A Framework for Correction of Multi-Bit Soft Errors in L2 Caches Based on Redundancy

Koustav Bhattacharya, *Student Member, IEEE*, Nagarajan Ranganathan, *Fellow, IEEE*, and Soontae Kim, *Member, IEEE*

***Abstract*—With the continuous decrease in the minimum feature size and increase in the chip density due to technology scaling, on-chip L2 caches are becoming increasingly susceptible to multi-bit soft errors. The increase in multi-bit errors could lead to higher risk of data corruption and potentially result in the crashing of application programs. Traditionally, the L2 caches have been protected from soft errors using techniques such as: 1) error detection/correction codes; 2) physical interleaving of cache bit lines to convert multi-bit errors into single-bit errors; and 3) cache scrubbing. While the first two methods incur large area overheads for multi-bit errors, identifying the time interval for scrubbing could be tricky. In this paper, we investigate in detail the multi-bit soft error rates in large L2 caches and propose a framework of solutions for their correction based on the amount of redundancy present in the memory hierarchy. 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. We also propose several techniques to control/mine the redundancy in the memory hierarchy to further improve the reliability of the L2 cache. The proposed techniques were implemented in the Simplescalar framework and validated using the SPEC 2000 integer and floating point benchmarks for L2 cache vulnerability, global cache miss-rate, average cycle count and main memory write back rate, considering the area and power overheads. Experimental results indicate that the vulnerability of L2 caches can be decreased by 40% on the average for integer benchmarks and 32% on the average for floating point benchmarks, with an average multi-bit error coverage of about 96%, with significantly less area and power overheads and with virtually no performance penalty. The proposed techniques are applicable to both single and multi-core processor-based systems.**

***Index Terms*—Control/mine redundancy, error detection and correction, l2 caches, multi-bit errors, soft errors.**

I. Introduction

THE trends in technology scaling have helped the design of modern microprocessors for higher performance and lower power consumption through the rapid shrinking of the minimum feature size as well as the reduction of supply voltages [5]. At the same time, microprocessors are being built with higher degree of spatial parallelism and deeper pipelines to increase the clock frequency [14]. Unfortunately, however, these trends make them more susceptible to transient faults [10], [13], [16], [18], [31]. 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 soft errors that do not affect the program output are considered benign as no error is observed by the user. This situation can occur, for example, in branch prediction logic or in the instructions from the misspeculated execution sequences which never commit and thus, will never lead to visible error states. Soft errors which affect the program output are typically defined in terms of *failures in time* (FIT) [4], [26]. The chip manufacturers typically set budgets on soft error rates which should be met by the design.

Several different strategies have been investigated in the past to avoid, detect and recover from soft errors [2]. These solutions are applied at various levels of the system, from process technology, circuit to microarchitecture levels. However, solutions are more effective at the architectural level, primarily because a fault transforms into an error depending upon the current architectural state of the processor. Moreover, typical solutions at the architectural level consume much less area than the corresponding solutions in the process technology or circuit levels. Memory structures have been considered as dominant sources of transient errors in computer systems [6], [24], [27], [28], [36]. These include on-chip caches, DRAMs, register files, and other on-chip memory structures. Two competing factors determine the *soft error rate* (SER) of memory, as the device feature size decreases. With the shrinking of device geometries, the critical charge (Q_{crit}) required for the occurrence of soft errors, decreases. However, as the active silicon area of the cells also decrease due to scaling, the probability of radiation strike also decreases. Thus, the SER of caches has remained almost constant due to technology scaling [32].

Although, with technology scaling the SER in SRAMs has remained constant for a given cache size, the rate of multi-bit errors has increased significantly with the shrinking device geometries. 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. This is because due to the higher packing of the cells in the same active area, a single radiation strike can

Manuscript received September 15, 2007; revised December 05, 2007. First published December 09, 2008; current version published January 14, 2009. This research is supported in part by a grant from Semiconductor Research Corporation (SRC) under Contract 2007-HJ-1596 and by National Science Foundation (NSF) Computing Research Infrastructure under Grant CNS-0551621.

K. Bhattacharya and N. Ranganathan are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: kbhhatac@cse.usf.edu; ranganat@cse.usf.edu).

S. Kim is with the School of Engineering at Information and Communication University, Daejeon 305-714, Korea (e-mail: skim@icu.ac.kr).

Digital Object Identifier 10.1109/TVLSI.2008.2003236

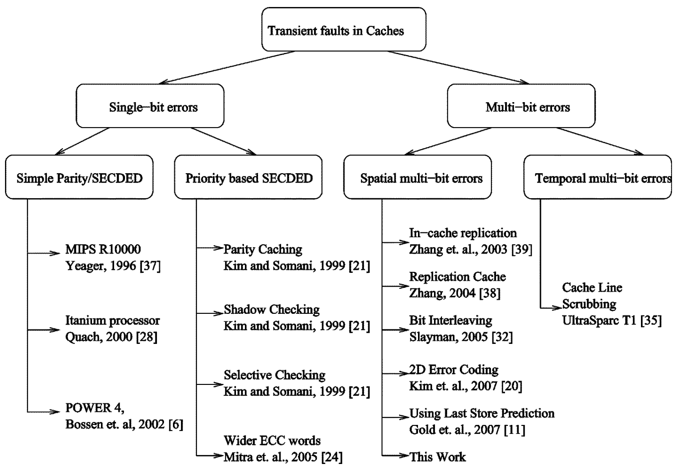
Transient faults in Caches
Single−bit errors
Multi−bit errors
Simple Parity/SECDED
Priority based SECDED
Spatial multi−bit errors
Temporal multi−bit errors
MIPS R10000 Yeager, 1996 [37]
Itanium processor Quach, 2000 [28]
POWER 4, Bossen et. al, 2002 [6]
Parity Caching Kim and Somani, 1999 [21]
Shadow Checking Kim and Somani, 1999 [21]
Selective Checking Kim and Somani, 1999 [21]
Wider ECC words Mitra et. al., 2005 [24]
In−cache replication Zhang et. al., 2003 [39]
Replication Cache Zhang, 2004 [38]
Bit Interleaving Slayman, 2005 [32]
2D Error Coding Kim et. al., 2007 [20]
Using Last Store Prediction Gold et. al., 2007 [11]
This Work
Cache Line Scrubbing UltraSparc T1 [35]

Fig. 1. Taxonomy diagram: Works related to transient faults on caches.

now affect multiple cells simultaneously, potentially leading to multi-bit errors. Maiz *et al.* [23] report that double-bit spatial errors constitute 1% and 2% of all transient errors in SRAMs with 130- and 90-nm technologies, respectively. With further scaling of device geometries, spatial multi-bit errors will become the significant contributor to on-chip SER [24]. Multi-bit errors can also arise in larger device geometries due to multiple single bit errors over time. Thus, temporal multi-bit errors occur when multiple independent particles, affect bits in the same word at different times. Temporal multi-bit errors can occur predominantly in large caches due to the larger mean lifetime of cache lines [35].

As shown in the taxonomy diagram given in Fig. 1, the L2 caches have been traditionally protected against soft errors using *Error correction codes* (ECC) codes [6], [28], [36]. The tasks of 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 [32]. However, with higher interleaving factors multiple word lines are needed to be driven and data need to regrouped or routed for read/write operations, thus increasing the cache access latency. Multi-bit errors can be avoided by correcting single-bit errors during scrubbing, before they develop into temporal multi-bit errors by another particle strike. However, choosing the right scrub interval is often difficult [25]. Most importantly, scrubbing cannot eliminate spatial multi-bit errors since spatial multi-bit errors occur due to a single particle strike rather than evolving over time.

Several schemes have been proposed in the literature to reduce the area overhead associated with protecting memory by ECC codes [39]. In [21], error protection is suggested for frequently accessed cache lines. In [38], Zhang *et al.* described 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 [34]. 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 [37], a small fully associative "replication cache" is maintained to maintain replicas of writes which are used to detect and correct errors. In [33], Sridharan *et al.* have mentioned of using redundancy for area efficient error protection. However, detailed results in the context of multi-bit errors have not been provided. Recently, in [3], [11], and [20], several techniques have been proposed for area efficient multi-bit error correction. In [11], Gold *et al.* have proposed to reduce the multi-bit soft-errors of L1 caches using last store prediction. In [20], Kim *et al.* have proposed the use of 2-D error codes which can correct clustered 32×32 errors with significantly smaller overheads in area, performance and power. In [3], the use of the redundancy present in the memory hierarchy for area-efficient error correction, has been explored.

