Vulnerability Analysis of L2 Cache Elements to Single Event Upsets

Hossein Asadi Vilas Sridharan Mehdi B. Tahoori David Kaeli

Dept. of Electrical & Computer Engineering, Northeastern University
360 Huntington Ave., Boston, MA 02115
E-mail: {gasadi,vilas,mtahoori,kaeli}@ece.neu.edu

Abstract

Memory elements are the most vulnerable system component to soft errors. Since memory elements in cache arrays consume a large fraction of the die in modern microprocessors, the probability of particle strikes in these elements is high and can significantly impact overall processor reliability. Previous work [2] has developed effective metrics to accurately measure the vulnerability of cache memory elements. Based on these metrics, we have developed a reliability-performance evaluation framework, which has been built upon the Simplescalar simulator.

In this work, we focus on the reliability aspects of L1 and L2 caches. Specifically, we present algorithms for tag vulnerability computation and investigate and report in detail on the vulnerability of data, tag, and status bits in the L2 array. Experiments on SPECint2K and SPECfp2K benchmarks show that one class of error, replacement error, makes up almost 85% of the total tag vulnerability of a 1MB write-back L2 cache. In addition, the vulnerability of L2 tag-addresses significantly increases as the size of the memory address space increases. Results show that the L2 tag array can be as susceptible as first-level instruction and data caches (IL1/DL1) to soft errors.

1. Introduction

Cache memory is a fundamental component used to enhance the performance of modern microprocessors. Errors in cache memories can be stored back to main memory and can easily lead to data integrity issues [12].

Soft errors, also called *Single Event Upsets* (SEU), are the largest contributors to the vulnerability of memory systems [16]. Most soft errors are the result of particle strikes on silicon devices. These strikes, which are non-repeatable due to the transient nature of the particles, inject charge into the devices which can alter saved or in-transit values. The most common incident particles are neutrons from cosmic rays and alpha particles from packaging materials. Currently, memory cells are the most vulnerable system components to soft errors [6, 14]. Estimated soft error rates of typical designs such as microprocessors, network processors, and network storage controllers show that sequential elements and unprotected SRAMs account for 49% and 40% of the overall soft error rate, respectively [14].

There has been prior research done that studies reliability issues and proposes solutions. Processor pipelines have been studied in [15, 23]. L1 caches have been studied in [2, 25] and subsequently in [4]. In addition, many modern processors implement some form of protection such as *error correction codes* (ECC) on their L2 data arrays [8, 3, 10]. However, only a few processors provide protection bits for their L2 tag arrays [1, 19, 24], since the extra delay imposed by ECC computation on tag bits increases cache hit and miss times. As on-chip L2 caches increase in size, the size of their tag arrays increases proportionally. This growth will increase the likelihood of errors in the tag arrays and will increase the vulnerability of L2 tag-addresses to soft errors.

When using a *write-back* policy for L2 caches, errors on tag-addresses can have a more serious impact on data integrity than errors on the data portion. This is because errors in the data portion can be recovered by time redundancy techniques [13]. But in the case of an error in a tag-address, the most recent data written to the block cannot be recovered because the original block address is no longer available.

In our previous work [2], we studied the vulnerability of L1 data caches and proposed mechanisms that improve reliability by performing refetching. In this work, we extend that analysis to explore the vulnerability of L2 caches. In particular, we investigate what level of protection is necessary for L2 tag arrays by comparing their vulnerability to that of DL1 and IL1 caches. We use the concepts of *Critical Words* (CWs) and *Critical Time* (CT) as our cache reliability metric [2]. CWs are those words in the cache that are either eventually consumed by the CPU or committed to memory by a write. CT can be either *Read-CT* or *Write-CT*. Read-CT is the interval between the cycle the word is brought into the cache (or updated) and the cycle it is used by the CPU. Write-CT is the interval from the cycle in which the word is last modified by the CPU to the cycle in which the word is written back to memory.

If an error in a CW is encountered during its critical time, this will result in an erroneous value being propagated out of the cache. However, if an error occurs during a non-critical time, no program failure will result. Thus, the reliability of the cache system only depends on the correctness of the CW words.

In this paper, we present effective algorithms to compute the vulnerability of both L1 and L2 caches and introduce the issue of the vulnerability of second-level caches (L2). We provide a breakdown of the vulnerability of data, tag-addresses, and status bits of cache memories. We also analyze in detail the sources of tag error vulnerability, assigning each error to one of three classes: 1) pseudo-hit, 2) multi-hit, and 3) replacement error.

