THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

On Efficient Measurement of the Impact of Hardware Errors in Computer Systems

BEHROOZ SANGCHOOLIE

Division of Computer Engineering
Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017

On Efficient Measurement of the Impact of Hardware Errors in Computer Systems
Behrooz Sangchoolie
Göteborg, Sweden, 2017
ISBN: 978-91-7597-557-3

Copyright © Behrooz Sangchoolie, 2017.

Doktorsavhandlingar vid
Chalmers tekniska högskola
Ny serie Nr 4238
ISSN 0346-718X

Technical Report No. 141D
Dependable Real-Time Systems Group
Department of Computer Science and Engineering
Chalmers University of Technology

Contact Information:

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
`http://www.chalmers.se/cse/`

Printed by Chalmers Reproservice
Göteborg, Sweden 2017

To my parents

On Efficient Measurement of the Impact of Hardware Errors in Computer Systems

Behrooz Sangchoolie
Department of Computer Science and Engineering
Chalmers University of Technology, Sweden

Abstract

Technology and voltage scaling is making integrated circuits increasingly susceptible to failures caused by *soft errors*. The source of soft errors are temporary hardware faults that alter data and signals in digital circuits. Soft errors are predominately caused by ionizing particles, electrical noise and wear-out effects, but may also occur as a result of marginal circuit designs and manufacturing process variations.

Modern computers are equipped with a range of hardware and software based mechanisms for detecting and correcting soft errors, as well as other types of hardware errors. While these mechanisms can handle a variety of errors and error types, protecting a computer completely from the effects of soft errors is technically and economically infeasible. Hence, in applications where reliability and data integrity is of primary concern, it is desirable to assess and measure the system's ability to detect and correct soft errors.

This thesis is devoted to the problem of measuring *hardware error sensitivity* of computer systems. We define hardware error sensitivity as the probability that a hardware error results in an undetected erroneous output. Since the complexity of computer systems makes it extremely demanding to assess the effectiveness of error handling mechanisms analytically, error sensitivity and related measures, e.g., error coverage, are in practice determined experimentally by means of *fault injection experiments*.

The error sensitivity of a computer system depends not only on the design of its error handling mechanism, but also on the program executed by the computer. In addition, measurements of error sensitivity is affected by the experimental set-up, including how and where the errors are injected, and the assumptions about how soft errors are manifested, i.e., the error model. This thesis identifies and investigates six parameters, or sources of variation, that affect measurements of error sensitivity. These parameters consist of two subgroups, those that deal with systems characteristics, namely, (i) the input processed by a program, (ii) the program's source code implementation, (iii) the level of compiler optimization; and those that deal with measurement setup, namely, (iv) the number of bits that are targeted in each experiment, (v) the target location in which faults are injected, (vi) the time of injection.

To accurately measure the error sensitivity of a system, one needs to conduct several sets of fault injection experiments by varying different sources of variations. As these experiments are quite time-consuming, it is desirable to improve the *efficiency* of fault injection-based measurement of error sensitivity. To this end, the thesis proposes and evaluates different error space optimization and error space pruning techniques to reduce the time and effort needed to measure the error sensitivity.

Keywords: soft errors, error sensitivity, fault Injection, efficiency, bit-flip errors

List of Publications

This thesis is based on the following appended peer reviewed conference papers. References to these papers will be made using the associated Roman numerals.

- ▷ **Paper I:** Behrooz Sangchoolie, Fatemeh Ayatolahi, Roger Johansson and Johan Karlsson, "A Comparison of Inject-on-Read and Inject-on-Write in ISA-Level Fault Injection," *11th European Dependable Computing Conference (EDCC), Paris, France, 2015.*

- ▷ **Paper II:** Behrooz Sangchoolie, Fatemeh Ayatolahi, Roger Johansson and Johan Karlsson, "A Study of the Impact of Bit-flip Errors on Programs Compiled with Different Optimization Levels," *10th European Dependable Computing Conference (EDCC), Newcastle upon Tyne, UK, 2014.*

- ▷ **Paper III:** Fatemeh Ayatolahi, Behrooz Sangchoolie, Roger Johansson and Johan Karlsson, "A Study of the Impact of Single Bit-Flip and Double Bit-Flip Errors on Program Execution," *32nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Toulouse, France, 2013.*

- ▷ **Paper IV:** Behrooz Sangchoolie, Karthik Pattabiraman and Johan Karlsson, "One Bit is (Not) Enough: An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors," *47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, USA, 2017.*

- ▷ **Paper V:** Domenico Di Leo, Fatemeh Ayatolahi, Behrooz Sangchoolie, Johan Karlsson and Roger Johansson, "On the Impact of Hardware Faults - An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions," *31st International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Magdeburg, Germany, 2012.*

- ▷ **Paper VI:** Behrooz Sangchoolie, Roger Johansson and Johan Karlsson, "Light-Weight Techniques for Improving the Controllability and Efficiency of ISA-Level Fault Injection Tools," *22nd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), Christchurch, New Zealand, 2017.*

The following papers are related but not covered in this thesis.

- ▷ Peter Folkesson, Fatemeh Ayatolahi, Behrooz Sangchoolie, Jonny Vinter, Mafijul Islam and Johan Karlsson, "Back-to-Back Fault Injection Testing in Model-Based Development," *34th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Delft, Netherlands, 2015.*

- ▷ Mafijul Md. Islam, Behrooz Sangchoolie, Fatemeh Ayatolahi, Daniel Skarin, Jonny Vinter, Fredrik Törner, Andreas Käck, Mattias Nyberg, Emilia Villani, Johan Haraldsson, Patrik Isaksson, Johan Karlsson, "Towards Benchmarking of Functional Safety in the Automotive Industry," *14th European Workshop on Dependable Computing (EWDC), Coimbra, Portugal, 2013.*

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

- ▷ Behrooz Sangchoolie, Fatemeh Ayatolahi and Johan Karlsson, "An Investigation of the Fault Sensitivity of Four Benchmark Workloads," *1st Workshop on Software-Based Methods for Robust Embedded Systems (SOBRES), Braunschweig, Germany, 2012.*

Acknowledgments

First of all, I would like to thank my advisor Professor Johan Karlsson for his excellent comments, feedback, support and most importantly, his confidence in me. I have enjoyed all discussions and corridor talks that we have had during the past few years. Johan, I am very grateful you gave me the privilege of being your student and I would like to truly thank you for all the encouragements and support you gave me in pursuing different research ideas.

I am also very thankful of my co-advisors Professor Jan Jonsson and Assistant Professor Risat Pathan for their continuous support during my Master and Ph.D. studies. Jan, you have always believed in me and I am so grateful of the trust you had in me in your courses on Real-Time Systems and Parallel and Distributed Real-Time Systems. Risat, thanks for all the time you spent discussing my research ideas and raising excellent questions. I have learned a lot from you both on research methods and pedagogy.

Special thanks and gratitude to my thesis examiner Professor Per Stenström and the rest of my Ph.D. follow-up committee members for their constructive feedback during my studies. I would also like to thank Associate Professor Roger Johansson for all his support on the experimental setup of my experiments; to Dr. Daniel Skarin for the development of Goofi-2, the tool I have used to conduct my fault injection experiments; to Associate Professor Karthik Pattabiraman for giving me the opportunity to join his research group at the University of British Columbia and for the excellent collaboration we have had; to Dr. Rolf Snedsböl for all his positivity and support in organizing my teaching duties; to Marianne Pleén Schreiber for all our "Swedish" conversations and our great working environment; to Eva Axelsson, Tiina Rankanen, Lotta Kegel Andersson, Monica Månhammar, Anneli Andersson and Birgitta Gustafsson for their excellent administrative support; to Peter Helander for his technical support; to Professor Gerardo Schneider for introducing me to the world of tango; to Dr. Cyril Cohen for bringing improvisation theatre to my life; to all other past and current colleagues of mine at the Department of Computer Science and Engineering at Chalmers for creating such a friendly environment.

Many thanks to Fatemeh Ayatolahi, an amazing friend, excellent researcher, and fun officemate for the great time that we spent together in the past few years. Fatemeh, if had not looked for a Master's thesis partner around six years ago, I might not have been where I am today!

I would like to thank the Swedish Agency for Innovation Systems (VINNOVA) for funding the first couple of years of my research under the BeSafe project. Special thanks goes to all the partners in this project including Dr. Mafijul Islam, Dr. Daniel Skarin, Dr. Jonny Vinter, Dr. Andreas Käck, Dr. Fredrik Törner, Dr. Mattias Nyberg, Mats Olsson, and Patrik Isaksson. I would also like to acknowledge the support I got in the past couple of years from the EU funded HiPEAC (High Performance and Embedded Architectures and Compilers) Network of Excellence as well as the Ericsson Research Foundation and Chalmers research fund.

My years of studying at Chalmers were filled with plenty of good memories (and some challenging moments) that I shared with amazing friends. I cannot wish for having better friends as they have always been there for me. It is almost impossible to name them all here, but they know who they are. Especial thanks to my friends with whom I spent most of my leisure time hanging out, traveling and climbing with, especially the guys at Impram, my improvisation theatre group, who helped me explore this amazing theatre form. I would also like to thank all my friends from back in Iran who always prioritize spending time with me and making sure I have a great time whenever I am there.

Last and most importantly, I would like to thank my family and Julia. Dear Mom and dear Dad, you have given me so much hope and energy throughout all these years and you have always believed in me and supported me unconditionally. Thank you for everything. Dear Bahman, Behzad, and Shadi, you guys are the best brothers and sisters one could wish for. Thanks for always backing me up and believing in me. Dear Julia <thanks for believing in us; bearing with the long distance in between us for more than two years; moving to Sweden; being my best friend and number one fan and supporter>.

