Fast Correction of Multiple Soft Errors in Highly Associative Cache with CAM-Based Tag

Hyuk-Jun Lee[1], Seung-Cheol Kim[1],, and Eui-Young Chung[2]

[1] School of Computer Science and Engineering, Sogang University
#1, Sinsu-dong, Mapo-gu, Seoul 121-742, Republic of Korea
{hyukjunl,vermouth}@sogang.ac.kr
http://ecl.sogang.ac.kr
[2] School of Electrical and Electronic Engineering, Yonsei University
#134 Sinchon-dong, Seodaemun-gu, Seoul 120-749, Republic of Korea
eychung@yonsei.ac.kr
http://dtl.yonsei.ac.kr

Abstract. Content addressable memory (CAM) is a key component to build the tag memory of a highly associative cache. As CMOS process technology scales,soft error rates (SER) in CAM cells increase significantly. Bit flipping in CAM cell leads to a false miss upon a cache access, which could be fatal from a system point of view. Previous schemes either focused on reducing the probability of soft errors or had an unbounded single soft error correction time. Compared with previous schemes, our approach completely detects and corrects multiple soft errors. The detection and correction in our scheme happens as a background process, does not interfere with concurrent cache accesses, and does not affect the performance of time-critical cache operations. In addition, we enhance the cache miss holding register structure of a non-blocking cache to avoid data corruption due to any false cache miss happening between occurrence and correction of errors.

Keywords: CAM, cache, soft error, error correction.

1 Introduction

Cache memory is used in microprocessors to fill the performance gap between high speed processor cores and slow external memories such as DRAM. Cache memory consists of a tag memory and data memory. A tag memory can be implemented with CAM or SRAM. Cache can be classified as CAM-RAM cache or RAM-RAM cache according to the implementation method [2]. CAM-RAM cache is adequate for the highly associative cache which provides a low miss rate. Achieving a low miss rate is critical in modern microprocessors. As CMOS process technology scales and low power schemes such as voltage scaling are used, the probability of soft errors increases in densely integrated memory cells. Soft errors in the tag portion of the cache turn into critical system problems without error detection and correction. For instance, bit flipping in the tag memory can

T.-h. Kim et al. (Eds.): SecTech/CA/CES[3] 2012, CCIS 339, pp. 424–431, 2012.

lead to *a false miss* in which case accessing cache block exists in the cache but a cache miss happens due to the bit flip in the tag. In CAM-RAM cache, a tag memory made out of CAM cannot be read upon a cache access. When a false miss happens, data in the missed cache block could be dirty, i.e., modified. In case of the write-back data cache which delays updating a lower level memory until replacement, a false miss fetches stale data from the lower level memory and causes data corruption.

Several architectural methods were proposed to reduce false misses [1][2]. Cache scrubbing discussed in [1] was not originally targeted for the CAM-based tag but for the SRAM-based tag. In addition, cache scrubbing does not completely prevent false misses and undetected false misses cause a fatal system failure. In [2], Hung and et al. proposed a scheme that prevents false misses by segmenting tags horizontally into two sub-tags and identifying the sub-tags that could be potentially hit without soft errors as local hits. They attempted to reduce the number of local hits by optimizing the mapping. However, the drawback of this scheme is that the correction time is unbounded. In summary, previous works incur performance penalty in normal cache operations and are not able to handle multiple soft errors. The work in [3] had proposed a cell architecture that can detect only a single bit soft error in stand-alone CAM, which has several issues if we apply the proposed technique directly to a cache memory. The first issue is that, as semiconductor technology scales, multi-bit soft errors can occur in vertically and horizontally neighboring cells. The second issue is that detection itself cannot avoid a system failure because a false miss can still occur before the correction happens. Unless the fetch initiated by a false miss is handled correctly, it can cause data corruption. For this reason, correction must be completed before the cache miss handling is completed.

To resolve these issues, we propose a circuit and architecture technique for the fast detection and correction of multiple soft errors in the CAM-RAM cache. This technique performs a background correction process which does not interfere with on-going cache operation. We enhance the miss status holding register structure of a non-blocking cache to handle the missed block fetched by a false miss. In addition, we optimize the detection and correction circuitry to make overall correction time less than the cache miss penalty, which is the requirement for correct operations. Our scheme works for both blocking and non-blocking cache. For simplicity, we present works only for a non-blocking cache.

