SimTag: Exploiting Tag Bits Similarity to Improve the Reliability of the Data Caches

Jesung Kim, Soontae Kim, Yebin Lee
Department of Computer Science
Korea Advanced Institute of Science and Technology (KAIST)
Daejeon 305-701, South Korea
{gecko1201, kims, smartcode}@kaist.ac.kr

Abstract— Though tag bits in the data caches are vulnerable to transient errors, few effort has been made to reduce their vulnerability. In this paper, we propose to exploit prevalent same tag bits to improve error protection capability of the tag bits in the data 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 similarity information is stored in the data caches as extra bits to be used later. When an error is detected in the tag bits, the similarity information is used to recover from the error in the tag bits. The proposed scheme has small area, energy, and performance overheads with error protection coverage of 97.9% on average. In contrast, the previously proposed In-Cache Replication scheme is shown to incur large performance and energy overheads.

I. INTRODUCTION

With continued technology scaling, caches are becoming more vulnerable to transient errors [2]. There have been many efforts made to address transient errors in the data arrays of the caches. However, errors in the tag bits of the caches are critical for data integrity, too. For example, transient errors in the tag bits can lead to false misses in the dirty cache lines and, consequently, stale data can be consumed. Therefore, addressing transient errors in the tag bits are critical for correction execution.

By our experiments with embedded benchmarks on an Intel Xscale-based simulator, most tag bits in the data caches have their replica in other cache sets. In other words, when a cache line is accessed or replaced, we can find an adjacent cache line with the same tag bits as those of the cache line accessed in a upper or lower cache set than the current set. This is called ***tag bits similarity*** in this paper. Tag bits similarity can be exploited for improving tag bits vulnerability against transient errors. For instance, when an error is detected using the conventional parity check bits, the error could be corrected if the same tag bits were present in one of adjacent cache lines. Faulty tag bits are simply replaced with correct tag bits from the adjacent cache line for error correction.

To exploit similar tag bits for transient error protection, we augment the conventional cache architecture with four simple hardware components. To access cache lines in a upper and/or lower cache set than currently accessed cache set, a shifter right after the decoder of a cache or a up/down counter is required. Second, an encoder for generating similarity information between tag bits is needed. Third, a small circuit is necessary for handling similarity bits on cache replacements. Finally, an error correction unit corrects transient errors in the tag bits using the same tag bits from adjacent cache lines. These extra components are simple structures and incur little energy, area, and latency overheads.

We evaluated our proposed scheme with in-cache replication (ICR) [4], which was originally proposed to reduce data array vulnerability but can also be applied to reduce tag bits vulnerability. From our experimental results, our scheme shows high error protection coverage of 97% with no virtual performance hit while ICR degrades overall system performance by around 10% and increases DRAM energy consumption by around 20%, on average.

This paper is organized as follows. Section 2 discusses prior schemes proposed to reduce the vulnerability of cache memories against transient errors. Section 3 describes our proposed cache architecture exploiting prevalent same tag bits. Section 4 presents our experimental results for our scheme and ICR. Finally, Section 5 concludes the paper.

II. RELATED WORK

Different techniques are proposed to protect against transient errors in microprocessors. Protection is generally achieved by employing redundancy; this redundancy may be in time [7], in area [4], or in information. Error Detection Code (EDC) and Error Correction Code (ECC) are used widely for protecting caches against transient errors [5]. However, the conventional ECC protection imposes significant area and latency penalties, making it practical only for large memories and second-level (L2) caches where the increased latency has little impact on performance [3]. To prevent latency increasing, first level (L1) caches tend to employ parity check codes that allow bit error detection, but no correction. Bhattacharya et al. investigate in detail multi-bit soft error rates in large L2 caches and propose a framework based on the amount of redundancy present in the memory hierarchy [11].

