

On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors

Shuai Wang, Jie Hu, and Sotirios G. Ziavras
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102
{sw63,jhu,ziavras}@njit.edu

Abstract—Energetic-particle induced soft errors in on-chip cache memories have become a major challenge in designing new generation reliable microprocessors. Uniformly applying conventional protection schemes such as error correcting codes (ECC) to SRAM caches may not be practical where performance, power, and die area are highly constrained, especially for embedded systems. In this paper, we propose to analyze the lifetime behavior of the data cache to identify its temporal vulnerability. For this vulnerability analysis, we develop a new lifetime model. Based on the new lifetime model, we evaluate the effectiveness of several existing schemes in reducing the vulnerability of the data cache. Furthermore, we propose to periodically invalidate clean cache lines to reduce the probability of errors being read in by the CPU. Combined with previously proposed early writeback strategies [1], our schemes achieve a substantially low vulnerability in the data cache, which indicate the necessity of different protection schemes for data items during various phases in their lifetime.

I. INTRODUCTION

With continuous technology scaling down, microprocessors are becoming more susceptible to soft errors induced by energetic particle strikes such as alpha particles from decaying radioactive impurities in packaging and interconnect materials, and high-energy neutrons induced by cosmic rays [2][3]. Due to their large share of transistor budget and die area, on-chip caches are suffering 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 [6][1][5][7].

Information redundancy is the fundamental in building reliable memory structures. Various coding schemes are used to protect the information integrity in latches, register files, and on-chip caches, providing different levels of reliability at different performance, power, and hardware costs. For example, simple parity coding is capable of detecting odd (or even) numbers of bit errors but is not able to recover from detected errors. On the other hand, error correcting codes (ECC) typically provide single error correction and double error detection (SEC-DED). However, the performance overhead and additional power 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][8]. In these schemes, cache lines are duplicated when they are brought into

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. Notice that maintaining redundant copies of cache lines presents great challenges to the bandwidth of the caches [5][8].

Despite the fact that most of the previous work has studied the trade-offs among performance, power, area overheads and the achieved cache reliability under the proposed schemes, there has not been a systematic study of cache vulnerability. Such a study should be able to provide enough insight into cache reliability issues that the designer could take advantage of to render highly cost-effective reliable caches. Two recent papers [7][9] present initial efforts towards such a cache vulnerability analysis. However, their cacheline- or word-based vulnerability characterization used some simple generation model [10] which did not explore the temporal vulnerability of the cache; i.e., how different lifetime phases of the cache data contribute to cache vulnerability. This temporal information is critical in determining *what data* in the cache should be protected at *what time* and with *which protection schemes*, in order to achieve high reliability. In this paper, we target such a bridge from perception to practice in designing reliable caches for embedded microprocessors.

We first develop a detailed lifetime model for the data in the L1 data cache to capture all possible activities that could happen to the data items. The data items under consideration can be at different granularities such as cacheline, subblock, word, half word, byte, or bit. The lifetime model distinguishes nine life phases for each data item according to the previous activity and the current one, and further categorizes them into two groups, vulnerable and non-vulnerable phases. A vulnerable phase is characterized by the fact that errors that occurred during this phase have the potential to propagate either to the CPU (by load operations) or to the L2 cache (via a dirty line replacement). We define the cache vulnerability factor (VF) as the percentage of data items in vulnerable phases over all possible data items that the cache can hold, an average along the time axis. The vulnerability factor indicates how reliable the cache is. A smaller value for VF implies that the cache is more resilient to soft errors because of a lower

chance that errors will exit the cache.

Our cacheline-based vulnerability analysis shows that the vulnerable *write-replace (WPL)* phase contributes the most to VF. A write-through cache can eliminate this phase by immediately writing back the data to the L2 cache after a store operation. However, the extra accesses to the L2 cache would degrade the performance and increase the power consumption. An alternative to solve this problem is the early writeback scheme which is proposed by previous work [1]. Further investigation indicates that this line-based analysis does not capture the nature of CPU accesses to the data cache. Since the unit size for data cache accesses is a byte, different bytes in the same cache line may be in different life phases at any given time; e.g., some bytes in a dirty cache line may be in the clean state. Treating all the bytes in a cache line equally may lead to inaccurate calculation of the actual vulnerability factor of the cache. Therefore, we also propose to characterize the vulnerability factor at finer granularities.