The results of our analysis can be used to develop efficient protection mechanisms for these memory cells. Experiments on the SPECint2K and SPECfp2K benchmarks show that the vulnerability of L2 tag-addresses significantly grows as the size of the memory address space increases. Our results also show that the L2 tag array is as susceptible to soft errors as the IL1/DL1 caches.

The rest of this paper is organized as follows: Section 2 describes the error rate and reliability background; Section 3 presents the algorithms used to compute the vulnerability of data, tag-addresses, and status bits; Section 4 presents our experimental setup; Section 5 presents our reliability profiling results; and Section 6 concludes the paper.

2. Background

The *Soft Error Rate* (SER) for a device is defined as the error rate due to SEUs. A system's error rate budget for repairable components is commonly expressed in terms of the *Mean-Time-Between-Failures* (MTBF) and the *Mean-Time-To-Repair* (MTTR); for non-repairable components, *Mean-Time-To-Failure* (MTTF) is normally used. *Failures-in-Time* (FIT) is another commonly used error rate metric. FIT error rates are inversely proportional to MTBFs if the reliability function obeys the exponential failure law [9]. One FIT is equal to one failure in a billion hours. Current predictions show that typical FIT rates for latches and SRAM cells (measured at sea level) vary between 0.001-0.01 FIT/bit [7, 17, 22].

The overall FIT rate of a chip is calculated by adding the effective FIT rates of all the individual components. The FIT rate of each component is the product of its *raw FIT rate* and its associated *Vulnerability Factor*. The *Vulnerability Factor* (VF) is defined as the fraction of faults that become errors [15]. Therefore, the FIT rate of the entire system can be computed as follows:

$$FIT_{Chip} = \sum_{i} raw\ FIT_{Element(i)} \times VF_{Element(i)} \quad (1)$$

The reliability of a chip during the period $[0, t]$ is defined as the probability that the chip operates correctly throughout this period [9]. The reliability of a chip at time t can be computed as follows:

$$Reliability_{Chip}(t) = e^{-FIT_{Chip} \times t} = e^{\frac{-t}{MTTF_{Chip}}} \quad (2)$$

2.1. Errors in data, address tags, and status bits

In this paper we use the *Critical Words* (CWs) and *Critical Time* (CT) definitions we have previously introduced in [2]. To investigate the impact of errors in address tags and status bits, we extend the classification provided in [2] and [12] and study how these errors individually affect tag reliability.

There is a tag address associated with every cache line. The width of the tag is a function of the size of the address space, the number of cache lines, and the associativity of the cache. Bit changes in the tag array may cause the following types of errors:

Pseudo-miss: The tag address of a line does not match the requested address tag, but the line should have matched.

Pseudo-hit: The tag address of a line matches the requested address tag, though no match should have been found on this entry.

Replacement error: The tag address of a line is changed after the line has been modified, but before it is written back to the next level of memory.

Multi-hit: The tag address of a line is changed to match another tag address in the same cache set.

We model status bits as follows. Status bits typically consist of a valid bit and, in a write-back data cache, a dirty bit. For an error that occurs in a dirty bit, when the error modifies a 0 to a 1, the flip does not affect data integrity. But when a soft error causes a change from 1 to 0 in a dirty bit, the new value of the line is lost if it is replaced before it is re-written.

For an error occurring in a valid bit, if the bit changes from 1 to 0, the impact to data integrity depends on the current state of the line. If the line was clean, there may be a slight impact on performance (i.e., an extra cache miss), but there will be no data corruption. If the line was dirty, however, the most recent data written to the line will be lost.

In our vulnerability computation algorithm for status-bits, we have computed the vulnerability exactly for the cases described above. Another situation that occurs for valid bits is when an error changes a valid bit from 0 to 1. This will change the status of an invalid line to a valid line. This would corrupt data only if the line is requested by the CPU before it is replaced by another clean line. In our algorithm, we over-estimate this vulnerability by the following equation:

$$Vulnerability_{StatusInvalidBit} = (CacheBlocks) \times (PercentageofInvalidBlocks) \quad (3)$$

Our experiments show that this vulnerability constitutes less than 0.2% of the overall vulnerability. So, considering the relative vulnerability of status bits, our estimation method is 99.8% accurate.

