On the Characterization and Optimization of On-Chip Cache Reliability against Soft Errors

Shuai Wang, *Student Member*, *IEEE*, Jie Hu, *Member*, *IEEE*, and Sotirios G. Ziavras, *Senior Member*, *IEEE*

Abstract—Soft errors induced by energetic particle strikes in on-chip cache memories have become an increasing challenge in designing new generation reliable microprocessors. Previous efforts have exploited information redundancy via parity/ECC codings or cacheline duplication for information integrity in on-chip cache memories. Due to various performance, area/size, and energy constraints in various target systems, many existing unoptimized protection schemes may eventually prove significantly inadequate and ineffective. In this paper, we propose a new framework for conducting comprehensive studies and characterization on the reliability behavior of cache memories, in order to provide insight into cache vulnerability to soft errors as well as design guidance to architects for highly efficient reliable on-chip cache memory design. Our work is based on the development of new lifetime models for data and tag arrays residing in both the data and instruction caches. Those models facilitate the characterization of cache vulnerability of stored items at various lifetime phases. We then exemplify this design methodology by proposing reliability schemes targeting at specific vulnerable phases. Benchmarking is carried out to showcase the effectiveness of our approach.

Index Terms—Cache, reliability, soft error, temporal vulnerability factor.

1 Introduction

With continuous technology scaling down, microprocessors are becoming more susceptible to soft errors induced by energetic particle strikes, such as high-energy neutrons from cosmic rays, and alpha particles from decaying radioactive impurities in packaging and interconnect materials [2], [3]. Due to their large share of the transistor budget and die area, on-chip caches suffer from an increasing vulnerability to soft errors [4]. As a critical requirement for reliable computing [5], protecting the information integrity in cache memories has captured a wealth of research efforts [5], [6], [7], [8], [9], [10], [11], [12], [13].

Information redundancy is fundamental to building reliable memory structures. Various coding schemes are used to protect information integrity in latches, register files, and on-chip caches, providing different levels of reliability at different performance, energy, and hardware costs. For example, simple parity coding is capable of detecting the odd number of bit errors but is not able to recover from detected errors. On the other hand, error-correcting codes (ECCs) typically provide single error correction and double error detection (SEC-DED). However, the performance overhead and additional energy consumption due to ECC encoding/decoding make ECC a reluctant choice for high-speed on-chip caches, i.e., L1 data cache and L1 instruction cache [5]. Another form of information redundancy is to maintain redundant copies of the data in cache memories [6], [14]. In these schemes, cachelines are duplicated when they are brought to L1 caches on read/write misses or on write operations. During a cache write (store), the replicas should also be updated with the latest value. On a cache read (load) operation, multiple copies may need to be read out and compared against each other to verify the absence of soft errors or to perform majority voting. Note that maintaining redundant copies of cachelines presents great challenges to the bandwidth and power dissipation of the caches [5], [14].

Despite the fact that most of the previous works have studied trade-offs between performance, energy consumption, area overheads, and the achieved cache reliability for their proposed schemes, a more systematic study of cache vulnerability is still needed. Such a study could provide enough insight into cache reliability behavior, which the designer could take advantage of to design highly cost-effective reliable caches. Recent papers [8], [9], [10], [1], [15], [16] present some initial efforts toward such a cache vulnerability analysis. However, their cacheline- or word-based vulnerability characterization used some simple generation model [17] that could not explore the temporal vulnerability of the cache, i.e., how different lifetime phases of the cache data contribute to vulnerability. This temporal information is of critical importance in determining *which data* in the cache should be protected at *what time* with *which protection schemes*, in order to achieve high reliability. In this paper, we target at providing such a bridge from perception to practice in designing reliable caches.

For the aforementioned purpose, we develop a detailed lifetime model for the data arrays in the L1 data and instruction caches, as the first step, to capture all possible activities that could involve these data items. A data item under consideration can be at various granularities such as cacheline, subblock, word, half word, byte, or even bit. In the data cache, the new lifetime model distinguishes among nine lifetime phases for each data item according to the previous activity and the current one, and further categorizes them into two groups, *vulnerable* and *nonvulnerable*

• *The authors are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, University Heights, Newark, NJ 07102. E-mail: {sw63, jhu, ziavras}@njit.edu.*

Manuscript received 29 May 2008; revised 5 Jan. 2009; accepted 15 Jan. 2009; published online 11 Feb. 2009.
Recommended for acceptance by D. Gizopoulos.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-05-0241.
Digital Object Identifier no. 10.1109/TC.2009.33.

phases. A vulnerable phase is characterized by the fact that any error that occurs during this phase has the potential to propagate either to the CPU (by load operations) or to the L2 cache (via a dirty line writeback). We define the cache temporal vulnerability factor (TVF) as the percentage of data items present in vulnerable phases over all possible data items that the cache can hold, an average along the time axis. Therefore, the temporal vulnerability factor indicates how reliable the cache is. A smaller value of TVF implies that the cache is more resilient to soft errors.

To derive highly cost-effective reliability schemes for on-chip cache memories, we propose a new design methodology driven by TVF characterization and analysis. First, we perform a cacheline-based TVF analysis on the entire data array. The results show that the vulnerable phase *write-replace* (WPL, the lifetime phase between the last write and the replacement without any read in between) contributes the most to TVF in the data cache. A writethrough data cache can effectively eliminate this phase by immediately writing back the data to the L2 cache after a store operation. However, the excessive accesses to the L2 cache degrade the performance and increase the energy consumption. An alternative to solve this problem is to early writeback dirty lines, such as the deadtime-based early writeback (DTEWB) scheme in [7]. Our further analysis indicated that this cacheline-based analysis cannot fully capture the nature of CPU accesses to the data cache. Since the unit size for data cache accesses is the byte, different bytes in the same cacheline may be in different lifetime phases at any given time, e.g., some bytes in a dirty cacheline may be in the clean state. Treating all the bytes in a cacheline equally may lead to inaccurate calculation of the cache TVF. We conclude that fine-grain (e.g., byte-based) lifetime models should be considered for more accurate TVF characterization. Based on the byte-level analysis, we also propose the multiple-dirty-bits (MDBs) scheme to further reduce the WPL vulnerable phase as well as the energy consumption during the writeback.

After WPL optimization, the vulnerable phase *read-read* (RR, the lifetime phase between two consecutive reads of a clean data item) with the potential to propagate errors to the CPU raises as another major part in the vulnerability factor of the data cache. Our study shows that a 87.8 percent majority of RRs have a short time interval (<= 0.5K cycles) and account for only 15.5 percent of the overall RR vulnerable intervals. Based on this observation, we propose a clean cacheline invalidation (CCI) scheme to invalidate clean lines after being idle for a certain period of time. Note that this scheme may result in performance loss when the invalidated cachelines are accessed lately by the CPU. However, by carefully choosing the invalidation interval, we can keep the induced performance overhead to a minimum. Our further analysis on data items in cachelines shows that a significant portion of stored data is narrow-width data, which complies with previous research findings [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. In this work, we propose to integrate a narrow-width value compression (NWVC) scheme with the lifetime models for further reducing the WPL, RR, and other vulnerable phases. The *Combined* scheme with DTEWB, MDB, CCI, and NWVC achieves a significantly reduced TVF of 3.5 percent compared to the original 39.2 percent of the data array in the data cache at a minor performance loss of 0.7 percent.

Being different from the data cache, the instruction cache is read-only (from the datapath side) and this read-only activity dramatically simplifies the lifetime model for the data array in the instruction cache. In this lifetime model, RR is the only vulnerable phase. To optimize this RR phase, we first exploit the CCI scheme, similarly to the data cache. However, the experimental results show that the performance loss due to the instruction cache CCI is much higher than for the data cache CCI. This is mainly because of the high pipeline stall penalty due to increased instruction cache misses incurred by the CCI scheme. To reduce the performance overhead, we propose a variation of the cacheline scrubbing (CS) scheme to scrub idle clean lines from the L2 cache. While reducing the RR phase without significantly impacting the performance, the scrubbing scheme dramatically increases the accesses to the L2 cache. Consequently, we further propose to combine the CCI and CS schemes to optimize the RR phase while minimizing the performance and energy overheads. Our evaluation results show that the CS-CCI scheme effectively reduces the TVF of the instruction cache data array from 19.9 to 5.3 percent at a 0.9 percent performance loss and a 29 percent energy increase in the L2 cache.

Previous work [29] has studied the fault behavior of content-addressable memory (CAM) tags and provided single-error-tolerant solutions to protect them. A functional-level design framework was also proposed in [30] for implementing a fault-tolerant/self-checking CAM architecture, with a focus on CAM cell designs. To provide a comprehensive view of cache reliability, we also strive to study the reliability behavior in the tag array for both the data and instruction caches. During an access to a set-associative cache, all tags in the same set are read out and compared simultaneously with the tag in the CPU-issued address, which puts the tags of valid cachelines into a vulnerable phase. However, if a single-bit error is assumed, Hamming-distance-one analysis (HDO) [31] can be employed to dramatically reduce the TVF of the tag array. We develop a new lifetime model for the tag array to extend the Hamming-distance-one analysis. Furthermore, we study the effect of the early writeback and clean cacheline invalidation schemes on optimizing the TVF of the tag arrays. In summary, the tag array TVF is reduced to 7.72 and 0.08 percent for the data and instruction caches from their original 46.7 and 0.3 percent, respectively.

The rest of the paper is organized as follows: The next section describes our experimental setup. In Section 3, we introduce our lifetime model of the data array in the data cache. Then, we propose and evaluate several schemes to reduce its TVF. In Section 4, we analyze the data array vulnerability to soft errors in the instruction cache and propose our improving schemes. The study of the tag array in the data and instruction caches is conducted in Section 5. Section 6 concludes this work.

2 Experimental Setup

We derive our simulator from SimpleScalar V3.0 [32] to model a contemporary high-performance microprocessor similar to Alpha 21364. In the new simulator, the original RUU (register update unit) structure is replaced by a separated integer issue queue, a floating-point issue queue, an integer register file, a floating-point register file, and an

TABLE 1
Parameters of the Simulated Processor

Processor Core	
Datapath Width	4 inst. per cycle
Int Issue Queue	20 entries
FP Issue Queue	15 entries
Load/Store Queue	64 entries
Active list (ACL)	80 entries
Int Register File	80 registers
FP Register File	72 registers
Function Units	4 IALU, 2 IMULT/IDIV 2 FALU, 1 FMULT/FDIV/FSQRT 2 MemPorts
Branch Predictor	
Branch Predictor	Alpha 21264 tournament predictor 32-entry RAS
BTB	2048-entry 2-way
Memory Hierarchy	
L1 I/DCache	64KB, 2 ways, 64B blocks, 2 cycles
L2 UCache	4MB, 8 ways, 128B blocks, 12 cycles
Memory	225 cycles first chunk, 12 cycles rest
TLB	Fully-assoc., 128 entries

active list (a.k.a. the reorder buffer). An MIPS R10000 style register renaming scheme is adopted in our implementation. Table 1 gives the detailed configuration of the simulated microprocessor. Cacti 3.2 [33] is used for energy profiling (at 70 nm technology) during the simulation.