In this work, we model the vulnerablity of the L2 caches due to multi-bit errors using a probabilistic formulation characterized by extensive simulations for multi-bit errors in various L2 cache organizations. Based on this study, we propose and investigate a framework of solutions based on redundancy for the correction of multi-bit soft errors. In our approach, simple error detection codes like hamming distance or cyclic redundancy codes (CRC) are used to detect the multiple-bit errors, and they are 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 existing between the clean data lines of the L2 cache and the main memory. We found that the bandwidth and power requirement of the write-through L1 cache can be sufficiently reduced by addition of a small merging write buffer between the L1 and L2 cache. We investigate methods to increase the amount of redundancy in the memory hierarchy by employing a redundancy-based replacement policy, the amount of redundancy being controlled is based on a redundancy threshold which is estimated using our probabilistic model. Finally, we investigate how redundancy can be mined at the word level by duplicating small memory values in the upper half of the memory word. Multi-bit errors in the lower half of the word is corrected using the duplicate copy in the upper half. The multi-bit errors which cannot be corrected using the inherent redundancy are corrected by using a small ECC cache.

The rest of this paper is organized as follows. In Section II, we model the vulnerablity of the L2 caches due to multi-bit errors using a probabilistic formulation characterized by extensive simulations for multi-bit errors in various L2 cache organizations. In Section III, we present several schemes to improve vulnerablity of the L2 cache based on exploiting the redundancy present in the memory hierarchy. In Section IV, we present schemes to control the redundancy for reducing the vulnerability of the L2 cache. Section V details the experimental methodology and the results. Section VI compares our redundancy based multi-bit error protection framework with several recent works in literature. Finally, Section VII provides some conclusions.

II. Characterization of Multi-Bit Errors in Conventional Caches

In this section, we provide a characterization of the multi-bit error rate in conventional caches. In particular, we are interested in characterizing how the multi-bit error rate changes with cache size, associativity and cache line size. We assume that error detection codes (EDC) like CRC or Hamming distance are maintained which require much less area overhead than error detection *and* correction codes like ECC. Multi-bit errors in the dirty bit lines of the L2 caches can be detected using these EDC codes. However, unlike clean cache lines, the multi-bit errors in the dirty cache lines cannot be corrected, as no duplicate of the correct data is maintained. We therefore define *the vulnerability of the L2 cache as the percentage of dirty cache lines within a given time interval*. Next, in Section II-A, we model the vulnerablity of the L2 caches due to multi-bit errors using a probabilistic formulation. In Section II-B, we characterize the probabilistic model through extensive simulations for multi-bit errors in various L2 cache organizations.

A. Probabilistic Characterization of Multi-Bit Error Rate

As discussed previously, the vulnerability of the L2 cache is given by the expected number of dirty cache lines in a time interval. The expected number of dirty cache lines (represented as $E(D)$) in a time interval of T, is the joint probability that a block with address X will be written and will not be replaced. This can be represented mathematically as

$$E(D) = N \int_0^T p((\mathrm{Blk} = B) \cap (\mathrm{Wrt}) \cap (\mathrm{Evict})^c)dt \quad (1)$$

where N is the number of blocks in the cache. Let $p(\mathrm{blk} = B)$ represent the probability that a particular block B is accessed, $p(\mathrm{Wrt})$ represent the probability that a write occurs at that block, and $p(Ev)$ represent the probability that the block is evicted during the time period T. Assuming that the events are independent, we obtain from the previous equation

$$E(D) = N \int_0^T p(\mathrm{blk} = B)p(\mathrm{Wrt})[1 - p(Ev)]dt. \quad (2)$$

A block B is evicted from the cache if the same set address as that of block B is generated, a tag match does not occur for none of the blocks in the set and the block B is selected for replacement by the replacement scheme. Representing this mathematically and again assuming independence, we have

$$p(Ev) = p(\mathrm{SetAd} = \mathrm{set}[B])[1 - p(M)]p(\mathrm{blkEv} = B). \quad (3)$$

In the previous equation, $p(\mathrm{SetAd} = \mathrm{set}[B])$ is the probability that a set address is generated that has the same set as block B, $p(M)$ gives the probability that a tag match succeeds, and $p(\mathrm{blkEv} = B)$ is the probability that the block B in that set is selected for replacement by the replacement algorithm. Based on a LRU replacement policy, for example, $p(\mathrm{blkEv} = B)$ gives the probability that the oldest block in the set is B.

Based on the previous equations, we can thus characterize the change in cache vulnerability due to changes in cache size. However, characterizing L2 cache vulnerability directly from the probabilistic model, due to changes in associativity and cache line size is difficult. Therefore, we performed extensive simulations on SPEC2000 benchmarks to characterize L2 cache vulnerability against changes in cache line size and associativity. Based on this study, we estimated the probabilities for our model.

B. Vulnerability of Conventional Cache Organizations

In this subsection, we describe the experiments conducted to study the vulnerability due to multi-bit errors for various L2 cache organizations for estimating the probabilities of the model described in the previous subsection. Fig. 2 shows the results for the SPEC2000 integer benchmarks. We varied cache sizes from 16 to 64 kB and 256 kB and cache line sizes from 16 to 32 bytes and 64 bytes, assuming direct and set-associative mapping. The vulnerabilities of the 16, 64, and 256 kB caches were obtained to be 28%, 37%, and 46%, on the average, respectively. Also, as shown in Fig. 2(d), changing associativity does not affect much the vulnerability. The 2- and 4-way caches show slightly lower vulnerability than the direct-mapped cache. The results for the floating point benchmarks were similar to that of the integer benchmarks as in Fig. 3. The vulnerability is observed to be 39%, 43%, and 49% for the 16, 64, and 256 kB caches, respectively. The floating-point benchmarks show higher vulnerability in small cache configurations than the integer benchmarks. The previous results are used to estimate the probabilities of the model described in the previous subsection.

III. Redundancy-Based Error Protection

In this section, we present two new schemes that can exploit the inherent redundancy existing in the memory hierarchy to improve the vulnerablity of L2 cache. In Section III-A, we present a scheme to exploit the redundancy existing between the write through L1 cache and the L2 cache to reduce the vulnerablity of the L2 cache. In Section III-B, we describe a scheme to exploit the redundancy between the L2 cache and the main memory to reduce the vulnerablity of the L2 cache.

A. Exploiting L1/L2 Redundancy

The redundancy inherent in the memory hierarchy of high performance processors can be exploited to impove the reliability of the L2 cache against soft errors [17]. Most commercial processors support a write-through L1 cache and a write-back L2 cache. We assumed that the L1 cache supports a no-write allocate policy and a merging write buffer exists between the L1 cache and the L2 cache which prevents bandwidth and power bottlenecks for the write-through L1 cache [30]. 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. 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* of the L2 cache.

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 be effective and affect program output. This however can be avoided as redundant correct data is present

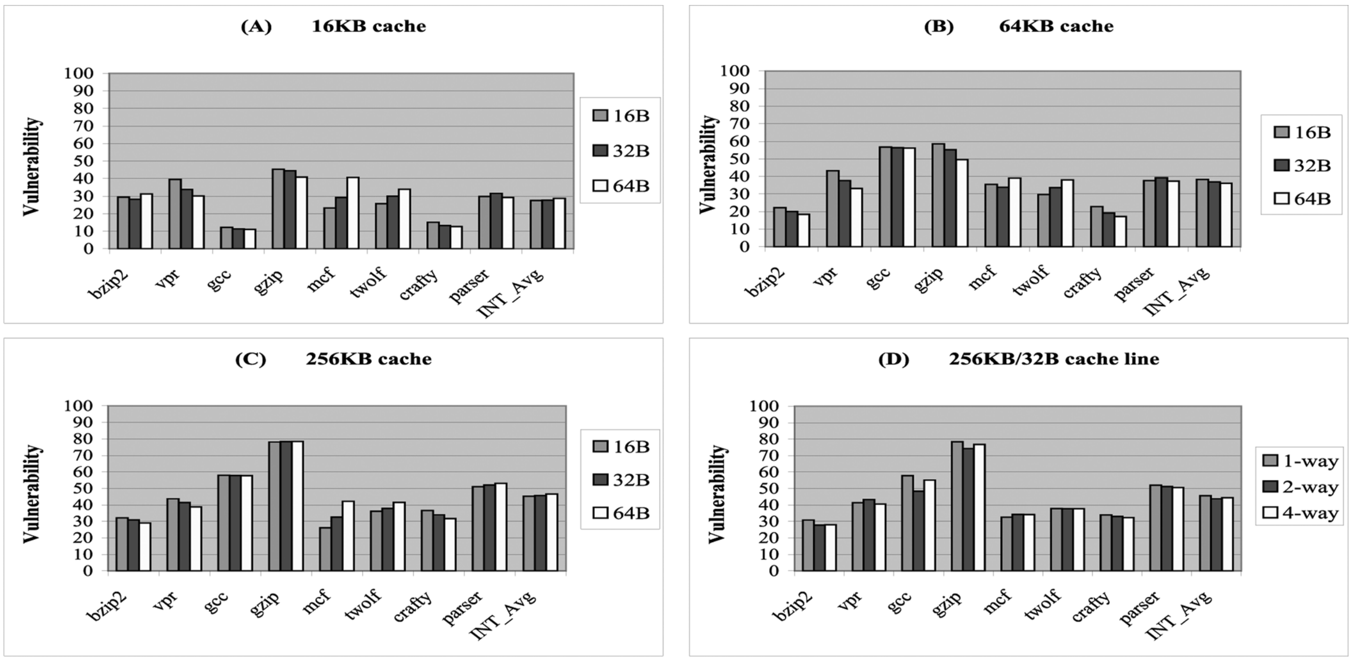

Fig. 2. Vulnerability of different cache organizations for SPECINT2000.

