THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Sources of Variations in Error Sensitivity of Computer Systems

FATEMEH AYATOLAHI

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2014

Sources of Variations in Error Sensitivity of Computer Systems
Fatemeh Ayatolahi

Technical report 116L
ISSN 1652-876X
Department of Computer Science and Engineering
Dependable Real-time Systems Group

Division of Computer Engineering
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 GÖTEBORG, Sweden
Phone: +46 (0)31-772 10 00

Author e-mail: fataya@chalmers.se

Printed by Chalmers Reproservice
Göteborg, Sweden 2014

Sources of Variations in Error Sensitivity of Computer Systems

Fatemeh Ayatolahi

Division of Computer Engineering, Chalmers University of Technology

ABSTRACT

Technology scaling is reducing the reliability of integrated circuits. This makes it important to provide computers with mechanisms that can detect and correct hardware errors. This thesis deals with the problem of assessing the hardware error sensitivity of computer systems. Error sensitivity, which is the likelihood that a hardware error will escape detection and produce an erroneous output, measures a system's inability to detect hardware errors. This thesis present the results of a series of fault injection experiments that investigated how error sensitivity varies for different system characteristics, including (i) the inputs processed by a program, (ii) a program's source code implementation, and (iii) the use of compiler optimizations. The study focused on the impact of transient hardware faults that result in bit errors in CPU registers and main memory locations. We investigated how the error sensitivity varies for single-bit errors vs. double-bit errors, and how error sensitivity varies with respect to machine instructions that were targeted for fault injection. The results show that the input profile and source code implementation of the investigated programs had a major impact on error sensitivity, while using different compiler optimizations caused only minor variations. There was no significant difference in error sensitivity between single-bit and double-bit errors. Finally, the error sensitivity seems to depend more on the type of data processed by an instruction than on the instruction type.

Keywords: fault injection, error sensitivity, bit flipping, fault tolerance, compiler optimization, transient fault

Acknowledgments

It is a great pleasure to express gratitude to all people supported and helped me during my good and hard days in my studies and in my life.

I would like to express my deepest gratitude to my supervisor, Johan Karlsson, for inspiring me to continue my studies in dependable computer systems field. Thank you for giving me this opportunity to work with you. Thank you for all your supports and invaluable guidance.

I would like to thank my examiner, Georgi Gaydadjiev and also Gerardo Schneider for the regular follow-up meetings to discuss direction of my studies. I would like to also thank Jan Jonsson and Sally McKee for their support.

Special thanks are due to Behrooz Sangchoolie, with whom I have the pleasure to collaborate and exchange valuable insights. Also special thanks to Domenico Di Leo for his great support and collaboration in my master thesis which inspired me to continue my studies in fault injection field. Indeed, special thanks to the countless support from Daniel Skarin and Roger Johansson to know more about Goofi-2 and troubleshoot hardware and compiler problems. I would also like to thank Raul Barbosa and Jonny Vinetr for their valuable discussions about fault injection.

I would like take this opportunity to thank BeSafe project team Mafijul Md. Islam1, Daniel Skarin, Jonny Vinter, Fredrik Törner, Andreas Käck, Mattias Nyberg, Johan Haraldsson, Patrik Isaksson, Mats Olsson, for interesting discussions, valuable feedback, and joyful meetings on benchmarking of functional safety in the automotive industry and ISO26262.

Many thanks to my friends and colleagues in department, I would like to men-

tion all names but I'm afraid to miss some, my colleagues at 4th floor (computer engineering division), PhD Council, PhD fika, persian fika!, TA teams,... you all made this department a great place to work and have fun!
Special thanks to management and administrative supports in the department: Tiina, Peter, Marianne, Eva, Peder, Rolf, ... thanks to make this place full of energy, fun, work and fika!
Life is too short to be anything but happy, to be anything but you!
Special thanks definitely goes to friends and family. My dear Shabneshini friends, fika\lunch Group, my friends from childhood, school, university, master studies, summer schools, conferences, who are now all around the world! thanks for being so kind, caring, encouraging and understanding.
My dear wonderful mom, I am blessed to have been able to look up to you, as a strong, independent, diligent woman. Thanks for constant inspiring and being such an amazing hero in my life. My dear wonderful dad, thanks for all your efforts, caring and being a constant support in every occasion in my life. My lovely sister and brother, my cute nieces and nephews, thanks for all joy, fun and support.
Dearest Sadegh, thanks for your love, your unbelievable patience, your kindness and your amazing support, not only in studies, not only these 2 years,....

