Improving the Reliability of On-chip L2 Cache Using Redundancy

K. Bhattacharya, S. Kim and N. Ranganathan
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
{kbhattac,skim,ranganat}@cse.usf.edu

Abstract

The reliability of large on-chip L2 cache poses a significant challenge due to technology scaling trends. As the minimum feature size continues to decrease, the L2 caches become more vulnerable to multi-bit soft errors. Traditionally, L2 caches have been protected from multi-bit soft errors using techniques like using error detection/correction codes or employing physical interleaving of cache bit lines to convert multi-bit errors into single-bit errors. These methods, however, incur large overheads in area and power. In this work, we investigate several new techniques for reducing multi-bit errors in large L2 caches, in which the multi-bit errors are detected using simple error detection codes and corrected using the data redundancy in the memory hierarchy. Further, we develop a reliability aware replacement policy that dynamically trades performance for reliability whenever the soft-error budget is exceeded. In order to further improve reliability, we propose the duplication of the data values in cache lines by exploiting their small data widths. The proposed techniques were implemented in the Simplescalar framework and validated using the SPEC 2000 integer and floating point benchmarks. The proposed techniques improve the reliability of L2 caches by 40% and 32% on the average, for integer and floating point applications respectively, with little impact on performance and area.

1. Introduction

The trends in technology scaling have helped the design of modern microprocessors for higher performance and low power consumption through the rapid shrinking of the minimum feature size as well as the reduction of supply voltages. Unfortunately, however, these trends make them more susceptible to transient faults [3, 6]. Transient faults occur due to several reasons, such as soft errors, power supply and interconnect noise, and electromagnetic interference. Soft errors occur when the energetic neutrons coming from space or the alpha particles arising out of packaging materials hit the transistors, which could change the states of the memory bits or the outputs of the logic gates. The chip manufacturers typically set budgets on soft error rates (SER) which should be met by the design.

Memory structures like on-chip caches, DRAMs, and register files have been considered as dominant sources of transient errors in computer systems [1, 11, 16]. Spatial multi-bit errors occur when a single particle strike upsets multiple adjacent cells. The rate of spatial multi-bit errors increases accross technology generations as device feature sizes shrink. With higher packing of the cells in the same active area, a single radiation strike can affect multiple cells simulataneously, potentially leading to multi-bit errors. Maiz et al. [10] report that double-bit spatial errors constitute 1% and 2% of all transient errors in SRAMs with 130nm and 90nm technologies, respectively. With further scaling of device geometries, spatial multi-bit errors will become the significant contributor to on-chip SER.

Traditionally, the L2 caches have been protected against soft errors using *Error correction codes* (ECC) codes [1, 11, 16]. Detecting and correcting soft errors using ECC codes, however, incur a large penalty in area. For example, *double error correction and double error detection* (DECDED) codes require 14 bits, for each 64-bit memory word, corresponding to a 22% area overhead. Multi-bit error protection using sophisticated ECC protection will also require more bit lines and wider sense amplifiers thus increasing the cache access latency and power consumption. Spatial multi-bit errors can also be avoided by using layout level techniques like physical interleaving [13]. However, with higher interleaving factors multiple word lines are needed to be driven and data need to re-grouped or routed for read/write operations, thus increasing the cache access latency.

Several schemes have been proposed in the literature to reduce the area overhead associated with protecting memory by ECC codes. In [7], error protection is suggested for frequently accessed cache lines. In [17], the authors de-

1-4244-1258-7/07/$25.00 ©2007 IEEE

scribed the use of a dead block prediction technique to hold the copy of data found in active cache blocks. A larger ECC word can also be used to compensate for the area overhead [15]. However, since the unit of memory read/write is based on word granularity, each memory read/write requires reading several data words to generate SECDED check bits. In [14], the authors have mentioned of using redundancy for area efficient error protection. However, detailed results in the context of multi-bit errors have not been provided.

