Copyright

by

Youngtaek Kim

2013

The Dissertation Committee for Youngtaek Kim Certifies that this is the approved version of the following dissertation:

Characterization and Management of Voltage Noise in Multi-Core, Multi-Threaded Processors

Committee:

Lizy K. John, Supervisor

Earl E. Swartzlander, Jr.

Adnan Aziz

Michael J. Schulte

Vijay Janapa Reddi

Characterization and Management of Voltage Noise in Multi-Core, Multi-Threaded Processors

by

Youngtaek Kim, B.S.E.E.; M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2013

Dedication

To my family and friends

Acknowledgements

First of all, I would like to express my profound gratitude to my academic advisor, Prof. Lizy K. John. She gave me priceless advice and help in completing my doctoral degree. In addition, I am deeply grateful to my dissertation committee members, Prof. Earl E. Swartzlander, Jr., Prof. Adnan Aziz, Dr. Michael J. Schulte, and Dr. Vijay Janapa Reddi, and Dr. Srilatha (Bobbie) Manne for their comments, guidance, and encouragement on my Ph.D. proposal and dissertation.

I am grateful to the current and the former AMDers, Jungseob Lee, Dr. Sanjay Pant, Dr. William Lloyd Bircher, Dr. Madhu S. Sibi Govindan, Tom Snodgrass, Indrani Paul, and Mgr. Andrew Kegel, who helped me to finish my internship at Advanced Micro Devices, Inc. with great results.

I am thankful to Laboratory for Computer Architecture (LCA) people, Jungho, Jee Ho, Faisal, Umar, Arun, Karthik, Jian, Ciji, Dimitris, Jeff, Ankita, and Don, for giving me many helps on campus life and attending and discussing my proposal and dissertation practice talks.

I would like to thank the current and the graduated UT Ph.D. students, Junyoung, Jaehong, Junsoo, Ikhwan, Dr. Hyunjin Kim, Dr. Yongchan Ban, Dr. Wooyoung Jang, Dr. Inwook Kong, for encouraging me and celebrating my achievements.

There are several friends I would like to acknowledge individually. First, I am enormously grateful to Myoungjun, one of my closest friends. His words encouraged me whenever I felt down. Secondly, I would like to thank Boongeo -Jinwoo, Ikjoong, and Hyunook, my good friends for more than twenty years.

Finally, I would like to thank my father, my mother, and my elder sister and her family for their devotion and support of my life and study. I am grateful to my extended family for their considering me. I would also like to thank my father-in-law, my mother-in-law, my grandmother-in-law, my sister-in-law and her family, and my brother-in-law for their hearted words and helps during my Ph.D. study. Last but most importantly, I am heartedly grateful to my wife, Kyoungpil, my lovely daughter, Hyemin, and my proud son, Minho, for they have cheered me on all through my work on my Ph.D. degree.

Characterization and Management of Voltage Noise in Multi-Core, Multi-Threaded Processors

Youngtaek Kim, Ph.D.

The University of Texas at Austin, 2013

Supervisor: Lizy K. John

Reliability is one of the important issues of recent microprocessor design. Processors must provide correct behavior as users expect, and must not fail at any time. However, unreliable operation can be caused by excessive supply voltage fluctuations due to an inductive part in a microprocessor power distribution network. This voltage fluctuation issue is referred to as inductive or di/dt noise, and requires thorough analysis and sophisticated design solutions. This dissertation proposes an automated stressmark generation framework to characterize di/dt noise effect, and suggests a practical solution for management of di/dt effects while achieving performance and energy goals.

First, the di/dt noise issue is analyzed from theory to a practical view. Inductance is a parasitic part in power distribution network for microprocessor, and its characteristics such as resonant frequencies are reviewed. Then, it is shown that supply voltage fluctuation from resonant behavior is much harmful than single event voltage fluctuations. Voltage fluctuations caused by standard benchmarks such as SPEC CPU2006, PARSEC, Linpack, etc. are studied.

Next, an AUtomated DI/dT stressmark generation framework, referred to as AUDIT, is proposed to identify maximum voltage droop in a microprocessor power distribution network. The di/dt stressmark generated from AUDIT framework is an

instruction sequence, which draws periodic high and low current pulses that maximize voltage fluctuations including voltage droops. AUDIT uses a Genetic Algorithm in scheduling and optimizing candidate instruction sequences to create a maximum voltage droop. In addition, AUDIT provides with both simulation and hardware measurement methods for finding maximum voltage droops in different design and verification stages of a processor. Failure points in hardware due to voltage droops are analyzed.

Finally, a hardware technique, floating-point (FP) issue throttling, is examined, which provides a reduction in worst case voltage droop. This dissertation shows the impact of floating point throttling on voltage droop, and translates this reduction in voltage droop to an increase in operating frequency because additional guardband is no longer required to guard against droops resulting from heavy floating point usage. This dissertation presents two techniques to dynamically determine when to tradeoff FP throughput for reduced voltage margin and increased frequency. These techniques can work in software level without any modification of existing hardware.

Table of Contents

Table of Contents ... ix

List of Tables ... xiii

List of Figures ... xv

Chapter 1: Introduction ... 1

1.1. Motivation ... 5

1.1.1. Automatic Stressmark Generation ... 5

1.1.2. Dynamic Management of Voltage Margins ... 6

1.2. Objectives ... 7

1.3. Thesis Statement ... 7

1.4. Contributions ... 8

1.5. Organization ... 9

Chapter 2: Background and Related Work ... 11

2.1. Power Distribution Network and Resonant Frequency ... 11

2.2. Current draw, Voltage Fluctuation, and Voltage Margin ... 15

2.3. Manual di/dt Stressmark ... 17

2.4. Charactersistics of Current Waveform and Voltage Fluctuation ... 19

2.5. Related Work ... 20

2.5.1. Characterizing Inductive Noise ... 20

2.5.2. Impact of Compiler Optimization ... 22

2.5.3. Managing Inductive Noise ... 23

Chapter 3: Methodology ... 27

3.1. Method for Characterizing Voltage Droop ... 27

3.1.1. Simulation Method ... 27

3.1.2. Hardware Measurement Method ... 29

3.1.2.1. System hardware ... 30

3.1.2.2. Measurement equipment ... 31

3.2. Benchmarks ... 34

3.3. Metrics 37

Chapter 4: Analysis of Voltage Noise in Microprocessor 39
4.1. Voltage Droop Analysis for Standard Benchmarks 39
4.1.1. Voltage Droop Analysis using Simulation 39
4.1.2. Voltage Droop Analysis using Hardware Measurement 42
4.2. Synchronization (Alignment) effect of Multiple Threads 46
4.3. Impact of Compiler Optimization on Voltage Droop 49
4.3.1. Compiler Impact on Performance, Power, and Reliability 50
4.3.2. Experimental Results 51
4.3.2.1. Experimental Setup 52
4.3.2.2. Result of miniFE 52
4.3.2.3. Result of High-Performance Linpack 55
4.3.2.4. Result of SPEC CPU2006 57
4.3.3. Summary of Compiler Optimization Impact 62
4.4. Summary 63

Chapter 5: AUDIT Framework to Generate di/dt Stressmarks 64
5.1. AUDIT Framework 65
5.1.1. Instruction Scheduling using the Genetic Algorithm 67
5.1.2. Framework Control using the Genetic Algorithm 68
5.1.3. Instruction Sequence Generation for the Genetic Algorithm .70
5.1.4. Dependency Control and Register Assignment 70
5.1.5. Stressmark Size and Resonant Frequency 71
5.1.6. Management of Search Space 72
5.1.6.1. Reduction of the number of instruction types 72
5.1.6.2. Reduction of code length scheduled using sub-blocking 73
5.1.7. Adoption of Dithering Algorithm for Guaranteed Alignment 74
5.1.8. Code Generation for Multiple Threads 77
5.1.9. Sweep for Finding a Resonant Frequency 77
5.2. Experimental Results using AUDIT Framework 79

5.2.1. Results of AUDIT Simulation Path 79
5.2.2. Results of AUDIT Hardware Measurement Path 86
5.2.2.1. Experimental setup..86
5.2.2.2. Voltage droop analysis..87
5.2.2.3. Voltage droop probability..90
5.2.2.4. Voltage droop vs. voltage at failure..........................91
5.2.2.5. AUDIT loop analysis 93
5.2.2.6. Impact of FP throttling...95
5.2.2.7. AUDIT on a different processor 97
5.3. Summary 97
Chapter 6: Dynamic Management of Supply Voltage Margin 99
6.1. Management of Voltage Guardband...100
6.2. Workload Induced Voltage Margins...102
6.2.1. Di/dt Stressmarks...102
6.2.2. Floating-Point Activity 103
6.2.3. Floating-Point (FP) Unit Throttling....................................105
6.3. Determining Voltage Margins 107
6.4. Dynamic FP Throttling 109
6.5. Experimental Results 111
6.5.1. Experimental Setup...111
6.5.2. Performance with FP Throttling 112
6.5.3. Energy Efficiency 115
6.5.4. Dynamic Scheme 117
6.5.5. Multi-core Execution 119
6.5.6. Improved Dynamic Scheme..121
6.6. Summary...124
Chapter 7: Conclusion and Future Research..126
7.1. Summary and Conclusion..126
7.2. Future Research 128
7.2.1. Extension to AUDIT 128

7.2.2. Improvement of Dynamic Scheme for Managing Voltage Margins 129
7.2.3. Charaterization and Management of Voltage Noise in GPU 129

Bibliography 131

List of Tables

Table 3.1: Comparison of sampling and triggering methods for hardware measurement. 32

Table 3.2: Standard benchmarks and stressmarks used in this dissertation........ 36

Table 4.1: An architecture configuration for SimpleScalar. 40

Table 4.2: Comparison of experimental methodology between Valluri and John [67] and this dissertation. 51

Table 4.3: Runtime, power, maximum voltage droop, and energy of *miniFE* with increase of the number of threads. 53

Table 4.4: Runtime, power, maximum voltage droop, and energy of High-Performance Linpack (8T) with different libraries and optimization levels. The values are normalized to the Original BLAS and O0 case. 56

Table 4.5: SPEC CPU2006 1T Results with –O3 (normalized to –O0) 58

Table 4.6: SPEC CPU2006 4T Results with –O3 (normalized to –O0) 60

Table 5.1: Base architecture configuration for SimpleScalar 81

Table 5.2: Five different PDNs for circuit simulation 81

Table 5.3: Maximum voltage droops of SPEC CPU2006, hand-coded [24], and automatic di/dt stressmarks. Supply voltage is 1V (Vnom=1V). 83

Table 5.4: Voltage at failure relative to *A-Res* 4T failure point......................... 93

Table 5.5: Impact of FP throttling on relative droop (relative to 4T *SM1*) and failure point (relative to 4T *A-Res*). 96

Table 5.6: Droop and failure results for a 45-nm AMD Phenom II processor. Droop and failure point are shown relative to SM2......................... 97

Table 6.1: Frequency boost possible with FP Throttling. 109

Table 6.2: Algorithm for dynamic voltage guardband manegement using FP-IPC..111

Table 6.3: Improved algorithm for dynamic voltage guardband management.122

List of Figures

Figure 1.1: Voltage margins to manage fluctuations. 5

Figure 2.1: Power distribution network (PDN)...12

Figure 2.2: 1st, 2nd, and 3rd resonance droops in the frequency domain. 13

Figure 2.3: 1st, 2nd, and 3rd resonance droops in the time domain. 14

Figure 2.4: Sudden current changes in a multi-core processor cause supply voltage fluctuation. When [aligned + resonant], voltage margin violations occur. 16

Figure 2.5: Current draws (top) and corresponding voltage fluctuations (bottom): 1st droop only excitation (left) and 1st droop resonance (right).......18

Figure 2.6: Scope shots of voltage fluctuation in production processor: 1st droop only excitation (left) and 1st droop resonance (right).......18

Figure 2.7: Different current pulse shapes (top) and corresponding voltage fluctuations (bottom). The current intensity is the same in HSPICE simulation...20

Figure 3.1: Current-voltage simulation. 28

Figure 3.2: AMD Bulldozer module. Adopted from Butler et al. [8]. 31

Figure 3.3: Hardware measurement set-up 33

Figure 3.4: Oscilloscope and differential probe used in this dissertation. 34

Figure 3.5: Thread configuration for multi-core system (T=thread). 37

Figure 4.1: Maximum voltage droop of SPEC CPU2006 in simulation. Vdd is 1.0V. 41

Figure 4.2: Occurrence (inclusive) at each voltage droop at Vdd=1.0V. 42

Figure 4.3: Hardware measurements of droop (relative to 4T *SM1*) for SPEC CPU2006...44
Figure 4.4: Alignment: misaligned-destructive (top) and aligned-constructive (bottom)...47
Figure 4.5: Scope shot of *natural dithering* due to OS interactions for resonant stressmark over a period of 100 ms. 49
Figure 4.6: Runtime, power, droop, and energy of *miniFE* (relative to 1T case)54
Figure 5.1: AUDIT framework for di/dt stressmark generation using simulators and hardware. 66
Figure 5.2: Simulation and hardware measurement paths to get max voltage droops..67
Figure 5.3: Conceptual instruction scheduling in the Genetic Algorithm. 68
Figure 5.4: Control of stressmark generation framework using Genetic Algorithm. 69
Figure 5.5: Instruction sequence generation for Genetic Algorithm. 70
Figure 5.6: Stressmark size and resonant period. 72
Figure 5.7: Periodic activity waveform for inducing power supply resonance and large voltage droops. 75
Figure 5.8: Code generation steps for multiple threads. 77
Figure 5.9: Frequency sweep to find resonance frequency. A loop length of 32 hits the resonant frequency and causes the largest voltage droop 79
Figure 5.10: Current waveform according to generation number.......................84
Figure 5.11: Current and voltage waveform of hand-coded and automated di/dt stressmark in Arch2-PDN2. 85

Figure 5.12: Hardware measurements of droop (relative to 4T *SM1*) for SPEC CPU2006, PARSEC, and stressmarks. 88

Figure 5.13: Frequency of droop events. 91

Figure 6.1: Workflow for dynamic voltage guardband management. 102

Figure 6.2: Relative voltage droop and power with varying issue rate of FP ops...105

Figure 6.3: Relative performance impact of FP throttling with and without frequency boost relative to *Base-3G*...114

Figure 6.4: IPC and performance for *St-FpThr-3.4G* relative to *Base-3G*. 115

Figure 6.5: Energy-delay product (E*D) with static and dynamic schemes. The values are relative to *Base-3G*. 116

Figure 6.6: Average FP-IPC for benchmarks. 118

Figure 6.7: Performance with 8T execution relative to *Base-3G* case. 120

Figure 6.8: Results with improved dynamic scheme (*Dyn2-FpThr-3.4G*). 123

Figure 6.9: Percent time spent with FP throttling enabled...............................124

Chapter 1: Introduction

For decades, microprocessors have been vastly used in our everyday lives; from electronic devices for personal use to workstations for business purposes and to supercomputers for scientific calculations. To support such various purposes and user experiences, microprocessor design needs to set different performance, power, and reliability goals according to the applications.

High performance is crucial for scientific calculations that need to be processed as fast as possible. Boosting the clock speed of a processor is one of the simplest ways of achieving the high performance goal. However, because of the increasing gap between processor and memory speeds, referred to as *memory wall*, such frequency boost is inefficient for increasing performance. To fill in the performance gap between processor and memory, designers have integrated a larger cache into the processor to compensate for a long latency of off-chip memory accesses.

Low power consumption is important for mobile hand-held devices to sustain a battery life as long as possible. Microprocessors also require low-power operations to reduce heat and resulting needs for expensive cooling method. As technology scales down, power density goes up and the increasing power density will cause more thermal hotspots on a microprocessor. A dynamic voltage and frequency scaling effectively reduces the total power consumption of a processor, and other techniques such as clock gating and power gating are useful to reduce dynamic power consumption.

Reliability is a fundamental requirement of processor design. Processors must work correctly across a range of applications regardless of process variations, voltage variations, environmental noise, and the aging of the system. However, guaranteeing

reliability is one of the most complicated tasks in microprocessor design; it is difficult to analyze and resolve a reliability problem that usually lies across multiple design stages in a complex architecture.

Recently microprocessor design entered this new era where power and reliability are prime design constraints. The traditional design goal, high performance, is hard to achieve by increasing CPU clock frequency only. Improving performance with frequency boost is not simple anymore because the frequency boost is limited by power and reliability constraints. Generally increasing frequency needs more power, and higher frequency makes a processor circuit more susceptible to voltage noise. As power increases, the processor's temperature quickly rises and gives more variations on reliability.

Using parallelism such as multiple cores and multiple threads can achieve high performance goal. However, multiple cores and multiple threads on the same processor not only require additional power consumption but also induce more switching noise, compared to a single core and a single thread. Multiple critical paths would exist and it is difficult to identify and analyze all of them.

Supply voltage noise issue caused by rapid current changes will be more critical to guarantee the reliability in future microprocessors. Cloud computing and big data require high performance and low power processors to serve many users simultaneously. The processor used in server systems will integrate multiple cores, even GPUs, into a single chip as many as possible to improve performance and power efficiency. Because of the power issues frequently referred to as the *power wall* problem, the maximum frequency and power will hardly go beyond 4GHz and 200W, respectively. Power supply voltages have been getting closer to the threshold voltages of underlying transistors, so

the magnitude of supply voltage fluctuations will not shrink much. However, due to the more complex architecture, analyzing critical paths will be more difficult. As technology scales, supply voltage noise will give more impact on cell delay [65]. In other words, process technology advance will increase the sensitivity of cell delay. In addition, the smaller feature size, the more susceptible to process and thermal variations microprocessor circuits are. Therefore, it is important to be aware of supply voltage noise issues because both the complexity of microarchitecture and the variability of microprocessor circuit will increase in future microprocessor design.

This dissertation focuses on reliability issues arising from supply voltage fluctuations. Supply voltage fluctuation, referred to as *inductive* or *di/dt noise*, is caused by sudden change of current draw in microprocessors and the power distribution network. Parasitic inductance on the die, package and the board often disturb the current flow from the voltage regulator on board to processor components on die. Such disturbance causes a temporary lack of electric charge that is needed for powering the processor components. Decoupling capacitance can be a solution for storing and providing electric charge to processor components when voltage emergency arises. However, capacitance can also induce voltage fluctuation because of the characteristics of RLC circuits.

The rate of current change is determined by program behavior. When an instruction sequence flows through a microprocessor architecture, internal microprocessor components will be turned on and then be turned off, and it changes current draw. It is difficult to predict the amount of current draw cycle by cycle because many instructions are on the fly across different pipeline stages and different paths. There are many sensitive paths on cores that can lead to catastrophic failures when the system is stressed by reduced noise margins, and it is imperative that one have the tools necessary

to identify these paths [45].

Increasing CPU clock frequency for high performance has been limited because of power constraints. One of the most effective ways to decrease power is to scale down the supply voltage. However, circuits become more susceptible to supply voltage noise due to near threshold voltage operations, and even a small amount of supply voltage fluctuation may cause reliability problems at the lower power supply voltages. Now designers need to analyze the supply voltage noise and devise solutions for guaranteeing reliable processor behavior with very low supply voltages. Low power goals and techniques need to be managed in tandem with the reliability goals of the processor.

Voltage margins (a.k.a. voltage guardband) are introduced to compensate for potential supply voltage fluctuations in the system. Fluctuations caused during program execution must stay within allowed margins, as shown in Figure 1.1. These margins need to be designed carefully to be power-efficient and prevent malfunctions from program-induced voltage fluctuations. The voltage margins guard against process variations, system power supply variation, and workload induced voltage droops. These margins are set conservatively, and are on the order of 15% to 20% of supply voltage [23]. However, as shown later in this dissertation, standard applications running under normal conditions do not exhibit voltage variations anywhere close to the worst case margins. By guarding against the worst case scenarios, a lot of performance is lost. For instance, according to Reddi et al. [55], a 20% voltage margin translates into a 33% frequency loss.

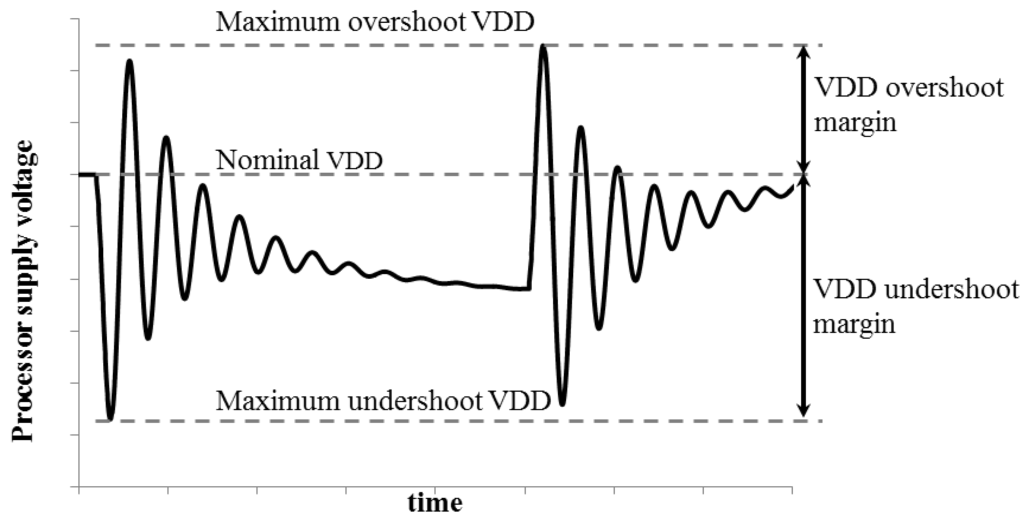


Figure 1.1: Voltage margins to manage fluctuations.

1.1. MOTIVATION

This section describes the motivation and background of this research.

1.1.1. Automatic Stressmark Generation

Specialized benchmarks, referred to as stressmarks, are used to study the susceptibility of processors to voltage fluctuations. Stressmarks may or may not be used to set the voltage margins; however, they are necessary to develop an understanding of the susceptibility of the system being analyzed. Stressmarks can be collected from existing benchmarks that have produced high di/dt stresses in the past. However, most existing benchmarks such as SPEC CPU2006 focus on high performance only, so they may not generate periodic, high and low current draw under normal condition [24]. Moreover, standard benchmarks require a long simulation time in an early design stage.

On the other hand, stressmarks can be specially designed to induce voltage

fluctuations in microprocessors. In many cases, designers manually generate a di/dt stressmark to test their processor/system. However, the manual generation of a di/dt stressmark is tedious and time-consuming. Designers need to recreate stressmarks whenever an architectural change occurs. In addition, the search space is extremely large, so it is not feasible for designers to manually generate and test every possible combination of parameters, configurations, and instruction scheduling to fully utilize a processor/system.

In this dissertation, an automatic di/dt stressmark generation framework is proposed to produce significant voltage droops. A Genetic Algorithm (GA) is utilized, and several techniques are developed to generate and optimize candidate di/dt stressmarks.

1.1.2. Dynamic Management of Voltage Margins

Stressmarks are benchmarks designed to stress a processor in various ways, such as generating the worst case power or the worst case voltage droops. Stressmarks designed to induce large di/dt voltage droops are used to determine the voltage guardband due to workload induced di/dt noise. Di/dt stressmarks consist of a region of high power instructions followed by a region of low power instructions [19][24][26][27]. Analysis presented in Chapter 5 of this dissertation has shown that on x86 processors the high power region typically contains a high number of floating point (FP) or Streaming SIMD Extensions (SSE) instructions, while the low power region generally contains NOPs. The high power region can consist of other types of instructions, such as instructions from the integer pipeline, but the resulting voltage droop from these

instructions is significantly less than the droop from instructions that execute on the FP path because operations that use FP pipeline dissipate relatively large amounts of power and thus lead to large di/dt fluctuations. Hence, the worst case guardband of the system is determined using operations that utilize the FP pipeline. If the workload does not have high FP pipeline utilization, then the system can be run with a lower voltage guardband, which can be translated into a higher operating frequency.

In this dissertation, two algorithms are presented to dynamically control FP throttling and adjust the operating frequency in order to trade off frequency for FP throughput to improve the performance of both FP-intensive and non-FP-intensive programs.

1.2. OBJECTIVES

The objective of this dissertation is to characterize di/dt noise and to develop a method to manage voltage margins. The specific objectives are as follows:

- Characterize di/dt noise in different microprocessors with various benchmarks
- Develop a method for generating effective di/dt stressmarks automatically in both simulation and real hardware environments
- Examine software optimization impact on di/dt noise
- Develop a technique for managing trade-offs between performance and di/dt noise for multi-core processors

1.3. THESIS STATEMENT

Automated stressmark generation framework to generate stressmarks to

characterize supply voltage noise in multi-core, multi-threaded processors can be constructed using genetic algorithms and a voltage fluctuation measurement/simulation framework. A dynamic voltage margin management scheme using functional unit throttling increases system performance while suppressing supply voltage noise.

1.4. CONTRIBUTIONS

This dissertation makes the following contributions:

- The existing di/dt stressmarks and their behavior in single and multi-core systems are discussed and analyzed.
- An automated stressmark generation framework is proposed, which
    - generates an effective di/dt stressmark without comprehensive knowledge of a microprocessor system,
    - utilizes a Genetic Algorithm to generate a benchmark that creates a maximum voltage droop in a given microprocessor and PDN,
    - reduces designers' time to generate a hand-coded di/dt stressmark and/or to simulate typical benchmarks that are possibly irrelevant to inducing maximum voltage droop,
    - utilizes real multi-core hardware to generate di/dt stressmarks quickly and automatically,
    - applies a novel method referred to as dithering to align stressmarks in multicore systems,
    - compares the maximum voltage droop and catastrophic behavior of standard benchmarks, manually generated stressmarks, and automated stressmarks, and

    - analyzes di/dt noise of a state-of-the-art x86 multi-core processor with multi-threading and architectural throttling effects.
- An FP throttling mechanism on a state-of-the-art x86 processor is analyzed, resulting in
    - a study of frequency boost, performance, and energy-delay product benefits made possible by FP throttling,
    - a study of the impact of FP throttling with multi-core execution, and
    - new algorithms to dynamically manage FP throttling and their analysis.

1.5. ORGANIZATION

Chapter 2 reviews the background of di/dt noise analysis and related work on the di/dt issue. Current draw, voltage fluctuations, and voltage margins are described, and power distribution networks and resonant frequencies are discussed as they affect nature of di/dt noise. Related work is categorized into three folds; characterization of di/dt noise, compiler impact on di/dt noise, and management of di/dt noise.

Chapter 3 presents the methodology of this dissertation. Simulation and hardware measurement environments are introduced. Benchmarks used in this work are categorized in several ways. Metrics are defined to characterize performance, power, and voltage noise.

Chapter 4 investigates voltage noise problems in microprocessors in more detail; the relationship of current waveform and voltage fluctuation and synchronization effect in multi-core, multi-threaded processors. Voltage noise under different compiler optimization levels is analyzed. Two different optimization levels are applied to various

benchmarks, and the variations in performance, power, voltage droop, and energy are compared.

Chapter 5 describes the AUDIT framework that generates a di/dt stressmark to characterize voltage noise. The application of a Genetic Algorithm (GA) is also explained in detail. Simulation and hardware measurement paths for pre- and post-silicon processor models are introduced.

Chapter 6 suggests a method to dynamically manage voltage margins. The effect of the existing throttling of floating-point units is measured and the amount of the voltage margin reduction is analyzed. An algorithm is introduced to find an optimal tradeoff between performance and voltage margin reduction. Chapter 7 concludes the dissertation and suggests possible future research directions.

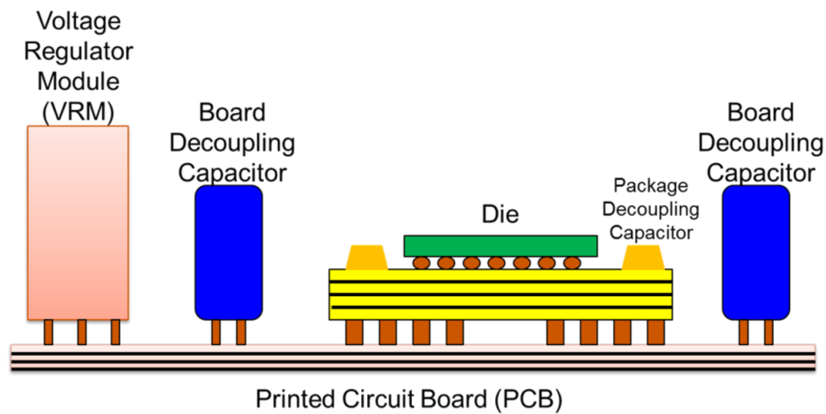
Chapter 2: Background and Related Work

In a microprocessor, the supply voltage is provided through a power distribution network (PDN), which can be represented as a distributed RLC circuit with resonance frequencies. Varying current (*di/dt*) can cause fluctuations of the supply voltage that are proportional to the inductance (L) of the circuit ($v = L \cdot di/dt$). Voltage droop is maximized if the periodic, large current variation occurs at the resonance frequency of the PDN. A resonance frequency in the mid-frequency (50 to 200MHz) range is the most significant [44]. Significant supply voltage droop may cause reliability problems in a microprocessor. Reddi [50] discusses the voltage fluctuation problem, and presents experiments illustrating the gravity of the situation. Low voltage increases the delay of signals, which could affect the timing between two flip-flops in a microprocessor circuit. Also, insufficient voltage could fail to set bit-signals properly and lead to soft errors. In this chapter, background on inductive (di/dt) noise is presented.