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 using conventional error detecting codes and corrected by fetching non-corrupt data from the L1 cache.

In order to support the previous scheme, an inclusion bit is maintaind 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-alloate 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 the L1 and L2 cache, the inclusion bit is set to 1 for the new cache line.

B. Fine Grain Dirtiness

The redundancy between L2 cache and main memory assumes the form of clean L2 cache lines. Errors in 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. This, however, assumes that whole data in the cache line are modified. 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 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* in the L2 cache. Fine-grain dirtiness can be supported, for example, if one dirty bit can be allocated for each memory word. Only the dirty bit corresponding to modified memory word is set to one and other dirty bits are not affected. 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 refetching the word from the main memory. Thus, we can correct multi-bit soft errors in the L2 cache and improve recover-ability of the L2 cache. Area overhead is small for fine-grain dirtiness: one dirty bit for each memory word, which is the same overhead as parity check code.

Supporting a dirty bit for each memory word does not increase the complexity of the cache hierarchy. 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 [29], the latency and power overhead due to additional dirty bits is much lower than 1% for a 256 kB L2 cache with 32 B cache lines.

Fig. 4 illustrates our memory hierarchy that utilizes inclusion property and supports fine-grain dirtiness. Without loss of generality, the L1 and the L2 cache line sizes have been assumed to be the same. Often, the larger L2 cache line size is assumed to be a multiple of the L1 cache line size as in [29], [31], and [30]. In this case, to support inclusion property, we consider that the L2 cache line is divided into blocks of sizes equal to the L1 cache size and provide multiple inclusion bits for the each of these blocks. As illustrated in the figure, a multi-bit error in the right half of the L2 cache line with inclusion bit 0 and dirty bits 10 can be corrected by refetching the matching data from the main memory since the right half has not been modified. A multi-bit error in the L2 cache line with inclusion bit 1 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 refetching the corresponding L1 cache lines.

Since the L1 cache lines are a small percentage of L2 cache lines, vulnerability of L2 cache does not reduce significantly

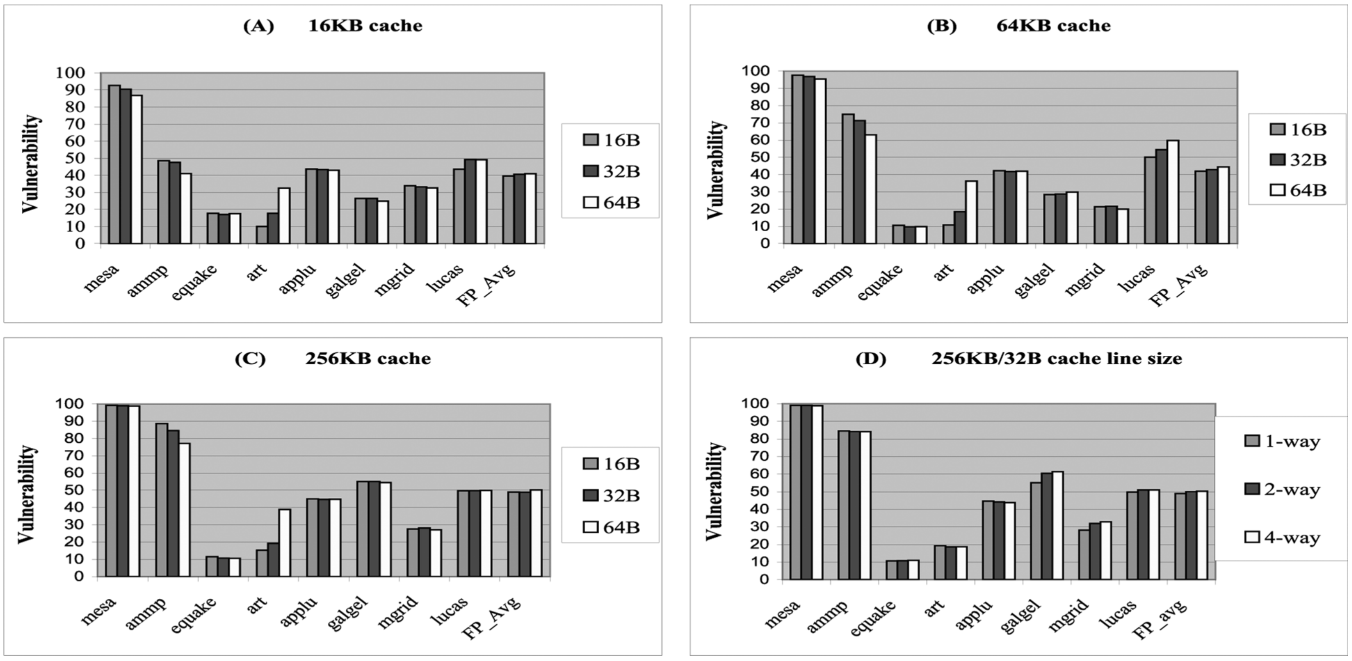

Fig. 3. Vulnerability of different cache organizations for SPECFP2000.

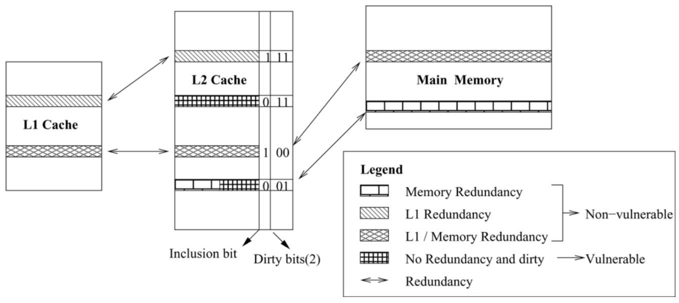

Fig. 4. Illustrating inclusion property and fine grain dirtiness.

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 Sections IV–VI which combined with the techniques already proposed will significantly reduce the vulnerability of the L2 cache.

IV. Improving Reliability by Controlling Redundancy

In this section, we propose two new schemes to mine/control the additional redundancy in the memory hierarchy. In Section IV-A, we propose a cache line replacement policy biased towards reliability. The dirty cache blocks which have no duplicates in the memory hierarchy are selected for replacement on a cache miss, thus implicitly increasing redundancy and improving reliability. In Section IV-B, we exploit small data values in cache lines to increase redundancy at the word level and hence further improve reliability of the L2 cache.

A. *Reliability-Centric Replacement Policy*

The conventional cache line replacement policies aim at improving memory performance by reducing miss rates. They are generally based on access history of cache lines such as recency and frequency of cache line accesses. For example, least recently used (LRU), most frequently used (MFU), and first-in first-out (FIFO) use recency or frequency information. 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 the process of selecting a victim cache line. If a dirty cache line is chosen as a victim, the number of dirty cache lines in the L2 cache per cycle will reduce and, thus, the vulnerability of the L2 cache will reduce. As blind cache line replacements may affect performance adversely, 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*. The estimated number of dirty cache lines, $E(D)$, derived from the probabilistic model discussed in Section II is used to determine the vulnerability threshold V_T as follows:

$$V_T = K_V \frac{E(D)}{N} \tag{4}$$

where K_V is a user-defined constant and N is the total number of blocks in the cache. Thus, the vulnerability threshold depends on the target application workload, which in our case is the SPEC2000 benchmarks, while a user-defined soft-error budget can be specified by controlling K_V. Thus, using the probabilistic model, average number of vulnerable blocks can be estimated based on the cache design parameters and therefore can be used to set the vulnerability threshold appropriately.

