Simulation Infrastructure for the Study of Performance / QOS / Energy Efficiency Trade-offs
Υποδομή προσομοίωσης για τη μελέτη της σχέσης μεταξύ επίδοσης / ποιότητας υπηρεσίας / ενεργειακής αποδοτικότητας.

by

Georgios Ioannis Kopanas

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Diploma of Computer and Communication Engineering

at the

UNIVERSITY OF THESSALY

February 2016

Author . . . .
Department of Electrical Engineering and Computer Science
15 February, 2016

Certified by . . . .
Christos D. Antonopoulos
Assistant Professor
Thesis Supervisor

Certified by . . . .
Nikolaos Bellas
Associate Professor
Thesis Supervisor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ενώ η βιομηχανία των ολοκληρωμένων κυκλωμάτων εξελίσσεται, ο αριθμός των τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα αυξάνεται με βάση των νόμο του Μουρ. Παράλληλα, η συχνότητα του ρολογιού εξελίσσεται με πολύ γοργούς ρυθμούς. Αυτό συνέβαινε μέχρι τις αρχές του 21ου αιώνα, όταν οι μικροεπεξεργαστές έφτασαν σε ένα φαινόμενο που ονομάζεται "Power Wall". Η κατανάλωση ενέργειας σε μια μικρή περιοχή είναι περιορισμένη για αρκετούς λόγους, κυρίως λόγω της παραγωγής θερμότητας. Το μαθηματικό μοντέλο της κατανάλωσης ενέργειας στους μικροεπεξεργαστές υποδεικνύει ότι αν συνεχίσουμε να αυξάνουμε το πλήθος των τρανζίστορ κρατώντας τη συχνότητα και τη τάση λειτουργίας σταθερά τότε οι μικροεπεξεργαστές σε μερικά χρόνια θα φτάσουν να έχουν power density αντίστοιχο των πυρηνικών αντιδραστήρων.

Πολλοί μικροεπεξεργαστές έχουν θερμικούς διακόπτες που μειώνουν ακαριαία την δραστηριότητα του κυκλώματος σε περίπτωση υπερθέρμανσης, μειώνοντας τη τάση λειτουργίας και τη συχνότητα. Σύμφωνα με το βιβλίο "Computer Architecture: A Quantitative Approach" των A.Patterson και J.Hennesy: "Η κατανάλωση ενέργειας είναι ένας από τους μεγαλύτερους περιοριστικούς παράγοντες στη σύγχρονη χρήση των τρανζίστορ [1].

Παρατηρώντας το πιο απλό μαθηματικό μοντέλο 1.1 για τη κατανάλωση ενέργειας

$$P = C * V^2 * f \tag{1.1}$$

Βλέπουμε ότι μόνο δύο από τους τρείς παράγοντες επηρεάζουν άμεσα την επίδοση ενός επεξεργαστή, το χωρητικό φορτίο και η συχνότητα, ο τρίτος παράγοντας, η τάση, επηρεάζει μόνο έμμεσα την επίδοση, λόγω του ότι υπάρχει ένα όριο στη τάση για κάθε συχνότητα λειτουργίας κάτω από το οποίο ο κατασκευαστής του επεξεργαστή δεν εγγυάται τη σωστή λειτουργία του, λόγω του κινδύνου για λάθη χρονισμού στα τρανζίστορ.

Αρκετοί ερευνητές τα τελευταία χρόνια προτείνουν τρόπου λειτουργίας των μικροεπεξεργαστών με τάσεις λειτουργίας κάτω από τα όρια των κατασκευαστών. Η διαδικασία αυτή δεν είναι τετριμμένη αφού η λειτουργία των τρανζίστορ σε ένα ολοκληρωμένο κύκλωμα με τάση κάτω από τα όρια μπορεί να προκαλέσει λάθη χρονισμού και απρόβλεπτη συμπεριφορά του κυκλώματος, κάτι το οποίο πρέπει να το λάβουμε υπόψιν μας. Το κέρδος από τη λειτουργία με τάση κάτω από τα όρια είναι ότι η κατανάλωση ενέργειας του μικροεπεξεργαστή μειώνεται ραγδαία αφού εξαρτάται από το τετράγωνο της τάση λειτουργίας.

Επομένως αξιόπιστοι υπολογισμοί κάτω από αναξιόπιστες συνθήκες είναι η επόμενη πρόκληση για την επιστημονική κοινότητα. Για να επιτευχθεί ένας τέτοιος στόχος νέα εργαλεία ανάλυσης της συμπεριφοράς των λαθών σε επίπεδο υλικού και λογισμικού πρέπει να δημιουργηθούν. Η ανάλυση των λαθών μπορεί να μας βοηθήσει να κατασκευάσουμε μια ιεραρχία από μονάδες του επεξεργαστή που χρειάζονται ενίσχυση για να πετύχουν την αξιοπιστία.

Για να πετύχουμε υψηλή κατανόηση των προηγούμενων φαινομένων, νέα εργαλεία πρέπει να κατασκευαστούν και να επεκταθούν τα ήδη υπάρχοντα. Η κύρια συνεισφορά αυτής της διπλωματικής εργασίας είναι η ανάπτυξη μιας ρεαλιστικής υποδομής προσομοίωσης εισαγωγής λαθών που παίρνει υπόψιν τη κατανάλωση ενέργειας και την σύνθεση των εντολών της εφαρμογής.