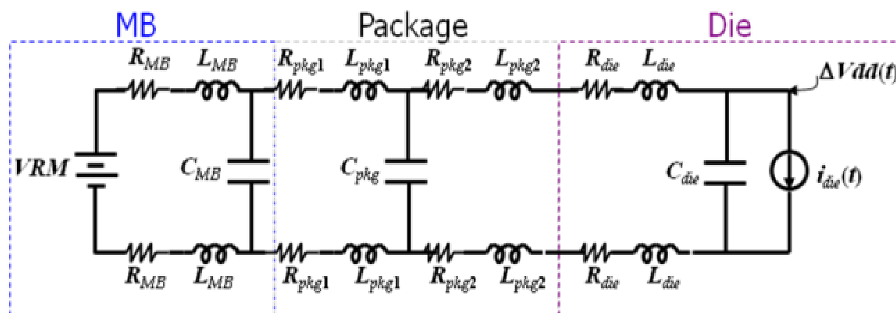
2.1. Power Distribution Network and Resonant Frequency

The power distribution network (PDN) of a typical microprocessor consists of inductive and resistance elements on the motherboard (*MB*) or Printed Circuit Board (*PCB*), package, and die (see Figure 2.1). The parasitic resistance of the network causes a droop (IR drop) in the power supply proportional to the current drawn from the network. In addition, the inductance in the network causes undershoots and overshoots in the power supply (referred to as the di/dt drop), which depend on the rate of change of the load-current. To mitigate the inductive noise in the power supply, decoupling capacitance, commonly referred to as decap, is added at different locations in the power

supply network as illustrated in Figure 2.1. The amount of added decap progressively increases away from the die to counter the effect of increasing inductance parasitics. The series combination of parasitic inductance (L) and decap (C) results in various resonance frequencies ($1/2\pi\sqrt{LC}$) in the network, as shown in Figure 2.2 and 2.3 in the frequency and time domains.



(a) Physical structure of power distribution network. Adopted from Popovich [22].



(b) Simplified RLC circuit model of power distribution network.

Figure 2.1: Power distribution network (PDN)

The prominent resonance frequencies shown in Figure 2.2 and 2.3 are the 1st droop resonance due to the interaction of package and on-die inductance (L_{pkg2} + L_{die}) with on-die decap (C_{die}), the 2nd droop resonance due to the interaction of socket and package inductance (L_{pkg1}) with package decap (C_{pkg}), and the 3rd droop resonance due to the interaction of board inductance (L_{MB}) with decap on the board (C_{MB}). A periodically varying load can induce one or more of these resonances and cause excessive undershoots and overshoots. Although 2nd and 3rd droop resonance can also impact the reliability of the system, they can be mitigated by techniques such as load line based voltage regulator modules [56] and are beyond the scope of this work. Although AUDIT in Chapter 5 is discussed in the context of first droops, it can be tuned to excite any of the three types of droops.

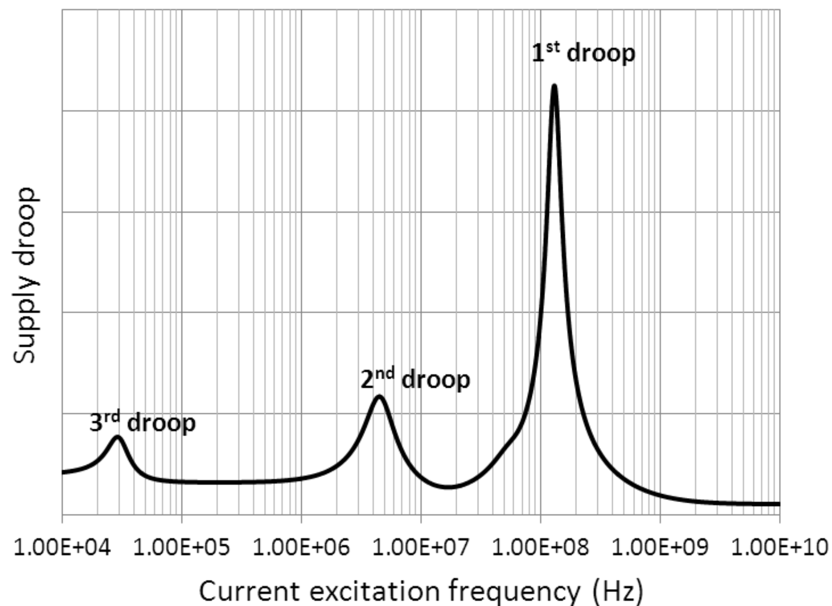


Figure 2.2: 1st, 2nd, and 3rd resonance droops in the frequency domain.

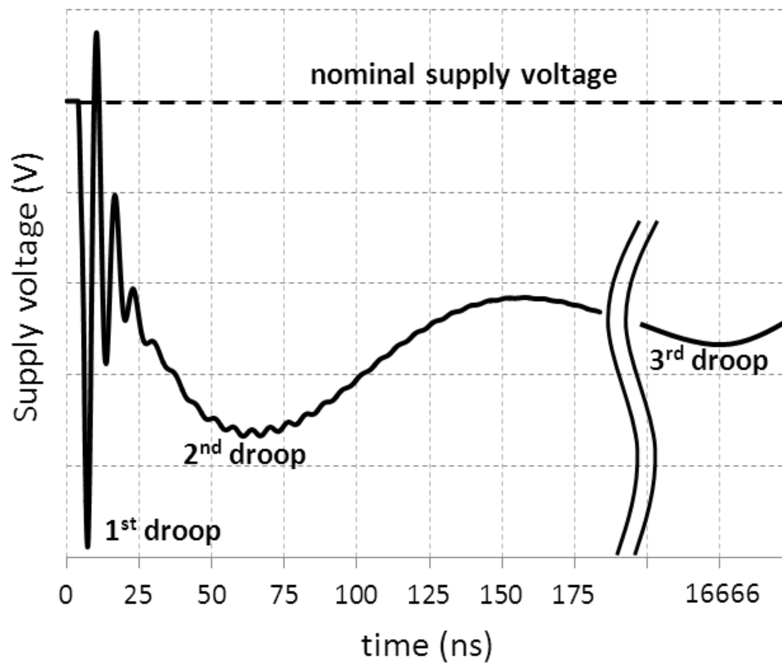


Figure 2.3: 1st, 2nd, and 3rd resonance droops in the time domain.

The 1st droop resonance is a strong function of package inductance (L_{pkg2}) and on-die decap (C_{die}), and is typically in the range of 50MHz – 200MHz. Examples of events causing large first droop are power wakeup of one or more blocks present in the design or a sudden up or down surge in the processor activity. When such rapid events occur periodically at the 1st droop resonance frequency, they may cause 1st droop resonance resulting in large, sustained undershoots and overshoots in the power supply. 1st droop can be mitigated by explicitly adding decoupling capacitance on the die [44]. However, there are limits to the feasibility of this approach due to area constraints and the leakage of the decap. Several architectural techniques that limit the rate of change of activity in

the processor are effective in suppressing the first droop [17][19][24][47], but they may have a negative impact on performance.

2.2. Current draw, Voltage Fluctuation, and Voltage Margin

Figure 2.4 shows possible current changes and the corresponding voltage fluctuations in a multi-core processor. Each core runs a program and its current changes by time are in the top of Figure 2.4. When rapid current change occurs, processor supply voltage fluctuates and then it is quickly damped (bottom of Figure 2.4). Interestingly the magnitude of fluctuation is affected by two factors: the sum of two cores' current (current intensity) and periodic behavior of current draw. When the changes from low to high current loads are aligned between two cores that share the same supply voltage, the total current is doubled and it causes larger voltage fluctuation. When the current changes are periodic and meet the resonant frequency of the PDN, the following voltage fluctuations are additive to the previous ones and then the magnitude of fluctuation increases.

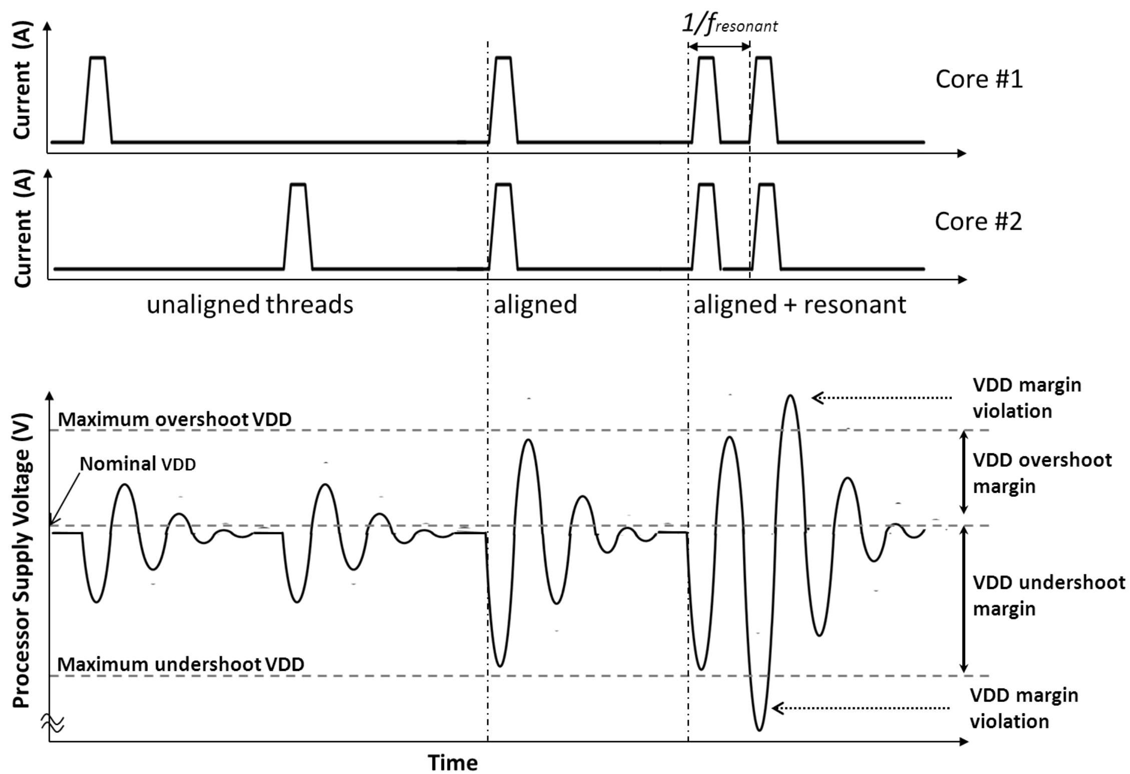


Figure 2.4: Sudden current changes in a multi-core processor cause supply voltage fluctuation. When [aligned + resonant], voltage margin violations occur.

Figure 2.4 also shows voltage margins from nominal voltage (VDD). When the voltage fluctuations go down under the nominal voltage, it is called an undershoot (droop). When voltage fluctuations go over the nominal voltage, it is called an overshoot. Voltage margins are set to guarantee correct behavior even with the worst case fluctuation. If the voltage fluctuation goes beyond the voltage margin, it causes a voltage margin violation and operations become unreliable. Overshoot is more harmful than undershoot because overshoot may cause permanent damage to the internal circuits of a microprocessor.

2.3. MANUAL DI/DT STRESSMARK

To analyze the impact of supply voltage droop on performance and reliability, it is imperative to create instruction patterns (stressmarks) that stress the PDN of the processor to cause large undershoots and overshoots. Traditionally, stressmarks have been generated manually with the knowledge of power consumption of different patterns of instructions and details of the PDN. This subsection describes a methodology for manually generating stressmarks.

The first step for manually designing stressmarks requires analysis of the power consumptions of different instruction patterns. To induce large di/dt droops, the processor cores should switch simultaneously from a low-power state to a high-power state as quickly as possible. For the low-power state, instructions such as a NOP that consume a low amount of power can be chosen. For the high-power state, floating point or SIMD instructions that consume lots of power and that can attain the highest IPC supported by the target machine are selected. If a single high-di/dt event occurs where the machine executes a pattern of low-power instructions followed by a pattern of high-power instructions, there will be a droop in supply voltage, but the droop will taper off quickly as shown on the left side of Figure 2.5. However, a pattern that repeats periodically at the resonant frequency of the PDN will build in amplitude and not only generate a larger droop than a single event but one that repeats regularly, thereby increasing the probability of system failure (right side of Figure 2.5). Both these conditions help to build an effective first droop stressmark. This dissertation covers 1st droop only excitation and 1st droop resonance in the analysis. Figure 2.6 is the screen shot of an oscilloscope that shows a production processor's voltage levels.

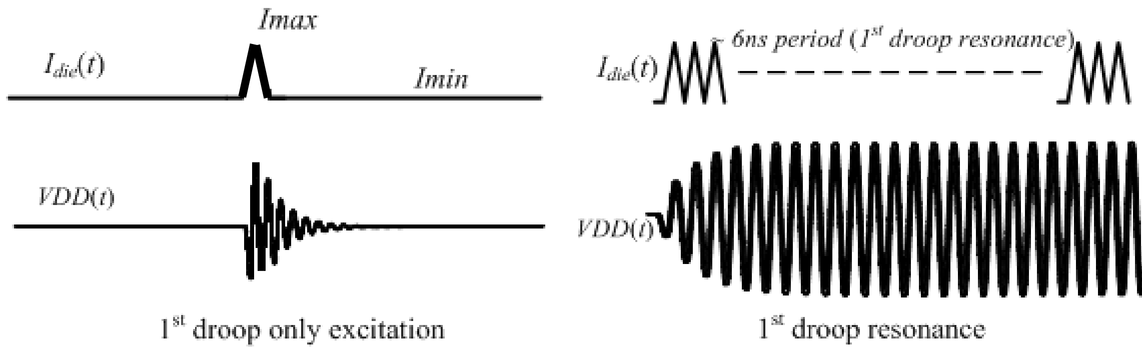


Figure 2.5: Current draws (top) and corresponding voltage fluctuations (bottom): 1st droop only excitation (left) and 1st droop resonance (right).

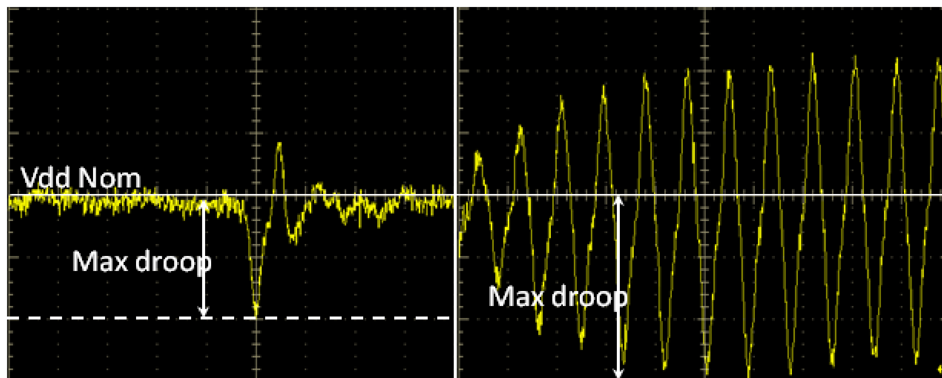


Figure 2.6: Scope shots of voltage fluctuation in production processor: 1st droop only excitation (left) and 1st droop resonance (right).

Several challenges exist in manually designing the stressmarks. First, resonant frequencies can change between different boards in the same product line and between different products (such as client versus server). Therefore, multiple stressmarks need to be developed to target different resonance frequencies and different system configurations. Second, because the periodicity of the low-power and high-power

instructions depends on the operating frequency, multiple stressmarks need to be written for the various operating frequencies of the system . Finally, it is very time consuming to explore different combinations of instructions manually for the high-power and low-power portions of the loop. This dissertation addresses all these issues by automating the process of generating di/dt stressmarks.

2.4. CHARACTERSISTICS OF CURRENT WAVEFORM AND VOLTAGE FLUCTUATION

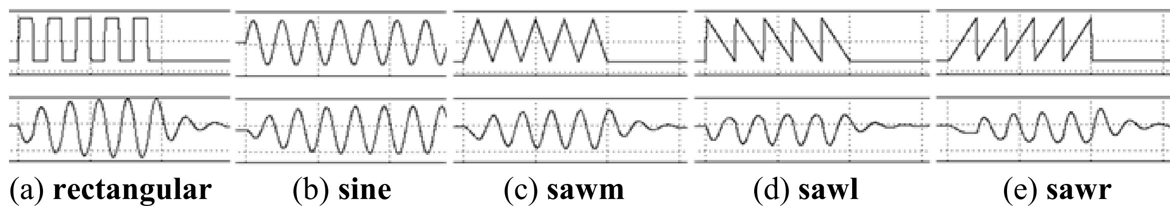
Voltage fluctuations by different shapes of current waveform are described in here. To induce a maximum voltage droop in a given environment, it is important to know the characteristics of di/dt voltage noise affected by current variations. Previous work [16][24] provide good analysis that this dissertation extends to add several factors to be considered when generating a di/dt stressmark.

Shape of current waveform: Maximum voltage droop is affected by the shape of the current waveform. Three types of current waveforms are generated and simulated: saw-tooth, sine, and rectangular. The corresponding voltage fluctuations show that rectangular-shaped current waveform is most effective to induce high voltage droop (Figure 2.7). It is difficult to make sudden current changes, but it is shown that the events like pipeline flush are able to generate huge interrupts of current draw in a program [52].

Ratio of high-to-low period: The width of the high-current pulse in the resonant period is adjusted and tested. Both wider and narrower widths than a half of resonant period alleviate the voltage droop. According to this observation, high and low current draw periods should be evenly distributed to induce a large voltage droop.

Difference of current intensity: Current intensity is very important to make a large voltage droop. However, even though a high current draw occurs, if the previous or the next current draw is also high enough, voltage hardly fluctuates.

The three aforementioned factors are critical to generate a di/dt stressmark for a defined microarchitecture and PDN. This dissertation uses these factors to analyze maximum voltage droop induced in different combinations of microarchitectures and PDNs.



(a) **rectangular** (b) **sine** (c) **sawm** (d) **sawl** (e) **sawr**

Figure 2.7: Different current pulse shapes (top) and corresponding voltage fluctuations (bottom). The current intensity is the same in HSPICE simulation.

2.5. Related Work

2.5.1. Characterizing Inductive Noise

There has been some previous work on hardware analysis of production systems. In [56], Reddi et al. measured and analyzed droops on a two-core Intel system and discussed constructive and destructive interference between processors and the difference

in droops between average and worst-case scenarios. This information was used to design a noise-aware thread scheduler to mitigate some of the di/dt stresses in the system. To date, the work by Reddi is the most detailed analysis of droops on hardware.

This dissertation expands on that work by analyzing a more complex system with multi-threading and up to eight logical processors. In addition, it is shown that constructive interference occurs more often than expected due to OS effects, and this knowledge is used to design effective stressmarks.

More recently, Miller et al. examined voltage emergencies in multi-core processors [34] with increasing numbers of cores, and showed how global synchronization points create large stresses in the system. This work used power variability as a proxy for di/dt stresses and examined the hardware at a coarse granularity of 1-ms intervals. This dissertation uses true voltage droop measurements and fine-grained sampling to detect first-order droops and discuss droop values as well as voltage failure points in hardware.

One of the major contributions of this dissertation is automatic stressmark generation using real hardware. Joseph, Brooks, and Martonosi presented a hand-coded di/dt stressmark [24]. Their basic idea was to create a sequence in which a high-current instruction follows a low-current instruction. The high-current component typically consisted of a memory load/store instruction and the low-current component consisted of a divide instruction followed by a dependent instruction, resulting in a long pipeline stall. However, their di/dt stressmark was manually crafted for a specific microarchitecture based on the knowledge of the current draw of various instructions. Furthermore, they focused only on memory-intensive behavior such as loads and stores and increased current draw by accessing L1 and L2 data caches. In contrast, the approach in this

dissertation does not require microarchitectural knowledge and relies on measured voltage droops in a closed-loop measurement infrastructure.

Ketkar and Chiprout proposed a di/dt stressmark generation methodology using integer linear programming (ILP) [27]. They extracted current draw for certain instructions from a register transfer language (RTL) model for the hardware. Linear programming with constraints was used to maximize voltage droop. However, they focused only on the ALU. It is difficult to make ILP relationships of instructions for all the pipeline stages and the caches; hence, it is difficult to apply their technique to an entire processor, especially one with out-of-order processing, multiple cores, and complex shared resource structures.

Joshi et al. [26] presented a methodology for generating maximum-power viruses and mentioned in passing that high-power and low-power instruction sequences from two different power optimizations can be interleaved to generate a di/dt stressmark. This was only a suggestion, without implementation details or results. Also, they did not talk about the importance of repeating the sequence at the PDN's resonant frequency. Neither di/dt effects nor voltage droops were the focus of Joshi's work.

A significant number of other studies have focused on preventing, reducing, or recovering from di/dt effects or voltage droops [3][5][6][7][8][9][10][15][17][18][21][22][23]. However, none of these focus on automatically generating di/dt stressmarks.

2.5.2. Impact of Compiler Optimization

Valluri and John [67] studied compiler optimization effects on performance and power. The conclusions are that (1) performance improvement by reducing the number of instructions brings energy reduction and that (2) performance improvement by increasing

the overlap in program increases average power dissipation. However, in Valluri and John's work, power is represented as the average power of overall execution. It is problematic because a voltage emergency occurs in a much shorter period than the program execution time, that is, from tens of nano-seconds to micro seconds compared to several minutes.

Reddi et al. proposed a dynamic scheduling workflow based on a checkpoint-and-recovery mechanism to suppress voltage emergencies [51]. Once a code part causes a voltage margin violation, it is registered as a hotspot, and NOP injection and/or code rescheduling is performed by the dynamic compiler. This flow is independent of the architecture or workload. However, users should be careful to set an initial voltage margin properly to avoid frequent voltage emergencies.

2.5.3. Managing Inductive Noise

A number of previous papers have explored how to reduce the voltage guardband of the system in order to achieve better performance. The work that comes closest in terms of a hardware implementation is the work by Lefurgy et al. [33] which addressed actively monitoring and managing the voltage guardband based on the use of a critical path monitor or CPM. The CPM monitors the critical pathways in the chip and increases the voltage guardband if the CPM detects potential errors. Although the CPM is a very effective mechanism, it requires additional hardware, monitoring mechanisms, and tuning of the CPM to detect and correct possible errors. The technique in this dissertation on the other hand, is very simple to implement and manage, and only requires a characterization effort to determine the frequency boost possible with FP throttling.

A number of papers have dealt with mitigating voltage droops using software techniques [17][19][57]. These techniques recognize the existence of repetitive code with high di/dt transition activity and dampen or eliminate this activity through software techniques. Software mitigation of noise does not guarantee that all errors will be eliminated. In fact, the software techniques learn from errors detected by hardware (such as a CPM) and adjust the software only after errors are detected. The FP throttling mechanism avoids errors altogether by suppressing the structures which generate the largest voltage droops.

Other work has examined using hardware techniques to manage high droops [40][47][48][49]. Some of the work focuses on using hardware to detect that a resonant droop is about to build and suppresses the droop before it reaches its peak droop value, while others focus on mechanisms to dampen the difference between the high and low power regions by techniques such as throttling issue rates or staged activation and deactivation of clock gated units. All these techniques address the issue of di/dt noise. However, this dissertation is the first to characterize the impact of FP throttling, translate that characterization to a frequency increase, and present results with static and dynamic schemes showing the benefits of the performance increase with FP throttling.

Another body of work explores detecting and mitigating errors via circuit techniques [10][11]. The research using Razor systems assumes that errors will occur and inserts redundancy within latches. Although effective, Razor requires significant new hardware and a completely different design methodology that fundamentally changes the way processors are designed. The FP throttling technique on the other hand, works well with existing systems where the floating point unit is a large contributor to the voltage droop in the system.

There are a wide range of architectural techniques that utilize some type of detection and recovery mechanism to deal with errors [2][16][37] and use redundant structures or replay mechanisms to detect and correct errors. All these techniques incur additional complexity or hardware overhead which FP throttling with frequency boost avoids.

Finally, there are other methods in which the processor frequency can be boosted [38][58]. Frequency boosting techniques such as Turbo Core or Turbo Boost ™ are in use in state-of-the-art systems from both AMD and Intel. Turbo Core allows the chip to run at a higher frequency than that visible to software. The highest frequency software visible ACPI P-state (P0) is determined under the assumption that all cores on the system are running a high power benchmark under worst case operating conditions. When those conditions are not met, either because not all cores are active or the threads are not high-power threads, the hardware allows the cores to enter a boosted frequency state based on the availability of power headroom. Once the power headroom is depleted, the application returns to a lower power, lower performance DVFS state until power headroom is once again available.

Turbo Core is available on the hardware this dissertation used for the experiments. There is one major difference between the frequency boost possible with Turbo Core and that resulting from FP throttling. With FP throttling, the processor can boost the frequency without an increase in voltage, resulting in a linear increase in power for a potentially linear increase in performance. Turbo Core, on the other hand, requires an increase in both frequency and voltage, resulting in a cubic increase in power. Hence, it is not as efficient as a method for boosting performance. However, it also does not incur any IPC loss due to FP throttling. Turbo Core was disabled for the

analysis presented in this paper in order to study the impact of FP throttling; however, combining the two techniques offers interesting research opportunities in the future.

Chapter 3: Methodology

This chapter describes the overall experimental methodology used in this dissertation. Either a simulation or a hardware measurement method can be used according to the availability of a post-silicon processor. Especially the hardware measurement method in this dissertation enables designers to capture maximum voltage droops on a post-silicon processor. But a simulator has advantages of reconfigurability and the technique in this dissertation can work with simulation models as well, To characterize di/dt behavior, various benchmarks and stressmarks are run from standard benchmarks, such as SPEC CPU2006, to automatically generated stressmarks. Runtime, power, and maximum voltage droop are useful metrics to characterize di/dt noise.

3.1. METHOD FOR CHARACTERIZING VOLTAGE DROOP

To characterize voltage droop on a microprocessor, designers can take either a simulation-based estimation or a hardware-based measurement method according to the current design stage or to the availability of the processor model.

3.1.1. Simulation Method

Simulation enables designers to estimate voltage fluctuations in early design stage even without silicon and to examine microprocessor events that cause voltage fluctuations in more detail, compared to hardware measurement. Simulation is also repeatable; it should give the same results if all the simulation inputs and parameters are the same as that of a previous simulation. However, the accuracy of simulation is the main issue, so the processor model should be well correlated to its current or expected

implementation on silicon.

Figure 3.1 shows the current-voltage simulation method. First, a program code is provided with C or assembly format. Next, the program code is compiled and run on a system simulator to estimate current draw per cycle in a microprocessor. During the system simulation, all the activities are counted every cycle and converted as power consumption per cycle. To get instantaneous current values, the obtained cycle power numbers are divided by a DC supply voltage. Then, the current trace from the system simulator is fed to the circuit simulator to simulate voltage fluctuation. After collecting the voltage trace, it is analyzed to identify a maximum voltage droop.

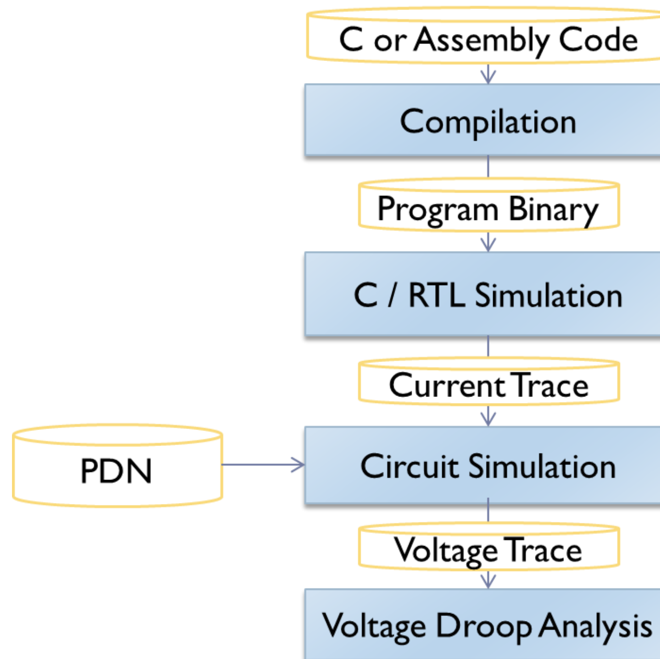


Figure 3.1: Current-voltage simulation.

In this dissertation, for the power (current) simulator, the combination of SimpleScalar [60] and Wattch [7] is selected to estimate current load variations per cycle in a microprocessor. The original simulator is modified to generate a current trace per cycle by dividing the power per cycle by the supply voltage. The modified power simulator is based on the activity counter of each unit in a microprocessor, so it is good at showing how much each unit in the microprocessor is utilized by benchmarks/stressmarks during each cycle. The modification for per-cycle power estimation can be applied to another system-level power simulator if such a simulator provides activity monitors for internal units. For circuit simulation, HSPICE [21] is used to simulate the current trace and to measure voltage droop. The current value per cycle from the system-power simulator is converted to HSPICE format as a current source. During the HSPICE simulation, maximum, minimum, and peak-to-peak values of voltage are measured. Instead of HSPICE simulation, the convolution of the processor's current trace and the PDN's impulse response can be used if the result shows enough accuracy compared to HSPICE circuit simulation.

3.1.2. Hardware Measurement Method

Hardware measurement can show real voltage fluctuations on silicon. Most benchmarks/stressmarks finish quickly, so the entire run of each benchmark is possible. However, it is not repeatable; it is difficult to get the same voltage fluctuation at the same time with the previous runs due to many sources of uncertainty in a real system such as OS scheduling for multiple threads. Hardware measurement methods require system hardware such as processors, packages/sockets, and motherboards (MB), and

measurement equipment such as an oscilloscope, differential probe and cable, etc.

3.1.2.1. System hardware

The multi-core processor mainly used in this dissertation is an AMD Orochi processor, which consists of four Bulldozer modules on the single processor chip. Each Bulldozer module (Figure 3.2) can execute two threads via a combination of shared and dedicated resources [8]. The front-end and floating-point logic is shared between two threads on the same module; however, the rest of the core components (integer and retire logic, load/store unit, first-level TLB, and first-level cache) are separate. Each thread can issue four integer instructions per cycle, however, the two threads together can only issue four floating point instructions per cycle due to the sharing of the floating point units. A thread can have a maximum IPC of four. One Bulldozer module has one 64 KB I-Cache, two 16 KB D-Caches for two hardware threads, and 2MB of L2 cache. Four Bulldozer modules share an 8MB L3 cache. A more detailed description of the Bulldozer module and architectural features is given in [8][12][68]. An AMD Phenom™ II X4 Model 925 processor is also used for hardware measurement. The package/socket used is an AM3 and the motherboard is specially designed to provide many test points including supply voltage rails.

