



University
of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

**Harnessing CPU Electromagnetic
Emanations for Power-Delivery
Network Characterization**

Zacharias Hadjilambrou

A dissertation submitted to the University of Cyprus in partial
fulfillment of the requirements for the degree of Doctor of
Philosophy

November, 2019

©Zacharias Hadjilambrou, 2019

Zacharias Hadjilambrou

VALIDATION PAGE

Doctoral Candidate: Zacharias Hadjilambrou

Doctoral Dissertation Title: Harnessing CPU Electromagnetic Emanations for Power-Delivery Network Characterization

*The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the **Department of Computer Science** and was approved on the 8/11/2019 by the members of the **Examination Committee**.*

Examination Committee:

Research Supervisor:  _____

Yiannakis Sazeides

Research Supervisor:  _____

Shidhartha Das

Committee Member:  _____

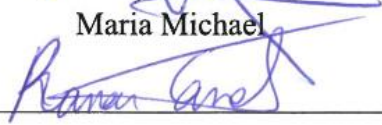
Constantinos S. Pattichis

Committee Member:  _____

Elias Athanasopoulos

Committee Member:  _____

Maria Michael

Committee Member:  _____

Ramon Canal

DECLARATION OF DOCTORAL CANDIDATE

The present doctoral dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Zacharias Hadjilambrou


.....

Περίληψη

Ο επαγωγικός θόρυβος τάσης είναι μια από τις σημαντικές αυξανόμενες ανησυχίες στους σύγχρονους επεξεργαστές. Η υπερβολική πτώση της τάσης μπορεί να προκαλέσει μη ορθή λειτουργία. Ενώ η υπερβολική αύξηση της τάσης αυξάνει την κατανάλωση ενέργειας, τη διάχυση της θερμότητας και μπορεί να επιταχύνει τη γήρανση του υλικού. Ο θόρυβος τάσης προκαλείται από ξαφνικές μεταβολές στην κατανάλωση ρεύματος. Σύγχρονες τάσεις σχεδίασης όπως οι τεχνικές χαμηλής κατανάλωσης ενέργειας (π.χ. power-gating, clock-gating), οι γρήγορες συχνότητες και η υψηλή κατανάλωση ρεύματος, επιδεινώνουν τον θόρυβο τάσης. Για την αντιμετώπιση του προβλήματος είναι υψίστης σημασίας να αναπτυχθούν εργαλεία που επιτρέπουν τον χαρακτηρισμό των δικτύων παροχής ισχύος (ΔΠΙ) και την παρακολούθηση του επαγωγικού θορύβου τάσης. Τέτοια εργαλεία παρακολούθησης είναι χρήσιμα κατά τη διάρκεια των δοκιμών για να βοηθούν στον επαρκή καθορισμό των περιθωρίων τάσης που θα επιτρέπουν την ομαλή και ενεργειακά αποδοτική λειτουργία του επεξεργαστή. Επίσης, αυτά τα εργαλεία παρακολούθησης χρησιμοποιούνται στην κατασκευή κυκλωμάτων μετριάσμου του θορύβου τάσης που προστατεύουν τον επεξεργαστή από χαμηλή τάση και εξασφαλίζουν αξιόπιστη λειτουργία.

Αυτή η διδακτορική θέση προτείνει μια νέα μεθοδολογία για τον χαρακτηρισμό ΔΠΙ. Η προτεινόμενη μεθοδολογία βασίζεται στην ασύρματη ανίχνευση της ηλεκτρομαγνητικής ακτινοβολίας (ΗΑ) που εκπέμπεται από το ΔΠΙ χρησιμοποιώντας μια κεραία και ένα όργανο για την ανάλυση των ληφθέντων σημάτων (όπως ένας αναλυτής φάσματος). Η προτεινόμενη προσέγγιση έχει τρία σημαντικά πλεονεκτήματα σε σύγκριση με κλασικές προσεγγίσεις χαρακτηρισμού ΔΠΙ: α) είναι μη παρεμβατική και βολική καθώς δεν φορτίζει, διακόπτει ή παρεμποδίζει το σύστημα με οποιονδήποτε τρόπο επιτρέποντας την παρακολούθηση του συστήματος ως έχει, β) θεωρητικά, είναι συμβατή με οποιαδήποτε πλατφόρμα καθώς οποιοδήποτε υλικό υπολογιστών εκπέμπει ΗΑ, γ) δεν επιβάλλει επιπλέον