For experimental evaluation, we use the SPEC CPU2000 benchmark suite [34] compiled for the Alpha instruction set architecture using the "-arch ev6-non_shared" option with "peak" tuning. We use the reference input sets for this study. Each benchmark is first fast-forwarded to its early single simulation point (*gap* and *ammp* use the standard single simulation point instead of the very large early single simulation point) specified by SimPoint [35]. We use the last 100 million instructions during the fast-forwarding phase to warm up the caches if the number of skipped instructions is more than 100 million. Then, we simulate the next 100 million instructions in detail.

3 Temporal Vulnerability Factor of the Data Array in Data Caches

3.1 A General Lifetime Model of the Data Array

In this section, we introduce our detailed lifetime model of the data array for the purpose of vulnerability characterization. A cacheline is first brought into the L1 data cache on a read or write miss. The cacheline will be accessed at most a couple of times, either by reads or writes, and then may wait for a long time before being replaced [17]. Such a cacheline generation information can be exploited for cache leakage optimization [17]. However, it is not sufficient for reliability analysis. Note that not all of the soft errors that occur in the data cache will result in a failure. If errors occur in the data field of invalid cachelines, they are simply masked off by the invalid bits and have no impact on the correctness of the execution. Errors occurring in the data field of clean cachelines after the last read are similarly masked off by the dirty bit ($= 0$), and therefore, are discarded at replacement.

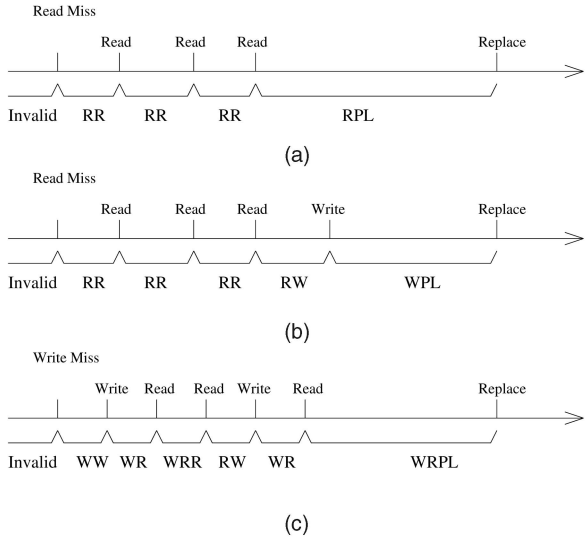


Fig. 1. The lifetime of a cacheline with respect to various access activities.

Other errors may be overwritten by subsequent writes before a CPU read or a writeback to the L2 cache, thus presenting no harm to reliability. In our new model, the lifetime of a data item, e.g., a cacheline, is divided into the following phases:

- `WRR`: lifetime phase between two consecutive reads of a dirty data item;
- `RR`: lifetime phase between two consecutive reads of a clean data item;
- `WR`: lifetime phase between a write and its first read;
- `WPL`: lifetime phase between the last write and the replacement without any read in between;
- `WRPL`: lifetime phase between the last read and the replacement of a dirty data item;
- `RPL`: lifetime phase between the last read and the replacement of a clean data item;
- `RW`: lifetime phase between the write and the last read before the write;
- `WW`: lifetime phase between two consecutive writes without any read in between;
- `Invalid`: lifetime phase when the data item is in the invalid state.

Fig. 1 shows the correlation among these lifetime phases for typical data cache activities. In this paper, we define a *vulnerable phase* as a lifetime phase in which errors may propagate out of the cache, either to the CPU or to the lower level memory hierarchy, i.e., L2 caches. Clearly, the first five phases, `WRR`, `RR`, `WR`, `WPL`, and `WRPL`, are vulnerable because errors that occur in phases `WRR`, `RR`, or `WR` will have the opportunity to be read by the CPU and errors that occur in phases `WPL` or `WRPL` will have the opportunity to propagate to the L2 cache. `RPL` and `Invalid` are *nonvulnerable phases* since errors that occur during these two phases will be discarded or ignored. However, phases `RW` and `WW` present different vulnerability behavior for data items at different granularities. If the data item is a byte, `RW` and `WW` are nonvulnerable phases. Otherwise, `RW` and `WW` are *potential vulnerable phases*. We elaborate the vulnerability characteristics of `RW` and `WW` in the following section.

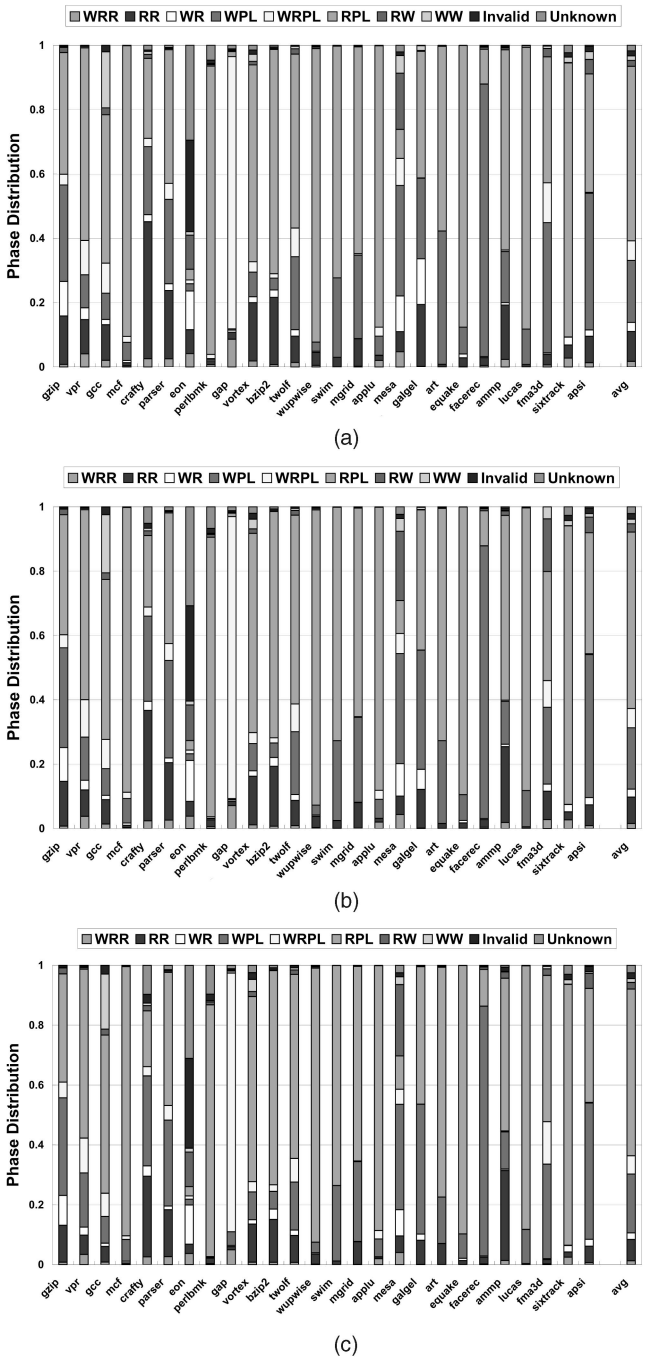


Fig. 2. The lifetime distribution of the data array in the data cache with cacheline sizes of 64, 32, and 16 bytes. (a) 64-byte cacheline. (b) 32-byte cacheline. (c) 16-byte cacheline.

3.2 TVF

The cache TVF introduced in this work is defined as the average rate of data items in vulnerable phases over the total data items that the cache can accommodate along the timeline. TVF can be calculated as follows:

$$TVF_{Cache} = \frac{\sum_i^n (data\_item_i * \sum_j vul\_phase_j)}{\sum (data\_item * Exec\_Time)}, \qquad (1)$$

where $data\_item_i$ can be a cacheline, a word, or a byte, vul_phase_j is the time of jth vulnerable phase of $data\_item_i$, and $Exec\_Time$ is the total time simulated for the benchmark.

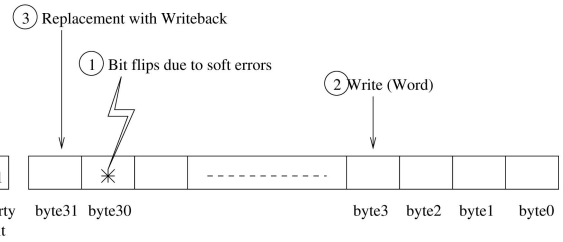


Fig. 3. A scenario of cache accesses and error occurrences that contribute RW or WW to vulnerable phases.

We use the vulnerability factor to evaluate the reliability of the data cache. If the data cache has a high vulnerability factor, it has more data items in the vulnerable phases during the execution, and hence, is more vulnerable to soft errors. Therefore, a main objective in designing a reliable data cache is to reduce its vulnerability factor. Note that the TVF is different from the architectural vulnerability factor (AVF) [36] of the data cache. Since soft errors induced during the vulnerable phases in the data cache only present the potential to crash the execution or the lower memory hierarchies, TVF defines the upper bound on AVF and can be estimated more accurately than AVF. Further, TVF is also different from the critical time [15] in that the critical time is calculated based on the word-level vulnerability analysis while TVF is derived from a flexible lifetime model for detailed vulnerability analysis at various granularities, e.g., a cacheline, a word, or a byte.

3.3 Data Array Vulnerability Characterization

In this section, we perform both cacheline-based and byte-based vulnerability characterization and analyze the deficiency of the cacheline-based scheme.

3.3.1 A Cacheline-Based Characterization

In conventional cache designs, each cacheline is associated with a dirty bit indicating whether it is a clean line or a dirty one. The dirty bit is set once the cacheline is written by the CPU. In writeback caches, the dirty cacheline is written back to the lower level caches upon replacement, as a single unit. Thus, it is very straightforward to perform data cache vulnerability analysis based on the cacheline lifetime information [8]. Applying our lifetime model, the data item here will be a cacheline. Obviously, the initial phase of all cachelines in the data cache is Invalid. Upon different CPU access activities, the cachelines enter different phases, i.e., RR, RW, WW, WR, WRR, RPL, WPL, or WRPL, at different time points.

