Error Detection and Correction using Bloom Filters

Battu Naveen [1], N.Pitcheswara Rao [2]

[1]PG Student, Electronics & Communication Engineering, DVR & Dr. HS MIC College of Tech., Kanchikacherla, A.P, India
[2]Assistant Professor, Electronics & Communication Engineering, DVR & Dr. HS MIC College of Tech., Kanchikacherla, A.P, India

Abstract: *The Bloom filter a way of using hash transforms to determine set member-ship. Bloom filters find application wherever fast set membership tests on large data sets are required. Such applications include spell checking, differential file updating, distributed network caches, and textual analysis. It is a probabilistic method with a set error rate. Errors can only occur on the side of inclusion, a true member will never be reported as not belonging to a set, but some non-members may be reported as members. Bloom filters use hash transforms to compute a vector (the filter) that is preventative of the data set. Membership is tested by comparing the results of hashing on the potential members to the vector. In its simplest form the vector is composed of N elements, each a bit. An element is set if and only if some hash transform hashes to that location for some key. Figure 2 shows such a filter with m = 4 hash transforms and N = 8 bits.*

Keywords *—bloom filters, error, detection, correction*

I. INTRODUCTION

Transient errors (also called soft errors, or single-event upsets) caused by external radiation events have become an important consideration for microprocessor design. Recent research shows that the uncorrected soft errors induce a failure rate higher than all the reliability mechanisms combined. Following the trends of shrinking feature sizes, low supply voltage and high frequency, future microprocessors will become increasingly vulnerable to soft errors.

Modern microprocessors typically employ cache memories to bridge the speed gap between the processor and the memory. Cache memories, however, are particularly susceptible to particle strikes since they consume a large fraction of on-chip area. In addition, the leakage control techniques aggressively used today for reducing cache leakage energy make the cache reliability problem even more severe. The accumulated charge from external particle strikes can invert the state of the SRAM cell, which can be easily propagated to the processor or lower-level memory, resulting in erroneous computation or system crash. Consequently, cache memories must be protected against soft errors to ensure dependable computing.

A number of approaches exist to improve cache reliability against soft errors, ranging from information redundancy to space redundancy (N Modular Redundancy). However, all these techniques come at additional costs in performance, energy, area or design time (which is called reliability cost in this paper). For microprocessors or embedded systems that are increasingly used in reliability-critical applications but with stringent cost constraints, it is a necessity to develop novel cost-effective fault tolerant techniques or to select the most cost-effective mechanism to meet the reliability goal. To achieve this goal, the first step is to understand and measure cache vulnerability to soft errors accurately and quantitatively. While over-estimation can result in excessive protection and thus higher reliability cost; under-estimation can lead to inadequate protection and hence is useless.

While cache memories are susceptible to particle strikes, not all soft errors occurred in caches can propagate to other system components. For instance, the soft errors occurred between the read and write operations are automatically overlapped by the later write operations, and thus do not affect other components. To quantify this effect, we define the cache vulnerability factor (CVF) to be the probability that soft errors in cache memories can be propagated to the processor or other memory hierarchy. In this paper, we also propose an approach to compute the CVF based on the cache line access patterns. Using the cache vulnerability factor as a reliability metric, we evaluate the degree of reliability for a variety of cache memories, including the Ll I-cache, the write-through data cache, the write-back data cache and the L2 cache. Our result quantitatively shows that the write-through data cache has the lowest CVF and thus the highest reliability. Since write-back data caches can usually achieve higher performance than write-through caches (assuming the same clock cycle time), we also propose two novel techniques to reduce the CVF of write-back data caches without impacting its high performance.

It is widely accepted that fault-inducing particle strikes are randomly and uniformly distributed, therefore, the probability that cache soft errors can be propagated to the processor or other components is equal to the average ratio that instructions/data are exposed to the susceptible intervals during the execution time, which is defined in the Equation. In this Equation, is any cache block that is loaded into the cache in study; and the Susceptible Time represents the time intervals that these blocks are susceptible to soft errors. Specifically, for instruction caches, the Susceptible Time is the R-R

intervals. For a write-through data cache, the Susceptible Time includes the R-R and W-R intervals. For a write-back data cache, however, the Susceptible Time includes the R-R, W-R and Dirty-Replacement intervals. The Lifetime in Equation represents the time interval between the time that the block is loaded into the cache and the time it is replaced. Since both the Susceptible Time and Lifetime for each block can be easily obtained from a cache or performance simulator, it would be straightforward to calculate the CVF for various cache memories.