Behrooz Sangchoolie
May 5, 2017

Contents

1
Thesis Summary

1 Introduction

Measurements play a vital role in science and engineering. Without measurements, we cannot improve products and processes. A measure is a number (or a vector of numbers) representing a property of an object or activity. It serves as a point of reference and enables comparison. The most common forms of measures are those that describe physical properties such as speed, weight, length, and temperature.

Another type of measures is the ones that describe system properties. Measuring properties of systems is not always as easy as measuring physical properties of objects. This is due to the complexity of systems. Examples of such measures are quality attributes of computer systems such as performance, energy consumption, dependability, and safety. The process of finding proper measures and obtaining them to assess and reason about these attributes requires a great deal of knowledge about different components of a system and its infrastructure.

Measuring and evaluating dependability properties of computing systems have become increasingly important in applications where computers are used to control safety-critical and mission-critical processes. Examples of such applications can be found in the automotive, avionic, and nuclear power industries.

This thesis deals with techniques for measuring the impact of an increasingly important class of computer errors called *soft errors*. Soft errors are caused by temporary hardware faults generated by ionizing particles, electrical noise, wear-out effects, etc. Such errors have become an increasingly important source of computer failures due to technology and voltage scaling [Bor05, ITR]. Modern computers are equipped with a range of error handling mechanisms to protect them from soft errors, as well as other types of errors, using different error detection and error correction techniques.

The focus of the thesis is on techniques for measuring the *error sensitivity* of systems, or programs, with respect to soft errors. We define error sensitivity as the probability that an error in a computer system results in an undetected erroneous output, also known as a silent data corruption.

Obtaining the error sensitivity of a computer system by purely analytical methods (such as Failure Mode Effects Analysis) is difficult, and often infeasible, due to the high complexity of computer systems. Researchers and engineers, therefore, measure the error sensitivity experimentally by means of *fault injection experiments*. Fault injection is an established method used for measurement, test and assessment of dependable computer systems. The inclusion of fault injection as a highly recommended assessment method in the ISO 26262 standard [ISO09] for functional safety of road vehicles demonstrates the increasing importance of experimentally validating error handling techniques in critical computer systems.

The basic approach of fault injection is to artificially insert faults into a system to enable an analysis of the system's behaviour in the presence of faults. The work presented in this thesis considers injection of bit-flip errors in instruction set architecture (ISA) registers and static random-access memory (SRAM) words using hardware-based fault injection, for emulating the effects of soft errors. In addition, it also considers software-based injection of bit-flip errors into a LLVM (Low Level Virtual Machine) compiler's intermediate code.

The remainder of this introduction includes a problem statement, which provides an overview of the technical challenges addressed by the thesis, and a summary of the thesis contributions.

1.1 Problem Statement

The use of fault injection experiments for measuring error sensitivity involves many technical and scientific challenges. This thesis focuses on two important aspects of such measurements:

- Sources of variation
- Measurement efficiency

In general, the error sensitivity of a system depends on the design and implementation of the system (including its error handling mechanisms), as well as the inputs processed by the system. In addition, when measuring the error sensitivity of a system (or a program) the results of the measurements may depend heavily on the experimental set-up, and especially on the fault model. Thus, when measuring error sensitivity, system evaluators need to consider multiple sources of variations when they interpret and assess the validity of the experimental results.

The thesis presents studies of six important sources of variation in fault injection experiments. These studies focus on measuring the error sensitivity of programs, i.e., they consider the case when the purpose of the measurements is to compare the error sensitivity of different programs, or program versions, under the assumption that the hardware platform is fixed.

Three of the sources of variation are related to system characteristics and program implementation: (i) the input processed by a program, (ii) the program's source code implementation and (iii) the level of compiler optimization. The other three are related to measurement setup, namely, (i) the number of bits that are targeted in each experiment, (ii) the location in which faults are injected and (iii) the time of injection.

It should be noted that the sources of variation addressed in this thesis represent either systematic variations in the true error sensitivity of the targets system related to its design, or systematic variations caused by the experimental set-up. They are conceptually different from "sources of uncertainties", such as those addressed by Skarin et al. [SBK10a], which are associated with measurement procedures, tools, non-repeatability of experiments, non-representative sampling, time instants chosen for collecting measurements, and so forth.

Concerning measurement efficiency, the thesis addresses the problem of input selection and error space pruning. In setting up a fault injection experiment, the evaluator must select the set of inputs that the target system will process during the experiments. Since the error sensitivity depends on the inputs processed by the system, error sensitivity measurements are usually conducted with multiple input vectors representing different use cases of the target system. (In this context, an input vector corresponds to the set of inputs processed by the target system for a specific use case, processing cycle, or program run.)

Thus, an important goal of error sensitivity measurements is to investigate how much the error sensitivity varies for different input vectors (use cases). Since measuring the error sensitivity for a single use case (one input vector) could be quite time consuming – it may require several thousands and even millions of fault injection experiments – it is desirable to develop methods that can distinguish between input vectors that are likely to produce different (or similar) outcome distributions before any fault injection experiments have been conducted.

To this end, the thesis presents a method for input selection that relies on profiling of the dynamic machine instructions executed for a given input vector. This method can increase measurement efficiency, since it helps an evaluator to avoid conducting experiments with several input vectors that are likely to yield similar results.

Another important challenge addressed in this thesis is the problem of error space reduction, also known as error space pruning. The purpose of error space pruning is to reduce the size of the error space from which the injected bit-flips are sampled, so that the injection of "uninteresting" bit-flips is avoided. Examples of such bit-flips include those that cannot be activated by program execution, or those that are overwritten by the program without affecting the behaviour of the target system. Another example is bit-flips whose impact can be determined a priori without the need for an experiment. (A concrete example of the latter is bit-flips in the most significant bit of the program counter, which in most systems always raises a hardware exception.)

Prior work have proposed pre-injection analysis techniques for error space reduction, e.g. [TP13, SHK$^+$12, BVFK05], while other authors have proposed heuristic pruning methods [HANR12, LT13] to improve the efficiency of fault injection-based measurement of dependability metrics.

This thesis proposes and evaluates two pre-injection analysis techniques for improving the efficiency of fault injection experiments. One of them identifies the type of data-items (address, data or control bits) held in a potential target location (an ISA register or memory word). This technique allows the evaluator to focus the injections to target locations that hold a specific type of data-item. The other pre-injection analysis technique avoids injection of bit-flips in certain bits which are known to always trigger a hardware exception (e.g., certain bits of program counter and stack pointer register). In addition, the thesis also presents an error pruning technique that reduces the error space when performing experiments with multiple-bit injections.

1.2 Main Contributions

The contributions of this thesis are presented in six papers referred to as Paper I – Paper VI. The summary of each paper and its contributions can be found in Section 3. This thesis, in addition to extending two fault injection tools, namely GOOFI-2 [SBK10b] and LLFI [TP13], makes the following main contributions. The contributions are divided into two categories; those that are related to sources of variation (*C1 – C6*) and those that are related to the measurement efficiency (*C7 – C9*).

C1. Evaluates the extent in which variation of a program's input affect its outcome distributions and identifies the input length as a program property that could be linearly correlated to the program's error sensitivity. *(Paper V)*

C2. Assesses the extent in which a program's error sensitivity is affected by its source code implementation and identifies different program characteristics that cause variations in the error sensitivity. *(Paper I, II, III, VI)*

C3. Evaluates the variation in the outcome distributions of programs compiled with different levels of compiler optimization and concludes that compiler optimization levels have only a minor impact on the error sensitivity. *(Paper II)*

C4. Compares the error sensitivity of programs with respect to the number of errors caused due to one or multiple hardware faults and concludes that single-bit errors, in most cases, yield a higher error sensitivity compared to multiple-bit errors. *(Paper III, IV)*

C5. Evaluates the outcome distributions of different target registers and memory words, including the significance of target bits and identifies bits that if targeted by errors, would never cause an erroneous output. *(Paper I, II, III)*

C6. Compares the error sensitivity of programs with respect to the time of injection using two fault injection techniques called inject-on-read and inject-on-write and concludes that it is unlikely that the latter technique would expose weaknesses that are not revealed by the former. *(Paper I, IV)*

C7. Designs and evaluates a technique called input selection that improves the efficiency of error sensitivity measurement, by identifying inputs that are likely to result in different outcome distributions. *(Paper V)*

C8. Designs and evaluates different error space pruning techniques that are used to improve the efficiency of error sensitivity measurement when using multiple bit-flip fault injection campaigns. *(Paper IV)*

C9. Designs and evaluates two pre-injection analysis techniques that improve the controllability and efficiency of fault injection campaigns by identifying the data type of data-items stored in target registers and memory locations. *(Paper VI)*

1.3 Thesis Structure

The remaining of this thesis is organized as follows. Section 2 briefly describes the background to this thesis. Section 3 summarizes the papers presented in Chapters 2 – 7 and discusses their contributions. Section 4 describes some of the conclusions and the future directions of this thesis. And finally, the six papers discussed in this thesis are included to Chapters 2 – 7.

2 Background

Measurement of error sensitivity is useful in several domains. Frontiers of these domains are the ones dealing with dependable and safety-critical computer systems, such as avionics and automotive industries, where failures in their systems could jeopardize systems' dependability and potentially result in loss of life. Error sensitivity of computer systems should be measured both with respect to software level and hardware level errors. As discussed in Section 1, hardware level error sensitivity measurement is specifically of utmost importance, since the rate of hardware errors are expected to increase in systems. This is because technology scaling is making transistors increasingly susceptible to soft errors caused by process variations, wear-out effects, and ionizing particles [Bor05].