2 Proposed Architecture

2.1 Overall Architecture

The main idea of the proposed scheme is that error detection and correction is performed as a background process immediately after an error occurs. This process does not interfere with a concurrent cache access unless the concurrent access attempts to access a corrupted cache entry. To handle the case where a corrupted cache entry is accessed before the error is fixed by the background process, we modify the miss status holding register (MSHR) logic [4].

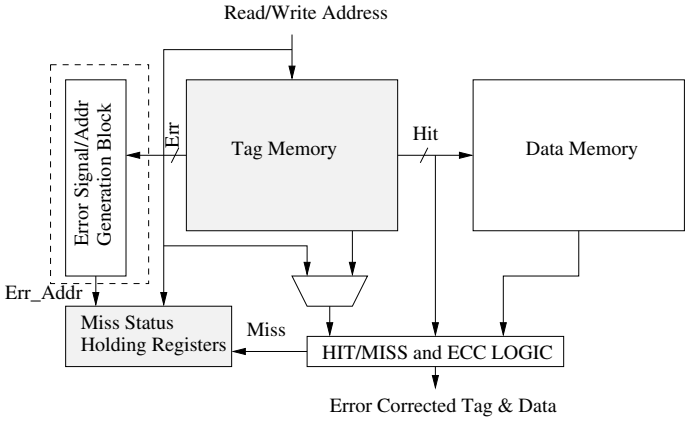


Fig. 1. Proposed Cache Architecture

As shown in Fig. 1, our architecture consists of two parts: conventional cache and newly added blocks. The block inside the dotted line is a new addition and blocks in gray color are modified ones. The conventional cache consists of a tag and data memory and hit/miss and ECC circuits, and MSHR. In our proposed architecture, the conventional cache part has been modified. First, the tag memory is modified to generate an error signal per tag word which is labeled as *Err* in Fig. 1. The width of *Err* is same as the number of tag words in the tag memory. When the cell of a tag word is corrupted, the error signal, *Err*, is asserted to the error signal/address generation block. Based on this *Err* signal, the signal/address generation block asserts a global error signal and generates the address of a corrupted tag word to the MSHR block. If the address of a corrupted tag is found in MSHR that stores the addresses of any on-going cache misses, it indicates that a false miss happened earlier. The MSHR block is enhanced in order to cancel a miss block fetch in case of a false miss.

2.2 Detecting and Correcting Multiple Soft Errors in a Single Interleaved Tag Word

To detect and correct multiple soft errors in a single tag word, a tag memory has been modified. The new tag architecture is shown in Fig. 2(a). In this architecture, two tags are interleaved to form an interleaved tag. C_{nm} stands for the m_{th} cell of n_{th} tag. Shaded cells belong to the tag word 0 whereas non-shaded cells belong to the tag word 1. P_0 and P_1 are parity bits for tag word 0 and 1 respectively. L_{nm} is a parity computing logic for C_{nm}. The parity computing logics for a tag word are connected to form a chain and drives an error detection NAND gate. The parity logic is implemented with small NMOS transistors to reduce an area penalty. To compensate for the slow speed of NMOS pass gate logic, a skewed NAND gate is used to quickly detect an error transition, e.g. the input of an NAND gate changing from 1 to 0. A key challenge in design is