$$CVF = \frac{\sum SusceptibleTime(block_i)}{\sum Lifetime(block_i)}$$

The CVF indicates the probability that soft errors of caches can impact other components. In other words, the CVF accurately discloses the capability of cache memories to mask soft errors by themselves without affecting other components. Therefore, we can develop new techniques to improve cache reliability by reducing its CVF (i.e., by optimizing the access patterns). Compared to conventional information redundancy or space redundancy techniques, the CVF-reduction techniques can improve cache reliability without providing redundancy. The concept of CVF even opens the avenues for software (e.g., compiler) to enhance cache reliability against soft errors by reordering the load/store operations and modifying the access intervals.

In addition, with a metric to evaluate cache reliability, it is possible to study the tradeoffs between performance and reliability quantitatively.

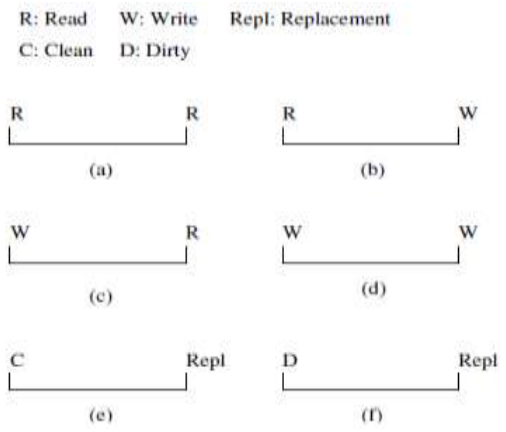

Figure 1.1 Access patterns (intervals) of cache lines (a) Read-Read pattern (b) Read-Write pattern (c) Write-Read pattern (d) Write-Write Pattern (e) Clean-Replacement pattern (f) Dirty-Replacement pattern

Table 1.1 Configuration parameters of simulated microprocessor

Configuration Parameter	Value
Processor	
Functional Units	4 integer ALUs 1 integer multiplier/divider 4 FP ALUs 1 FP multiplier/divider
Fetch Width	4 instructions/cycle
Cache and Memory Hierarchy	
L1 Instruction Cache	32KB, 1-way, 32 byte blocks 1 cycle latency
L1 Data Cache	32KB, 1-way, 32 byte blocks, WB 1 cycle latency
L2	1MB unified, 8-way LRU, WB 64 byte blocks 6 cycle latency
Memory	100 cycle latency
TLB Size	128-entry, 30-cycle miss penalty

A differential file contains a batch of database records to be updated. For performance reasons the database is updated only periodically (i.e., midnight) or when the differential file grows above a certain threshold. However, in order to preserve integrity, each reference/query to the database has to access the differential file to see if a particular record is scheduled to be updated. To speed-up this process, with little memory and computational overhead, we need bloom filters.

II. FAULT TOLERANT INTERLEAVED CACHE

A 4-way set associative cache efficient bloom filter memory as shown in fig.2.3 can be interleaved in a high-order fashion to make it fault tolerant. 8-individual lines of the cache can be locally addresses using 3 address lines. Here, the cache is divided into 2-sets, two least significant address lines (A_1A_0) can be used to point lines within a set. Whereas, the most significant address line A_2 is used as a select line of a multiplexer which lets the output data lines from the cache-sets pass through it. Here, if any faulty condition occur in lines of set#0 then this set can be removed from the cache structure simply by changing the bit value of A_2. But, high-order interleaved cache is slower and the speed advantage of cache memory is lost.