Fatemeh Ayatolahi
Göteborg, May 2014

List of Appended Papers

I Domenico Di Leo, **Fatemeh Ayatolahi**, Behrooz Sangchoolie, Johan Karlsson, Roger Johansson, "On the Impact of Hardware Faults — An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions," in *Proceedings of the* 31^{st} *International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2012)*, Magdeburg, Germany, 25-28 September, 2012.

II Behrooz Sangchoolie, **Fatemeh Ayatolahi**, Raul Barbosa, Roger Johansson, Johan Karlsson, "Benchmarking the Hardware Error Sensitivity of Machine Instructions," in 9^{th} *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE-9)*, Stanford, USA, 26-27 March, 2013.

III **Fatemeh Ayatolahi**, Behrooz Sangchoolie, Roger Johansson, Johan Karlsson, "A Study of the Impact of Single Bit-Flip and Double Bit-Flip Errors on Program Execution," in *Proceedings of the* 32^{nd} *International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2013)*, Toulouse, France, 24-27 September, 2013.

IV Behrooz Sangchoolie, **Fatemeh Ayatolahi**, Raul Barbosa, Roger Johansson, Johan Karlsson, "A Study of the Impact of Bit-flip Errors on Programs Compiled with Different Optimization Levels," in *Proceedings of the* 10^{th} *European Dependable Computing Conference (EDCC 2014)*, Newcastle upon Tyne, UK, May 13-16, 2014.

Contents

Part I

INTRODUCTION

1 Introduction

Technology scaling is making microprocessors and other integrated circuits more and more susceptible to radiation induced soft errors and aging faults [6]. This has made it increasingly important to provide computer systems with mechanisms that can detect and correct hardware errors. Since classical approaches to hardware fault tolerance are too expensive in terms of overhead for many applications, development of low-cost technique for hardware fault tolerance is currently an important field of research.

Techniques that aim to reduce the cost of redundancy often rely on cross-layer approaches [7, 9, 14, 21], which distribute the responsibility for tolerating errors over different layers of the system stack. Figure 1.1 shows a model of cross-layer fault tolerance adopted from [4] with three layers. Other authors have presented more detailed models where the hardware, software and system

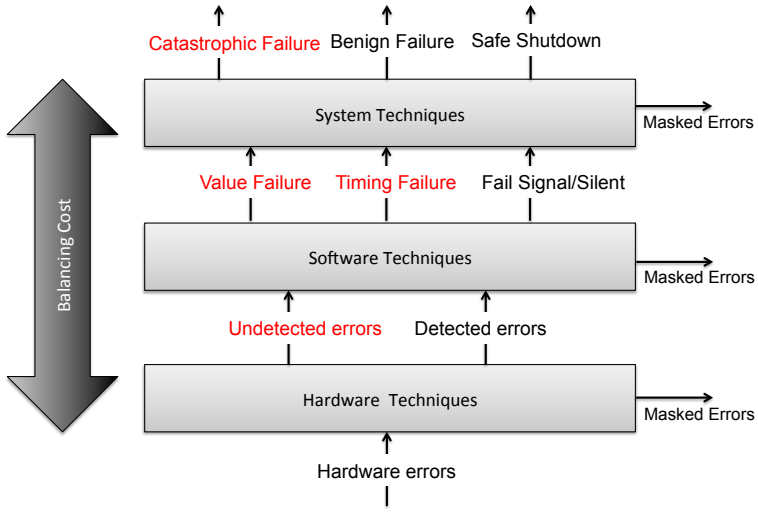

Figure 1.1: *Cross-layer fault tolerance approach*

layers are subdivided into several sub-layers.