We first analyze the impact of the cacheline size on the lifetime distribution, and thus, the vulnerability factor of the data array. Fig. 2 shows the distribution of the cacheline lifetime under three cacheline sizes, namely, 64, 32, and 16 bytes. For line-based lifetime analysis, previous research [8] considered only WRR, RR, WR, WPL, and WRPL (phase names here may be different from [8]) as vulnerable and all other phases as nonvulnerable. However, this is not accurate. As discussed in Section 3.1, the other two phases, RW and WW, have the potential to propagate errors to either the CPU side or the L2 caches. A scenario involving such an error propagation to L2 caches is illustrated in Fig. 3. If errors hit the clean bytes of a cacheline before a write updates other bytes, the error-corrupted clean bytes may also be written back to the L2 cache at a later replacement. However, if the

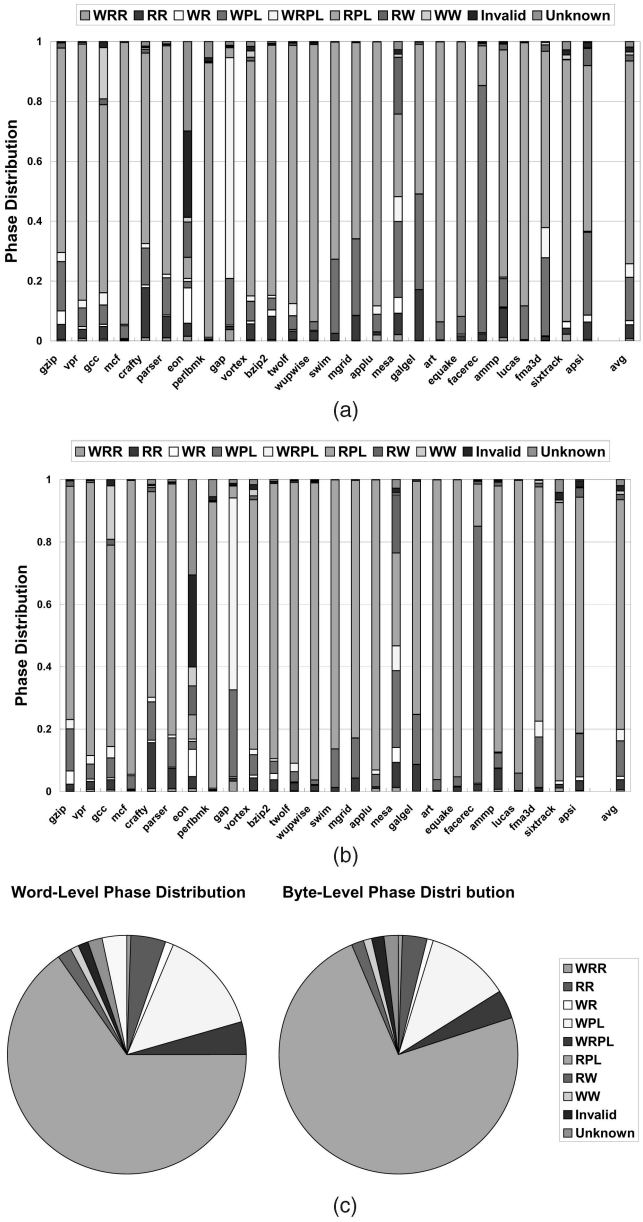


Fig. 4. The lifetime distribution of the data array in the data cache for fine granularity data items, word and byte (a) 64-bit word (b) 8-bit byte (c) An average of lifetime distribution.

erroneous clean bytes are overwritten by subsequent writes before CPU reads or a writeback operation to L2 caches, they present no harm to the correctness of program execution. Thus, we classify RW and WW as potential vulnerable phases.

Although the temporal vulnerability factor decreases as the cacheline size is reduced from 64 to 16 bytes, as shown in Fig. 2, the improvement is not significant. The TVF values for the data cache with 64, 32, and 16-bytes cachelines are 39.2 percent, 37.3 percent, and 36.3 percent, respectively. Moreover, if we simply reduce the cacheline size, the performance normally degrades because of the spatial locality property. For the default line size (64 bytes), Fig. 2a shows that the vulnerable phases account for about 39.2 percent of a cacheline's lifetime, among which, WPL and RR contribute about 19.3 percent and 9.3 percent, respectively. The two potential vulnerable phases RW and WW together account for 3.2 percent. The only truly nonvulnerable phases of the cacheline are RPL and Invalid. Note that Unknown represents a phase where the state cannot be determined because of the limited simulation time. RPL represents the largest part in the lifetime, around 54.3 percent, which is nonvulnerable. Therefore, to improve cache reliability, we should attempt to reduce the time spent by a cacheline in the WPL and RR phases.

TABLE 2
The Comparison of Vulnerability Characterization at Different Granularities

Granularity	Cacheline	Word	Byte
Vul. Phase	39.2%	25.7%	19.9%
Potential Vul.	3.2%	3.0%	0%

3.3.2 Vulnerability Characterization at Fine Granularities

Since the unit size for CPU data accesses is in the byte, a write operation does not update the entire cacheline. This characteristic of the data cache accesses results in different bytes in the same cacheline in different phases during the execution. For example, in a clean cacheline, if a byte write operation occurs, it will only update a particular byte in that cacheline and bring the entire cacheline to the dirty state. However, there is only one byte in the dirty state after the write, while others may still be in the clean state. Therefore, it is not accurate and efficient to assume that these clean bytes in a dirty cacheline are actually in the dirty state. Another problem with the line-based characterization is the inaccuracy in RW and WW profiling, as illustrated in Fig. 3. For more accurate data cache vulnerability characterization, we perform lifetime analysis while scaling down the data item granularity to a word (8-byte) or a byte. Each data item can only be in one particular phase at a given time.

Figs. 4a and 4b show the lifetime distribution based on word-level and byte-level characterization. The TVF based on word-level characterization is 25.7 percent, compared to 39.2 percent for cacheline-level analysis. This TVF value is further reduced to 19.9 percent for byte-based characterization, as shown in Fig. 4c. It is important to note that there is no potential vulnerable phase in the byte-based lifetime model. RW and WW are then true nonvulnerable phases as any error that occurred in a particular byte should be cleaned/overwritten by the subsequent write to the same byte. However, in the word-based lifetime model, RW and WW still contribute to potential vulnerable phases because of the same reason as for the cacheline-based model. Table 2 summarizes the comparison of the results by using different granularities for vulnerability characterization.

3.4 The Impact of Different Cache Write Policies

3.4.1 Writethrough versus Writeback

From our cacheline-based lifetime model, phase WPL alone contributes about 19.3 percent toward the 39.2 percent temporal vulnerability factor of the data array. A straightforward solution to reduce the WPL phase is to use a writethrough cache, where a write operation updates both L1 data cache and L2 cache. In a writethrough cache, phase WPL is effectively converted to the nonvulnerable phase RPL.

However, besides reliability, performance and energy consumption are also key factors to consider in processor

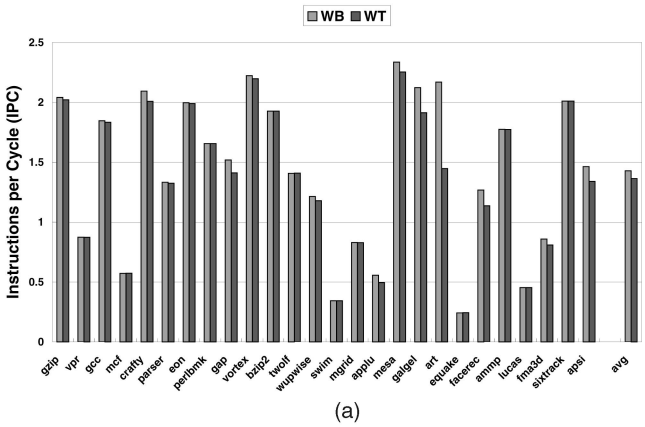


Fig. 5 (a) The comparison of IPCs between writethrough and writeback caches, and (b) the comparison of dynamic energy consumption in the L2 cache for writethrough and writeback data caches.

design. In general, the writethrough cache needs to update the L2 cache with every write to the L1 data cache. We performed a similar study as in [14], [8]. Fig. 5a compares the performance of writethrough and writeback caches. We implement the writethrough cache with an eight-entry write buffer in order to alleviate the high pressure on the bandwidth and to reduce the write stalls. We find that for the simulated benchmarks, a writethrough cache incurs a performance loss of 3.8 percent as compared to a writeback data cache. Furthermore, Fig. 5b shows that the energy consumption in the L2 cache is more than doubled if the L1 data cache changes its policy from writeback to writethrough. Therefore, for applications that require high performance and low energy consumption, the writeback cache is still preferable.

3.4.2 MDB Data Cache

From the results of line-based and byte-based vulnerability analysis, a major contributor to the TVF in a writeback cache is phase WPL. Based on the same idea as for the byte-level lifetime model, we find that if the clean bytes in a dirty cacheline are not written back to lower level caches during a replacement, any error occurring in clean bytes will be simply discarded. Thus, the WPL phase can be reduced as well as other vulnerable phases, as shown in Fig. 4b. To achieve a similar TVF with the byte-level lifetime model, we propose an MDB scheme.

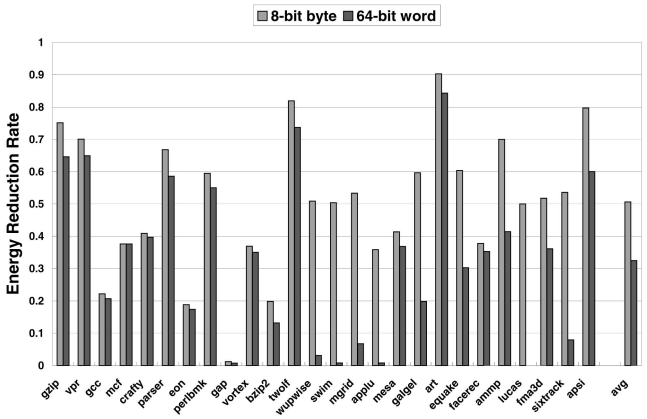


Fig. 6. The energy savings in cache writeback when applying the MDB scheme at various granularities.