Με σκοπό, λοιπόν τη μελέτη των αποτελεσμάτων των λαθών χρονισμού σε διάφορες εφαρμογές και συστήματα αρχιτεκτονικής, έχει αναπτυχθεί προσομοιωτής, επεκτείνοντας τον Gem5 [2], στον οποίο ο χρήστης έχει τη δυνατότητα να παρέχει ένα αρχείο εισόδου στο οποίο καθορίζει ποιες εντολές και τι είδους λάθη προκύπτει να εισαχθούν στην εκτέλεση. Οι περιορισμοί αυτής της προσέγγισης είναι ότι δεν λαμβάνεται υπόψιν ο έλεγχος ροής η αναλογίες διαφορετικών εντολών της εφαρμογής για να παραχθούν πιο ρεαλιστικές πιθανότητες. Το πρόβλημα δημιουργείτε λόγο του ότι ο χρήστης καθορίζει στατικά πριν τη προσομοίωση τα λάθη. Για παράδειγμα αν μια εντολή πολλαπλασιασμού βρίσκεται σε ένα επαναληπτικό βρόγχο που το πλήθος των επαναλήψεων καθορίζεται από τα δεδομένα είσοδο, δεν υπάρχει τρόπος να καθοριστεί πριν το χρόνο εκτέλεσης το μείγμα εντολών

Το πρώτο μέρος αυτής της διπλωματικής εργασίας ήταν η επέκταση του προαναφερόμενου συστήματος παρομοίωσης και η δημιουργία εργαλείου που δυναμικά εισάγει λάθη κάτω από ρεαλιστικές πιθανότητες σε όλο το pipeline μιας ARM αρχιτεκτονικής. Πρώτων μεταφέραμε την ήδη υπάρχουσα λειτουργικότητα εισαγωγής λαθών στο ARM ISA . Δεύτερων δώσαμε στον χρήστη τη δυνατότητα να επιλέγει διαφορετικές πιθανότητες εισαγωγής λαθών για κάθε pipeline stage και για κάθε κλάση εντολών στις οποίες τα λάθη θα εισάγονται δυναμικά κατά τη διάρκεια του χρόνου εκτέλεσης. Επίσης ταιριάξαμε το εργαλείο που φτιάξαμε με τις δυνατότητες DVFS του Gem5 για να προσομοιώσουμε ετερογενείς αρχιτεκτονικές με διαφορετικές τάσεις λειτουργείας σε κάθε πυρήνα ώστε να υπάρχουν διαφορετικές πιθανότητες λαθών ανά πυρήνα. Το τελευταίο μέρος ήταν να υπολογίσουμε τον κατανάλωση ενέργειας χρησιμοποιώντας στατιστικά δεδομένα από τη προσομοίωση χρησιμοποιώντας το McPAT, ένα εργαλείο με μαθηματικές μεθόδους που προσεγγιστικά υπολογίζει την κατανάλωση ενέργειας πολυπύρηνων αρχιτεκτονικών.

Όλα αυτά τα βήματα έγιναν με στόχο να δημιουργήσουμε μια υποδομή προσομοίωσης που μπορεί να χρησιμοποιηθείς για τη μελέτη της σχέσεις μεταξύ επίδοσης / ποιότητας υπηρεσίας / ενεργειακής αποδοτικότητας για διαφορετικές ετερογενείς αρχιτεκτονικές που υποστηρίζουν μη-έμπιστους πυρήνες.

Αυτό το έγγραφο αποτελείτε από δύο μέρη. Το πρώτο εισάγει τον αναγνώστη στο θεωρητικό υπόβαθρό και το δεύτερο περιγράφει το σύστημα προσομοίωσης και τη πειραματική μελέτη. Στο πρώτο μέρος, τα κεφάλαια 1-3 παρουσιάζουμε την έννοια του δεπενδαβιλιτψ, το μοντέλο των λαθών που πρόκειται να χρησιμοποιήσουμε και σχετική δουλειά που έχει γίνει από άλλους. Στο δεύτερο μέρος, κεφάλαια 4-5, περιγράφουμε το σύστημα προσομοίωσης και τη πειραματική μελέτη παρουσιάζονται τα αποτελέσματα των μετρήσεων που κάναμε.

Chapter 2

Introduction

While the CPU industry is evolving, the number of transistors inside the integrated circuit are being driven by the well know Moore's Law. Simultaneously, the clock frequency is very rapidly increasing. This was the case until the start of the 21st century, when the CPU manufacturing reached a phenomenon called "Power Wall". The power consumption in a small area that we can produce is limited for various reasons, mainly, because of heat emission. The power consumption equation indicates that if we keep increasing the crowd of the transistors and the frequency in which they are operating by keeping the voltage constant, then CPUs are going to reach the power density of a nuclear reactor in a matter of years.

A lot of microprocessors have thermal switches which drop instantly the activity of a chip in case of overheating by reducing the voltage operating point and frequency. According to the A.Patterson and J.Hennesy book "Computer Architecture: A Quantitative Approach": Power consumption is the greatest limiting factor in the modern use of transistors [1].

By observing the most simple mathematical model 2.1 of power consumption

$$P = C * V^2 * f \tag{2.1}$$

we can see that only two of the three factors affect directly the performance of a microprocessor, the capacitive load (symbol C) and the frequency (symbol f), the third factor, voltage (symbol V), affects performance only indirectly, because there is a threshold voltage for every operating frequency of a given microprocessor in which the behavior of the chip is guaranteed without the danger of timing errors. A lot of research in the past years is suggesting ways to operate microprocessors in below-nominal voltage values. This is not trivial since in below-nominal voltage values the transistors may come to timing errors and have unpredicted behavior, which we must take into account. The gains of this below-nominal operation of the microprocessor is that the power consumption drops rapidly since it depends by the square of the operating voltage.

Reliable computing under unreliable circumstances is one of the next challenges the computing community must solve. To achieve such a difficult task we need to perform a thorough analysis of the way hardware faults manifest errors to architectural components and how on their turn affect the applications behavior. The analysis of the faults may help us construct a hierarchy of target-modules that need to be enhanced in order to achieve robustness.

To achieve a high grasp of the previously mentioned phenomenon new tools had to be constructed and the existing ones to be extended by adding new functionalities. The main contribution of this thesis is to provide a realistic fault injection infrastructure which takes into account the power consumption and the application's instruction mix.

In order to study the effects of timing faults on various applications we have developed such a framework on top of Gem5 [2], in this framework the user is able to provide a set-up file which determines the specific instructions and the type of of faults that are going to be injected. The restrictions of this approach is that you cannot take into consideration the control flow and the instruction mix of the application to provide realistic probabilities of faults, since they are statically determined before the simulation i.e if a multiplication instruction is to be executed inside a loop which the iterations depend in the input data, there is no easy way to determine the instruction mix.

The first part of this diploma thesis was to extend this framework and create a tool which dynamically injects faults under realistic probabilities throughout the pipeline of an ARM architecture. Firstly, we imported the previously existing fault injection functionality in the ARM instruction set. Secondly, we provided the user an interface and a functionality to pick different probabilities per pipeline stage and per instruction class in which the errors are going to be manifested dynamically during run-time. Also we coupled the tool that we created with the DVFS functionality of Gem5 to simulate heterogeneous cores with different voltage levels that have various fault injection probabilities per core. The last part was to compute the power consumption by using statistical data of the simulation by using McPAT [3], a tool that has mathematical functions which approximately computes the power performance of many core and multi-core architectures.