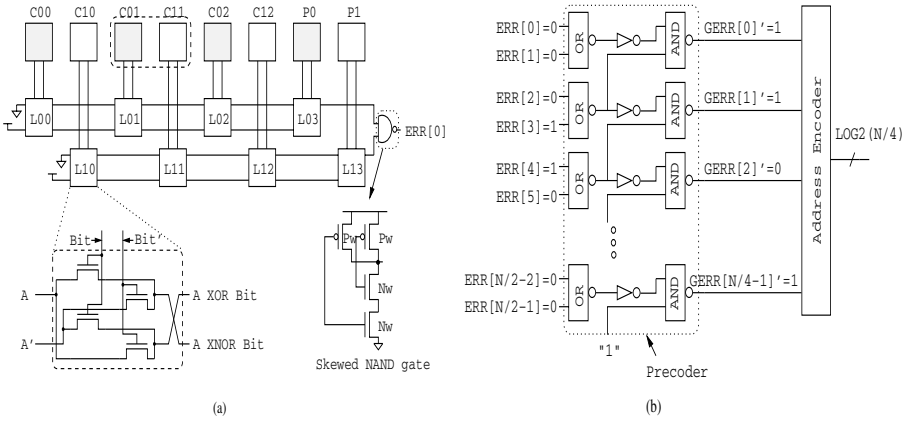


Fig. 2. (a) Cell Architecture for multi-bit correction (MBC): Interleaving two tag words to detect and correct double bit errors. (b) Error address encoder which consisting of a pre-coder and an address encoder.

to optimize both speed and area by sizing these transistors while maintaining good noise margin. Thus extensive simulations are done and the results will be discussed in the simulation section.

Since one interleaved tag represents two different tags, soft errors in two contiguous cells of the interleaved tag, e.g. C_{01} and C_{11}, turn out to be two single-bit soft errors for two different tags. Two parity bits for an interleaved tag should be sufficient to detect two contiguous soft errors where each parity bit covers one of two tags respectively. This scheme can be readily extended to detect N soft errors by interleaving N tag words. A key point is that the number of parity bits required per tag is only one for detecting N soft errors. Another important benefit can be found in the complexity of ECC. In this scheme, we only need to have an ECC that can correct a single bit error per tag. A potential drawback of this scheme is to make the width of interleaved tag two times larger, which may increase an interconnect length. However, this does not become an issue because CAM itself has been partitioned horizontally and vertically to improve its timing.

2.3 Detecting and Correcting Multiple Soft Errors in Adjacent Tag Words

To detect and correct multiple errors in vertically adjacent tag words, the address of erred words should be determined. This is done by the error address generation block, shown in Fig. 1. This block contains an error address encoder.

Error Address Encoder. When an alpha particle hit corrupts multiple adjacent tag words, a priority encoder should be used to find the address location of the errors. Implementation of a priority encoder is costly compared to an address encoder. To take advantage of the fact that only adjacent tag words can

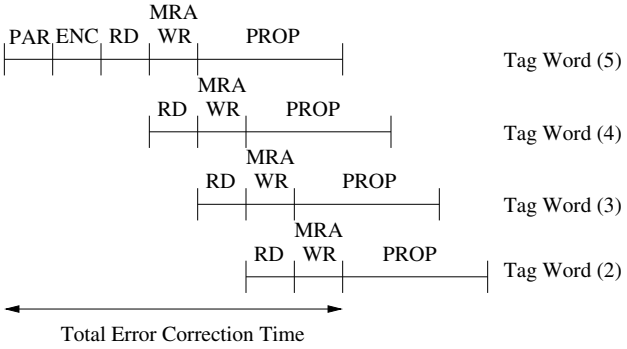


Fig. 3. Pipeline diagram for the detection and correction for soft errors in Fig. 2(b)

Table 1. Definition for abbreviated words in Fig 3

Pipeline Stage	Delay Component
PAR	Parity chain
ENC	Error address encoder
RD	Reading tag CAM and error computation
WR	Writing tag CAM
MRA	Miss Status Holding Register access
PROP	Parity chain and error address encoder

be corrupted simultaneously, we propose a scheme that only requires an address encoder.

In our scheme, the error address encoder block consists of a pre-coder and an address encoder as shown in Fig 2(b). The pre-coder, shown inside the dotted box of Fig. 2(b), has two functions. The first is to OR several adjacent error signals using NOR gates. This is done to reduce the complexity of an address encoder. In Fig. 2(b), error signals from two adjacent tag words are ORed together by the leftmost NOR gates. So we only need an $\frac{N}{2}$-input address encoder instead of an N-input address encoder. The side-effect of this, if only one of two adjacent tag words is corrupted, we need to access both to determine which one is corrupted. The second function of a pre-coder is to give a tag with a higher address more priority so that it can be detected first. This second function is achieved using 2-input NAND gates. This pre-coding circuit makes only one word corruption appear at a time and thus an address encoder is sufficient to detect the address of a corrupted word. In the example shown in Fig. 2(b), $ERR[3]$ and $ERR[4]$ are one, which means that both tag word 3 and 4 are corrupted. Due to the gating logic, however, only $G\_ERR[2]'$ is de-asserted, which indicates that the tag group containing tag word 4 and 5 is a starting point of corruption. Thus, the correction process reads tags in the order of tag word 5, 4, 3, 2 until it does not detect a corrupted word any more.