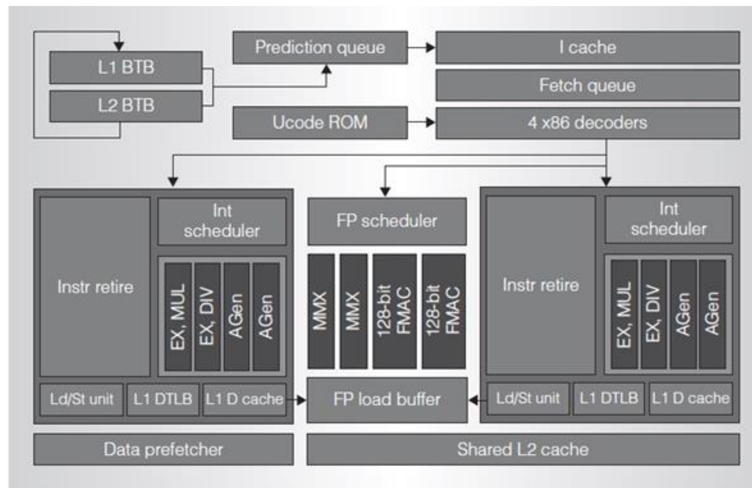


Figure 3.2: AMD Bulldozer module. Adopted from Butler et al. [8].

3.1.2.2. Measurement equipment

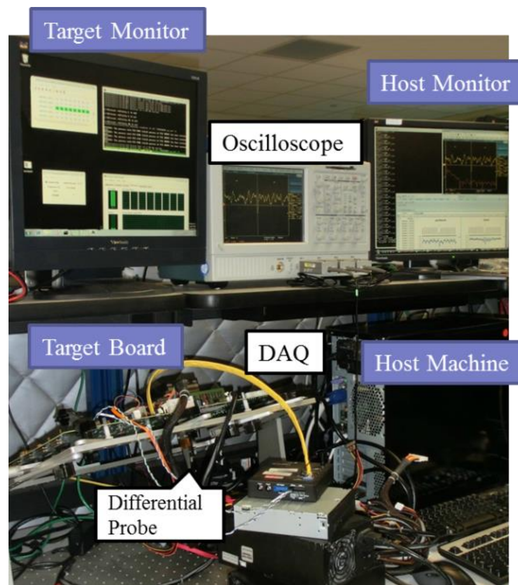
Voltage variations can be measured by probing power supply rails on the hardware system. There are two possible methods to measure voltage variations on hardware: sampling and triggering. The pros and cons of each measurement method are summarized in Table 3.1. This dissertation used sampling for power measurement and triggering for voltage droop measurement, because the sampling rate is not high enough to capture 1st droop and because triggering is good at capturing extreme values such as maximum voltage droop.

	Sampling with DAQ	**Triggering with Oscilloscope**
Pros	- Users can easily handle the measurement	- Accurate (error < 5mV) - FFT analysis is possible
Cons	- Typically sampling rate is too low to capture di/dt events - DAQ cable is too far from on-die VDD rails → false noise	- High-bandwidth oscilloscope is required - Probing test pins is difficult
Application	Good for power measurement	Good for di/dt measurement

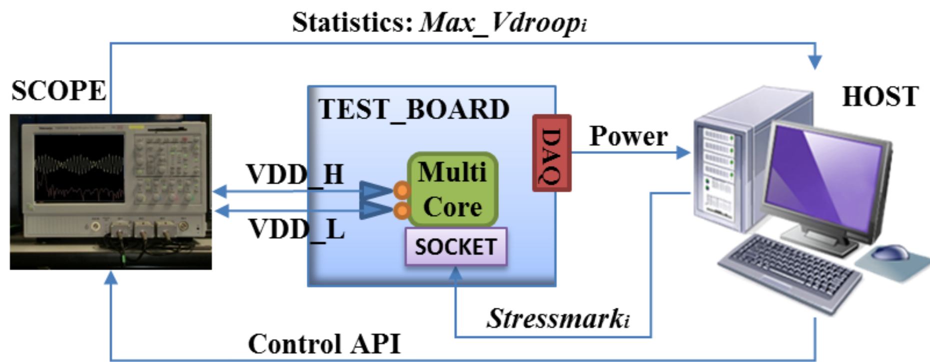
Table 3.1: Comparison of sampling and triggering methods for hardware measurement.

The experimental set-up for hardware in this dissertation is shown in Figure 3.3. Voltage droops on the hardware are measured with a Tektronix TDS5104B oscilloscope (shown in Figure 3.4.(a)) and a 1.7-GHz Tektronix P6248 differential probe (shown in Figure 3.4.(b)) for triggering on large voltage droops. The probing points for the power supply voltage are attached to the package and on-die connections to enable accurate voltage droop measurements. The oscilloscope triggers and records the di/dt events at a sampling rate of 5 gigasamples/second (GS/s).

Power is profiled using a National Instrument's Data Acquisition (DAQ) card (NI-PCIe 6353), whose sampling rate is up to 1.2 megasamples/second (MS/s) (Figure 3.3). A differential cable transfers multiple signals from the power supply lines on the motherboard to the DAQ card in the PC.

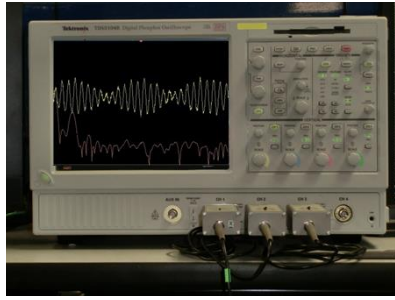


(a) Lab environment

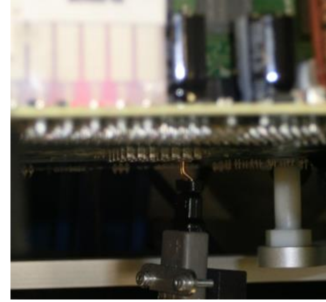


(b) Hardware measurement

Figure 3.3: Hardware measurement set-up



(a) Oscilloscope: 1GHz bandwidth Tektronix TDS5104B shows voltage response and frequency analysis



(b) Differential Probe: 1.7-GHz Tektronix P6248 probes processor's power rail

Figure 3.4: Oscilloscope and differential probe used in this dissertation.

3.2. BENCHMARKS

To analyze a processor's normal behavior, standard benchmark suites such as SPEC CPU2006 [62] and PARSEC [5] are mainly used because they represent various programs frequently running on the processor. However, standard benchmarks may not fully exercise a given architecture. For example, a load/store-intensive standard benchmark cannot frequently activate execution units such as the ALU. For some of the studies, supercomputing benchmarks such as *miniFE* [35] and *High-Performance Linpack* [20] are used.

In order to study the susceptibility of processors to voltage fluctuations, designers often resort to specialized benchmarks called di/dt stressmarks. The di/dt stressmarks are either collected from benchmarks that have produced high di/dt stresses in the past or manually designed to induce voltage fluctuations in microprocessors. Stressmarks based on excerpts from real programs may not expose many vulnerabilities of a system; hence often manual generation of stressmarks is done by engineers who are familiar with the

intricacies of the design. However the manual generation of stressmarks is tedious; several works addressed the complexities involved with stressmark generation and developed tools and methodologies to generate stressmarks automatically [1][2][3][4].

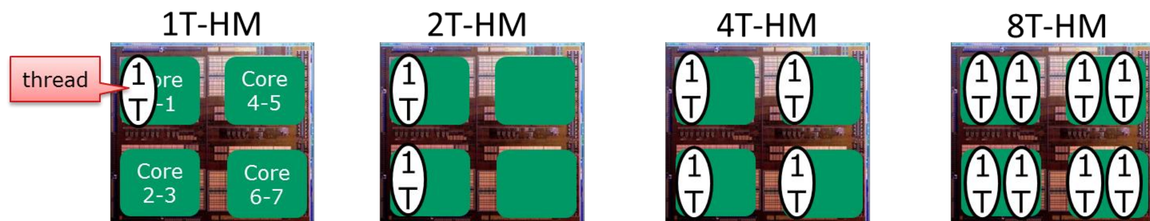
In this dissertation, both standard benchmarks and stressmarks are run to represent the usual and extreme user cases. SPEC CPU2006 and PARSEC are selected as standard benchmarks, the existing power and di/dt stressmarks are collected, and synthetic di/dt stressmarks are generated manually or automatically. Table 3.2 lists the benchmarks used in this dissertation.

Name	Benchmark Type	Multi-Threading	Etc.
SPEC CPU2006	Standard Benchmark Suite	Multi-programmed (NO *dithering*)	CINT(integer): 12 benchmarks CFP(floating-point): 19 benchmarks
PARSEC	Standard Benchmark Suite	Multi-threaded	12 benchmarks
SM1, SM2, SM-Res	Manual di/dt Stressmark	Multi-threaded / Multi-programmed (*dithering*)	Industry-level [43]
miniFE	Supercomputing (HPC) Benchmark	Multi-threaded	Highly scalable
HP Linpack	Supercomputing (HPC) Benchmark	Multi-threaded	High-performance,
A-Ext, A-Res	Automated di/dt Stressmark	Multi-programmed (*dithering*)	AUDIT in this dissertation

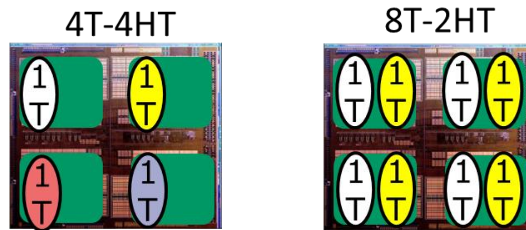
Table 3.2: Standard benchmarks and stressmarks used in this dissertation.

Thread configuration for multi-threading in a multi-core system is set as in Figure 3.5. With up to 4T (4 threads) only one thread runs on each core. When more than 4T one or more cores will run multiple threads. The number of threads will be evenly

distributed. Each program of multi-programmed runs needs to be fixed to a designated processor using *process affinity*, and both Windows and Linux OSes provide the functionality. The configurations in Figure 3.5.(a) are for homogeneous threads, i.e., multiple copies of the same program, and those in Figure 3.5.(b) for heterogeneous threads.



(a) Thread configurations for Homogeneous (HM) threads.



(b) Thread configurations for Heterogeneous (HT) threads.

Figure 3.5: Thread configuration for multi-core system (T=thread).

3.3. METRICS

Voltage droop is the main metric of this dissertation. Voltage droop is measured as the difference between the nominal voltage and the measured power supply voltage. Maximum voltage droop during the benchmark run is measured at the difference between the nominal voltage and the lowest power supply voltage during the entire run. The unit

of voltage droop used is mili-volt, ***mV***.

In hardware measurement, checking power is necessary to verify that voltage droop is a reasonable range. **Power** can be represented as several ways. Average power, P, is supply voltage (V) times average current (I), $P = V \times I$. Instantaneous power reflects voltage and current variations at a specific time, $p(t) = v(t) \times i(t)$. The unit of power is wattage, ***W***. However, high power does not always mean high di/dt noise.

Performance is defined as the reciprocal of runtime. Runtime is required to estimate energy or energy-delay. **Energy** is computed as the product of runtime and average power, and represents trade-offs between performance and power. Energy is multiplied by runtime to obtain energy-delay product.

Chapter 4: Analysis of Voltage Noise in Microprocessor

This chapter analyzes voltage droops caused by various benchmarks under different conditions. First, voltage droops caused by standard benchmarks such as SPEC CPU2006 and PARSEC are measured on the multi-core hardware and are analyzed to understand typical use case of programs. Next, synchronization effect by multiple threads is discussed and examined to see whether the corresponding di/dt noise is critical on multi-core, multi-threaded systems. The *natural dithering* effect caused by OS interference is observed and introduced. Finally, this chapter studies the compiler optimization impact on voltage droops with various benchmarks by adding a new perspective to Valluri and John's prior discussion [60].

4.1. VOLTAGE DROOP ANALYSIS FOR STANDARD BENCHMARKS

4.1.1. Voltage Droop Analysis using Simulation

The simulation method is used to analyze voltage droops on standard benchmarks in more detail. Simulation can identify the number of occurrences of voltage droops as well as the maximum voltage droop. Table 4.1 shows the architecture configuration used in the experiments in this section. The configuration targets a general 4-wide processor. In here, 22 benchmarks in the SPEC CPU2006 suite were run, and each benchmark runs 100 million instruction cycles using SimPoint [59]. The simulator was warmed-up for 10 million instruction cycles, and then traced di/dt for 90 million instruction cycles. PDN [61] has 5 to 16A of current swing, and two RLC stages. The nominal voltage was set to 1.0V.

Parameter	Values
CPU Clock	3 GHz
Fetch/Decode/Issue	4- / 4- / 4-instruction per cycle
EXU	2 alu, 2 mul/div, 2 falu, 2 fmul/fdiv, 2 mem-port
RUU / LSQ	128 / 64
Branch Predictor	Combined, 64Kb
BTB	1K entries
L1 I/D-Cache	64KB / 16KB, 2-way
L2 Cache	2MB, 16-way

Table 4.1: An architecture configuration for SimpleScalar.

Figure 4.1 shows the maximum voltage droop of the SPEC CPU2006 benchmarks. Y axis of Figure 4.1 is the maximum voltage droop in mV from the nominal voltage set to 1V. Among 22 benchmarks, *namd* has the largest maximum voltage droop, and *libquantum* has the smallest maximum voltage droop. The overall maximum voltage droop in the SPEC CPU2006 varied from 3.4% to 8.5%. One can identify the maximum droop from Figure 4.1, but the graph does not tell whether the maximum droop comes from a single, rare event, in other words, whether how many similar droops occur or not. Figure 4.2 shows the distribution of voltage droop levels during the entire run of each SPEC CPU2006 benchmark. X axis is the percentage of droop from the nominal voltage of 1.0V. Y axis is the number of occurrences in logarithmic scale. The number of occurrences in Figure 4.2 is inclusive, that is, 0.0% includes all the number of occurrences of voltage droops. Even though *GemsFDTD*'s maximum voltage droop is less than *namd*, *GemsFDTD* has more frequent voltage droop at 7% from nominal

voltage. Benchmarks *leslie3d* and *gromacs* do not have an 8% or 7% of voltage droop, but the number of occurrences of two benchmarks is much higher than others. This result implies that a voltage droop analysis only with the maximum voltage droop could ignore the possibility of real failures resulting from highly frequent and comparably larger droops. Voltage at failure analysis is considered and explained in the following chapter.

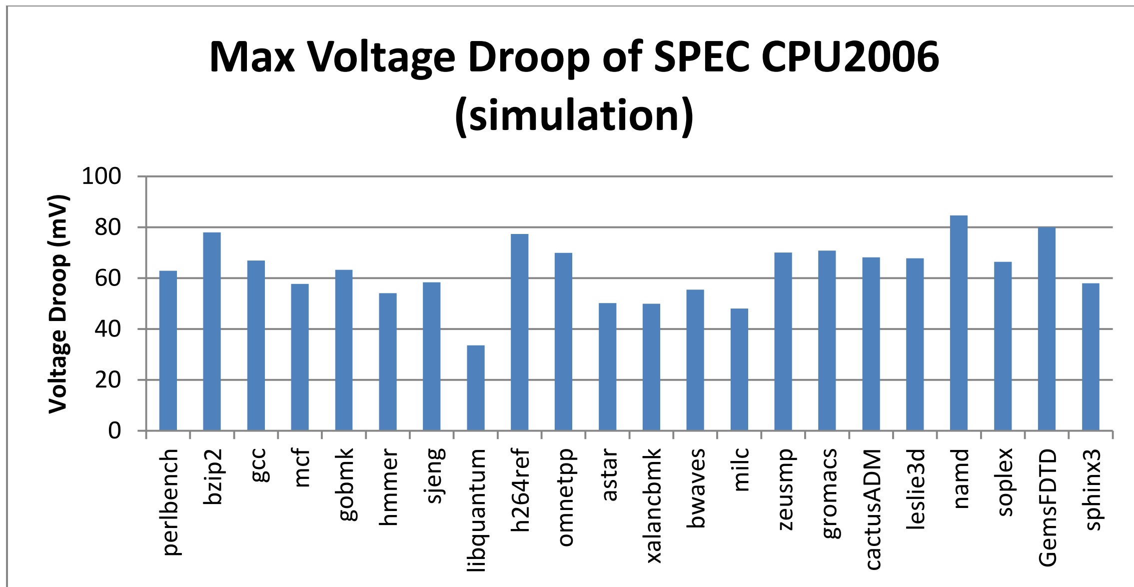


Figure 4.1: Maximum voltage droop of SPEC CPU2006 in simulation. Vdd is 1.0V.

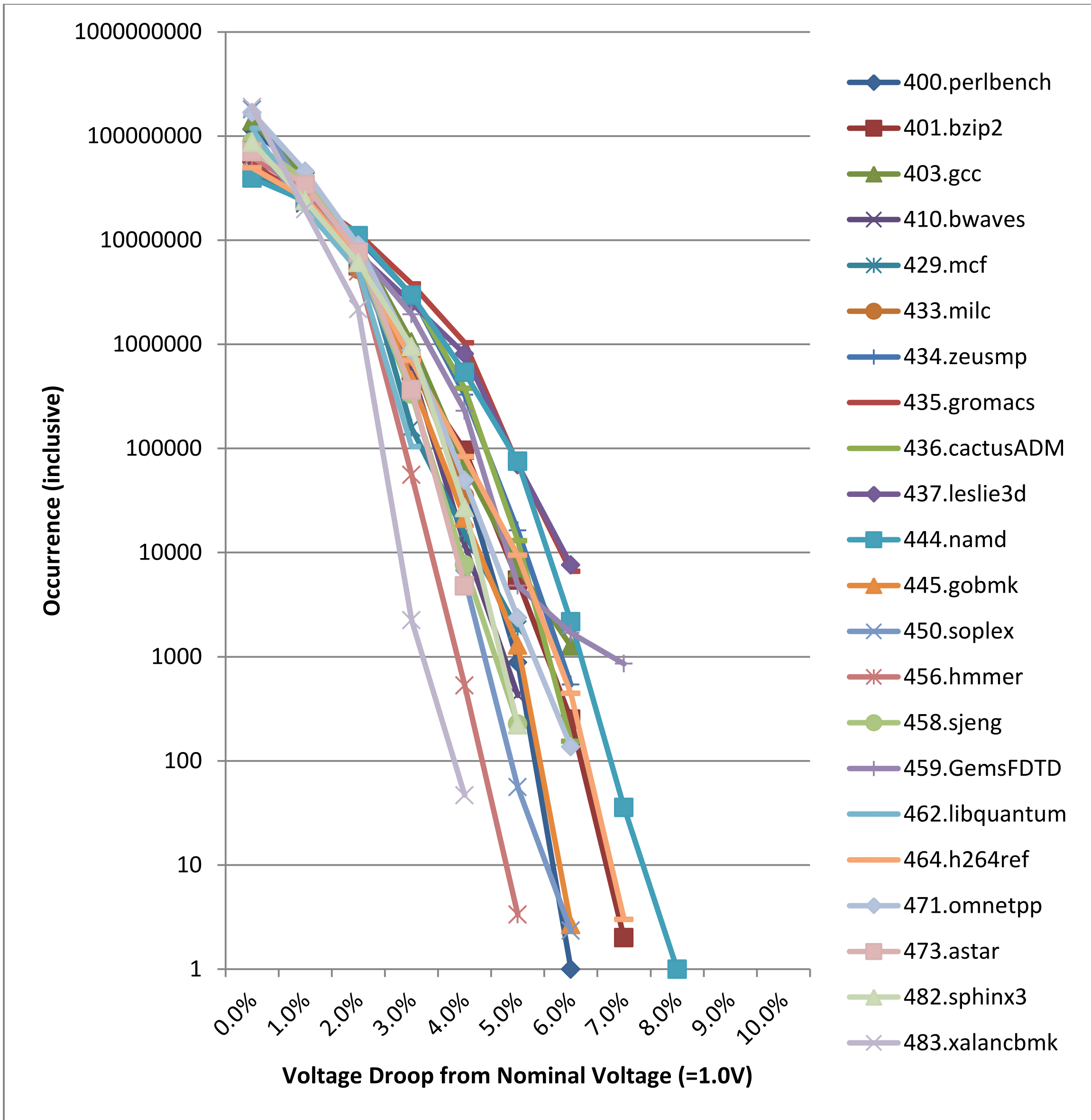


Figure 4.2: Occurrence (inclusive) at each voltage droop at Vdd=1.0V.

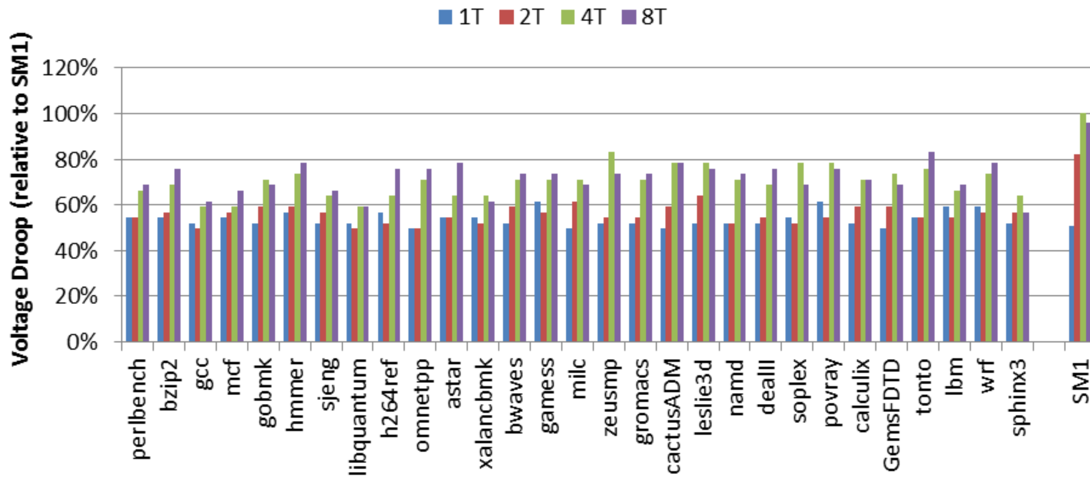
4.1.2. Voltage Droop Analysis using Hardware Measurement

Figure 4.3 shows the maximum droop measured from running the SPEC

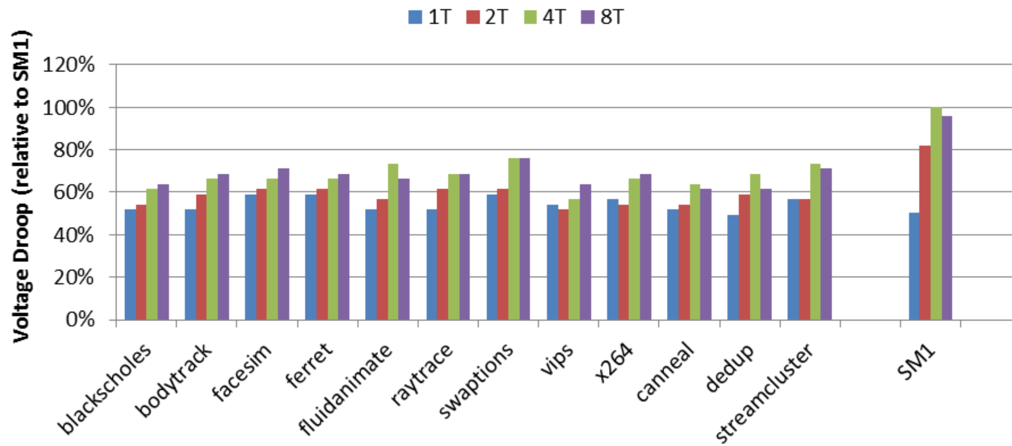
CPU2006 benchmarks and PARSEC multi-threaded benchmarks in configurations of one-, two-, four-, and eight-thread runs (1T, 2T, 4T, and 8T). For SPEC CPU2006, the program is replicated and executed on multiple cores, similar to **SPECrate**. Given the shared nature of the cores in the Bulldozer module, higher voltage droops occur for a given number of threads when threads are spatially distributed across modules. The evaluation processor has four Bulldozer modules, each with two cores. Hence, for the 1T, 2T, and 4T runs, each thread is assigned to a different module. For the 8T runs, there are two threads assigned to each module. All droop results are shown relative to the 4T *SM1,* an industry-level manual stressmark, and higher numbers indicate larger droops. The values are measured with the load line of the voltage regulator module (**VRM**) disabled to remove any load-line droop effects [33]. Hence, the results show the droop due to di/dt stresses only.

Figure 4.3 shows that, in general, the magnitudes of the voltage droops increase with the number of threads for 1T, 2T, and 4T configurations. The 8T configurations do not always follow this trend due to multi-threading in the Bulldozer module (explained later in this section). Figure 4.3 also shows that most of the SPEC CPU2006 and PARSEC benchmarks except *zeusmp* and *tonto* have 20% less droop than SM1, the manual stressmark. It can be concluded that one needs di/dt stressmarks for worst case analysis of di/dt events. In the SPEC CPU2006 benchmarks, the averaged maximum droop of all the floating point benchmarks is larger than that of all the integer benchmarks. This could mean that floating point execution path in the processor is more susceptible to voltage noise. However, because voltage droop of SPEC CPU2006 is not significant compared to that of SM1, further analysis is required even though it is known that floating point execution units in Bulldozer consume significantly more power than

any other units in the module.



(a) Relative maximum voltage droop for SPEC CPU2006 benchamrks.
(relative to *SM1* manual stressmark)



(b) Relative maximum voltage droop for PARSEC benchamrks.
(relative to *SM1* manual stressmark)

Figure 4.3: Hardware measurements of droop (relative to 4T *SM1*) for SPEC CPU2006.

As noted in Chapter 2, one way to generate a significant droop is to have a large change in activity from idle to full execution. For high-performance pipelines, such a change in activity occurs naturally with certain pipeline events, such as pipeline recovery after a branch misprediction stall or high execution activity after a load miss resolves [53]. These events are commonplace in complicated pipelines, and how they interact with each other in a multi-threaded scenario dictates how large a droop they produce. Destructive interference may occur between threads in a multi-core system such that when one thread is in a high-power state others are in a low-power state. Reddi et al. describe the issue of thread misalignment for the SPEC CPU2006 benchmarks, examine constructive and destructive interference in a dual-core system, and discuss co-scheduling threads to reduce voltage droops [56][56][23].

The PARSEC multi-threaded benchmark suite could have alignment between threads through its use of synchronization primitives. The expectation was that higher droops would be seen due to the natural alignment resulting from barrier operations in benchmarks such as *fluidanimate* and *streamcluster* as discussed in [34]. However, the results show no significant difference in droops between the PARSEC and the SPEC CPU2006 suites.

To further evaluate this, a barrier stressmark was designed that repeatedly synchronizes on a barrier operation and then runs the high-power virus in a 4T configuration. This was expected to result in a large voltage droop due to all cores being aligned and idle at the barrier operation followed by high activity on the cores. The resulting droop, however, was not significant. On further examination, it is noticed that a natural misalignment occurs between the cores when released from a barrier. On the Bulldozer module, there is no explicit mechanism to synchronize the barrier release

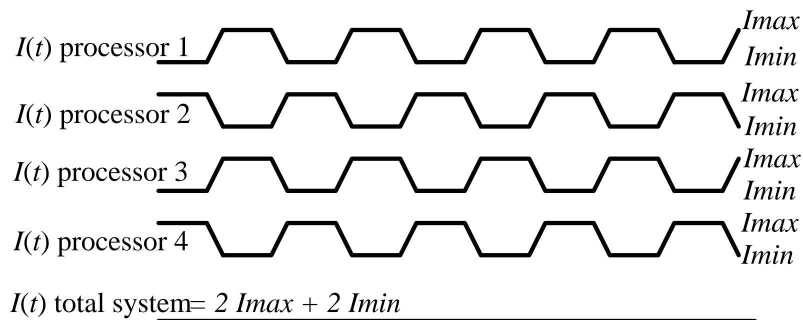
signal, and the signal naturally reaches each core at different times based on from where in the memory hierarchy the core receives its data. This perturbs the start of activity across the cores by enough cycles to dampen the first droop excitation resulting from the synchronization operation.

Miller et al. claimed that, in a many-core system, the synchronization effect by multi-threading is the main cause of supply voltage fluctuation rather than high activity from each individual core [34]. However, the authors in [34] examined a different x86 processor that may have different characteristics. In addition, they use fluctuations in average power estimated at intervals of 1 ms on hardware as a proxy for expected di/dt variations. This may capture third droop excitations, but not first droop excitations that occur over the course of nanoseconds. The measurement technique in this dissertation is capable of identifying the high-frequency first droop variations in voltage. Hence, the worst-case droops in PARSEC are most likely the result of the same microarchitectural events that align across multiple threads in the SPEC CPU2006 suite. The authors in [34] also note that barriers are not the only cause of high power swings and point to microarchitectural events such as long-latency cache misses followed by bursts of activity as other potential inducers of high droop. Furthermore, the effect in [34] is pronounced for cases with 32 threads, whereas the experiments in this dissertation did not include such configurations.

