On the characterization and optimization of system-level vulnerability for instruction caches in embedded processors

Shuai Wang*, Guangshan Duan

State Key Laboratory of Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, Jiang Su, China

A R T I C L E   I N F O

Article history:
Available online 8 October 2015

Keywords:
Embedded processors
Instruction cache
Reliability
Soft errors
Vulnerability factor

A B S T R A C T

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 embedded processors. In our model, the error masking/detection effects in instructions based on the Instruction Set Architecture (ISA) are studied. Our experimental results show that the self-error-masking/detection in instructions will reduce the VF of the instruction caches compared to the previous study. We also exemplify our design methodology by proposing several optimizing schemes to improve the reliability. Benchmarking is carried out to demonstrate the effectiveness of our vulnerability model and optimization approach, which can provide an insightful guidance for the future reliable instruction cache and ISA design.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

Ionizing radiation induced soft errors in semiconductor memories have been recognized for a long time as a major reliability issue in electronic systems [2,3]. 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 [4]. 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 [5], protecting the information integrity in cache memories has captured a wealth of research efforts [5–16].

Recent work has made progress towards the cache vulnerability analysis and reliability optimization based on the analysis [6,12,13,15,17–19]. For example, early write-back schemes [9,15,17] were proposed to reduce the vulnerability factor (VF) of dirty cachelines in a write-back data cache. Wang et al. [13] 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. Some recent work [20–22] focuses on the soft error reliability in multi or heterogeneous core environment. Another form of information redundancy is to maintain redundant copies of the data items in the cache memories [10,23]. 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 [5,23]. 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 [24], 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 [13] for both data and instruction cache vulnerability characterization.

* Corresponding author. Tel.: +862589640908. E-mail address: swang@nju.edu.cn (S. Wang).

http://dx.doi.org/10.1016/j.micpro.2015.09.011
0141-9331/© 2015 Elsevier B.V. All rights reserved.
Despite the fact that previous work has conducted some studies on the vulnerability factor characterization of the on-chip caches [12,13,15,17,25], we have no possession of a system-level vulnerability study on instruction caches against soft errors, especially for embedded processors. In this paper, we first analyze the vulnerability of the instruction cache in the embedded processor based on the previously proposed TVF [13] 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. In order to further improve the reliability, we proposed several schemes to reduce the system vulnerability of the instruction cache, which can reduce the SICVF to 5.8% compared to the original 20.7%, with negligible performance overheads. 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 embedded processors.

The rest of the paper is organized as follows. The next section discusses the background and related work. Section 3 describes instruction cache vulnerability characterization based on the TVF model. In Section 4, 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. Optimizing schemes to reduce the vulnerability of the instruction cache are proposed in Section 5. The experimental setup and evaluation are presented in Section 6. Section 7 concludes this work.

2. 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.

\[ \text{FIT}_{\text{System}} = \sum \text{FIT}_{\text{Components}} \]  \hspace{1cm} (1)

Architectural Vulnerability Factor (AVF) [24] 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 follows:

\[ \text{FIT}_{\text{Effective}} = \text{FIT}_{\text{Raw}} \times \text{AVF} \]  \hspace{1cm} (2)

In [13], 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. [26] 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 (ISA) for embedded processors. 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.

Due to the power and area constraints, the reliable embedded processor design differs from those in the high-performance processor domain [27,28]. Traditional fault-tolerant techniques like triple-modular redundant (TMR) may be too costly for embedded systems [29]. Therefore, the major challenge in reliable embedded processor design is maintaining the target reliability with minimized overheads. For example, recent work [30] proposed a low-cost software scheme to improve the reliability of the commodity embedded processors. For the instruction cache in embedded processors, we also need such low-cost solutions to improve its reliability under the power and area constraints.

3. TVF based instruction cache vulnerability characterization

To characterize the vulnerability of the on-chip instruction cache in embedded processors, we first analyze the vulnerability factor by only considering the temporal behavior of the instruction cache itself. We utilize the TVF model proposed in [13]. 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.

