Abstract

With continuous scaling down of the semiconductor technology, the soft errors induced by energetic particles have become an increasing challenge in designing current and next-generation reliable microprocessors. Due to their large share of the transistor budget and die area, cache memories suffer from an increasing vulnerability against soft errors. Previous work based on the vulnerability factor (VF) analysis proposed analytical models to evaluate the reliability of on-chip data and instruction caches. However, we have no possession of a system-level study on the vulnerability of instruction caches. In this paper, we propose a new analytical model to estimate the system-level vulnerability factor for on-chip instruction caches. In our model, the error masking/detection effects in instructions based on the Instruction Set Architecture (ISA) are studied. Our experimental results using SPEC benchmark suite show that the self-error-masking/detection in instructions will reduce the VF of the instruction caches compared to the previous study. We also conduct an evaluation on the effectiveness of the reliability optimization techniques for instruction caches under our system-level VF characterization. Our proposed vulnerability model can provide an insightful guidance for the reliable instruction cache and ISA design.

Keywords

Instruction Cache, Reliability, Soft Errors, Vulnerability Factor.

I. INTRODUCTION

Ionizing radiation induced soft errors in semiconductor memories have been recognized for a long time as a major reliability issue in electronic systems [1][2]. Due to their large share of the transistor budget and die area, on-chip caches suffer from a significantly higher soft error rate (SER) than other on-chip components at the current and future technologies [3]. Incorrect data values/instructions once read out from the data/instruction caches may crash the subsequent computation/communication, external memory, or storage systems, leading to overall system failures or program inaccuracy. As a critical requirement for reliable computing [4], protecting the information integrity in cache memories has captured a wealth of research efforts [4][5][6][7][8][9][10][11][12][13][14].

Recent work has made progress towards the cache vulnerability analysis and reliability optimization based on the analysis [5][11][12][14][15][16][17]. For example, early write-back schemes [8][14][15] were proposed to reduce the vulnerability factor (VF) of dirty cachelines in a write-back data cache. [12] proposed a clean cacheline invalidation (CCI) scheme in data and instruction caches to improve their reliability. Information redundancy is another fundamental approach in building reliable memory structures. Various coding schemes, such as the parity and ECC codings, are used to enhance information integrity in latches, register files, and on-chip caches, providing different levels of reliability at different performance, energy, and hardware costs. Another form of information redundancy is to maintain redundant copies of the data items in the cache memories [9][18]. In these schemes, the cachelines in the data array are duplicated when they are brought into L1 caches on read/write misses or on write operations. However, maintaining redundant copies of cachelines presents great challenges to the effective bandwidth and energy dissipation of caches [4][18]. Note that soft errors in memory structures are not related to the correctness of the design. Therefore, they cannot be captured by formal verification or testing. Furthermore, soft errors are extremely difficult to predict due to the random nature of their occurrences, which makes cost-effective reliable processor designs against soft errors an increasing challenge. In [19], Mukherjee et al. proposed an architectural vulnerability factor (AVF) for reliability quantification based on whether each bit during execution will affect the final system output. To provide an accurate upper bound estimation for AVF, a temporal vulnerability factor (TVF) was proposed in [12] for both data and instruction cache vulnerability characterization.

Despite the fact that previous work has conducted some studies on the vulnerability factor characterization of the on-chip caches [11][12][14][15], we have no possession of a system-level vulnerability study on instruction caches against soft errors. In this paper, we first analyze the instruction cache vulnerability based on the previously proposed TVF [12] model. After
the detailed study on the error masking/detection effects of the Instruction Set Architecture (ISA) to the instruction caches, we propose our System-level Instruction Cache Vulnerability Factor (SICVF) model. To conduct a further study on our proposed model, we apply the SICVF analysis on an instruction cache reliability optimization scheme, the Clean Cacheline Invalidation (CCI) [12]. The experimental results confirm that our study should be able to provide enough insight into the instruction cache reliability issues, which can be taken advantage of to design highly cost-effective reliable microprocessors.