After *WPL* optimization, the vulnerable *read-read (RR)* phase that has the potential to propagate errors to the CPU raises as another major part in the vulnerability factor. Our study shows that a majority of 91.5% *RRs* have a short time interval ($\leq 0.5k$ cycles) and account for only 11% of the total accumulated *RR* vulnerable intervals. Based on this observation, we propose to invalidate clean cache lines after being idle for a certain amount of time. Note that this scheme could result in performance loss when the invalidated cache lines are accessed later by the CPU. However, by carefully choosing the invalidation interval, we can control the induced performance overhead to the minimum. By combining the early writeback scheme with our periodical invalidation scheme, the combined scheme achieves a significantly low VF of the data cache, which makes the data cache much more reliable.

The rest of the paper is organized as follows. The next section describes our experimental setup. In Section 3, we introduce the general lifetime model and our profiling scheme. We use Section 4 to analyze data cache vulnerability to soft errors at different granularities. We compare in Section 5 the effectiveness and impact of various write policies on reducing the vulnerability factor. In Section 6, we propose and evaluate our clean cacheline invalidation scheme and the combined scheme with early writeback strategies. Section 7 concludes this work.

II. EXPERIMENTAL SETUP

We derive our simulators from SimpleScalar V3.0 [11] to model a contemporary embedded microprocessor similar to the PowerPC 740 [12] with a 16KB data cache. Table I shows the detailed information for the configuration of the simulated microprocessor. For experimental evaluation, we use 13 Media benchmarks [13] compiled for PISA instruction set architecture.

Processor Core	
Issue Width	4 instructions
Load/Store Queue	16 entries
RUU	16 entries
Function Units	2 IALU, 1 IMULT/IDIV, 1 FALU 1 FMULT/FDIV/FSQRT, 2 MemPorts
Branch Predictor	
Branch Predictor	bimodal predictor with 512-entry table, 64-entry, 4-way, BTB
Memory Hierarchy	
L1 I/DCache	16KB, 8 ways, 32B blocks, 1 cycle latency
L2 UCache	256KB, 4 ways, 64B blocks, 6 cycle latency
TLB	128 entries, 2 ways

TABLE I
PARAMETERS FOR THE SIMULATED MICROPROCESSOR.

III. DATA CACHE LIFETIME MODEL

A. A General Model

In this section, we introduce our detailed lifetime model for the data cache. A cache line is first brought into the L1 data cache on a read miss or a write miss. The cache line may be accessed a couple of times, either for reads or writes, and then may wait for quite a long time before it is replaced [10]. Such generation information is exploited for cache leakage optimization, however, is not enough for reliability analysis. Notice that not all soft errors occurring in the data cache will result in a failure. If errors happen in invalid cache lines, they are simply masked off by the invalid bit and have no impact on the correctness of the execution. Errors occurring in clean cache lines after the last read are similarly masked off by the dirty bit ($= 0$) and discarded at replacement. 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 cache lifetime model, the lifetime of a data item, e.g., a cache line, is divided into the following nine phases: *WRR*, *RR*, *WR*, *WPL*, *WRPL*, *RPL*, *RW*, *WW*, and *Invalid*.

- **WRR**: time between the first read and the last read after a write
- **RR**: time between the first read and the last read of a clean data item
- **WR**: time between the last write and the first read
- **WPL**: time between the last write and the replacement without any read in between
- **WRPL**: time between the last read and the replacement of a dirty data item
- **RPL**: time between the last read and the replacement of a clean data item
- **RW**: time between the last read and the first write
- **WW**: time between the first write and the last write without any read in between
- **Invalid**: time in the invalid state

Figure 1 shows the correlation among these nine lifetime phases for typical data cache activities. We define in this paper the vulnerable phase as the lifetime phase in which errors have the opportunity to propagate out of the cache, either

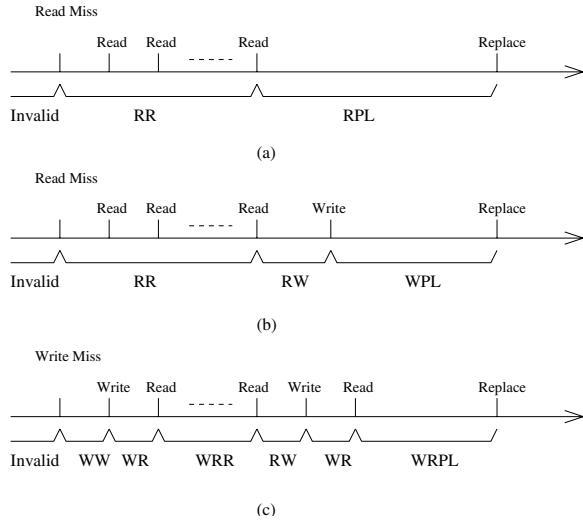


Fig. 1. The lifetime of a cache line.

