, and low order bits, many close hits will results from the coincidences of the sub-tags containing the high order bits.

5.3 Tag Encoding Scheme

We propose an original tag encoding scheme, shown in Fig. 6. A tag is divided into two parts which contain respectively the high and low order bits. The part containing the low order bits is used as first sub-tag. The second sub-tag is produced by XORing the two parts. Since the diversity of the high order bits of the tags is low, by XORing the two parts together, we essentially impose the diversities of the low order bits to the high order bits. The frequency of close hits therefore is expected to be reduced.

Our encoding scheme does not lose the tag data information. The high order bits of a tag can be reproduced by XORing the two sub-tags stored in the sub-CAMs.

Fig. 8 shows the close hit rates of the data cache using proposed tag encoding scheme. The cache configuration and simulation method are identical to those described in Section 5.1. Compared to the results shown in Fig. 4, the close hit rates are reduced greatly for all benchmarks. The proposed tag encoding scheme is therefore very effective.
Noteworthy, the detection (or correction) of soft errors should be based on the encoded tags. Specifically, the check bits should be directly computed from TAG1 and TAG2 in the Fig. 6. If the check bits are computed from the original tags otherwise, a bit corruption in TAG1 would generate another bit corruption when restoring the high order bits (by XORing TAG1 and TAG2). The doubling of bit corruption may exceed the error detection/correction ability provided by the check bits.

### 6. Overhead

This section discusses how circuit modifications impact the cache access latency. The impact on the processor performance is also considered.

#### 6.1 Cache Access Latency

The access time of the 32KB, 32-way, 32B line CAM-RAM cache is evaluated using Cacti tool [10]. Since the original Cacti is only able to model a single fully-associative CAM cache, we modified it to model the cache consisting of multiple CAM-RAM macros. Taking into account the time to route and select a macro.

The access times of the cache, before and after modification, are indicated respectively in Equation 2 and Equation 3. Here, we assume 0.18 um process technology. The access time is the sum of 1) the time to route and select a CAM-RAM macro (\( macro\_select \)), 2) the time to search the CAM (\( cam\_search \)), and 3) the time to read data from RAM (\( ram\_read \)).

\[
\text{access\_time} = \text{macro\_select}(0.21\text{ns}) \\
+ \text{cam\_search}(0.66\text{ns}) \\
+ \text{ram\_read}(0.51\text{ns}) \\
= 1.38\text{ns} \quad (2)
\]

\[
\text{access\_time} = \text{macro\_select}(0.21\text{ns}) \\
+ \text{cam\_search}(0.29\text{ns}) \\
+ \text{ram\_read}(0.51\text{ns}) \\
= 1.01\text{ns} \quad (3)
\]

The CAM search time dominates access time in the original cache. Subdivision of the CAM into two sub-CAMs in the modified cache reduces the lengths of the match lines that are needed to be driven, and results in considerable improvement in CAM search time (from 0.66ns to 0.29ns).

We also evaluated that the time required for tag encoding is 0.067ns. Since tag encoding can perform in parallel with macro selecting, the overhead of tag encoding is completely hidden.

Modifications to the cache to support our technique therefore do not only increase but, on the other hand, help decrease the cache access time.

#### 6.2 Processor Performance

Our modified cache requires checking in close hit case to verify whether the close hit is a true miss or a false miss. Such checking needs additional execution cycles and degrades processor performance. We used execution-driven, cycle-accurate simulator from SimpleScalar toolset [5] to measure the performance degradation of processor using our modified data cache. We modeled a processor similar to Intel XScale processor [1]. The processor is a 5-stage in-order scalar processor, having 32KB, 32B line, 32-way set associative instruction cache, and data cache. Cache hit latency is 1 clock cycle. Memory is idealized with first chunk and inter chunk access latencies are respectively 1 and 32 cycles. All benchmarks are run for 10 billion instructions. We assumed that checking for each close hit costs additional one-cycle penalty.

Fig. 8 shows the evaluation results. Execution time for all benchmarks increases no more than 0.005%. Such very small performance degradation results from the great reduction of close hit rate by our tag encoding scheme.

### 7. Conclusion

The circuit structure and the access nature of Content Addressable Memory (CAM) make detection of false miss problem caused by soft error, difficult. This paper proposes a cost-effective technique to mitigate the false miss in highly associative CAM-RAM cache.

The technique involves subdividing the tags and providing...
backup checking for cases the tags are partially matched. An
original tag encoding scheme is also proposed to effectively
reduce the frequency of back-up checking. Evaluation results
indicate that the circuit modifications to support the tech-
nique do not increase cache access latency. The performance
degradation is extremely low (below 0.005%).

Acknowledgement

This research is partially supported by Grant-in-Aid for
Fundamental Scientific Research B(2) #13480077, B(2)
#16300013 from Ministry of Education, Culture, Sports, Sci-
cence and Technology Japan, and by Semiconductor Tech-
nology Academic Research Center (STARC) Japan, CREST
project of Japan Science and Technology Corporation, and
by 21st century COE project of Japan Society for the Pro-
motion of Science.

文献

inelscale.
Devices–Part I: Three Radiation Sources. IEEE Transac-
S. J. Straudus, and L. T. Clark. A Low-Power 2.5-GHz 90-
nm Level 1 Cache and Memory Management Unit. Journal
Cache Scrubbing in Microprocessors: Myth or Necessity. In
IEEE Pacific Rim International Symposium on Dependable
and P. Helland. Fault-tolerant Features in the HaL Mem-
ory Management Unit. IEEE Transactions on Computers,
[10] P. Shivakumar and N. P. Jouppi. CACTI 3.0: An Integrated
Cache Timing, Power, and Area Model. Technical Report
Associative Cache Design for Embedded Processors. In
IEEE International Conference on Computer Design, pages
Low-Power Processors. In Kool Chips Workshop, 33rd In-