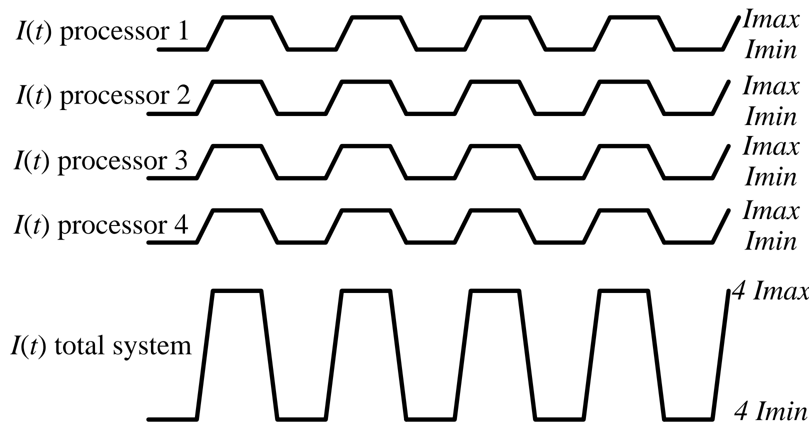
4.2. Synchronization (Alignment) effect of Multiple Threads

As noted in other papers [9][15][34][56], multiple threads running simultaneously can have a constructive or destructive impact on droops (Figure 4.4). If the threads align

correctly, they produce significantly larger droops than without alignment. At first glance, thread alignment would seem to be a low-probability event in multi-core machines with complex, out-of-order cores and shared and non-shared resources. However, the analysis in this dissertation shows that alignment occurs relatively often when the stressmark consists of short loops due to natural perturbations in the threads caused by OS thread scheduling. This phenomenon is referred to as *natural dithering*.



Completely misaligned system: no swings in total current



Completely aligned system: large amplified current swing

Figure 4.4: Alignment: misaligned-destructive (top) and aligned-constructive (bottom).

Figure 4.5 shows an example of *natural dithering* over the course of 100 ms when running a four-threaded resonant stressmark in which the threads are the same and consist of short loops. Each major grid point represents 10 ms and the y axis shows measured processor voltage (*Vdd*) values using a 100 megasamples/second (MS/s) sampling rate. Approximately every 16 ms, which corresponds to the OS timer tick on Windows systems, *Vdd* variability changes. When the threads align constructively, as is the case near the center point of the scope shot, the droop is maximized.

This data shows that small, repetitive loops occurring across multiple threads at the same time can result in significant di/dt stresses in the system due to natural dithering resulting from OS interaction. This phenomenon would have been difficult, if not impossible, to observe in a simulation environment. This type of behavior is more likely to occur in certain high-performance computing applications that consist of short, repeated loops. Relying on OS behavior to align threads is not a reliable method to determine the worst-case droop. Hence, a dithering algorithm is required, which guarantees a worst-case droop within a fixed amount of time once OS interrupts are disabled.

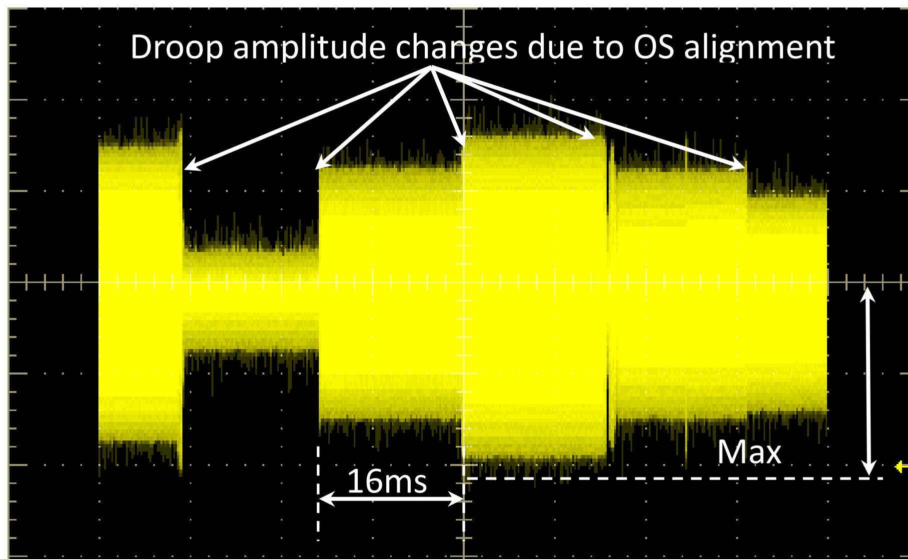


Figure 4.5: Scope shot of *natural dithering* due to OS interactions for resonant stressmark over a period of 100 ms.

4.3. IMPACT OF COMPILER OPTIMIZATION ON VOLTAGE DROOP

Voltage droops due to di/dt effects have been studied in the past. However, no prior work studies the effect of compiler optimizations on voltage droops. Past work has studied the impact of compiler optimizations on performance and power, but not reliability. In this section, voltage droops are analyzed with different compiler optimization levels. Also corresponding performance, power and energy results are reported to put the results into perspective. No clear trends could be observed regarding the effect of compiler optimizations on voltage droops. Therefore, it can be concluded that dynamic voltage noise mitigation is necessary because voltage noise reduction with static compiler optimization cannot be guaranteed.

4.3.1. Compiler Impact on Performance, Power, and Reliability

The order of an instruction sequence, instruction scheduling, is important in performance and power. Compilers usually optimize scheduling for high performance, but not for low power because it is difficult to provide power details to compilers. Compilers also affect the choice of instructions used to accomplish the task.

The work in this chapter is highly motivated by Valluri and John [67], where the authors studied compiler impact on performance and power. Valluri and John studied the impact of compiler optimizations on performance and power, and concluded that (1) performance improvement by reducing the number of instructions induces energy reduction and (2) performance improvement by increasing the overlap in programs induces power increase. In here, Valluri and John's discussion is extended by adding a new perspective and by studying reliability with voltage droops while running various programs on a real multi-core, multi-threaded processor hardware.

This chapter is based on measurement on actual hardware. In contrast to Valluri and John's methodology [67], the differences made are described in Table 4.2. Performance, power, and reliability are measured with two different compiler optimization levels, –O0 and –O3. Then, energy is calculated with the measured performance and power numbers. The performance between the cases with no optimization versus full optimization is compared, because there were only slight differences among -O1 to -O3.

	Valluri and John [67]	This Dissertation
Power/Reliability	Average power	Average power Voltage droop
Run Method	Simulation study based on SimpleScalar/Wattch	Hardware Measurement based on Post-Silicon Processor
Run Thread	Single-core/single-thread	Multi-core/multi-threaded
Runtime	Partial run	Entire run
Benchmark	SPEC95	SPEC CPU2006 (multi-programmed) miniFE (multi-threaded) HPC Linpack (multi-threaded)

Table 4.2: Comparison of experimental methodology between Valluri and John [67] and this dissertation.

4.3.2. Experimental Results

To analyze performance, power, and voltage droops with different compiler optimization levels, various benchmarks run from a small but highly scalable program to a high performance program and a standard benchmark suite. Also a real hardware system is used rather than a simulator. Through the measurements on silicon, it is expected that the study significantly reduces possible errors and uncertainty in the

abstraction and modeling steps of a processor for simulation method.

4.3.2.1. Experimental Setup

To analyze the impact of software optimization on power and voltage droop, an **AMD Orochi** processor is used for hardware measurement. For benchmarks, *miniFE* [35], a scalable, multi-threaded program, which adjusts the problem size according to the number of multiple cores is used. Another program used is *High-Performance Linpack* (*HP Linpack*) [20], which is well known for benchmarking Top 500 supercomputers [66]. Both *miniFE* and *HP Linpack* use *OpenMPI* [39] library for scalable, multi-threading technique to maximize the parallelism of multi-core processor. As a standard benchmark, SPEC CPU2006 [62] can show various, normal program behavior by compiler optimization effect. Each benchmark is compiled by *gcc/gfortran* 4.6.2 with –O0 and –O3 levels separately on *RedHat Enterprise Linux 6* OS.

Several metrics are used to analyze the impact of the compiler optimization levels on the programs. Performance is measured in runtime or in the inverse of runtime, and is reported by the benchmark program itself. Power is measured in wattage, and is calculated from supply voltage and current variations measured as voltage drop on a unit resistor. Voltage droop is measured with an oscilloscope and differential probes, which are attached to main supply voltage pins on the processor package.

4.3.2.2. Result of miniFE

miniFE [35] is a mini-application that mimics finite element generation, assembly and solution for an unstructured grid problem. Table 4.3 shows runtime, power, voltage

droop values according to the number of multiple threads. Each value is normalized to 1 thread (1T) case.

Metric	#of Threads				
	1T	2T	4T	8T	16T
Runtime	1.00	0.51	0.28	0.20	0.20
Power	1.00	1.38	2.16	2.86	2.81
Vdroop	1.00	1.00	1.04	0.96	0.96
Energy	1.00	0.71	0.61	0.58	0.57

Table 4.3: Runtime, power, maximum voltage droop, and energy of *miniFE* with increase of the number of threads.

Figure 4.6 shows the relative values of runtime, power, droop and energy for different numbers of threads, compared to those of one thread case. The values are saturated starting at 8T because of the processor's architectural limitation (2 threads per Bulldozer module; hence 8 threads from 4 modules).

The following trends are observed:

- At 8T or 16T, the runtime is reduced to one fifth of 1T.
- At 8T, power increases up to three times compared to 1T.
- Voltage droop slightly changes according to #of Ts.
- Average power variation does not significantly affect voltage fluctuation.

Voltage scaling will be needed for 8T and 16T if a system exceeds its TDP, i.e.,

power constraint. However, if the voltage margin is not enough, voltage scaling is not applicable due to margin violation and frequency scaling is required despite performance degradation. Another conclusion is that energy starts to saturate from 4T. Therefore, if thermal constraint should be considered, 4T is optimal not only for sustaining the same battery life but also for keeping good performance.

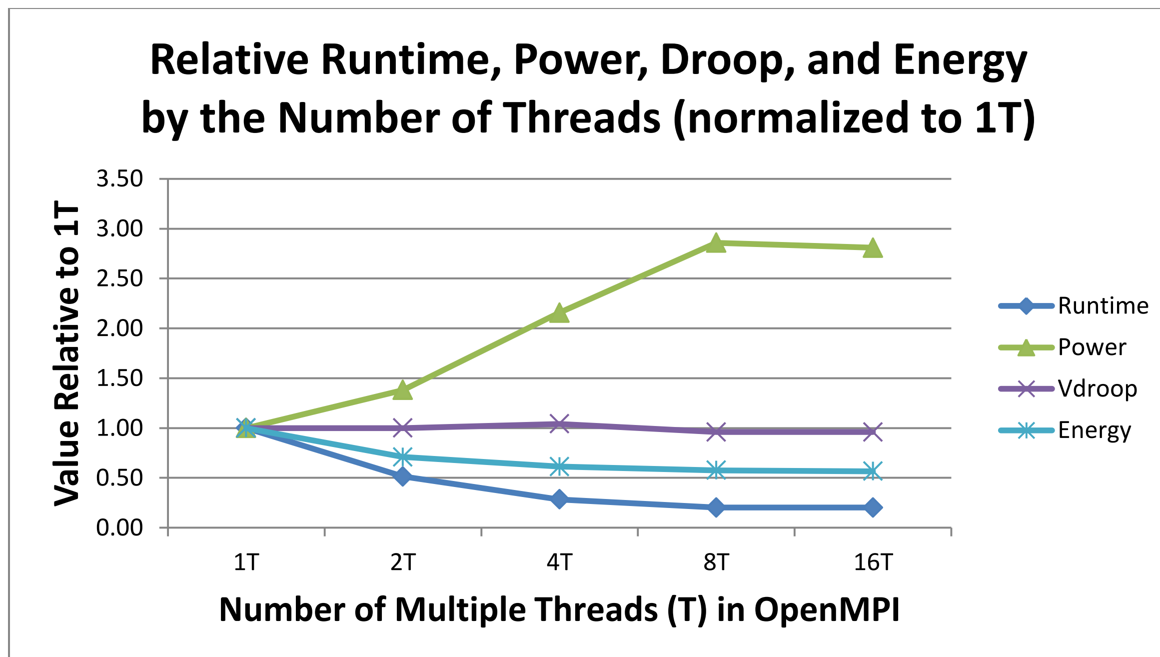


Figure 4.6: Runtime, power, droop, and energy of *miniFE* (relative to 1T case)

Voltage droops in *miniFE* are not seriously changed by compiler optimization levels because it highly depends on the *OpenMPI* library, which is already optimized with -O3. For Table 4.3 and Figure 4.6, its dimension was set to nx=150, ny=150, and nz=150. The high level of compiler optimization used in the library makes the difference between unoptimized and optimized code fairly small.

4.3.2.3. Result of High-Performance Linpack

The *High-Performance Linpack* (*HPL*) benchmark is a very popular way to measure performance of supercomputers (Top 500 [66]). First, the benchmark was compiled and run with two different optimization levels, -O0 and -O3, but there is no difference in performance, power, and droop between -O0 and -O3. It is found out that the benchmark highly depends on *Basic Linear Algebra Subprogram* (*BLAS*) library such as *daxpy* and *dgemm*. The library is necessary for multi-threading and is usually provided by a processor vendor for a specific architecture. For the processor vendor's pre-compiled *BLAS* library on its developer's site [6], source codes of the library that are required to recompile the library with different compiler optimization levels could not be obtained.

Finally, the original *BLAS* library on the national lab's web page [10] was obtained and used for the compiler optimization experiments. The performance of the original *BLAS* library is much worse than that of the processor vendor's *BLAS*, but the clear changes could be seen in performance, power, and voltage droop according to compiler optimization levels.

In Table 4.4, *HP Linpack*'s performance is highly affected by optimization methods. The compiler optimization gives five times performance improvement than no optimization, and the library optimization by the processor vendor increases performance more than five times beyond the compiler optimized case. The performance results also show the importance of the library optimization for a multi-threaded application.

The increases in performance usually are accompanied by increases in power consumption, but it is reduced by 35% from -O0 to -O3. Even with **AMD** *BLAS*, power remains the same compared to -O0 of original *BLAS*. It is interesting that 29X

performance improvement can be obtained at the same power, by compiler optimizations. The tradeoffs between performance and power are not uniform between various benchmarks or between various optimizations. Maximum voltage droop of HP Linpack changes by 7% with compiler optimization and by 5% with library optimization, compared to maximum voltage droop with no optimization. If very tight voltage margins are used to save power and energy, these voltage fluctuations can cause unreliable operation. Energy is calculated using the execution time and power values. It is interesting that 89% and 97% of energy can be dramatically reduced by compiler and library optimizations, in the original BLAS and vendor-BLAS respectively. For HP Linpack, the processor vendor's recommended configuration was used for the problem dimension parameters [1].

Metric	Original BLAS		AMD BLAS
	O0	O3	O3
Performance	1.00	5.83	29.17
Power	1.00	0.65	1.00
Vdroop	1.00	1.07	1.05
Energy	1.00	0.11	0.03

Table 4.4: Runtime, power, maximum voltage droop, and energy of High-Performance Linpack (8T) with different libraries and optimization levels. The values are normalized to the Original BLAS and O0 case.

4.3.2.4. Result of SPEC CPU2006

All the CINT2006 and CFP2006 benchmarks in the SPEC CPU2006 suite were run to see performance, power, and voltage variation in a multi-programmed manner. Each SPEC benchmark is a single-threaded program, so multiple copies of the same program are running on each Bulldozer module to calculate the SPECrate. First, each benchmark with a single thread was run on one of four Bulldozer modules with an affinity in order not to cause thread migration effect that could distort voltage droop measurement results. Then, 4 copies of the same program were run on each Bulldozer module (there are 4 Bulldozer modules in the current processor) and 4T cases were compared to 1T cases.

The single thread performance is discussed first. Table 4.5 presents the relative values of -O3 normalized to -O0 values. Every SPECrate value in Table 4.5 is greater than 1.00 meaning that performance is always improved with compiler optimization. However, power does not show any uniform trend with compiler optimization. Voltage drdroop changes from -15% to +15% according to benchmark, but it has no trend, either.

Benchmark	SPECrate	$Power_{avg}$	Vdroop	Energy
perlbench	1.41	0.98	1.09	0.70
bzip2	2.41	1.00	0.85	0.41
gcc	1.91	1.00	1.00	0.52
mcf	1.88	1.05	1.00	0.56
gobmk	1.78	1.00	1.00	0.56
hmmer	3.54	1.01	1.00	0.29
sjeng	1.72	0.99	0.89	0.57
libquantum	2.05	0.98	0.89	0.48
h264ref	2.79	1.01	0.96	0.36
omnetpp	2.14	0.96	1.00	0.45
astar	2.21	0.97	0.96	0.44
xalancbmk	5.17	0.97	1.09	0.19
bwaves	3.38	1.04	1.15	0.31
gamess	2.82	1.02	1.09	0.36
milc	3.45	0.97	0.89	0.28
zeusmp	3.43	1.01	1.04	0.29
gromacs	2.39	1.01	0.92	0.42
cactusADM	4.05	0.99	1.00	0.25
leslie3d	5.48	1.02	1.00	0.19
namd	3.70	1.00	1.00	0.27
dealII	8.12	0.95	0.96	0.12
soplex	2.59	0.93	1.00	0.36
povray	2.73	0.96	1.00	0.35
calculix	9.39	0.97	1.00	0.10
GemsFDTD	4.85	1.00	0.96	0.21
tonto	1.88	0.99	0.96	0.53
lbm	1.94	0.94	1.08	0.48
wrf	4.85	0.96	1.00	0.20
sphinx3	3.38	1.00	1.04	0.30

Table 4.5: SPEC CPU2006 1T Results with –O3 (normalized to –O0)

Next, this dissertation studies the impact of compiler optimization on 4T cases (Table 4.5), and compares the effects to 1T cases. Performance improvement (SPECrate), due to optimization, is less in 4T case compared to single thread case. It is because of the contention of multiple threads for shared resources such as L3. Even though none of four threads run on the same Bulldozer module, contentions are unavoidable for L3 and memory accesses.

Most benchmarks in SPEC CPU2006 take less power in -O3 compared to -O0 indicated by the ratios in column 2 except *gamess*, *gromacs* and *tonto*. This could be because idle time out of total runtime increases due to resource contentions. The voltage droop ranges from -15% to +15% in various benchmarks, but it has no clear trend with optimizations. In some benchmarks such as *h264ref*, voltage droop increases when compiler optimization for 4T cases, but voltage droop decreases with compiler optimization for 1T case. With compiler optimization, energy is reduced by 30% to 90% in 1T (*perlbench* and *calculix*, respectively in Table 4.5), and by 8% to 89% in 4T (*lbm* and *calculix*, respectively in Table 4.6). Due to the degradation of performance in 4T, 4T's energy reduction ratio with compiler optimization decreases, compared to 1T.

Benchmark	SPECrate	Power	Vdroop	Energy
perlbench	1.43	0.94	1.20	0.65
bzip2	2.39	0.96	1.00	0.40
gcc	1.72	0.93	0.93	0.54
mcf	1.24	0.96	0.86	0.77
gobmk	1.77	0.96	1.16	0.54
hmmer	3.54	0.98	1.04	0.28
sjeng	1.71	0.94	1.00	0.55
libquantum	1.04	0.92	0.89	0.89
h264ref	2.79	1.00	1.08	0.36
omnetpp	1.53	0.85	1.04	0.55
astar	1.97	0.91	0.93	0.46
xalancbmk	3.99	0.84	0.92	0.21
bwaves	2.35	0.96	1.14	0.41
gamess	2.82	1.05	1.03	0.37
milc	1.46	0.79	1.00	0.54
zeusmp	2.86	0.96	1.03	0.34
gromacs	2.41	1.01	0.87	0.42
cactusADM	3.17	0.92	1.14	0.29
leslie3d	2.07	0.85	1.00	0.41
namd	3.69	1.06	0.91	0.29
dealII	7.59	0.94	1.04	0.12
soplex	1.35	0.82	0.90	0.61
povray	2.72	0.98	1.04	0.36
calculix	9.34	1.00	0.96	0.11
GemsFDTD	1.76	0.83	1.03	0.47
tonto	1.82	1.01	1.00	0.55
lbm	0.99	0.92	0.91	0.92
wrf	3.76	0.91	1.03	0.24
sphinx3	2.28	0.84	1.00	0.37

Table 4.6: SPEC CPU2006 4T Results with –O3 (normalized to –O0)

The following observations are noted from these experiments:

- Regarding compiler optimization and its effect on performance: programs compiled with -O3 are faster than those with -O0 in most cases. However, *lbm* did not follow this trend with 4T.
- Regarding compiler optimization and power: codes compiled with -O0 optimization level (i.e. unoptimized code) need more power than codes compiled with -O3. This observation is consistent with the observation in Vallu and John [67].
- Regarding compiler optimization and supply voltage droop: nothing can be concluded because of mixed trend in the results. About one third of the benchmarks show more droops in the optimized case and about a third of the benchmarks show less droop in the optimized case. About a third of the benchmarks show no difference in the droop between optimized and unoptimized code.
- Regarding compiler optimization and energy: energy reduction is always observed with higher optimization, but the amount of savings can vary largely from one benchmark to another.

Another perspective is the impact of multithreading on voltage fluctuations and hence reliability. When SPEC programs are run in the SPECrate mode, the droops increase as they go from 1T to 4T cases.

4.3.3. Summary of Compiler Optimization Impact

The experiments were conducted to study the impact on compiler optimizations on the voltage fluctuations during program execution. Several programs were run with optimized and unoptimized versions of code from the same program, and performance, power, energy and voltage fluctuations studied.

Energy can be dramatically reduced by increasing the number of threads and performing compiler optimization. Performance can be significantly improved by compiler optimizations. Trends in average or maximum power during execution cannot be correlated in a systematic manner with performance. The intricacies of the code sequences and the functional units bring unpredictable trends between performance and power trade-offs.

Generally one cannot predict how voltage droop would change with optimization levels. In some cases, performance, power, and voltage droop can be improved with compiler optimization (*mcf*, *lbm*, etc.). Short-runtimes could give different results in voltage droop. Therefore, designers utilizing simulations for such studies should be careful interpreting simulation results, which are run with very short time compared to real hardware.

Voltage droop does not change much with compiler and library optimizations in *miniFE* and *HP Linpack*. Therefore, with compiler optimization only, supply voltage reliability is mainly affected by the load-line effect [33], i.e., the resistive part rather than the inductive part of the processor and power distribution network circuits. If one cannot predict voltage droop change with static compiler optimization, a dynamic mitigation method can be useful.

4.4. SUMMARY

Voltage droop in SPEC CPU2006 and PARSEC was analyzed in the first section of this chapter. The voltage droop of standard benchmarks is not significant compared to that of the manual stressmark, and multi-threading effect is not significant, either. From these results one can conclude that di/dt stressmarks are required for worst case analysis of di/dt problem.

Synchronization of multiple threads was thought to make a large single droop in a processor. However, this work showed that the 1st droop excitation does not occur because the launching time differences among multiple threads are much larger than the 1st droop resonant period. 2nd and 3rd droop excitation is possible because the start and end time can be managed within sub-micro-second time period.

There is no clear trend in compiler optimization impact on di/dt noise. Significant performance improvement by compiler optimization does not necessarily result in power tradeoffs, so energy can be reduced dramatically in many cases.

Chapter 5: AUDIT Framework to Generate di/dt Stressmarks

Rapid current changes (large di/dt) can lead to significant power supply voltage droops and timing errors in modern microprocessors. To test a processor's resilience to such errors and to determine appropriate operating conditions, engineers manually create di/dt stressmarks that have large current variations at close to the resonance frequency of the power distribution network (PDN) to induce large voltage droops. Although this process can uncover potential timing errors and be used to determine processor design margins for voltage and frequency, it is time-consuming and may need to be repeated several times to generate appropriate stressmarks for different system conditions (e.g., different frequencies or di/dt throttling mechanisms). Furthermore, generating efficient di/dt stressmarks for multi-core processors is difficult due to their complexity and synchronization issues. It will be valuable if a di/dt stressmark can be generated without tedium and without detailed knowledge of the microarchitecture.

In this chapter, an AUtomated DI/dT stressmark generation framework, refered as AUDIT, is proposed to test maximum voltage droop in a microprocessor power distribution network. The di/dt stressmark from the framework is an instruction sequence which draws periodic high and low current pulses that maximize voltage fluctuations including voltage droops. In order to automate di/dt stressmark generation, a code generator is devised with the ability to control instruction sequencing, register assignments, and dependencies. AUDIT uses a Genetic Algorithm in scheduling and optimizing candidate instruction sequences to create a maximum voltage droop. In addition, AUDIT provides with both simulation and hardware measurement methods for finding maximum voltage droops in different design and verification stages of a

processor.

Using the simulation path, the results show that the automatically generated di/dt stressmarks achieved more than 40% average increase in voltage droop compared to hand-coded di/dt stressmarks and typical benchmarks in experiments covering three microprocessor architectures and five power delivery network (PDN) models.

Using the hardware measurement path, measurement and analysis of di/dt issues are conducted on state-of-the-art multi-core x86 systems using real hardware. It is shown that AUDIT has capabilities to adjust to microarchitectural and architectural changes. A dithering algorithm is adapted to address thread alignment issues on multi-core processors.

5.1. AUDIT FRAMEWORK

Figure 5.1 shows the basic framework for AUDIT. AUDIT takes as input the instructions used to generate the stressmark and some control parameters such as the cost function and exit conditions. This information is fed to a code generator to produce a population of potential stressmarks. The initial population of stressmarks either can be generated randomly or seeded with existing benchmarks or stressmarks to improve the convergence rate.

Figure 5.1 includes two possible paths for stressmark generation, simulation and hardware. With the simulation path (top of Figure 5.2), the voltage droops of generated instruction sequences are evaluated using a cycle-accurate simulator that produces current draw information followed by SPICE simulation. This path is most appropriate when hardware for performing di/dt stressmark generation is not available. With this approach, the assembly code instruction sequence is compiled into a simulator-friendly format (e.g.,

x86 binaries). The compiled code is executed on the cycle-accurate simulator and every cycle the simulator calculates the current draw of the processor based on the activity of internal modules of the processor. This methodology is similar to that used in other work [14][24][48]. AUDIT converts the per-cycle current profile into a current sink in HSPICE simulation using a lumped RLC model of the PDN. The HSPICE simulation produces a series of voltage droops over time from which the maximum voltage droop can be obtained.

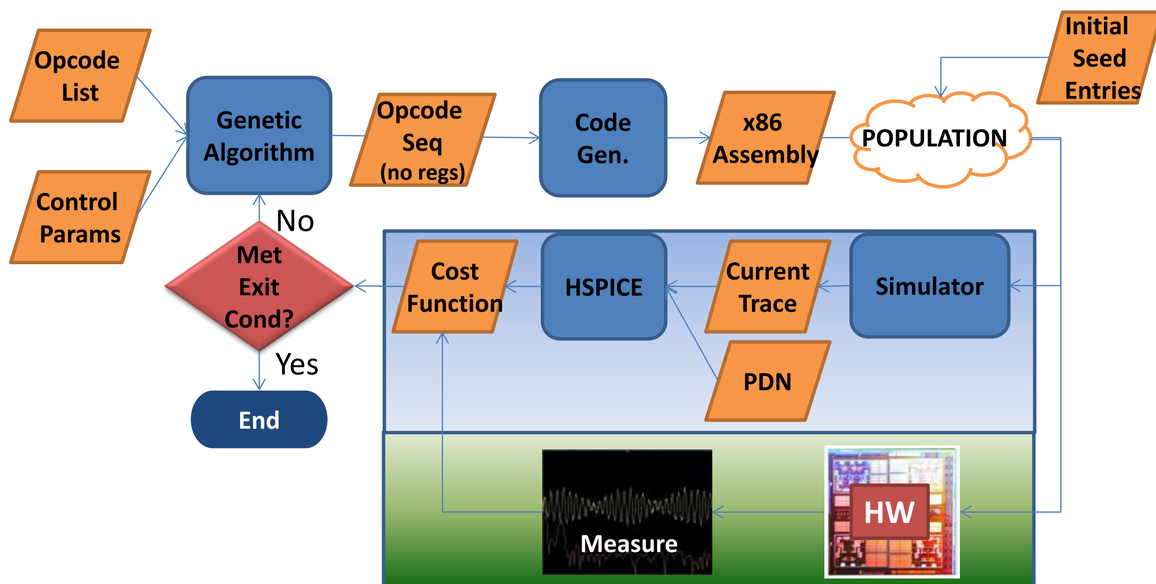


Figure 5.1: AUDIT framework for di/dt stressmark generation using simulators and hardware.

With the hardware path (bottom of Figure 5.2), the stressmarks are run on a processor board and measurement tools capture voltage droops, power dissipation, and any other information necessary to evaluate the cost function of the stressmark. The stressmarks and their associated cost values are fed to the GA for further refinement until

the exit conditions are met (e.g., the maximum voltage droop produced by AUDIT does not increase for several generations).