Results that are never re-referenced (results of dynamically dead instructions [15]) contribute to the vulnerability only if they are written to the next level of memory. Results which are referenced by dynamically-dead instructions will contribute to the vulnerability. Since our methodology is focused on computing the reliability of the cache and not the entire system, we say a word is vulnerable if it propagates from the cache to another part of the system (either to the processor or to the memory). In addition, the vulnerability of silent stores [13] is computed properly: assuming no intervening reads, the first store will not contribute to the cache vulnerability.

2.2. Reliability computation

If a CT is assigned to every CW, then the cache system dependability parameters can be computed as follows [2]:

$$FIT_{Cache} = \frac{raw\ FIT\ per\ bit \times bpw \times \sum_{i=1}^{N} CT_i}{TT} \tag{4}$$

$$Vul_{Cache} = \frac{bpw \times \sum_{i=1}^{N} CT_i}{TT} \tag{5}$$

$$FIT_{Cache} = raw\ FIT\ per\ bit \times Vulnerability \tag{6}$$

$(TT = Total\ Execution\ Time, N = Number\ of\ CWs,$ $bpw = Bits\ per\ word, Vul = Vulnerability)$

Note that the metric of Vul_{Cache} is a dimensionless quantity. Expressions 4, 5, and 6 will be used in our experiments to evaluate the FIT and the vulnerability of caches. As the length of CTs increases, the vulnerability of the cache system increases as well. In other words, the longer that critical data stays in the cache, the greater the probability that an error in the cache will be propagated to the outside.

3. Vulnerability Estimation Algorithms

In this section, we present effective algorithms to compute the vulnerability of data, tag-addresses, and status bits. In the first algorithm, we describe how to compute the vulnerability of data RAMs. We associate one variable ($AccessTime$) with each word in the cache. $AccessTime$ is reset on a FILL or WRITE HIT event. On a READ HIT, we first update the total vulnerable time and then reset $AccessTime$.

We then multiply the length of the vulnerable interval by the word size in bits (ws). In case of a line REPLACEMENT or line FLUSH for writeback caches, we add the vulnerable time of words if the line is dirty. No action is needed when we do line REPLACEMENT or line FLUSH for a writethru cache, since there are no dirty lines in the cache.

To efficiently simulate the vulnerability of the memory system while running real applications, we utilize recent advances in simulation technlogy that utilize representative program samples. This can allow us to reduce our simulation time, without any significant loss in accuracy in our results. In our simulations, we utilize *SimPoint* [18], only executing a limited number of program instructions rather than the whole program execution (see Table 1). However, to account for the vulnerability of those dirty lines that are still in the cache, we flush those lines at the end of simulations. Finally, we divide the computed data vulnerability by the number of simulated cycles according to equation 5.

Benchmark	SimPoint	Benchmark	SimPoint
art-110	33,500 M	wupwise	58,500 M
bzip2-source	59,300 M	swim	600 M
gcc-166	10,000 M	mgrid	700 M
gzip-source	31,700 M	applu	1,900 M
mcf	97,800 M	galgel	315,100 M
mesa	9,000 M	equake	19,500 M
vpr-place	6,800 M	ammp	213,100 M
crafty	100 M	lucas	3,600 M
parser	1,700 M	fma3d	29,900 M
twolf	3,200 M	apsi	4,700 M

Table 1. SPEC2000 benchmarks in this paper (1M = one million).

Our second algorithm computes the vulnerability of tag-addresses and status bits. As described before in Section 2, tag vulnerability consists of pseudo-hit vulnerability, multi-hit vulnerability, and replacement vulnerability. We associate three variables with each block to compute these vulnerabilities ($BlockFillTime$, $BlockMultihitTime$, and $BlockWriteTime$).

To compute the pseudo-hit vulnerability, we compare the tag of the requested address (tag_r) to all tag addresses inside the target set (tag_{b_i}) on a miss. If tag_r and tag_{b_i} differ by one bit, it means that tag_{b_i} was vulnerable between the time tag_{b_i} is brought into the cache and the time tag_r is requested ($now - BlockFillTime[b_i]$). $BlockFillTime$ keeps the fill time of the block.