The probabilistic formulation decouples vulnerability, which is a characteristic of the application and the cache architecture, from the soft error rate which is characteristic of the environment in which the system is operating. Performance can also be traded for higher reliability of the L2 cache by controlling K_V.

Algorithm 1 The algorithm for L2-cache access for multi-bit soft error protection

```
  if CACHE HIT then
    if $\mathrm{cmd} = \mathrm{WRITE}$ then
      if value generated is small then
        set the corresponding small value bit /* Small Value
        Detection */
      end if
      if matched block in set-address(addr).dirty-bit =
      TRUE then
        set-address(addr).written-bit (NMW bit) = TRUE
      end if
      set-address(addr).dirty-bit = TRUE
    end if
else
    if No. of dirty blocks/Total No. of blocks $< V_T$ then
      /* Use LRU replacement */
    else
      Select a Block for replacement such that
      set-address(addr).dirty-bit = TRUE and
      set-address(addr).written-bit (NMW bit) = FALSE
      and
      set-address(addr).inclusion-bit = FALSE
      If other blocks in this set are found with this property,
      write these to lower level as well /* Clustered Cleaning
      */
    end if
end if
/* Maintain inclusion property */
```

The hybrid replacement policy is supported by the addition of a bit per cache line called "No More Write" (NMW). Generational behavior of cache lines is exploited by using the NMW bit [19], [12]. Generational behavior of cache states that, cache lines are brought in from the main memory on cache misses, used frequently for a short period of time, and, then, not used (dead) until they are evicted by another cache miss. 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. Thus, the NMW bits acts as a *1-bit predictor* of whether the cache line will be written soon. Vulnerable cache lines which are dirty but have their NMW bit 0 are in their dead write time and can be cleaned and made non-vulnerable. The LRU bits along with the NMW bit are used for selecting the victim cache line to be replaced so that the cache line are close to (or already in) their dead time. The cache lines with their NMW bit set will likely to be written onto very soon and thus will be vulnerable again if cleaned. If the prediction is incorrect, i.e., cache line has not yet reached its dead time but has a NMW bit 0 (and becomes 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 by cleaning other dirty cache lines on a replacement. When there are dirty cache lines in the same set as that of the replaced cache line and they are expected not to be modified for a long time, they can be cleaned together with the victim cache block. This will not increase the cache miss rate but can make the L2 cache more immune to errors by reducing the average number of dirty cache lines per cycle. When an L2 cache line is replaced, the other lines in the same set are also checked for their NMWs. The cache lines with their NMW bits set to 0 are written back together to the main memory since the lines are not likely to be modified soon. If this *clustered cleaning* of dirty cache lines is accurate, i.e., the lines will not be modified for a fairly long time and then replaced, the vulnerability of the L2 cache will be reduced and there will be no performance penalty.

B. *Exploiting Small Data Value Size*

It is commonly known that a large percentage of memory values are small [7], [15], [22]. Small memory values use at most half of the memory word bits. These small memory values can be exploited to increase redundancy and improve the reliability of the L2 cache. The small memory values can be duplicated in their upper half of memory word bits, which increases the degree of redundancy in the L2 cache. 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" 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 due to the duplication is the same as that of a parity bit: one bit for each memory word.

The tasks of detecting, duplicating, and unduplicating small memory values in the L2 cache require hardware overhead. Detecting small memory values can be performed by adding zero detectors that can check the upper half bits of memory word. Duplicating memory values can be done with 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 results. Similarly, unduplicating small memory values can be performed with multiplexers that can select between zeros and upper half bits of the memory values for the upper half bits of results. When the duplicate bit from the L2 cache is 1, zero is selected as the output of the multiplexor. A typical hardware architecture for this scheme is shown in Fig. 6. The tasks of zero detection, duplication and unduplication 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. An outline of the cache line access/replacement algorithm to control or mine redundancy presented in this section is provided in Algorithm 1.

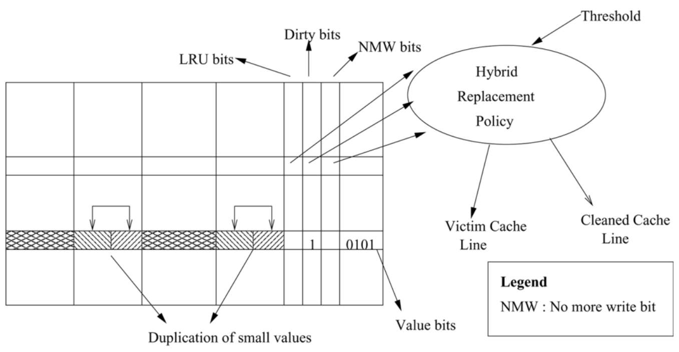

Fig. 5. Illustrating reliability-centric replacement and small value duplication.

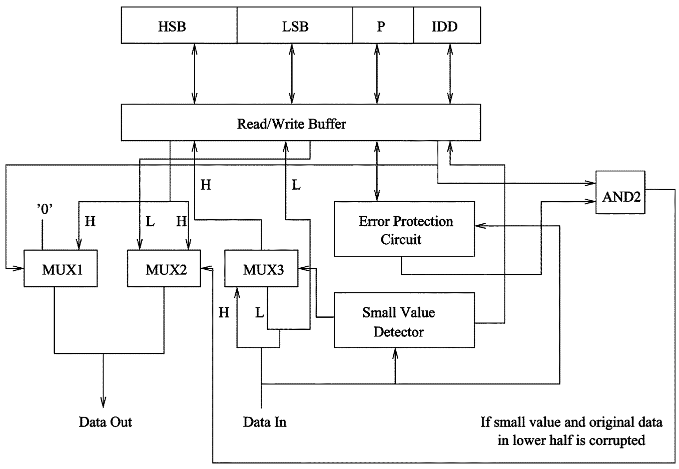

Fig. 6. Hardware architecture for small value detection and duplication.

TABLE I
DESCRIPTION OF THE SCHEMES USED IN EXPERIMENTS

Scheme	**Description**
Baseline	Conventional L2 cache
I	Exploit inclusion property
IM	Add multiple dirty bits
	Exploit inclusion property
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 V_T with V_T=0.25
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 V_T with V_T=0.1
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 V_T with V_T=0.25

V. Experimental Setup and Results

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

A. *Experimental Setup*

We modified the SimpleScalar version 3 tool suite [9] for this study. Since we target high performance embedded processor and/or desktop processors, our baseline processor models an out-of-order four-issue superscalar processor with a split transaction memory bus. Table II summarizes the simulation parameters of this processor. Since SimpleScalar models a write back L1 cache, we modified SimpleScalar to support a write-through L1 cache. We also implemented a merging write buffer with fully associative eight entries between the L1 and L2 cache and each entry of the buffer contained four words. Inclusion property is maintained between L1 and L2 caches. When an L2 cache line with its inclusion bit set to one is replaced, the corresponding cache line in the L1 cache is invalidated to maintain inclusion property. The replacement policy for the L2 cache can be easily extended to implement reliability-centric replacement; we only add an NMW bit for each L2 cache line and a finite state machine for the replacement function is modified to take into account dirtiness, the NMW bit, and the inclusion bit of the cache line. If the number of dirty cache lines is larger than dirtiness threshold, the reliability-centric replacement policy is enabled while the conventional LRU policy is used otherwise. Small values are detected dynamically and maintained using a small value bit. Multiple dirty bits for each cache line are maintained to implement fine grain dirtyness. Our simulations have been performed with a subset of SPEC2000 benchmarks [1]. 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 the benchmarks were fast-forwarded for one billion instructions to avoid initial startup effects and then simulated for another one billion committed instructions. We also simulated for another one billion instructions after fast forwarding 10 billion instructions. For all simulations, the reference input sets were used.