Detection and Correction Process. When an alpha particle affects M adjacent words, we fix tag words one by one from the higher to the lower address word in a pipelined fashion as you can see from Fig. 3. The definition for PAR, ENC, RD, WR+MRA, and PROP used in Fig. 3 are described in Table 1. Since we do not know, how many tag words are affected by an alpha particle, we read them one by one starting from the lower address tag word until we find a tag with no error. By pipelining correction process and optimizing circuits for fast detection, we can significantly reduce the latency for correcting errors in multiple adjacent tags.

2.4 False Cache Miss Handling

Typical handling of a cache miss in state-of-the-art non-blocking write-back cache proceeds as follows. First, a cache miss is registered into MSHR if no previous miss on the same cache line is found in MSHR. Second, a block to be replaced is picked and its contents are written back if the line is dirty. Third, a missed block fetch is initiated. Finally, when the block is fetched from the lower level memory, it is written into the cache and its MSHR entry is invalidated.

This procedure is modified to handle a false miss. The major modification includes following steps. First, when a soft error is detected, its corresponding tag is read and corrected. This corrected tag is searched in the MSHR. If it is found, it indicates that a false miss happened early and thus invalidates the valid bit of the corresponding entry in the MSHR. When the missed block is fetched from the lower level memory and it is not found in MSHR (which means its valid bit is invalidated), the fetched block is abandoned. To make this procedure work, we need an upper bound on the correction time, which is shown in Fig. 3. That is, if the correction is not completed before the false miss fetches a block, an entry in MSHR is not invalidated and the fetched block is treated as a fetch for a true miss. This can corrupt the cache block and lead to a system failure. Thus, the equation (1) should hold.

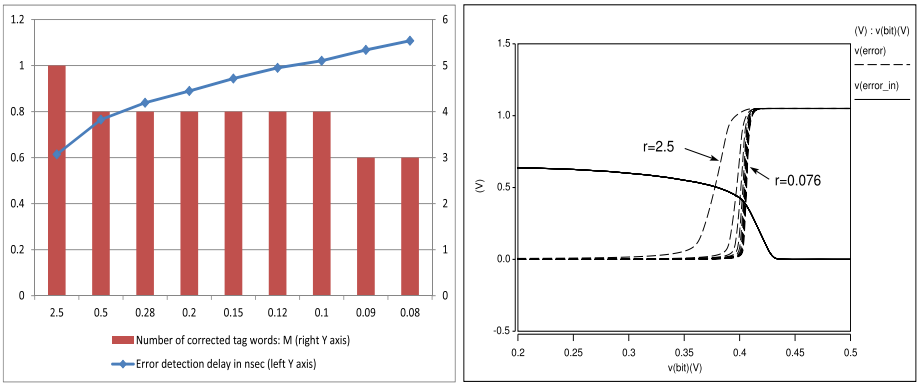
$$Correction\ Time < Cache\ Miss\ Penalty \tag{1}$$

This equation implies that given a cache miss penalty, there is an upper bound on how many tag words can be corrected. Thus, the optimization of the correction time is crucial. This will be discussed in next section.