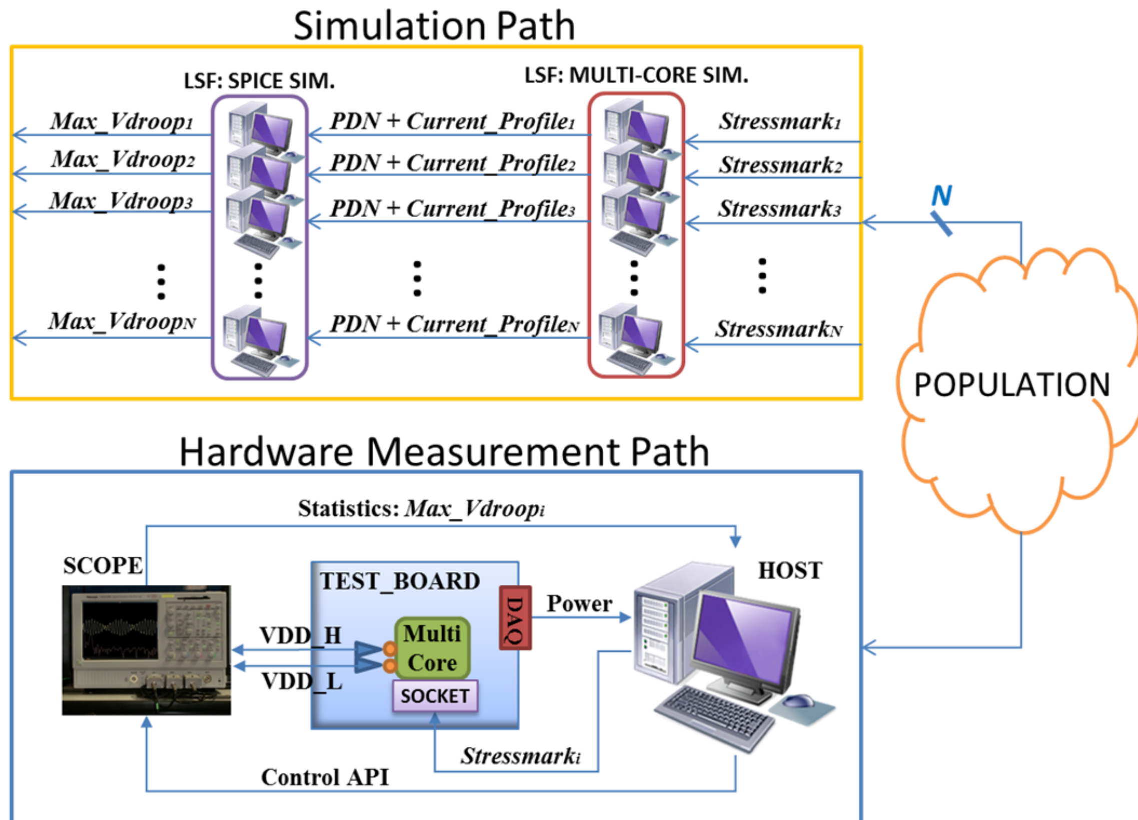


Figure 5.2: Simulation and hardware measurement paths to get max voltage droops.

5.1.1. Instruction Scheduling using the Genetic Algorithm

Genetic Algorithms (GAs) are known to be very efficient in solving optimization problems by finding a best fitness value for the problem by killing inferior candidates and promoting superior ones. In the AUDIT framework, di/dt stressmark generation is considered as an instruction scheduling problem, and the objective function is set to

maximize the supply voltage droop. Then, GA optimizes the instruction scheduling. Figure 5.3 shows a conceptual instruction scheduling using GA. The AUDIT framework initially generates random instruction sequences, and they are forced to reproduce, mutate, and compete for maximizing the voltage droop as the algorithm proceeds.

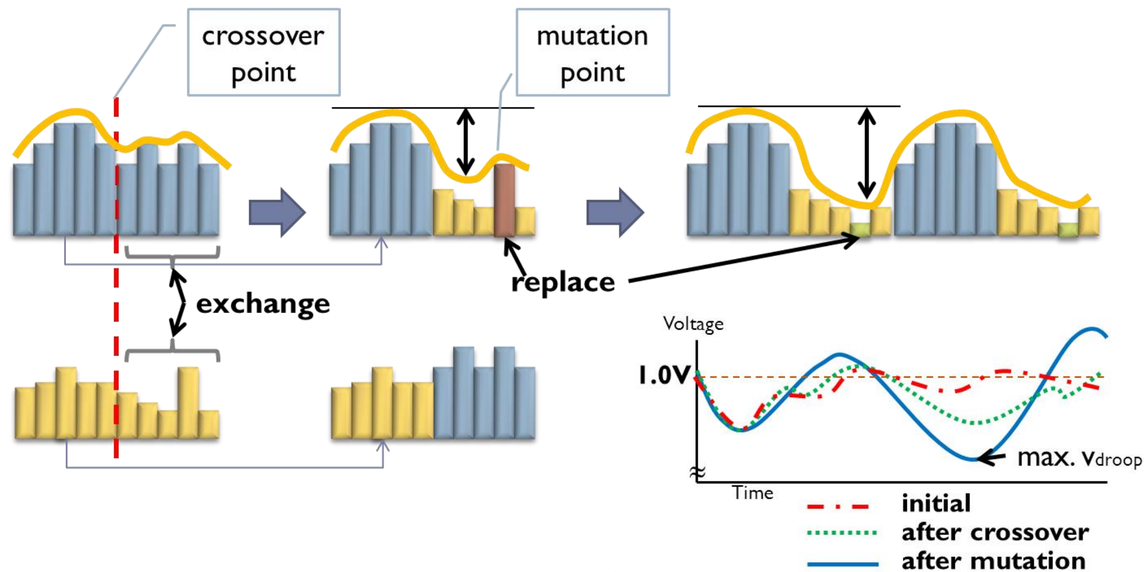


Figure 5.3: Conceptual instruction scheduling in the Genetic Algorithm.

5.1.2. Framework Control using the Genetic Algorithm

The Genetic Algorithm guides the AUDIT stressmark generation framework as shown in Figure 5.4, and generates a di/dt stressmark as an output. With a control parameter setting, initial instruction sequences are generated and consist of a population in the first generation. All the individuals in the population are evaluated for the objective

function – maximum voltage droop - with multiple simulations. Then, two of the highly ranked individuals in the population are selected for reproduction, and they exchange a certain number of instructions with each other. The rate of reproduction is called *crossover rate* and it affects the overall optimization results because the crossover rate determines the speed of convergence of the algorithm. After crossover, the characteristic of each individual can be changed by mutation that converts one or multiple bits of an individual instruction. Such GA operations repeat for a given number of generations, and a maximum voltage droop is determined at the end of the last generation.

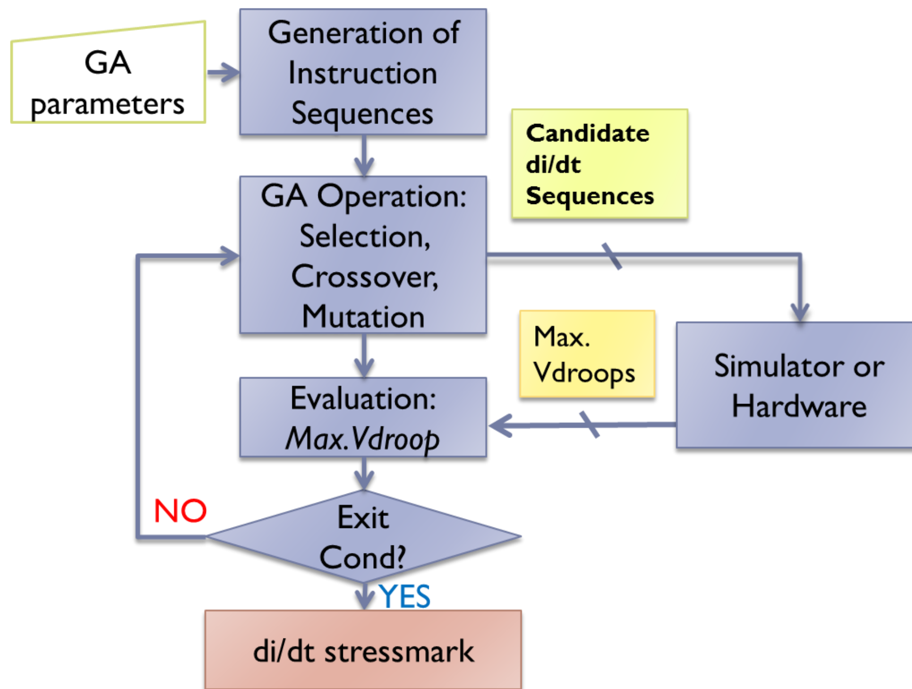


Figure 5.4: Control of stressmark generation framework using Genetic Algorithm.

5.1.3. Instruction Sequence Generation for the Genetic Algorithm

Figure 5.5 depicts instructions, stressmark size, and candidates for the di/dt stressmark in the Genetic Algorithm. An instruction consists of an opcode (OPCODE), operands (OR), and dependencies and is represented as a bit-string for the *chromosome*. A certain number of chromosomes are placed in an *individual* (stressmark size) that becomes an instruction sequence and a possible di/dt stressmark. *Population* is a collection of individuals and corresponds to one *generation*.

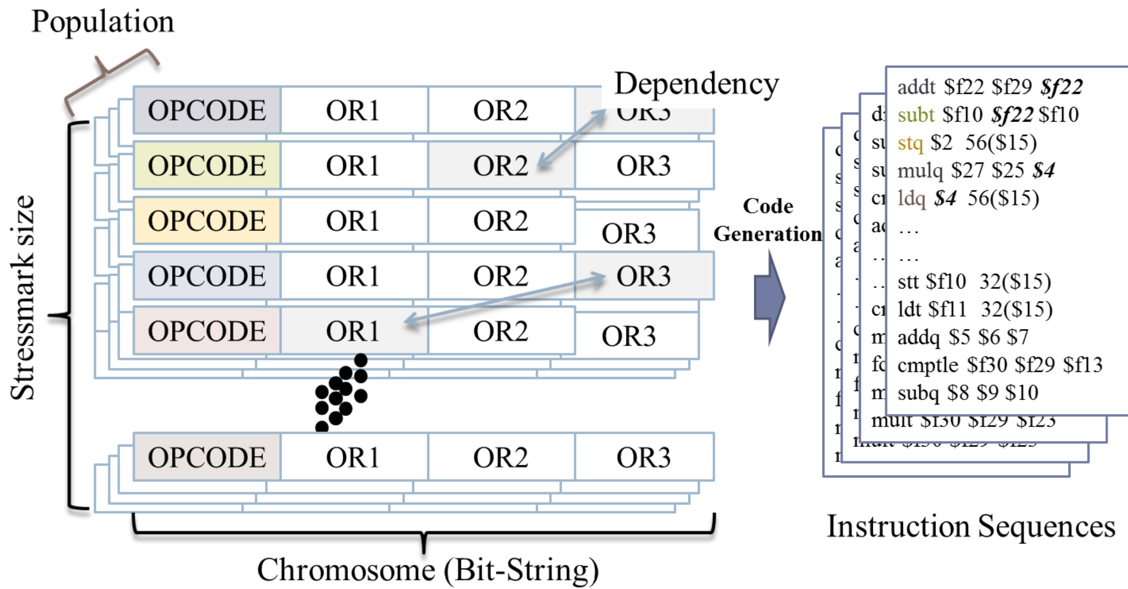


Figure 5.5: Instruction sequence generation for Genetic Algorithm.

5.1.4. Dependency Control and Register Assignment

One of the knobs in the automatic framework is the dependency between

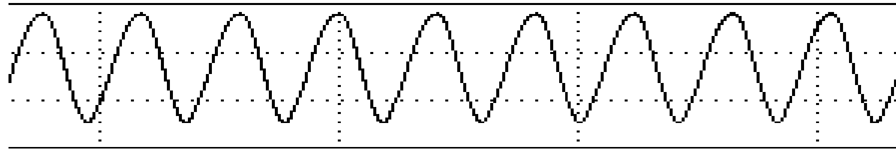
instructions. Data dependencies cause a pipeline stall in a processor until it is resolved. In Joseph et al. [24], dependencies are used to cause low current draw during part of a resonant period, and the same register is assigned to a target register of an instruction and a source register of a following instruction. Prior research [24] chose a floating-point divider instruction, ***divt***, as the only stalling instruction, but this dissertation does not impose this limitation. Any instruction is able to have a dependency with the previous instructions, and its operand registers are assigned according to the dependency

5.1.5. Stressmark Size and Resonant Frequency

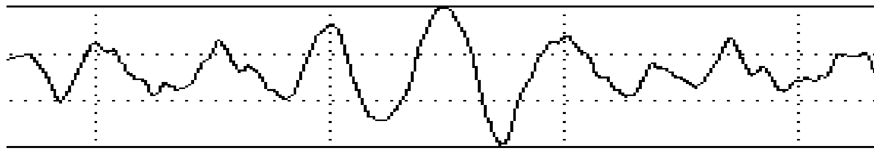
The stressmark size can be selected based on a given resonant frequency (1/resonant period). Figure 5.6 shows the relationship between stressmark size and resonant period.

Stressmark size ≈ resonant period (Figure 5.6.(a)): A short instruction sequence that matches the resonant frequency is repeated to produce a maximum voltage droop.

Stressmark size > resonant period (bottom of Figure 5.6.(b)): A long instruction sequence, which is three to five times longer than a resonant period, can be used to find a maximum voltage droop. It eliminates interferences from the neighbor sequences when instructions have a long latency or a high dependency to each other. However, the possibility to find a maximum voltage droop is reduced because the search space increases exponentially according to the stressmark size.



(a) when stressmark size ≈ resonant period, resonance.



(b) when stressmark size > resonant period, no resonance but single large droop.

Figure 5.6: Stressmark size and resonant period.

5.1.6. Management of Search Space

The stressmark solution space for AUDIT is a function of the number of cycles in the repeated loop (the loop length), the issue width of the processor, and the number of instructions being evaluated for code generation. The combination of loop length, issue width, and the number of instructions can result in a large solution space. The loop length for first droop resonance is determined by the resonance frequency, which can result in a large solution space. For example, a 3GHz processor with a resonance frequency of 50MHz has a loop length of 60 cycles. Assuming a four-wide processor, this results in 240 instruction slots for AUDIT to schedule.

5.1.6.1. Reduction of the number of instruction types

In order to explore the instruction scheduling space efficiently, reduction of the number of instructions can be considered. The search space is almost impossible to be

enumerated with all the different types of opcodes and register combinations. Therefore, this step is necessary before searching the instruction scheduling space to eliminate redundant combinations of instructions and to reduce search time significantly. Each instruction can be categorized into one of a few groups: data type, arithmetic, logic, load/store, bit-level, conditional move, and branch/jump.

Example of Alpha ISA: For data type, both integer and floating-point types are used to utilize the execution units maximally, but only the quad-word (64-bit) type for integer and the double precision type for floating-point are selected to draw large current due to multiple-bit changes. For example, instructions such as add-bytes, add-words, and add-double-words are not used in an instruction sequence. The arithmetic and load/store instructions use different execution units with different latencies, so they are considered individually. For logic, bit-level, conditional move, and branch/jump groups, one instruction can represent other instructions if they use the same execution unit with the same latency such as cmple (compare less than or equal) and cmpeq (compare equal).

5.1.6.2. Reduction of code length scheduled using sub-blocking

To converge in a reasonable time (this dissertation defines reasonable time as a few hours), AUDIT uses a hierarchical generation policy. First, AUDIT separates each member of the population into a high-power (HP) and a low-power (LP) region. Initially, the LP region consists of NOPs. Second, AUDIT breaks the HP region into S replicated sub-blocks of length K. For example, a 24-cycle HP loop can be composed of four ($S = 4$) sub-blocks of length six cycles ($K = 6$). The GA algorithm in AUDIT generates the

instructions for each subsection, and the full stressmark composed of an HP region of S sub-blocks of length K and an LP region of NOPs is evaluated in hardware using the dithering algorithm.

At the end of the AUDIT run for the HP region, a stressmark is generated, which has been synthesized to produce high power for the HP region of the stressmark. This dissertation also evaluated using AUDIT to generate the LP region of the stressmark using long-latency operations with dependencies as proposed in [24]. However, for the system evaluated, a sequence of NOPs produced comparable power values to a sequence of long-latency, dependent operations. NOPs are designed to be very low-power instructions in the experimental processor in this dissertation, so the rest of the evaluation uses NOPs for the LP portion of the stressmark.

The hierarchical implementation was compared to the basic implementation and the results showed that sub-blocking provided faster convergence as well as better results - 19% higher droop in less than five hours compared to a 30-hour run without hierarchical generation.

5.1.7. Adoption of Dithering Algorithm for Guaranteed Alignment

A dithering technique [43] for multiple threads is adopted and implemented for the AUDIT framework. In Chapter 4, it is shown that misalignment among multiple threads may result in a destructive effect in current draw, and that there is natural dithering by OS scheduling. However, to guarantee the alignment of multiple threads in an AUDIT generated di/dt stressmark, a dithering method [43] can be used in the AUDIT framework. The following explains the detail of the dithering method [43].

Figure 5.7 shows a periodic stress pattern with high- and low-power portions of duration H and L cycles, respectively. This waveform meets the requirements of an ideal di/dt-inducing resonant pattern described in Chapter 4. This periodic pattern is repeated for *M* cycles to produce a large resonant droop. The goal of the dithering algorithm is to guarantee that for *C* cores, the stressmarks running on each core align across all *C* cores for at least *M* cycles. Note that a first droop excitation is different in that it requires a low region followed by a high region where the sum of the regions is not necessarily periodic at the resonance frequency.

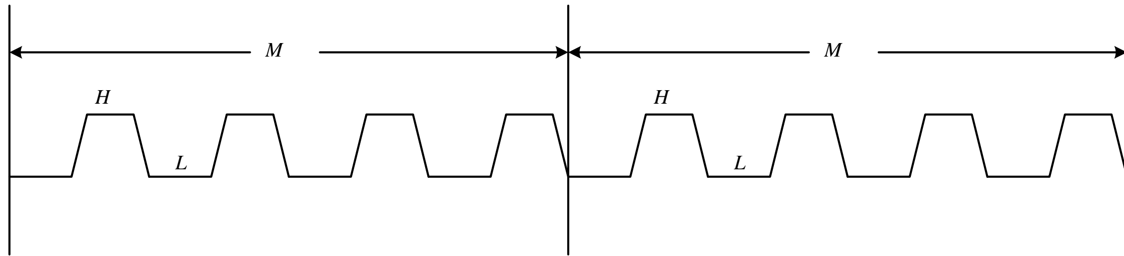


Figure 5.7: Periodic activity waveform for inducing power supply resonance and large voltage droops.

For a high-low sequence of length $H+L$ cycles running on C cores, the misalignment in cores 1 through $C-1$ can be represented as a $C-1$ dimensional variable x = $(x_1, x_2, \ldots, x_{C-1})$, where $x_i \in \{0, 1, \ldots, L+H-1\}$. Core 0 is considered the reference core. The search space for perfect alignment of all cores is therefore $(L+H)^{(C-1)}$ possible alignments. This search space can be fully traversed in $M \times (L+H)^{(C-1)}$ cycles, where M is the number of cycles required to cause and sustain supply droop resonance.

The dithering algorithm uses the following NOP padding procedure to align the threads and achieve resonance in a processor with *C* cores:

- Core 0: Apply no dithering and no extra padding of NOPs. Core 0 simply executes the periodic low-high activity sequence shown in Figure 5.7 repetitively.
- Core c, where $1 \leq c \leq C\text{-}1$: Apply one cycle worth of NOP padding every $M \times (L+H)^{(c-1)}$ cycles.
- The maximum number of cycles to guarantee alignment is $M \times (L+H)^{(C-1)}$.

As long as the number of processors is reasonably small, the alignment algorithm works well. However, the time required for alignment becomes prohibitively large for more than four cores. For example, on a 4-GHz system with $L+H$=24 and M=24×40=960, the time required to align four cores is 3.3 ms, but eight cores require 18.35 minutes. The alignment must be done for each candidate stressmark in each generation of the GA.

To expand dithering to many-core systems, this dissertation uses an approximate algorithm that sets a bound on the maximum misalignment between threads. Assume that the maximum mismatch allowed among the activities of different cores is δ cycles. Then, $L+H$ is chosen such that it is a multiple of $(\delta + 1)$ and $(L+H) \times f$ is close to the resonance frequency of the PDN, where f is the operating frequency of the system.

The search space for alignment of all cores within the maximum allowed mismatch of δ cycles then becomes $[(L+H)/(\delta + 1)]^{(C-1)}$, which can be fully traversed in $M \times [(L+H)/(\delta + 1)]^{(C-1)}$ cycles. The dithering algorithm proceeds as before; however, for core c, where $1 \leq c \leq C\text{-}1$, $(\delta + 1)$ cycles worth of NOP padding is applied every $M \times k^{(c-1)}$ cycles, where $k = (L+H)/(\delta + 1)$. If a δ of 3 is used in the previous example of eight cores, the maximum time required to reach alignment with the approximate algorithm shrinks from 18.35 minutes to 67 ms per candidate stressmark.

5.1.8. Code Generation for Multiple Threads

Figure 5.8 shows the code generation steps for multiple threads. Once a high-low power pattern is generated, it is repeated to make a resonance. Then an initialization part and a dithering part are attached to the beginning and the end of the core pattern, respectively.

1. Prepare a core part – one high-low power pattern
2. Make multiple copies of <1> to increase the intensity of resonance
3. Add a header part that contains initialization codes
4. Make multiple copies of <3> according to the number of threads
5. Attach dithering parts to each thread for alignment

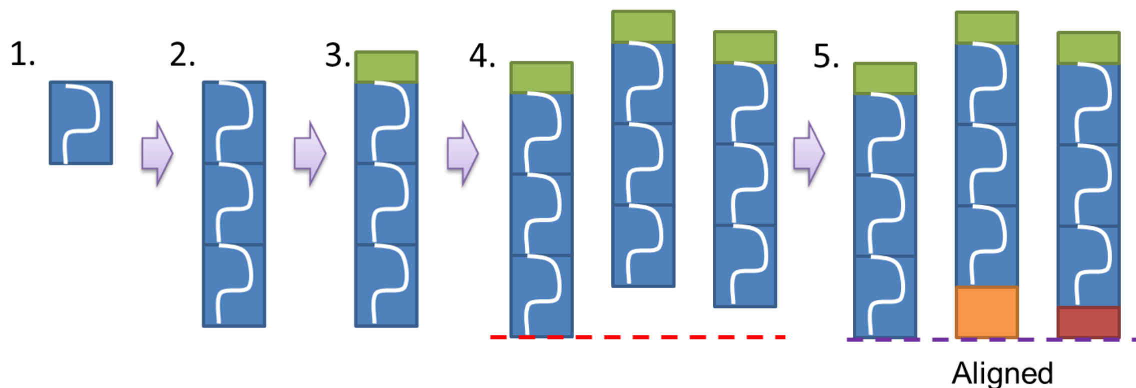


Figure 5.8: Code generation steps for multiple threads.

5.1.9. Sweep for Finding a Resonant Frequency

There are some additional complexities when x86 multi-core systems are used. First, it is observed that data values used for the stressmark have a measureable impact on the final droop values, on the order of 10%. To take data values into account, an

alternating set of values is used, which guarantees maximum toggling between consecutive instructions executing on the same functional unit. Second, the resonance frequencies of the system can vary across different boards or even within the same board if the components of the board change (e.g., using a different processor on the same board, as is done later in this dissertation). Therefore, AUDIT does a sweep for the resonance frequency before attempting to generate a first-order resonant droop.

To determine the resonance frequency, AUDIT constructs a trivial stressmark consisting of a loop of high-power instructions and NOP instructions (Figure 5.9). It varies the number of cycles in the loop to determine the loop length that produces the worst-case droop. The number of cycles in the loop that produces the worst-case droop exercises the resonant frequency of the processor. For example, the number of cycles in the loop is increased by 8 cycles from 8 to 56 cycles. The plot in Figure 5.9 shows that when the total loop length is 32 the largest maximum voltage droop is reached. To make a resonant droop stressmark, the total loop length of 32 is selected.

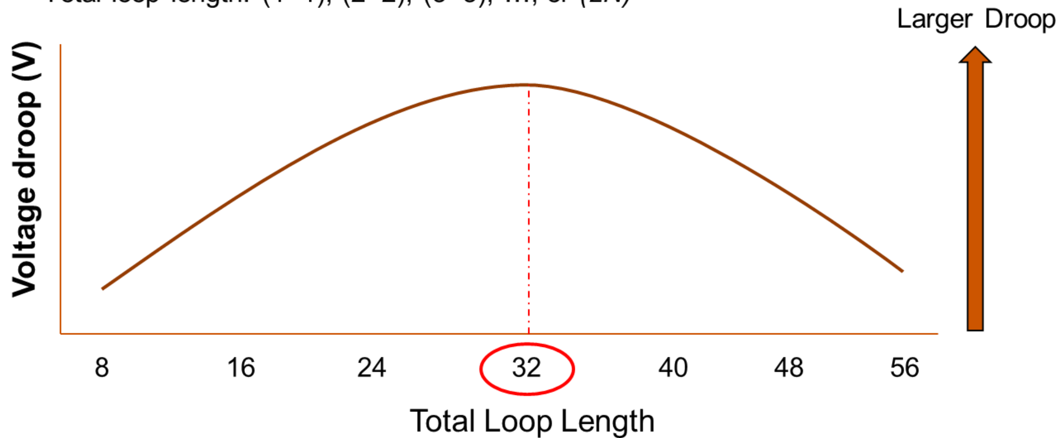


Figure 5.9: Frequency sweep to find resonance frequency. A loop length of 32 hits the resonant frequency and causes the largest voltage droop

5.2. Experimental Results using AUDIT Framework

5.2.1. Results of AUDIT Simulation Path

This section shows the results when using the simulation path. The simulators and their configurations and power distribution networks are carefully selected from the previous studies. As a cycle-accurate power simulator, Wattch [7] that includes SimpleScalar is modified and used. The current traces extracted from Wattch are fed to HSPICE as a current sink in a PDN circuit. The nominal supply voltage is set to 1.0V.

To apply the Genetic Algorithm to the simulation environments in this dissertation, GAUL [13] is used, which provides an open source utility library for

Genetic Algorithms including population creation, evolution, and evaluation. Population, crossover rate, and mutation rate are set to 200, 0.8, and 0.2, respectively, using the library default values. The number of chromosomes, i.e., the stressmark size, is set to 30 for **PDN1**, **PDN2**, and **PDN4** and to 50 for **PDN3**, considering the ratio of CPU clock frequency and resonant frequency.

Three different architectures are configured to see the effectiveness and the architecture dependency of the di/dt stressmark generation method in this dissertation (Table 5.1). The base architecture configuration, **Arch1** shown in Table 5.1, is an 8-wide microprocessor with 3 GHz clock speed, based on the Pentium 4, similar to the configuration in Joseph et al. [24]. For the second architecture, **Arch2**, the number of memory ports is decreased from 4 to 2 in order to reduce memory accesses, and other parameters were also adjusted to a 4-wide microprocessor. The last configuration, **Arch3**, is nearly the same as **Arch1**, but the latency of a key component, ***fdiv*** unit, is increased from 12 to 18 cycles to see the architecture dependency of the di/dt generation method in this dissertation.

Then, the five different power delivery network (PDN) circuits are taken from the previous studies [9][15][47][61]. **PDN1** [47] is simple, but shows mid-frequency behavior which dominates the PDN's characteristic. **PDN2** [15] is an implementation of the Pentium 4's PDN. **PDN3** [9] is also for Pentium 4, but has different resonant frequency, current swing, and number of RLC stages from **PDN2**. **PDN4-A** and **PDN4-B** [61] are the same circuits with different decoupling capacitance values.

Parameter	**Arch1**	**Arch2**	**Arch3**
CPU Clock	3 GHz	3 GHz	(1) Latency of ***fdiv*** is changed from 12 to 18. (2) Other parameters are the same as **Arch1**
Fetch/Decode/Issue	8 / 8 / 8 instr.	4 / 4 / 4 instr.	
EXU	8 alu, 2 mul/div, 4 falu, 2 fmul/fdiv, 4 mem-port	4 alu, 2 mul/div, 2 falu, 2 fmul/fdiv, 2 mem-port	
RUU / LSQ	128 / 64	128 / 64	
Branch Predictor	Combined, 64Kb	Combined, 32Kb	
BTB	1K entries	512K entries	
L1 I/D-Cache	64KB, 2-way	32KB, 2-way	
L2 Cache	2MB, 8-way	1MB, 8-way	

Table 5.1: Base architecture configuration for SimpleScalar

	PDN1 [47]	PDN2 [15]	PDN3 [9]	PDN4-A [61]	PDN4-B [61]
Resonant Frequency	100MHz	100MHz	68MHz	150MHz	200MHz
Current Swing	6-50A	3-20A	2-12A	5-16A	5-16A
#of RLC Stages	1	4	5	2	2

Table 5.2: Five different PDNs for circuit simulation

To compare the effectiveness of the di/dt stressmark in this dissertation to that of other methods, this dissertation runs the SPEC CPU2006 suite with 100 million

instructions, and programs the hand-coded assembly code in [24]. The hand-coded di/dt stressmark consists of two parts; one is for low current draw, and the other is for high current draw. The low current draw part is implemented with the divider instruction, ***divt***, which has a fixed, long latency. The high current draw part uses a store instruction, ***stq***, which store data to main memory through L1 and L2 caches. This dissertation finds the best maximum voltage droop by increasing the number of the ***stq*** instruction from 0 to 200 under the given architecture and PDN configurations. Effort is made to create the best possible hand-coded baseline stressmark for comparison.

Table 5.3 compares the maximum voltage droop in milli-Volts for SPEC CPU2006, the hand-coded stressmarks, and the AUDIT di/dt stressmarks. A larger number means a larger maximum voltage droop, and only the worst voltage droop is shown among the 22 SPEC benchmarks. Overall, the AUDIT di/dt stressmarks always invokes larger maximum voltage droops than the other two methods. For **Arch1**, compared to SPEC CPU2006 and the hand-coded stressmark, 35.7% and 15.7% average increases in voltage droop are achieved by AUDIT di/dt stressmark for the different PDNs, respectively. In **Arch2**, architecture difference between **Arch1** and **Arch2** affects the performance of the di/dt stressmark, but AUDIT di/dt stressmark is less architecture-dependent because the hand-coded di/dt stressmark depends heavily on the number of memory ports due to the store instruction. Considering **Arch1** and **Arch3**, it is shown that the hand-coded di/dt stressmark significantly depended on the specific instruction, ***divt***, executed in the ***fdiv*** unit whose latency is changed from 12 to 18 cycles. In contrast, the automated di/dt stressmark and SPEC benchmarks for **Arch3** make a similar range of voltage droops as **Arch1** regardless of the execution cycle change of the divider unit. This can also reveal that the automated di/dt stressmark generation technique in this

dissertation is architecture-independent.