A major benefit of cross-layer approaches is that they allow error coverage to be tuned to the needs of different applications. Such tuning typically involves evaluating the error coverage of different candidate solutions by means of fault injection experiments. However, drawing accurate conclusions based on the results of fault injection experiments concerning the relative merits of different design solutions can be difficult. The reason for this is that results of fault injection experiments can vary substantially depending on the configuration of the target system and the design of the experiments. A good understanding of sources of variation in the results of fault injection experiments is therefore essential for researchers and engineers who assess and compare different low-cost approaches to hardware fault tolerance.

This thesis presents a study of sources of variation in error sensitivity. We define error sensitivity as the probability that a computer will produce an erroneous result without any error indication [1] as a result of a hardware error.

[1] A.K.A silent data corruption (SDC)

The study specifically addresses transient faults that result in bit errors in CPU-registers and main memory locations. We investigate the following sources of variation:

- *Inputs processed by a program.*The inputs processed by a program determines the sequence of dynamic instructions and thereby error propagation process. Error sensitivity therefore depends on the inputs processed by program.

- *Types of machine instructions of a workload.* We define different categories of assembly instructions such as load, store, arithmetic, branch, logical, and move. We investigate how the mix of the instruction types influences the error sensitivity for different programs.

- *Fault model.* We investigate variations in the impact of single and double bit errors. We also study the impact of the location of bit errors within a register or memory word.

- *Source code implementation.* The programming style, i.e., the way in which a programmer implements a program specification, determines the structure of the executable program. This means that two programs implementing the same functionality may have different error sensitivities.

- *Level of compiler optimization.* Different compiler optimizations generate different executable programs. Compiler optimizations can therefore affect error sensitivity.

The study is based on a series fault injection experiments with several benchmark programs from the MiBench suite [1]. All fault injection campaigns are performed by the Goofi-2 fault injection tool [29], which was designed and implemented in our department. We use nexus-based fault injection to control and inject bit-flip errors in CPU-registers and main memory words. The target programs were compiled using the GCC compiler and executed on an MPC565 microcontroller from Freescale.

The remainder of this chapter is organized as follows. Section 1.1 gives an overview of related work. Section 1.2 presents our research questions, while Section 1.3 provides a summary of the appended papers. Our conclusions and a discussion of future work are presented in Section 1.4.

1.1 Related Work

Various fault injection tools have been developed in the past decades to assess dependability properties of computer systems. Popular fault injection techniques include pin level injection [19], software implemented [16], fault injection via debug interfaces such as Nexus [29, 34], hardware implemented [11], and simulation-based [33]. Recent studies try to make fault injection tools smarter. These studies propose techniques that make it easier to find weaknesses in a program by performing less fault injection experiments [5, 18].

Numerous studies [3, 16, 19] have assessed the effectiveness of hardware detection mechanisms using different fault models (such as pin level injection, stuck at byte, and bit flipping). In addition, an emerging research trend focuses on the implementation of software techniques to tolerate hardware errors. Different implementation of software techniques at source level [2, 24] as well as at the assembly levels [20, 25] has been assessed. These studies targeted a large variety of workloads and fault tolerance mechanisms. More recently, researchers have started to investigate cross-layer approaches that combines hardware and software techniques [7, 9, 14, 21]. An assessment of the effectiveness of these techniques are discussed in [21] and a solution for High-performance computing (HPC) is presented in [14].

A method for calculating the confidence interval for estimates obtained by fault injection experiments is presented in [17]. Powell et al. discuss coverage estimation issues related to stratified sampling [8, 23]. A general analysis of sources of uncertainty in measurements is provided in [13]. These sources of uncertainty include non-representative sampling, determinism of the target system, instrumental uncertainty, assumptions in the measurement procedure, initialization uncertainty, spatial and temporal intrusiveness, etc. These sources

of uncertainty are also highlighted in [28] where their relation to fault injection experiments and to what extent they are considered in Goofi-2 fault injection tool are discussed. These studies provide good insights about uncertainties in measurement of error sensitivity which are basically related to fault injection tool and experimental setup. However, there are few studies focusing on the sources of variations in error sensitivity with respect to the configuration of the target system and the design of the experiments. As stated in previous section, these sources of variations include input processed by a program, fault model, source code implementation, compiler optimization, etc.