All these steps where made so we can have a simulation infrastructure that can be used to study the performance/QOS/Energy efficiency trade-offs of different heterogeneous architectures that have unreliable cores.

This document is constructed in two blocks. The first one introduces the theoretical background and the second describes our framework and the experimental evaluation. In the first part, chapters 1-3 we discuss the concept of dependability , the fault model that we are going to use and related work done by others. In the second part chapters 4-6 we describe the framework that we created and the experimental evaluation by displaying the result of some experimental campaigns.

Chapter 3

Background

As mentioned in the previous chapter we need achieve reliable computing under unreliable circumstances. In order to do that we need to set up systems that are highly tolerant to faults. Measuring the fault tolerance of a system is not trivial. In this chapter we will introduce definitions which will be used to help us categorize faults, measure fault tolerance and connect these to power consumption.

3.1 Dependability

Dependability is a generic concept including as special case such attributes as reliability, availability, safety, integrity, maintainability. The original definition of dependability is the ability to deliver service that can justifiably be trusted. An alternate definition is the ability of a system to avoid service failures that are more frequent and more severe than is acceptable[4]. The second definition of dependability suits more in our needs. For the purpose of clarification we will formally define the attributes that dependability encompasses

- **availability**: readiness for correct service;
- **reliability**: continuity of correct service;
- **safety**: absence of catastrophic consequences on the user(s) and the environment;

- **integrity**: absence of improper system alterations;
- **maintainability**: ability to undergo modifications, and repairs.

3.1.1 Factors of Dependability

We will generalize the reasons why a system may not perform as it is intended to. The causes and the reasons that a system deviates from it's intended behavior are called factors of dependability[5].

- **Fault** is a physical defect, imperfection, or flaw that occurs within some hardware or software component;
- **Error** is a deviation from accuracy or correctness and is the manifestation of a fault;
- **Failure**: is the non-performance of some action that is due or expected.

We have to mention here that there is a confusion in the bibliography between terms, fault and error, which can be used vice - versa. In our study faults have secondary importance since we don't take into consideration the reasons errors occur, but we evaluate systems based on the manifestation of these faults, for that reason we will refer only to errors from now on, even if they are mentioned as faults.

3.1.2 Methods of Unreliable Computing under Unreliable Environment

As mentioned in [6], fault tolerance uses methods to ensure that faults do not cause failure. Such methods are:

- **Detecting Errors**: is primarily dependent on redundancy, including natural redundancy (e.g. error detecting codes in hardware, executable assertions in software) and artificial redundancy (e.g. modular redundancy in hardware and N-version programming in software);

- **Assessing Damage**: This step involves assessing the extent of the damage caused by the fault on the state of the computation, and making a decision on what recovery action must be taken;

- **Error Recovery**: This step involves correcting the current state of the computation according to the findings of the damage assessment step, and resuming the computation;

- **Fault Removal**: While the three preceding steps deal with the current computation and are performed on-line for the sake of salvaging the current computation, this step deals with the system itself, and can be performed off-line. It consists of identifying the fault that caused the error and removing it.

These methods are based on traditional architectures. In the past decade more modern and exotic architectures [7] and programming models are assuming that processors can function in two states, reliable and unreliable, in the reliable state the processor takes all necessary precautions to run the code reliably, but the power consumption and the speed are secondary goals. With this assumption, abusing the programmers knowledge on assessing in which areas of the code faults are acceptable, makes system utilize their hardware much more efficiently.

3.1.3 Methods of Decreasing Power Consumption

- **Dynamic frequency scaling** (also known as CPU throttling) is a technique whereby the frequency of a microprocessor can be automatically adjusted "on the fly," either to conserve power or to reduce the amount of heat generated by the chip.

- **Dynamic voltage scaling** is a power management technique, where the voltage used in a component is increased or decreased, depending upon circumstances. Dynamic voltage scaling to increase voltage is known as overvolting; dynamic voltage scaling to decrease voltage is known as undervolting, in order to conserve power. Overvolting is done in order to increase computer performance or reliability.

- **Dynamic voltage and frequency scaling** (DVFS) is the combination of the previous techniques and it is widely used in modern CPUs.

From equation 2.1 the reader can easily deduce, firstly that decreasing frequency linearly decreases power consumption while the execution time becomes longer. Secondly, decreasing voltage exponentially decreases power consumption by affecting reliability since timing errors of the transistors become more probable.

3.2 Analysis of Instruction-level Vulnerability

Recent studies show that errors don't manifest uniformly during the execution of different instructions. In fact, instruction level vulnerability studies partition instructions into three equivalent classes based on their error probabilities while varying the voltage and the temperature[8]. Based on these studies, all exercised instruction in the integer pipeline of ARM ISA are partitioned into three classes for the full range of operating condition: (i) the logical and arithmetic instructions, (ii) the memory instructions, and (iii) the multiply and divide instructions.

3.3 Gem5 Simulator

For the development of our simulator we extended Gem5[2], a popular open-source system simulator. Gem5 is a versatile tool that proved to have all the features that we need for the simulation part of the infrastructure we are setting up.

3.3.1 Gem5 Features

In this section we will present some of the key features that Gem5 encompasses. For a detailed documentation the reader is redirected to the official websites.

- **Multiple interchangeable CPU models.** Gem5 provides four interchangeable CPU models: a simple one-CPI CPU; a detailed model of an in-order CPU, and a detailed model of an out-of-order CPU. The CPU models use a common high-level ISA description. In addition, gem5 features a KVM-based CPU that uses virtualisation to accelerate simulation.

- **Homogeneous and heterogeneous multi-core.** The CPU models and caches can be combined in arbitrary configurations, creating homogeneous, and heterogeneous multi-core systems. This features is one of the important tools that make simulating exotic architecture that where described in earlier sections to be simulated in an accurate manner.

- **Multiple ISA support.** Gem5 decouples ISA semantics from its timing CPU models, by that enabling different CPU models is ISA-independent. Currently gem5 supports the Alpha, ARM, SPARC, MIPS, POWER and x86 ISAs.