In conventional data caches, there is only one dirty bit per cacheline, which prevents us from identifying whether a particular byte is dirty or not. In our MDB cache, we provide each byte with a dirty bit and update these dirty bits according to the read and write operations. For example, when the CPU writes an 8-byte word to the data cache, the eight dirty bits associated with that word in a particular cacheline are set to one. When a dirty line is to be replaced, the dirty bits control which bytes should be written back to the L2 cache. Furthermore, by writing back only dirty bytes in a dirty cacheline, the cache energy consumption can also be reduced, due to reduced energy in data transfer bus and the L2 cache [37]. Notice that the dirty bit of a dirty byte is vulnerable, because if it flips to zero, the dirty byte will not be written back to the L2 cache at replacement time. However, the dirty bit of a clean byte is not vulnerable (when a single-bit error is assumed), because if a soft error flips that dirty bit, it will only cause the clean byte to be written back.

Although the MDB scheme may incur an area overhead similar to that of providing a parity bit for each byte, our scheme has a negligible performance overhead. If the die area is highly constrained, we can relax the requirement by using a dirty bit per each word. As the comparison shown in Table 2, the vulnerability factor is slightly increased to 25.7 percent if we associate one dirty bit for each word (8 bytes). On the other hand, the area overhead is reduced to one-eighth of the byte-level dirty bit scheme. Fig. 6 also shows the energy savings of 50.6 percent and 32.5 percent, in the writeback when applying the byte-level and word-level MDB schemes, respectively.

3.4.3 Dead-Time-Based Early Writeback (DTEWB)

Previous work [7], [8] proposed early writeback schemes to reduce the vulnerable WPL phase while avoiding a dramatic increase in the accesses to the L2 cache. Early writeback schemes can be either LRU-based or dead-time-based [8]. A major design issue in the early writeback scheme is to decide when to perform the writeback in order to reduce the WPL phase as well as the accesses to the L2 cache.

The dead-time-based early writeback (DTEWB) scheme [8] could be a solution. We conducted a study based on different dead times. Fig. 7a shows that the dynamic energy consumption in the L2 cache decreases when the dead time (the idle time interval for dead prediction) increases from 500 to 4K cycles, on comparing with writethrough and

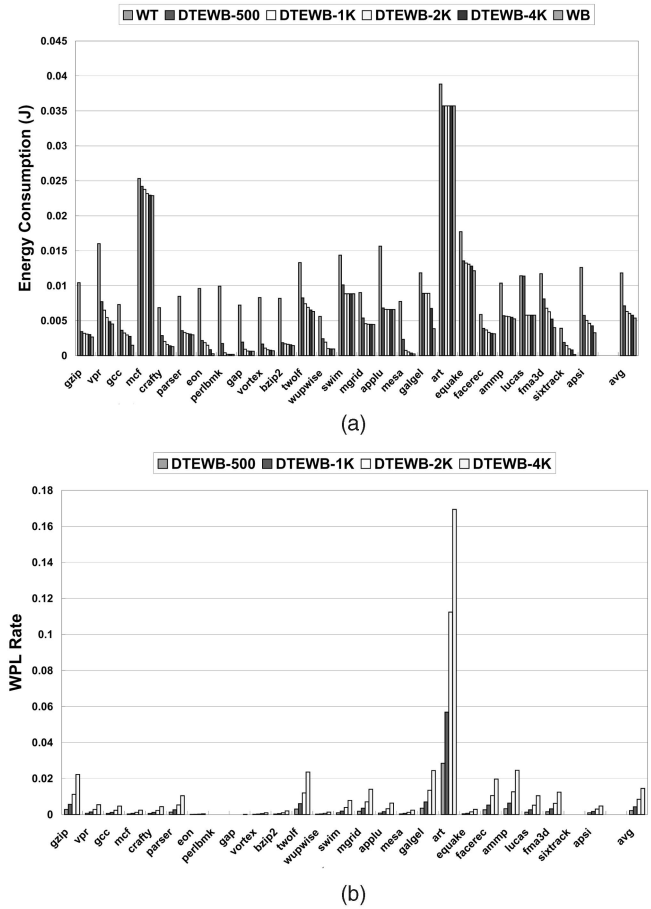


Fig. 7. (a) The comparison of dynamic energy consumption in the L2 cache at different dead times (b) WPL rates at different dead times.

writeback caches without DTEWB scheme. Fig. 7b shows how different dead times affect the vulnerable WPL phase. From these two figures, DTEWB with 2K or 4K cycles can be good choices, which can dramatically reduce the vulnerable phase WPL to 0.8 percent or 1.4 percent, at an increase of the energy consumption of 59 percent or 37 percent in the L2 cache over the conventional writeback scheme. Notice that from our simulation results, the DTEWB scheme has a negligible performance overhead compared to the write-back cache.

3.5 CCI

In the data array of the data cache, the RR phase, which is the time between two reads in a clean cacheline, contributes the second largest share to the vulnerability factor. This share becomes even dominant once the DTEWB scheme is employed, making the RR optimization of critical importance to achieving further improvement of TVF.

The basic idea for RR optimization is to reduce the time that a clean data item, i.e., a cacheline, resides in the data cache by invalidating the cleanlines after being idle for some predefined intervals. Notice that if the clean cacheline is accessed subsequently, additional performance overhead incurs due to the additional cache misses as well as the energy overhead. However, if there is no subsequent access, this invalidation does not cause any performance loss and neither reduces the RR time. Thus, there is a clear trade-off between the improved TVF and

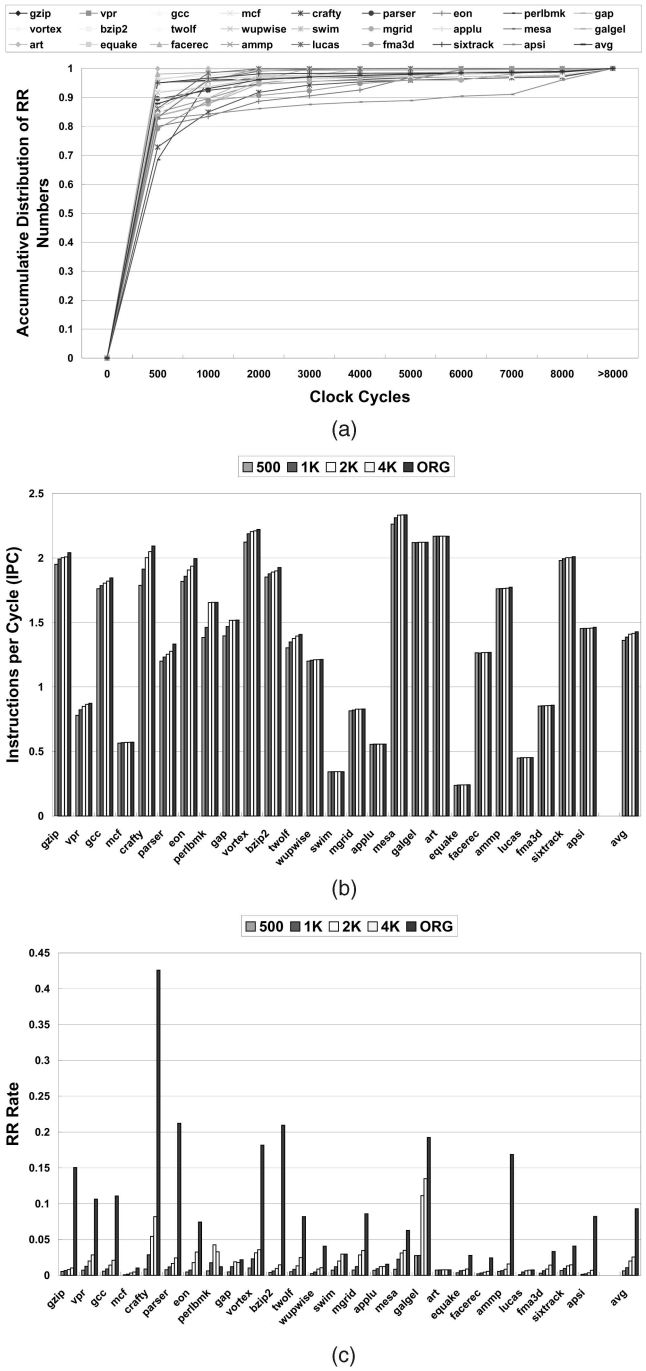


Fig. 8. (a) Cumulative distribution of the time intervals between two reads in clean cachelines, (b) the IPC comparison of different invalidation intervals, and (c) the RR phase comparison of different invalidation intervals. (ORG is the conventional data cache without the invalidation scheme.)

the performance degradation. The key is to locate such an idle interval for RR such that the RR time reduction can be maximized while the performance loss is minimized.

As shown in Fig. 8a, we profiled the number of instances with two consecutive reads to the clean cachelines based on the time interval between the two reads. The figure shows the cumulative distribution and clearly indicates that most read-read instances, around 87.8 percent (or 93.1 percent) of them, have an interval less than 500 (or 1,000) cycles. However, our results also show that a small number of

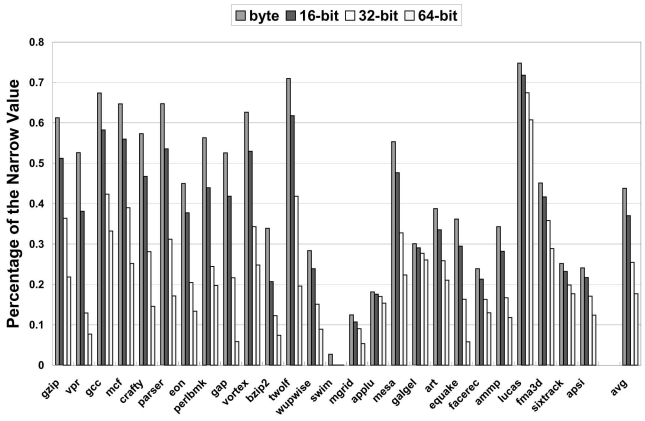


Fig. 9. The percentage of narrow-width values in active cachelines at different granularities.

read-read instances with intervals ($>= 1{,}000$ cycles) dominate the overall RR time, 84.5 percent on the average. The profile results convince us that a scheme capturing only long read-read instances should be able to substantially reduce RR time while keeping the performance loss to a minimum. Our experimental results in Figs. 8b and 8c show that 4K cycles is a good choice for this cleanline invalidation. The performance loss is only 0.7 percent and the RR phase is reduced from 9.3 percent to 2.6 percent.