to the CPU or to the lower level memory hierarchy, i.e., L2 cache. Clearly, the first five phases, WRR, RR, WR, WPL, and WRPL, are vulnerable because errors occurring in phases WRR, RR, or WR will have the opportunity to be read by the CPU, and errors occurring in phases WPL or WRPL will have the opportunity to propagate to the L2 cache. RPL and Invalid are non-vulnerable phases since errors occurring in these two phases will be discarded or ignored. However, phases RW and WW present different vulnerability features for data items at different granularities. If the data item under consideration is a byte, RW and WW are non-vulnerable phases. Otherwise, RW and WW are potential vulnerable phases. We are going to elaborate on this vulnerability characteristic of RW and WW in the following section.

B. Temporal Vulnerability Factor (TVF)

The cache temporal vulnerability factor (TVF) is defined as the average rate of data items in vulnerable phases over the total data items that the data cache can accommodate during the execution. TVF can be calculated as follows:

$$TVF_{DCache} = \frac{\sum_i (data_item_i * \sum_j vul_phase_j)}{\sum_i (data_item_i * Exec_Time)} \quad (1)$$

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 vulnerable phases during the execution, thus it is more vulnerable to soft errors. Therefore, the main objective of designing a reliable data cache is to reduce its vulnerability factor. Notice that the temporal vulnerability factor (TVF) is different from the architectural vulnerability factor (AVF) [14] 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

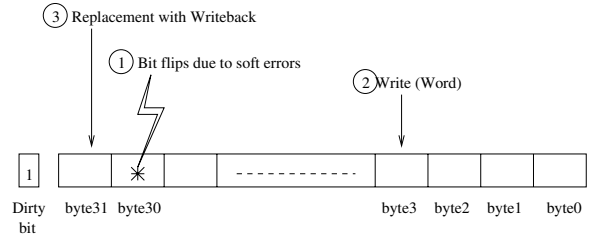


Fig. 2. A scenario of cache accesses and error occurrences that contribute RW or WW to vulnerable phases.

memory hierarchies, TVF defines the upper bound on AVF and can be estimated more accurately than AVF.

IV. DATA CACHE VULNERABILITY CHARACTERIZATION

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

A. A Cacheline-based Characterization

In conventional cache designs, each cache line is associated with a dirty bit indicating whether it is a clean line or a dirty line. The dirty bit is set once the cache line is written by the CPU. In write-back caches, the dirty cache line is written back to the lower level cache upon replacement, as a single unit. Thus, it is very straightforward to perform data cache vulnerability analysis based on the cacheline lifetime information [7]. Applying our lifetime model, the data item here will be each cache line. Obviously, the initial phase for all cache lines in the data cache is *Invalid*. Upon different CPU access activities, the cache lines enter different phases, *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 thus the vulnerability factor of the data cache. Figure 3 (a), (b), and (c) show the distribution of the cache lifetime under three cache line sizes, 32, 16, and 8 bytes. For line-based lifetime analysis, previous research [7] only considered *WRR*, *RR*, *WR*, *WPL*, and *WRPL* as vulnerable and other phases as non-vulnerable. However, this is not accurate. As discussed early in Section III, the other two phases *RW* and *WW* have the potential to propagate errors to either the CPU side or L2 caches. A scenario of such an error propagation to L2 caches is illustrated in Figure 2. If errors hit the clean bytes 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 erroneous clean bytes were overwritten by later writes before CPU reads or writeback to L2 caches, then they present no harm to the correctness of the program execution. Thus, we classify *RW* and *WW* as potential-vulnerable phases.