- **Power and energy modeling.** Gem5's objects are arranged in OS-visible power and clock domains, enabling a range of experiments in power and energy efficiency. With out-of-the-box support for OS-controller Dynamic Voltage and Frequency (DVFS) scaling, gem5 provides a complete platform for research in future energy-efficient systems.

3.3.2 GemFI, a Fault Injection extension

GemFI[9] is a fault injection tool based on Gem5, a cycle accurate full system simulator. It provides fault injection methods with a predefined set of faults by the user with an input file. Every fault is defined as a set of three attributes, the number of the instruction that the fault is going to be injected, the pipeline stage and the type of fault. GemFI supports 3 kind of faults:

1. *"stack-at-one"* and *"stack-at-zero"*, which sets all the bits of a value to 0 or 1

2. *"flip bit"* which flipping the running value at bit locations

3. *"immediate value"* which assigns an immediate value provided by the user.

It also supports multiple processor models and allows fault injection in both functional and cycle-accurate simulations. Moreover, it facilitates the parallel execution of campaign experiments on a network of workstations. GemFI is developed using C++ and Python. Currently it fully supports only the Alpha instruction set architecture. One key contribution of this thesis is extending GemFI to ARM ISA.

3.4 Power Consumption Model

To assess the gains of unreliable computing in power consumption we need a model. We are going to use McPAT[3], a well-known framework for power, area and timing modeling that supports comprehensive design space exploration for multicore and manycore processor configurations ranging from 90nm to 22nm and beyond. During the experimental process we observed that McPAT is a tool that will provide the user a generic sense on the magnitude of power consumption and not predict the actual power consumption a processor is going to have, although since our study is comparative McPAT model is fitting it's purpose.

Chapter 4

Related Work

Fault tolerant systems is a concept that needs the development of new architectures and programming models. In this chapter we will present the reader, firstly, the most recent work that has been done in the computer architecture field, and secondly, the already developed tools for simulating executions that incorporate fault injection.

4.1 Fault Tolerant Architectures

While researchers tried to solve the problem of unreliable computing they designed fundamentally novel architectures which make sure either to detect and resolve all errors either to make sure that errors happen only in non-critical parts of the execution which will give error acceptable by the user.

In the first case a trademark architecture is RAZOR which ensures error detection and correction of timing errors of the critical path due to near-threshold operation of the processors [10]. The main idea behind this is coupling each flip-flop in the design with a so-called shadow latch which is controlled by a delayed clock. When error is detected from the shadow latch an error signal is raised, the circuit is delayed for one cycle so the next pipeline stage will take input from the latch instead of the miss-calculated flip-flop.

For the second case, ERSA architecture[7] is featuring a combination of cheap, unreliable compute power together with a small fraction of reliable processor cores for running system software, controlling application flow, and recovering from errors generated on unreliable cores.

Chapter 5

The Simulator

5.1 Fault Model

In this section we will present the Fault model that we chose as the base for our implementation. During our study we wanted to focus on errors that occur during voltage and temperature variations. We want to assess how processors and applications behave in below-nominal voltage values, in result, these kind of errors simulate better our case of study.

In general, proving the sufficiency of a fault model is very difficult. It is more realistic to assume that a fault model is sufficient and justify this assumption to the greatest extent possible with experimental and historical data or results published in literature. That is the reason that lead us to create our fault model based on the experimental research[8], which was the first that introduced the notion of instruction-level vulnerability(ILV) to expose variation and its effects to the software stack. To compute ILV, they quantified the effect of voltage and temperature variations on the performance and power of a 32-bit, RISC, in-order processor in 65nm TSMC technology at the level of individual instructions. Results show 3.4ns (68FO4) delay variation and 26.7x power variation among instructions, and across extreme corners. Their analysis shows that ILV is not uniform across the instruction set. In fact, ILV data partitions instructions into three equivalence classes.

Based on that study we decided to implement a model in which the user had the capability to provide different fault probabilities in each pipeline stage of the following:

- **Fetch**: This fault corrupts the binary form of the instruction;
- **Decode**: This fault covers corruptions of the opcode field;
- **Execution**: The execution fault corrupts the results of the ALU unit, in arithmetic instructions this means wrong operation results, in memory instructions means wrong calculation of the next-PC;
- **Memory(Load/Store)**: corrupts the read/write values of the memory;

That is not enough since ILV is not uniform across the instruction set, through experimental studies we partitioned the instructions into three equivalent classes, resulting that every class has different fault probability for every pipeline stage:

- **Logical & Arithmetic**: Includes all instructions of addition substraction logical and binary operations;
- **Multiplication & Division**: Includes all instructions of multiplication and division ;
- **Memory**: Includes all memory operations;

A user can provide the error probabilities in a very intuitive way through the setup python script that Gem5 already uses to set-up the architecture of the system, below we provide an example, which shows how the user can provide probabilities for a 2-core system.

```
[...]
system.cpu[0].p_Iew_arithm=0.021
system.cpu[1].p_Iew_arithm=0.022
system.cpu[0].p_Iew_loadstore=0.023
system.cpu[1].p_Iew_loadstore=0.024
system.cpu[0].p_Iew_muldiv=0.025
system.cpu[1].p_Iew_muldiv=0.026

system.cpu[0].p_Lds_arithm=0
system.cpu[1].p_Lds_arithm=0
system.cpu[0].p_Lds_loadstore=0.029
system.cpu[1].p_Lds_loadstore=0.030
system.cpu[0].p_Lds_muldiv=0
system.cpu[1].p_Lds_muldiv=0

system.cpu[0].p_Decode_arithm=0.031
system.cpu[1].p_Decode_arithm=0.032
system.cpu[0].p_Decode_loadstore=0.033
system.cpu[1].p_Decode_loadstore=0.034
system.cpu[0].p_Decode_muldiv=0.035
system.cpu[1].p_Decode_muldiv=0.036

system.cpu[0].p_Fetch=0.037
system.cpu[1].p_Fetch=0.038
[...]
```

Since the Fetch stage is the same for all instructions the user cannot provide different probabilities for the 3 classes.