3.6 Narrow-Width Value Compression (NWVC)

We also observed that value awareness can be exploited for reliability enhancement [23], [24]. Narrow width as one form of value awareness has been exploited for energy and performance optimization [18], [19], [20], [21], [22], [25], [26], [27], [28]. In [23], [24], narrow-width values are duplicated in the register file and the data cache, thus improving their reliability with both error detection and recovery capabilities via information redundancy. Different from these duplication approaches, we explore lifetime-model-driven reliability optimization through narrow-width value compression (NWVC), which is adopted from the narrow value identification schemes in [24]. NWVC uses additional narrow tag bits to mask leading zeros in a narrow-width value. The narrow tag bit masking can be applied at different granularities, for each 8-bit (byte), 16-bit, 32-bit, or 64-bit (word) data item. For instance, byte-level masking sets the narrow tag bit to one if the corresponding byte contains all zeros. Otherwise, the tag bit is reset to zero. When the data in the cacheline is accessed, the narrow tag bits are checked. If the tag bit is one, it means that the corresponding byte contains all zeros. If any error occurred in this byte, it is simply masked off by the narrow tag bit. Therefore, all the bits in the zero byte are converted into a nonvulnerable state, leading to lower TVF. Moreover, the energy consumption in the data cache can be also reduced with NWVC schemes [38], [39] since reading all-zero bytes can be avoided. Fig. 9 shows the percentage of narrow-width values in L1 data cache at different granularities. For the byte, 16-bit, 32-bit, and word-level narrow tag scheme, the percentage of narrow values is 43.8 percent, 37.0 percent, 25.5 percent, and 17.7 percent, respectively, implying the potential for TVF reduction at similar level. Notice that the narrow tag bit of a nonzero item is vulnerable, because if it flips to one, the nonzero item will be mistreated as zero. However, the narrow tag bit of a zero item is nonvulnerable if a single-bit error is assumed. This is because if the error occurs in the tag bit, the zero item will be treated as a regular value that is still zero, and will not be affected by that single-bit error.

3.7 The Combined Scheme

With the above schemes each targeting at a particular aspect in the lifetime model, we propose to evaluate the possibility and effectiveness of combining the DTEWB, MDB, CCI, and NWVC schemes so as to further improve the data array reliability, i.e., reducing the TVF of the data array. In our evaluation, we choose a 4K-cycle interval for both deadness prediction and cleanline invalidation. We use a similar implementation as in the cache decay scheme [17]. Each cacheline maintains a 2-bit local counter which is ticked every 1K cycles by a global counter. Both the dead-time-based early writeback scheme [7], [8] and the clean cacheline invalidation scheme use the same local counter. The dirty bit of the cacheline controls whether a simple invalidation or an early writeback should be performed when the local counter saturates. Considering the hardware and energy overheads, we choose the word-level tag bits for both the MDB and NWVC schemes, which associate each 64-bit word with two tag bits. For the energy evaluation, all additional tag bits are included. Fig. 10 presents the temporal vulnerability factor, performance, and cache energy consumption for data caches with and without the combined scheme. By combining DTEWB, MDB, CCI, and NWVC, we achieve a vulnerability factor as low as 3.5 percent, which significantly improves the data array reliability in the data cache, at a small performance loss of 0.7 percent. The total dynamic energy consumption in L1 data cache and L2 caches almost remains the same because of the energy saving from the MDB and NWVC schemes. Table 3 summarizes the overhead of our combined scheme.

4 Analyzing the Data Array of the Instruction Cache

4.1 The Lifetime Model

The lifetime model of the data array in the instruction cache is much simpler compared to that of the data cache, because of the read-only property. There are only three phases in this model: RR, RPL, and Invalid, with the same definition as in the model for the data cache. The only vulnerable phase in this model is RR, i.e., the time between two reads.

Unlike the data cache, all data items accessed in the instruction cache are of the same size, which is the 32-bit instruction in our simulated processor. Therefore, in a fine-granularity characterization, the 32-bit based model is accurate enough for the data array in the instruction cache. Fig. 11 shows that the TVF of the data array in the instruction cache is 19.9 percent and 16.2 percent for the cacheline-based and 32-bit-based models, respectively. We notice the small reduction in the vulnerability factor when applying 32-bit characterization. This can be explained by the access behavior in the instruction cache, which usually exploits the spatial locality for sequential accesses to instructions in the same cacheline.

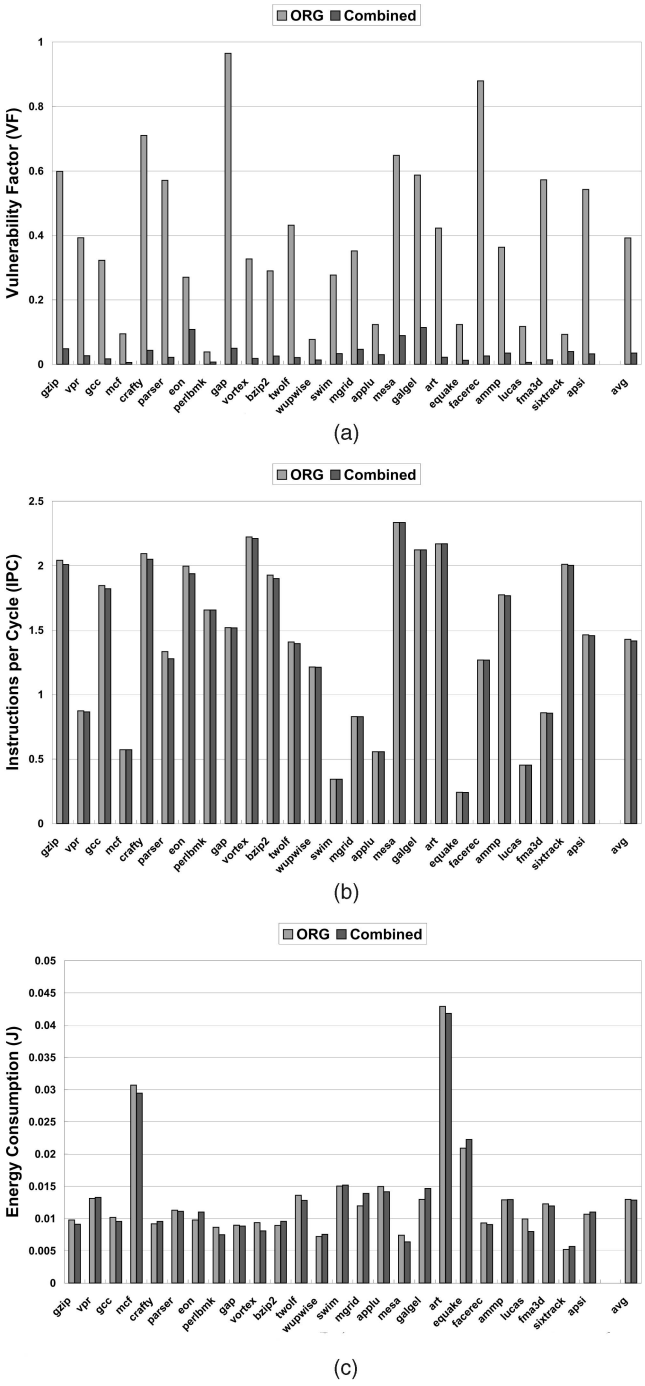


Fig. 10. The comparison between the data cache employing the combined scheme and the conventional data cache for the (a) TVF, (b) performance (IPC) impact, and (c) energy consumption in L1 data cache and L2 cache.

4.2 CCI Scheme for TVF Optimization

Since the RR phase is the only contributor to the TVF of the instruction cache data array, the proposed clean cacheline invalidation (CCI) scheme can be the option for TVF optimization. We evaluate the CCI scheme for the instruction cache with the invalidation interval ranging from 1K cycles to 16K cycles. The results shown in Fig. 12 indicate a clear trade-off between TVF and performance. If a 1K-cycle interval is used, though the TVF can be significantly reduced to 0.9 percent from the original 19.9 percent, the performance overhead is also tremendous, 20 percent performance loss, on the average. This extremely high performance loss is mainly because of the high pipeline stall penalty due to increased instruction cache misses incurred by the CCI scheme and is not affordable in high-performance designs. On the other hand, if a 16K-cycle interval is chosen, the performance loss is well under 0.9 percent, while the TVF goes back to 8.0 percent. Even with a 4K-cycle interval, CCI achieves a TVF of 4.1 percent at a performance loss of 5.8 percent. Therefore, simply applying CCI to the instruction cache will not be as effective as for the data cache. We seek solutions in next section specifically to address the performance issue in the CCI scheme for the instruction cache.

TABLE 3
Overhead of the Combined Scheme

Dirty tag bits for MDB (per line)	1 byte (per 64-byte)
Narrow tag bits for NWVC (per line)	1 byte (per 64-byte)
Interval Counters shared by DTEWB and CCI (per line)	2 bits (per 64-byte)
Total storage overhead	3.5% (18b/512b)
Performance loss	0.7%
Energy impact	negligible

4.3 CS

An accessed cacheline in the instruction cache is very likely to be accessed again due to the temporal locality property. The CCI scheme, on the other hand, invalidates the cacheline after it has been idle for a predefined time interval and incurs performance loss due to an extra cache miss if the line is to be reaccessed after the invalidation. To avoid this performance loss while still optimizing the TVF, we propose to consider cacheline scrubbing instead of invalidation, i.e., a cache miss is triggered to refetch the cacheline from the L2 cache. For this study, we assume that the L2 cache is protected by some means of ECC coding, and therefore, is error-free. To minimize the performance overhead, the cache miss to refetch the cacheline can be scheduled during cache idle cycles. Notice that our scrubbing scheme is different from the schemes in [40], [31], [41] that scrub the data by recomputing the ECC

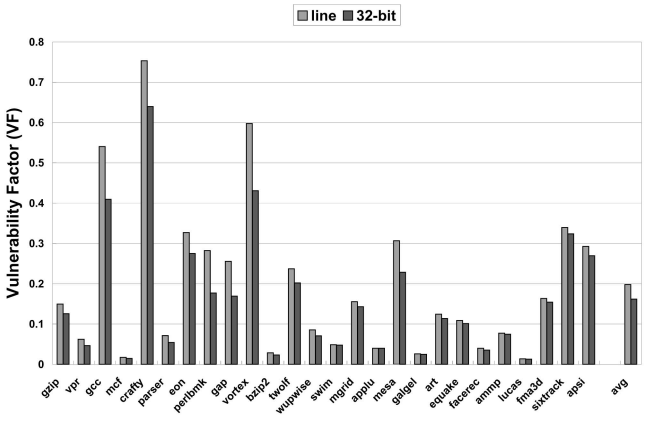


Fig. 11 The temporal vulnerability factor of the data array in the instruction cache at different granularities of a cacheline or 32-bit data.