To compute the multi-hit vulnerability, on a hit, we compare the requested tag (tag_r) to all tag addresses inside the target set (tag_{b_i}) excluding the target block. If there is exactly one bit difference between tag_r and tag_{b_i}, it means that tag_{b_i} was vulnerable between its fetch time and the time tag_r is requested ($now - BlockMultihitTime[b_i]$). To avoid double counting of the multi-hit vulnerability, we associate $BlockMultihitTime[b_i]$ with each block. Initially, we set $BlockMultihitTime[b_i]$ to the fill time. Thereafter, during every time interval we update the multi-hit vulnerability, and we reset $BlockMultihitTime[b_i]$ to the current cycle (now).

To compute the replacement vulnerability, we associate one variable with each block ($BlockWriteTime$). This variable is set if this is the first write to the block. On a replacement or flush of block r, we calculate the total time the block was dirty, and add that to the replacement vulnerability ($now - BlockWriteTime[r]$). To compute the

Config. Parameter	Value
Processor	
Functional Units	4 integer ALUs, 4 FP ALUs 1 integer multiplier/divider 1 FP multiplier/divider
LSQ Size / RUU Size	8 Instructions / 16 Instructions
Fetch / Decode	4 / 4 instructions/cycle
Issue / Commit Width	4 / 4 instructions/cycle
Fetch Queue Size	4 instructions
Cycle Time	1 ns
Cache and Memory Hierarchy	
L1 Instruction Cache (IL1)	64KB, 1-way, 64 byte lines 1 cycle latency, 18Kbit tag array
L1 Data Cache (DL1)	64KB, 4-way, 64 byte lines Write-thru, no allocate on write miss 1 cycle latency, 20Kbit tag array
L2	1MB unified, 8-way, 128 byte lines 6 cycle latency, write-back
Memory	100 cycle latency
Branch Logic	
Predictor	Combined, bimodal 2KB table 2-level 1KB table, 8 bit history
BTB	512 entry, 4-way
Mis-prediction Penalty	3 cycles

Table 2. Default configuration parameters used in our simulations.

status-bit vulnerability, we compute the interval time that replacement vulnerability matters. As described before, we flush the cache at the end of the simulation period to account for the vulnerability of dirty lines still in the cache. Finally, we divide the computed vulnerabilities by the number of simulated cycles ($TotalSimulationCycle$) according to equation 5.

4. Experimental Setup

For these experiments, we used the *sim-outorder* processor simulator provided in *SimpleScalar 3.0* [5]. To evaluate the reliability of DL1, IL1, and L2 caches, we have extended the SimpleScalar source code to integrate our reliability estimation method. Our evaluation uses the SPECint2K and SPECfp2K benchmark suite [21] compiled for the Alpha ISA [11]. We utilize the SimPoint early simulation points in all of our results [18], as shown in Table 1.

The default system parameters (cache size, associativity, etc.) are detailed in Table 2, and were chosen to be representative of modern state-of-the-art processors. The cache configuration of a sample of current high-performance processors has been shown in Table 3. As can be seen in Table 3, the typical size of the first-level data and instruction caches (DL1 and IL1) and the typical size of the second-level caches (L2) in current high-performance processors are on the order of 64KB and 1MB, respectively.

Processor	DL1	IL1	L2
Pentium IV [8]	8K/16K	8K/16K	256K/512K/1M
POWER4 [3]	64K/128K	64K	1.41M
POWER5 [10]	96K*	96K*	1.875M
AMD Athlon64 [1]	64K	64K	1M
Itanium2 [19]	16K	16K	256K
IBM 390 [20]	256K	256K	4M

Table 3. Survey of some cache organizations in current microprocessors (*: per core).

5. Reliability Profiling Results

Figure 1 shows the L2 vulnerability profiling for twenty SPEC2000 benchmark programs. Our work shows that the data vulnerability constitutes more than 95% of the total L2 vulnerability. As an example, the reliability of an unprotected L2 cache when the target application is *gcc* ($vulnerability_{gcc} = 6 \times 10^6$) for a 6-month execution can be computed as follows (assume the raw error rate is 0.005 FIT/bit, which is a typical error rate at sea level [7, 17, 22]):

$$FIT_{L2} = 0.005FIT_{bit} \times 6 \times 10^6 = 30,000FIT$$

$$Reliability_{L2} = e^{-4320 \times 30,000 \times 10^{-9}} = 0.88$$

This level of reliability is not acceptable for critical applications, so current processors utilize ECC to protect the data portion of memory. L2 data arrays are protected in all current high-performance processors [1, 8, 3, 10, 19, 24]

Since ECC protection reduces the effective vulnerability of the L2 data array to zero, the primary source of L2 vulnerability is the tag array. Figure 2 shows the vulnerability of tag addresses versus total IL1 and DL1 (data, tag, and status) vulnerability. As demonstrated in this figure, the L2 tag vulnerability is on the order of IL1 and DL1 vulnerability, and is greater than the IL1 and DL1 vulnerability for half of the benchmarks (*bzip, gcc, mcf, vpr, swim, mgrid, applu, galgel, ammp*, and *lucas*).