B. *Simulation Results*

We measure the vulnerability of the L2 cache by computing the average number of dirty blocks per cycle without any duplicates in the memory hierarchy. Figs. 7 and 8 present vulnerabilities of the L2 cache for various schemes we have proposed in Table I including the baseline cache. 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. The *mesa*, *gcc*, and *gzip* benchmarks show higher than 90% vulnerability. Scheme **I** reduces vulnerability to 61.4% and 58%, on the average, for the floating-point and integer benchmarks, respectively. These percentages are 53.9%, 43.4%, 41%, and 39.5%, on the average, for schemes **D**, **DC**, **IDC-T1**, and **IDC-T2**, respectively, for the floating-point benchmarks. These

TABLE II
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

TABLE III
COMPARISON WITH RECENT WORKS IN LITERATURE

Scheme	Area overhead	Performance Penalty	Error Coverage
ICR [38]	< 1%	3.6%	65%
Shadow Cache [21]	28%	40%	85%
Last Store Prediction [11]	7.98%	15%	88%
R-cache [37]	10%	7.31%	100%
PERC [30]	19%	< 1%	100%
This work	6%	< 1%	96%

percentages are 51.3%, 43.1%, 40.6%, and 38.3% for the integer benchmarks. The maximum benefit from scheme **I** is limited to 6.25% since at most 16 kB of dirty data can be redundant between the 16 kB L1 data cache and the 256 kB L2 cache in our baseline processor configuration. Scheme **D** does not show good results when baseline vulnerability is high. For example, in *mesa*, *applu*, *gcc*, and *gzip*, scheme **D** shows small reductions in vulnerability. 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, scheme **D** works well with *ammp*; vulnerability of the L2 cache reduces to 26.2% from 82.9% in the baseline. Since L2 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, 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** consistently reduces vulnerability by 10.5% and 8.3%, on the average, over and above scheme **D**. Scheme **DC** works very well for *mesa* and *parser*, in which scheme **D** was not effective in reducing the vulnerability. Scheme **IDC-T1** reduces additional 2.4% and 2.5% of vulnerability for the floating-point and integer benchmarks, respectively. A vulnerability threshold of 10% further reduces the vulnerability of the L2 cache. The vulnerability reduces by 1% and 2.3% for the floating-point and integer benchmarks, respectively. The *ammp*, *bzip2*, and *crafty* benchmarks benefit most from 10% threshold.

The fine grain dirtiness based method was implemented by having four dirty bits per cache line for a cache line size of four words. Scheme **IM** reduces the vulnerability of the L2 cache to 43% and 39.6%, on the average, for the floating-point and integer benchmarks, respectively. Reductions in vulnerability are

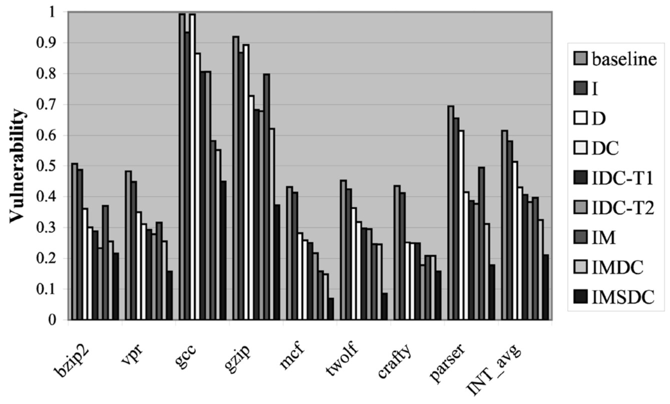

Fig. 7. Vulnerability of the L2 cache for various schemes proposed in this paper for SPECINT2000.

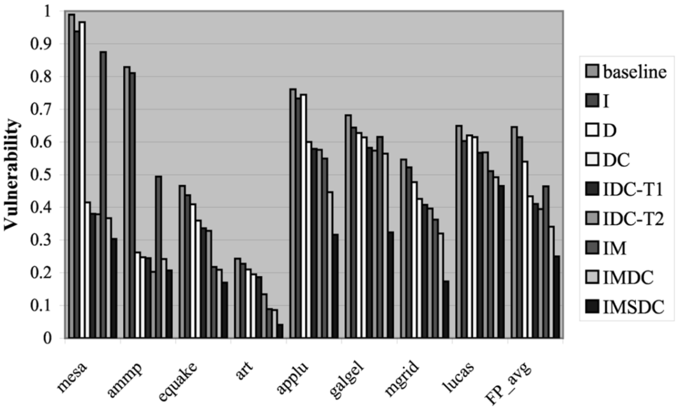

Fig. 8. Vulnerability of the L2 cache for various schemes proposed in this paper for SPECFP2000.

18% and 21%, respectively, compared to **Baseline**. Scheme **IM** is comparable to scheme **IDC-T1** in reducing vulnerability as can be observed in Figs. 7 and 8. In most floating-point benchmarks, scheme **IM** shows better results than scheme **IDC-T1**. Only *mesa* and *galgel* show worse results with scheme **IM**. Half of the integer benchmarks show better results and the other half show worse results with scheme **IM**. Scheme **IMDC** further reduces the vulnerability to 33.5% and 32.4%, on the average, for the floating-point and integer benchmarks, respectively. The *applu*, *mgrid*, *gzip*, and *parser* benchmarks show large reductions in vulnerability with scheme **IMDC** compared to scheme **IM**. As shown in the figures, we have also experimented with our proposed scheme to exploit small memory values. Our combined optimization scheme **IMSDC** reduces L2 cache vulnerability by 40% on the average for the integer benchmarks. Floating point benchmarks show a lesser decrease in vulnerability, 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 later, with the significant reduction of the vulnerability by exploiting/mining redundancy and with the addition of a small direct mapped ECC cache, for the error-correction of the vulnerable blocks, an average multi-bit error coverage of about 96% can be achieved with our approach.

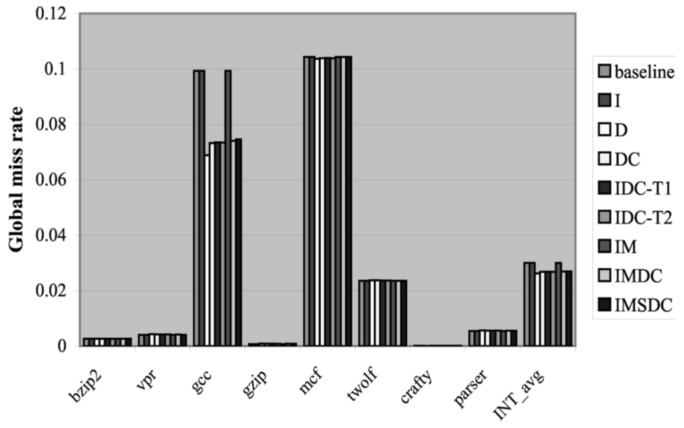

Fig. 9. Global miss rates of the L2 cache for various schemes proposed in this paper for SPECINT2000.

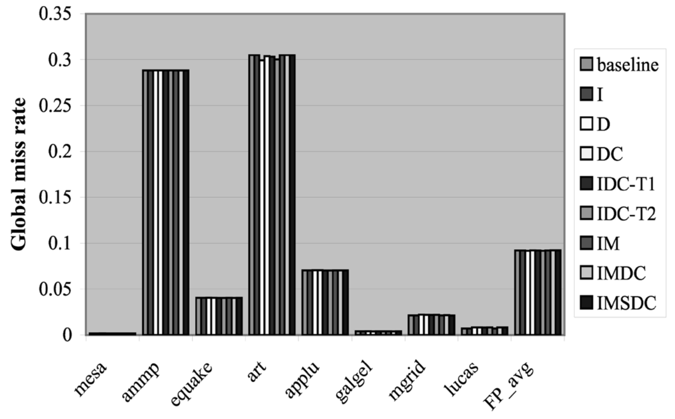

Fig. 10. Global miss rates of the L2 cache for various schemes proposed in this paper for SPECFP2000.

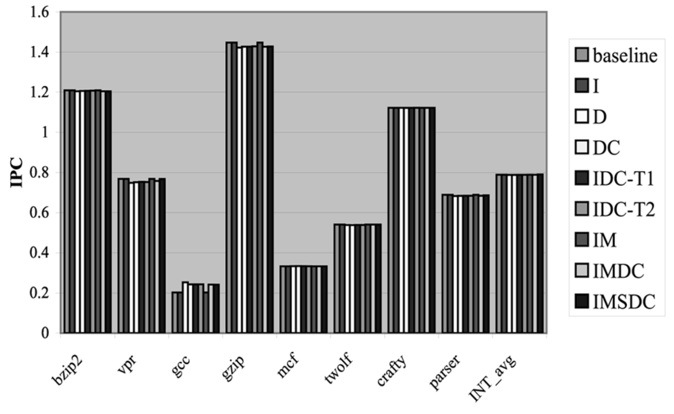

Fig. 11. IPCs for various schemes proposed in this paper for SPECINT2000.