978-3-9810801-6-2/DATE10 © 2010 EDAA

TABLE I. **ERROR PROTECTION TECHNIQUES FOR TAG BITS**

Technique	No Detection		Parity		SECDED		Parity+SimTag		SECDED+SimTag		Parity+ICR		SECDED+ICR	
Cache Line	*Clean*	*Dirty*	*Clean*	*Dirty*	*Clean*	*Dirty*	*Clean*	*Dirty*	*Clean*	*Dirty*	*Clean*	*Dirty*	*Clean*	*Dirty*
False Hit	Error		Hit → Miss		Hit → Miss		Hit → Miss		Hit → Miss		Hit → Miss		Hit → Miss	
False Miss	Ignore	Error	Ignore	Error	Ignore	Correct only 1-Bit Error	Ignore	Correct Odd bits error if same tag exists	Ignore	Correct 2-bit error if same tag exists	Ignore	Correct Odd bits error if Replica exists	Ignore	Correct 2-bit error if replica exists
Replacement Error	Ignore	Error	Ignore	Error	Ignore	Correct only 1-Bit Error	Ignore	Correct Odd bits error if same tag exists	Ignore	Correct 2-bit error if same tag exists	Ignore	Correct Odd bits error if Replica exists	Ignore	Correct 2-bit error if replica exists

Despite the fact that most of the previous work has studied effectiveness in terms of performance, energy, and area overheads, it targets data bits reliability with the assumption that tag bits are intact. However, tag bits also are vulnerable in caches and they have different inherent properties compared to data bits.

Kim *et al.* classify tag bits faults into pseudo-hit, pseudo-miss (also called false-hit or false-miss in [10]), and multi-hit [2]. Asadi *et al.* present L1 and L2 cache vulnerability computation algorithms and also deal with algorithms for tag vulnerability computation [6]. They analyze in detail the sources of tag bits vulnerability. In-Cache Replication (ICR) has been proposed in [4] to replicate frequently accessed cache blocks to dead blocks. Replicated blocks can be used to correct tag bits errors in the active blocks. However, the dead block prediction technique is not always accurate. Thus, ICR increases cache miss and write-back rates resulting in large performance loss and increased energy consumption.

III. OUR PROPOSED APPROACH

A. *Effects of tag bits corruptions*

Transient errors in tag bits manifest themselves as false-hits, false-misses, and replacement errors. A false-miss makes cache hit as a cache miss because of transient error in tag bits. Consequently, the datapath gets wrong data on a read and updates a wrong location on a write. A false-hit refers to a cache hit that is actually a miss in the absence of a transient error. If tag bits are corrupted after the line is modified, it may write back to a wrong location in the next level of memory, which is classified as a replacement error [6].

Table I shows tag bits error protection techniques including our proposed scheme and ICR. Except for no detection, clean cache lines do not need error recovery. If erroneous data are in a clean cache line, they can be recovered by invalidating the cache line and by fetching correct data from the next level of memory. If an error occurs in a dirty cache line, hardware exception will be generated and an error handling mechanism will take over for error recovery. Parity check code can cover transient errors on clean caches but it cannot protect dirty cache lines. Single Error Correction Double Error Detection (SEC-DED) can detect 2-bit errors and correct 1-bit errors. These error codes can be augmented with our proposed technique. If single or multi-bit errors are detected by parity or SEC-DED, our technique will correct the errors using the same tag bits from adjacent cache lines. Our scheme and ICR use location information for error protection. However our scheme exploits prevalent same tag bits while ICR replicates tag bits into other locations by force.

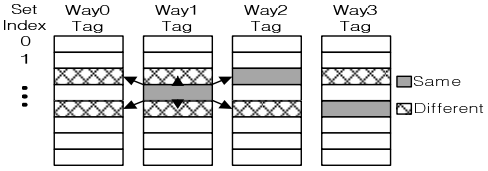

Figure 1. **Same tag bits in the adjacent sets**

B. *Exploiting spatial locality*

It is highly probable that same tag bits exist in adjacent cache sets (see Figure 1). This is a consequence of spatial locality of programs. The basic idea of our scheme is to exploit the same tag bits in an adjacent set for correcting erroneous tag bits. Additional bits are required to encode location information which points to exact location of the same tag bits in a upper or lower set. These extra bits are called *"Same Tag Information"* (STI). STI bits consist of three logical parts; a valid bit, a set location bit, and way location bits. The valid bit indicates that tag bits have the same bits in an adjacent set. The set location bit denotes a lower or upper set and way location bits represent a specific cache way which has the same tag bits.