In this work, we develop several low-cost mechanisms to improve the reliability of the L2 caches against multi-bit errors. Simple error detection codes like Hamming distance or Cyclic Redundancy Codes (CRC) are used to detect multiple-bit errors, and corrected using the redundancy existing in the memory hierarchy. We demonstrate that multi-bit errors in the L2 cache can be corrected by exploiting the redundancy existing between the write-through L1 cache and the L2 cache and the redundancy between the clean data in the L2 cache and the main memory. To mine more redundancy in the memory hierarchy and hence further improve the reliability of the L2 cache, we propose a novel replacement policy biased toward reliability and detect and duplicate small values at word level.

The rest of the paper is organized as follows. In Section 2, we present our proposed schemes to improve the reliability of the L2 cache based on exploiting the redundancy present in the memory hierarchy. In Section 3, we present our schemes to control the redundancy in the memory hierarchy for further improving the reliability. Section 4 details the experimental methodology and illustrates the results. Finally, Section 5 concludes the paper.

2 Redundancy-based Error Protection

In this section, we present two new schemes that exploit the inherent redundancy existing in the memory hierarchy to improve the reliability of L2 cache. In Section 2.1, we present a scheme to exploit the redundancy existing between the write through L1 cache and the L2 cache. In Section 2.2, we describe a scheme to exploit the redundancy between the L2 cache and the main memory to improve the reliability of the L2 cache.

2.1 Exploiting L1/L2 Redundancy

The redundancy inherent in the memory hierarchy of high performance processors can be exploited to improve the reliability of the L2 cache against soft errors. Most commercial processors support a write-through L1 cache with no-write-allocate-policy and a write-back L2 cache with a write buffer in between. As the L1 cache is write-through, the write operations are performed on both the L1 and the L2 cache thus maintaining redundant copies of the data.

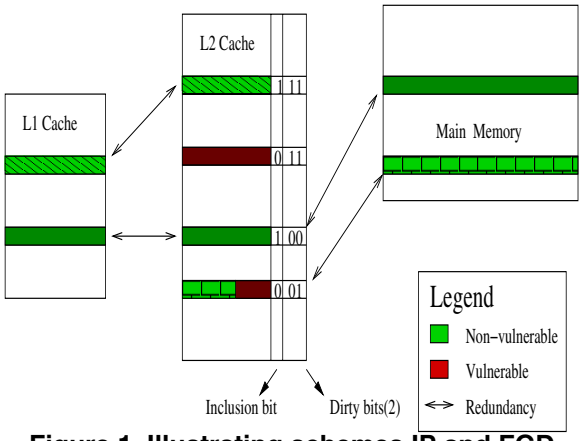

Figure 1. Illustrating schemes IP and FGD

Also, there are many cache lines that reside in both the L1 cache and the L2 cache since they are placed in both of them on L2 cache read misses. We define this implicit redundancy between the L1 and the L2 cache lines as the *inclusion property* (IP) of the L2 cache.

The soft errors become effective when the data items with errors are replaced from the L2 cache and written into the main memory. If the data items are referenced again from the main memory, the errors will affect program output. This however can be avoided as redundant correct data is present in the L1 cache. Thus, when a L2 cache line is replaced, they have to be checked for soft errors. All multi-bit errors can be detected with conventional error detecting codes and corrected by fetching non-corrupt data from the L1 cache.

We assume that the small L1 cache is ECC-protected and thus duplicate data found in L1 cache can be used to correct errors in the L2 cache. In order to support our scheme, an inclusion bit is maintained with each L2 cache line. On a read operation, with a L1 cache miss but a L2 cache hit, the inclusion bit is set to 1 for the corresponding L2 cache block. Also, the L1 cache block that is being replaced due to the miss will cause the corresponding L2 cache block to have no duplicates in the L1 cache. So the inclusion bit of the L2 cache block corresponding to the replaced block from the L1 cache is reset to zero. On a write operation, with a miss on both the L1 cache and the L2 cache, the inclusion bit is reset to zero for the L2 cache block (no-write-allocate policy for L1). The L1 cache line is also invalidated corresponding to the replaced L2 cache block. On a read operation, with a miss on both L1 and L2 cache, the inclusion bit is set to 1 for the new L2 cache line.