As we explained in previous chapters to simulate dependability errors we need to have different three kind of errors "all-one, all zero", "immediate value" and "flip bit", which we support, see section 3.3.2. For the shake of simplicity, without sacrificing functionality we introduce equal probabilities for any of these kind of errors in the execution stage, but we keep only "1-flip-bit" errors in all other pipeline stages.

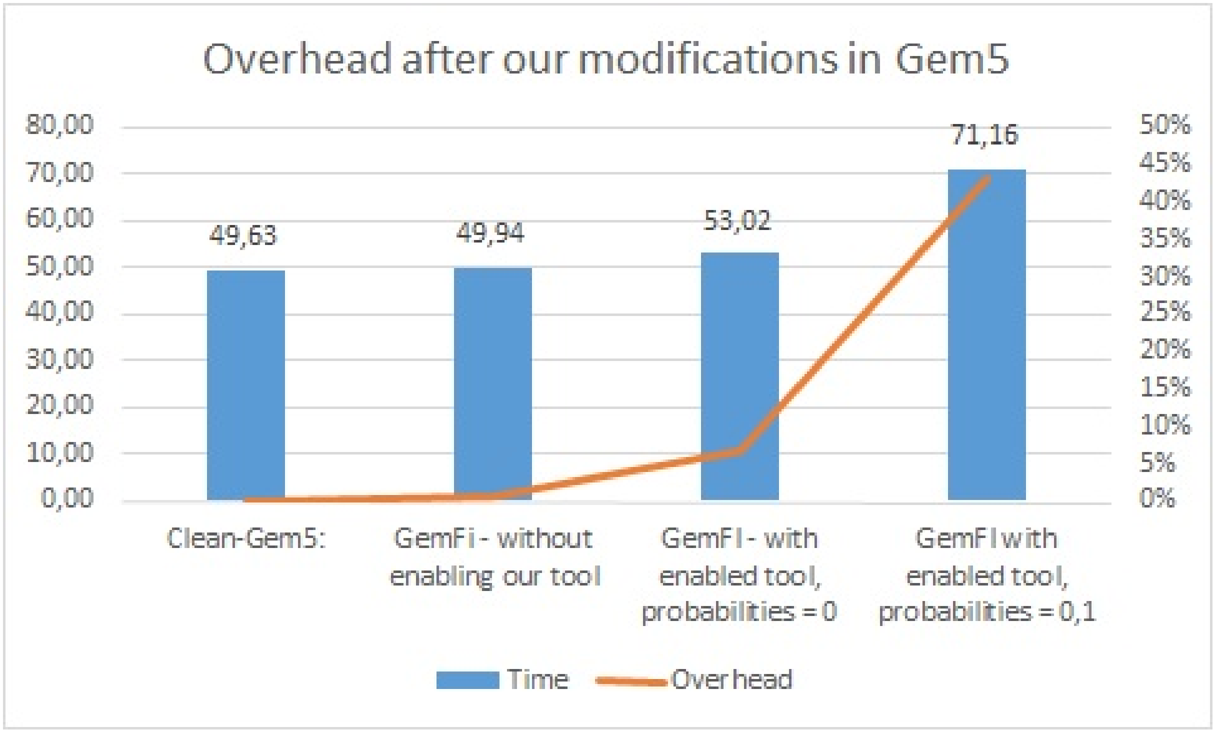


Figure 5-1: GemFI Overhead

5.1.1 Time Overhead

When we finished developing the fault injection model, we wanted to evaluate the time overhead that we added in the original Gem5 simulator. To do that we made 10 executions of 4 diffrent runs of a 32x32 matrix multiplication:

- Clean Gem5, Gem5 without any modifications
- GemFI without enabling our tool
- GemFI with enabled tool, but probabilies = 0
- GemFI with enabled tool and probabilities = 0.1

In the last case we need to mention that the fault injection were done in a way that would not change the results so the application would run smoothly and not crash so the time can be compared in a fair way. The results5-1 show that the overhead is minimal, close to non existant when the tool is disabled something very critical as a goal for our implementation, and when the probabilities are close to 0 the overhead is satisfactory.

5.2 Check-pointing

Simulations offer many advantages: ease of use, portability, reproducibility, low cost etc. One of those is the ability to retrieve a snapshot of the state of the system to the hard drive and retrieve from that point whenever the user wants. This gives the ability to the user to study in depth special cases of consecutive errors by reproducing and take a deeper look to a specific series of faults that were introduced and led to a specific program behavior. Although Gem4 already provided this kind of mechanism there were limitations which did not suit our purposes.

Checkpoint on full system detailed simulation was achieved with two ways. For the first method CPUS were switched from detailed to atomic mode create the checkpoint and afterwards switching again from atomic to detailed mode in order to continue the simulation. For this to be done the pipeline stages on the detailed mode where flushed prior taking the checkpoint and thus there could be a potential accuracy loss in our fault injection framework. The other method was achieved by simulating MOESI hammer cache coherency protocol. This method did not switched between detailed and atomic modes however the simulation time increased dramatically.

Due to the previous limitations we had to turn to a Linux based checkpoint package which checkpointed the simulator's state from an outer scope. DMTCP is distributed under the terms of Lesser GNU Public License (LGPL) and supports checkpointing the state of multiple of multiple simultaneous applications, including multi-threaded and distributed applications.

The main reason for choosing DMTCP was based on the ability to take checkpoints not only inside of the simulator by calling functions given by the API of DMTCP but also outside of the simulator. The ability to invoke DMTCP inside of the simulator gave us the opportunity to keep the already existed checkpointing front-end of the Gem5 simulator (special instruction added on ISA so that applications may call a

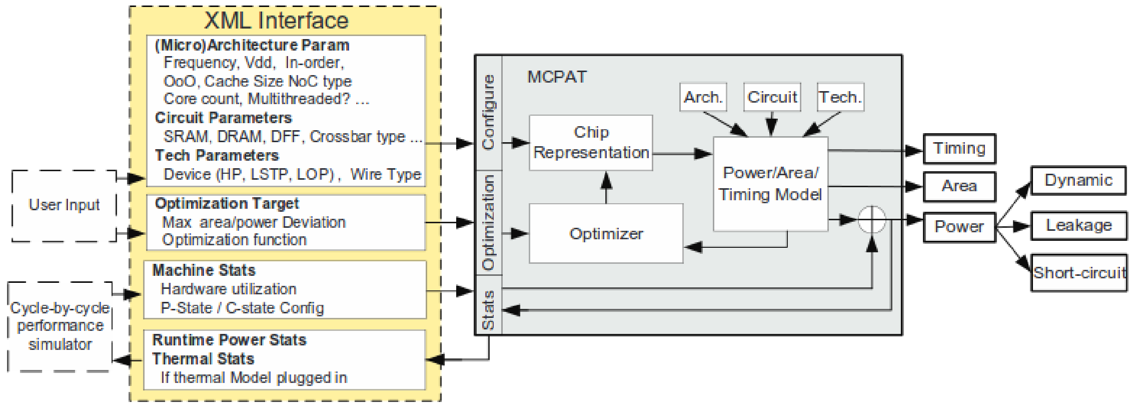


Figure 5-2: Block diagram of the McPAT framework

checkpoint internally) and alter the back-end of the checkpointing method (use the DMTCP API to checkpoint instead).