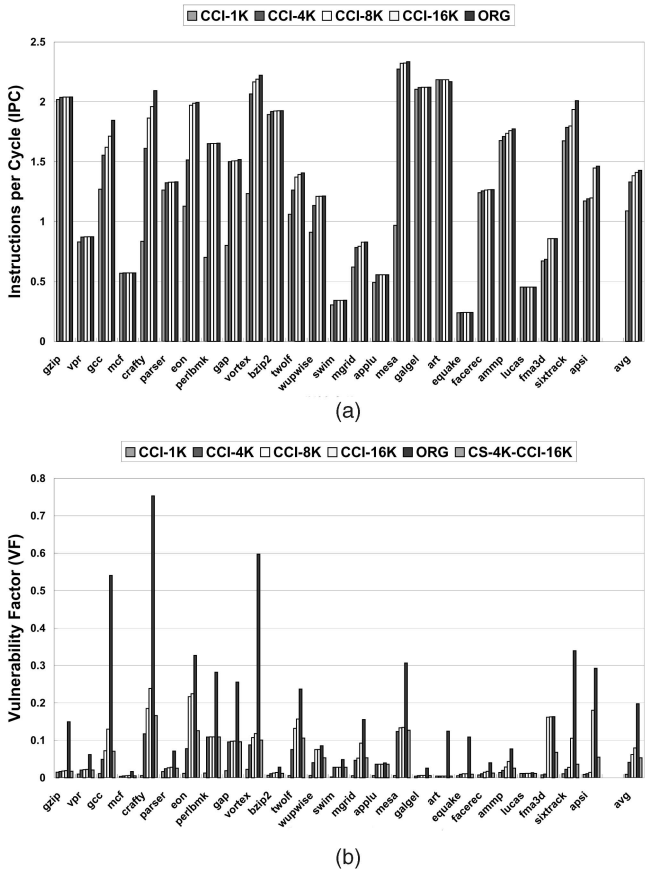


Fig. 12. (a) The IPC comparison at different invalidation intervals, and (b) the TVF comparison at different invalidation intervals while applying the CCI scheme to the instruction cache. (ORG is the conventional instruction cache without CCI. CS-4K-CCI-16K is the combined scheme with 4K-cycle CS interval and 16K-cycle CCI interval.)

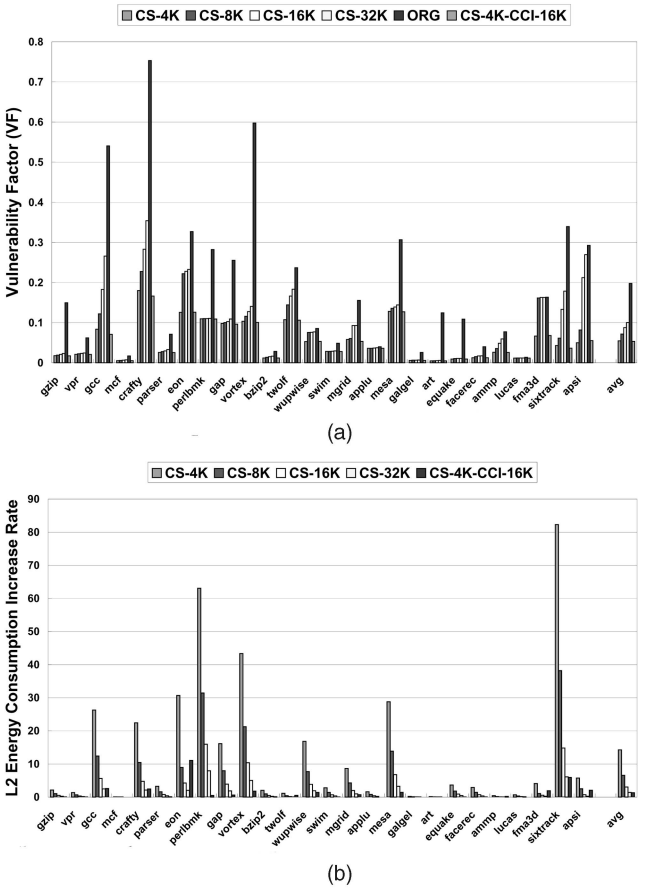


Fig. 13. (a) The TVF comparison at different scrubbing intervals, and (b) the comparison of the energy consumption increase rate in the L2 cache at different scrubbing intervals. (ORG is the conventional instruction cache without scrubbing. CS-4K-CCI-16K is the combined scheme with 4K-cycle CS interval and 16K-cycle CCI interval.)

codes to eliminate single-bit errors based on a fixed scrubbing interval.

Fig. 13a shows the TVF of the instruction data array employing the CS scheme with different scrubbing intervals. With a 4K-cycle scrubbing interval, the TVF is reduced to 5.5 percent. If the scrubbing interval increases to a larger one, such as 32K cycles, the TVF also increases to 10.0 percent. Furthermore, if smaller intervals are chosen, there will be a huge increase in the number of accesses to the L2 cache. As shown in Fig. 13b, the energy consumption in the L2 cache is 14.3 times that of the original one if the instruction cache scrubs with a 4K-cycle interval. Even if the interval increases to 32K cycles, the energy consumption in the L2 cache still becomes 1.4 times that of the original one. Once again, we are facing a reliability-energy trade-off. Without a solution to this energy issue, cacheline scrubbing may not be acceptable in energy-efficient designs.

4.4 The Combined (CS-CCI) Scheme

CCI benefits the most from capturing large RRs, while cacheline scrubbing (CS) optimizes relatively small RRs with negligible performance impact. To exploit the strength of both CCI and CS, we propose to explore combining CCI and CS for TVF optimization in the instruction cache. In the proposed combined scheme, we first scrub an idle cacheline after a small time interval. If the cacheline continues to be idle for a long interval, we invalidate it in order to prevent further (unnecessary) scrubbing. From our simulation results, we choose a 4K-cycle interval for CS and a 16K-cycle interval for CCI. The results in Fig. 12b show that the TVF of the CS-4K-CCI-16K combined scheme is 5.3 percent compared to the 8.0 percent of the CCI-16K scheme. Further, the performance of the CS-4K-CCI-16K scheme is almost the same as for the CCI-16K scheme, which is within 0.9 percent of the original scheme. Fig. 13b shows that the L2 cache energy consumption of the CS-4K-CCI-16K scheme is about 1.29 times of that for the original scheme, as compared to the 14.3 times for the CS-4K-only scheme.

5 TVF Characterization of Tag Arrays

5.1 Tag Array of the Data Cache

5.1.1 Lifetime of the Tag Array

The lifetime model of the tag array is quite different from that of the data array. This is because of the unique access pattern in the tag array. In the data array, if a clean cacheline is to be replaced, it is simply discarded, which makes the RPL time nonvulnerable. However, the RPL time of the tag array is still vulnerable. For example, during an access to a set-associative cache, all tags of different ways in the mapped set need to be read out and compared with the

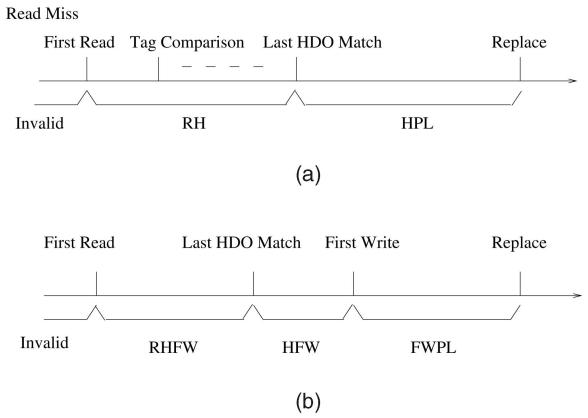


Fig. 14. The tag lifetime of a cacheline in the writeback cache. (a) Clean line. (b) Dirty line.

address tag field simultaneously. If one tag matches, the current access hits the cache. Otherwise, a cache miss is signaled. Thus, before a cacheline is selected as the candidate for replacement during a cache miss, its tag has been compared and the result is an unmatch. Now, if there are errors in the tag, it is possible to cause a false match on this cache access. Furthermore, there is no update operation on the tag. Thus, the nonvulnerable phases `RW` and `WW` in the data array are not suitable for the tag array.

5.1.2 False Hit and False Miss

If errors occur in the tag array, it may cause erroneous cache hits or misses. However, false hit and false miss have different impacts on TVF characterization. A false hit happens when a tag struck by soft errors matches the tag field of the address, which was supposed to be a cache miss. On the other hand, a false miss happens when an error-affected tag does not match the coming address tag, which should be a cache hit. A false hit will cause an incorrect execution by loading data from or updating a wrong cacheline. However, a false miss causes an additional cache miss, and thus, incurs performance loss. Its impact on TVF depends on whether it is in a clean line or a dirty line, since a false miss in a dirty cacheline will load stale data from the L2 cache. In a writeback cache, if the tag of a dirty cacheline is flipped by soft errors, the cacheline will be written back to a wrong location in the L2 cache, which is likely to cause an erroneous output.

5.1.3 Lifetime Model Based on the Extended Hamming-Distance-One (HDO) Analysis

If a single-bit error is assumed, the false hit will happen only when the tag has one single bit different from the incoming address tag and this particular bit is flipped by the soft error. We utilize the Hamming-distance-one analysis [31] to track false hits and further extend this HDO analysis method to characterize the TVF of the tag array. Notice that if a tag entry (its original value) matches an incoming address tag, any bit flipped by a soft error will cause a false miss. For tag entries with multiple bits different from the incoming tag, no false hit or false miss will happen. Furthermore, only the single different bit in the HDO tag entry is vulnerable for a clean cacheline. However,

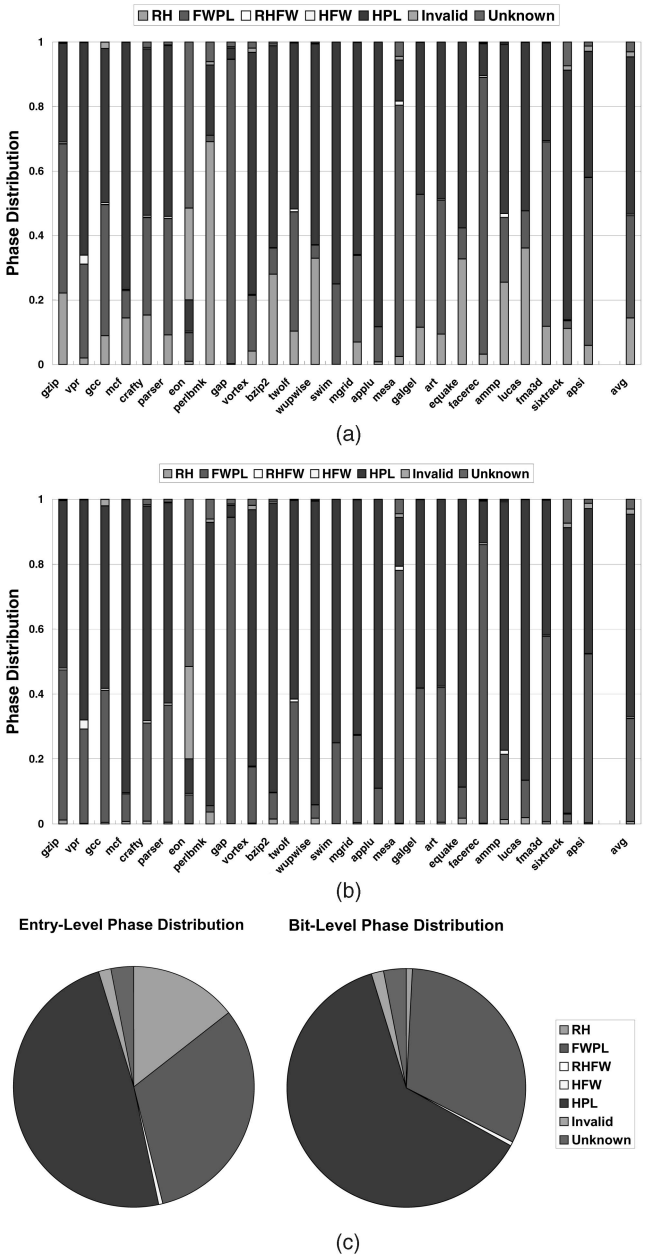


Fig. 15. The lifetime distribution for the tag array in the writeback data cache at different granularities: (a) entry level and (b) bit level. (c) An average for both entry level and bit level.

in a writeback cache, all bits in the tag entry of a dirty cacheline are vulnerable since either a false hit or a false miss will load erroneous data or corrupt the L2 cache.