2.2 Fine Grain Dirtiness

Errors in the clean L2 cache lines can be corrected by refetching them from the main memory, whereas, the errors

in the dirty cache lines are not correctable. In the standard cache architecture, even when only one word is modified, the dirty bit for the entire cache line containing that word is set to one. Thus, we lose the information that the other words in the cache line are clean. This problem can be alleviated by adding more dirty bits for each cache line. We define this as supporting *fine-grain dirtiness* (FGD) in the L2 cache. FGD can be supported, for example, if one dirty bit is allocated for each memory word. When an error is detected in a clean L2 cache word during a cache read or a cache line replacement, the error can be corrected by re-fetching the word from the main memory. The area overhead is small for FGD, as only one dirty bit for each memory word is maintained. Further, FGD can be supported at the sub-block level for various multiples of the word-size. Thus, one can trade area overhead with vulnerablity of the L2 cache.

Supporting a dirty bit for each memory word is straightforward. On a read miss in the L2 cache, all dirty bits are reset to zero. The dirty bit corresponding to the modified memory word is set to one on a L2 cache write. From CACTI simulation [12], the latency and power overhead due to additional dirty bits is much lower than 1% for a 256KB L2 cache with 32B cache lines.

Figure 1 illustrates a L2 cache, with the cache line size of two words, that utilizes the inclusion property and supports fine-grain dirtiness. A multi-bit error in the right half of the L2 cache line with inclusion bit 0 and dirty bits 01, can be corrected by re-fetching the matching data from the main memory since the left half has not been modified. A multi-bit error in the L2 cache line with inclusion bit 0 and dirty bits 00, will cause no writeback when it is replaced thus correcting the error. All L2 cache lines with their inclusion bits 1 can be recovered from soft errors by re-fetching the corresponding L1 cache lines.

We note that as the L1 cache lines are a small percentage of L2 cache lines, the reliability of the L2 cache does not improve significantly using this scheme. Also correcting a clean cache word by accessing the corresponding memory word can create a performance bottleneck. Therefore, we suggest more aggressive techniques in the next section which combined with the techniques proposed in this section will significantly improve the reliability of the L2 cache.

3 Controlling Redundancy for Reliability

In this section we propose two new schemes to mine/control the additional redundancy in the memory hierarchy. In Section 3.1, we propose a cache line replacement policy biased towards reliability. In Section 3.2, we exploit small data values in cache lines to increase redundancy at the word level further improving reliability of the L2 cache.

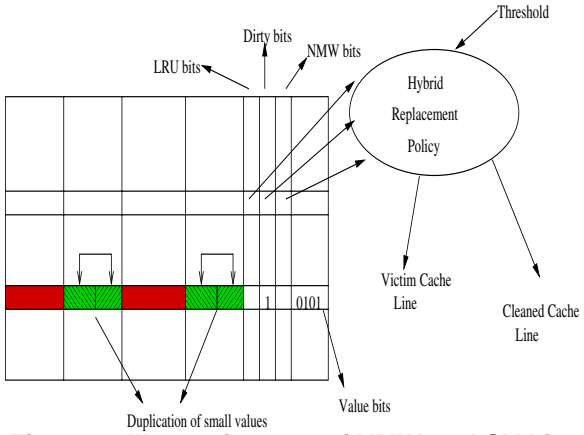

Figure 2. Illustrating use of NMW and SV bits

3.1 Reliability-centric Replacement

The conventional cache line replacement policies are generally based on access history of cache lines such as recency and frequency of cache line accesses. However, the cache line replacement policy can be adapted to improve the reliability of the L2 cache. In addition to recency and frequency information, we can also include dirtiness of the cache blocks in selecting a victim cache line. A hybrid replacement policy has been developed by combining the conventional LRU policy with the dirtiness-based replacement policy. When there is no dirty cache line in the accessed set of the L2 cache line, the LRU cache line is replaced. When the LRU cache line is clean and a next LRU cache line is dirty, the next LRU line is selected as a victim. Only the LRU replacement policy is considered when the number of dirty blocks in the L2 cache is below a *vulnerability threshold*. Performance can be traded for higher reliability of the L2 cache by controlling this threshold.