Fig. 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 ARM embedded 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.

4. 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 [13]. In order to provide more comprehensive understanding on the vulnerability of instruction caches, a system-level vulnerability analysis for instruction caches in embedded processors is needed.
4.1. 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 ARM ISA, there are some SBZ (should be zero) bits in certain instruction formats [31]. Fig. 2(a) shows an example of the 4-bit SBZ in the MOV instruction. In the ARM ISA, the MOV instruction is to write a value to the destination register (Rd). The bit 19–21 are SBZ, which means that they should be zero in this instruction. 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. Similarly, there are SBO (should be one) bits in certain instructions in ARM ISA, as shown in Fig. 2(b).

4.2. 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 [13,32,33]. 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. Fig. 3 shows an example of the ARM instruction set format for load/store instructions. In this format, Op1 and Op2 are used together to designate a certain load/store instruction. For example, the 00000.01 (Op1.Op2 in binary) designates the STRH (Store Register Halfword) instruction. If the bit 21 in the Op1 field of the STRH instruction is flipped by the soft error, it will become 00010.01, which is not defined (illegal) in the ARM ISA. This error will be detected during the instruction decoding. Thus, it contributes to the un-ACE bit. If the bit 6 in the Op2 field of the STRH instruction is flipped by the soft error, it will become 00000.11, which is the STRD (Store Register Doubleword) instruction. Thus this bit contributes to the ACE bit. Note that all the bits in operands fields, such as R (source), R (destination), and R (base), without default values are ACE bits.

4.3. System-level instruction cache vulnerability factor (SICVF)

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

\[
SICVF_{\text{Cache}} = \frac{\sum_i v_{\text{data}_{item_i}} \times \sum_j v_{\text{data}_{item_j}} \times \text{Exec}_{\text{Time}_i}}{\sum_{\text{data}_{item}} \times \text{Exec}_{\text{Time}}}
\]

where \(v_{\text{data}_{item_i}}\) is the vulnerable (ACE) bit in an instruction, \(v_{\text{data}_{item_j}}\) is the time of jth vulnerable phase of \(v_{\text{data}_{item}}\), data item means every bit in the instruction cache, and \(\text{Exec}_{\text{Time}}\) is the total execution time for each simulated benchmark (i.e., the lifetime of each data item in each benchmark). Note that different from the TFV proposed in [13], 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 TFV model, where all the data items are profiled 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 TFV in [13] with more accuracy on the system level.

5. Vulnerability optimizations based on SICVF

5.1. Hamming-distance-two coding

From our Hamming-Distance-One analysis, if all the Opodes have Hamming-Distance-Two for all the instructions in the ISA (which means if a single bit in the Opcode field is flipped by the soft error, this instruction will become an illegal instruction.), the vulnerability of the instruction cache can be greatly improved. However, this Hamming-Distance-Two (HDT) coding is not feasible for the ISA like ARM, where most of the coding space of the Opcode field is utilized for defining usable instructions.

5.2. Opcode swapping based on usage frequency

Since redesigning the whole ARM ISA using the Hamming-Distance-Two coding is not feasible, one of possible solutions is to make a swap on the Opcode bits between a frequently used instruction with more vulnerable (ACE) bits and a less frequently used (or rarely used) instruction with less vulnerable bits. For example, instruction Instr_A is frequently used and more vulnerable to soft errors and instruction Instr_B is rarely used and less vulnerable to soft errors based on our SICVF analysis, we can swap the coding of the Opodes for these two instructions. After that, the reliability of the entire ISA will be improved.

Therefore, we proposed our Opcode Swapping based on the Usage Frequency of the Instructions (OSUF) scheme to improve the vulnerability of the ISA and instruction cache in the embedded processors.
Table 1
Parameters of the simulated ARM processor.