With respect to input variations, in [27], matrix multiplication and selection sort are fed with three and two inputs, respectively. The fault model includes zero-a-byte, set-a-byte and two-bit compensation that differs from ours. Authors in [12] also estimated the error coverage for quicksort and shellsort, both executed with 24 different inputs. It would be beneficial to extend this work with more programs to draw conclusions about the results.

With respect to the fault model, the impact of device-level faults which manifest as single bit-flips in the CPU-registers and main memory has been studied in literature [22, 30]. However, researchers in the field of reliability physics predict that single event upsets (i.e., bit errors caused by strikes of single ionizing particles, such as cosmic neutrons) will be likely to generate multiple-bit upsets (MBUs) in circuits that will become available within a few years from now [32]. Some recent studies have targeted SRAMs and DRAMs to MBUs [26] in order to investigate geometric effect of MBU faults. In addition, the authors of [33] investigated the impact of single/multiple bit-flips in the LEON2 processor using fault injection in a VHDL simulation model. We study another level of abstraction where we mimic bit-flips in CPU-registers and memory of a real hardware platform.

Considering the impact of compiler optimization, Alexandersson et al. [2] performed a fault injection-based study on the impact of -O3 optimization on two programs equipped with different software implemented hardware fault tolerance techniques. The main focus of the paper is, however, on different fault tolerant mechanisms. Compared to their paper, not only we evaluated more

programs, but also we addressed all four optimization levels defined by GCC. There are some studies on the impact of compiler optimizations on architectural vulnerability factor (AVF). Authors in [15] made a detailed study on the impact of each compiler optimization flag on performance and AVF. They also compared the results generated by -O2 and -O3 optimizations. Authors concluded that these optimization levels decrease the performance and increase the AVF. This is a surprising conclusion that further research is required in order to clarify the origin of these results. In [10] the impact of different compiler optimization levels on reliability is evaluated with the help of a metric called expected number of failures during the application's execution (EF). This metric is calculated using execution time of the program, AVF, and IntrinsicFIT rate. This study aims at evaluating the impact of compiler optimization on the microarchitectural level. Authors of the paper concluded that compiler optimization increases the number of instructions in-flight and significantly decreases the execution time, which leads to fewer expected failures during program's execution (EF).

Compiler optimizations and variation in program implementations are also studies in [31]. Authors of the paper introduced a metric called program vulnerability factor (PVF). They claim that PVF is independent of the underlying microarchitecture and therefore it is more generic than AVF. Using PVF results of different implementations or compiler optimizations, a software developer can choose the implementation/optimization that is more reliable. They also mentioned that PVF could be estimated using techniques such as fault injection, even though they used a dynamic instructions analysis called ACE analysis that would be beneficial to be compared with fault injection estimations.

1.2 Research Questions

As already explained, the error sensitivity of an executing program depends on several sources of variations. The overall objective of this thesis is to evaluate how significant are the effects of the stated sources of variations on error sensitivity. This thesis focuses on the following research questions:

RQ1. *How significant is the effect of input processed by a program on error sensitivity?*

It is clear that input has effects on the execution of a program, e.g. depending on the program's input some conditional statements become true and some function calls take place. Therefore, the inputs processed by a program determines the sequence of dynamic instructions and thereby error propagation process. However, we have no estimation on how significant is this effect on error sensitivity, and how can we benchmark a program with respect to all possible input sets. Our experimental study gives insights in how significant this effect is. We also investigate if there is a correlation between the input features and error sensitivity. To benchmark a program with several input sets, it would be more efficient to limit the number of fault injection campaigns by identifying input sets that are likely to cause significantly different error sensitivities.

RQ2. *Is there a correlation between the type of machine instruction targeted by fault injection experiment and the outcome of the experiment?*

Here we define six categories of target instructions (Load, Store, Arithmetic, Branch, Logical, and Move). We investigate if there is a correlation between the category of the targeted instruction and the fault injection outcome. For instance, is it more probable that fault injection experiments result in SDCs if we have arithmetic-intensive program or there is no such a correlation.