The results also show that the instruction cache vulnerability is twice the data cache vulnerability on average. This is for two major reasons. First, there are twice as many accesses to the IL1 than to the DL1 on average. This leads to a greater fraction of critical words resident in the IL1, and a corresponding increase in IL1 vulnerability. Second, there are no writes to the IL1. Writes change any preceding residence time of a word to non-critical for future reads (they “reset” the beginning of the critical period). This has the effect of lowering the DL1 vulnerability compared to the IL1. Note that this conclusion would be different for a write-back DL1: [2] shows that a 64KB write-back data cache has 8 times the vulnerability of a 64KB write-thru cache.

To further investigate the vulnerability of L2 tag-addresses, we have profiled the L2 tag vulnerability in terms of pseudo-hit vulnerability, replacement vulnerability, multi-hit vulnerability, and status vulnerability. We have computed these vulnerability values according to the algorithms given in Section 3. Figure 3 shows that replacement vulnerability makes up almost 85% of the total tag vulnerability. This is due to the fact that tag-addresses be-

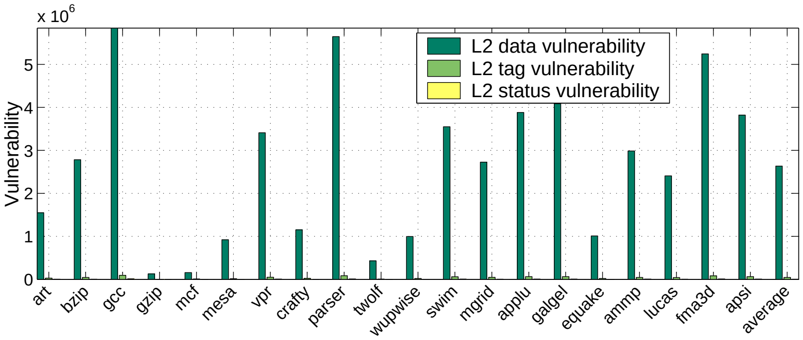

Figure 1. L2 Cache Vulnerability Profiling.

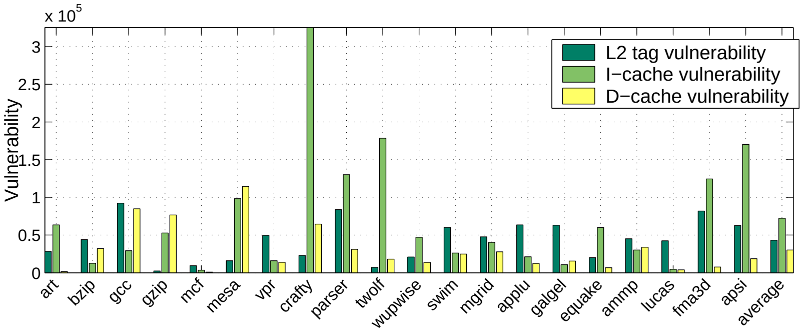

Figure 2. Comparison of L2 Tag Vulnerability with IL1 and DL1 Vulnerability.

come more susceptible once the first write occurs in the block and remain susceptible to SEUs until the block is replaced or flushed to lower levels of memory.

We have also explored the impact of increasing the memory address space on the vulnerability of tag-addresses. Figure 4 shows the relationship between the size of the memory address space and the vulnerability of tag-addresses. As can be seen in this figure, the cache tag array of a 48-bit memory address is almost 80% more susceptible to soft errors than the tag array of a 32-bit memory address. As an example, the tag vulnerability for a 48-bit memory address when running *gcc* increases to 1.8×10^5. The reduction in reliability for a 2-year period can be computed as follows (assuming a raw error rate of 0.05 FIT/bit at a very high altitude city (like Leadville, CO) which has 10x higher cosmic ray flux than sea level [26]):

$$FIT_{L2tag_{32bit}} = 0.05FIT_{bit} \times 9 \times 10^4 = 4,500FIT$$

$$Reliability_{L2_{32bit}} = e^{-17,532 \times 4,500 \times 10^{-9}} = 0.92$$

$$FIT_{L2tag_{48bit}} = 0.05FIT_{bit} \times 1.8 \times 10^5 = 9,000FIT$$

$$Reliability_{L2_{48bit}} = e^{-17,532 \times 9,000 \times 10^{-9}} = 0.85$$

The reliability decreases from 0.92 to 0.85 when increasing the address size from 32 to 48 bits.