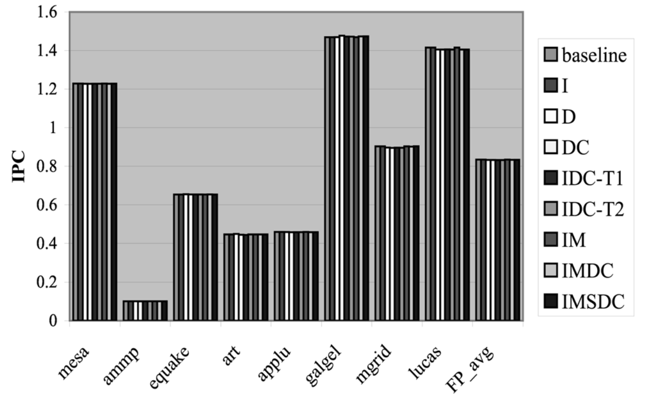

Fig. 12. IPCs for various schemes proposed in this paper for SPECFP2000.

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 notice any significant changes from the 1 bit predictor case. We do not show these results here for brevity. We also measured L2 cache miss rate change since our proposed schemes use either the conventional LRU policy or the proposed reliability-centric policy depending on vulnerability of the L2 cache at a particular time and the chosen vulnerability threshold. Figs. 9 and 10 present L2 cache miss rate for various schemes proposed in this paper. We use global cache miss rate in the figure. Cache miss rates increase by 0.4%, 0.1%, 0.1%, and 0.4%, on the average, for schemes **D**, **DC**, **IDC-T1**, and **IDC-T2**, respectively, for the floating-point benchmarks. These percentages are 12.6%, 10.7%, 10.7%, and 10.7% for the integer benchmarks. The *gcc* benchmark shows a decrease in miss rate, which demonstrates that the conventional LRU policy is not optimal for all benchmarks. As shown in the figure, the miss-rates reduces significantly when the replacement policy is changed from LRU replacement policy to a replacement policy favoring replacement of dirty lines. We note that replacement scheme based on LRU policy is based on the approximation that the least recently used block will not be used in the near future. As we also select that dirty cache line in the set to replace which is oldest in terms of LRU, our simulations show that the replacement scheme using such a technique predicts cache lines in their dead time very accurately and hence has a significantly lower miss rate compared LRU.

Figs. 11 and 12 plots IPC results for various schemes proposed in this paper. IPC reductions are 0.2%, 0.2%, 0.3%, and 0.3%, on the average, for schemes **D**, **DC**, **IDC-T1**, and **IDC-T2**, respectively, for the floating-point benchmarks. These percentages are 0.1%, 0.1%, 0.1%, and 0.1% for the integer benchmarks. IPC reduces slightly due to additional write back traffic in our schemes. The *gcc* benchmark shows IPC increase of 25% for scheme **D**. The benchmark showed high miss rate reduction in Fig. 9 which translated directly into improved performance. The other benchmarks show slight decreases or increases in IPC. Our proposed scheme, especially **IDC-T1**, reduces vulnerability by 23.6%, on the average, for the floating-point benchmarks with 0.3% performance penalty. For the integer benchmarks, vulnerability reduces by 23.1%, on the average, with less than 0.1% performance loss.

Since our replacement policies favor dirty cache lines, we also measured the write back traffic rate from the L2 cache to the main memory as shown in Figs. 13 and 14. The write back traffic rate is measured as the ratio of the number of writes from the L2 cache to all L2 cache accesses. The write back traffic is increased by 1.1% and 191.7%, 2.5%, and 2.8%, on the average, for schemes **D**, **DC**, **IDC-T1**, **IDC-T2**, respectively, the floating-point benchmarks. These percentages are −10.8%, 163.3%, 0.9%, and 0.8% for the integer benchmarks.

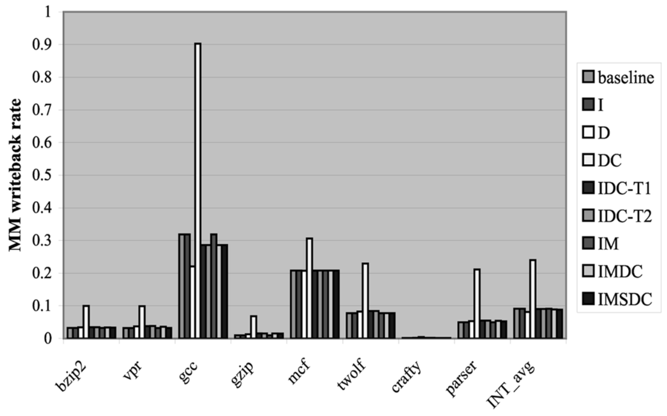

Fig. 13. Write back traffic rate to the main memory for various schemes proposed in this paper for SPECINT2000.

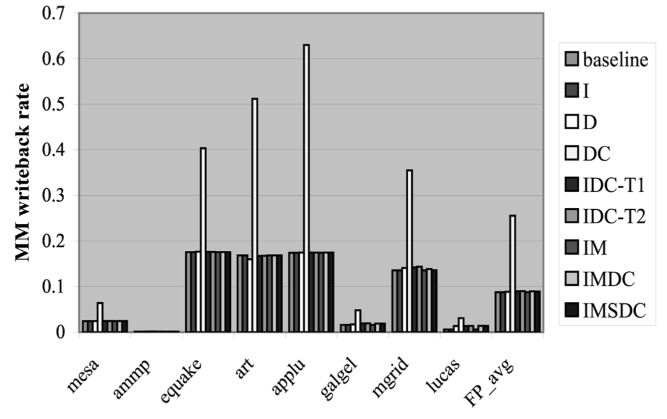

Fig. 14. Write back traffic rate to the main memory for various schemes proposed in this paper for SPECFP2000.

Scheme **DC** increases memory write traffic significantly since it performs clustered cleaning of dirty cache lines. In contrast, scheme **IDC-T1** shows little difference in write back traffic since it takes inclusion bits into account. 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 reduce vulnerable cache lines in scheme **DC**. Scheme **IDC-T1** does not clean redundant cache lines since they are highly likely to be written soon. Schemes **D** and **IDC-T1** even decrease memory write back traffic for the integer benchmarks mainly because of *gcc*, where cache miss rate decreases significantly for schemes **D** and **IDC-T1**, which reduces dirty write backs from the L2 cache. **IDC-T2** shows a similar behavior to **IDC-T1**.

As previously discussed, we assumed that a small ECC cache is maintained for error correction of the vulnerable cache blocks, i.e., those dirty cache blocks that have no duplicates in the memory hierarchy. The multi-bit error correction codes for only the vulnerable blocks are maintained in this small ECC cache. A multi-bit soft error is always detected by the low cost error detection codes. If a L2 cache block is non-vulnerable, it is corrected by exploiting the redundancy existing in the memory hierarchy, while vulnerable blocks are corrected using the small ECC cache. The significant reduction in vulnerability of the L2 cache by exploiting/mining the redundancy in the memory hierarchy allowed a ECC cache of significantly smaller size. We found that a direct-mapped ECC cache, of size 8 kB was sufficient for up to 6-bit error protection using our redundancy based approach for most SPEC2000 benchmarks on 256 kB L2 cache. Our simulations suggest that with a ECC cache of size 4% of that of the L2 cache together with exploiting our redundancy based approach, can provide a multi-bit error error coverage of about 96% with significantly less area/power overhead and with marginal performance penalty.

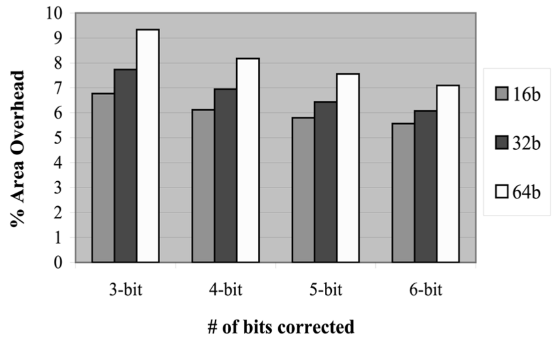

Fig. 15. Area overhead for a L2 cache with redundancy based error protection compared to a baseline L2 cache with no error protection.

We estimated the area overhead of our redundancy based scheme with a small ECC cache for multi-bit error protection of the vulnerable blocks. We estimated the area overhead of multi-bit error correction coding by using the following formulation. As codes obtained by multi-bit errors from a valid codeword must be disjoint from each other for correction to a distinct valid code, we have for p-bit error correction scheme for a m-bit word requiring r check bits

$$(N+1)\star 2^m \leq 2^{m+r} \tag{5}$$

where N is number of possible ways a p-bit error can happen on a m-bit word. Since this is the same as the number of ways of choosing $1, 2, \ldots, p$ objects from $m+r$ objects

$$N = C_1^{m+r} + C_2^{m+r} + \cdots C_p^{m+r}. \tag{6}$$

Solving these equations for r, gives us an estimate of area overhead for complete multi-bit error protection. The area overhead for our redundancy-based multi-bit error correction approach was estimated by considering the area overhead for the small ECC cache and adding the number of status bits (inclusion bit, small value bits, etc.) required for implementing our redundancy-based approach. The area overhead of our redundancy-based multi-bit error protection for fixed number of ECC cache blocks is shown in Fig. 15. As shown in the figure, our redundancy based multi-bit error protection for 6-bit errors on a L2 cache with line size of 32 B incurs only a marginal area overhead of 6%.