3 Results

3.1 Simulations

To verify the functional correctness of our scheme, a cache simulator based on SimpleScalar [9] was implemented. In addition, SPICE simulations are done to measure the error detection time. The time includes the delay through the parity chain, pre-coder, and address encoder. Transistor and interconnect models for SPICE simulations are acquired from [5]. One key point to bear in mind is that



(a) (b)

Fig. 4. (a) Performance of the multi-bit correction scheme: error detection delay (left Y axis) vs the number of corrected tag words (right Y axis) with respect to the skew ratio of the error detection NAND gate (X axis). (b) Noise margin with varying NAND gate skew ratio.

the error detection delay does not affect the performance of cache operations since detection and correction is done as a back-ground process. However, we optimize the delay so that the total time for correcting M words is less than the L1 cache miss penalty of a modern processor. We perform numerous simulations using a 45 nm CMOS process technology. Fig. 4(a) shows the sum of PAR and ENC delay for the multi-bit correction scheme in Fig. 2(a). This delay includes the delay through the error parity chain and error address encoding. A horizontal axis indicates the PMOS-to-NMOS skew ratio of an error detection NAND gate shown in Fig 2(a). As the skew ratio increases, the detection speed becomes faster but noise margin is decreased. If the detection speed gets faster, it is likely to finish correcting more tag words before a false miss fetch returns from the lower level memory. In this simulation, we get processor data from [6] and assume that a processor core is running at 2.933 GHz and the L2 cache miss penalty is 3.4 nsec. Based on this information, we calculate how many tag words can be corrected as the detection speed gets faster due to skewing a NAND gate. The line labeled as corrected tag words indicates the number of corrected tags given speed of detection, which increases from 3 to 5 as the skew ratio increases. The noise margin with respect to varying NAND gate skew ratios is shown in Fig. 4(b). The simulation result shows that it decreases from 0.23 volts to 0.12 volts as the skew ratio increases. Thus, there is a trade-off between noise margin and number of corrections.

3.2 Cost of Proposed Scheme

Additional area penalties are introduced by mainly three factors: new CAM cell design including the parity circuit, additional parity bits, and an error address encoder. We use CACTI [7] and magic [8] to estimate the cache area of dif-

ferent configurations. The cache area includes both tag and data memory. For comparison, we choose a 32-way set-associative 32-KByte cache with a 32-byte line size and a 24-bit tag, which is a typical 1st-level on-chip cache [2]. Whereas the area for a baseline cache with no correction takes 1.9235 mm^2, the area for the multi-bit correction scheme (MBC) takes 2.0036 mm^2. Compared with the baseline, the cache area is increased only by 4.16 % in MBC. This is because a tag memory takes up a relatively small portion of total cache size compared with a data memory. For that reason, the area penalty in our scheme is small.

4 Conclusion

Our proposed scheme can be used effectively to detect and correct multiple soft errors in the tag portion of CAM-RAM cache. It detects and corrects false misses immediately and prevents fatal system errors and performance degradation whereas it requires a very small additional area.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2011-0023798, No. 2010-0025423, and 2010-0026822) and by the Sogang University Research Grant of 2011 (No.201110026).

References

1. Mukherjee, S., Emer, J., Fossum, T., Reinhardt, S.: Cache Scrubbing in Microprocessors: Myth or Necessity. In: Proc. IEEE Pacific Rim International Symposium on Dependable Computing, Papeete, Tahiti, pp. 37–42 (March 2004)
2. Hung, L., Goshima, M., Sakai, S.: Mitigating Soft Errors in Highly Associative Cache with CAM-based Tag. In: Proc. International Conference on Comupter Design, San Jose, California, USA, pp. 342–347 (Octorber 2005)
3. Lee, H.-J.: Immediate soft error detection using pass gate logic in content addressable memory. IEE Electronics Letters 44, 269–270 (2008)
4. Farkas, K.I., Jouppi, N.P.: Complexity/Performance Tradeoffs with Non-Blocking Loads. In: International Symposium on Computer Architecture, Los Alamitos, CA, USA, pp. 211–222 (1994)
5. `http://www.eas.asu.edu/~ptm/` (accessed December 2009)
6. Molka, D., Hackenberg, D., Schone, R., Muller, M.: Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System. In: International Conference on Parallel Architectures and Compilation Techniques, Raleigh, NC, USA, pp. 261–270 (2009)
7. Thoziyoor, S., Muralimanohar, N., Ahn, J.-H., Jouppi, N.: CACTI 5.1, technical report, HP Laboratories (April 2, 2008), `http://www.hpl.hp.com/techreports/2008/HPL-2008-20.html`
8. `http://opencircuitdesign.com/magic/` (accessed January 2011)
9. `http://www.simplescalar.com/` (accessed February 2011)