The rest of the paper is organized as follows. The next section discusses the background and related work. Section III describes instruction cache vulnerability characterization based on the TVF model. In Section IV, we study different error masking/detection effects of the instruction set architecture to instruction caches and propose our new system-level instruction cache vulnerability factor model. The experimental setup and evaluation are presented in Section V. Section VI concludes this work.

II. BACKGROUND AND RELATED WORK

Soft Error Rate (SER) is an error rate metric for the system vulnerability due to soft errors. Failures in Time (FIT) is another widely-used error rate metric, which is inversely proportional to Mean Time to Failure (MTTF). The FIT rate of a component or a system is the number of failures it incurs over one billion \((10^9)\) hours. One of the advantages of using the FIT metric is that the FIT rates can be added in an intuitive fashion. Therefore, the FIT rate of a system can be calculated according to the following equation.

\[
FIT_{\text{System}} = \sum_i FIT_{\text{Components}}
\]  

(1)

Architectural Vulnerability Factor (AVF) [19] is a recently developed metric that provides insight into the structural vulnerability to soft errors. AVF can be used to scale a raw FIT rate into an effective FIT rate. The effective FIT can be calculated as follow:

\[
FIT_{\text{effective}} = FIT_{\text{raw}} \times AVF
\]  

(2)

In [12], a Temporal Vulnerability Factor (TVF) was proposed to analyze the vulnerability of both on-chip data and instruction caches against soft errors. Different from AVF, TVF captured the upper bound of the cache vulnerability factor. However, TVF did not provide a system-level characterization on the cache vulnerability. Haghdoost et al. [20] extended the TVF model to estimate the system-level vulnerability factor of both write-through and write-back data caches by taking account of the read frequency and ALU masking. In this paper, we propose a system-level vulnerability model for instruction caches based on the error masking/detection effects of the instruction set architecture. Compared to TVF, which only considered the temporal vulnerability within the instruction cache itself, our System-level Instruction Cache Vulnerability Factor (SICVF) also takes into account the error masking/detection effects after the instructions being fetched out of the instruction cache and during the execution. The idea is based on that the error bit in an instruction may be masked or detected due to the design of the ISA.

III. TVF BASED INSTRUCTION CACHE VULNERABILITY CHARACTERIZATION

To characterize the vulnerability of the on-chip instruction cache, we first analyze the vulnerability factor by only considering the temporal behavior of the instruction cache itself. We utilize the TVF model proposed in [12]. In this lifetime model, the lifetime of a data item in the instruction cache is divided into three phases according to the previous activity and the current one. They are:

- **RR**: lifetime phase between two consecutive reads of a data item,
- **RPL**: lifetime phase between the last read and the replacement of a data item,
- **Invalid**: lifetime phase when the data item is in the invalid state.

![Figure 1. The lifetime of a data item in the instruction cache.](image-url)
Figure 1 shows the correlation among three lifetime phases for typical instruction cache activities. These three phases are further categorized into two groups, vulnerable and nonvulnerable. The vulnerable phase is defined by the fact that errors occurring in this phase have the possibility to propagate to the CPU. The RR phase is vulnerable phase since error occurring in this phase will propagate to the CPU by the instruction fetch. The RPL and Invalid are nonvulnerable since the errors occurring during these two phases will be discarded. Note that the data item in the instruction cache can be a cacheline, a 32-bit instruction, or a single bit. It is different from the data item in the data cache where the data item normally refers to a cacheline, a word, or a byte. Since all data items accessed in the instruction cache are of the same size, which is the 32-bit instruction in our simulated Alpha processor, we can choose the instruction-based (32-bit) characterization in our TVF study. Although the instruction-based characterization is accurate enough for the vulnerability analysis within the instruction cache, it will become inaccurate for the system-level vulnerability estimation, where we need to track every bit in an instruction for its vulnerability. Therefore, a bit-based characterization will be performed for our SICVF model. The details will be elaborated in the following section.

IV. SYSTEM-LEVEL INSTRUCTION CACHE VULNERABILITY CHARACTERIZATION

The previous section discussed a conventional instruction cache vulnerability characterization model based on the TVF analysis. However, this model only provides an upper bound for the vulnerability factor of the instruction cache [12]. In order to provide more comprehensive understanding on the vulnerability of instruction caches, a system-level vulnerability analysis for instruction caches is needed.