Another important feature DMTCP incorporates is the fact that can checkpoint multiple applications that run simultaneously. A DMCTP enabled application consists of all processes connected to a given coordinator. To have two simultaneous DMTCP computations on the same host, you will need two DMTCP coordinators listening to different port numbers. The command dmtcp_coordinator generates a new coordinator.

5.3 Calculating Power Consumpion

As we mentioned earlier we will use McPAT to calculate power consumption of our simulation. McPAT is the first integrated power, area, and timing modeling framework for multithreaded and multicore/manycore processors. It is designed to work with a variety of processor performance simulators (and thermal simulators, etc.) over a large range of technology generations. McPAT allows a user to specify low-level configuration details. It also provides default values when the user decides to specify only high-level architectural parameters. The block diagram provided by the developers of McPAT gives a good sense of how that tool can be used and coupled with simulators.5-2

Eventhough McPAT provides a lot of functionality and gives the user intuitive ways to provide input this tool is created in a generic way so it can work with all availiable simulator that can provide the data that it needs to run.

So we modified a script to convert Gem5 simulation output statistics to McPAT compatible Machine statistics inputs. We also created a template that gives all the availiable information for an ARM A15 architecture with 2 cores as Architecture input parameters to McPAT.

Chapter 6

Running Experiments

After we finished the development of the tools that we mentioned in the previous chapters, we run a series of experiments to validate and demonstrate the capabilities of the infrastructure that we created.

6.1 Setting up the Workstations

As previously mentioned, one of the main disadvantages that fault injection simulations has is the huge time consumption, which is measured close to 10.000 times slower than a native execution. For the purpose of our experiments we need to run 2000 experiments.

To speed up the experiments we had in our disposal 22 computers from the laboratory of the University of Thessaly. Each PC has an Intel processor (Xeon CPU E5606) with four cores clocked at 2.13 Ghz and 6Gb of memory (RAM). All PC's have access to a Network File system(NFS) and a local file system(10-1). Since the experiments are independent from each other, in the best case scenario we can run 88 experiments simultaneously, one in each core. To control the campaign of the experiments we used multiple Linux tools, also python and bash script language.

First we used one of the computers as the main node of our computation. That node compiled and setted up in NFS all the things that the simulation needs: scripts ,executable, image disks etc. Once these are ready the "coordinator" script 6-1 is