Although the vulnerability factor decreases as the cache line size is reduced from 32 bytes to 8 bytes, as shown in Figure 3, the improvement is not significant. The TVF values for the data cache with 32-byte, 16-byte, and 8-byte cachelines are 35.7%, 34.6%, and 33.8%, respectively. Since

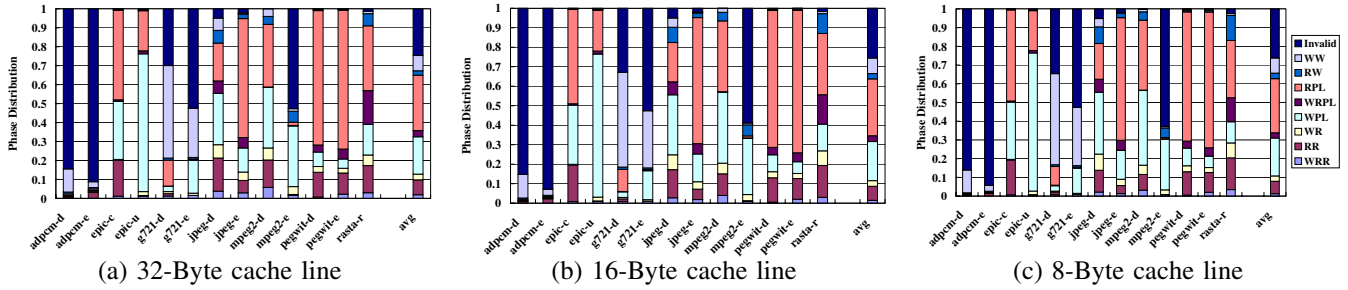


Fig. 3. Lifetime distribution of the data cache at different cacheline sizes, 32-byte, 16-byte, and 8-byte.

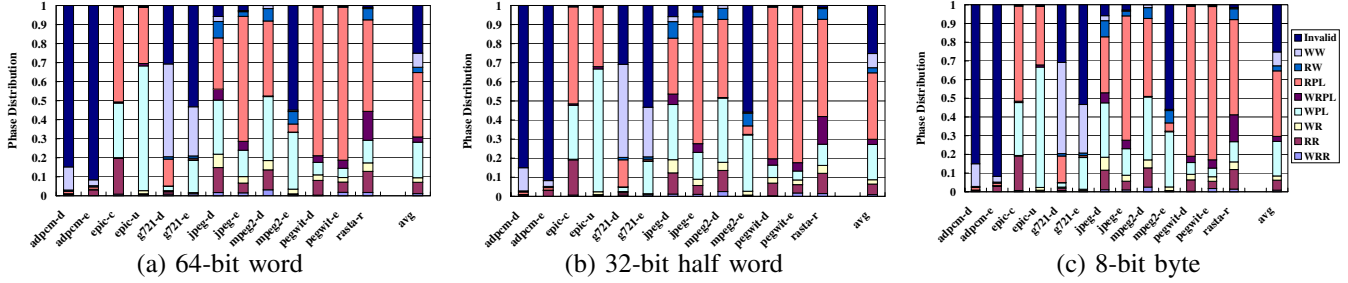


Fig. 4. Lifetime distribution of the data cache at fine granularity data items, word, half word, and byte.

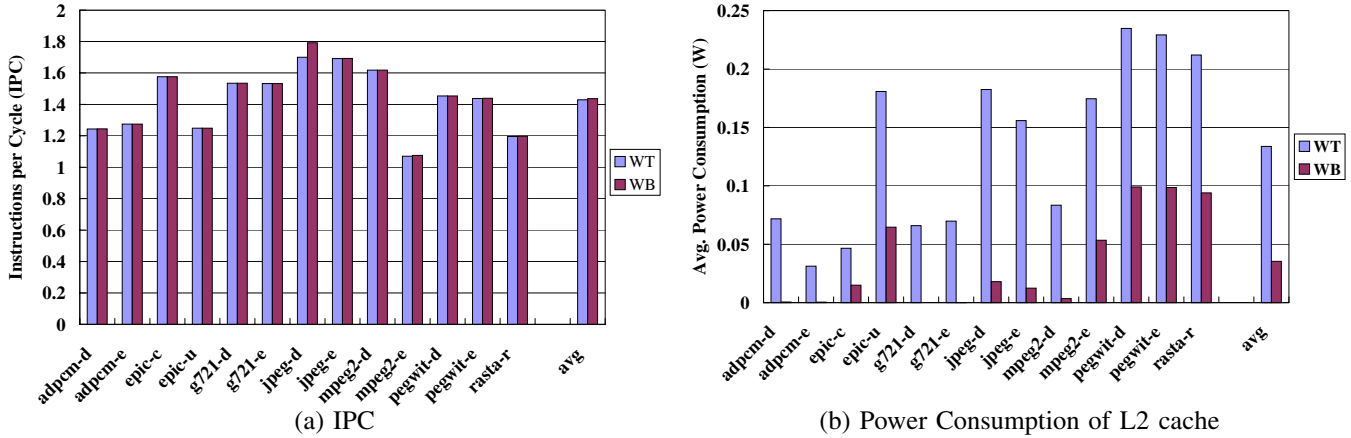


Fig. 5. (a) A comparison of IPCs between write-through (WT) and write-back (WB) data caches, and (b) a comparison of dynamic power consumption in the L2 cache for write-through (WT) and write-back (WB) data caches.