A. Error Detection based on Default Values

For a certain instruction set architecture, not all the bits in an instruction are used all the time. For example, in the Alpha ISA, there are some SBZ (should be zero) bits in certain instruction formats [21]. Figure 2 shows the 3-bit SBZ in the Alpha operate instruction format. In the Alpha ISA, the operate instruction format is used to perform integer register to integer register operations. It has one destination operand ($R_c$) and two source operands ($R_a$, $R_b$, or LIT). One of the source operands must be an integer register ($R_a$). The other can be either an integer register ($R_b$) or a literal constant (LIT). Bit 12 is used to distinguish between these two cases. In the case of two integer register operands shown in Figure 2 (a), bit 15-13 are SBZ, which means that they should be zero in this instruction format. If an error occurs in the SBZ bits of an instruction during the temporal vulnerable phase, the error will be detected (if the SBZ bits are checked during the instruction decoding) or just be discarded (masked). Therefore, the SBZ bits in the instruction cache will not contribute to ACE (architecturally correct execution) bits, i.e., they have no impact on the correctness of the instruction execution and program output. Thus, in the system-level characterization, the SBZ bits are nonvulnerable.

Besides the SBZ bits, there are other bits in an instruction with default values. In some instructions, one of the source operands is not needed and is set to a default value. For example, in the sign extend instruction, only one source operand is needed. In the Alpha ISA, the SEXTB and SEXTW are sign extend instructions. The byte or word in register $R_b$ is sign-extended to 64 bits and written to register $R_c$. $R_a$ must be the integer register $R_{31}$. Similar to the SBZ bits, if these $R_a$ bits are error corrupted, the error will be either detected or ignored. Therefore, such bits with a default value in the instruction are also nonvulnerable bits.
B. Error Detection based on Hamming-Distance-One (HDO) Analysis

Given the current low soft error rate (SER) in real world situations, single-bit error was assumed in the previous cache memory vulnerability studies [12][22][23]. If the single-bit error is assumed, only one bit in the instruction may be flipped by the soft error. Based on the observation that in most ISA, not all the combinations of the bits in the Opcode field are used to designate a certain instruction, we propose a Hamming-Distance-One (HDO) based analysis to determine whether a certain bit in the instruction is vulnerable or not. For example, if a single bit in the Opcode field is flipped by the soft error, this bit will contribute to ACE only if the error corrupted Opcode becomes another legal Opcode for another instruction. If the error corrupted Opcode turns into an illegal one, the error will be detected during the instruction decoding and can be corrected by re-fetching the instruction from lower memory hierarchy. Figure 2 and 3 show different instruction formats for the Alpha ISA. For operate instruction format, memory instruction with function code format, and floating-point operate instruction format, Alpha uses the Opcode and Function bits together to designate a certain instruction. For example, the 10.00 (Opcode.Function in hexadecimal) designates the ADDL (Longword Add) instruction. If the least significant bit (LSB) in the Function field of the ADDL instruction is flipped by the soft error, it will become 10.01, which is not defined (illegal) in the Alpha ISA. This error will be detected during the instruction decoding. Thus, it contributes to the un-ACE bit. If the second LSB in the Function field of the ADDL instruction is flipped by the soft error, it will become 10.02, which is the S4ADDL (Scaled Longword Add) instruction. Thus, this bit contributes to the ACE bit. Note that all the bits in theRa, Rb, Rc, Memory_disp, and Branch_disp fields without an default value are ACE bits.

C. System-level Instruction Cache Vulnerability Factor (SICVF)

By combined the TVF analysis and the error masking/detection effects of the instruction set architecture, we propose the System-level Instruction Cache Vulnerability Factor (SICVF), defined as follow:

\[
SICVF_{Cache} = \frac{\sum_{i} \left(\sum_{j} vul_{data\_itemi} \cdot vul\_phasej\right)}{\sum_{i} (data\_itemi \cdot Exec\_Time)}
\]  

where vul_data_itemi is the vulnerable (ACE) bit in an instruction, vul_phasej is the time of jth vulnerable phase of vul_data_itemi, data_item means every bit in the instruction cache, and Exec_Time is the total time simulated for the benchmark. Note that different from the TVF proposed in [12], only the bit-based characterization is used in SICVF definition, since we analyze every bit in the instruction cache for its system-level vulnerability. The data item calculated here is also different from the TVF model, where all the data items are profiling for the temporal vulnerability. In our SICVF, by considering the error masking/detection effects of the instruction set architecture, only the vulnerable (ACE) bits in an instruction are calculated towards the vulnerability factor. Intuitively, our SICVF will be less than the TVF in [12] with more accuracy on the system level.