We also estimated the power overhead of our redundancy based multi-bit error protection. We ported the Simplescalar-based framework implementing our approach to the Wattch 2.0 [8] framework. Wattch performs architectural level power analysis for the cache by maintaining counters for number of read/write accesses to the cache and multiplying it with the average

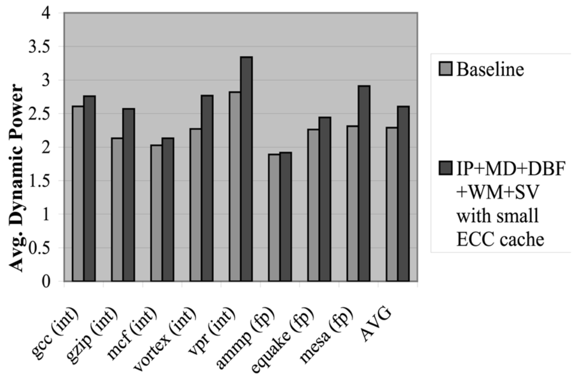

Fig. 16. Average dynamic power consumption for a L2 cache with a 8 kB ECC cache compared with baseline L2 cache with no error protection.

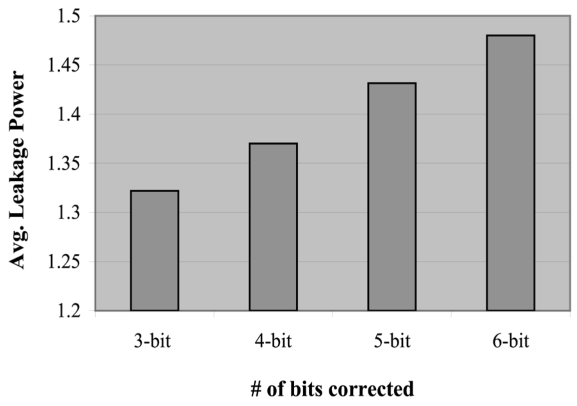

Fig. 17. Average leakage power consumption for a L2 cache with small ECC cache with fixed number of blocks for different sized multi-bit errors.

power required for a single read/write cache access in a particular process technology. We also estimated the leakage power at the architectural level, which is a significant portion of total power in current technology nodes. We used CACTI 4.2 [29] for this analysis, which is a detailed cache access and power analysis tool. The cache size estimates made previously was provided to CACTI to obtain estimates of leakage power. With a 70 nm process technology model, the dynamic power overhead of our redundancy based multi-bit error correction scheme with different SPEC2000 benchmark circuits is plotted in Fig. 16. As shown in the figure, our scheme increases the dynamic power overhead by only about 13.75%. As shown in Fig. 17, marginal overhead is also incurred in leakage power for our area efficient multi-bit error correction scheme for different sized multi-bit errors. This, thus makes the total power overhead of our approach much smaller than that of most works found in literature for moderately sized multi-bit errors.

VI. Comparison With Related Works

We note that many competing solutions have been proposed in the literature for protecting caches against multi-bit errors with low area/performance overhead. For comparison of our work with recent works, we have used data reported in the results of the corresponding research papers and interpolated the results according to our simulation setup. We have assumed an average IPC of 2.5 and that the instruction mix contains 30% memory reference instructions and 10% stores as is typical of most SPEC benchmarks [1].

"In-Cache Replication (ICR)," has been proposed in [38] to exploit "dead" blocks in the data cache to store the replicas of the "hot" blocks. These duplicate blocks can be used to correct multi-bit errors in the active blocks. Although an area overhead of less than 1% has been reported with a modest performance penalty of 3.6%, the parity based multi-bit error protection scheme provides an error coverage of only 65%. Our redundancy based approach has a error coverage of 96% with a performance penalty of less than 1%. "Shadow Caching" [21], maintains copies of most frequently used (MFU) cache lines in separate shadow caches. In the context of error correction, at least two shadow caches should be maintained to support correction using majority voting. The approach although significantly better than blind NMR (N-modular redundancy), however, incurs significant area overhead. For example a 4-way associative shadow cache of 128 entries has an area overhead of about 28%. Also, as multiple copies of data are read for error detection the cache access latency is increased, resulting in a performance overhead of about 40% with a modest error coverage of about 85%. These overheads scale exponentially as higher order multiple-bit errors are considered. In comparison, our redundancy based approach can achieve 96% error coverage with about 6% area overhead with very little performance penalty.

The "R-cache" approach [37], maintains a small fully associative "replication cache" to detect and correct multi-bit errors using copies of dirty data. The method achieves 100% multi-bit error coverage. However, as multi-bit error detection is achieved by parallel access of the R-cache and the data cache, a large latency overhead is incurred. For example, with a 2 cycle load latency for reads as reported in the work, a performance penalty of about 7.31% can be incurred. As illustrated in Section V, the performance penalty for our redundancy based multi-bit error protection scheme is less than 1% with high multi-bit error coverage. The "Last Store" prediction technique [11], proposes the use of an accurate program-counter (PC) trace-based predictor which immediately initiates a writeback after observing a PC trace with a sequence of store instructions. However, the hardware structures like "history table" and "signature table" incur an area overhead of about 8%. Assuming that updating these hardware structures takes at least 1 cycle latency during stores, a performance penalty of about 15% can be incurred which is quite high compared to our redundancy based approach. Our approach also achieves a higher error coverage than that reported in their work. "Punctured Error Recovery Cache (PERC)" [30], decouples error detection and correction by maintaining the "punctured" error correction codes in a separate cache. As error detection and correction is separated, little performance overhead is incurred. However, as the number of vulnerable blocks is not actively reduced, complete multi-bit error coverage requires about 19% area overhead.

We note that the techniques proposed in this work, although primarily targeted at single core processors, however can be extended and applied to multi-core processors. The bandwidth required by the write-through L1 cache used in our approach can be significantly reduced by employing a merging write-buffer between the L1 and the L2 caches. For example, when a fully associative merging write buffer with eight entries and with each entry of the size of four words is placed between the L1 and the L2 cache, a negligible increase of write-through bandwidth is observed. Also, the techniques to mine redundancy using small values and reliability centric replacement can be applied to a cache hierarchy with non-inclusive L1 caches. In multi-core systems, the cache coherence protocol between the local L1 caches and a shared L2 cache, which also acts as a synchronization point, can lead to increased exploitation of the inclusion property between the L1 and the L2 cache. A cache read of data in the local L1 caches of a processor that has been modified by another processor in a multi-core environment will lead to invalidation requests by the cache controller and re-fetching of the modified data from the shared L2 cache. As inclusion property is naturally enforced between the L1 and the L2 caches, this leads to increased redundancy between local L1 caches and shared L2 caches.

VII. Conclusion

Soft errors are increasing in the L2 cache with technology scaling and increasing integration. Multi-bit soft-errors are increasingly being observed on large L2 caches. Higher order bit-interleaving have high latency overheads while, powerful multi-bit ECC codes are prohibitive in terms of area. Scrubbing reduces temporal multi-bit errors but deciding the scrub interval is tricky. Moreover, scrubbing does not correct spatial multi-bit errors. We have proposed several cost-effective schemes that can improve the reliability of the L2 cache especially against spatial multi-bit soft errors. The best combination of proposed schemes shows improvement in reliability by up to 40% for integer benchmarks and 32% for floating point benchmarks. With the significant reduction of the vulnerability by exploiting/mining redundancy and with the addition of a small direct mapped ECC cache, for the error-correction of vulnerable blocks, an average multi-bit error coverage of about 96% can be achieved. These reliability improvements of the L2 cache can be accomplished with significantly less area and power overheads and with virtually no performance penalty.

References

[1] SPEC, " CPU 2000 Benchmarks," 2000. [Online]. Available: http://www.specbench.org/cpu2000/

[2] H. Asadi, V. Sridharan, M. Tahoori, and D. Kaeli, "Balancing performance and reliablity in the memory hierarchy," in *Proc. Symp. Perform. Anal. Syst. Softw.*, 2005, pp. 269–279.

[3] K. Bhattacharya, S. Kim, and N. Ranganathan, "Improving the reliability of on-chip l2 cache using redundancy," in *Proc. ICCD*, 2007, pp. 224–229.

