A PGM based multi-level reliability analysis method for Data Cache

Jiajia Jiao¹, Xing Han², and Yuzhuo Fu²
¹ College of Information Engineering, Shanghai Maritime University, Shanghai 201306
² School of Microelectronics, Shanghai Jiao Tong University, Shanghai 200240
a) jiaojiajia@shmtu.edu.cn

Abstract: With scaling technology node, soft error has dominated in the integrated circuit failure. To tradeoff the design cost and reliability, efficient reliability analysis methods are required to select the appropriate reliable schemes. In this paper, we propose a multi-level Probabilistic Graphical Models (PGM) based method for the soft error analysis of data Cache structure. The proposed method includes two points: 1) Exacting the error masking dependencies from the ALU instruction execution procedure, and calculating the first-level masking rate between ALU and register file; 2) Drawing the error spreading dependencies graph from the load and store instruction processing, and computing the higher-level masking rate between ALU and Data Cache. The simulation results of SPEC2K demonstrate that, compared with the existing methods, the proposed method achieve up to 16.19% accuracy improvement and 52.72× speedup.

Keywords: soft error, PGM, Data Cache, error masking, error spreading

Classification: Electron devices, circuits, and systems

References

[1] R. C. Baumann: IEEE Trans. Device Mater. Rel. 5 (2005) 305. DOI:10.1109/TDMR.2005.853449
[2] K. Constantinides and S. Plaza: High-Performance Computer Architecture (2006) 5. DOI:10.1109/HPCA.2006.1598108
[3] M. Maniatakos, C. Tirumurti and A. Jas: 2011 16th IEEE European Test Symposium (ETS) (2011) 87. DOI:10.1109/ETS.2011.42
[4] A. Biswas, R. Cheveresan, J. Emer and S. S. Mukherjee: International Symposium on Computer Architecture (2005) 532. DOI:10.1145/1080695.1070014
[5] A. Haghdoost and H. Asadi: IEEE Dependable Computing (PRDC) (2010) 157. DOI:10.1109/PRDC.2010.9
[6] Y. Cheng, A. Ma, Y. Wang, Y. Tang and M. Zhang: J. Central South Univ. 19 (2012) 3129. DOI:10.1007/s11771-012-1388-4
[7] J. Jiao, D. C. Juan, D. Marculescu and Y. Fu: Microelectron. Reliab. 55 (2015) 251. DOI:10.1016/j.microrel.2014.09.011
[8] S. Z. Shazli, M. Abdul-Aziz, M. B. Tahoori and D. R. Kaeli: IEEE International Test Conference (2008) 1. DOI:10.1109/TEST.2008.4700615
[9] J. Jiao and Y. Fu: 2014 IEEE 23rd Asian Test Symposium (ATS) (2014) 33. DOI:10.1109/ATS.2014.18
1 Introduction

Soft errors, also known as transient faults or single-event upset, are caused by external radiation or electrical noise. Due to lower supply voltages and higher integration density, the soft error rate dramatically increases as the technology node scales down [1]. To effectively tradeoff the design cost (e.g., area) with higher reliability, accurate and efficient estimation of soft error impacts is required at an early design stage [2]. Such estimates, conventionally, are used to identify the components with a high vulnerability to soft errors, and thereby system designers can effectively deploy mitigation strategies to minimize the soft error impacts without introducing much design overhead.

Architectural Vulnerability Factor (AVF), a widely-used metric to quantify the reliability of a processor, represents the probability that a single bit upset will result in a user-visible error in the final output. AVF estimation is usually done by using (1) fault injection (FI) via Monte Carlo simulation. A large number of simulations will be required if one wants to obtain an accurate estimation of AVF, which is very inefficient due to long simulation time (usually up to days) [3]; and (2) utilization-based fault free analysis such as Architecturally-Correct Execution (ACE) analysis [4]. For example in Fig. 1, the period between Write and Read is an ACE piece, where a soft error may have harmful effects on correct execution. Otherwise, the unACE piece between Read and Write is not harmful. ACE provides an alternative to estimate AVF by using only one (or at most two) of simulation. Compared to FI, ACE is faster. However, to make a valid estimation, these methods need to be conservative, which leads to a pessimistic estimation: on average the AVF estimated by ACE is 2–3 times higher than the AVF estimated by FI.

To tradeoff the accuracy and speed well, Haghdoost A. et al. proposed a mixed method of FI and ACE for characterizing the soft errors impacts on Cache structure [5]. And Cheng Y. et al also analyzed the partial masking rate for a more effective evaluation [6]. In the meanwhile, the effective Probabilistic Graphical Models (PGM) based method is applied into soft error analysis of register file in our previous work [7]. Therefore, we are inspired to extend the PGM based reliability analysis region to data Cache for the guaranteed estimation efficiency.