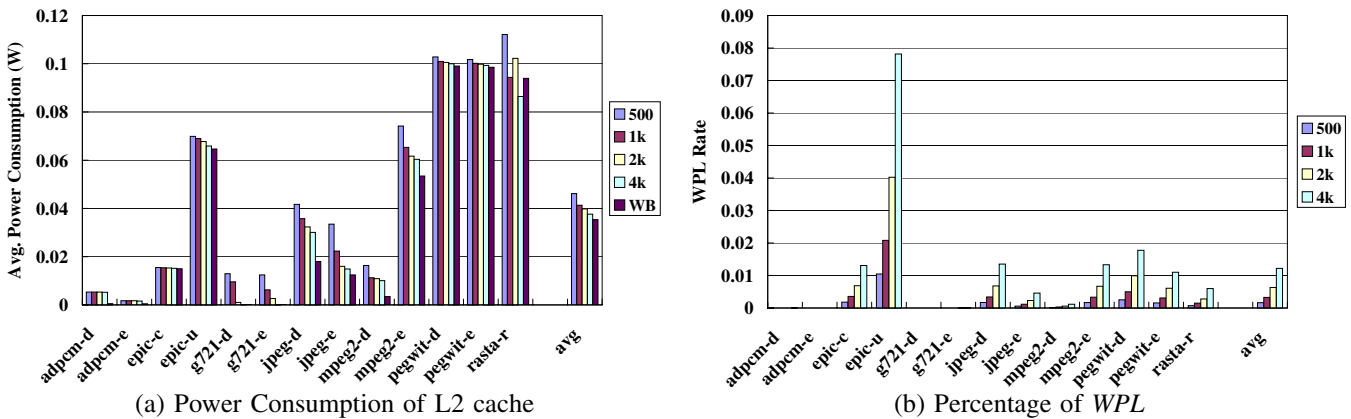


Fig. 6. (a) A comparison of dynamic power consumption in the L2 cache at different dead times (b) WPL rates at different dead times.

the characterization is performed at the cache line level, we cannot get the detailed information of each word or byte within a cache line. Phases *WR*, *RR*, *WR*, *WPL*, and *WRPL* are conventional vulnerable phases which are also mentioned in [7]. For the default line size (32 bytes), Figure 3 (a) shows that these five parts constitute about 35.7% in the lifetime of a cache line, among which *WPL* and *RR* contribute about 19.7% and 7.8%, respectively. The two potential vulnerable phases *RW* and *WW* together account for 10.4%, which is also a large part. The only truly non-vulnerable phases of the cache line are *RPL* and *Invalid*. *RPL* presents the largest part in the lifetime, around 53.8%, which is non-vulnerable. Therefore, in order to improve the reliability of caches, we must target the reduction of the time of cachelines in *WPL* and *RR* phases.

B. Vulnerability Characterization at Fine Granularities

Since the unit size for CPU data accesses is the byte, a write operation does not update the whole cache line. This characteristic of data cache accesses results in different bytes of the same cacheline to potentially belong to different phases. For example, in a clean cacheline, if a byte write operation occurs, it will only update a particular byte in this cacheline and the whole cacheline becomes a dirty line. 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 to assume that the clean bytes in a dirty cacheline are in the dirty state. Another problem with the line-based characterization is the inaccuracy in *RW* and *WW* profiling, as illustrated in Figure 2. For more accurate data cache vulnerability characterization, we perform lifetime analysis while scaling down the data item granularity to word (8 bytes), half word (4 bytes) and byte. Each data item can only be in one particular phase at any given time.