<table>
<thead>
<tr>
<th>Processor core</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode width</td>
<td>2 inst. per cycle</td>
</tr>
<tr>
<td>Issue width</td>
<td>2 inst. per cycle</td>
</tr>
<tr>
<td>Commit width</td>
<td>2 inst. per cycle</td>
</tr>
<tr>
<td>RBU</td>
<td>16 entries</td>
</tr>
<tr>
<td>Load/Store Queue</td>
<td>8 entries</td>
</tr>
<tr>
<td>Function units</td>
<td>2 IALU, 1 IMULT/IDIV</td>
</tr>
<tr>
<td></td>
<td>2 FALLU, 1 FMULT/FDIV</td>
</tr>
<tr>
<td></td>
<td>2 MemPorts</td>
</tr>
<tr>
<td>Branch predictor</td>
<td></td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Binmod</td>
</tr>
<tr>
<td>BTB</td>
<td>8-entry RAS</td>
</tr>
<tr>
<td>Memory hierarchy</td>
<td></td>
</tr>
<tr>
<td>L1 I/DCache</td>
<td>32KB, 4 ways, 328 blocks, 1 cycles, LRU</td>
</tr>
<tr>
<td>L2 UCache</td>
<td>256KB, 4 ways, 64B blocks, 6 cycles, LRU</td>
</tr>
<tr>
<td>Memory</td>
<td>18 cycles first chunk, 2 cycles rest</td>
</tr>
<tr>
<td>ITLB</td>
<td>Fully-assoc., 64 entries</td>
</tr>
<tr>
<td>DTLB</td>
<td>Fully-assoc., 128 entries</td>
</tr>
</tbody>
</table>

against soft errors. First, we need to do the profiling on our studied ARM processor to find the most frequently used instructions during the executions. Then, we need to conduct our SICVF analysis to identify the high vulnerable ones among these frequently used instructions. The instructions with less ACE bits in the rarely used group also need to be identified and picked up for swapping. Based on the previously study and the spirit of the RISC design, a small set of simple instructions dominate instruction frequency. Thus, if we can improve the vulnerability of these small number of frequently used instructions, the reliability of the entire ISA and instruction cache will be improved. The advantage of our OSUFI scheme is that it will only modify a small portion of the original ISA design and keep most of the remaining portions untouched, which is favorable by both hardware and software designs.

5.3. 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, [13] 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. Therefore, we propose to also adopt the CCI scheme in our optimization design in the instruction cache.

6. Evaluation

6.1. Experimental setup

We derive our simulator from SimpleScalar/ARM Tool Set [34] to model an embedded microprocessor similar to ARM Cortex-A9 [35]. This simulator implements a very detailed out-of-order issue processor with a two-level memory system and speculative execution support, and this simulator is a performance simulator, tracking the latency of all pipeline operations. Table 1 gives the detailed configuration of our simulated embedded processor. For experimental evaluation, we use the MiBench [36] benchmark suite compiled for the ARM Instruction Set Architecture (ISA). Eighteen benchmarks were randomly selected for the evaluation.

6.2. 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 embedded processor. Fig. 4 shows that the TVF of the instruction cache (RR vulnerable phase) is about 26.7%. Note that in order to be consistent with the following SICVF characterization, we use the bit-based analysis in our TVF study.

6.3. 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. Fig. 4 shows that the SICVF of the instruction cache is around 20.7%. Therefore, our system-level characterization model incurs a 22.5% reduction on 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. Fig. 5 shows that around 52% of the nonvulnerable bits is DV-ED and the rest 48% is contributed by HDO-ED.

We also conduct the experiments on the SICVF for different cache replacement policies. Fig. 6 shows the SICVF for FIFO, LRU, and Random policies. The results show that although intuitively the LRU policy will more likely replace the long-idle cacheline and reduce the vulnerability, there are no huge differences on the system level vulnerability for different replacement policies.
6.4. Vulnerability optimizations based on instruction usage frequency

In order to adopt our Opcode Swapping based on the Usage Frequency of the Instructions (OSUFI) scheme to optimize the vulnerability, we first need to identify the most frequently used instructions in our ARM ISA. Based on our profiling results, the top seventeen frequently used instructions contribute to 77.86% of the total executed instruction on average for all simulated benchmarks. Table 2 shows the details on the percentage of top seventeen instructions in simulated ARM processor. Therefore, if we can improve the vulnerability of these seventeen instructions, the overall vulnerability of the entire ISA and instruction cache will also be improved.