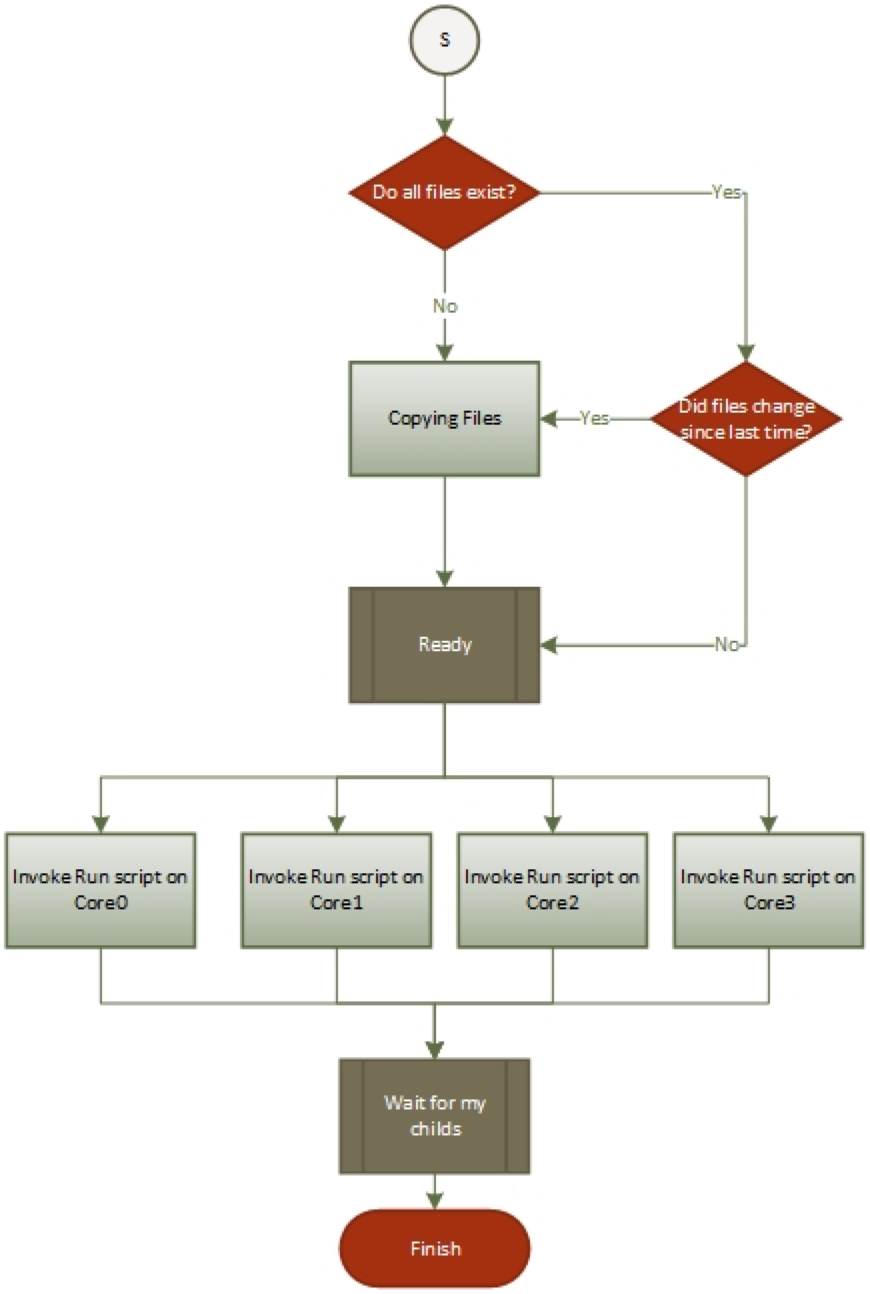

S
Do all files exist?
Yes
No
Copying Files
Yes
Did files change since last time?
Ready
No
Invoke Run script on Core0
Invoke Run script on Core1
Invoke Run script on Core2
Invoke Run script on Core3
Wait for my childs
Finish

Figure 6-1: The control flow of the coordinator script

launched that copies all these files to the local HDD and runs 4 children which execute the "Run" script that is responsible for running an individual set of experiment in the specific core.

Each set of experiments consists from one full simulation that is responsible to boot the OS on the simulator and create a checkpoint just before the region of interest in the code. Then we restart the simulation from that point as many times we need to complete the number of experiments this core is responsible.

It is worth mentioning that a third script is running on the background which checks periodically if an experiment is taking too long (stuck in an infinite loop), if the output of the experiments exceeds some limits or if another user is logged in the machine; on all cases the experiment is killed and a message is printed on the output. On the last case additionally a checkpoint is created for each experiment in order to respect the other user.

Besides the scripts the following linux tools where used to help the workstation setup:

- **Wake on Lan (WON):** remotely boot each PC by sending a special Packet to the Network card;
- **Secure Shell (SSH):** is a cryptographic (encrypted) network protocol to allow remote login and other network services to operate securely over an unsecured network.
- **Secure File Transfer (SFTP)**: is a standard network protocol used to transfer computer files between a client and server on a computer network.

Apart of running experiments in parallel we tried to reduce the duration of each experiment individually. This was achieved by a major optimization. Gem5 has developed a special instruction which allows to switch between CPU models. When the instruction is executed the simulator pauses and a Python script initializes the next CPU model and then restored the execution with the new CPU model. In some cases except the of the CPU model the memory model is also switched (atomic -

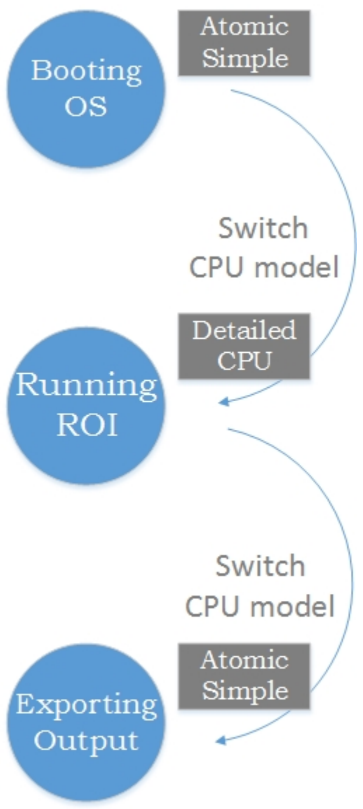
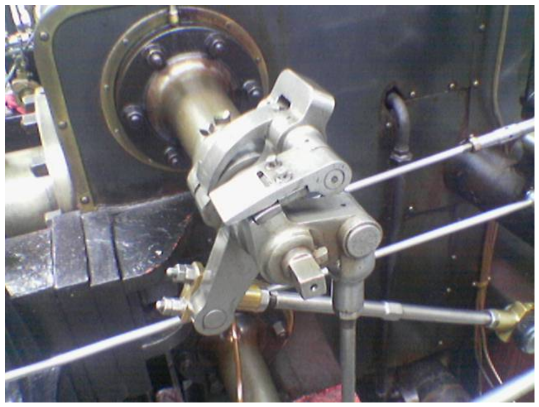


Figure 6-2: Optimize the speed of the experiment by switching CPUs

simple). In our case everything that is not happening with fault injection enable is not a Region of Interest (ROI) so we run it on Atomic Simple mode and then we switch when fault injection is enabled, similarly we switch when the ROI is over and we want to export the results to the HDD. The process is described in the figure 6-2

6.2 Sobel Filter Experiment

The Sobel filter, is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector


(a) Input Image

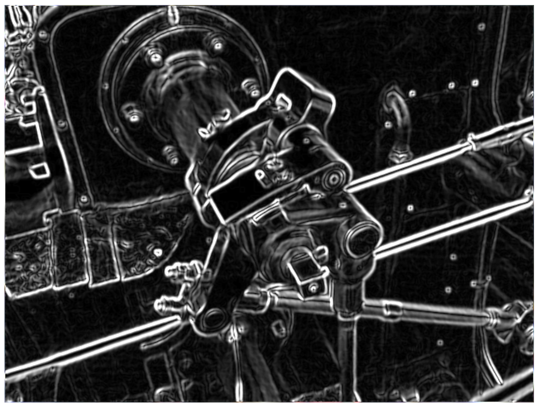

(b) Output Image

Figure 6-3: Sobel Operation example

6.2.1 The mathematical formula

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:

$$G_x = \begin{bmatrix} -1 & 0 & -1 \\ -2 & 0 & -2 \\ -1 & 0 & -1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Where * here denotes the 2-dimensional signal processing convolution operation and A is the image array. The x-coordinate is defined here as increasing in the "right"-direction, and the y-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$G = \sqrt{G_y^2 + G_x^2}$$

An example Output of the Sobel Operation is shown in the figure 6-3

6.2.2 Fault Injection Results

In this section we will present the results from the experimental fault injection campaign that we did by using a simple Sobel filter application that takes as an input a black&white image with dimensions 512×512 and has as an output the filtered image written in a headless raw binary file.

