Position Paper

1 Introduction

Aggressive technology downscaling and wide spread use of digital systems in hazardous environments, have made reliability as one of the major concerns for processor designers. Soft errors or transient faults are considered as the main source of unreliability in modern processors, for instance, it has been demonstrated that the soft errors are hundred times more frequent than hard errors in DRAM memory cells [1]. Since soft errors occur due to environmental condition such as high-energy particles, they cannot be detected during manufacturing testing, and, since soft errors normally do not have any permanent impact on the device, employing on-line error detection/correct techniques seems to be the only way to cope with the thread of soft errors. Such fault tolerant techniques can be implemented in different abstraction-level; from circuit-level hardening design techniques till user-level code fault tolerant techniques. Among these possible solutions, compiler techniques have several attractive features such as flexibility (they can be used for some programs or even some parts of a program), and no need to hardware support is needed. Because of these properties, we believe that the soft errors problem can be solved efficiently by adopting appropriate compiler level techniques.

2 Compilers for Soft Errors

Compiler can mitigate the impact of soft error in several ways. It can apply redundancy and check strategy in different granularity, i.e, thread- or instruction- level. In these type of solutions two exactly identical streams of instructions get executed, and at some specific points of execution the redundantly computed results get checked against each other for error detection. These type of methods can be extremely effective in terms of fault detection, but they usually impose a significant runtime overhead to the system. For example, ZDC [2] an in-thread instruction-replication technique, is able to detect all soft errors in different hardware components by duplicating virtually all instructions, but it suffers from a significant (about 2.1X) execution time overhead.

Compiler can also decrease the sensitivity of a program to soft errors by taking advantage of protected hardware components. For instance, if the memory subsystem is protected by ECC, compiler can alleviate the chance of error on the value of some long-live registers by spilling them into the memory. It is also possible to alter the scheduling of instructions in a way to accelerate the execution time of the critical instructions, and therefore decrease the chance of error on them by reducing their soft-error exposure time. These type of methods can increase the program reliability in cost of negligible performance degradation. For instance, as [4] shows, by writing live variables to protected memory the registerfile vulnerability can decrease by 30% in price of 2% runtime overhead.

Compiler detectors/assertions is another way that a compiler can increase the reliability of the program's execution. Compiler detectors are checking instructions which are inserted in the program and check some dynamically calculated values against the statistically expected values. For instance, the value of the index variable after the execution of a loop can get estimated in compile time. Then, by comparing this speculated value against the runtime calculated value, soft error failure rate reduction will be possible. For example, [3] shows that by placing low-cost error detectors 90% reduction in silent data corruption is achievable in cost of 10% performance overhead.

3 Soft error failure rate estimation

Another important challenge in reliability world is how to quantify the effect of soft error on a design or how effective is a fault tolerant technique. We believe that the impact of soft error on a design or the effectiveness of a proposed technique can be quantified by meticulous vulnerability analysis rather than error-prone and time-consuming statistical fault injection. As we showed in [5], most of existing signature-based control flow checking (CFC) mechanisms not only impose a significant runtime overhead to a system, but also make the execution of a program more susceptible to soft errors. The main reason behind this is that since CFC instructions are inserted between program instructions, they actually increase the live-time of registers, and, it negatively impact the reliability of the program. In fact, inaccurate evaluation was the main reason that the inefficiency of such CFC techniques was covered for more than one decade.

In order to ease the soft-error failure rate estimation, we are developing a soft error failure rate estimation tool, called gemV, which can precisely calculate the soft-error vulnerability of a design by a single simulation run. We define bit b as vulnerable in cycle t, if and only if a bitflip on b at cycle t leads to a program failure. By using runtime information and post-simulation analysis, gemV models the effect of various masking effects, ranging from low-level micro-architectural masking till software-level masking effects, and declares the soft error vulnerability of a program execution by accumulating all vulnerable bit-cycle during the execution of a program.

References

[1] T. J. Dell. System ras implications of dram soft errors. *IBM Journal of Research and Development*, 52(3):307–314, 2008.

[2] M. Didehban and A. Shrivastava. Zdc: A compiler technique for zero silent data corruption. In *Design Automation Conference (DAC), 2016 53st ACM/EDAC/IEEE*. IEEE, 2016.

[3] S. K. S. Hari, S. V. Adve, and H. Naeimi. Low-cost program-level detectors for reducing silent data corruptions. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[4] J. Lee and A. Shrivastava. A compiler optimization to reduce soft errors in register files. In *LCTES '09: Proceedings of the 2009 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 41–49, 2009. ISBN 978-1-60558-356-3.

[5] A. Shrivastava, A. Rhisheekesan, R. Jeyapaul, and C.-J. Wu. Quantitative analysis of control flow checking mechanisms for soft errors. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.