After assessing a system's software and hardware components, one should also measure the error sensitivity of the whole system as one. This is due to the existing interactions amongst different hardware and software units. For example, errors in a microprocessor may affect execution of a program that runs on it. In fact, it is likely that future microprocessors will exhibit an increasing rate of incorrect program executions caused by hardware related errors.

The rest of this section is organized as follows. First, some threats to dependability are presented. Then, fault tolerance mechanisms are presented that can be used to mitigate the negative impact of these threats. This is followed by a presentation of different failure modes. Then, some dependability measures are compared. This is followed by a presentation of some fault injection tools, including the ones used in this thesis to measure the error sensitivity.

2.1 Dependability Threats

According to Avižieniz et al. [ALRL04], the threats to dependability are *faults*, *errors*, and *failures*. A failure is termination of a system's ability to perform a function as required. The cause of this termination is called an error. In other words, an error is raised as a result of a discrepancy between a measured value and a theoretically correct value [ISO09]. The cause of this discrepancy is known as a fault. For example, a fault is when a hardware defect makes a certain location in the main memory unusable. An error is raised as a result of an access to the faulty memory location. And a failure happens when the error causes the system to produce an incorrect output.

Faults can also be classified according to their nature of persistence, namely, *permanent*, *transient*, and *intermittent*. Permanent faults remain in or at the boundary of a system once they occur, requiring the faulty component to be repaired or replaced. Transient faults, on the other hand, occur at a specific time, remaining only for a short period of time in the system. Intermittent faults occur repeatedly at the same location, i.e., repeatedly appearing transient faults are called intermittent faults. Note that any of these three types of faults can jeopardize the system's safety by increasing the error sensitivity of its components. That is why fault tolerance mechanisms are needed to tolerate and mitigate the impact of these faults to the system.

2.2 Fault Tolerance Mechanisms

Fault tolerance mechanisms are implemented in different layers of abstraction, such as hardware, software, and system. Hardware is the first line of defence where different error detection and error correction mechanisms detect and signal many errors as well as correcting a fraction of the detected ones. These mechanisms include hardware exceptions (e.g., illegal opcodes and invalid memory access), parity checking, error correcting codes (ECCs) and triple modular redundancy (TMR), etc. If a detected error cannot be tolerated, it is signalled to allow higher-layer mechanisms to take appropriate action.

Software-implemented fault tolerance mechanisms are implemented in software and are normally used in conjunction with hardware-level mechanisms. Software-implemented mechanisms are cost-effective and flexible as they do not require extra hardware components. Popular software-implemented fault tolerance techniques include the followings. Time-redundant execution [AVFK02] corresponding to when a program is being executed multiple times and the results of all executions are being compared to; software assertions [And79] that corresponds to placement of some checks on programs variables, such as boundary checking of an integer variable; aspect-oriented fault tolerance [AK11, Gal02] which implements the fault tolerance mechanism separately as *aspects* instead of placing the fault tolerance code as part of the main function that needs to be protected; software watchdog timers that detect errors that have caused a program to hang or to be stuck in an infinite loop; control flow checking [MHGT92], which are several checks that are placed in the program verifying that the code is executing in a allowed order; other examples include recovery blocks [HLMSR74] and N-version programming [CA78].

System-layer fault tolerance mechanisms are the last line of defence. In a distributed computer system, this layer of fault tolerance should detect and tolerate failures in the computer systems and data-buses. This is normally achieved by physical replication of the nodes and buses in conjunction with other types of fault tolerance protocols.

2.3 Failure Modes

A hardware fault can affect a program (or system) in one of the following ways:

- *Crash.* An exception is raised and the program is terminated.
- *Hang.* The program runs for a significantly longer time than normal. Watchdog timers could be used to detect these errors.
- *Silent data corruption (SDC), a.k.a. undetectable value failure.* The program terminates normally, but the output is incorrect (based on a bit-wise comparison), and there is no indication of the failure.
- *Detectable value failure.* The program terminates normally, but the output is incorrect; however, the program can detect the failure.
- *Timing failure.* The program delivers the output too late, or too early. Watchdog timers could be used to detect these errors.
- *Silent failure.* The program, unexpectedly, delivers no output. The lack of output delivery could be combined with watchdog timers to detect these errors.
- *Signaled failure.* The program sends a failure signal, indicating an error detection.
- *Benign.* The program terminates normally and the fault does not affect the program's output. This category could be the result of internal robustness of the program.

Except the benign category, all other categories correspond to different types of failure modes. Failure modes describe the nature of a failure, i.e., the way in which a program (or system) can fail. Among the different failure modes, SDCs are considered the most severe, because users will trust the program's output in the absence of an error indication; whereas for all other categories, upon the detection of an error, a recovery routine could be called to recover the system. Moreover, there is no generic method to detect SDCs without re-executing the entire program and checking for a mismatch, or without a significant amount of hardware redundancy, both of which are expensive.

2.4 Dependability Measures

This section presents different measures that could be used to evaluate the dependability of computer systems. These measures include the *architectural vulnerability factor (AVF), program vulnerability factor (PVF), error coverage, error resiliency* and *error sensitivity*.

The AVF is designed with the idea that not all faults in a microarchitectural structure affect the final outcome of a program. For example, a single-bit error in a branch predictor will not affect the sequence or results of any committed instructions. Therefore, AVF is defined as the probability that a fault in a processor structure will result in a visible error in the final output of a program. The AVFs of different processor structures could be estimated using an approach that tracks the subset of processor state bits required for architecturally correct execution (ACE) [MWE$^+$03]. AVF, however, is intricately tied to the microarchitectural design of a processor, and cannot be used to reason about software resilience in isolation.

Sridharan et al. [SK09] separate the hardware-specific component of AVF from the software-specific component and propose the PVF, which is a systematic method to efficiently evaluate the error resilience of software under hardware faults. PVF can also be used for predictive and comparative analysis studies to understand the effect of different protection techniques or code transformations on the error resilience. However, PVF does not distinguish between failure modes (see Section 2.3) and, essentially, treats all of them as equally severe. Therefore, using PVF to estimate application error resilience and inform the protection mechanisms often leads to overprotecting applications, thereby resulting in unnecessary performance and energy overheads.

Error coverage – usually denoted by c – is defined as the conditional probability that the program recovers, given the occurrence of a fault [BCS69, Arn73]. Similar to PVF, error coverage does not distinguish between different failure modes. However, as mentioned in Section 2.3, in practice, SDCs are the more important class of failures, as the erroneous outputs are generated with no indication of failure, making them very difficult to detect. Therefore, instead of the error coverage, some researchers have used error resiliency [FLP$^+$16, LFW$^+$15] as the dependability metric.

Error resiliency is defined as the conditional probability that the program does not produce an SDC after a transient hardware fault occurs and impacts the program state. In other words, similar to work such as [dKNS10, FGAM10, KM14], it deals with faults passing the hardware and seen by the software. Error sensitivity, on the other hand, is a complementary metric to error resiliency and is defined as the probability that an SDC occurs in the program's output, given that a transient hardware error has occurred in a hardware unit. In other words, the error sensitivity is equal to 1 minus the error resiliency metric.

2.5 Fault Injection

Fault injection techniques have been extensively used to evaluate the effectiveness of error handling mechanisms as well as to improve the accuracy of measures such as the ones presented in Section 2.4. Several fault injection techniques have been proposed in the past decades for injecting hardware faults into systems. These techniques can be categorized into *simulation-based*, *software-implemented*, *hardware-based*, and *radiation-based* techniques. Table 1.1 shows a summary of some of the fault injection tools developed in the past 30 years along with the fault injection techniques they use.

Simulation-based techniques inject faults into hardware models (e.g. VHDL models) as opposed to an actual physical system or prototype. Software-implemented (SWiFI) techniques, on the other hand, emulate the effects of physical faults in software. SWiFI

Table 1.1: Some Fault Injection Techniques and Tools Used in the Past 30 Years to Inject Hardware Faults into Computer Systems.

Fault injection tools	**Fault injection techniques**			
	simulation-based	**software-implemented**	**hardware-based**	**radiation-based**
FIAT [SVS^{+}88]		✓		
MESSALINE [ACL89]			✓	
Karlsson et al. [KGLT91]			✓	✓
FERRARI [KKA92]		✓		
Czeck et al. [CS92]	✓			
Goswami and Iyer [GI93]	✓			
FINE [KIT93]		✓		
RIFLE [MS94, MRMS94]			✓	
Karlsson et al. [KLD^{+}94]				✓
MEFISTO [JAR^{+}94]	✓			
FTAPE [TI95]		✓		
DOCTOR [HSR95]		✓		
Xception [CMS98]		✓		
FIMBUL [FSK98]			✓	
AFIT [MGM^{+}99]			✓	
GOOFI [AVFK01]		✓	✓	
MAFALDA [AFR02]		✓		
INTERTE [YRLG03]			✓	
Xception [CMCS03]		✓	✓	
Fidalgo et al. [FAF06a]			✓	
GOOFI-2 [SBK10b]		✓	✓	
MODIFI [SVET10]	✓	✓	✓	
Relyzer [HANR12]	✓			
LLFI [TP13]		✓		
GemFI [PTAB14]	✓			
FAIL* [SHK^{+}12, SHD^{+}15]	✓		✓	

can emulate faults in various parts of the hardware (such as CPU registers, the arithmetic logic units and the main memory) and could be divided into runtime and pre-runtime techniques. Hardware-based techniques, however, are applied on actual implementations or prototypes of a system, instead of the system's model. While the observability and controllability can be limited, the efficiency is often high. Techniques for hardware-based fault injection can be divided into pin-level fault injection, test port-based fault injection, and power supply disturbances. And finally, in radiation-based techniques, faults can be injected into processors by exposing them to external disturbances such as Electromagnetic Interference (EMI) and particle radiation.