6. Conclusions

This paper has presented new algorithms for calculating cache vulnerability along with a complete modeling methodology that captures the different types of possible soft errors that can occur in the cache hierarchy. We have built our reliability models on top of the SimpleScalar framework, in order to be able to assess reliability and performance in the same infrastructure.

We have presented a detailed reliability profiling approach for assessing both level-1 and level-2 data caches (data bits and tag bits) using SPEC2K benchmark suite. Our experimental data shows that although the vulnerability of the tag addresses in the L2 caches is far less than that of data bits, L2 tag vulnerability is comparable with the vulnerability of L1 data and instruction caches. L2 data bits are normally protected by ECC whereas L2 tag bits are left unprotected in most architectures. Experiments on the SPECint2K and SPECfp2K benchmarks show that dirty-line writeback errors make up 85% of the total tag vulnerability of a 1MB write-back L2 cache. These results confirm that effective protection schemes are required for L2 tag bits.

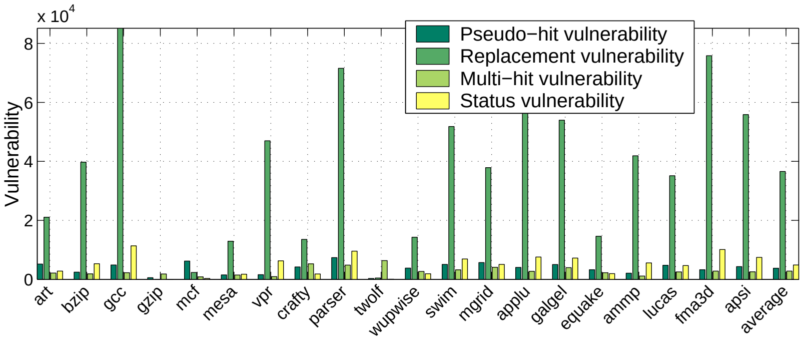

Figure 3. Tag Vulnerability Profiling

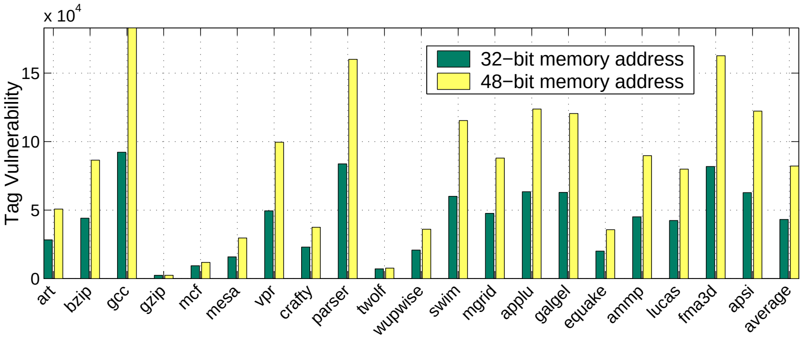

Figure 4. L2 Tag Vulnerability Increase versus Memory Address Space

References

[1] AMD Athlon(TM) 64 Processor, http://www.amd.com.

[2] H. Asadi, V. Sridharan, M. B. Tahoori, D. Kaeli, “Balancing Performance and Reliability in the Memory Hierarchy,” Proc. of the IEEE Intl. Symp. on Performance Analysis of Systems and Software (ISPASS), Austin, Texas, March 2005.

[3] S. Behling, R. Bell, P. Farrell, H. Holthoff, F.O. Connell, and W. Weir, “The POWER4 Processor Introduction and Tuning Guide,” IBM redbooks, www.redbooks.ibm.com,Nov. 2001.

[4] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, “Computing Architectural Vulnerability Factors for Address-Based Structures,” Proc. of the 32nd Annual Intl. Symp. on Computer Architecture (ISCA’05), pp. 532-543, 2005.