Figure 4 (a), (b), and (c) show the lifetime distribution based on word-level, half-word-level, and byte-level characterization. The vulnerability factor (TVF) based on word-level characterization is 30.9%, comparing to 35.7% from the cacheline-level analysis. This TVF is further reduced to 29.8% and 29.6% for half-word- and byte-based characterization. We notice the small reduction in the vulnerability factor when applying byte-level characterization. This can be explained by the behavior of store operations in these media benchmarks, which usually apply sequential accesses to the data items (mostly in bytes) in the cache line. Thus, the percentage of clean data items in a dirty line is very low in the simulated media benchmarks. However, in general purpose applications, this percentage is normally quite high, resulting in a much smaller vulnerability factor in byte-based characterization. For example, from our experimental results of 14 SPEC2000 benchmarks, the TVF is reduced from 36.1% in line level to 16.1% in byte level analysis.

It is also important to notice that there is no potential vulnerable phase in the byte-based lifetime model. *RW* and *WW* now are true non-vulnerable phases as any error that happened in a particular byte shall be cleaned/overwritten by the later write to the same byte. However, in word-based and

half-word-based lifetime models, *RW* and *WW* still contribute to potential vulnerable phases because of the same reason as in cacheline-based model.

V. THE IMPACT OF DIFFERENT CACHE WRITE POLICIES

A. Write Through vs. Write Back

From our lifetime model, the largest contribution to the vulnerable phases comes from the *WPL* which account for about 19.7%. One method to reduce this part is to use a write-through cache, where a write operation updates the L1 data cache, as well as the L2 cache. It seems that the write-through cache converts the vulnerable part *WPL* to the non-vulnerable part *RPL*.

However, besides reliability, the performance and power consumption are also the key factors to consider in embedded systems design. In general, the write-through cache needs to update the L2 cache on every write to the L1 data cache. Figure 5 (a) compares the performance of the write-through cache and the write-back cache. We implement the write-through cache with a 4-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 media benchmarks, the performance of a write-through cache is only 0.5% worse than that of a write-back cache. However, Figure 5 (b) shows that the accesses to the L2 cache in the write-through scheme consume 3.78 times the dynamic power of such accesses in the write-back scheme. Therefore, for the applications which require low power consumption, the write-back cache is much more preferred.

B. Dead Time based Early Write Back

Previous work [1][7] proposed early writeback schemes to reduce the vulnerable *WPL* phase while avoiding dramatic increases in accesses to the L2 cache. Early writeback schemes can be either LRU-based or dead time based [7]. A major issue in 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 could be a solution. We conduct a study based on different dead times. Figure 6 (a) shows the different dynamic power consumptions of the L2 cache with different dead times. Figure 6 (b) shows how the different dead times affect the vulnerable *WPL* phase. By comparing these results, the 500- and 1K-cycle dead time are good choices. They have 0.2% and 0.3% *WPL* phases and the average power consumption is only 1.31 and 1.17 times that of the writeback scheme.

VI. CLEAN CACHELINE INVALIDATION

In a data cache, the *RR* phase, which is the time between the first read and the last read in a clean cache line, contributes the second largest share to the vulnerability factor. Optimizing *RR* time is of critical importance to achieve a further improvement of TVF once early writeback schemes were employed.

The basic idea for *RR* optimization is to reduce the time that a clean data item, i.e., a cache line, resides in the data