Config.	PDN	SPEC CPU2006 (worst case) (mV)	Hand-Coded Droop (mV)	Automated Stressmark Droop (mV)	Improvement (Auto. vs. SPEC/ Auto. vs. Hand.)
Arch1	PDN1	65.3	75.8	78.8	20.7% / 4.0%
	PDN2	111.9	112.9	121.5	8.6% / 7.6%
	PDN3	69.2	123.9	134.8	94.8% / 8.8%
	PDN4-A	101.1	107.6	137.5	36.0% / 27.8%
	PDN4-B	140.4	151.2	189.6	35.0% / 25.4%
Arch2	PDN1	26.4	29.0	34.9	32.2% / 20.3%
	PDN2	53.8	55.8	82.2	52.8% / 47.3%
	PDN3	41.8	45.8	60.5	44.7% / 32.1%
	PDN4-A	44.0	39.1	56.8	29.1% / 45.3%
	PDN4-B	53.7	46.5	82.4	53.4% / 77.2%
Arch3	PDN1	65.3	39.3	73.9	13% / 88%
	PDN2	110.9	62.8	130.2	17% / 107%
	PDN3	69.2	113.9	143.5	107% / 26%
	PDN4-A	101.7	80.5	153.0	50% / 90%
	PDN4-B	139.8	75.4	191.6	37% / 154%
Average (Overall)		79.6	77.3	111.4	40% / 44%

Table 5.3: Maximum voltage droops of SPEC CPU2006, hand-coded [24], and automatic di/dt stressmarks. Supply voltage is 1V (Vnom=1V).

Figure 5.10 depicts the current waveform in different generations of the Genetic Algorithm. Figure 5.10 (top) is one of the best in the first generation, *G1*. It seems to be periodic, but its shape is irregular and similar to **sawm** of Figure 2.7. At the tenth generation, *G10*, the current waveform (Figure 5.10 (middle)) looks a mix between **sawm** and **rectangular** in Figure 2.7. The last generation, *G20*, has the current waveform (Figure 5.10 (bottom)) which caused the maximum voltage droop in **Arch2-PDN2**. Its current shape is now similar to rectangular, which induces the largest voltage droop for the given current swing.

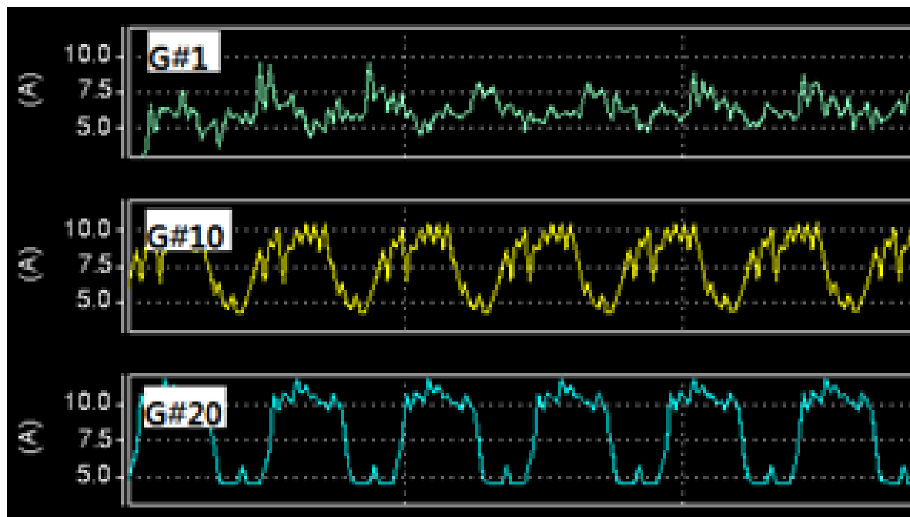
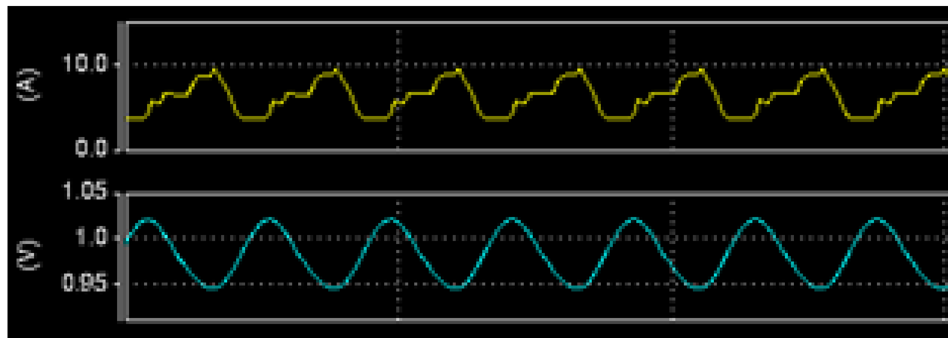


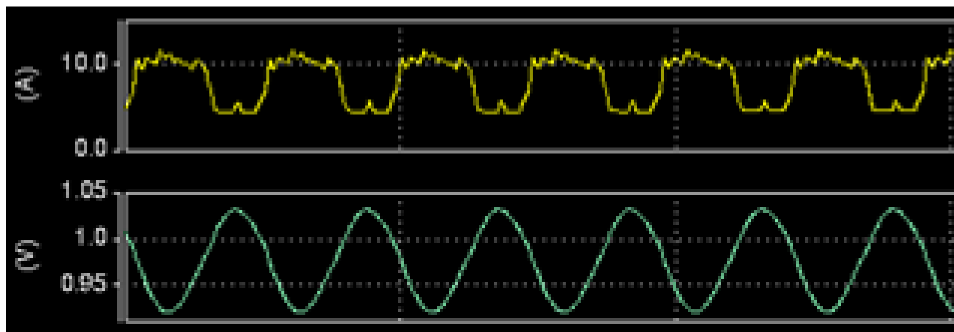
Figure 5.10: Current waveform according to generation number.

Figure 5.11 shows the current and the corresponding voltage waveform of the hand-coded and the automated di/dt stressmarks. In **Arch2-PDN2**, the maximum voltage droops are 55.8 mV and 88.2 mV for the hand-coded and the automated di/dt stressmarks, respectively. The difference in maximum voltage droop comes from the shape of the current waveform. The current waveform of the hand-coded stressmark is

similar to **sawr** of Figure 2.7, which is the worst among the waveform types. In contrast, the automated di/dt stressmark in this dissertation is successful in generating the rectangular shape of current draw, which is the most effective to induce the di/dt effect.



(a) hand-coded di/dt stressmark



(b) automated di/dt stressmark

Figure 5.11: Current and voltage waveform of hand-coded and automated di/dt stressmark in Arch2-PDN2.

Regarding the runtime for each PDN, the di/dt stressmark generation is 10 times faster than the typical benchmark suite is. A di/dt stressmark is generated within 3 hours with the parameters described above, and the SPEC CPU2006 runs take more than 32 hours on the same machine. Since HSPICE simulation dominates the whole runtime, the

same runtime relationship is expected between SPEC runs and the di/dt stressmark generation, even if the complexity of HSPICE netlist of a PDN increases.

5.2.2. Results of AUDIT Hardware Measurement Path

This section covers the results obtained for multi-core stressmark generation. This dissertation compares and analyzes standard benchmarks, existing stressmarks, and AUDIT-generated stressmarks. For each benchmark (stressmark), this dissertation presents its maximum voltage droop and analyze the processor's ability to operate under degraded voltage conditions. This dissertation also presents results showing AUDIT's ability to adapt to microarchitectural and architectural changes.

5.2.2.1. Experimental setup

To evaluate AUDIT, recent multi-core x86-64 processors are targeted due to their widespread use. The primary processor for the evaluation of AUDIT is the AMD Orochi processor, and, in later experiments, it is replaced with an older-generation 45-nm AMD Phenom™ II X4 Model 925 processor to showcase AUDIT's ability to adapt to different systems and requirements.

AUDIT's code generation methodology is able to utilize all x86 instruction types, including integer, floating-point, and SIMD. General-purpose registers and 64-bit and 128-bit media registers are used for source and destination operands. Assembly code instructions are generated in NASM format and are compiled with NASM 2.09.08 [63]. SPEC CPU2006 benchmarks and stressmarks run on Windows® 7 OS, and PARSEC [5] benchmarks run on Red Hat Enterprise 6 Linux OS.

5.2.2.2. Voltage droop analysis

Figure 5.12 shows the maximum droop measured from running SPEC CPU2006 benchmarks, PARSEC multi-threaded benchmarks, and a set of existing and AUDIT-generated stressmarks in configurations of one-, two-, four-, and eight-thread runs (1T, 2T, 4T, and 8T). Benchmarks *zeusmp* and *swaptions* are selected as the worst case in each standard benchmark suite. Unfortunately, the *dithering* methodology (Section 5.1.7) is not easily applicable to SPEC CPU2006 benchmarks or the PARSEC suite because they do not consist of a regular, repeatable loop that can be shifted to produce alignment between the threads. Although the lack of *dithering* for SPEC CPU2006 results in a smaller droop than is theoretically possible with ideal alignment [56], it also reflects the reality of multi-processor execution in which the natural misalignment between threads may counteract some worst-case stress generating behavior. The multi-threaded AUDIT stressmarks (*A-Ex* and *A-Res*) and the hand-generated resonant stressmark *SM-Res* use the *dithering* methodology described in Section 5.1.7 to align the threads for a worst-case voltage droop. The 2T and 4T configurations use the exact algorithm, and the 8T configuration uses the approximate algorithm with a δ of 3.

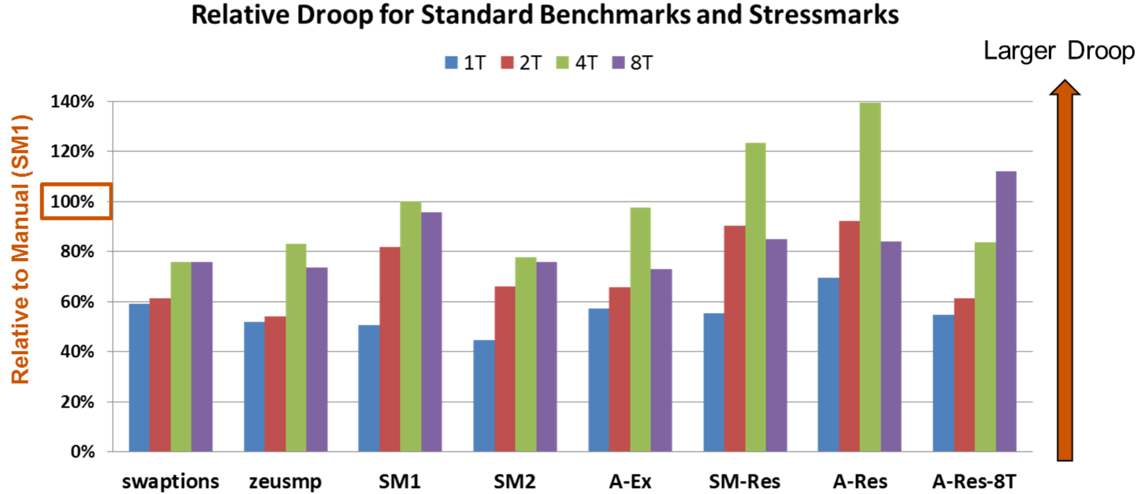


Figure 5.12: Hardware measurements of droop (relative to 4T *SM1*) for SPEC CPU2006, PARSEC, and stressmarks.

Figure 5.12 shows the results for various stressmarks, either manually collected or hand-generated (*SM1*, *SM2*, and *SM-Res*) or automatically generated by AUDIT (*A-Ex* and *A-Res*). *A-Ex* is a first-droop excitation stressmark, and *A-Res* is a first-droop resonant stressmark. *SM1* and *SM2* contain both single-droop and resonant excitations, and *SM-Res* is a hand-generated resonant stressmark. The manual stressmarks are the result either of past di/dt issues or a non-trivial design effort (on the order of a week per stressmark) from a highly skilled engineer with detailed knowledge of the pipeline architecture. The goal of AUDIT is to generate similar or better stressmarks without detailed knowledge of the pipeline in question.

To produce the *A-Ex* and *A-Res* stressmarks in Figure 5.12, AUDIT was instructed to generate a homogeneous stressmark with four identical threads, one assigned to each module. For the resonant stressmark, a high-power sub-block of length six cycles is used and repeated as many times as necessary to produce the high-power

region. The low-power region of the stressmark, for the reasons noted in Section 5.1.6.2, consists of NOPs. The stressmark generation takes less than five hours to complete without human intervention.

With the exception of *SM2*, all stressmarks produce significantly greater droops than the standard benchmarks. As will be shown in Section 5.2.2.4, *SM2* is still a viable stressmark because it exercises the sensitive paths on the processor. The two resonant stressmarks (*SM-Res* and *A-Res*) produce significantly larger droops than all other stressmarks for the reasons described in Section 2.3. Both AUDIT-generated stressmarks (*A-Ex* and *A-Res*) produce droops that are either comparable or greater than that of the existing stressmarks. This highlights AUDIT's ability to produce results that are comparable to well-engineered stressmarks that require significantly more effort and knowledge to generate.

The stressmarks produce larger droops for the 4T case than for the 8T case. All stressmarks contain some amount of floating-point instructions, and the floating point unit (FPU) is shared between the two threads in each Bulldozer module in the 8T runs. This results in interference between the threads; this shifts the loop lengths, making it difficult to align the first droop excitation across the threads or to oscillate at the resonant frequency. The same interference may not exist in the standard benchmarks depending on the density of floating-point operations and how the threads align.

The *A-Ex* and *A-Res* stressmarks are generated using four homogeneous threads assigned one per module. Hence, the GA in AUDIT is not trained to deal with the shared FPU in the 8T run. To test the hypothesis, AUDIT was run again to use eight homogeneous threads, with two threads per module, to generate a new stressmark (*A-Res-8T*). The resulting data is shown in Figure 5.12. The 8T results for *A-Res-8T* are

significantly better than the 8T results for *A-Res* or *SM-Res*. However, the 1T, 2T, and 4T results suffer for the same reason that the 8T run suffers for the other stressmarks -- because the characteristics assumed for the stressmark generation are not valid in some of the multi-threaded configurations. These results show that (1) system characteristics (such as shared resources) must be considered when generating stressmarks, (2) one type of stressmark may not apply to all configurations of a multi-core system, and (3) AUDIT is robust and flexible enough to find patterns that can exercise the characteristics of the system being evaluated with minimal manual intervention.

5.2.2.3. Voltage droop probability

Figure 5.12 shows the worst-case droop for the benchmarks and stressmarks. However, it does not show how often the droop occurs. The more frequently a large voltage droop occurs, the more likely it is to result in a catastrophic failure. Not only does first droop resonance produce larger droops than first droop excitation (see Figure 2.5), it also produces more such events.

In Figure 5.13, the hardware measurement tools in this dissertation are used to produce a histogram of voltage droops for *zeusmp*, *SM1*, and *A-Res*. Each plot contains 8 million samples. The *x* axis shows the measured *Vdd* and the *x* axis range is the same for all figures. The *y* axis shows the number of samples for the given *Vdd*. Values to the left (right) of center indicate voltage droops (overshoots).

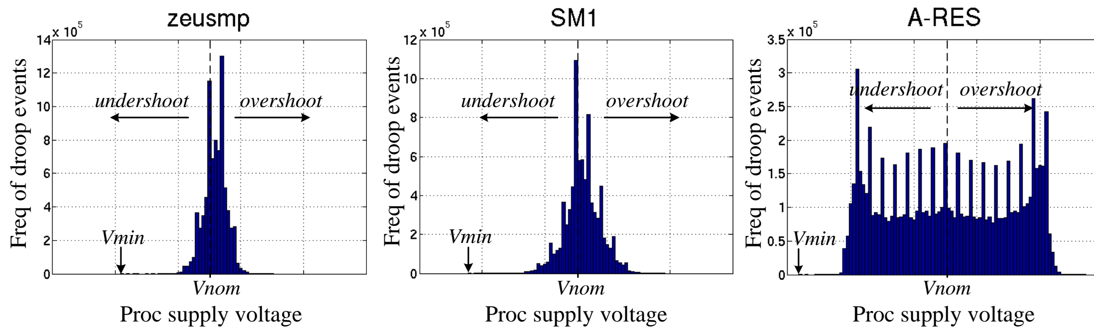


Figure 5.13: Frequency of droop events.

The *zeusmp* benchmark has the least variation in voltage, as expected from the results in Figure 5.12. Stressmark *SM1* has a larger range of measured *Vdd*, yet the largest number of samples is centered at the nominal *Vdd* with a sharp reduction for lower voltages. There are spikes along the way, most likely due to code regions with resonant behavior, but the application has a long tail for both droops and overshoots. The resonance stressmark has the opposite characteristic with the highest number of events occurring near the worst-case droop values. Both stressmarks have a tail of low-probability droop events, but what dictates the failure point of these benchmarks is the higher-probability droop events near the tail. With hardware measurement, these characteristics can be evaluated across the entire run of the program, which is not possible in simulation. The next section evaluates how these droop characteristics translate into failure points for each application.

5.2.2.4. Voltage droop vs. voltage at failure

The size of the maximum voltage droop is an indirect indicator of the voltage

operating margin of the program. The ultimate test is to determine the point at which failure occurs for each configuration. In the next experiment, running 4T configurations for two standard benchmarks with the largest droop (*swaptions* and *zeusmp*) and the stressmarks, the operating voltage is reduced in steps of 12.5 mV until failure occurs. The higher the voltage at failure, the better the program is at stressing the system.

Table 5.4 shows the results relative to *A-Res*, which fails at the highest voltage (V_F). The other resonant stressmark, *SM-Res*, fails at a value 12 mV lower. The next to fail are the other stressmarks, with zeusmp and swaptions failing last as *Vdd* is reduced.

As discussed earlier, the largest droops in the standard benchmarks are the result of a first droop excitation that tapers off quickly, as shown in the left side of Figure 4.6. Hence, they may or may not cause system failure depending on whether the droop occurs when critical paths are being exercised. *A-Ex* also generates a first droop excitation, but it is large enough to cause a failure at higher voltages. *SM1* and *SM2* have both first droop excitation and first droop resonance, and they fail at a higher voltage than the standard benchmarks. This is expected for *SM1* due to its large droop. *SM2*, however, has a droop that is comparable to the standard benchmarks yet is more sensitive to the voltage levels. This is because *SM2*, unlike the benchmarks, is designed to exercise sensitive paths in the architecture.

What these results show is that the voltage droop is one indicator of potential failure, but not the only one. This insight would be difficult to gather from a cycle-accurate simulator that does not detect droop-induced system failures, and benchmarks such as *SM2* would be discarded as potential stressmarks.

As currently implemented, AUDIT's cost function for selecting successful populations is based on the measured droop in the system. However, it is trivial to adjust

the cost function to reward the use of certain types of instructions that exercise critical paths if they are known. The key is that AUDIT is agile enough to manage these changes with little effort.

A-Res	SM-Res	SM1	A-Ex	SM2	zeusmp	swaptions
VF	VF – 12 mV	VF – 62 mV	VF – 75 mV	VF – 87 mV	VF – 125 mV	VF – 125 mV

Table 5.4: Voltage at failure relative to *A-Res* 4T failure point.

5.2.2.5. AUDIT loop analysis

To determine how AUDIT is able to produce large droops, the main loop of the resonant stressmarks, *SM-Res* and *A-Res*, was analyzed. *SM-Res* is hand-designed and regular in using floating-point and SIMD instructions during the high-power phase of the loop. *A-Res* uses a combination of integer and floating-point operations and high- and low-power instructions, including some NOPs in the high-power phase.

By mixing integer and floating-point operations, it is able to exercise multiple schedulers and execution clusters in the pipeline. What is more difficult to assess is why sprinkling NOPs in the code increases the droop.

To further understand the effect of the NOPs, the NOPs in the high-power region were replaced with independent, integer ADD operations and the resulting droop was measured. If the pipeline flow remains the same, the ADDs should produce a higher droop than the NOPs since they are a higher power operation than NOPs. The modified

A-Res stressmark generated a smaller droop (by 40 mV) than the original stressmark. In addition, the frequency of the di/dt pattern shifted lower than the ideal resonant frequency, indicating that the duration of the loop increased due to the inclusion of the ADD operations. Unlike ADDs, NOPs consume fetch and decode resources but do not affect other structures in the pipeline such as the schedulers, physical registers, or result busses. The use of the NOPs enabled the stressmark to attain resonance. Although the pipeline and the modified *A-Res* stressmark are constructed to attain a throughput of four instructions per cycle, resource hazards such as physical register availability, decode width capabilities, token-based scheduling restrictions, and result bus utilization impact the final outcome. AUDIT with its GA-based algorithm was able to construct a stressmark that worked around the pipeline hazards to produce a large droop. AUDIT's ability to accommodate pipeline restrictions is examined further in Section 5.2.2.6.

One valid concern is that AUDIT stressmarks are unrealistic because the droops generated by them are much worse than normal benchmarks or other stressmarks. As noted earlier, instead of using the stressmarks to set voltage margins, they can be used to understand the bounds of the problem and sensitivities of the pipeline. For example, the *A-Res* stressmark shows that it is possible to generate large droops by selecting both the floating-point and integer execution clusters in the pipeline rather than just focusing on the floating-point pipeline. Additionally, as will be shown in Section 5.2.2.6, when one di/dt stress path is blocked through droop mitigation mechanisms, AUDIT can find other high-stress paths in the pipeline. As noted by Patel [45], there are many sensitive paths on cores that can lead to catastrophic failures when the system is stressed by reduced noise margins, and it is imperative that one have the tools necessary to identify these paths.

5.2.2.6. Impact of FP throttling

Floating-point and SIMD instructions are generally the highest-power instructions available in the execution pipeline and they are used extensively in the high-power portion of the stressmark. A number of papers have noted that hardware and software architectural throttling schemes reduce di/dt stresses by limiting the rate of change in the execution of high-power instructions [9][14][16][19][24][40][47][53][56]. This dissertation utilizes a FP throttling scheme that statically limits the maximum number of FP instructions executed in a cycle.

This dissertation measured the droop on some of the stressmarks with FP throttling enabled to determine the maximum droop and maximum voltage at failure. The results are shown in Table 5.5. As before, all droop data are relative to the 4T *SM1* stressmark with FP throttling disabled. FP throttling is highly effective for *A-Res* and *SM-Res*, but less so for SM1. SM1 is composed of multiple high-stress code sequences, and FP throttling does not affect all stress paths in SM1. Although the results vary, the droop and voltage at failure improve with FP throttling. These results show that FP throttling functions as expected by limiting di/dt stresses; however, the results so far do not show whether AUDIT can find another stressmark that can produce a significant droop with FP throttling enabled.

	Stressmark	Relative Droop	Failure Point
No Throttling	SM1	1	VF – 62 mV
	A-Res	1.39	VF
	SM-Res	1.25	VF– 12 mV
FP Throttling	SM1	0.93	VF– 75 mV
	A-Res	0.86	VF– 100 mV
	SM-Res	0.78	VF– 113 mV
	A-Res-Th	0.98	VF– 75 mV

Table 5.5: Impact of FP throttling on relative droop (relative to 4T *SM1*) and failure point (relative to 4T *A-Res*).

AUDIT was used to generate a new stressmark (*A-Res-Th*) to determine if there are other opportunities to generate a large droop in conjunction with FP throttling. The AUDIT stressmark generation was repeated using four threads, but with FP throttling enabled. Table 5.5 shows the droop and failure levels for the new stressmark *A-Res-Th*. AUDIT was able to generate a stressmark that works around the FP throttling restrictions to increase the size of the droop. However, it is not able to match the droops seen without FP throttling because it is now limited to using fewer high-power floating-point and SIMD operations. With FP throttling enabled, *A-Res-Th* exceeds the *SM1* stressmark for droop and matches it for sensitivity to voltage. It also highlights another stress path through the processor for engineers to evaluate.

The results show the experimental FP throttling scheme works well for reducing voltage droops in the system, and AUDIT, in a relatively short time (~5 hours) has identified another path that can still produce significant voltage droops with FP throttling

enabled.

5.2.2.7. AUDIT on a different processor

To present AUDIT's ability to adjust to microarchitecture and system changes, the Bulldozer-based processor in the experimental system was replaced with an older-generation 45-nm AMD Phenom II X4 Model 925. The rest of the board remained unchanged. Each core in the AMD Phenom processor has local L1 and L2 caches, no multi-threading, and less variation between high- and low-power regions because it does not manage power as aggressively as the Bulldozer-based system. New resonant stressmarks for the AMD Phenom processor were generated using AUDIT and the results are shown in Table 5.6. Running SM1 on the older processor was not possible due to incompatible instructions. As with the Bulldozer-based system, AUDIT was able to generate stressmarks that were comparable to or better than hand-tuned stressmarks, highlighting the capabilities of automatic stressmark generation tools such as AUDIT.

	zeusmp	*SM2*	*A-Res*
Relative Droop	0.82	1	1.10
Failure Point	VF – 50 mV	VF	VF

Table 5.6: Droop and failure results for a 45-nm AMD Phenom II processor. Droop and failure point are shown relative to SM2.

5.3. SUMMARY

This chapter presented a framework to automatically generate voltage droop

stressmarks. Standard benchmarks do not produce the same levels of first-order droop as the stressmarks. On the experimented processor, this is true even for benchmarks that have global synchronization resulting from barriers.

There are many different ways to construct a stressmark in a multi-core system depending on what structures and types of configurations one is trying to exercise. Therefore, a stressmark that works well for one configuration (such as *A-Res* for 4T runs) may not produce the best results for other configurations. AUDIT's flexibility and ease of use can be leveraged to develop a suite of stressmarks that can effectively exercise all significant usage scenarios in the system.

The measured droop is not the only indicator of sensitivity to failure. The paths exercised by the stressmark and the number of times the droop event occurs also have an impact on overall program susceptibility. AUDIT is able to match or exceed the droops produced by benchmarks and other stressmarks by exercising a richer set of paths in the pipeline.

Chapter 6: Dynamic Management of Supply Voltage Margin

Voltage guardbands are used to minimize errors due to inductive noise; however, this leads to lower performance operation because the voltage and frequency points are set to deal with voltage droops from a worst case benchmark or stressmark. Even though most applications do not approach the voltage droop caused by the stressmark (see Section 4.1), there is no mechanism to guarantee correct operation outside of the tested range.

In this chapter, a hardware technique, floating point (FP) unit issue throttling, is examined, which provides a reduction in worst case voltage droop. Voltage droop stressmarks utilize the floating point path to generate the worst case voltage droop. By dampening the issue rate in the floating point scheduler, the processor can significantly reduce the maximum voltage droop in the system. This chapter shows the impact of floating point throttling on voltage droop, and translates this reduction in voltage droop to an increase in operating frequency because an additional guardband is no longer required to guard against droops resulting from heavy floating point usage. This chapter then examines the impact of floating point (FP) unit throttling and guardband reduction on the SPEC CPU2006 benchmarks and shows that some benchmarks benefit from the frequency improvements with FP throttling while others suffer due to reduced FP throughput. Finally, this chapter presents two techniques to dynamically determine when to tradeoff FP throughput for reduced voltage margin and increased frequency, and shows performance improvements of up to 15% for CINT2006 benchmarks and up to 8% for CFP2006 benchmarks.

6.1. MANAGEMENT OF VOLTAGE GUARDBAND

Processor designers use a voltage guardband (a.k.a. voltage margins) to guarantee correct operation under worst case conditions. The voltage margins guard against process variations, system power supply variation, and workload induced voltage droops. These margins are set conservatively, and are on the order of 15% to 20% of supply voltage [23]. By guarding against the worst case scenarios, a lot of performance is lost. For instance, according to [56], a 20% voltage margin translates into a 33% frequency loss.

Stressmarks are benchmarks designed to stress a processor in various ways, such as generating the worst case power or the worst case voltage droops. Stressmarks designed to induce large di/dt voltage droops are used to determine the voltage guardband due to workload induced di/dt noise. Analysis presented in Chapter 5 has shown that on x86 processors the high power region typically contains a high number of floating point (FP) or SSE instructions, while the low power region generally contains NOPs. The high power region can consist of other types of instructions, such as instructions from the integer pipeline, but the resulting voltage droop from these instructions is significantly less than the droop from instructions that execute on the FP path of the experimented processor because operations that use FP pipeline dissipate relatively large amounts of power and thus lead to large di/dt fluctuations. Hence, the worst case guardband of the specific x86 system is determined using operations that utilize the FP pipeline. If the workload does not have high FP pipeline utilization, then the system can be run with a lower voltage guardband, which can be translated into a higher operating frequency.

This chapter investigates the use of a hardware-based FP throttling mechanism that limits the maximum issue rate of the FP pipeline. By enabling FP throttling, lower

workload induced voltage droops are guaranteed, which enables the processor to operate with a smaller voltage guardband. The reduced voltage guardband is translated into an increase in operating frequency. The results show a significant performance improvement for non-FP-intensive programs, but some performance loss for FP-intensive programs due to restrictions on FP issue rate. Finally, this chapter presents algorithms to dynamically enable and disable FP throttling and adjust the operating frequency in order to trade off frequency for FP throughput to improve the performance of both FP-intensive and non-FP-intensive programs.

The contributions of this chapter are:

- analysis of an FP throttling mechanism on a state-of-the-art x86 processor,
- quantification of frequency boost, performance, and energy-delay product benefits made possible by FP throttling,
- analysis of the impact of FP throttling with multi-core execution, and
- new algorithms to dynamically manage FP throttling and an analysis of their benefits.

Figure 6.1 shows the general workflow of dynamic voltage guardband management introduced in this paper. Frequency sweeps using various di/dt stressmarks identify voltage margins and critical paths to determine the amount of frequency boost and microarchitecture units throttled. An algorithm, which dynamically controls throttling and boost mechanism, can improve both performance and energy.