To improve the vulnerability of these most frequently used instructions, we need to swap them with rarely used instructions. However, not all the frequently used instructions need to be swapped. We only need to swap the instructions with high vulnerability. Based on our SICVF analysis, only fourteen of the seventeen frequently used instructions need to be swapped. Note that the instructions chosen to be swapped in the rarely used group need to have as less vulnerable bits as possible. Table 3 shows the fourteen pairs of the instructions we choose to swap in our study. After the swapping, the vulnerability of the opcode field in the instruction will be improved. Fig. 7 shows that on average after swapping the vulnerability of the opcode will be improve by 63% and the SICVF for the entire instruction cache is reduced to 14.5% as shown in Fig. 10.

6.5. Clean cacheline invalidation (CCI)

For the clean cacheline invalidation (CCI) scheme, as shown in Fig. 8, we profiled the number of instances with two consecutive reads to the 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 99% of them, have an interval less than 16K cycles. However, our results also show that a small number of read-read instances with intervals (≥16 K cycles) dominate the overall RR time, 78.1% on the average. The profiler 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 Fig. 9 show that 16K cycles is a good choice for this clean cacheline invalidation. The performance loss is only 0.3%.

We further evaluate our proposed OSUFI scheme and the CCI scheme by using the new SICVF model. Based on the study above, we
use 16K invalidation interval for the CCI scheme. Our experimental results show that by combining the OSUFI and CCI schemes, the SICVF of the instruction cache in the embedded processor can be reduced to 5.8% with negligible 0.3% performance degradation, as shown in Figs. 10 and 9. It confirms us that the proposed schemes are very effective for the system-level vulnerability optimization in instruction caches.

6.6. Statistical error injection

In order to evaluate our SICVF metric and the optimizing design, we also conduct the statistical error injection during the execution-driven simulation. The soft error injection flips one bit or multiple bits in the selected cacheline. For each benchmark, 100 errors are randomly injected into the instruction cache during the execution. Notice that errors injected in our simulation are accelerated and with higher rate than real ones. As a general way to perform architectural-level error injection [9], accelerated error numbers are assumed in order to expose the error behavior and evaluate the reliability of the system. Although the accelerated error number is adopted, there are no double/multiple-bit errors occurring in our simulation.

We defined our Silent Data Corruption (SDC) rate as the total number of injected errors that cannot be detected during the execution and contribute to the ACE bits over the total number of injected errors. Fig. 11 shows the SDC rates for the original instruction cache and our optimized instruction cache with OSUFI-only and OSUFI+CCI schemes. The results also confirm the effectiveness of our design and the achieved reliability.

7. Conclusions

In this paper, we propose a SICVF model to characterize the system-level vulnerability of instruction caches in the embedded processors. 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 instruction set architecture will further reduce the vulnerability factor on the system level compared to the previous study. We also proposed several instruction cache reliability optimization techniques based on our system-level vulnerability factor characterization. Results confirm that our proposed model and optimization designs will provide an insightful guidance for the future reliable instruction cache and ISA design.

Acknowledgments

A preliminary version of this work [1] appeared in the proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems. We have extended the work (i) by re-conducting the entire study on the ARM based embedded processor instead of the original Alpha processor, (ii) by proposing several optimizing schemes including the Opcode Swapping based on the Usage Frequency of the Instructions (OSUFI) scheme to improve the vulnerability of the instruction cache, and (iii) by evaluating the effectiveness of our System-level Instruction Cache Vulnerability Factor (SICVF) analysis and optimizing schemes through error injection.

References