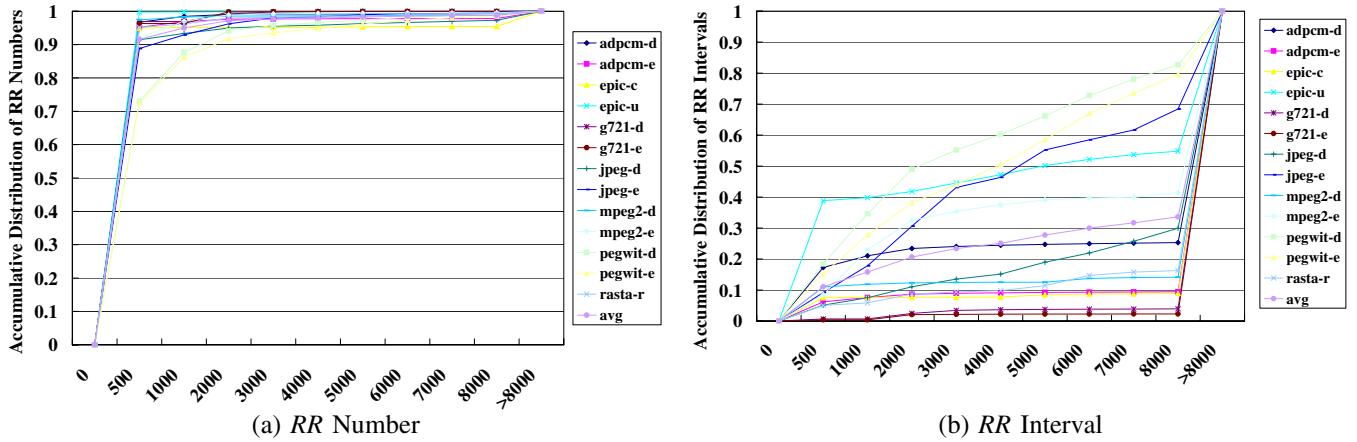


Fig. 7. (a) Accumulative distribution of numbers of different intervals between two reads in clean cache lines, (b) Accumulative distribution of time of different intervals between two reads in clean cache lines.

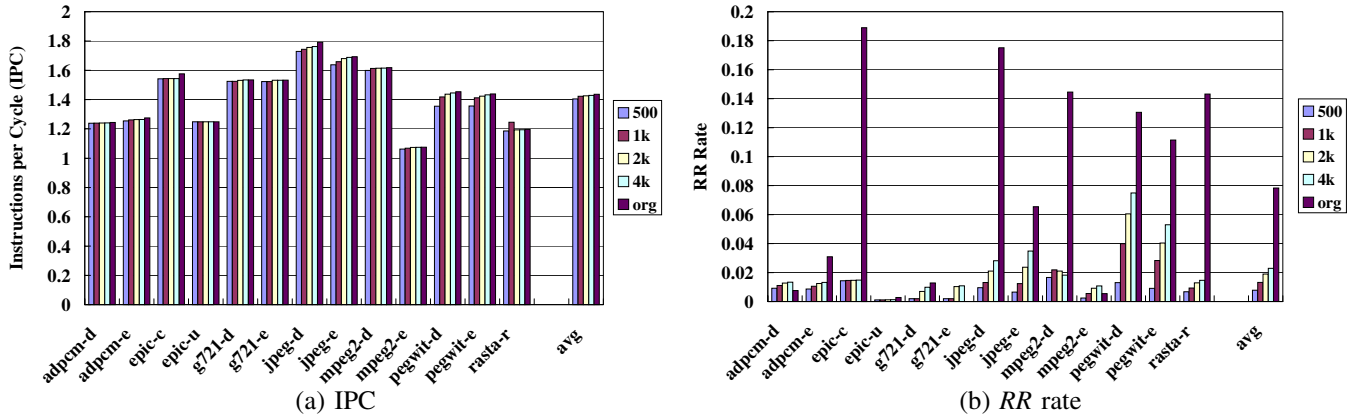


Fig. 8. (a) An IPC comparison of different invalidation intervals, and (b)RR phase comparison of different invalidation intervals. *org* is the data cache without the invalidation scheme.

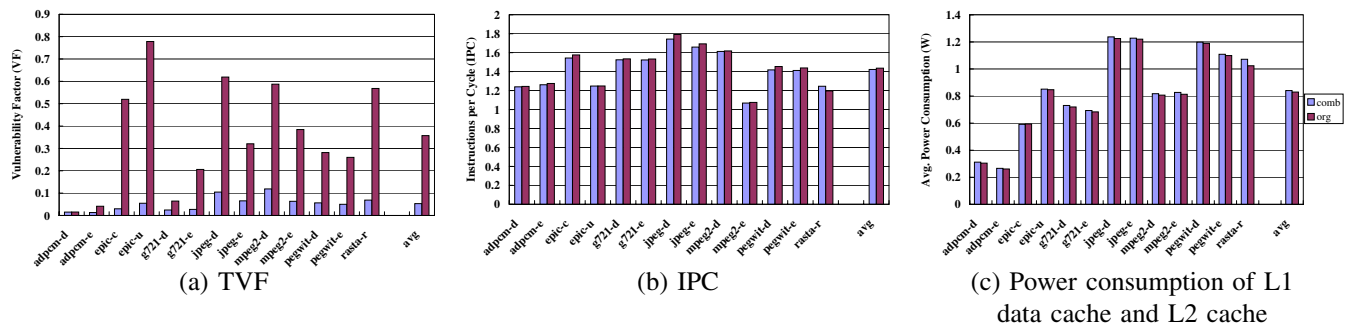


Fig. 9. A comparison of a data cache employing the combined scheme and a normal cache on the vulnerable factor, performance, and L1 data cache and L2 cache dynamic power consumption.

cache by invalidating the clean lines after being idle for some predefined interval. Notice that if the clean cache line is accessed later, additional performance overhead is incurred due to the additional cache misses as well as the power overhead. However, if there is no later access, this invalidation does not cause any performance loss, neither reduces the *RR* time. Thus, there is a tradeoff between the improved TVF and the performance degradation. The key is to locate such an idle interval for *RR* that the *RR* time reduction can be maximized while the performance loss is minimized.