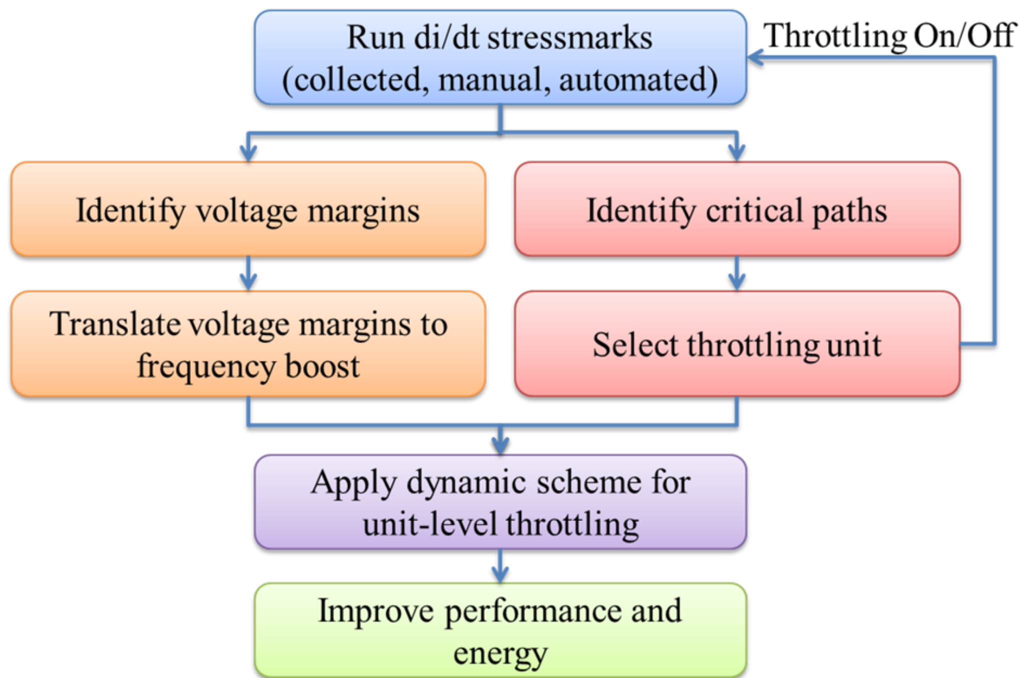


Figure 6.1: Workflow for dynamic voltage guardband management.

6.2. Workload Induced Voltage Margins

This section covers the details of di/dt noise, stressmarks and how stressmarks are used to determine the worst case voltage margin for a processor. This section also discusses the FP throttling mechanism and examines the impact of FP throttling on the worst case voltage droop.

6.2.1. Di/dt Stressmarks

Specialized micro benchmarks known as di/dt stressmarks are used to generate worst case workload induced voltage droops [24][26][27]. The general framework for the di/dt stressmark consists of a high power region followed by a low power region as

shown in Figure 4.2. One high to low transition event is required for 1st droop excitation, while a waveform repeating at the periodicity of the resonance frequency of the system is required to generate a 1st droop resonance stressmark. Figure 2.6 shows scope shots of first droop excitation and first droop resonance, respectively, as seen on the experimental system described in Chapter 3. In general, resonant droops are more likely to induce failures than excitations droops, because they are larger and repeat periodically.

A subset of the stressmarks described in Chapter 5 is used to determine the voltage guardband necessary during normal operation and when FP throttling is enabled. In particular, the *SM1* and *SM-Res* stressmarks are used. *SM1* is a stressmark that contains both 1st droop excitation and 1st droop resonance. Because it is not physically constructed to operate at a particular frequency, it is less sensitive to FP throttling. Stressmark *SM-Res* is a stressmark specifically developed to induce 1st droop resonance, and it is tuned for the baseline operating frequency of 3GHz. As discussed in Chapter 5, 1st droop resonance is highly sensitive to the operating frequency, and *SM-Res* needs to be re-tuned every time operating frequencies change, while *SM1* is more tolerant of frequency changes. *SM1* and *SM-Res* are used to analyze workload induced voltage droops with and without FP throttling enabled. In addition, the AUDIT tool in Chapter 5 is used with FP throttling enabled to generate a new stressmark (*A-FPTh*) that utilizes other processor pipelines to generate a high voltage droop.

6.2.2. Floating-Point Activity

The Bulldozer modules used in the analysis in this dissertation contain separate

integer and floating-point scheduler units [12]. The floating point (FP) unit has a maximum issue rate of four instructions per cycle, and is shared between the module's two cores. Hence, the instructions to the FP unit can come from one or both cores. The floating point scheduler handles all floating point operations (with the exception of floating point loads, which go through the integer scheduler) and all SSE operations. FP and SSE operations are high power operations and are part of the high power region of the stressmark code (Chapter 5).

As noted in Section 6.2.1, a 1st droop stressmark consists of a sequence of high power operations followed by a sequence of low power operations. Given that the FP and SSE operations are among the highest power operations, the high power region of the stressmark contains a large percentage of these operations. If the processor can reduce the rate at which operations go through the FP pipeline, it can also reduce the amount of power consumed in the high power region and the size of the resulting workload induced voltage droop.

Figure 6.2 shows the voltage droop and average power resulting from 1st droop resonant stressmarks that execute and commit four instructions per cycle during the high power region of the stressmark. Each stressmark executes one (*SM-1FP*), two (*SM-2FP*), three (*SM-3FP*), or four (*SM-4FP*) FP pipeline operations per cycle with the remaining operations coming from the integer pipeline. SM-Res was modified to generate the stressmarks: SM-4FP is the same as the SM-Res stressmark.

Both power and voltage droop values are shown relative to the SM-4FP case. For this particular experiment, the second core in the module is inactive and the active core has full use of the FP scheduler unit. Figure 6.2 shows that the droop increases with the issue rate of FP operations, indicating that the FP pipeline issue rate is critical

for inducing the largest workload dependent voltage droop. This is because the power in the high power region is correlated to the number of high power FP ops executed per cycle. The larger the power in the high power region of the stressmark, the larger the difference between the high and low power regions, and the larger the resulting voltage droop.

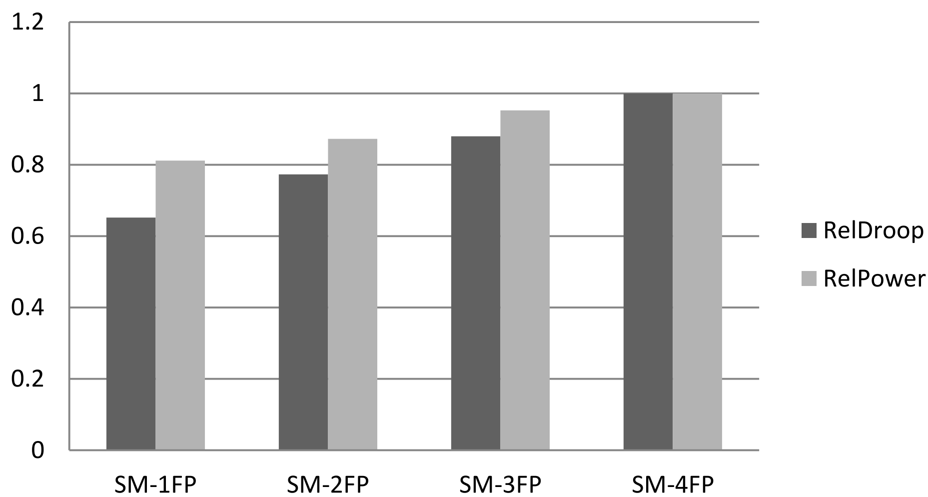


Figure 6.2: Relative voltage droop and power with varying issue rate of FP ops.

6.2.3. Floating-Point (FP) Unit Throttling

As shown in Figure 6.2, the number of floating-point operations executed each cycle has a large impact on the voltage droop. If the maximum number of operations can be limited using the FP pipeline each cycle, the maximum workload induced voltage droop can also be limited. The Bulldozer module has a hardware mechanism, known as FP throttling, to limit the number of FP operations issued per cycle. FP throttling was

configured to restrict the maximum issue width from four to two, meaning that at most two FP pipeline operations can execute per cycle. Furthermore, for some FP operations that are restricted to certain execution pipes, at most one operation can take place on these pipes every four cycles. The FP throttling mechanism only limits the FP issue rate, and not the fetch or decode rate. By enabling this mechanism, one can effectively throttle the maximum voltage droop seen on the system and reduce the voltage guardband for the processor.

With FP throttling enabled, the worst case droop for the stressmarks is reduced by 15% to 35% (Figure 6.2). *SM1* shows the least reduction in droop. *SM1* contains a variety of events that generate 1st droop excitation and 1st droop resonance. Although FP throttling impacts some of these events, there are obviously other events that do not require the full use of the FP pipeline. *SM-Res* shows the largest decrease from FP throttling. This is because *SM-Res* uses the FP pipeline exclusively for the high power region of the stressmark and FP throttling effectively cuts its issue rate by at least one-half or more. Not only does this limit the power during the high power region, but it also changes the execution time of the high power region so that the stressmark no longer hits the resonance frequency of the system. Hence, the large voltage droops resulting from 1st droop resonance are not seen after FP throttling.

As noted in Chapter 5, there may be other paths in the processor pipeline that could be used to generate large voltage droops. In order to test this principle, AUDIT was used to generate a new stressmark (*A-FpTh*) with FP throttling enabled. *A-FpTh* utilizes the integer (INT) pipeline to compensate for the FP pipeline throughput restrictions. This stressmark produces large voltage droops even when FP throttling is enabled. The next section shows how these three stressmarks, *SM1*, *SM-Res*, and *A-*

FPTh, are used to determine the additional frequency boost possible with FP throttling.

6.3. DETERMINING VOLTAGE MARGINS

FP throttling reduces the maximum workload induced voltage droop. This reduced droop can be translated into a performance improvement. In other words, one can run the processor at a higher frequency for the same voltage. However, the translation is not straightforward because the voltage guardband is set based on a number of metrics, and workload induced voltage droop is only one of them. Therefore, it needs to be ensured that one retains the guardband necessary for process and power supply variations for instance, but reduces the guardband necessary for workload induced voltage droops.

To determine the potential frequency boost resulting from FP throttling, the voltage at failure is used as described in Chapter 5. The voltage at failure determines the headroom available between the guaranteed safe point of operation and the point of failure. The voltage at failure is determined by keeping the frequency constant and reducing the supply voltage by steps of 12.5 mV. For example, if a processor at 1.25V and 3.0GHz is guaranteed to run correctly but fails at 1.20V and 3.0GHz, then the voltage headroom between correct execution and failure is 50mV.

To determine the maximum safe frequency of operation when workload induced voltage droop is reduced via FP throttling, one must match the voltage headroom seen when FP throttling is disabled. To do this, the part with FP throttling enabled at higher frequencies is run in 100 MHz increments until the voltage headroom seen with FP throttling enabled and the higher frequency equals the voltage headroom seen with FP

throttling disabled and the guaranteed safe frequency. Using the example above, if a voltage headroom value of 50mV is seen when running at 3.3GHz, then the frequency boost possible with FP throttling enabled is 300MHz.

The methodology described was implemented using the stressmarks SM1, SM-Res, and *A-FpTh* with four threads and FP throttling enabled. As noted in Chapter 5, although the evaluation board used for the experiments in this chapter can run a total of eight threads, running one thread per module (four threads) generates the worst case voltage droop due to the shared resources on a Bulldozer module. An additional complication occurs for resonant stressmarks as the frequency is increased. The resonant stressmarks (*SM-Res*) are tuned to the operating frequency of the part. Hence, retuning the stressmark is needed for every new frequency tested. *SM-Res* was retuned by removing instructions from the high power and low power regions as the frequency increased. With *A-FpTh*, a new AUDIT stressmark was generated for each new frequency.

Table 6.1 shows the frequency boost that can be achieved with each stressmark. As noted earlier, because it is optimized for a throttled FP unit, *A-FpTh* achieves the largest voltage droop with FP throttling and also determines the maximum safe frequency boost available with FP throttling: 400MHz. These results show two insights. First, the resonant stressmark must be tuned to the new frequencies, and second, one cannot blindly rely on existing stressmarks if notable changes to the FP issue rate are made as is done with FP throttling. New stressmarks are needed to deal with the new FP issue rate. The boosted frequency resulting from *A-FpTh* (400MHz) is used to run the benchmarks with FP throttling enabled.

Stressmark	Frequency Boost
SM1	> 600MHz
SM-Res	600MHz
A-FpTh	400MHz

Table 6.1: Frequency boost possible with FP Throttling.

6.4. DYNAMIC FP THROTTLING

The frequency boost possible with FP throttling is beneficial for programs that do not contain a large number of operations that go through the FP pipeline. For FP or SSE intensive programs, however, the IPC loss resulting from reducing the FP pipeline issue rate from four to one or two instructions per cycle (depending on which pipeline they execute on) may offset any performance benefits from increased frequency. As shown in Section 6.5, there is a significant performance loss for some benchmarks with FP throttling. Therefore, one also needs a mechanism to dynamically enable and disable FP throttling. This mechanism enables FP throttling and provides a frequency boost for programs with low FP pipeline usage, and disables FP throttling for programs with high FP pipeline usage.

Ideally, one would like the dynamic FP throttling mechanism to be implemented in hardware so that the decisions can be rapidly made without interrupting the rest of the system. Unfortunately, such a mechanism is not available in current hardware, so this dissertation relied on performance counters and OS control to analyze the algorithms in this chapter. However, the algorithms developed in this chapter can be easily

implemented in future hardware.

The dynamic FP throttling mechanism using performance counters was implemented to determine the rate of operations being issued from the FP pipeline (Table 6.2). The counter is sampled every 10 ms. If the average FP pipeline issue rate exceeds a given threshold, then FP throttling is disabled. Otherwise, FP throttling is enabled and the frequency is boosted. This simple mechanism makes a number of assumptions. First, it assumes that past behavior is indicative of future behavior. Second, it tracks FP issue rate at a fairly coarse level, assuming that FP activity does not change rapidly. Third, it does not take into consideration that average values do not show the distribution of events and can hide large variances in the FP issue rate.

This algorithm is chosen because it is simple to implement and is not too intrusive at a sampling rate of 10ms. There is an overhead with sampling performance counters too frequently and with enabling and disabling throttling. Therefore, although programs might show fine-grained regions of high and low FP pipeline usage, the mechanism in this work may not take advantage of them because of the overhead required to detect and manage these events. In general, it is being tried to find reasonably large regions or phases of the program, and the algorithm chosen works well given the requirements in this work. The algorithm could be expanded by collecting more details such as determining the rate of change in the number of FP pipeline instructions issued, or collecting data based on architectural events such as instruction stream misses or ITLB misses which indicate changes in program behavior.

Algorithm 1: Dynamic scheme (Dyn) using FP_IPC

```
Given: FP_IPC_THRESHOLD, CONSECUTIVE,
            DECISION_INTERVAL, P0, and PB
 1:  while TRUE  do
 2:      for each core, C_i  do
 3:          get fp_ipc;  /* FP-IPC */
 4:          if (fp_ipc > FP_ IPC_THRESHOLD)  then
 5:              if  FP_throttling.enabled    then
 6:                  Frequency.adjust(P0);
 7:                  FP_throttling.Off()
 8:              end if
 9:              fp_ipc_consecutive = 0;
10:          else /* fp_ipc <= FP_ IPC_THRESHOLD */
11:              if  !FP_throttling.enabled    then
12:                  if (fp_ipc_consecutive > CONSECUTIVE)  then
13:                      FP_throttling.On();
14:                      Frequency.adjust(PB);
15:                  end if
16:              end if
17:              fp_ipc_consecutive = fp_ipc_consecutive + 1;
18:          end if
19:      end for
20:      Sleep.time(DECISION_INTERVAL);
21:  end while
```

Table 6.2: Algorithm for dynamic voltage guardband manegement using FP-IPC

6.5. Experimental Results

6.5.1. Experimental Setup

The AMD Orochi [12] processor with four Bulldozer modules is used for evaluating the dynamic management of supply voltage margins. SPEC CPU2006 benchmarks are used for this analysis. These benchmarks vary substantially in their use of FP and SSE instructions. SPEC CPU2006 benchmarks are compiled and highly

optimized with gcc-4.6.2 and gfortran-4.6.2, which support the latest SSE instructions in Bulldozer. All the benchmarks and the stressmarks run on RedHat Enterprise Linux 6 OS. All SPEC CPU2006 benchmarks were run with and without FP throttling. The baseline case disables FP throttling and uses a default 3GHz frequency. With FP throttling enabled, a frequency boost of 400MHz is assumed, based on the analysis in Section 6.3.

As noted earlier, boosting frequency with FP throttling can help some benchmarks but hurt others. For benchmarks that do not utilize the FP pipe, a maximum performance benefit of approximately 13.3% (3.4GHz versus 3.0GHz) is expected if the application is highly CPU-bound, i.e., highly sensitive to core performance. On the other hand, if the application is memory bound, then it is expected to see less improvement from a frequency boost. For FP and/or SSE intensive applications, FP throttling with frequency boosting may help or hurt performance. Overall performance will degrade if the IPC loss resulting from reduced FP bandwidth is greater than the performance benefit due to the increased frequency.

6.5.2. Performance with FP Throttling

Figure 6.3 shows the performance results with FP throttling enabled. All results were gathered while running one thread. Performance is measured using runtime and is shown relative to the baseline case (*Base-3G*) with no FP throttling and no frequency boost (frequency of 3.0GHz). For *St-FpThr-3G*, FP throttling is enabled but the processor is still operating at 3GHz to show the impact of FP throttling on IPC. *St-FPThr-3.4G* represents results with FP throttling enabled and a 400MHz frequency boost.

Both *St-FPThr-3G* and *St-FpThr-3.4G* use a static scheme where FP throttling is always enabled. *Dyn-FpThr-3.4G* is a dynamic scheme that decides on the fly whether or not to enable FP throttling and frequency boosting. The dynamic scheme recognizes which applications benefit from FP throttling in addition to adjusting for phase behavior within an application. The dynamic scheme is analyzed in more detail in Section 6.5.4.

FP throttling does not impact CINT2006 benchmarks with the exception of a slight performance loss for *omnetpp* and *xalancbmk*. However, the performance loss on CFP2006 benchmarks is significant, up to 38% for calculix. This data is not surprising given the nature of the benchmarks. With the frequency boost possible with FP throttling, a performance increase of up to 15% is seen in the CINT2006 benchmarks. The FP benchmarks also improve relative to the case of FP throttling with no frequency boosting (*St-FpThr-3G*) but many of them suffer relative to the baseline case.

There are some interesting points to note about the results. First, the performance improvement on some CINT2006 benchmarks (*perl*, *bzip2*, *sjeng*) is greater than the expected maximum of ~13.3% given the frequency boost. The Bulldozer module contains complex, out-of-order cores, and the increase in core frequency changes pipeline behavior. The boosted frequency is for the Bulldozer cores, only, but the NorthBridge and memory continue to operate at the same frequency. Hence not only is one increasing the operating frequency, but also creating small changes in application IPC. It is difficult to predict whether IPC will increase or decrease with a frequency boost, but for some benchmarks, there are small improvements in IPC. IPC was measured when running the CINT2006 benchmarks, and small improvements in IPC were noticed for the cases where the performance improvement exceeds 13.3%.

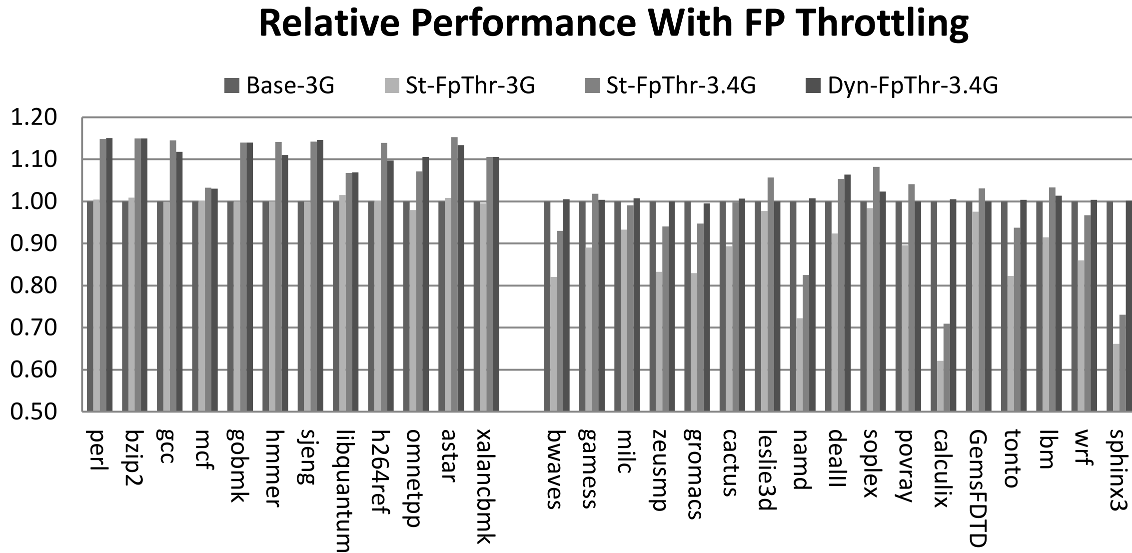


Figure 6.3: Relative performance impact of FP throttling with and without frequency boost relative to *Base-3G*.

The second interesting aspect of the results is that some CFP2006 benchmarks improve with FP throttling and frequency boosting by up to 8% (*soplex*). *St-FPThr-3.4G* generally increases performance relative to the baseline when the performance loss resulting from FP throttling is not significant. As noted earlier, if the performance improvement resulting from a higher frequency is greater than the IPC loss from FP throttling, performance will improve. Figure 6.4 shows the relative IPC and performance for CFP2006 benchmarks. The data is for *St-FpThr-3.4G* relative to the *Base-3G* case. When the application is not significantly impacted by FP throttling (relative IPC close to 1), then the resulting performance improves with FP throttling.

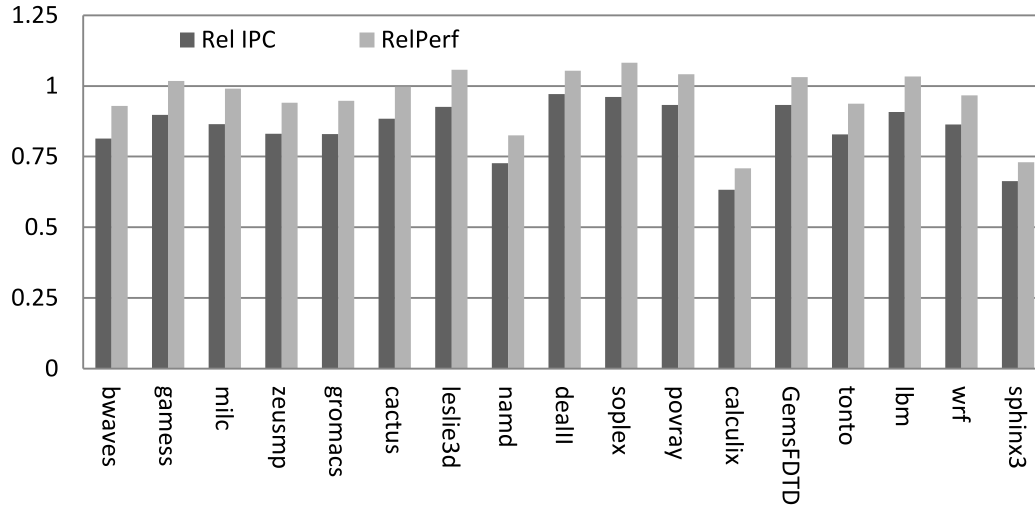


Figure 6.4: IPC and performance for *St-FpThr-3.4G* relative to *Base-3G*.

6.5.3. Energy Efficiency

One of the benefits of FP throttling is that frequency can be boosted without a subsequent increase in voltage. Both the 3GHz and 3.4GHz frequencies use the same voltage. Hence, power increases linearly with frequency.

Figure 6.5 shows the energy-delay product (E*D) for all applications. The results are shown relative to the baseline case (*Base-3G*), and values lower than 1.0 means better efficiency. Dynamic power dissipation in the core is expected to scale linearly with frequency. However, the leakage power should not vary significantly

since voltage remains constant between the baseline case and the boosted case with FP throttling enabled. The only reason leakage power may increase is due to the increase in temperature resulting from higher dynamic power dissipation.

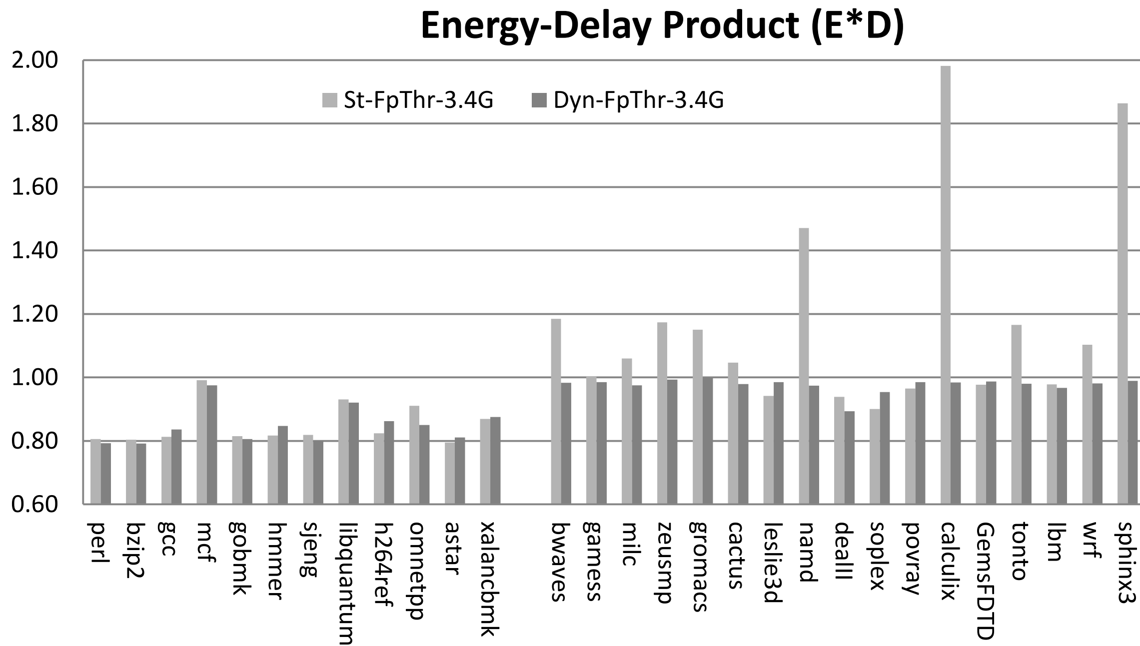


Figure 6.5: Energy-delay product (E*D) with static and dynamic schemes. The values are relative to *Base-3G*.

The results show that energy-delay product (E*D) improves for all CINT2006 benchmarks. This is expected given that the performance improvements for these benchmarks more than compensate for the linear increase in power. The results for CFP2006 benchmarks are more mixed. There is a significant increase in the energy-delay product (E*D) for benchmarks with large performance losses (*namd*, *calculix*,

sphinx3). However, benchmarks that improved in performance with FP throttling and a 400MHz boost also show a small decrease in the E*D product. The results for *Dyn-FpThr-3.4G* will be discussed further in Section 6.5.4.

6.5.4. Dynamic Scheme

The data in Figure 6.3 clearly shows the need for a dynamic scheme for determining when to tradeoff FP throughput for a frequency increase. In this section, the methodology and metrics used are discussed to implement a dynamic FP throttling scheme.

Figure 6.3 and Figure 6.5 also show results for the dynamic scheme (*Dyn-FpThr-3.4G*) implemented using performance counters to measure the FP IPC over a sampling period. The scheme is simple – if FP throttling is currently enabled and FP IPC is greater than a pre-determined threshold, disable FP throttling and run at 3GHz; if FP throttling is disabled and the FP IPC is less than the predetermined threshold for three sampling periods in a row, then enable FP throttling and boost the frequency to 3.4GHz.

To determine the optimal threshold at which to disable FP throttling, the average number of instructions per cycle executed in the FP pipeline (FP-IPC) for the base case (*Base-3G*) was measured. FP-IPC represents any operations issued to the execution units in the FP pipeline which includes not just traditional FP operations but also, among others, SSE operations. The FP-IPC results for all SPEC CPU2006 benchmarks are shown in Figure 6.6. As expected, the FP-IPC rate in the CFP2006 benchmarks is significantly higher than that of CINT2006 benchmarks.

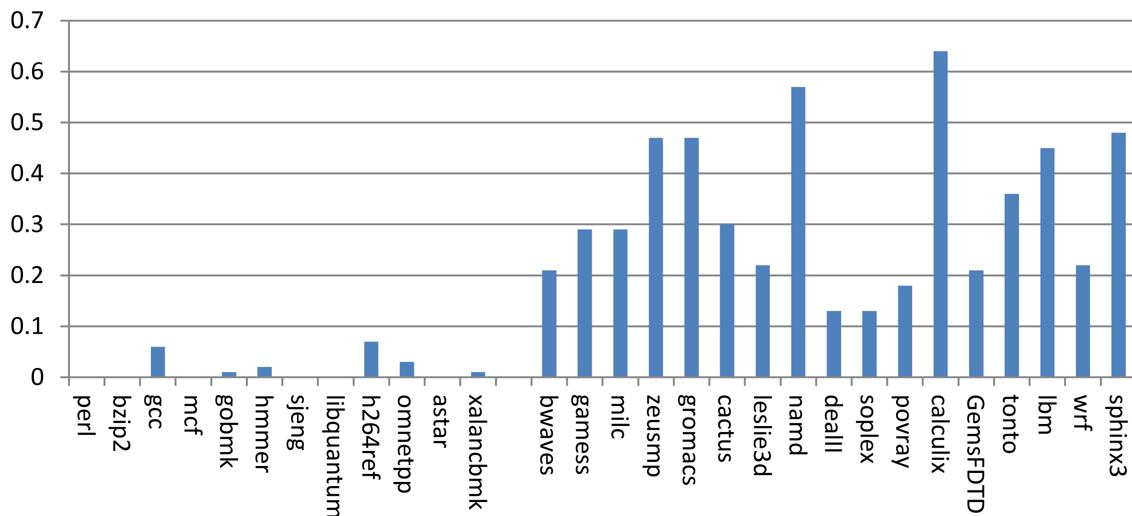


Figure 6.6: Average FP-IPC for benchmarks.