V. EVALUATION

A. Experimental Setup

We derive our simulators from SimpleScalar V3.0 [24] 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 the active list (a.k.a. the re-order buffer). Table I gives the detailed configuration of the simulated microprocessor.

For experimental evaluation, we use the SPEC CPU2000 benchmark suite compiled for the Alpha Instruction Set Architecture (ISA) 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 specified by SimPoint [25]. 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.

B. TVF

To compare our SICVF characterization with the conventional vulnerability analysis, we first calculate the temporal vulnerability factor of the instruction cache for our simulated microprocessor. Figure 4 shows that the TVF of the instruction cache (RR vulnerable phase) is about 16.2 percent. Note that in order to be consistent with the following SICVF characterization, we use the bit-based analysis in our TVF study.
C. SICVF

Based on the TVF characterization, we further track every bit in each instruction during the execution to determine whether it is a default value error detection (DV-ED) bit or a HDO error detection (HDO-ED) bit. From our system-level analysis, the vulnerable RR phases of these bits (DV-ED and HDO-ED bits) do not contribute to the SICVF. Figure 4 shows that the SICVF of the instruction cache is around 14.3 percent. Therefore, our system-level characterization model incurs a 13.7 percent reduction on the average in the vulnerability factor estimation.

To further study the details of our SICVF model, a breakdown for the percentage of the DV-ED and HDO-ED contributing to the nonvulnerable bits is profiled. Figure 5 shows that around 19.9 percent of the nonvulnerable bits is DV-ED and the rest 80.1 percent is contributed by HDO-ED.

D. Clean Cacheline Invalidation (CCI)

Based on the observation that most read-read (RR) instances have small intervals (less than 1K cycles) and these RR instances with small intervals only contribute a small percent of the overall RR time, [12] proposed a clean cacheline invalidation (CCI) scheme to reduce the vulnerability factor of the instruction cache by invalidating the cachelines after being idle for some predefined intervals. By carefully choosing the invalidation interval, the CCI scheme can improve the reliability of the instruction cache with a negligible performance overhead. Considering the performance overhead, we choose a 16K invalidation interval in our study. Figure 6 shows that the CCI scheme (16K) reduces the TVF of the instruction cache to 8.0 percent with only the performance loss under 0.9 percent.

We further evaluate the CCI scheme by using our new proposed SICVF model. To compare with the TVF model, we also use 16K invalidation interval for SICVF. Our experimental results show that the CCI scheme reduces the SICVF from 14.3 percent to 7.0 percent, as shown in Figure 4 and 6. It confirms us that the CCI is also an effective scheme for the system-level vulnerability optimization in instruction caches.

VI. CONCLUSIONS

In this paper, we propose a SICVF model to characterize the system-level vulnerability of instruction caches. Compared to the previous temporal vulnerability factor (TVF) model, our SICVF takes into account the error masking/detection effects of the instruction set architecture. We divide the error masking/detection effects into two categories, the DV-ED (default value error detection) and HDO-ED (Hamming-distance-one error detection). Our experimental results show that error masking/detection in the instructions will further reduce the vulnerability factor on the system level compared to the previous study. We also evaluate the effectiveness of an instruction cache reliability optimization technique, the clean cacheline invalidation (CCI), under the system-level instruction cache VF characterization. Results confirm that our proposed model will provide an insightful guidance for the reliable instruction cache and ISA design.
REFERENCES


vulnerability factors for a high-performance microprocessor,” in Proc. of the 36th Annual IEEE/ACM International Symposium on
Microarchitecture, December 2003.
Support for Programming Languages and Operating Systems, October 2002.