The use of fault injection has recently been increased in the embedded systems and automotive industries. For example, Table 1.2 shows that fault injection is either recommended or highly recommended in more than 10 assessment activities proposed by ISO 26262 [ISO09] standard for functional safety of road vehicles.

In this thesis, the error sensitivity is measured using hardware-based fault injection and software-implemented fault injection (SWiFI). To this end, two distinct fault injection tools, namely GOOFI-2 [SBK10b] and LLFI [TP13] are used (and extended). GOOFI-2 facilitates hardware-based fault injection using test port-based fault injection;

Table 1.2: Assessment Activities Recommended/Highly Recommended to be Performed by Fault Injection According to ISO 26262 Standard.

Development Level	Assessment Context
System	System design verification
	Correct implementation of technical safety requirements at the hardware-software level
	Effectiveness of a safety mechanism's diagnostic coverage at the hardware-software level
	Correct implementation of functional safety and technical safety requirements
	Effectiveness of a safety mechanism's failure coverage at the system level
	Correct implementation of the functional safety requirements at the vehicle level
	Effectiveness of a safety mechanism's failure coverage at the vehicle level
Hardware	Hardware design verification
	Hardware integration tests
Software	Methods for software unit testing
	Methods for software integration testing

whereas LLFI facilitates SWiFI. These tools use the *bit-flip* errors to mimic transient hardware faults caused by soft errors that occur in the processor's register file, ALUs, and in different pipeline registers that eventually manifest as a data corruption in a source or destination register or memory word.

2.5.1 GOOFI-2 (Generic Object-Oriented Fault Injector)

GOOFI [AVFK01] was first presented in 2001 and then enhanced with support for more target systems and new injection techniques in its next version GOOFI-2 [SBK10b] in 2010. GOOFI-2 can be configured to conduct test port-based fault injection as well as two software-implemented fault injection (SWiFI) techniques. The first SWiFI technique places the fault injection code in exception-handling routines intended for debugging, while the second one injects faults by instrumenting the executable file with fault injection code before it is downloaded to the target system.

This thesis uses GOOFI-2 to conduct test port-based fault injection, which includes techniques that use TAPs (test access ports) to inject faults. Examples of TAPs used for fault injection include JTAG (standard test access port and boundary-scan architecture) [JTA01], BDM (background debug mode) [How96], and Nexus [Nex]. GOOFI-2's test port-based technique uses the Nexus [Nex] port to inject errors into ISA (instruction set architecture) registers and memory segments of MPC565 and MPC5554, PowerPC-based microcontrollers from Freescale. Nexus is a standard on-chip debug interface, which provides read/write access to processor's resources. Using the test port-based technique, GOOFI-2 conducts fault injection without altering the software of the target system.

GOOFI-2 defines a fault injection experiment to be the injection of one fault and the monitoring of its impact on the program. A fault injection campaign, on the other hand, is a set of fault injection experiments using the same fault model on a given workload. And a workload is a program running with a given input. During each experiment, GOOFI-2 controls the program under test using a development environment called winIDEA [win] in conjunction with the iC3000 debugger [iC3]. GOOFI-2 stores the acquired data of each experiment into a database, which can later on be used to classify the outcome of the experiments.

2.5.2 LLFI (LLVM-Based Fault Injector)

LLFI [TP13] is an open source fault injection tool that facilitates SWiFI. SWiFI is often faster than hardware-based injection, and requires no extra hardware support. LLFI injects faults into the LLVM [LA04] framework's intermediate code of a program. LLVM is a collection of reusable compiler tools and components, and allows analysis and optimization of code written in multiple programming languages. The key component of LLVM is its intermediate representation (IR), an assembly-like language that abstracts out the hardware and ISA-specific information.

3 Summary of Papers and Contributions

This section presents a summary of the papers included in Chapters 2 – 7. For each paper, the problem statement and related work are presented as well as the paper's contributions and their implications. A statement of contributions of each co-author is also presented for each paper explaining the division of work between the authors.

3.1 Paper I: A Comparison of Inject-on-Read and Inject-on-Write in ISA-Level Fault Injection

Problem statement and related work 80-90% of randomly injected faults are often not even activated [MS94, YRLG03]. Therefore, one could improve the efficiency of fault injection campaigns by eliminating faults with no possibility of activation. Examples of these are faults placed in a *target location* (i.e., instruction set architecture (ISA) registers and memory words) just before the location is written into (and is overwritten), and faults that are injected into unused locations. In other words, the efficiency of fault injection campaigns could be improved by merely targeting live target locations. *Inject-on-read* is one of the techniques used for targeting live locations, where a fault is only injected into a source target location before it is read by an instruction [TP13, SHK$^+$12, BVFK05]. Using this technique, Barbosa et al. [BVFK05] managed to reduce the error space of workloads by two to five orders of magnitude.

In the inject-on-read technique, all faults targeting a specific bit of a given target location, from the time the location is written into until it is read, are considered equivalent. However, this technique needs to take into account the exposure time of each location and bit to a potential hardware fault, due to the fact that the longer a location is live and waiting to be read, the more likely it is for a fault to occur in that location. Here, we refer to the exposure time of a location as the *lifetime* of the location. Therefore, to obtain an accurate estimate of measures such as the error sensitivity, it is necessary to apply a weight factor corresponding to the lifetime of each target location [BVFK05, SBS15]. This way, we could give a higher weight to the failure modes of locations that have a higher lifetime.

In this paper, we enhance the inject-on-read technique by measuring the lifetime of different target locations. The lifetime of a target location to a fault should ideally be the number of CPU cycles between the time that the location is accessed (read or written into) and when the location is read. However, we instead estimate the lifetime by counting the number of instructions executed between the time that a location is accessed and the when it is read.

This paper also addresses the *fault model*, as an essential parameter used in fault injection techniques. Fault model specifies the type of faults that are emulated by a fault injection technique. Here, we compare two fault models by means of two distinct fault injection techniques, namely inject-on-read and inject-on-write. The former injects an error into the source register (or memory word) before it is read by an instruction, whereas the latter injects an error into the destination register after it is written into. The inject-on-read technique [TP13, RCMM07] emulates soft errors that occurred in the processor's register file or a memory cell. However, soft errors occurred in other hardware units such as the ALU is more likely to corrupt the destination register of an ISA instruction, instead of the source register. Therefore, the inject-on-write is used to model these types of faults.

Contributions and their implications This paper measures the percentage of SDCs (see Section 2.3) for when the inject-on-read and inject-on-write ISA-level fault injection techniques are used. These techniques are used to analyse the impact of *(i)* the weight factor, and *(ii)* the time of injection to the SDC results. To this end, a new pre-injection analysis technique is implemented on GOOFI-2 [SBK10b] to facilitate the injection of errors into destination locations of assembly-level instructions, allowing us to conduct inject-on-write fault injection campaigns. For each program under test, results are reported for the entire program as well as the different locations targeted, namely general purpose registers, program counter registers, miscellaneous registers and memory words.

Results of our experiments show that the weight factor could significantly affect the fault injection results by causing variations in the percentage of SDCs. Moreover, we identify three factors that contribute to these variations. The first one is the type of data stored in different target locations. We use *data-item* as a generic term for denoting the content of an ISA register or memory word. A data-item can contain *a datum*, *an address* or *a condition flag*. The second factor corresponds to the distribution of different types of data-items in the programs under test. And the third factor corresponds to the average lifetime of different data-items in the programs under test. In general, when using the weight factor, the percentage of SDCs is biased more towards the percentage of SDCs of data-items that has high distributions as well as high lifetime.

When it comes to the time of injection (before reading a location or after writing into a location), one could already expect that in case a data-item is only read once (after being updated by an instruction), there is no difference between the SDC results of injecting errors into source and destination locations, given that the same target location and the same bit, or bits, are corrupted in the two cases. In contrast, for data-items that are read multiple times, injecting errors into the source registers (locations) results in a larger fault space compared to when injecting errors into the destination registers. Therefore, we observed that the SDC results are more biased towards the SDC results of data-items that are read multiple times.

Furthermore, this study shows that the impact of the weight factor and the time of injection to the SDC results are highly dependent on a subset of the data-items targeted. This implies that these data-items and their data types could be identified before conducting any fault injection experiments allowing us to improve the efficiency of fault injection campaigns by focusing on a subset of data-items that are of interest.

Statement of contributions The paper was coauthored with Fatemeh Ayatolahi, Roger Johansson, and Johan Karlsson. Behrooz Sangchoolie came up with the original idea behind the paper, designed and conducted the experiments and was the main contributor in writing the paper. Fatemeh Ayatolahi provided feedback during different phases of the study and contributed to designing the fault injector for brake-by-wire application (which is one of the applications targeted by faults in this study). Johan Karlsson provided feedback during different phases of the study. Roger Johansson helped in solving technical issues around the fault injection setup. All authors contributed to the writing of the paper.

3.2 Paper II: A Study of the Impact of Bit-flip Errors on Programs Compiled with Different Optimization Levels

Problem statement and related work One of the main motivations for optimizing a program is to minimize its execution time. Compilers such as GCC facilitate this by reducing the number of dynamic instructions through enabling different optimization flags. A reduction in the number of dynamic instructions reduces the risk that a program is affected by transient hardware errors. This is due to the fact that the program under optimization would have a lower exposure to this type of errors. In other words, compiler optimizations have a positive effect on system reliability in terms of a lower error occurrence probability (exposure time). However, the impact of compiler optimizations on system reliability and failure modes is elusive. Therefore, this study mainly investigates the impact of compiler optimizations on failure mode distributions. The optimized codes are produced with the optimization levels -O1, - O2, -O3, and -Os defined by GCC compiler.