[4] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proc. ISCA*, 2005, pp. 532–543.

[5] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Apr. 1999.

[6] D. Bossen, J. Tendler, and K. Reick, "Power4 system design for high reliability," *IEEE Micro*, vol. 22, no. 2, pp. 16–24, Feb. 2002.

[7] D. Brooks and M. Martonosi, "Dynamically exploiting narrow width operands to improve processor power and performance," in *Proc. HPCA*, 1999, pp. 13–22.

[8] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. ISCA*, 2000, pp. 83–94.

[9] D. Burger and T. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH Comput. Arch. News*, vol. 25, no. 3, pp. 13–25, 1997.

[10] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, Apr. 2003.

[11] B. Gold, M. Ferdman, B. Falsafi, and K. Mai, "Mitigating multi-bit soft errors in L1 caches using last-store prediction," presented at the Federated Comput. Res. Conf., San Diego, CA, 2007.

[12] S. Gopal, T. Vijaykumar, J. Smith, and G. Sohi, "Speculative versioning cache," in *Proc. ISCA*, 1998, pp. 195–205.

[13] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2586–2594, Dec. 2000.

[14] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The microarchitecture of the pentium 4 processor," *Intel Technol. J.*, vol. 1, pp. 1–13, 2001.

[15] J. Hu, S. Wang, and S. Ziavras, "In-register duplication: Exploiting narrow-width value for improving register file reliability," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2006, pp. 281–290.

[16] N. Jha and S. Kundu*, Testing and Reliable Design of CMOS circuits*. Norwell, MA: Kluwer, 1990.

[17] M. Kadiyala and L. Bhuyan, "A dynamic cache sub-block design to reduce false sharing," in *Proc. ICCD*, 1995, pp. 313–318.

[18] 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," in *Dig. Symp. VLSI Circuits*, 2001, pp. 61–62.

[19] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. ISCA*, 1930, pp. 240–251.

[20] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proc. IEEE Micro*, 2007, pp. 197–209.

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

[22] H. Lee, G. Tyson, and M. Farrens, "Eager writeback-a technique for improving bandwidth utilization," in *Proc. Symp. Microarch.*, 2000, pp. 11–21.

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

[24] S. Mitra, N. Kee, and S. Kim, "Robust system design with built-in soft-error resilience," *IEEE Computer*, vol. 38, no. 2, pp. 43–52, Feb. 2005.

[25] S. Mukherjee, J. Emer, T. Fossum, and S. Reinhardt, "Cache scrubbing in microprocessors: Myth or necessity?," in *Proc. Int. Symp. Dependable Comput.*, 2004, pp. 37–42.

[26] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," *IEEE Computer*, vol. 39, no. 1, pp. 118–120, Jan. 2006.

[27] R. Phelan, "Addressing soft errors in arm core-based SOC," ARM Ltd., White Paper, Dec. 2003.

[28] N. Quach, "High availability and reliability in the itanium processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, May 2000.

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

[30] N. Sadler and D. Sorin, "Choosing an error protection scheme for a microprocessor's L1 data cache," in *Proc. ICCD*, 2006, pp. 499–505.

[31] N. Seifert, D. Moyer, N. Leland, and R. Hokinson, "Historical trend in alpha-particle induced soft error rates of the alpha microprocessor," in *Proc. Int. Reliab. Phys. Symp.*, 2001, pp. 259–265.

[32] 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.

[33] V. Sridharan, H. Asadi, M. Tahoori, and D. Kaeli, "Reducing data cache susceptibility to soft errors," *Trans. Depend. Secure Comput.*, vol. 3, no. 4, pp. 353–364, 2006.

[34] T. Tanzawa, T. Tanaka, K. Takeuchi, R. Shirota, S. Aritome, H. Watanabe, G. Hemink, K. Shimizu, S. Sato, Y. Takeuchi, and K. Ohuchi, "A compact on-chip ECC for low cost flash memories," *IEEE J. Solid-State Circuits*, vol. 32, no. 5, pp. 662–669, May 1997.

[35] Sun, "UltraSparc T1," 2005. [Online]. Available: http://www.sun.com/processors/whitepapers

[36] K. Yeager, "The mips r10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–41, Feb. 1996.

[37] W. Zhang, "Enhancing data cache reliability by the addition of a small fully-associative replication cache," in *Proc. Int. Conf. Supercomput.*, 2004, pp. 12–19.

[38] 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.*, 2003, pp. 291–300.

[39] W. Zhang, M. Kandemir, A. Sivasubramaniam, and M. Irwin, "Performance, energy, and reliability tradeoffs in replicating hot cache lines," in *Proc. Int. Conf. Compilers, Arch. Synthesis Embedded Syst.*, 2003, pp. 309–317.

Koustav Bhattacharya (S'06) received the B.Tech. degree in computer engineering from Kalyani University, West Bengal, India, in 2002, and the Master's degree in computer technology from the Indian Institute of Technology, Delhi, India, in 2004. He is currently pursuing the Ph.D. degree in computer science and engineering from the University of South Florida, Tampa.

In 2004, he worked as a Design Engineer with ST Microelectronics, Noida, India. His research interests include design and optimization for reliable and secure VLSI systems, VLSI design automation, and computer architecture.

Mr. Bhattacharya was a recipient of the Richard E. Merwin Scholarship in 2007.

Nagarajan Ranganathan (S'81–M'88–SM'92–F'02) received the B.E. (honors) degree in electrical and electronics engineering from Regional Engineering College, Tiruchirapalli, University of Madras, Madras, India, in 1983, and the Ph.D. degree in computer science from the University of Central Florida, Orlando, in 1988.

He is a Distinguished University Professor with the Department of Computer Science and Engineering, University of South Florida, Tampa. During 1998–1999, he was a Professor of electrical and computer engineering with the University of Texas, El Paso. His research interests include VLSI circuit and system design, VLSI design automation, multi-metric optimization in hardware and software systems, biomedical information processing, computer architecture, and parallel computing. He has developed many special purpose VLSI circuits and systems for computer vision, image and video processing, pattern recognition, data compression and signal processing applications. He has coauthored over 220 papers in refereed journals and conferences, 4 book chapters, and has received 6 U.S. patents with one pending. He has edited 3 books titled *VLSI Algorithms and Architectures: Fundamentals* (IEEE Press, 1999), *VLSI Algorithms and Architectures: Advanced Concepts* (IEEE, 1993) and *VLSI for Pattern Recognition and Artificial Intelligence, World Scientific Publishers* (World Scientific, 1995), and coauthored a book titled *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits* (Springer, 2008).

Dr. Ranganathan was elected as a Fellow of IEEE in 2002 for his contributions to algorithms and architectures for VLSI systems. He is a member of the IEEE, IEEE Computer Society, IEEE Circuits and Systems Society, and the VLSI Society of India. He has served on the editorial boards for the journals: *Pattern Recognition* (1993–1997), *VLSI Design* (1994-2008), IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS (1995–1997), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (1997–1999), and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (1997–2000). He was the chair of the IEEE Computer Society Technical Committee on VLSI during 1997–2001. He served on the steering committee of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS during 1999–2001, the steering committee chair during 2002–2003, and the Editor-in-Chief for two consecutive terms during 2003–2007. He served as the program co-chair for ICVLSID'94, ISVLSI'96, ISVLSI'05, and ICVLSID'08 and as general co-chair for ICVLSID'95, IWVLSI'98, ICVLSID'98, and ISVLSI'05. He has served on technical program committees of international conferences including ICCD, ICPP, IPPS, SPDP, ICHPC, GLSVLSI, ASYNC, ISQED, ISVLSI, ISLPED, CAMP, ISCAS, MSE, and ICCAD. He was a recipient of the USF Outstanding Research Achievement Award in 2002, the USF President's Faculty Excellence Award in 2003, USF Theodore-Venette Askounes Ashford Distinguished Scholar Award in 2003, and the SIGMA XI Scientific Honor Society Tampa Bay Chapter Outstanding Faculty Researcher Award in 2004. He was a corecipient of three Best Paper Awards at the International Conference on VLSI Design in 1995, 2004, and 2006.

Soontae Kim (M'06) received the Bachelor's and Master's degrees in computer science from Chung-Ang University, Chung-Ang, Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science from Pennsylvania State University, Philadelphia, in 2003.

He was an Assistant Professor with the Department of Computer Science and Engineering, University of South Florida, Tampa, from 2004 to 2007. He is currently an Assistant Professor with the School of Engineering, Information and Communication University, Korea. His research interests include embedded system, multimedia, power-aware IT, and reliable IT.