RQ3. *Does the single bit-flip model provide optimistic or pessimistic estimates of error sensitivity compared to the double bit-flip model?*

We define two fault models; single bit-flips and double bit-flips. In double bit-flip model, double bits are selected from same target location. This study is partly motivated by the fact that researchers in the field of reliability physics predict that single event upsets will be likely to generate MBUs in circuits that will become available within a few year from now [32]. While it is still an open question how these MBUs will manifest at the instruction set architecture level in detail, it is clear that we can

expect an increasing rate of hardware errors that will manifest as multiple bit errors in main memory words and CPU-registers. Although our study only addresses on double bits errors, it provides insights into the problem of defining multiple-bit fault models for dependability benchmarking experiments.

RQ4. *How error sensitivity varies for different bit positions, within a register or memory word?*

By answering this question we identify which bit positions within a register or memory word are more sensitive, i.e. having an error in those bits would result in silent data corruptions. This study helps in configuring fault injection experiments to find weaknesses more effectively. For instance, there is no need to inject in bits that 100% result in the hardware exceptions or they make no impact on the outcome of a program.

RQ5. *Does optimized program code have higher error sensitivity than non-optimized code?*

This question is fundamental for understanding the overall impact of compiler optimizations on system reliability. If the error sensitivity is significantly higher for optimized code than for non-optimized code, then designers of safety- and mission-critical systems must carefully analyze whether the use of optimized code is of advantage or disadvantage in meeting safety and reliability requirements. On the other hand, if the difference in error sensitivity is small between optimized and non-optimized code, then such an analysis is not necessary. (Still, of course, designers must consider other possible negative side effects of using optimized code, such as an increased risk of systematic faults.) However, a reduction in the number of executed instructions reduces the risk that an executing program is affected by transient hardware errors. Hence, it is clear that compiler optimization has a positive effect on system reliability in terms of a lower error occurrence probability (or error rate). However, it is not clear how compiler optimizations affect a program's error sensitivity.

RQ6. *To what extent do variations in the source code implementation of a program affect error sensitivity?*

To answer this question, we investigate how differences in source codes of functionally equivalent versions of a program affect the error sensitivity. These differences come from different programming styles such as using lookup tables, number of function calls, which types of data structures used, e. g., pointers, unions, structs, etc., and how to implement a calculation (e.g. make use of shift or multiplication operators).

1.3 Research Contributions

This thesis presents the results of extensive fault injection experiments conducted to address the research questions presented in previous section. The contributions are presented in four papers referred to as Paper I — Paper IV. The main goal of the thesis is to study the impact of different sources of variations on error sensitivity. In each contribution, we identify the impact of one or two particular sources of variations. We also get insights on how to evaluate programs in presence of hardware faults. Indeed, it is helpful to consider these sources of variations in design of cross-layer fault tolerance techniques. Therefore, these contributions will be beneficial for whom are interested in experimental benchmarking of error sensitivities.

- The first contribution (Paper I) discusses the impact of input processed by a program on its error sensitivity; how significant the impact is, and how error sensitivity is correlated to input features. The error sensitivity and its variation is application dependent. We could find a linear correlation between input length and SDCs for some applications while there were no correlations in some other applications. In this study we perform fault injection experiments on four programs from MiBench suite [1]. We selected nine different inputs for each program. This study shows significant variation in error sensitivity of a program executed with different inputs. For instance, in an extreme case SDCs of CRC application varies 30 percentage points from

one input with 0 characters to another input with 99 characters.

- Paper I, in addition, propose an approach to correlate the dynamic fault-free behavior of a program with fault injection outcomes particularly SDCs. We observed significant variations in the error sensitivities among different workloads[2]. Hence, a program should be evaluated by all possible inputs. We propose a way to identify inputs that result in significantly different fault injection outcomes. To this end, we use assembly metrics defined based on machine instructions of fault-free execution of a program. We cluster the workloads based on these assembly metrics and compare these clusters with the ones generated based on SDC outcomes. We discovered that workloads with similar SDC outcomes have also similar assembly metric clusters. Thus, the workloads that end up in a same cluster have also similar SDC outcomes. In this way we identify input sets that are likely to cause significantly different error sensitivities. Consequently, we can limit the number of needed fault injection campaigns to the number of clusters and perform only campaigns that generate significantly different error sensitivities.

- In Paper I, we also evaluate the error sensitivity of the programs equipped by a software technique to tolerate hardware faults. This technique performs triple-time redundant execution and majority voting. It is interesting that this simple technique (simple in design even though it is not time or energy efficient) can successfully decrease SDCs, on the average, from 25% to less than 3%.