This paper also investigates the extent in which variations in the source code implementation of a program (compiled with and without an optimization level) affects its hardware error sensitivity.

Alexandersson et al. [AK11] study the impact of optimizing a program on the error sensitivity only for the -O3 optimization level. Demertzi et al. [DAH11] analyse the effect of standard optimization levels on the application's vulnerability. The authors use ACE analysis [MWE$^+$03] (see Section 2.4) to evaluate the applications' resilience, which is typically much less accurate than fault injection [WMP07]. Sridharan et al. [SK09] also use the ACE analysis, however along with PVF (see Section 2.4), to study variations cause due to different program implementations and two compiler optimizations flags. Jones et al. [JOE08] study the impact of individual flags (as opposed to optimization levels) on application vulnerability and attempt to find a set of flags that offer both resilience and performance. Similar to [DAH11], here the authors use AVF to measure vulnerability and do not consider the final outcome of the application caused by the error. Narayanamurthy et al. [NPR16] also study the impact of individual flags and use genetic algorithms to find the application-specific set of compiler optimizations that can boost performance without degrading the application's error resilience.

Contributions and their implications The results show that compiler optimization levels had only a minor impact on the error sensitivity of the programs under test. This suggests that compiler optimizations can be used in safety- and mission-critical systems without a significant increase in the risk that the system produces undetected erroneous outputs.

The results imply that one could improve the efficiency of measuring hardware error sensitivity, by first optimizing the program under test, since this reduces the number of dynamic instructions without having a significant impact on the error sensitivity measured.

Furthermore, the results of our study show that the source code implementation of programs has a significant impact on their hardware error sensitivity. The cause of this variation is due to the varying characteristics of the programs, e.g., in terms of the number of dynamic instructions, number of memory accesses, size of the input variable, type of data structures used in program, etc.

In this paper, we also conduct a detailed analysis of the hardware error sensitivity of different types of data that were targeted for fault injection. This analysis allowed us to identify data-item types with high hardware error sensitivity, which could be candidates for being protected by software-based fault tolerance techniques.

Statement of contributions The paper was coauthored with Fatemeh Ayatolahi, Roger Johansson, and Johan Karlsson. Behrooz Sangchoolie and Fatemeh Ayatolahi came up with the original idea behind the paper, designed and conducted the fault injection experiments and were the main contributors in writing the paper. Johan Karlsson provided feedback during different phases of the study. Roger Johansson helped in solving technical issues around the fault injection setup. All authors contributed to the writing of the paper.

3.3 Paper III: A Study of the Impact of Single Bit-Flip and Double Bit-Flip Errors on Program Execution

Problem statement and related work The fault model used in SWiFI techniques needs to be both straightforward to implement, and representative of real hardware faults. The single bit-flip model has been a popular engineering approximation to mimic particle induced soft errors both in the combinational logic and storage elements (e.g., flip-flops). However, earlier studies have found that many soft errors manifest as multiple-bit errors at the application level [CMC$^+$13, KKA93, ZCM$^+$96], which has led researchers to question the validity of the single bit-flip used for modelling the effect of soft errors.

In this paper, we compare the impact of single and double bit-flip errors injected into the same target locations. Comparing the results obtained for these two fault models, provides insights to an important open question, namely, whether the single bit-flip model provides optimistic or pessimistic estimates of error sensitivity. Lu et al. [LFW$^+$15] and Adamu-Fika et al. [AFJ15] also compare the results of injecting single bit-flip errors with injecting double bit-flip errors. They, however, conducted the fault injection experiments at the LLVM (Low Level Virtual Machine) [LA04] compiler's intermediate code level as opposed to the assembly level, which is used in our paper. Touloupis et al. [TMCW07], use VHDL (VHSIC Hardware Description Language) simulation model to investigate the impact of single and double bit-flip errors.

Contributions and their implications The results show that the proportion of SDCs (see Section 2.3) in the program output is almost the same for single and double bit-flip errors. This suggests that it is unlikely that experiments with double-bit errors would expose weaknesses that are not revealed by single bit-flip injection. This suggests that one could improve the efficiency of measuring hardware error sensitivity by only conducting single bit-flip injections.

This study also presents detailed statistics about the hardware error sensitivity of different target registers and memory locations, including bit positions within registers and memory words. The results show that the hardware error sensitivity varies significantly between different bit positions and registers. However, we also observe that injections in certain bit positions always have the same impact regardless of when the error is injected. This allows us to reduce the error space size of future experiments by avoiding injection of faults into these bit positions as we know *a priori* the impact of targeting them with errors.

Statement of contributions The paper was coauthored with Fatemeh Ayatolahi, Roger Johansson, and Johan Karlsson. Behrooz Sangchoolie and Fatemeh Ayatolahi came up with the original idea behind the paper, designed and conducted the experiments and were the main contributors in writing the paper. Johan Karlsson provided feedback during different phases of the study. Roger Johansson helped in solving technical issues around the fault injection setup. All authors contributed to the writing of the paper.

3.4 Paper IV: One Bit is (Not) Enough: An Empirical Study of the Impact of Single and Multiple Bit-Flip Errors

Problem statement and related work The necessity of conducting multiple bit-flip fault injection campaigns was already discussed in Section 3.3 (Paper III). However, in that paper, we limited the analysis to only double bit-flip errors injected in the same word. In this paper, we go beyond the double bit-flip injections by injecting up to 30 bit-flip errors in single words as well as different words in each program run.

In order to conduct multiple bit-flip injections, there are two main challenges that need to be addressed, namely (i) how to select a representative number of bits to flip as well as (ii) how to deal with the large error space size. These challenges are caused due to the fact that there is no commonly agreed model to map transient faults, caused due to soft errors, to their software-level manifestation. Furthermore, conventional techniques for reducing (a.k.a pruning) the error space may not be applicable as almost all the existing techniques for pruning the error space [BVFK05, HANR12, VMHA16] work with the single-bit fault model, and are not easily extensible to multiple-bit errors. Therefore, it is essential to find and evaluate error space pruning techniques that are well-suited for multiple bit-flip campaigns.

Jiantao Pan [Pan99] introduces a model called dimensionality to pin-point the number of function call parameters that are responsible for a failure. The model is used in a subsequent work [PKS99] to improve software robustness. However, multiple errors are only introduced to the parameters of each interface, which may not be representative of multiple errors that occur in variables used within the function. Moreover, the number of errors that are introduced to each interface is limited by the number of parameters used by the interface. There are also studies addressing intermittent faults, which could potentially model some multiple-bit errors. For example, Rashid et al. [RPG15] build an intermittent fault model at the microarchitectural level using intermittent stuck-at-last-value and stuck-at-zero/one models. However, they assume that *(i)* a microarchitectural unit may be affected by at most a single intermittent fault and *(ii)* at most a single microarchitectural unit may be affected by an intermittent fault. These assumptions may not hold for transient faults due to soft errors, which is our focus.

Contributions and their implications To overcome the challenge of selecting a representative number of bits to flip, we propose a systematic error space exploration that is based on error space clustering, where each cluster is represented by two parameters, *(i)* the number of bit-flip errors that could occur in the cluster; and *(ii)* the distance (in terms of the number of dynamic instructions) between consecutive injections. These parameters allow us to quantify the maximum (upper bound) number of multiple bit-flip errors needed to cause pessimistic percentage of SDCs (see Section 2.3).

We also found that the single bit-flip model mostly results in pessimistic percentage of SDCs compared to the multiple bit-flip model; and even when it does not, in most cases, at most three errors are enough to result in a pessimistic percentage of SDCs.

To overcome the challenge of dealing with the large error space size, we propose three ways of pruning the error space based on the fault injection results obtained. Using these error pruning techniques, we derive new insights about how the results of single bit-flip experiments can be used to prune the multiple bit-flip error space by targeting only a fraction of these errors, that reveal weaknesses of the programs under test (in terms of the number of SDCs) that are not revealed by the single bit-flip model. In fact, we can leverage the results from the single bit-flip fault injections to choose the locations for multiple bit-flip injections to get conservative SDC results.

Statement of contributions The paper was coauthored with Karthik Pattabiraman and Johan Karlsson. Behrooz Sangchoolie and Karthik Pattabiraman came up with the original idea behind the paper and were the main actors in different phases of the study and writing of the paper. Behrooz Sangchoolie designed and conducted the fault injection experiments. Johan Karlsson provided feedback during different phases of the study and contributed to the writing of the paper.

3.5 Paper V: On the Impact of Hardware Faults –An Investigation of the Relationship between Workload Inputs and Failure Mode Distributions

Problem statement and related work The error resilience for software-implemented error handling techniques often depends on the input vector processed by the target system. Thus, to assess the variability in error resilience, it is essential to conduct fault injection experiments with different input vectors. In [SVS$^+$88], matrix multiplication and selection sort are fed with three and two inputs, respectively. Folkesson and Karlsson in [FK99] estimated the error coverage for quicksort and shellsort, both executed with 24 different inputs.

When assessing the variability in error resilience of a program, it is unfeasible to cover all possible inputs, especially since conducting fault injection experiments is time-consuming. Therefore, this paper presents a technique called *input selection*, where inputs are selected such that they are likely to result in widely different failure mode distributions. The input selection technique uses hierarchical clustering [JMF99] analysis to divide the input sets into homogenous groups based on assembly-level signature of execution flows.