As shown in Figure 7 (a), we profiled the number of instances with two consecutive reads to the clean cache lines according to the time interval between the two reads. The figure shows the cumulative distribution and clearly indicates that most read-read instances, around 91.5%, have an interval less than 500 cycles. However, from Figure 7 (b) showing a cumulative distribution of read-read time according to different intervals, we find that a small number of read-read instances with long intervals (≥ 1000 cycles) dominate the overall *RR* time, 84.2% on the average. These two figures convince us that a scheme capturing only long read-read instances should be able to substantially reduce the *RR* time while keeping the performance loss to a minimum. Our experimental results in Figure 8 show that 1K cycles is a good choice for this clean line invalidation. The performance loss is only 1% and the *RR* phase is reduced to 1.3% compared to the previous 7.8%.

In order to reduce the overall vulnerability factor (TVF) of the data cache, we propose to combine the dead time based early writeback and clean cacheline invalidation. In our evaluation, we choose 1K-cycle as the interval for both deadness prediction and line invalidation. We use a similar implementation as in the cache decay scheme [10]. Each cache line maintains a 2-bit local counter which is ticked every 256 cycles by a global counter. Both the dead time based early writeback scheme [1][7] and the clean line invalidation scheme use the same local counter. The dirty bit controls whether a simple invalidation or an early writeback will be performed when the local counter saturates. Figure 9 shows the vulnerability factor, the performance and the L2 cache power consumption. By combining these two schemes, we achieve a vulnerability factor as low as 5.3%, which significantly improves the data cache reliability, at a small performance loss of 1% and a 1.3% increase in the total dynamic power consumption of the L1 data cache and L2 caches.

VII. CONCLUSIONS

In this paper, we performed a detailed study on data cache vulnerability for the lifetime of data items in the cache. This study identified major contributors to the data cache vulnerability, which may help future work in designing highly cost-effective reliable caches. We also studied the impact of different data cache write policies and early writeback schemes on reducing the dirty cache lines staying in the vulnerable *WPL* phase. Then we proposed the clean line invalidation scheme to reduce the clean cache lines staying in the vulnerable *RR* phase. By combining dead time based early writeback with our

clean cache line invalidation scheme, the data cache attains a substantially improved reliability.

REFERENCES

- [1] L. Li *et al.*, "Soft error and energy consumption interactions: A data cache perspective," in *Proc. of ISLPED'04*, 2004, pp. 132–137.
- [2] J. F. Ziegler *et al.*, "IBM experiments in soft fails in computer electronics (1978 - 1994)," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 3–18, January 1996.
- [3] C. Weaver *et al.*, "Techniques to reduce the soft errors rate in a high-performance microprocessor," in *Proc. of ISCA-31*, 2004.
- [4] P. Shivakumar *et al.*, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. International Conference on Dependable Systems and Networks*, June 2002, pp. 389–398.
- [5] S. Kim and A. Somani, "Area efficient architectures for information integrity checking in cache memories," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, May 1999, pp. 246–255.
- [6] R. Phelan, "Addressing soft errors in arm core-based soc," ARM White Paper, ARM Ltd., Dec 2003.
- [7] W. Zhang, "Computing cache vulnerability to transient errors and its implication," in *Proc. of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, Oct. 2005.
- [8] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam, "Icr: in-cache replication for enhancing data cache reliability," in *Proc. of the International Conference on Dependable Systems and Networks*, 2003.
- [9] G. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Balancing reliability and performance in the memory hierarchy," in *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS05)*, March 2005.
- [10] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *Proc. the Int'l Symposium on Computer Architecture*, 2001.
- [11] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," Computer Sciences Department, University of Wisconsin, Tech. Rep. 1342, 1997.
- [12] M. Datasheet, "Powerpc 740 and powerpc 750," in *CMOS 0.20 um Copper Technology, PID-8p, PPC740L and PPC750L, dd3.2*.
- [13] C. Lee, M. Potkonjak, and W. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. the 30th Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 330–335.
- [14] 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," in *Proc. the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2003.