- Paper II considers the types of machine instructions (Load, Store, Arithmetic, Branch, Logical, and Move) targeted in fault injection experiments. We investigate if certain types of machine instructions are more likely to cause SDCs. Although we could not identify a particular group of instructions that is more likely to result in SDCs for all benchmark programs, our results do provide some interesting observations regarding the error sensitivity of different instruction categories. In general, we conclude that the error sensitivity

[2] A workload is an executing program with a given input.

of a machine instruction seems to depend more on the type of data it processes rather than the instruction itself. This suggests that the effectiveness of fault tolerance techniques targeting specific instructions may vary rather significantly for different programs, and such techniques therefore need to be tailored to the programs they are intended to protect.

- The third study (Paper III) discusses the necessity of considering double bit-flip errors in dependability benchmarking experiments. This paper presents the results of an extensive experimental study of bit-flip errors in CPU-registers and main memory words. Comprising more than two million fault injection experiments conducted with thirteen benchmark programs, the study provides insights on whether the double bit-flip model provides optimistic or pessimistic estimates of error sensitivity compared to the single bit-flip model. The results show that the proportion of SDCs, is almost the same for single and double bit errors. Furthermore, single-bit flips resulted in slightly more SDCs for some campaigns which means it is a better model due to its ability to finds more weaknesses (SDCs) in a program.

- In addition, we study how error sensitivity varies for different bit positions within a register or memory word. We present detailed statistics about the variations in error sensitivity with respect to bit positions. These results show that the error sensitivity varies significantly for different bit positions. An important observation is that injections in certain bit positions always have the same outcome regardless of when the error is injected. For instance, all injections in more significant bit positions of program counter register (PCR) (e.g. bit position 17 to 32) are detected by hardware exceptions. It is notable that these results depend on the programs, underlying microprocessor and memory allocations to some extent.

- In Paper IV we investigate the impact of compiler optimizations on the error sensitivity of twelve benchmark programs. We conducted extensive fault injection experiments where bit-flip errors were injected in CPU-registers and main memory locations. The results show that the percentage of SDCs in the output of the optimized programs is only marginally higher compare to

that observed for the non-optimized programs. This suggests that compiler optimizations can be used in safety- and mission-critical systems without increasing the risk that the system produces undetected erroneous outputs. It is notable that program execution time reduces significantly by aid of compiler optimization. Therefore, the programs are also less exposed to faults.

- Paper IV also discusses the impact of different source code implementations of a functionally equivalent program on error sensitivity. To this end, we perform experiments on five bit count programs included in the MiBench suite [1]. These programs basically differ in data types used to store results, using a lookup table for some pre-calculated values, and different ways of implementing a calculation. The results of the fault injection experiments show that the source code implementation has a significant impact on error sensitivity. To provide insights into the reasons for the variation in error sensitivity, we conducted a detailed analysis of the error sensitivity of different types of data stored in registers and memory words (that were targeted for fault injection). This analysis is helpful in identifying registers and memory sections with high error sensitivity, which thus are candidates for being protected by fault tolerance techniques. In this investigation, we consider the impact of the implementation as well as compiler optimization. The results of these experiments give valuable insights into how compiler optimization can be beneficial to reduce number of registers and memory sections that are sensitive to errors (result in high percentage of SDCs).

1.4 Concluding Remarks and Future Work

In this thesis we investigated the effect of some sources of variation on error sensitivity. A general limitation of this work is that we have not studied all possible sources of variations. Each paper gives some insights about the one or two sources of variation studied in that paper, however, to investigate all sources of variations we would need unlimited time and facilities. For instance, there is no agreement on multiple bit-flip fault models and there would be several dif-

ferent ways to define each model and then perform thousands of fault injection experiments to evaluate the effect of that model on the error sensitivity.