We adopted the hierarchical clustering [JMF99] due to the fact that unlike other clustering techniques (e.g., K-means [JMF99]), it does not require a preliminary knowledge of the number of clusters. Thus, we can validate a posteriori if the execution flows are clustered as expected. The hierarchical clustering adopted in this work evaluates the distance between two clusters according to the centroid method [JMF99]. A similar approach is used by Natella et al. [NCDM13].

Contributions and their implications Results show a clear variation among the percentage of SDCs (see Section 2.3) of programs using different inputs. In fact, there is a linear correlation between the percentage of SDCs and the length of input in two of the target programs. On the other hand, results illustrate that the percentage of errors detected by the hardware exceptions is program dependent, i.e., it is not affected by the input. The results obtained are from single bit-flip fault injection experiments into four programs where faults are injected in CPU registers and main memory of the target system. For each program, fault injection experiments are conducted for nine different inputs. The inputs are chosen to represent input lengths that are common in real applications.

The study shows that similar inputs (e.g., same length inputs) result in a similar failure distribution; thus, the input selection technique helps to reduce the number of fault injections. This is done in three steps. First, the fault-free executions of a program for a large set of inputs are profiled using assembly code metrics. Then a cluster analysis is used to form clusters of similar execution flows. Finally, one representative execution flow from each cluster is selected for conducting fault injection analysis.

This paper also addresses a software-implemented hardware fault tolerant (SIHFT) technique [RCV$^+$05, RRV04] that relies on triple-time redundant execution, majority voting and forward recovery (TTR-FR) [AK11]. Thus, in addition to the basic version of the programs, faults are also injected on programs equipped with the TTR-FR. Results of this analysis show that a simple software-implemented hardware fault tolerant mechanism, TTR-FR, can successfully increase the error coverage, on the average, to more than 97%, regardless of the input.

Statement of contributions The paper was coauthored with Fatemeh Ayatolahi, Domenico Di Leo, Roger Johansson, and Johan Karlsson. Domenico Di Leo came up with the original idea behind the paper. Domenico Di Leo, Behrooz Sangchoolie and Fatemeh Ayatolahi provided feedback during different phases of the study, designed and conducted the fault injection experiments and were the main contributors to the writing of the paper. Johan Karlsson provided feedback during different phases of the study. Roger Johansson helped in solving technical issues around the fault injection setup. All authors contributed to the writing of the paper.

3.6 Paper VI: Light-Weight Techniques for Improving the Controllability and Efficiency of ISA-Level Fault Injection Tools

Problem statement and related work Fault injection techniques could be characterized based on different properties such as *repeatability, observability, reachability, intrusiveness, controllability* and *efficiency*. In this paper we present two pre-injection analysis techniques, namely *data type identification* and *fault space optimization*, that improve the controllability and efficiency of fault injection techniques, where the former refers to the ability to control when and where a fault is injected while the latter refers to the time and effort needed to conduct a fault injection campaign. Our techniques rely on object code analysis, i.e., no source code is required.

The controllability property is addressed by works such as [SBK10b] where faults are identified by time-location pairs according to a fault-free execution of a program.

In this paper, we improve the controllability by identifying the type of *data-items* that are potential fault injection candidates, prior to conducting fault injection experiments. Here data-item refers to the content of a register or memory word and the type of a data-item could be a *data variable*, *memory address*, or *control information.* The motivation for the data type identification comes from prior work (Paper I, Paper II, and Paper III) where we learned that the outcome of a fault injection experiment is highly dependent on the type of data-items targeted.

The efficiency property is also addressed by work such as [TP13, SHK$^+$12, BVFK05] where pre-injection analysis is used to identify live target locations resulting in the reduction of the error space size. However, the error space size of programs under test could still be significantly large which is why most fault injection tools only sample a subset of the error space or employ different heuristic pruning methods [HANR12, LT13] to cover the complete error space.

The fault space optimization technique improves the efficiency of fault injection campaigns by fault pruning, i.e., by avoiding injection of faults that are known *a priori* to be detected by the system under test. The fault space optimization technique identifies certain bits of specific registers and memory segments that would always raise a hardware exception and exclude them from the fault space. This technique leverages the fact that faults in certain bits in the program counter and the stack pointer registers are always detected by machine exceptions (see Paper III).

Contributions and their implications Using the data type identification technique, we managed to successfully identify the type of data-items in 84-100% of target locations. Knowing the type of different data-items, provides us with a better control over the selection of locations where faults should be injected into.

This implies that we can design cost-efficient fault injection campaigns that only target sensitive data-items, as well as providing us with useful information about locations that need to be hardened by fault tolerance mechanisms. Moreover, one could also design cost-efficient fault tolerant mechanisms that are data-type-specific, suitable for tolerating faults in data-items with specific data types.

According to the result of our fault injection campaigns, we managed to, on average, prune 25% of the fault space using the fault space optimization technique. This reduction in the error space size significantly reduces the time it takes to conduct a fault injection campaign.

Statement of contributions The paper was coauthored with Roger Johansson and Johan Karlsson. Behrooz Sangchoolie came up with the original idea behind the paper, designed and conducted the fault injection experiments and was the main contributor in writing the paper. Johan Karlsson provided feedback during different phases of the study. Roger Johansson helped in solving technical issues around the fault injection setup. All authors contributed to the writing of the paper.

4 Concluding Remarks and Future Work

This thesis presents several studies aimed at investigating sources of variations in fault injection-based measurements of the error sensitivity of computer systems. In addition, it proposes several error pruning and pre-injection analysis techniques for improving the accuracy and efficiency of fault injection experiments. The thesis focuses entirely on the problem of measuring error sensitivity with respect to soft errors, i.e., errors caused by temporary hardware faults.

An important contribution of the thesis is a detailed study of how the inputs processed by a program affect its soft error sensitivity. Our study confirms results from previous research, which shows that the likelihood for a program to exhibit an SDC (a silent data corruption, see Section 2.3) due to a soft error strongly depends on the input to the program. Thus, when assessing the error sensitivity of a program by fault injection, it is desirable to perform experiments with several inputs. However, as fault injection campaigns are time-consuming in nature, a technique for input selection is proposed for selecting inputs such that they are likely to result in widely different outcome distributions. To this end, a set of 47 assembly metrics corresponding to different types of instructions and access types (read, write) to registers and memory words are adopted. The input selection technique seems promising for programs with a linear relation between an input property (i.e., length) and the failure distribution, however, additional assembly metrics are required for programs with no linear relation between the input length and the failure distribution. Therefore, looking forward, it would be worthwhile to investigate additional software profiling metrics to improve the confidence of the input selection technique.

Hardware error sensitivity of programs also shows great variations to different source code implementations and programming styles. The thesis shows that small differences in the C code implementations (e.g. making use of pointer structures instead of union structures), can significantly affect the hardware error sensitivity. In fact, one could improve the error resiliency of programs by simply applying different data structures or programming styles without using additional fault tolerance mechanisms. However, as our observations are based on one rather small program, there is a need to further investigate the impact of source code implementation on error sensitivity for other programs, especially for programs with higher code size.

Concerning variations in error sensitivity resulting from the use of different levels of compiler optimization, we show that the GCC compiler optimization levels (-O1, -O2, -O3, -Os) have only a minor impact on the hardware error sensitivity of the investigated programs. This suggests that compiler optimization is unlikely to increase error sensitivity and therefore can be used in safety- and mission-critical systems without a significant increase in the risk that the system produces undetected erroneous outputs. However, as our study is limited to 12 rather small programs, further experiments with larger programs are needed to confirm our observation that complier optimization have a minor impact on error sensitivity of a program.

Regarding differences in the impact of single and multiple bit-flips, our experiments show that single bit-flip errors resulted in a higher proportion of SDCs compared to multiple bit-flip errors, for most of the conducted fault injection campaigns. Thus, we observed that the single bit-flip model, on average, provides more pessimistic estimates of error sensitivity than our multiple bit-flip model. However, we also saw a few campaigns where the injection of multiple-bit errors resulted in a higher proportion of SDC compared to the corresponding campaigns conducted with single-bit errors, which suggest that investigating the impact of multiple bit errors may still be relevant for some systems. For such studies, we propose error space pruning techniques that significantly reduce the size of the error space in experiments where multiple bit-flips are injected.

When investigating variations in the error sensitivity of different ISA-registers and memory words, we noted that errors in certain bit positions always have the same impact regardless of when the error is injected. For example, errors injected in some bits of the stack pointer and the program counter registers are always detected by hardware exceptions. Based on this observation, we designed a pre-injection analysis technique that excludes injection of errors in such bit positions.

Finally, the thesis also presents a comparison of the inject-on-read and inject-on-write techniques. It shows that inject-on-read causes a higher percentage of SDCs than inject-on-write. However, in cases where a data-item is read only one time (after being updated by an instruction), there is no difference between the SDC results of the inject-on-read and inject-on-write techniques. The difference between the percentages of SDCs observed for the two techniques comes from registers an memory words that are read multiple times. In fact, the error space of the inject-on-read technique is much larger than the inject-on-write technique due to the fact that a register could be read multiple times after being updated by an instruction. Depending on the error sensitivity of the registers that are read multiple times, either of the inject-on-read and inject-on-write techniques could result in pessimistic SDC results. Therefore, there is a need to conduct more detailed studies to gain a better understanding of the differences in impact of the two techniques.

Bibliography

[AAA$^+$90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *IEEE Transactions on Software Engineering*, 16(2):166–182, 1990.