We partition our results in 4 categories:

- **Crashed**: Includes all experiments which fail to terminate successfully and dont produce an output file;
- **Inexact**: Includes all experiments which have a visible alteration of the result, we consider as visible alteration of the result a PSNR lower than 30db ;
- **Correct**: Includes all experiments that have alteration in the results but we consider that it is not visible to the user, which means a PSANR greater than 30db;
- **Bit-wise Exact**: Includes all experiments which produce bit-wise exact results;

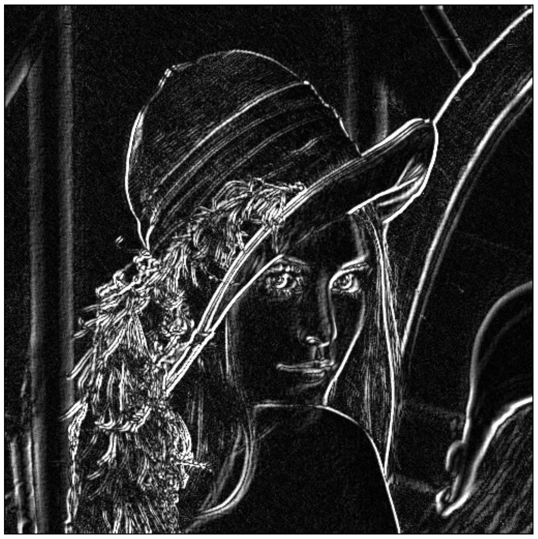
We did two series of experiments, the first one has the following error probabilities in the ROI where the Fault Injection tool is enabled:

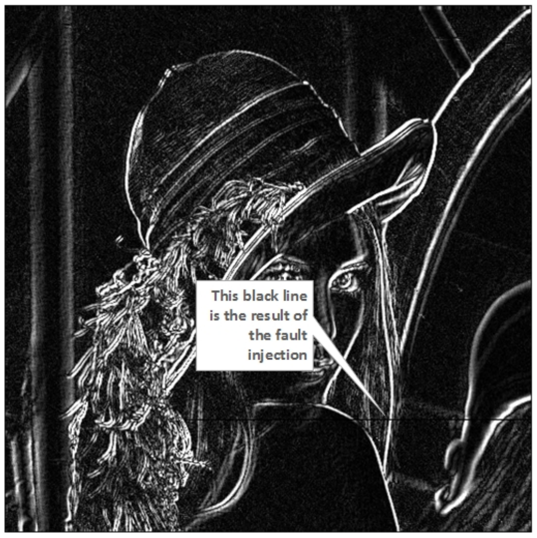
```
[...]
system.cpu[i].p_Iew_arithm=1e-08          system.cpu[i].p_Iew_loadstore=1e-08
system.cpu[i].p_Iew_muldiv=1e-08          system.cpu[i].p_Lds_arithm=0
system.cpu[i].p_Lds_loadstore=1e-08       system.cpu[i].p_Lds_muldiv=0
system.cpu[i].p_Decode_arithm=1e-08       system.cpu[i].p_Decode_loadstore=1e-08
system.cpu[i].p_Decode_muldiv=1e-08       system.cpu[i].p_Fetch=1e-08
[...]
```

The fault injection tool is enabled at the start of the computations for the Sobel filter and we disable it the moment we finish. The output is written in a safe environment without faults.

The distribution of the 2100 executions in the previously mentioned clashes are shown in figure 6-5 (a)

In the second series of experiments we lowered the probability of error in the following values:


(a) Golden Output Image

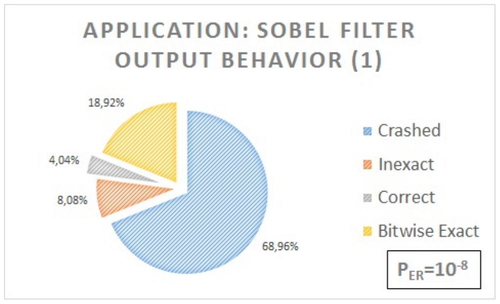


(b) FI Output Image

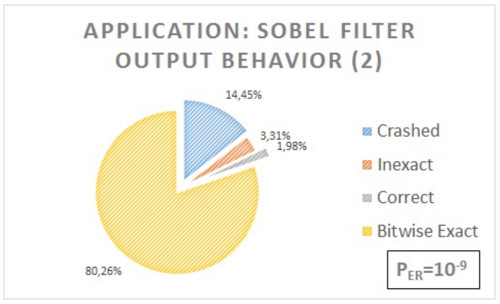
Figure 6-4: Image (a) is the output of Sobel that we used for the experiments campaign without any fault injected. Image (b) is an example output that is corrupted because of the fault injection, if you look closely you can see a horizontal black line that is not supposed to be there.

```
[...]
system.cpu[i].p_Iew_arithm=1e-09          system.cpu[i].p_Iew_loadstore=1e-09
system.cpu[i].p_Iew_muldiv=1e-09          system.cpu[i].p_Lds_arithm=0
system.cpu[i].p_Lds_loadstore=1e-09       system.cpu[i].p_Lds_muldiv=0
system.cpu[i].p_Decode_arithm=1e-09       system.cpu[i].p_Decode_loadstore=1e-09
system.cpu[i].p_Decode_muldiv=1e-09       system.cpu[i].p_Fetch=1e-09
[...]
```

Similarly for the new probabilities we ran 2100 executions and the results are shown in figure 6-5 (b).



(a) Series of 2100 experiments with error probability of 1e-08



(b) Series of 2100 experiments with error probability of 1e-09

Figure 6-5

Bibliography

[1] J. L. Hennessy and A.Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann; 4 edition, (September 27, 2006).

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[3] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on.* IEEE, 2009, pp. 469–480.

[4] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[5] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[6] H. H. Ammar, B. Cukic, A. Mili, and C. Fuhrman, "A comparative analysis of hardware and software fault tolerance: Impact on software reliability engineering," *Annals of Software Engineering*, vol. 10, no. 1-4, pp. 103–150, 2000.

[7] H. Cho, L. Leem, and S. Mitra, "Ersa: Error resilient system architecture for probabilistic applications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 4, pp. 546–558, 2012.

[8] A. Rahimi, L. Benini, and R. K. Gupta, "Analysis of instruction-level vulnerability to dynamic voltage and temperature variations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.* IEEE, 2012, pp. 1102–1105.

[9] 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 *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on.* IEEE, 2014, pp. 622–629.

[10] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on.* IEEE, 2003, pp. 7–18.