[5] D. Burger and T. M. Austin, “The SimpleScalar Tool Set, Version 2.0,” University of Wisconsin-Madison, Computer Science Dept., Technical Report No. 1342, June 1997.

[6] J. Gaisler, “Evaluation of a 32-bit Microprocessor with Built-in Concurrent Error-Detection,” Proc. of 27^{th} Intl. Symp. on Fault-Tolerant Computing(FTCS-27), pp. 42-46, June 1997.

[7] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walstra, and C. Dai, “Impact of CMOS Scaling and SOI on Soft Error Rates of Logic Processes,” Symp. on VLSI Technology, Digest of Tech. Papers, pp. 73-74, June 2001.

[8] Intel Pentium IV Processor, http://www.intel.com.

[9] B. W. Johnson, “Design & analysis of fault tolerant digital systems,” A&W Longman Publishing, ISBN:0-201-07570-9, Boston, MA, 1988.

[10] R. Kalla, S. Balaram, J.M Tendler, “IBM Power5 Chip: a Dual-Core Multithreaded Processor,” IEEE Micro, pp. 40-47, Vol. 24 , Issue 2, Mar-Apr 2004.

[11] R. Kessler, “The Alpha 21264 Microprocessor,” IEEE Micro, 19(2):24–36, March 1999

[12] S. Kim and A. K. Somani, “Area Efficient Architectures for Information Integrity in Cache Memories,” Proc. of the Intl. Symp. on Computer Architecture (ISCA), pp. 246-255, Atlanta, Georgia, 1999.

[13] K. M. Lepak and M. H. Lipasti, “Silent Stores for Free,” Proc. of the Intl. Symp. on Microarchitecture (MICRO-33), pp. 22-31, 2000.

[14] S. Mitra, N. Seifert, M. Zhang, Q. Shi and K. Kim, “Robust System Design with Built-In Soft-Error Resilience”, IEEE Computer, vol. 38, pp. 43-52, Feb. 2005.

[15] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, “A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor,” Proc. of the Intl. Symp. on Micro-architecture (MICRO-36), 2003.

[16] H. T. Nguyen and Y. Yagil, “A Systematic Approach to SER Estimation and Solutions,” Proc. of the Intl. Reliability Physical Symp., pp. 60-70, Texas, 2003.

[17] E. Normand, “Single Event Upset at Ground Level,”IEEE Trans. on Nuclear Science, Vol. 43, No. 6, pp. 2742-2750, Dec. 1996.

[18] E. Perelman, G. Hamerly, and B. Calder “Picking Statistically Valid and Early Simulation Points,” Proc. of the Intl. Conference on Parallel Architectures and Compilation Techniques, Sep. 2003.

[19] S. Rusu, H. Muljono, and B. Cherkauer, “Itanium 2 processor 6M: higher frequency and larger L3 cache,” IEEE Micro, pp. 10-18, Vol. 24, Issue 2 , Mar-Apr 2004.

[20] T.J. Slegel, E. Pfeffer, and J.A. MaGee, “The IBM eServer z990 microprocessor,”, IBM Journal of Research and Development, Vol. 48, No. 3/4, pp. 295-310, April 2004.

[21] SPEC CPU2000 Benchmarks, http://www.specbench.org/osg/cpu2000.

[22] Y. Tosaka, S. Satoh, K. Suzuki, T. Suguii, H. Ehara, G. A. Woffinden, and, S.A.Wender, “Impact of Cosmic Ray Neutron Induced Soft Errors on Advanced Submicron CMOS circuits,” Symposium on VLSI Technology, Digest of Technical Papers, pp. 148-149, 1996.

[23] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, “Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor” Proc. of the Intl. Symp. on Computer Architecture (ISCA’04), pp. 264-275, June 2004.

[24] D. Wendell, et. al., “A 4MB On-Chip L2 Cache for a 90nm 1.6GHz 64b SPARC Microprocessor,” Proc. of IEEE Intl. Solid-State Circuits Conference (ISSCC), [Digest of Technical Papers], Feb. 2004.

[25] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Siavasubramaniam, “ICR: In-Cache Replication for Enhancing Data Cache Reliability,” Proc. of the Intl. Conference on Dependable Systems and Networks (DSN), pp. 291-300, June 2003.

[26] J. F. Ziegler, “Terrestrial Cosmic Rays,” IBM Journal of Research and Development, pp. 19-39, Vol.40, No.1, Jan. 1996.