Another main source of variation would be the underlying hardware including the processor design. Even though it is feasible to assess error sensitivity of different processors using simulation tools, it is not trivial to use real processors. For instance, we do not have access to pipeline stage structures and flip-flops through Nexus port. Therefore, we only inject faults in instruction set architecture registers and main memory words. As Goofi-2 implemented in a generic way, it is possible to add plug-ins for other processors. Indeed, using simulation-based fault injections to investigate the effect of processor design is also interesting. It would be more convenient to use different processor simulations and compare injecting faults in underlying CPU structures such as pipeline registers. A comparison between the results of simulation and real hardware will be helpful to evaluate accuracy of processor simulation and corresponding simulation-based fault injection tool.

The comparison of fault injection results can be performed on different levels of abstraction, e.g. on Simulink model, C code and binary code. This evaluation technique is called back-to-back testing which would be interesting to perform in future work.

We studied one type of several possible multiple bit-flip fault models, which is double bit-flips in same target location (paper II). The outcome of this study encouraged us to use single bit-flips due to the fact that increasing the number of bit flips increases the chance that the injected error would raise hardware exceptions. In this way, the use of double and multiple bit injections would lead to fewer observations of silent data corruptions. This suggests that it is unlikely that experiments with double-bit errors would expose weaknesses that are not revealed by single bit-flips. To further assess whether single bit flips can be trusted to generate the most pessimistic results (the highest number of SDCs), a possible future work include experiments where bit-flips are injected in different target locations.

We also investigated how source code implementation would affect error sensitivity (paper III). However, as we stated we used five different implemen-

tations of functionally same program and we chose a simple program to make a thorough analysis on the results. We could draw some interesting conclusions, though further studies with more complicated programs and variety of functionality would help to generalize the results. Especially considering different programming styles such as object-oriented vs structured programming and compare those with respect to dependability of the software would be interesting for safety-critical domain.

All these studies are expendable with implementing software techniques to tolerate hardware faults and investigate the error sensitivity of the program equipped with fault tolerance techniques. We have not done it at the first place to find out if we can affect error sensitivity without using any additional cost in terms of fault tolerance mechanisms. For instance, we see that compiler optimizations could reduce the execution time significantly while the error sensitivity is not increased or increased marginally for some applications. Also, we can reduce silent data corruptions by changing the programming style. However, it is obvious that we cannot reach to desirable error coverage without fault tolerance techniques in safety-critical applications. Thus, it is relevant to examine different fault tolerant techniques and come up with cost-efficient solutions.

Another line of future work is to consider a system which includes different software components and investigate how error sensitivity of the components would affect the error sensitivity of the whole system.

Bibliography

[1] Mibench version 1.0. `http://www.eecs.umich.edu/mibench/`. Accessed April 22, 2014.

[2] R. Alexandersson and J. Karlsson. Fault injection-based assessment of aspect-oriented implementation of fault tolerance. In *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 303–314, June 2011.

[3] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. Leber. Comparison of physical and software-implemented fault injection techniques. *Computers, IEEE Transactions on*, 52(9):1115–1133, Sept 2003.

[4] R. Barbosa. *Layered Fault Tolerance for Distributed Embedded Systems*. PhD thesis, Chalmers University of Technology, Chalmers University of technology, 12 2008.

[5] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson. Assembly-level pre-injection analysis for improving fault injection efficiency. In *Proceedings of the 5th European Conference on Dependable Computing*, EDCC'05, pages 246–262, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, Nov 2005.

[7] N. Carter, H. Naeimi, and D. Gardner. Design techniques for cross-layer resilience. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1023–1028, March 2010.

[8] M. Cukier, D. Powell, and J. Ariat. Coverage estimation methods for stratified fault-injection. *Computers, IEEE Transactions on*, 48(7):707–723, Jul 1999.

[9] A. DeHon, H. Quinn, and N. Carter. Vision for cross-layer optimization to address the dual challenges of energy and reliability. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1017–1022, March 2010.

[10] M. Demertzi, M. Annavaram, and M. Hall. Analyzing the effects of compiler optimizations on application reliability. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 184–193, Nov 2011.

[11] A. Fidalgo, G. Alves, and J. Ferreira. Real time fault injection using enhanced ocd – a performance analysis. In *Defect and Fault Tolerance in VLSI Systems, 2006. DFT '06. 21st IEEE International Symposium on*, pages 254–264, Oct 2006.