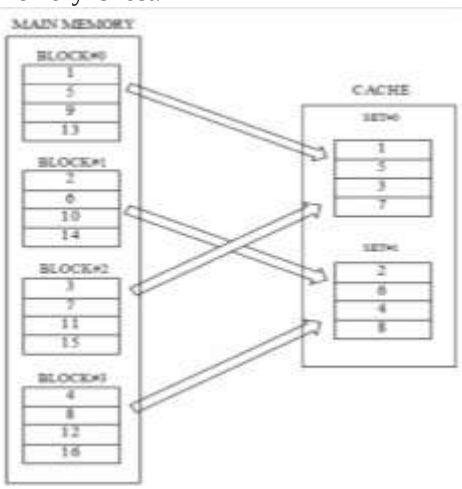

Fig. 2.1 4-way Set-associative Cache with 8 lines

Fig. 2.2. Cache replacement within 4-way set-associative Cache

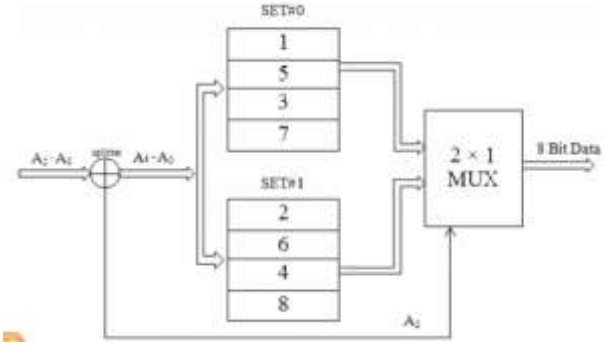

Fig. 2.3. High-order interleaved 4-way set-associative Cache

If the cache memory is low-order interleaved then it turns out to be a high speed one. But, low-order interleaved structures are not modular in nature and therefore, inherently fault-intolerant. Fig. 2.4 gives an idea of low-order interleaved cache structure. The cache is divided into 2-sets, two most significant address lines (A_2A_1) can be used to point lines within a set. Whereas, the least significant address line Ao is used as a select line of a multiplexer which lets the output data lines from the cache-sets pass through it. Here, if any faulty condition occur in lines of set#0 then this set cannot be removed from the cache structure by changing the bit value of Ao as it would jeopardize the entire cache addressing. Therefore, a fine grained approach is needed to remove the faulty line(s) from the cache address space without disturbing the contiguity and set-associativity of the cache memory. Fig. 2.5 And fig. 2.6 depicts a clear picture of the proposed structure in case of single and multiple faults respectively within this low-order interleaved cache structure. In fig. 2.5 as fault lies in line with address 100, this cache line is bypassed and the line addresses are reallocated in such a pattern that it avoids any gap in address space. So, address 100 now points to the next non-faulty cache line which was previously addresses by 101, address 101 points to the next cache line (previous address 110) and so on. The last cache line address points to the faulty cache line and thus address 111 holds a bit sequence in high impedance state (ZZZZZZZZ).

III.IMPLEMENTATION

Input is given as a cache text in the mat lab .mat lab software assigns the memory as input to the Xilinx blocks via system generator

Table 3.1 Input to the cache memory

Way 0	Way 1	Way 2	Way 3
00001000	00001001	00001010	00001000
00001001	00001011	00001101	00001110
00001010	00001001	00001000	00001011
00001011	00001000	00001100	00001101
00001100	00001111	00001110	00001011
00001101	00001010	00001011	00001001
00001110	00001100	00001111	00001111
00001111	00001101	00001101	00001100

Tag bits errors

Recently, a regular taxonomy for error detection classifies errors as unobserved, detected however unmanageable (DUE), or detected and correctable errors. Single event upsets in logic path will probably cause silent knowledge corruptions (SDCs) in stark system. SDCs square measure unobserved errors that cause incorrect machine results. DUE happens once detected errors exceed error protection capability. Tag arrays square measure usually protected by error protection codes, like parity or SEC–DED codes.

Transient errors in tag bits raise the chances of pseudo hits, pseudo misses, replacement errors, and multisite. Fig. 3(a) shows AN example of a pseudo hit. A pseudo hit refers to a success that's really a miss within the absence of a transient error. The pseudo hit could lead the knowledge path to use the incorrectly matched data. However, pseudo hit may be detected by storing check bits (parity or SEC–DED codes) for every row of tag bits.

Once AN access for a given input ends up in a success, the corresponding check bits of the tag bits square measure browse out then compared with the check bits computed from input tag bits. Any couple denotes that the tag bits are corrupted and therefore the hit is definitely a pseudo hit. A pseudo miss refers to a miss that's really a success once there's no transient error. Fig. 3(b) shows AN example of a pseudo miss. Pseudo misses could cause {a knowledge acknowledge an information} integrity drawback in a very write-back data cache, that holds dirty knowledge. A pseudo miss on dirty knowledge triggers a fetch and use of stale knowledge from a lower level. Justin case of a write-through cache, however, a pseudo miss doesn't cause a retardant. It simply incurs a cache miss.