The FP pipeline can sustain four operations per cycle with optimized code, but FP-IPC values in Figure 6.6 are all significantly less than the maximum. There are many reasons for the large discrepancy between maximum and measured FP-IPC. First, each core has a commit width of 4 IPC, which means that there are other ops, most likely loads and stores that are required to support a high issue rate in the FP pipe. Second, although the FP unit can execute 4 instructions per cycle, not every execution unit can execute every operation. Hence, there is a limitation on the combinations of operations that can execute per cycle. Finally, there are data dependencies that restrict the ILP of the application. Hence, it is unlikely that the average issue rate will reach anywhere near the max sustainable issue rate. However, as seen by the results with FP throttling enabled, reducing the issue rate detrimentally impacts performance by reducing the FP throughput

during the critical stages of the application. A very conservative FP-IPC threshold of 0.1 was used in order to aggressively disable FP throttling when necessary. The goal is to improve all the CINT2006 cases without incurring a performance loss in the CFP2006 benchmarks.

Figure 6.3 shows that the dynamic scheme eliminates the performance losses in CFP2006 benchmarks while retaining most of the performance gains of the CINT2006 benchmarks. Performance losses are no longer seen for *namd*, *calculix*, and *sphinx3*. However, the dynamic scheme also reduced the benefits in some benchmarks such as *h264ref*, *povray*, and *leslie3d*. Subsequently, the dynamic scheme also had slightly worse E*D numbers (Figure 6.5) for the same benchmarks although it improves the average E*D numbers across all benchmarks.

6.5.5. Multi-core Execution

The results so far have focused on single threaded runs where one thread is running on one core within a Bulldozer module. The AMD Orochi processor, however, is capable of running eight threads. Multi-core execution adds additional complexity to the problem. First, the L2 cache, the fetch unit, and the floating point pipeline are shared between two cores in a Bulldozer module. This sharing can result in contention between threads, making them less capable of benefiting from a frequency boost. Second, depending on the homogeneity of the threads and how they execute, the two threads within a Bulldozer module may have differing requirements. Although each Bulldozer module can run at different frequencies, the two cores within a module must run at the same frequency.

Eight threaded SPECrate runs were performed and relative performance information was collected as shown in Figure 6.7. Once again, the numbers for the static and dynamic schemes are shown relative to *Base-3G* case. As before, CINT2006 benchmarks benefit the most from FP throttling while CFP2006 benchmarks suffer the most. However, the dynamic scheme is able to detect the applications that benefit from FP throttling and eliminate the performance loss on CFP2006 benchmarks.

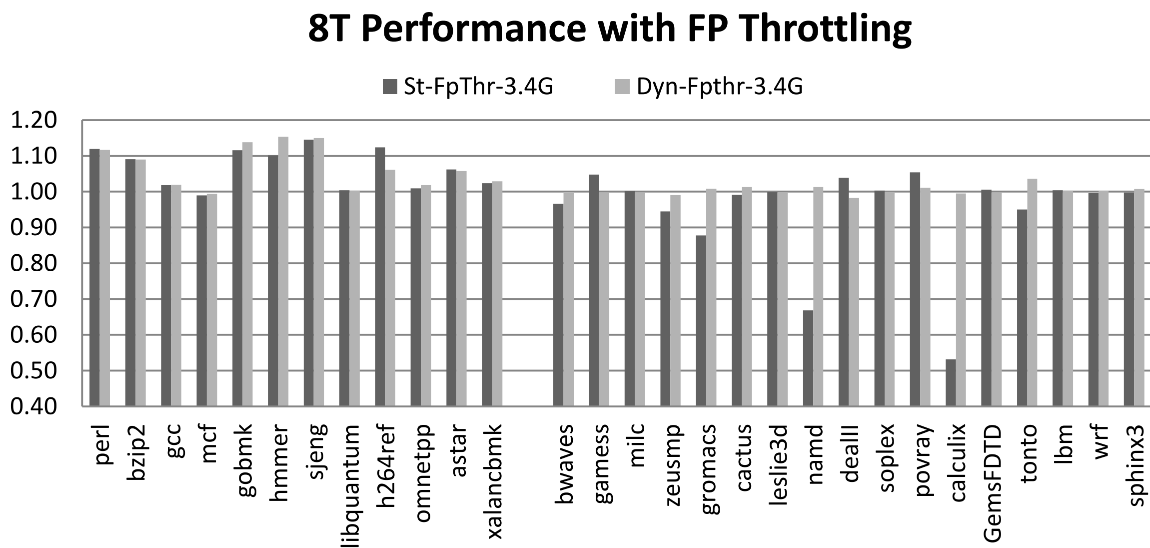


Figure 6.7: Performance with 8T execution relative to *Base-3G* case.

There are some differences in the results from the 1T runs. The number of CFP2006 benchmarks that benefit from FP throttling is not as great as the 1T case. The reason for this is that the FP pipeline is shared between two cores in a Bulldozer module. Although some CFP2006 benchmarks are not as sensitive to FP IPC the problem is exacerbated with the shared FP pipeline. Hence the utilization of a shared FP pipeline

increases when multiple threads run in the same module.

6.5.6. Improved Dynamic Scheme

The dynamic scheme using FP-IPC improved overall results, but there were some cases where the static scheme outperformed the dynamic scheme. There are a couple of reasons for the performance loss. The first is that the dynamic scheme is very simple and only examines one metric: FP-IPC. Thrashing between FP throttling enabled and disabled states will occur if the FP-IPC of the application hovers around the predetermined threshold value. This is the case for *h264ref*. In addition, an FP-IPC higher than the threshold does not always indicate that the application performance is dependent on FP throughput. For instance, a high overall IPC may indicate a large amount of ILP in the program, and FP-IPC is not as critical to performance. Both *dealII* and *povray* show this characteristic.

The dynamic scheme was modified, based on the above observations by examining not just the FP-IPC but also the ratio of FP-IPC to overall IPC (Table 6.3). The new scheme (*Dyn2-FpThr-3.4G*) is shown in Figure 6.8 for a subset of interesting benchmarks from SPEC CPU2006. The static and both dynamic schemes are shown relative to the *Base-3G* case. The new dynamic scheme (*Dyn2-FpThr-3.4G*) improves the performance of most of the benchmarks and brings them back up to the static levels (*St-FpThr-3.4G*) in many cases.

Algorithm 2: Improved dynamic scheme (Dyn2) using FP_IPC and FP_RATIO

```
Given: FP_IPC_THRESHOLD, FP_RATIO_THRESHOLD,
            CONSECUTIVE, DECISION_INTERVAL, P0, and PB
 1:  while TRUE  do
 2:      for each core, Ci  do
 3:          get fp_ipc;        /* FP-IPC */
 4:          get fp_ratio;   /* ratio: FP-IPC to IPC  */
 5:          if  fp_ipc  > FP_IPC_THRESHOLD  then
 6:              if  fp_ratio > FP_RATIO_THRESHOLD  then
 7:                  if  FP_throttling.enabled  then
 8:                      Frequency.adjust(P0);
 9:                      FP_throttling.Off()
10:                  end if
11:                  fp_mac_consecutive = 0;
12:              else /* fp_ratio <= FP_RATIO_THRESHOLD  */
13:                  if  fp_ratio_consecutive > CONSECUTIVE  then
14:                      if  !FP_throttling.enabled  then
15:                          FP_throttling.On();
16:                          Frequency.adjust(PB);
17:                      end if
18:                  end if
19:                  fp_ratio_consecutive = fp_ratio_consecutive + 1;
20:              end if
21:              fp_ipc_consecutive = 0;
22:          else /* fp_ipc  <= FP_IPC_THRESHOLD  */
23:              if  fp_ipc_consecutive > CONSECUTIVE  then
24:                  if  !FP_throttling.enabled  then
25:                      FP_throttling.On();
26:                      Frequency.adjust(PB);
27:                  end if
28:              end if
29:              fp_ipc_consecutive = fp_ipc_consecutive + 1;
30:          end if
31:      end for
32:      Sleep.time(DECISION_INTERVAL);
33:  end while
```

Table 6.3: Improved algorithm for dynamic voltage guardband management.

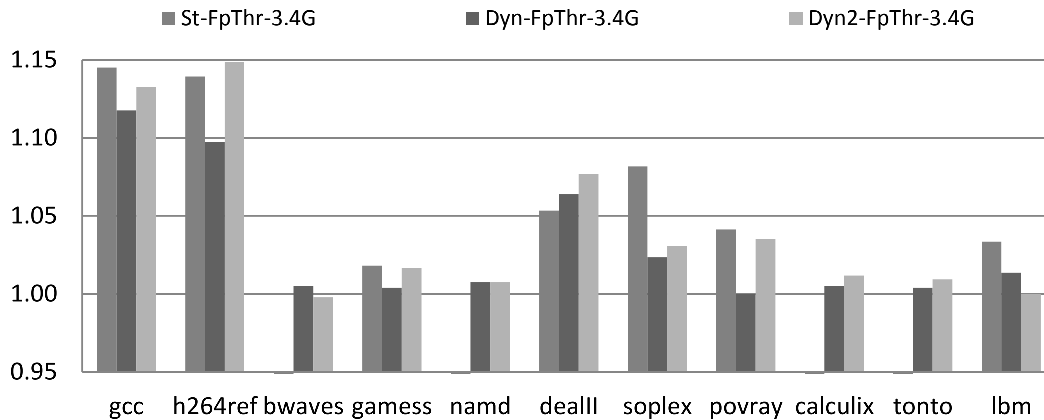


Figure 6.8: Results with improved dynamic scheme (*Dyn2-FpThr-3.4G*).

Figure 6.9 shows the percent of time FP throttling was enabled with the two different dynamic schemes. In many cases, the dynamic scheme is alternating between throttled and non-throttled states. In *dealII* for instance, by spending a small portion of the total run time with throttling disabled, the *Dyn2-FpThr-3.4G* scheme is able to improve on the static scheme. In *h264ref* and *povray*, the *Dyn2-FpThr-3.4G* scheme is able to enable FP throttling for nearly the entire run and improve performance when compared to the *Dyn-FpThr-3.4G* scheme. These results show that the dynamic scheme is not only able to detect which applications require FP throttling and which do not, but is also able to determine phases within an application that can take advantage of throttling.

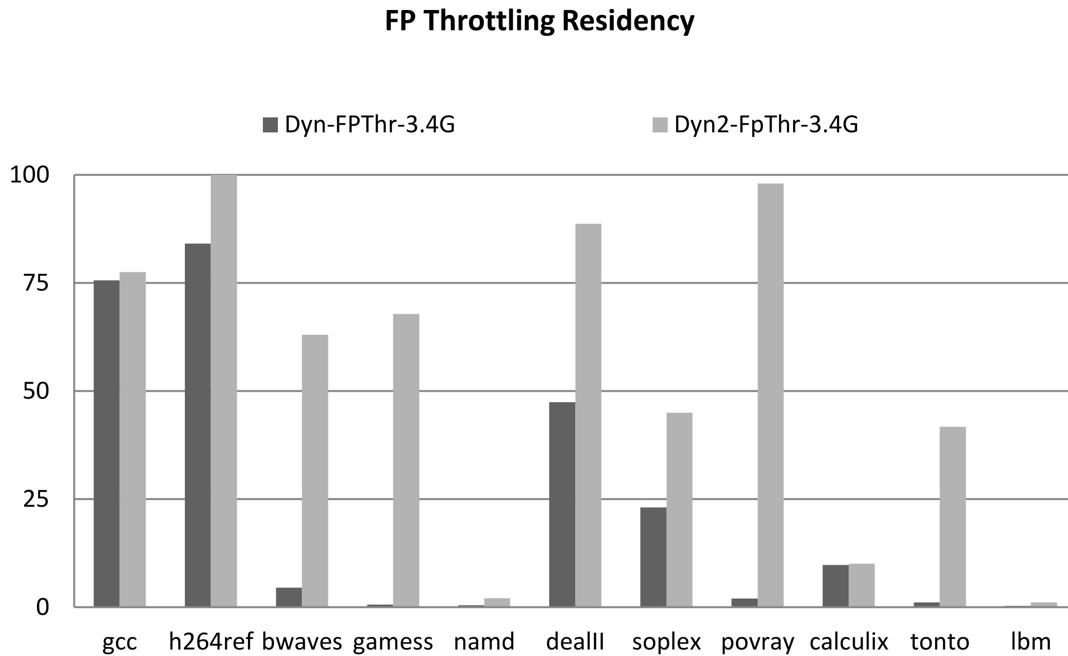


Figure 6.9: Percent time spent with FP throttling enabled.

6.6. SUMMARY

In this chapter, the issue of the voltage guardband was addressed and how one can reduce the guardband to increase performance in existing processors was discussed. A hardware technique is used to limit the issue rate in the floating point scheduler which resulted in a reduction in the worst case voltage droop. A number of existing and newly generated stressmarks were used to translate the reduction in voltage guardband into a

frequency boost, and the impact this has on the SPEC CPU2006 benchmarks was shown. Based on the evaluation in this chapter, a dynamic scheme was developed to detect when to trade off floating point throughput for a frequency increase. Two dynamic schemes were implemented in software and it is shown that these schemes can improve the performance of several benchmarks without sacrificing the performance of others.

Chapter 7: Conclusion and Future Research

7.1. Summary and Conclusion

This dissertation focused on di/dt issues in microprocessors. The characterization methodology in this dissertation helps designers analyze large current draws and the following voltage fluctuations that may lead to system failure. Also, one technique that dynamically manages voltage margins is suggested to enable users to select optimum points in performance and power tradeoffs of a processor.

To characterize di/dt noise in a processor, automating the generation of di/dt stressmarks frees designers from the tedious task of writing manual stressmarks. The complexities involved with designing a di/dt stressmark, especially for multi-core systems, are enormous. A dithering algorithm is implemented to aid in multi-core stressmark generation and an automatic stressmark generation tool, AUDIT, that uses sub-blocking and dithering to produce stressmarks is presented. It is shown how AUDIT stressmarks compare to existing stressmarks and standard benchmarks, and results showing AUDIT's ability to adjust to different processor characteristics such as shared resources, FP throttling, and different processors are presented. The ability to generate various stressmarks automatically will likely enable the designer to thoroughly study the susceptibility of processors to voltage fluctuations and to design appropriate mechanisms for reliable processor operation.

It is interesting and important to study the impact on compiler optimizations on the voltage fluctuations during program execution. Several programs were run with optimized and unoptimized versions of code from the same program, and performance, power, energy and voltage fluctuations studied.

Energy can be dramatically reduced by increasing the number of threads and performing compiler optimization. Unfortunately, trends in average or maximum power during execution cannot be correlated in a systematic manner with performance. The intricacies of the code sequences and the functional units bring unpredictable trends between performance and power trade-offs. Generally one cannot predict how voltage droop would change with optimization levels. In some cases, performance, power, and voltage droop can be improved with compiler optimization. Short-runtime could give different results in voltage droop. Therefore, designers utilizing simulations for such studies should be careful interpreting simulation results, which are run with very short time compared to real hardware. Voltage droop is not much changed with compiler and library optimizations in *miniFE* and *HP Linpack*. Therefore, only with compiler optimization, supply voltage reliability is mainly affected by load-line effect [33], i.e., resistive part rather than inductive part of the processor and the power distribution network circuit. If one cannot predict voltage droop change with static compiler optimization, a dynamic mitigation method can be useful.

The issue of a voltage guardband is addressed, and a discussion on how to reduce the guardband to increase performance in existing processors is presented. A hardware technique to limit the throughput in the floating point scheduler was used, which resulted in a reduction in the worst case voltage droop. A number of existing and newly generated stressmarks were used to translate the reduction in voltage guardband into a frequency boost and the impact this has on the SPEC CPU2006 benchmarks is presented. Based on the evaluation, a dynamic scheme is developed to detect when to trade off floating point throughput for a frequency increase. Finally, two dynamic schemes were implemented in software and it is shown that performance for some benchmarks can be

improved without deteriorating the performance for others.

Recent supercomputers are built with Intel's Xeon, AMD's Opteron, and IBM's Power processors [66]. In those processors, FP-units consume the maximum dynamic power by running SIMD-style, fused, and/or combined FP operations such as (V)FPMADD (Mult+Add), (V)FP+LD/ST, etc. [3][33]. A sequence consisting of no activity followed by a large number of FP operations is the biggest culprit in causing voltage droops on many processors. Such voltage droops can cause failures. Fault-free execution of HPC workloads is going to be increasingly difficult without a large voltage margin, but higher margins mean consuming higher power than needed or sacrificing performance by reducing maximum supported frequency. Therefore, this dissertation will provide a practical guide to characterize and manage voltage droops in HPC processors and workloads.

7.2. Future Research

This research can be extended in the following directions:

7.2.1. Extension to AUDIT

AUDIT can be extended to different ISAs and processor architectures. Besides Alpha and x86, ARM's or IBM's processor is also popular and would be excellent targets. In addition to the extension to a different ISA/processor, the simulation path in AUDIT can be modified to include a Register-Transfer-Level (RTL) simulation flow. The processor model described in RTL is more cycle-accurate than that in a C-level simulator. RTL synthesis can give reliable power numbers for the system, so a post-silicon measurement is not necessary. The simulation technique of using a computing

farm can dramatically compensate for RTL's long runtime so that the runtime would not be a problem. Lastly, using heterogeneous threads in AUDIT will be an interesting challenge; it requires more ideas to synchronize different threads, which have different code sizes and contents. Also, to generate heterogeneous threads, the search space for the best instruction scheduling exponentially increases.

7.2.2. Improvement of Dynamic Scheme for Managing Voltage Margins

The research presented here on guardband management was constrained by the specific features of the AMD processor used in the experiments. In an unconstrained environment, there might be other types of events that can be monitored to decide what to throttle and when to throttle. One obvious possibility is to continuously monitor the occupancy of integer and floating point reservation stations. This information, more than the average FP-IPC, may be a better indicator of the application's need for a wide FP pipe. But detailed studies would be needed before concluding that fined-grain monitoring of processor events can help. There is the overhead of switching modes and frequent switching of modes may result in an unstable system.

7.2.3. Charaterization and Management of Voltage Noise in GPU

Recently GPUs have been studied with various points of views. Combining a CPU with GPUs on a single chip is one of the viable research problems. In a CPU-GPU chip, the GPUs take a large area and consume significant power, comparable to a multi-core CPU. To characterize power and voltage noise is important, and more effective management techniques for power and voltage noise will be required. Di/dt stressmark

generation for GPUs is an interesting extension to AUDIT. Also a smart, dynamic throttling on execution units in GPU will increase performance while suppressing voltage noise.

Bibliography

[1] AMD Core Math Library, Available: http://developer.amd.com/tools/cpu-development/amd-core-math-library-acml/

[2] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proceedings of The 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, pp. 196-207, 2009.

[3] R. Bertran, A. Buyuktosunoglu, M. Gupta, M. Gonzalez, and P. Bose, "Systematic energy characterization of CMP/SMT processor systems via automated micro-benchmarks," in *Proceedings of The 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, pp. 199-211, 2012.

[4] K. Bharath, E. Engin, and M. Swaminathan, "Automatic package and board decoupling capacitor placement using genetic algorithms and M-FDM," in *Proceedings of The 45th Design Automation Conference (DAC)*, pp. 560-565, 2008.

[5] C. Bienia, Benchmarking modern multiprocessors, Ph.D. Thesis, Princeton University, 2011.

[6] BLAS library, Available: http://www.netlib.org/blas/

[7] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of The 27th International Symposium on Computer Architecture (ISCA-27)*, pp. 83-94, 2000.

[8] M. Butler, L. Barnes, D. Sarma, and B. Gelinas, "Bulldozer: an approach to multithreaded compute performance," *IEEE Micro, Vol. 31, Iss. 2*, pp. 6-15, 2011.

[9] W. El-Essawy and D. Albonesi, "Mitigating inductive noise in SMT processors," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 332-337, 2004.

[10] D. Ernst, N. S. Kim, S. das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proceedings of The 36th Annual IEEE/ACM*

International Symposium on Microarchitecture (MICRO-36), pp. 7-18, 2003.

[11] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, K. Flautner, "RAZOR: circuit-level correction of timing errors for low power operation," *IEEE Micro*, *Vol. 24, Iss. 6,* pp. 10-20, 2004.

[12] T. Fischer et al., "Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU," in *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 78-80, 2011.

[13] Genetic Algorithm Utility Libray (GAUL), Available: http://gaul.sourceforge.net/

[14] E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-8)*, pp. 7-16, 2002.

[15] M. Gupta, J. Oatley, R. Joseph, G. Wei, and D. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1-6, 2007.

[16] M. Gupta, K. Rangan, M. Smith, G. Wei, and D. Brooks, "DeCoR: a delayed commit and rollback mechanism for handling inductive noise in processors," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, pp. 381-392, 2008.

[17] M. Gupta, K. Rangan, M. Smith, G. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 123-128, 2007.

[18] M. Gupta, V. Reddi, G. Holloway, G. Wei, and D. Brooks, "An event-guided approach to reducing voltage noise in processors," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE),* pp. 160-165, 2009.

[19] K. Hazelwood and D. Brooks, "Eliminating voltage emergencies via microarchitectural voltage control feedback and dynamic optimization," in

Proceedings of International Symposium on Low Power Electronics and Design (ISLPED), pp. 326-331, 2004.

[20] High Performance LINPACK (HPL), Available: http://www.netlib.org/benchmark/hpl/

[21] HSPICE, Available:http://www.synopsys.com/tools/Verification/AMSVerification/CircuitSimulation/HSPICE/

[22] M. Popovich, A. Mezhiba, and E. Friedman, *Power Distribution Networks with On-Chip Decoupling Capacitors*, Springer, 2008.

[23] N. James, P. Restle, J. Friedrich, B. Houtt, and B. McCredie, "Comparison of split-versus connected-core supplies in the POWER6 microprocessor," in *Proceedings of the International Solid State and Circuits Conference (ISSCC)*, pp. 298-300, 2007.

[24] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-9)*, pp. 79-90, 2003.

[25] Russ Joseph, Zhigang Hu, and Margaret Martonosi, "Wavelet Analysis for Microprocessor Design: Experiences with Wavelet-Based dI/dt Characterization," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-9),* pp. 36-46, 2004.

[26] A. Joshi, L. Eeckhout, L. John, and C. Isen, "Automated microprocessor stressmark generation," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, pp. 229-239, 2008.

[27] M. Ketkar and E. Chiprout, "A microarchitecture-based framework for pre- and post-silicon power delivery analysis," in *Proceedings of The 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, pp. 179-188, 2009.

[28] Y. Kim and L. John, "Automated di/dt stressmark generation for microprocessor

power delivery networks," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 253-258, 2011.

[29] Y. Kim, L. John, S. Manne, M. Schulte, and S. Pant, “Impact of compiler optimizations on voltage droops and reliability of an SMT, multi-core processor,” in *Proceedings of the 1st International Workshop on Secure and Resilient Architectures and Systems (SRAS '12) (in conjunction with PACT),* pp. 1-6, 2012.

[30] Y. Kim, L. John, S. Pant, S. Manne, M. Schulte, W. Bircher, and M. Govindan, “AUDIT: stress-testing the automatic way”, in *Proceedings of The 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, pp. 212-223, 2012.

[31] W. Kim, M. Gupta, G. Wei, and D. Brooks, “System level analysis of fast, per-core DVFS using on-chip switching regulators,” in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, pp. 123-134, 2008.

[32] N. Kurd, J. Douglas, P. Mosalikanti, and R. Kumar, "Next generation Intel® Core micro-architecture (Nehalem) clocking architecture," *IEEE Journal of Solid-state Circuits*, *Vol. 44, Iss. 4,* pp. 1121-1129, 2009.

[33] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tiemo, and J. Carter, "Active management of timing guardband to save energy in POWER7," in *Proceedings of The 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*, pp. 1-11, 2011.

[34] T. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *Proceedings of The 39th International Symposium on Computer Architecture (ISCA-39)*, pp. 249-260, 2012.

[35] miniFE, Available: https://software.sandia.gov/mantevo/about.html

[36] F. Mohamood, M. Healy, S. Lim and H. Lee, "A floorplan-aware dynamic inductive noise controller for reliable 2D and 3D microprocessors," in *Proceedings*

of The 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39), pp. 3-14, 2006.

[37] S. Mukherjee, M. Kontz, S. Reinhardt, “Detailed design and evaluation of redundant multi-threading alternatives,” in *Proceedings of The 29th International Symposium on Computer Architecture (ISCA-29)*, pp. 99-110, 2002.

[38] S. Nussbaum, “AMD Trinity Fusion APU,” in *Hot Chips: A Symposium on High Performance Chips (HC24)*, 2012.

[39] OpenMPI, Available: http://www.open-mpi.org/

[40] M. Pant, P. Pant, D. Willis, and V. Tiwari, "Architectural solution for the inductive noise problem due to clock-gating," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 255-257, 1999.

[41] M. Pant, P. Pant, D. Wills, V. Tiwari, “Inductive noise reduction at the architectural level,” in *Proceedings of 13th International Conference on VLSI Design*, pp. 162-167, 2000.

[42] M. Pant, P. Pant, and D. Wills, “On-chip decoupling capacitor optimization using architectural level prediction,” *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, *Vol. 10, Iss. 3,* pp. 319-326, 2002.

[43] S. Pant, AMD, personal communication.

[44] S. Pant and E. Chiprout, "Power grid physics and implications for CAD," in *Proceedings of The 43rd Design Automation Conference (DAC)*, pp. 199-204, 2006.

[45] J. Patel, "CMOS process variations: a critical operation point hypothesis," Online Presentation, 2008.

[46] M. Popovich, M. Sotman, A. Kolodny, and E. Friedman, “Effective radii of on-chip decoupling capacitors,” *IEEE Transactions on Very Large Scale Integration Systems (TVLSI), Vol. 16, Iss. 7,* pp. 894-907, 2008.

[47] M. Powell and T. Vijaykumar, "Exploiting resonant behavior to reduce inductive noise," in *Proceedings of The 31st International Symposium on Computer*

Architecture (ISCA-31), pp. 288-299, 2004.

[48] M. Powell and T. Vijaykumar, “Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage,” in *Proceedings of The 30th International Symposium on Computer Architecture (ISCA-30)*, pp. 72-83, 2003.

[49] M. Powell and T. Vijaykumar, “Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise,” in *Proceedings of International Sumposium on Low Power Electronics and Design (ISLPED)*, pp. 223-228, 2003.

[50] V. Reddi, Software-assisted hardware reliability: enabling aggressive timing speculation using run-time feedback from hardware and software, Ph.D. Thesis, Harvard University, 2010.

[51] V. Reddi, S. Campanoni, M. Gupta, M. Smith, G. Wei, D. Brooks, K. Hazelwood, “Eliminating voltage emergencies via software-guided code transformations,” *ACM Transactions on Architecture and Code Optimization (TACO), Vol. 7, Iss. 2, No. 12*, 2010.

[52] V. Reddi, M. Gupta, G. Holloway, M. Smith, G. Wei, and D. Brooks. “Predicting voltage droops using recurring program and microarchitectural activity,” *IEEE Micro*, *Vol. 30, Iss. 1,* pp. 110, 2010.

[53] V. Reddi, M. Gupta, G. Halloway, G. Wei, M. Smith, and D. Brooks, "Voltage emergency prediction: using signatures to reduce operating margins," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-15)*, pp. 18-29, 2009.

[54] V. Reddi, M. Gupta, M. Smith, G. Wei, D. Brooks, and S. Campanoni, “Software-assisted hardware reliability: abstracting circuit-level challenges to the software stack,” in *Proceedings of The 46th Design Automation Conference (DAC)*, pp.788-793, 2009.

[55] V. Reddi, M. Gupta, K. Rangan, S. Campanoni, G. Holloway, M. Smith, G. Wei, and D. Brooks, “Voltage noise: why it’s bad, and what to do about it,” in

Proceedings of IEEE Workshop on Silicon Errors in Logic – System Effects (SELSE), 2009.

[56] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. Smith, G. Wei, and D. Brooks. "Voltage noise in production processors," *IEEE Micro, Vol. 31, Iss. 1*, pp. 20-28, 2011.

[57] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. Smith, G. Wei, and D. Brooks, "Voltage smoothing: characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Proceedings of The 43th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-43)*, pp. 77-88, 2010.

[58] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weisman, "Power management architectures of the Intel microarchitecture code-named Sandy Bridge," in *Hot Chips: A Symposium on High Performance Chips (HC 23)*, 2011.

[59] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*, pp. 45-57, 2002.

[60] SimpleScalar, Available: http://www.simplescalar.com/

[61] Stephen Kosonocky, AMD, personal communication.

[62] SPEC CPU2006, Available: http://www.spec.org/cpu2006/

[63] The NASM Development Team, "NASM - the netwide assembler," Available: http://www.nasm.us.

[64] A. Taparia, B. Banerjee, and T. Viswananthan, "Power-supply noise reduction using active inductors in mixed-signal systems," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI), Vol. 19, Iss. 12*, pp. 1960-1968, 2010.

[65] M. Tehranipoor, K. Peng, and K. Chakrabarty, *Test and Diagnosis for Small-Delay Defects*, Springer, 2011.

[66] Top 500 Supercompuer Site, Available: http://www.top500.org/

[67] M. Valluri and L. John, “Is compiling for performance == compiling for power?” in *Proceedings of 5th Annual Workshop on Interaction Between Compilers and Computer Architectures (INTERACT-5)*, pp. 101-115, 2001.

[68] D. Weiss, M. Dreesen, M. Ciraula, C. Henrion, C. Helt, R. Freese, T. Miles, A. Karegar, R. Schreiber, B. Schneller, and J. Wuu, "An 8MB Level-3 cache in 32nm SOI with column-select aliasing," in *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 258-260, 2011.