To find same tag bits and set STI bits properly, extra components are required. The tag bits of the missed data are compared with the tag bits of adjacent sets during fetching data from the next level of memory on a cache miss. If there is a match, the STI bits for the missed data are generated and stored in the data cache. It is possible that replaced tag bits are indicated by the STI bits of adjacent sets (A). If this situation is not handled properly, the STI bits will point non-existing tag bits. To solve this problem, another extra component reads STI bits from lower and upper sets A. If there are matching STI bits, new STI bits are generated. All of these procedures are performed while pipelines are stalled due to cache misses. Therefore, there is no performance degradation virtually.

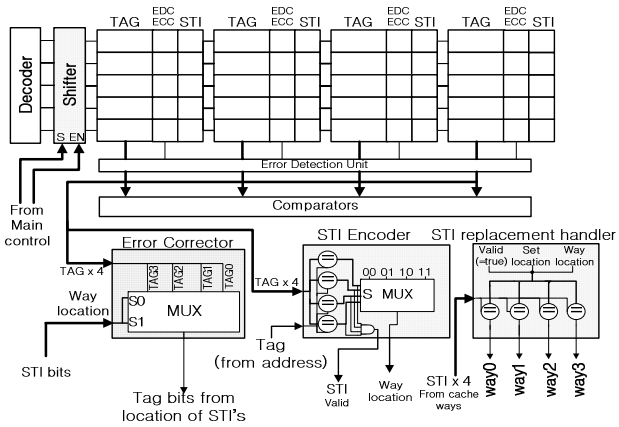

Figure 2. **Detailed architecture of SimTag**

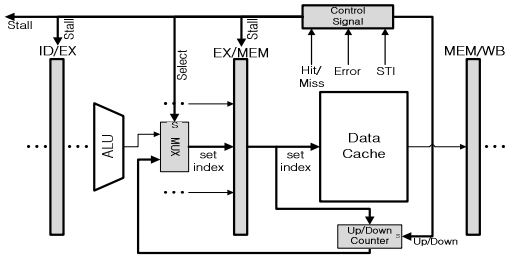

Figure 3. **Counter-based technique to access adjacent sets**

C. *Proposed architecture*

In this subsection, we present our proposed architecture and explain additional components which are required to implement it. Figure 2 shows the microarchitecure-level schematic of our architecture called ***SimTag***. Detailed operation of each component is described below.

Shifter. Our approach uses a shifter for accessing lower or upper cache lines. This is simple and intuitive but it may increase the critical path of data cache access. An alternative approach is to use a counter for hiding the decoding latency. Figure 3 shows the counter-based technique for set index control. Basic operation is same as in the shifter approach but it operates in parallel with cache access.

STI Encoder. To generate STI bits, STI encoder compares the tag bits of cache missed data with the tag bits from lower and upper sets during pipelines are stalled due to cache misses.

STI Replacement Handler. STI replacement handler checks the STI bits in the upper and lower sets on cache replacement. If the STI bits point to replaced tag bits in question, then simply invalidate the STI valid bits and generate new STI bits by finding other same tag bits.

Error Corrector. When errors are detected, this component fetches uncorrupted tag bits from an adjacent cache set by using STI bits (if same tag bits exist) for error correction.

Main Controller. On cache misses or tag bits errors, the pipelines are stalled and, at the same time, the main controller signals the additional shifter (or counter) to access adjacent sets.

There is little area overhead due to the additional components. A set index bits-wide shifter or counter is put into the cache. A 4-to-1 tag bits-wide multiplexer is used for tag matching in the Error Corrector. STI Encoder uses a 2-bits multiplexer, a 1-bit AND gate and tag bits-wide comparators.