If tag bits are corrupted when their corresponding cache line is changed, the changed knowledge is written back to a wrong location once write-back caches are used. Obviously, keep knowledge in backing storage are going to be lost and knowledge modification can't be mirrored in backing storage. Kind this sort} of error type is classed as a replacement error, as shown in Fig. 3(c). Additionally, transient errors could lead to multisite errors once caches ar associative. Fig. 3(d) shows associate degree example of a multisite in an exceedingly four-way set associative cache. At the most one tag bit should be matched in one cache set, however it's doable that multiple tag bits are matched in an exceedingly single set thanks to a pseudo hit. Multi hit errors occur sometimes, however their rate is exaggerated in extremely associative caches .

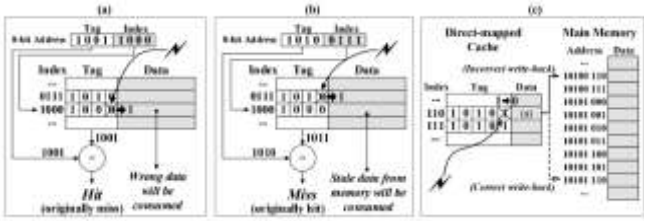

Fig. 3.1. Effects of tag bits error. (a) Pseudo-hit. (b) Pseudo-miss. (c) Replacement error

Table 3.2 Encoded pattern

Sti way 0	Sti way 1	Sti way 2	Sti way 3
0000000000100 0	0000000000100 1	0000000000101 0	0000000000100 0
0010100000100 1	0000000000101 1	0000000000110 1	0000000000111 0
0000000000101 0	0000000000100 1	0000000000100 0	0010100000101 1
0000000000101 1	0011110000100 0	0000000000110 0	0000000000110 1
0000000000110 0	0000000000111 1	0000000000111 0	0011110000101 1
0000000000110 1	0000000000101 0	0011110000101 1	0000000000100 1
0000000000111 0	0000000000110 0	0000000000111 1	0010100000111 1
0000000000111 1	0000000000110 1	0000000000110 1	0000000000110 0

From our experimental results, it's extremely probable that very same tag bits exist in adjacent cache sets. Fig. three shows locations of same tag bits associate degree exceedingly in every four-way set associative cache as an example. Once a cache line is loaded, a similar tag bits as those of the fresh fetched line square measure simply found in adjacent sets in spite of associativity. This is often a consequence of spatial neighbourhood of memory accesses. Even once the cache isn't absolutely fetched, tag bit similarity continues to be current to existing

Cache lines, since they're additionally fetched from the memory supported the abstraction neck of the words. Fig. 4 shows abstraction neck of the words of tag bits thoroughly. During this example, eight-bit main memory address Associate in nursing eight-entry direct-mapped cache area unit assumed. Once information (a) area unit accessed and transferred to the cache, the lower 3 bits of its address area unit used as a cache set index and higher 5 bits area unit used as tag bits [19]. As a result of the tag bits of memory address, most of tag bits area unit a similar in adjacent cache entries. Once concerning the information (a), information (b) settled close to the information (a) area unit accessed with a high likelihood due to the abstraction neck of the woods. If each information (a) and information (b) area unit cached, there are often 2 completely different information with a similar tag bits within the cache memory.

Therefore, there exist several same tag bits within the cache memory, albeit their information values area unit completely different . The fundamental plan of our theme is to take advantage of same tag bits in adjacent sets to correct inaccurate tag bits. By exploiting a similar tag bit values, we are able to extremely enhance error correcting capability of tag bits solely with negligible overheads. To achieve this goal, further bits area unit needed to cypher location data that points to an explicit location of a similar tag bits in Associate in nursing higher or lower set. These additional bits area unit referred to as same tag data (STI) bits.

STI bits include 3 logical parts: 1) a legitimate bit; 2) a group location bit; and 3) approach location bits. The valid bit indicates whether or not bound tag bits have a similar bits in Associate in Nursing adjacent set or not. The set location bit denotes Associate in Nursing higher or lower set and approach location bits represent a selected cache approach that has same tag bits. The length of approach location bits are often completely different as a result of it depends on the associativity of caches. For instance, a direct-mapped cache doesn't need the approach location bits as a result of its one approach. just in case of four-way set associative caches, however, 2 approach location bits area unit needed to contain approach data.