2 Problem formulations

Cache structure in processors is demonstrated to be the extremely sensitive components, where 92% of system reboots are initiated by soft errors [8]. Therefore, we focus on an effective method for evaluating cache structure.

![Fig. 1. ACE pieces for cache structure based on access types](image-url)
The formulation to compute AVF of Data Cache is described as follows. First, the following four input are given (1) target architecture (e.g., alpha processor); (2) workloads (e.g., SPEC2000 (INT)); (3) the component size (e.g., 64 KB L1Data Cache); (4) each ACE piece in cycles and the total number of ACE pieces. The decision variable is the masking rate for each ACE piece, which depends on the method used. The objective is to use the more accurate masking rate for an efficient AVF estimation using PGM methodology.

3 PGM based multi-level reliability analysis

According to the soft error propagation process, we divide the reliability analysis of Cache structure into two stages: 1) one is related to error masking effects between storage elements and ALU; 2) the other is only the error spreading between different storage structures (register file and L1 Cache, L1 Cache and L2 Cache, and even L2Cache and L3Cache). Here, considering the similarity of error spreading in cache hierarchy, we take L1Data Cache as a case study in this paper.

First, the first-level masking rate calculation of L1Data Cache is based on stage one of reliability analysis. The load instructions are the start points of transferring data from L1Data Cache to register file. Therefore, we gives the PGM representation of load instruction in Fig. 2(b) from the original instruction execution in Fig. 2(a). PGM includes three factors: representation, structure and node learning, inference. One ACE piece in Fig. 2(a) is a random variable in PGM structure in Fig. 2(b) and the instruction masking ability is expressed in PGM node parameters. The inference algorithm is used to calculate the marginal probability (the masking rate). Unlike the exact policy in the previous work [7], we propose an approximate PGM method to compute the first-level average masking rate fast. The key difference is to use several parameters to construct the Bayesian network fast like [9].

Error Masking Depth (EMD), denotes the count of error masking instructions. It represents the right branch depth in red of each node in Fig. 2. Error Propagation Width (EPW), means the count of sequential ACE pieces and shows the left branch width in green of each node. It is easy to statistic the average value $EPW_{avg}$, but $EMD$ is hard to capture directly. Here, we first can get the maximum $EMD$ (denoted as $Max_{EMD}$) can be estimated by the percent of ACE pieces related to store instructions (RSI) fast. The store instruction is assumed as the error observation point, which means if an error appears in a store instruction, the error masking
effects analysis will end. Generally, each output comes from two input operands. So the maximum \(EMD\) is computed by Eq. (1), where \(RSI\) is achieved by the statistical computation. \(RSI\) is the total store instruction count over the total instruction count. If \(Max_{EMD}\) is 3 and \(EPW_{avg}\) is 2, we can get the serial approximate structures in Fig. 3. Thus, the node learning in [7] and VE (Variable Elimination) inference are used for the first-level average masking rate in Eq. (2).

\[
\frac{RSI \ast (1 + 2 + 2^2 + \ldots + 2^{Max_{EMD}}) = 1}{(1)}
\]

\[
MR_{level-1} = \left( \sum_{i=0}^{[Max_{EMD}]-1} RSI \ast 2^i \ast (1 - (1 - IMR_{avg}^{EPW_{avg}})^{(i+1)}) \right) + \left( 1 - \sum_{i=0}^{[Max_{EMD}]-1} RSI \ast 2^i \ast (1 - (1 - IMR_{avg}^{EPW_{avg}})^{Max_{EMD}}) \right) \ .
\]

Then, the masking rate \(MR_{level-2}\) between L1Cache and ALU depends on the first-level masking rate \(MR_{level-1}\), as well as the error spreading between register file and L1Data Cache. As Fig. 4 shows, the cache access is mapped into PGM structure via converting one ACE piece into a random variable. The error spreading dependency behaves a line structure with unique parameter \(EPW\), while \(MR_{level-1}\) is the key to node parameter learning. Furthermore, we use the same average \(EPW_{avg-2}\) as stage1 to compute the new AVF via calling VE inference in Eq. (3).

\[
MR_{level-2} = MR_{level-1}^{EPW_{avg-2}}
\]
Finally, the new AVF value is the product of \((1 - MR_{level-2})\) and original AVF for L1 Data Cache. Like L1 Data Cache, the reliability analysis of other Cache structure like L2, L3Cache can be done by repeating the higher-level masking rate.