TABLE II. **BASELINE SYSTEM CONFIGURATION**

Configuration Parameter	Value
Processor	
Functional unit	1 Integer/Floating-point ALU 1 Multiply-Accumulate (MAC) Unit
Frequency	600MHz
Cache and Memory Hierarchy	
L1 Instruction Cache	32KB, 32-way, 32byte blocks
L1 Data Cache	8KB, 4-way, 32byte blocks Write-back
Memory	32M SDRAM

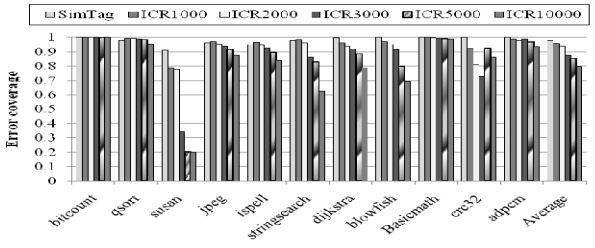

Figure 4. **Error protection coverage**

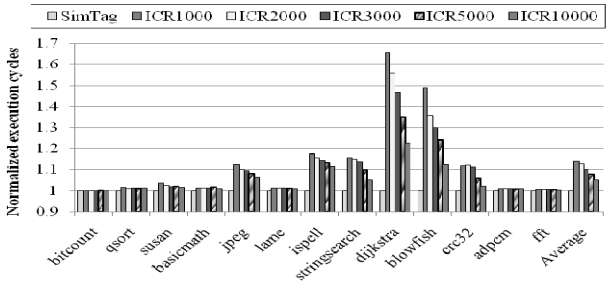

Figure 5. **Normalized Execution Cycles**

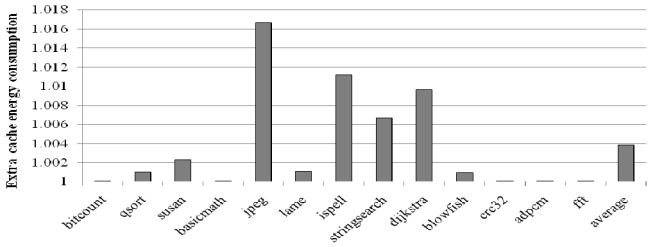

Figure 6. **Normalized cache energy consumptions of SimTag**

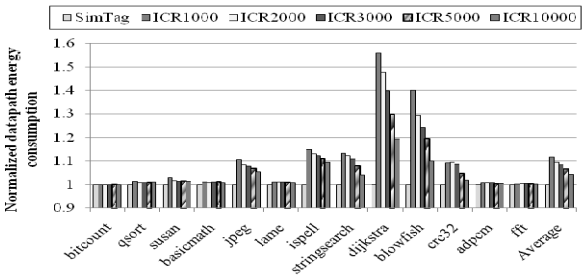

Figure 7. **Normalized datapath energy consumptions**

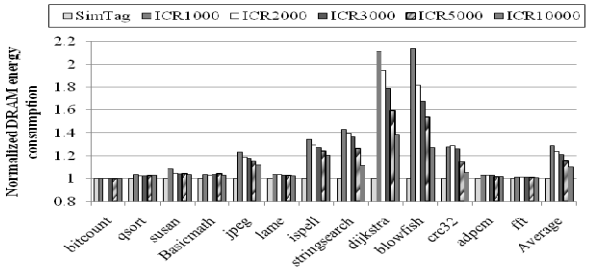

Figure 8. **Normalized DRAM Energy Consumption**

Also, STI bits-wide comparators are required for STI Replacement Handler. A few bits are required for the way location. These bits depend on the number of sets inside the data cache. If there are *M* sets, the size of way location bits is $\log_2(M)$

IV. Experimental Results

A. Experimental setup

For our experimental evaluation of the proposed SimTag, XEEMU [1], an improved Intel XScale PXA80200 power simulator, is used. Table II shows the detailed configuration of the simulated microprocessor. We implemented our proposed architecture on top of the XEEMU and also implemented ICR for comparison. For experimental evaluation, we use a subset of MiBench benchmarks [8] compiled for the XScale instruction set architecture.

B. Error coverage

As we discussed previously, the STI bits are set up when new tag bits are loaded into the cache. In case of clean cache lines, tag bits can be recovered from transient errors by interpreting tag bits errors into cache misses. Therefore, we count only the dirty blocks which have the same tag bits in upper or lower sets in calculating error protection coverage.

Figure 4 shows error protection coverage results of our proposed technique with ICR with various decay windows for dead block predictions. As we expect, ICR shows lower error coverage as decay window becomes larger. The average error coverage of our scheme is 97.9% while ICR covers 95.7%, 93.6%, 87.3%, 85.3% and 79.6% when decay window size is 1000, 2000, 3000, 5000 and 10000, respectively.