The hybrid replacement policy is supported by the addition of a bit called "No More Write" (NMW) for each cache line. Generational behavior of cache lines [9], is exploited by using the NMW bit. The NMW bit in a cache line is maintained using the following algorithm. The NMW bit is reset to 0 when an L2 cache line is brought into the L2 cache. When the cache line is written more than one time, its NMW bit is set to 1, indicating that they are likely to be modified soon. NMW bits of L2 cache lines are reset to zero periodically, resembling the popular CLK algorithm implemented to maintain LRU bits. The optimal time interval of periodic clearing of the NMW bit is determined experimentally through simulations. Thus the NMW bits acts as a *1-bit predictor* of whether the cache line will be written soon. The cache lines with their NMW bit set will likely to be written very soon and thus will be vulnerable again if cleaned. Cache lines which are dirty but have their NMW bit 0 are predicted to be in their "dead time" [9] and

can be cleaned to make them non-vulnerable. If the prediction is incorrect, i.e., cache line has not yet reached its dead time but has a NMW bit 0 (and became a candidate for eviction), the cache block will suffer a cache miss, thus causing a performance penalty.

The hybrid replacement policy can be extended to further improve the reliability of the L2 cache. When there are other dirty cache lines in the same set as that of the replaced cache line with their NMW bits set to 0, they can be cleaned together with the victim cache block. Cache lines with their NMW bits 0 are written back together to the main memory since the lines are not likely to be modified soon. As these cache blocks are not evicted, this will not increase the L2 cache miss rate. However, the reliability of the L2 cache is improved as the number of dirty cache lines with no duplicates in the memory hierarchy is reduced. If this *clustered cleaning* of dirty cache lines is accurate the reliability of the L2 cache will be improved and there will be no performance penalty.

3.2 Exploiting Small Data Value Size

It is commonly known that a large percentage of memory values are small in size and value [5, 8]. The small memory values, which use at most half of the memory word, can be duplicated in their upper half of memory word bits, thus increasing the degree of redundancy in the L2 cache at the word level. If the value of the memory word is small, a detected multi-bit error in the lower half bits can be corrected by using the duplicate data found in the top half bits. To implement the duplication of small memory values, each memory word requires a "small value bit" (SV bit) for indicating that the value stored in the word is small and, thus, duplicated in the upper half bits of the memory word. The area overhead for supporting this scheme is small, as only one bit is added for each memory word. Further, area overhead can be traded for vulnerability by maintaining SV bits only for frequently accessed cache lines in a smaller, faster cache.

The tasks of detecting, duplicating, and un-duplicating small memory values in the L2 cache require hardware overhead. Small memory values can be detected by adding zero detectors that can check the upper half bits of memory word. Memory values are duplicated using multiplexers that can select between the lower half bits and the upper half bits of the memory values for the upper half bits of the memory word. Similarly, small memory values can be un-duplicated with multiplexers that can select between zeros and upper half bits of the memory values for the upper half bits of results. The tasks of zero detection, duplication and un-duplication are performed between the L2 cache and the main memory to augment L2 cache line fillings and replacements, and between the L1 data cache and the L2 cache to support write-through requests from the write buffer.

Figure 2 illustrates how redundancy/vulnerability of the L2 cache can be controlled by using a hybrid reliability-centric replacement policy and small value duplication. For simplicity, each cache line is assumed to contain four words and SV bits are supported for each memory word.

4 Experimental Setup and Results

In this section we describe our experimental setup and the simulation results of the various schemes proposed in the paper for improving the reliability of the L2 cache. Table 1 summarizes the various schemes that we have experimented in our simulations.

Table 1. List of proposed schemes

scheme	description
Baseline	conventional L2 cache
I	exploit inclusion property
IM	exploit inclusion property add multiple dirty bits
D	replace a dirty cache line with NMW bit 0
DC	replace a dirty cache line with NMW bit 0 clean dirty cache lines with NMW bit 0 in the same set
IDC-T1	exploit inclusion property replace a dirty cache line with NMW bit 0 clean dirty cache lines with NMW bit 0 in the same set enabled when the L2 cache vulnerability is higher than 25%
IDC-T2	exploit inclusion property replace a dirty cache line with NMW bit 0 clean dirty cache lines with NMW bit 0 in the same set enabled when the L2 cache vulnerability is higher than 10%
IMDC	exploit inclusion property add multiple dirty bits replace a dirty cache line with NMW bit 0 clean dirty cache lines with NMW bit 0 in the same set
IMSDC	exploit inclusion property add multiple dirty bits duplicate small memory values replace a dirty cache line with NMW bit 0 clean dirty cache lines with NMW bit 0 in the same set enabled when the L2 cache vulnerability is higher than 25%

We next describe our experimental setup followed by the simulation results on our various proposed schemes.