[ACK$^+$03] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber. Comparison of physical and software-implemented fault injection techniques. *IEEE Transactions on Computers*, 52(9):1115–1133, 2003.

[ACL89] J. Arlat, Y. Crouzet, and J. C. Laprie. Fault injection for dependability validation of fault-tolerant computing systems. In *[1989] Digest of Papers. The 19th International Symposium on Fault-Tolerant Computing*, pages 348–355. IEEE Computer Society, 1989.

[AFJ15] F. Adamu-Fika and A. Jhumka. An investigation of the impact of double bit-flip error variants on program execution. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 799–813. Springer International Publishing, 2015.

[AFR02] J. Arlat, J. C. Fabre, and M. Rodriguez. Dependability of COTS microkernel-based systems. *IEEE Transactions on Computers*, 51(2):138–163, 2002.

[AK11] R. Alexandersson and J. Karlsson. Fault injection-based assessment of aspect-oriented implementation of fault tolerance. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 303–314. IEEE Computer Society, 2011.

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[And79] D. M. Andrews. Using executable assertions for testing and fault tolerance. In *9th Fault-Tolerance Computing Symposium*, 1979.

[Arn73] T. F. Arnold. The concept of coverage and its effect on the reliability model of a repairable system. *IEEE Transactions on Computers*, 22(3):251–254, 1973.

[ASJK13] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson. A study of the impact of single bit-flip and double bit-flip errors on program execution. In *Proceedings of the 32nd International Conference on Computer Safety, Reliability, and Security*, SAFECOMP 2013, pages 265–276. Springer-Verlag New York, Inc., 2013.

[AVFK01] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. GOOFI: generic object-oriented fault injection tool. In *2001 International Conference on Dependable Systems and Networks*, pages 83–88. IEEE Computer Society, 2001.

[AVFK02] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. Experimental evaluation of time-redundant execution for a brake-by-wire application. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 210–215. IEEE Computer Society, 2002.

[BCS69] W. G. Bouricius, W. C. Carter, and P. R. Schneider. Reliability modeling techniques for self-repairing computer systems. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 295–309. ACM, 1969.

[Bor05] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[BRIM98] A. Benso, M. Rebaudengo, L. Impagliazzo, and P. Marmo. Fault-list collapsing for fault-injection experiments. In *Proceedings of the 1998 Annual Reliability and Maintainability Symposium*, pages 383–388. IEEE Computer Society, 1998.

[BVFK05] 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 Dependable Computing Conference*, EDCC 5, pages 246–262. Springer Berlin Heidelberg, 2005.

[CA78] L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Digest of Papers. 8th Annual International Conference on Fault Tolerant Computing*, FTCS-8, pages 3–9, 1978.

[CMC$^+$13] H. Cho, S. Mirkhani, C. Y. Cher, J. A. Abraham, and S. Mitra. Quantitative evaluation of soft error injection techniques for robust system design. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–10. IEEE Computer Society, 2013.

[CMCS03] D. Costa, H. Madeira, J. Carreira, and J. G. Silva. Xception^{TM}: A software implemented fault injection tool. In *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*, pages 125–139. Springer, 2003.

[CMS98] J. Carreira, H. Madeira, and J. G. Silva. Xception: a technique for the experimental evaluation of dependability in modern computers. *IEEE Transactions on Software Engineering*, 24(2):125–136, 1998.

[CS92] E. W. Czeck and D. P. Siewiorek. Observations on the effects of fault manifestation as a function of workload. *IEEE Transactions on Computers*, 41(5):559–566, 1992.

[DAH11] M. Demertzi, M. Annavaram, and M. Hall. Analyzing the effects of compiler optimizations on application reliability. In *2011 IEEE International Symposium on Workload Characterization (IISWC)*, pages 184–193. IEEE Computer Society, 2011.

[dKNS10] M. d. Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 497–508. ACM, 2010.

[ESC05] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, pages 2–12. IEEE Computer Society, 2005.

[FAF06a] A. V. Fidalgo, G. R. Alves, and J. M. Ferreira. Real time fault injection using a modified debugging infrastructure. In *12th IEEE International On-Line Testing Symposium*, IOLTS'06. IEEE Computer Society, 2006.

[FAF06b] A. V. Fidalgo, G. R. Alves, and J. M. Ferreira. Real time fault injection using enhanced OCD – a performance analysis. In *2006 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 254–264. IEEE Computer Society, 2006.

[FGAM10] S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: Probabilistic soft error reliability on the cheap. *SIGPLAN Notices*, 45(3):385–396, 2010.

[FK99] P. Folkesson and J. Karlsson. Considering workload input variations in error coverage estimation. In *Proceedings of the 3rd European Dependable Computing Conference on Dependable Computing*, EDCC-3, pages 171–190. Springer-Verlag, 1999.

[FLP$^+$16] B. Fang, Q. Lu, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi. ePVF: An enhanced program vulnerability factor methodology for cross-layer resilience analysis. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 168–179. IEEE Computer Society, 2016.

[FSK98] P. Folkesson, S. Svensson, and J. Karlsson. A comparison of simulation based and scan chain implemented fault injection. In *Digest of Papers. 28th Annual International Symposium on Fault-Tolerant Computing*, FTCS '98, pages 284–293. IEEE Computer Society, 1998.

[Gal02] A. Gal. On aspect-orientation in distributed real-time dependable systems. In *Proceedings of the 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. (WORDS 2002)*, pages 261–267. IEEE Computer Society, 2002.

[GI93] K. K. Goswami and R. K. Iyer. Simulation of software behavior under hardware faults. In *The 23rd International Symposium on Fault-Tolerant Computing*, FTCS-23, pages 218–227. IEEE Computer Society, 1993.

[GKI04] W. Gu, Z. Kalbarczyk, and R. K. Iyer. Error sensitivity of the linux kernel executing on PowerPC G4 and Pentium 4 processors. In *International Conference on Dependable Systems and Networks*, pages 887–896. IEEE Computer Society, 2004.

[GKT89] U. Gunneflo, J. Karlsson, and J. Torin. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In *[1989] Digest of Papers. The 19th International Symposium on Fault-Tolerant Computing*, pages 340–347. IEEE Computer Society, 1989.

[GRE$^+$01] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization*, WWC '01, pages 3–14. IEEE Computer Society, 2001.

[GS95] J. Guthoff and V. Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *Digest of Papers. The 25th International Symposium on Fault-Tolerant Computing*, pages 196–206. IEEE Computer Society, 1995.

[HANR12] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran. Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 123–134. ACM, 2012.

[HLMSR74] J. J. Horning, H. C. Lauer, P. M. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. In *Proceedings of an International Symposium on Operating Systems*, pages 171–187. Springer-Verlag, 1974.

[How96] S. Howard. A background debugging mode driver package for modular micro-controllers. *Semiconductor Application Note AN1230/D, Motorola Inc*, 1996.

[HSR95] S. Han, K. G. Shin, and H. A. Rosenberg. DOCTOR: an integrated software fault injection environment for distributed real-time systems. In *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium*, pages 204–213. IEEE Computer Society, 1995.

[HTI97] M. C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault injection techniques and tools. *Computer*, 30(4):75–82, 1997.

[iC3] iC3000 debugger. `http://www.isystem.com/products/11-products/89-ic3000-activeemulator`. Accessed: 2017-04-22.

[ISO09] ISO/DIS 26262-1 - road vehicles - functional safety - part 1 glossary. Technical report, Geneva, Switzerland, 2009.

[ITR] 2015 international technology roadmap for semiconductors (ITRS). `https://www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/`. Accessed: 2017-05-2.

[JAR$^+$94] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault injection into VHDL models: the MEFISTO tool. In *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, FTCS-24, pages 66–75. IEEE Computer Society, 1994.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[JOE08] T. M. Jones, M. F.P. O'boyle, and O. Ergin. Evaluating the effects of compiler optimisations on AVF. In *Workshop on Interaction Between Compilers and Computer Architecture (INTERACT-12)*, 2008.

[JTA01] IEEE standard test access port and boundary scan architecture. *IEEE Std 1149.1-2001*, pages 1–212, 2001.

[KGLT91] J. Karlsson, U. Gunneflo, P. Liden, and J. Torin. Two fault injection techniques for test of fault handling mechanisms. In *Proceedings of the 1991 International Test Conference*, pages 140–149. IEEE Computer Society, 1991.

[KIT93] W. I. Kao, R. K. Iyer, and D. Tang. FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults. *IEEE Transactions on Software Engineering*, 19(11):1105–1118, 1993.

[KKA92] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham. FERRARI: a tool for the validation of system dependability properties. In *[1992] Digest of Papers. The 22nd International Symposium on Fault-Tolerant Computing*, MICRO-47, pages 336–344. IEEE Computer Society, 1992.

[KKA93] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham. EMAX: An automatic extractor of high-level error models. In *Proceedings of the 9th AIAA Computing in Aerospace Conference*, pages 1297–1306, 1993.

[KKS98] N. P. Kropp, P. J. Koopman, and D. P. Siewiorek. Automated robustness testing of off-the-shelf software components. In *Proceedings of the 28th Annual International Symposium on Fault-Tolerant Computing*, FTCS '98, pages 230–239. IEEE Computer Society, 1998.

[KLD$^+$94] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo. Using heavy-ion radiation to validate fault-handling mechanisms. *IEEE Micro*, 14(1):8–23, 1994.

[KM14] D. S. Khudia and S. Mahlke. Harnessing soft computations for low-budget fault tolerance. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, pages 319–330. IEEE Computer Society, 2014.

[LA04] C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, pages 75–86. IEEE Computer Society, 2004.