Based on extended HDO analysis, we propose to divide the lifetime of the tag array in a writeback cache into six phases: `RH`, `FWPL`, `RHFW`, `HFW`, `HPL`, and `Invalid`.

- `RH`: lifetime phase between the first read and the last Hamming-distance-one (HDO) match of a clean cacheline;
- `FWPL`: lifetime phase between the first write and the replacement of a dirty cacheline;
- `RHFW`: lifetime phase between the first read and the last HDO match before the first write of a dirty cacheline;

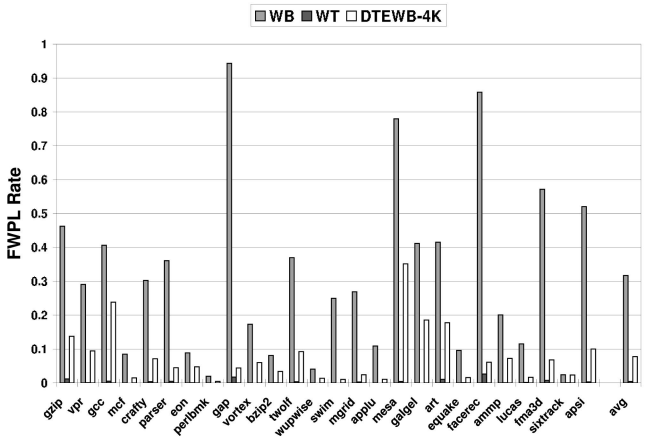


Fig. 16. The FWPL rate comparison for the tag array in writeback (WB), writethrough (WT), and DTEWB caches.

- HFW: lifetime phase between the last HDO match and the first write of a dirty cacheline;
- HPL: lifetime phase between the last HDO match and the replacement of a clean cacheline;
- Invalid: lifetime phase in the invalid state.

Fig. 14 shows the correlation among the lifetime phases for typical tag activities. The RH, FWPL, and RHFW phases are vulnerable because errors occurring in the RH and RHFW phases will cause false hits, and errors occurring in the FWPL phase will cause incorrect writebacks to the L2 cache or erroneous data load. Phases HFW, HPL, and Invalid are nonvulnerable because errors occurring in the HFW phase will only cause a false miss on the first write in a clean cacheline, and errors occurring in the HPL phase will be discarded at replacement. Fig. 15a shows the phase distribution of the tag entry in a writeback data cache. About 14.4 percent of the tag entry lifetime is in the RH phase. The FWPL phase contributes about 31.7 percent. Phases RHFW and HFW together account for 0.47 percent. Consequently, the TVF of the tag array is around 46.7 percent.

However, to improve the accuracy, TVF characterization based on the extended HDO analysis needs to be performed at the bit level. The bit-level analysis results in Fig. 15b show that the RH vulnerable phase is reduced to 0.76 percent

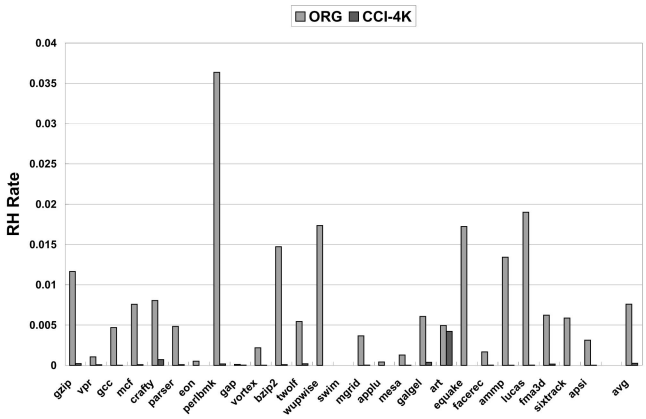


Fig. 17. The RH rate comparison between the original and CCI schemes in the data cache.

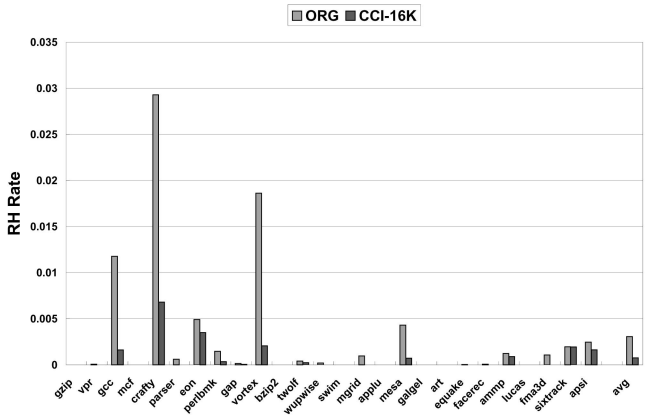


Fig. 18. The RH rate comparison between the original and CCI schemes for the tag array in the instruction cache.

from 14.4 percent in the entry-level analysis (as shown in Fig. 15a). Fig. 15c summarizes this comparison between entry-level and bit-level phase distributions, an average for all benchmarks. Notice that the FWPL vulnerable phase remains the same because all the bits in the FWPL phase are vulnerable. In the following study, we use the bit-level analysis for TVF characterization.

In a writethrough cache, the FWPL phase is eliminated. The lifetime of read-only cachelines in writethrough caches is similar to that of the clean lines in writeback caches. However, the lifetime of cachelines with write operations in the writethrough cache is quite different from that of the dirty lines in the writeback cache. In order to illustrate this difference, we compare the TVF of the cachelines with write operations in both the writethrough and writeback data caches. Fig. 16 shows that the TVF of the cachelines with write operations in the writethrough cache is only 0.4 percent, as compared to 31.7 percent for the writeback cache.

5.1.4 The Impact of DTEWB and CCI on the TVF of the Data Cache Tag Array

The DTEWB and CCI schemes in the data array also help reduce the TVF of the tag array. The DTEWB scheme will reduce the FWPL phase of a tag entry in a writeback cache, while the CCI scheme will reduce the RH phase. In order to be consistent with the data array, we use the same 4K-cycle interval for both DTEWB and CCI in the tag array study. Fig. 16 shows that the DTEWB scheme

TABLE 4
Summary of Targeting Vulnerable Phases of All Proposed Schemes

Scheme	Data Cache		Instruction Cache	
	Data Array	Tag Array	Data Array	Tag Array
DTEWB	WPL	FWPL	-	-
MDB	WPL	-	-	-
CCI	RR	RH	RR	RH
NWVC	Overall	-	-	-
CS	-	-	RR	-
Combined	Overall	Overall	-	-
CS-CCI	-	-	RR	RR

TABLE 5
Comparison of All Proposed Schemes

Scheme	Description
DTEWB	Reducing the WPL and FWPL vulnerable phases of the data and tag arrays in the writeback data cache, while minimizing the performance and energy overheads as compared to the writethrough cache.
MDB	Reducing the vulnerable phases (mainly WPL) of the data array in the data cache by preventing writing back clean data items to the L2 cache. Additional dirty tag bits are needed (1/64 storage overhead for the word-level tag).
CCI	Reducing the RR and RH vulnerable phases of the data and tag arrays in both the data and instruction caches. However, for the instruction cache, the reduction effect comes at the cost of high performance loss.
NWVC	Reducing all vulnerable phases by exploiting the narrow width values in the data array in the data cache. Additional narrow tag bits are needed (1/64 storage overhead for the word-level tag).
CS	Reducing the RR vulnerable phase of the data array in the instruction cache. However, the reduction effect comes at the cost of high energy consumption.
Combined	Reducing the overall vulnerable phases in the data cache with minimized performance and energy overheads by combining the DTEWB, MDB, CCI, and NWVC schemes (word-level tag bits for both MDB and NWVC, 4K-cycle intervals for both DTEWB and CCI).
CS-CCI	Reducing the overall vulnerable phases in the instruction cache with minimized performance and energy overheads by combining the CS and CCI schemes (a 4K-cycle interval for CS and a 16K-cycle interval for CCI).

reduces the dirty line tag FWPL to 7.7 percent and Fig. 17 shows that the CCI scheme reduces the RH rate of the clean line tags to 0.02 percent.

5.2 Tag Array of the Instruction Cache

The lifetime of the tag array in an instruction cache has only three phases: RH, HPL, and Invalid, among which only the RH phase contributes to TVF. Fig. 18 shows that the TVF of the tag array in the instruction cache is only about 0.3 percent. Among the TVF reduction schemes for the data array of the instruction cache, the CCI scheme can also help reduce the TVF of the tag array. However, the CS scheme does not have any noticeable improvement on TVF. Therefore, we only consider the CCI scheme and conduct a study on the CCI with the same 16K-cycle invalidation interval as in the combined scheme for the data array. The results in Fig. 18 show that the CCI scheme reduces the tag array TVF to only 0.08 percent.