Simulink block diagram for encoding

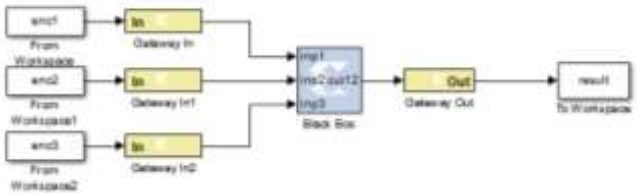

Fig. 3.2 Encoder design

Simulink blocks diagram for correction.

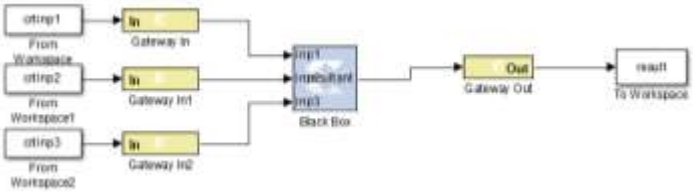

Fig. 3.3 Corrector

IV. 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.

Processor caches already play a critical role in the performance of today's computer systems. At the same time, the data integrity of words coming out of the caches can have serious consequences on the ability of a program to execute correctly, or even to proceed. The integrity checks need to be performed in a time-sensitive manner to not slow down the execution when there are no errors as in the common case, and should not excessively increase the power budget of the caches which is already high. A novel solution to this problem by allowing in-cache replication, wherein reliability can be enhanced without excessively slowing down cache accesses or requiring significant area cost increases. The mechanism is fairly power efficient in comparison to other alternatives as well. In particular, the solution replicates data that is in active use within the cache itself while evicting those that may not be needed in the near future. Our experiments show that a large fraction of the data read from the cache have replicas available with this optimization.

References

[1] "Optimal false-positive-free bloom filter design for scalable multicast forwarding", IEEE/ACM Transactions on Networking, vol. 23, no. , pp. 1832-1845, Dec. 2015, doi:10.1109/TNET.2014.2342155

[2] "The Bloom Paradox: When Not to Use a Bloom Filter", IEEE/ACM Transactions on Networking, vol. 23, no. , pp. 703-716, June 2015, doi:10.1109/TNET.2014.2306060

[3] Jiangbo Qian, Qiang Zhu, Yongli Wang, "Bloom Filter Based Associative Deletion", IEEE Transactions on Parallel & Distributed Systems, vol. 25, no. , pp. 1986-1998, Aug. 2014, doi:10.1109/TPDS.2013.223

[4] Joao Trindade, Teresa Vazao, "A Performance Evaluation of HRAN: A Hybrid Routing Protocol Using Bloom Filters for Wireless Mobile Ad Hoc Networks", 2013 IEEE 12th International Symposium on Network Computing and Applications, vol. 00, no. , pp. 139-142, 2012, doi:10.1109/NCA.2012.17

[5] Gang Wang, Jing Liu, Xiao-Guang Liu, Guang-Jun Xie, Jun Lee, "K-Divided Bloom Filter Algorithm and Its Analysis", Future Generation Communication and Networking, vol. 01, no. , pp. 220-224, 2007, doi:10.1109/FGCN.2007.157

[6] Yuh-Jzer Joung, An-Hsun Cheng, "Probabilistic file indexing and searching in unstructured peer-to-peer networks", , vol. 00, no. , pp. 9-18, 2004, doi:10.1109/CCGrid.2004.1336543

[7] Hyesook Lim, Miran Shim, Jungwon Lee, "Name prefix matching using bloom filter pre-searching", 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), vol. 00, no. , pp. 203-204, 2015, doi:10.1109/ANCS.2015.7110141

Author's Profile

BATTU NAVEEN is a PG student pursuing his M.Tech in VLSI & ES specialization in DVR & Dr. HS MIC College of Technology, Vijayawada.

N.PITCHESWARA RAO is working as Assistant Professor in Dept. Of ECE in DVR & Dr. HS MIC College of Technology, Vijayawada. He has published several papers on his research work and guided so many PG students.