4.1 Experimental Setup

We modified the SimpleScalar Version 3 Tool Suite [2] for this study. Since we target high performance embedded processors and/or desktop processors, our baseline processor models an out-of-order four-issue superscalar processor with a split transaction memory bus. Table 2 summarizes the simulation parameters of this processor. Our simulations have been performed with a subset of SPEC2000 benchmarks [4]. These were compiled with DEC C V5.9-008, Compaq C++ V6.2-024, and Compaq FORTRAN V5.3-915 compilers using high optimization level. Eight programs from each of floating-point and integer benchmarks are randomly chosen for our evaluation. All benchmarks are fast-forwarded for one billion instructions to avoid initial start-up effects and then simulated for another

Table 2. Baseline processor configuration

Parameter	Configuration
Issue window	64-entry RUU 32-entry LSQ
decode, issue and commit rate	4 instructions per cycle
Functional units	4 INT add, 1 INT mult/div 1 FP add, 1 FP mult/div
L1 instruction cache	16KB 4-way, 32B line, 1-cycle
L1 data cache	16KB 4-way, 32B line, 1-cycle
L2 cache	unified 256KB, 4-way, 32B line, 10-cycle
Main memory	8B-wide, 100-cycle
Branch prediction	2-level, 2K BTB, 32-entry RAS
Instruction TLB	64-entry, 4-way
Data TLB	128-entry, 4-way

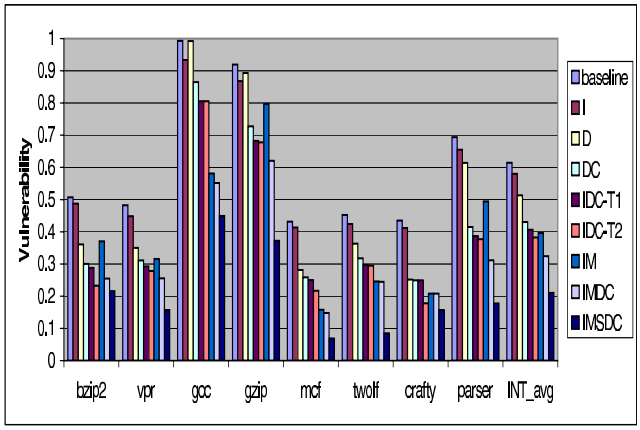

Figure 3. Vulnerability for SPECINT2000.

one billion committed instructions. For all simulations, reference input sets are used.

4.2 Simulation Results

We measure the *vulnerability* of the L2 cache by computing the average number of dirty blocks per cycle without any dupicates in the memory hierarchy. Thus faults are not specifically modelled/injected in our simulation setup. Figures 3 and 4 illustrates the vulnerabilities of the L2 cache for various combination of schemes we have described in Table 1. Vulnerability of the L2 cache for the baseline configuration is 64.6% and 61.4%, on the average, for the floating-point and integer benchmarks, respectively. Schemes **I**, **D**, **DC**, and **IDC-T1**, on the average, reduces vulnerability to 61.4% , 53.9%, 43.4%, and 41%, respectively, for the floating-point benchmarks. These percentages are 58%, 51.3%, 43.1%, and 40.6% for the integer benchmarks. The maximum benefit from scheme **I** is limited to 6.25% since at most 16KB of dirty data can be redundant between the 16KB L1 data cache and the 256KB L2 cache in our baseline processor configuration. Scheme **D** does not show good results when baseline vulnerability is high. This is because, in these benchmarks, most of cache lines are dirty and, thus, there is little difference between our reliability based replacement and the LRU policies. In contrast, since L2