[LAS$^+$12] D. Di Leo, F. Ayatolahi, B. Sangchoolie, J. Karlsson, and R. Johansson. On the impact of hardware faults — an investigation of the relationship between workload inputs and failure mode distributions. In *Proceedings of the 31st International Conference on Computer Safety, Reliability, and Security*, SAFECOMP '12, pages 198–209. Springer-Verlag, 2012.

[LFW$^+$15] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman. LLFI: An intermediate code-level fault injection tool for hardware faults. In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 11–16. IEEE Computer Society, 2015.

[LT13] J. Li and Q. Tan. Smartinjector: Exploiting intelligent fault injection for SDC rate analysis. In *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 236–242. IEEE Computer Society, 2013.

[MACARC$^+$12] A. Martinez-Alvarez, S. Cuenca-Asensi, F. Restrepo-Calle, F. R. Palomo Pinto, H. Guzman-Miranda, and M. A. Aguirre. Compiler-directed soft error mitigation for embedded systems. *IEEE Transactions on Dependable and Secure Computing*, 9(2):159–172, 2012.

[MGM$^+$99] R. J. Martinez, P. J. Gil, G. Martin, C. Perez, and J. J. Serrano. Experimental validation of high-speed fault-tolerant systems using physical fault injection. In *Dependable Computing for Critical Applications 7*, pages 249–265. IEEE Computer Society, 1999.

[MHGT92] G. Miremadi, J. Harlsson, U. Gunneflo, and J. Torin. Two software techniques for on-line error detection. In *[1992] Digest of Papers. The 22nd International Symposium on Fault-Tolerant Computing*, FTCS-22, pages 328–335. IEEE Computer Society, 1992.

[MiB] MiBench. `http://vhosts.eecs.umich.edu/mibench/`. Accessed: 2017-04-22.

[MRMS94] H. Madeira, M. Z. Rela, F. Moreira, and J. G. Silva. RIFLE: A general purpose pin-level fault injector. In *Proceedings of the 1st European Dependable Computing Conference on Dependable Computing*, EDCC-1, pages 199–216. Springer-Verlag, 1994.

[MS94] H. Madeira and J. G. Silva. Experimental evaluation of the fail-silent behavior in computers without error masking. In *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, pages 350–359. IEEE Computer Society, 1994.

[MW79] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Transactions on Electron Devices*, 26(1):2–9, 1979.

[MWE$^+$03] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 29–40. IEEE Computer Society, 2003.

[NCDM13] R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, 39(1):80–96, 2013.

[Nex] The Nexus 5001^{TM} Forum standard for a global embedded processor debug interface (IEEE-ISTO 5001^{TM}). `http://nexus5001.org`. Accessed: 2017-04-22.

[NPR16] N. Narayanamurthy, K. Pattabiraman, and M. Ripeanu. Finding resilience-friendly compiler optimizations using meta-heuristic search techniques. In *2016 12th European Dependable Computing Conference (EDCC)*, pages 1–12. IEEE Computer Society, 2016.

[NSG11] G. Nazarian, C. Strydis, and G. Gaydadjiev. Compatibility study of compile-time optimizations for power and reliability. In *2011 14th Euromicro Conference on Digital System Design*, pages 809–813. IEEE Computer Society, 2011.

[Pan99] J. Pan. The dimensionality of failures - a fault model for characterizing software robustness. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, 1999.

[PKS99] J. Pan, P. Koopman, and D. Siewiorek. A dimensionality model approach to testing and improving software robustness. In *Proceedings of the 1999 IEEE AUTOTESTCON*, pages 493–501. IEEE Computer Society, 1999.

[PSC$^+$11] K. Pattabiraman, G. P. Saggese, D. Chen, Z. Kalbarczyk, and R. Iyer. Automated derivation of application-specific error detectors using dynamic analysis. *IEEE Transactions on Dependable and Secure Computing*, 8(5):640–655, 2011.

[PTAB14] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas. GemFI: A fault injection tool for studying the behavior of applications on unreliable substrates. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 622–629. IEEE Computer Society, 2014.

[RCMM07] P. Racunas, K. Constantinides, S. Manne, and S. S. Mukherjee. Perturbation-based fault screening. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 169–180. IEEE Computer Society, 2007.

[RCV$^+$05] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August. SWIFT: Software implemented fault tolerance. In *Proceedings of the 2005 3rd International Symposium on Code Generation and Optimization*, CGO '05, pages 243–254. IEEE Computer Society, 2005.

[RGYdA08] J. C. Ruiz, P. Gil, P. Yuste, and D. de Andrés. Dependability benchmarking of automotive control systems. *Dependability Benchmarking for Computer Systems*, pages 111–140, 2008.

[RPG15] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan. Characterizing the impact of intermittent hardware faults on programs. *IEEE Transactions on Reliability*, 64(1):297–310, 2015.

[RRV04] M. Rebaudengo, M. S. Reorda, and M. Violante. A new approach to software-implemented fault tolerance. *Journal of Electronic Testing*, 20(4):433–437, 2004.

[SAJK14] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson. A study of the impact of bit-flip errors on programs compiled with different optimization levels. In *2014 10th European Dependable Computing Conference (EDCC)*, pages 146–157. IEEE Computer Society, 2014.

[SAJK15] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson. A comparison of inject-on-read and inject-on-write in ISA-level fault injection. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 178–189. IEEE Computer Society, 2015.

[SBK10a] D. Skarin, R. Barbosa, and J. Karlsson. Comparing and validating measurements of dependability attributes. In *Proceedings of the 2010 European Dependable Computing Conference*, EDCC '10, pages 3–12. IEEE Computer Society, 2010.

[SBK10b] D. Skarin, R. Barbosa, and J. Karlsson. GOOFI-2: A tool for experimental dependability assessment. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pages 557–562. IEEE Computer Society, 2010.

[SBS15] H. Schirmeier, C. Borchert, and O. Spinczyk. Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors. In *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '15, pages 319–330. IEEE Computer Society, 2015.

[Sch07] D. K. Schroder. Negative bias temperature instability: What do we understand? *Microelectronics Reliability*, 47(6):841–852, 2007.

[Sem07] Freescale Semiconductor. MPC565 reference manual, 2007.

[SHD$^+$15] H. Schirmeier, M. Hoffmann, C. Dietrich, M. Lenz, D. Lohmann, and O. Spinczyk. FAIL*: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 245–255. IEEE Computer Society, 2015.

[SHK$^+$12] H. Schirmeier, M. Hoffmann, R. Kapitza, D. Lohmann, and O. Spinczyk. FAIL*: Towards a versatile fault-injection experiment framework. In *ARCS Workshops (ARCS), 2012*, pages 1–5. IEEE Computer Society, 2012.

[SJPB95] D. T. Smith, B. W. Johnson, J. A. Profeta, and D. G. Bozzolo. A method to determine equivalent fault classes for permanent and transient faults. In *Proceedings of the 1995 Annual Reliability and Maintainability Symposium*, pages 418–424. IEEE Computer Society, 1995.

[SK08] D. Skarin and J. Karlsson. Software implemented detection and recovery of soft errors in a brake-by-wire system. In *2008 7th European Dependable Computing Conference*, pages 145–154. IEEE Computer Society, 2008.

[SK09] V. Sridharan and D. R. Kaeli. Eliminating microarchitectural dependency from architectural vulnerability. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pages 117–128. IEEE Computer Society, 2009.

[SRS$^+$12] J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.

[STW00] S. Satoh, Y. Tosaka, and S. A. Wender. Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's. *IEEE Electron Device Letters*, 21(6):310–312, 2000.

[Suh12] J. Suh. Models for soft errors in low-level caches. University of Southern California, 2012.

[SVET10] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren. MODIFI: a model-implemented fault injection tool. In *International Conference on Computer Safety, Reliability, and Security*, pages 210–222. Springer, 2010.

[SVS$^+$88] 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 *[1988] Digest of Papers. The 18th International Symposium on Fault-Tolerant Computing*, pages 102–107. IEEE Computer Society, 1988.

[TI95] T. K. Tsai and R. K. Iyer. Measuring fault tolerance with the FTAPE fault injection tool. In *Proceedings of the 8th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation: Quantitative Evaluation of Computing and Communication Systems*, MMB '95, pages 26–40. Springer-Verlag, 1995.

[TMCW07] E. Touloupis, J. A. Flint Member, V. A. Chouliaras, and D. D. Ward. Study of the effects of SEU-induced faults on a pipeline protected microprocessor. *IEEE Transactions on Computers*, 56(12):1585–1596, 2007.

[TP13] A. Thomas and K. Pattabiraman. LLFI: An intermediate code level fault injector for soft computing applications. In *Workshop on Silicon Errors in Logic System Effects (SELSE)*, 2013.

[VMHA16] R. Venkatagiri, A. Mahmoud, S. K. S. Hari, and S. V. Adve. Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–14. IEEE Computer Society, 2016.

[win] winIDEA iSystems integrated development environment. `http://www.isystem.com/products/software/winidea`. Accessed: 2017-04-22.

[WMP07] N. J. Wang, A. Mahesri, and S. J. Patel. Examining ACE analysis reliability estimates using fault-injection. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 460–469. ACM, 2007.

[YRLG03] P. Yuste, J. C. Ruiz, L. Lemus, and P. Gil. Non-intrusive software-implemented fault injection in embedded systems. In *Proceedings of the 1st Latin-American Symposium on Dependable Computing*, pages 23–38. Springer Berlin Heidelberg, 2003.

[ZCM$^+$96] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. IBM experiments in soft fails in computer electronics (1978-1994). *IBM Journal of Research and Development*, 40(1):3–18, 1996.