### 4 Simulations and analysis

We use sim-soda [10] for our proposed method implementation. FDD/TDD instruction identification is post-commit-analysis based on a 40 K size instruction chain. We add one pre-simulation to track all the instruction numbers of FDD or TDD and then, we reload the list of FDD/TDD to initialize the instruction masking rate to 1. Second, the FI approach is implemented, where the md5sumcode of fault injected case (output data and address of all store instructions) is compared with that of no error case to estimation the average masking rate in [5]. This is repeated 1000 times. Table I shows the simulation configuration. And we use SPEC2000 (INT) suite for all results. To evaluate the proposed method, we compare its AVF and time cost with the existing works in [5, 6] (denoted ‘ref[5]’ and ‘ref[6]’).

Firstly, Fig. 5(a) show that our proposed method and ‘ref[5]’ provide the more accurate AVF than ‘ref[6]’. Because the two methods both capture the error masking and error spread in AVF computation via PGM analysis and FI respectively. Therefore, the AVF values of the proposed method and ‘ref[5]’ have more 8.98% and 16.19% accuracy improvement than that of ACE analysis in ‘ref[6]’. Especially, among the twelve given benchmarks, ‘bzip’ using proposed method has up to 50.27% accuracy improvement due to its more frequent execution of masking instructions. Such a result reflects that ‘bzip’ analysis satisfies our assumptions of proposed approximate PGM method well, such as average policy. Furthermore, we observe that the proposed method have a more accuracy than ‘ref[5]’.

| Table I. Simulation parameters configuration |
|---------------------------------------------|
| Parameter                     | Value                                    |
| Pipeline depth                | 7                                        |
| Integer ALUs/multi            | 4/4                                      |
| Integer ALU/multi latency     | 1/7                                      |
| Fetch-slot/map/issue/commit width | 4/4/4/4/11 instructions per cycle       |
| Issue queue size              | 20                                       |
| Reorder buffer size           | 80                                       |
| Register file size            | 80                                       |
| Load/store queue size         | 32                                       |
| MSHR entries                  | 8/cache                                  |
| Pre-fetch MSHR                | entries 2/cache                          |
| Victim buffer                 | 8 entries, 1-cycle hit latency           |
| Return address stack          | 32-entry                                 |
| L1 Cache                      | 64 KB instruction/64 KB data, 2-way, 64 B line, 3-cycle latency |
| L2 Cache                      | 2 MB, direct mapped, 64 B line, 7-cycle latency    |
| TLB size                      | 128-entry ITLB/128-entry DTLB, fully-associative |
| Branch predictor              | Hybrid, 4 K global + 2-level 1 K local + 4 K choice |
| Mis-prediction penalty        | 7 cycles                                 |

Finally, the new AVF value is the product of \((1 - MR_{level-2})\) and original AVF for L1 Data Cache. Like L1 Data Cache, the reliability analysis of other Cache structure like L2, L3Cache can be done by repeating the higher-level masking rate.
PGM method, while ‘ref[5]’ has a smaller error masking coverage due to the limited 1000 times fault injection.

Then, Fig. 5(b) show the comparative results of runtime cost. As a hybrid of FI and ACE, the method ‘ref[5]’ is the slowest estimation. Its low efficiency results from the time consuming FI for average masking rate. Instead, our proposed method as well as the ACE based analysis in ‘ref[6]’ is about 52.72× and 114.82× faster than ‘ref[5]’. The extra time of proposed method over ‘ref[6]’ is consumed on the additional simulation to mark FDD/TDD instructions for instruction-level masking rate (IMR) modeling.

From a joint perspective, the proposed method takes advantages of the high speed of ACE based analysis: up to 52.72× speedup over fault injection based method in [5]. And it also captures the error masking effects completely for a more accurate estimation instead of long time simulation in mixed method: 16.19% overestimation drop than [6]. In all, the approximate PGM based method provides guaranteed accuracy improvement as well as high efficiency.

5 Conclusions

In this paper, we propose a multi-level Probabilistic Graphical Models (PGM) based method for the soft error analysis of data Cache. The proposed method can fast characterize both the error masking effects between ALU and register file, and the error spreading effects in Cache hierarchy. The simulation results of SPEC2K demonstrate that, compared with the existing methods, the proposed efficient method achieves 16.19% accuracy improvement and 52.72× speedup.

Acknowledgments

This work was supported by Innovation Program of Shanghai Municipal Education Commission (grant numbered 14ZZ018) and National Natural Science Foundation of China (grant numbered 61472244).