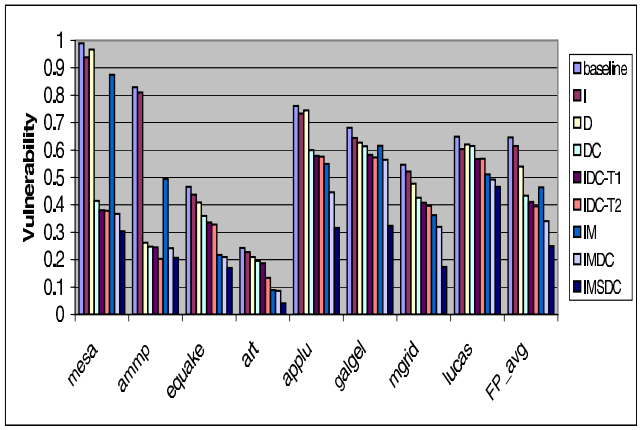

Figure 4. Vulnerability for SPECFP2000.

cache miss rate is very high (28.8%) and, thus, IPC is very low (0.1) in *ammp*, (cache lines remain dirty when pipelines are stalled for a long time due to the L2 cache misses thus increasing vulnerability per cycle), scheme **D** makes those dirty cache lines non-vulnerable by evicting them from the L2 cache, reducing vulnerability per cycle. Scheme **DC** works very well for *mesa* and *parser*, in which scheme **D** was not effective in improving the vulnerability. A vulnerability threshold of 10% implemented in scheme **IDC-T2**, further improves the vulnerability of the L2 cache, by 1% and 2.3% for the floating-point and integer benchmarks, respectively.

Scheme **IM**, which provided a dirty bit for each word in a cache line, reduces the vulnerability of the L2 cache to 43% while scheme **IMDC** further reduces the vulnerability to 33.5% for the integer benchmarks. These percentages are 39.6% and 32.4%, respectively, for the floating-point benchmarks. The combined optimization scheme **IMSDC**, which exploits small memory values as well, reduces L2 cache vulnerability by 40% on the average for the integer benchmarks compared to the baseline configuration. Floating point benchmarks show a lesser decrease in vulnerablity, of about 32%, primarily because the floating point values include a sign bit, exponent and mantissa fields and hence cannot be detected by the small value detector.

As discussed previously the NMW bit provides a 1 bit prediction for whether the cache line will be written soon. We also experimented with 2 bit predictors but we did not find any significant improvements in results from the 1 bit predictor case. We also measured L2 cache miss rate change since our proposed schemes use either the conventional LRU policy or the proposed reliability-centric policy. We noted that using the reliability centric replacement policy does not increase the miss rate significantly compared to the LRU policy. We also noted that the schemes proposed decreases IPC by only 0.1% and 0.3% for the integer and floating point benchmarks, respectively. For brevity, we

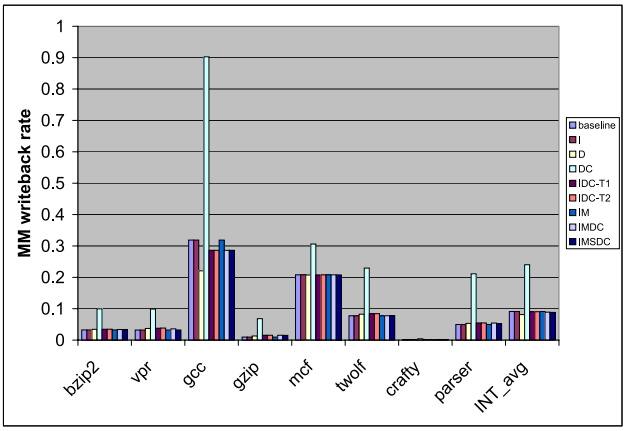

Figure 5. Write back rate for SPECINT2000.

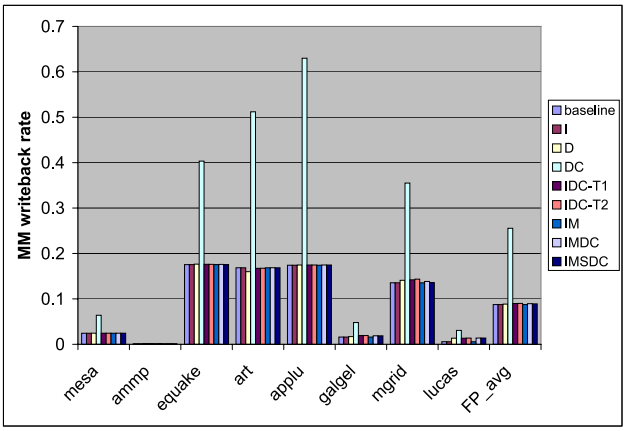

Figure 6. Write back rate for SPECFP2000.

have ommitted illustrating these results here.