C. Performance

Figure 5 depicts the normalized execution cycles. ICR increases the miss rates caused by wrong prediction of dead-block. These results indicate that dead block prediction is not always accurate and, therefore, it can suffer from significant performance degradation. The average performance degradation is 13.9%, 12.6%, 10.1%, 7.9% and 5.0% when decay window size is 1000, 2000, 3000, 5000 and 10000, respectively. Our proposed technique does not incur performance degradation.

D. Energy consumption

Additional energy consumption. Figure 6 shows additional energy consumptions of SimTag normalized to baseline cache energy consumption. Average energy consumption increases is only 0.4%. We are not able to calculate exact additional energy consumption by ICR because it does not disclose its detailed architecture.

Datapath energy consumption. Figure 7 depicts the normalized datapath energy consumption excluding cache energy consumption. ICR incurs additional energy consumption due to increased execution cycles. It increases datapath energy consumption by 11.7%, 9.7%, 8.4%, 6.5% and 4.2% when decay window size is 1000, 2000, 3000, 5000, 10000, respectively. Our proposed architecture does not increase the datapath energy consumption because it does not increase execution cycles.

DRAM energy consumption. Due to the dead block prediction, ICR has more chances to evict dirty blocks to the main memory. This will increase DRAM power consumption naturally. Figure 8 shows normalized DRAM energy consumption. Average increased DRAM energy consumption is 29.0%, 23.9%, 20.9%, 16.2% and 9.9% when decay window size is 1000, 2000, 3000, 5000, and 10000, respectively. Our scheme does not increase DRAM energy consumption.

V. Conclusion

With the trend of increasing soft error rate, it is becoming important to provide error detection and correction capability for hardware circuits, especially for cache memories. However, most of the previous techniques focus only on data bits without considering tag bits corruption. Most tag bits in the data caches have their replica in adjacent cache sets from our experiments. We exploit this tag bits similarity against transient errors. Faulty tag bits are simply replaced with correct tag bits from the adjacent cache lines for error correction. Our technique shows 97.9% error protection coverage with low extra cache energy consumption, less than 0.4%, on average.

References

[1] Z. Herczeg, Ákos Kiss, D. Schmidt, N. Wehn, and T. Gyimóthy. Energy simulation of embedded XScale system with XEEMU. *Journal of Embedded Computing*, 3(3), 2009.

[2] S. Kim and A. Somani, "Area efficient architectures for information integrity in cache memories," In *Proc. ISCA*, pp. 246–255, 1999.

[3] S. Kim, "Reducing Area Overhead for Error-Protecting Large L2/L3 Caches," *IEEE Trans. on Computers*, Vol. 58, no. 3, pp. 300-310, March 2009.

[4] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam,"ICR: In-cache replication for enhancing data cache reliability," In *Proc. Int. Conf. Depend. Syst. Netw.*, pp. 291–300, 2003.

[5] C. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 397–404, May 2005.

[6] H. Asadi *et al*., "Vulnerability analysis of L2 cache elements to single event upsets," In *Proc. Design, Automation and Test in Europe*, pp. 1–6, Mar. 2006.

[7] Austin, T. M., "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," In *Proc. MICRO*, 1999.

[8] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. "MiBench: A Free, Commercially Representative Embedded Benchmark Suite." In *Proc. WWC-4*, Dec. 2001.

[9] S. Hong, S. Kim, "TEPS: Transient Error Protection Utilizing Sub-word Parallelism," In *Proc. IEEE Computer Society Anal. Symp. on VLSI,* pp.286-291*,* 2009.

[10] L.D. Hung, M. Goshima, and S.Sakai, "Mitigating soft errors in highly associative cache with CAM-based tag," In *Proc. IEEE Int.Conf. On Compoter Design* 2005, pp.342-347, 2005.

[11] K. Bhattacharya, N. Ranganathan, and S. Kim, "A Framework for Correction of Multi-Bit Soft Errors in L2 Caches Based on Redundancy," *IEEE Trans. VLSI.*, Vol. 17, issue. 2, pp. 196-206, 2009.