6 Conclusions

In this paper, we performed a detailed study on the cache vulnerability to soft errors based on new lifetime models of data and tag arrays in both the data and instruction caches. This study identified major contributors (vulnerable phases) to the cache vulnerability. It aims to provide insights into cache vulnerability behavior as well as guidance in designing highly cost-effective reliable caches. Driven by the results from our TVF characterization, we proposed reliability schemes targeting at specific vulnerable phases, which are summarized in Tables 4 and 5. First, we studied the impact of different data cache write policies, early writeback schemes, and the proposed MDB scheme on reducing the vulnerable WPL phase of dirty cachelines. We proposed a clean cacheline invalidation (CCI) scheme to reduce the time when clean cachelines stay in the vulnerable RR phase and studied the narrow-width value compression (NWVC) scheme in reducing the overall vulnerable phases. By combining the DTEWB, MDB, CCI, and NWVC schemes, the data array in the data cache attains a substantially improved reliability. For the data array in an instruction cache, we proposed a variation of the CS scheme to reduce its vulnerable phase. Combined with the CCI scheme, the CS-CCI scheme achieves a lower TVF with the minimized performance and energy overheads. We also developed a new lifetime model for the tag array based on extended Hamming-distance-one (HDO) analysis. Our results with HDO analysis indicate that the tag array has a potentially low TVF, except the writeback data cache, and the DTEWB and CCI schemes can substantially improve the reliability of the tag arrays in the cache.

References

[1] S. Wang, J. Hu, and S.G. Ziavras, "On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors," *Proc. Sixth Int'l Conf. Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS 2006)*, pp. 14-20, July 2006.

[2] J.F. Ziegler et al., "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)," *IBM J. Research and Development*, vol. 40, no. 1, pp. 3-18, Jan. 1996.

[3] C. Weaver et al., "Techniques to Reduce the Soft Errors Rate in a High-Performance Microprocessor," *Proc. 31st Ann. Int'l Symp. Computer Architecture*, 2004.

[4] P. Shivakumar et al., "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proc. Int'l Conf. Dependable Systems and Networks*, pp. 389-398, June 2002.

[5] S. Kim and A. Somani, "Area Efficient Architectures for Information Integrity Checking in Cache Memories," *Proc. Int'l Symp. Computer Architecture*, pp. 246-255, May 1999.

[6] R. Phelan, "Addressing Soft Errors in ARM Core-Based Soc," ARM white paper, ARM Ltd., Dec. 2003.

[7] L. Li et al., "Soft Error and Energy Consumption Interactions: A Data Cache Perspective," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 132-137, 2004.

[8] W. Zhang, "Computing Cache Vulnerability to Transient Errors and Its Implication," *Proc. 20th IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems*, Oct. 2005.

[9] V. Sridharan, H. Asadi, M.B. Tahoori, and D. Kaeli, "Reducing Data Cache Susceptibility to Soft Errors," *IEEE Trans. Dependable and Secure Computing*, vol. 3, no. 4, pp. 353-364, Oct.-Dec. 2006.

[10] H. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Vulnerability Analysis of L2 Cache Elements to Single Event Upsets," *Proc. Conf. Design, Automation, and Test in Europe*, Mar. 2006.

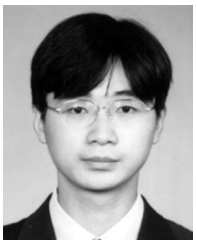
[11] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J.C. Hoe, "Multi-Bit Error Tolerant Caches Using Two-Dimensional Error Coding," *Proc. 40th IEEE/ACM Int'l Symp. Microarchitecture*, pp. 197-209, Dec. 2007.

[12] N.N. Sadler and D.J. Sorin, "Choosing an Error Protection Scheme for a Microprocessor L1 Data Cache," *Proc. Int'l Conf. Computer Design*, Oct. 2006.

[13] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M.J. Irwin, "Soft Errors Issues in Low-Power Caches," *IEEE Trans. Very Large Scale Integration Systems*, vol. 13, no. 10, pp. 1157-1166, Oct. 2005.

[14] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam, "Icr: In-Cache Replication for Enhancing Data Cache Reliability," *Proc. Int'l Conf. Dependable Systems and Networks,* 2003.
[15] G. Asadi, V. Sridharan, M.B. Tahoori, and D. Kaeli, "Balancing Reliability and Performance in the Memory Hierarchy," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software,* Mar. 2005.
[16] J. Yan and W. Zhang, "Evaluating Instruction Cache Vulnerability to Transient Errors," *ACM SIGARCH Computer Architecture News,* vol. 35, no. 4, pp. 21-28, Sept. 2007.
[17] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," *Proc. Int'l Symp. Computer Architecture,* 2001.
[18] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance," *Proc. Fifth Int'l Symp. High Performance Computer Architecture,* Jan. 1999.
[19] O. Ergin et al., "Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure," *Proc. 37th Ann. Int'l Symp. Microarchitecture,* pp. 304-315, 2004.
[20] G.H. Loh, "Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth," *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* 2002.
[21] M.H. Lipasti et al., "Physical Register Inlining," *Proc. 31st Ann. Int'l Symp. Computer Architecture,* pp. 325-335, June 2004.
[22] S. Wang, H. Yang, J. Hu, and S.G. Ziavras, "Asymmetrically Banked Value-Aware Register Files," *Proc. IEEE CS Ann. Symp. Very Large Scale Integration,* pp. 363-368, 2007.
[23] J. Hu, S. Wang, and S.G. Ziavras, "In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 281-290, June 2006.
[24] O. Ergin, O. Unsal, X. Vera, and A. Gonzalez, "Exploiting Narrow Values for Soft Error Tolerance," *IEEE Computer Architecture Letters,* vol. 5, no. 2, p. 12, July-Dec. 2006.
[25] A. Aggarwal and M. Franklin, "Energy Efficient Asymmetrically Ported Register Files," *Proc. IEEE Int'l Conf. Computer Design,* pp. 2-7, 2003.
[26] M. Kondo and H. Nakamura, "A Small, Fast and Low-Power Register File by Bit-Partitioning," *Proc. 11th Int'l Symp. High-Performance Computer Architecture,* pp. 40-49, 2005.
[27] S. Wang, H. Yang, J. Hu, and S.G. Ziavras, "Asymmetrically Banked Value-Aware Register Files for Low Energy and High Performance," *Microprocessors and Microsystems,* vol. 32, no. 3, pp. 171-182, May 2008.
[28] O. Ergin, "Exploiting Narrow Values for Energy Efficiency in the Register Files of Superscalar Microprocessors," *Proc. 16th Int'l Workshop Power and Timing Modeling, Optimization and Simulation,* pp. 477-485, 2006.
[29] J.-C. Lo, "Fault-Tolerant Content Addressable Memory," *Proc. 1993 Int'l Conf. Computer Design,* pp. 193-196, 1993.
[30] F. Salice, M. Sami, and R. Stefanelli, "Fault-Tolerant Cam Architectures: A Design Framework," *Proc. 17th IEEE Int'l Symp. Defect and Fault-Tolerance in VLSI Systems,* pp. 233-244, 2002.
[31] A. Biswas et al., "Computing Architectural Vulnerability Factors for Address-Based Structures," *Proc. IEEE Int'l Symp. Computer Architecture,* June 2005.
[32] D. Burger, A. Kagi, and M.S. Hrishikesh, "Memory Hierarchy Extensions to Simplescalar 3.0," Technical Report TR99-25, Dept. of Computer Sciences, The Univ. of Texas at Austin, 2000.
[33] P. Shivakumar and N. Jouppi, "Cacti 3.0: An Integrated Cache Timing, Power, and Area Model," technical report, Compaq Western Research Lab, 2001.
[34] Spec cpu2000 v1.3, http://www.spec.org/cpu2000/, 2009.
[35] T. Sherwood et al., "Automatically Characterizing Large Scale Program Behavior," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* Oct. 2002.
[36] S.S. Mukherjee, C.T. Weaver, J. Emer, S.K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture,* Dec. 2003.
[37] P. Pujara and A. Aggarwal, "Increasing the Cache Efficiency by Eliminating Noise," *Proc. 12th Int'l Symp. High-Performance Computer Architecture,* Feb. 2006.
[38] L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction," *Proc. 33rd Ann. Int'l Symp. Microarchitecture,* pp. 214-220, 2000.
[39] N.S. Kim, T. Austin, and T. Mudge, "Low-Energy Data Cache Using Sign Compression and Cache Line Bisection," *Proc. Workshop Memory Performance Issues,* 2002.
[40] S.S. Mukherjee, J. Emer, T. Fossum, and S.K. Reinhardt, "Cache Scrubbing in Microprocessors: Myth or Necessity," *Proc. 10th Int'l Symp. Pacific Rim Dependable Computing,* Mar. 2004.
[41] A.M. Saleh, J.J. Serrano, and J.H. Patel, "Reliability of Scrubbing Recovery-Techniques for Memory Systems," *IEEE Trans. Reliability,* vol. 39, no. 1, pp. 114-122, Apr. 1990.

Shuai Wang received the BS degree in computer science from Nanjing University, China, in 2003. Currently, he is working toward the PhD degree in the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, and is a member of the Computer Architecture and Parallel Processing Lab (CAPPL). His research interests include power/thermal-aware systems design, reliable circuits and systems, reconfigurable computing architectures, and embedded systems. He is a student member of the IEEE.



Jie Hu received the BE degree in computer science and engineering from Beijing University of Aeronautics and Astronautics, China, in 1997, the ME degree in signal and information processing from Peking University, China, in 2000, and the PhD degree in computer science and engineering from Pennsylvania State University up in 2004. Since 2004, he has been an assistant professor in the Electrical and Computer Engineering Department at New Jersey Institute of Technology. His research interests are in the areas of computer architecture, power-aware systems design, power-efficient memory hierarchy, high-performance microprocessors, complexity-effective processor microarchitecture, power-efficient reliable systems, compiler optimizations for performance and power consumption, and reconfigurable computing architecture. He is a member of the ACM, the ACM SIGARCH, the IEEE, and the IEEE Computer Society.

Sotirios G. Ziavras received the diploma degree in electrical engineering from the National Technical University of Athens, Greece, in 1984, the MSc degree in computer engineering from Ohio University in 1985, and the DSc degree in computer science from George Washington University in 1990, where he was also a distinguished graduate teaching assistant. He is currently a professor in the ECE Department at NJIT, where he has also served for four years as the associate chair for graduate studies. From 1988 to 1989, he carried out research in supercomputing for computer vision at the Center for Automation Research in the University of Maryland, College Park. In Spring 1990, he was a visiting assistant professor at George Mason University. He has authored about 140 research papers. He is listed, among others, in *Who's Who in Science and Engineering*, *Who's Who in America*, *Who's Who in the World,* and *Who's Who in the East*. His main research interests are advanced computer architecture, reconfigurable computing, embedded computing systems, parallel and distributed computer architectures and algorithms, and network router design. He is a senior member of the IEEE.