Since our replacement policies favor dirty cache lines, we also measured write back traffic rate from the L2 cache to the main memory. Write back traffic rate is measured as the ratio of the number of writes from the L2 cache to all L2 cache accesses. As shown in Figures 5 and 6, Scheme **DC** increases memory write traffic significantly since it performs clustered cleaning of dirty cache lines, however, scheme **IDC-T1** shows little difference in write back traffic. Since redundant cache lines between L1 and L2 caches are most active cache lines, they are likely to be modified frequently. Cleaning these redundant cache lines does not help in reducing vulnerability of the L2 cache

5 Conclusions

Multi-bit soft-errors are increasingly being observed on large L2 caches due to technology scaling and higher device densities. Higher order bit-interleaving have high latency overheads while, powerful multi-bit ECC codes are prohibitive in terms of area. We have proposed several area-efficient schemes that can improve the reliability of the L2 cache especially against spatial multi-bit soft errors. The best combination of our proposed schemes improves the reliability of the L2 cache by 40% and 32%, on the average, for integer and floating point benchmarks, respectively. These reliability improvements of the L2 cache can be accomplished with virtually no performance penalty and with very little area overhead.

References

[1] D. Bossen, J. Tendler, and K. Reick. Power4 system design for high reliability. *IEEE Micro*, 22(2):16–24, 2002.

[2] D. Burger and T. Austin. The simplescalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, 1997.

[3] P. Hazucha and C. Svensson. Impact of cmos technology scaling on the atmospheric neutron soft error rate. *Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[4] http://www.specbench.org/osg/cpu2000/. Spec2000 benchmarks.

[5] J. Hu, S. Wang, and S. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. *Proc. of the Intl. Conf. on Dependable Systems and Networks*, 0:281–290, 2006.

[6] T. Karnik, B. Bloechel, K. Soumyanath, V. De, and S. Borkar. Scaling trends of cosmic ray induced soft errors in static latchesbeyond 0.18 μ. *Digest of the Symp. on VLSI Circuits*, pages 61–62, 2001.

[7] S. Kim and A. Somani. Area efficient architectures for information integrity in cache memories. *Proc. of ISCA*, 1999.

[8] S. Kumar and A. Aggarwal. Reduced resource redundancy for concurrent error detection techniques in high performance microprocessors. *Proc. of HPCA*, 2006.

[9] H. Lee, G. Tyson, and M. Farrens. Eager writeback-a technique for improving bandwidth utilization. *Proc. of the Symp. on Microarchitecture*, pages 11–21, 2000.

[10] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of multi-bit soft error events in advanced srams. *Digest of IEDM*, pages 21–24, 2003.

[11] N. Quach. High availability and reliability in the itanium processor. *IEEE Micro*, 20(5):61–69, 2000.

[12] G. Reinman and N. Jouppi. An integrated cache timing and power model. *Compaq WRL Report*, 1999.

[13] C. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *Trans. on Device and Materials Reliability*, 5(3):397–404, 2005.

[14] V. Sridharan, H. Asadi, M. Tahoori, and D. Kaeli. Reducing data cache susceptibility to soft errors. *Trans. on Dependable and Secure Computing*, 3(4):353–364, 2006.

[15] T. Tanzawa, T. Tanaka, K. Takeuchi, R. Shirota, S. Aritome, H. Watanabe, G. Hemink, K. Shimizu, S. Sato, Y. Takeuchi, et al. A compact on-chip ECC for low cost flash memories. *Journal of Solid-State Circuits*, 32(5):662–669, 1997.

[16] K. Yeager. The mips r10000 superscalar microprocessor. *IEEE Micro*, 16(2):28–41, 1996.

[17] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. Icr: in-cache replication for enhancing data cache reliability. *Proc. of the Conf. on Dependable Systems and Networks*, pages 291–300, 2003.