[12] P. Folkesson and J. Karlsson. Considering workload input variations in error coverage estimation. In *Proceedings of the Third European Dependable Computing Conference on Dependable Computing*, EDCC-3, pages 171–190, London, UK, UK, 1999. Springer-Verlag.

[13] J. C. for Guides in Metrology (JCGM). Evaluation of measurement data âĂŤ guide to the expression of uncertainty in measurement.

[14] C.-H. Ho, M. de Kruijf, K. Sankaralingam, B. Rountree, M. Schulz, and B. De Supinski. Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 510–519, Sept 2012.

[15] T. M. Jones and M. F. P. OâĂŹboyle. Evaluating the effects of compiler optimisations on avf, 2008.

[16] G. Kanawati, N. Kanawati, and J. Abraham. Ferrari: a tool for the validation of system dependability properties. In *Fault-Tolerant Computing, 1992. FTCS-22.*

Digest of Papers., Twenty-Second International Symposium on, pages 336–344, July 1992.

[17] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 502–506, April 2009.

[18] J. Li and Q. Tan. Smartinjector: Exploiting intelligent fault injection for sdc rate analysis. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*, pages 236–242, Oct 2013.

[19] H. Madeira, M. Rela, F. Moreira, and J. Silva. Rifle: A general purpose pin-level fault injector. In K. Echtle, D. Hammer, and D. Powell, editors, *Dependable Computing âĂŤ EDCC-1*, volume 852 of *Lecture Notes in Computer Science*, pages 197–216. Springer Berlin Heidelberg, 1994.

[20] A. Martinez-Alvarez, S. Cuenca-Asensi, F. Restrepo-Calle, F. Pinto, H. Guzman-Miranda, and M. Aguirre. Compiler-directed soft error mitigation for embedded systems. *Dependable and Secure Computing, IEEE Transactions on*, 9(2):159–172, March 2012.

[21] S. Mitra, K. Brelsford, and P. Sanda. Cross-layer resilience challenges: Metrics and optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1029–1034, March 2010.

[22] B. Nicolescu, Y. Savaria, and R. Velazco. Software detection mechanisms providing full coverage against single bit-flip faults. *Nuclear Science, IEEE Transactions on*, 51(6):3510–3518, Dec 2004.

[23] D. Powell, E. Martins, J. Arlat, and Y. Crouzet. Estimators for fault tolerance coverage evaluation. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 228–237, June 1993.

[24] M. Rebaudengo, M. Reorda, and M. Violante. A new approach to software-implemented fault tolerance. *Journal of Electronic Testing*, 20(4):433–437, 2004.

[25] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August. Swift: software implemented fault tolerance. In *Code Generation and Optimization, 2005. CGO 2005. International Symposium on*, pages 243–254, March 2005.

[26] S. Satoh, Y. Tosaka, and S. Wender. Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on dram's. *Electron Device Letters, IEEE*, 21(6):310–312, June 2000.

[27] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin. Fiat-fault injection based automated testing environment. In *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, pages 102–107, June 1988.

[28] D. Skarin. *On Fault Injection-Based Assessment of Safety-Critical Systems*. PhD thesis, Chalmers University of Technology, Chalmers University of technology, 12 2010.

[29] D. Skarin, R. Barbosa, and J. Karlsson. Goofi-2: A tool for experimental dependability assessment. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 557–562, June 2010.

[30] D. Skarin and J. Karlsson. Software implemented detection and recovery of soft errors in a brake-by-wire system. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 145–154, May 2008.

[31] V. Sridharan and D. Kaeli. Eliminating microarchitectural dependency from architectural vulnerability. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 117–128, Feb 2009.

[32] J. Suh. *Models for soft errors in low-level caches*. PhD thesis, University of Southern California, University of Southern California, 1 2012.

[33] E. Touloupis, J. Flint, V. Chouliaras, and D. Ward. Study of the effects of seu-induced faults on a pipeline protected microprocessor. *Computers, IEEE Transactions on*, 56(12):1585–1596, Dec 2007.

[34] P. Yuste, J. Ruiz, L. Lemus, and P. Gil. Non-intrusive software-implemented fault injection in embedded systems. In Lemos, editor, *Dependable Computing*, volume 2847 of *Lecture Notes in Computer Science*, pages 23–38. Springer Berlin Heidelberg, 2003.