κόστος στον κατασκευαστή καθώς δεν απαιτεί υλικούς πόρους πάνω στον επεξεργαστή. Η προτεινόμενη μεθοδολογία βασίζεται στην παρατήρηση ότι η ΗΑ μεγάλης ισχύος κοντά στη συχνότητα συντονισμού του ΔΠΙ συσχετίζεται με υψηλό θόρυβο τάσης. Αυτή η δουλειά παρέχει τη θεωρητική βάση και την απόδειξη ότι αυτή η παρατήρηση ισχύει. Εκμεταλλευόμαστε αυτή την παρατήρηση με δύο τρόπους: α) για να μετρήσουμε τη συχνότητας συντονισμού του ΔΠΙ και β) για να θέσουμε αποδοτικά περιθώρια τάσης με τη βοήθεια ειδικών προγραμμάτων που μεγιστοποιούν το θόρυβο τάσης, αυτά τα προγράμματα παράγονται μέσω ενός γενετικού αλγόριθμου που μεγιστοποιεί την ΗΑ. Η προσέγγιση μας αξιολογείται σε διάφορους σύγχρονους επεξεργαστές. Σε όλους τους δοκιμασμένους επεξεργαστές με την προτεινόμενη μεθοδολογία μπορέσαμε να παρακολουθήσουμε με επιτυχία τον θόρυβο τάσης, να αναγνωρίσουμε τη συχνότητα συντονισμού, να δημιουργήσουμε προγράμματα που προκαλούν υψηλό θόρυβο τάσης και να θέσουμε αποδοτικά περιθώρια τάσης που μειώνουν την κατανάλωση ενέργειας. Επίσης, αυτή η διδακτορική θέση προσφέρει την πρώτη δημόσια διαθέσιμη εφαρμογή για αυτόματη δημιουργία στρες προγραμμάτων. Η εφαρμογή βασίζεται σε γενετικούς αλγόριθμους και έχει δοκιμαστεί με επιτυχία σε πολλούς διαφορετικούς επεξεργαστές.

Abstract

Inductive dI/dt voltage-noise is a growing concern in modern processors. Voltage-noise is caused by sudden transitions in current consumption. Voltage-noise is manifested as large voltage droops and overshoots below and above the nominal supply-voltage. These events are a threat to reliability, robust execution and hardware longevity. Voltage droops may cause unreliable execution, whereas voltage overshoots increase power consumption, heat dissipation and may accelerate the hardware aging. Modern design trends such as: a) aggressive low-power techniques (e.g. power-gating, clock-gating), b) high dynamic-power range of multicore processors, and c) high current consumption, exacerbate voltage-noise rendering power-delivery a critical concern. Tools for characterizing Power Delivery Networks (PDN) and monitoring inductive dI/dt voltage-noise are required for: a) processor voltage and frequency margining at the post-silicon chip testing phase and b) for designing voltage-noise mitigation circuits that protect processor from voltage droops and ensure reliable operation.

This thesis proposes a novel methodology for PDN characterization. The proposed methodology is based on wirelessly sensing the PDN emanated electromagnetic (EM) radiation using an antenna and an instrument for analyzing the received signals such as a spectrum analyzer. The proposed approach has three major advantages compared to state-of-the-art PDN characterization approaches: a) it is non-intrusive and convenient as it doesn't load, interrupt or physically interfere with the system in any way allowing monitoring the system as it is, b) it is cross-platform as all hardware emanates EM radiation, c) it adds zero-overhead as it does not require development effort nor on-chip/on-package resources. The proposed EM methodology is based on the observation that high amplitude EM emanations at the PDN resonance frequency are correlated with high voltage-noise. This thesis provides the theoretical basis and conclusive evidence that this correlation holds

true. We exploit this correlation for measuring the PDN resonance frequency and for voltage margining CPUs with the help of dI/dt stress-tests generated with a genetic algorithm (GA) that maximizes the amplitude of EM emanations. The EM approach is successfully evaluated on five different processors. On all tested processors we are able to successfully monitor PDN oscillations, identify the resonance-frequency and generate dI/dt stress-tests that cause higher voltage-noise and have higher minimum operational voltage (V_{MIN}) than conventional workloads.

Additionally, this thesis delivers GeST (Generating Stress-Tests), a GA based framework for automatic stress-test generation that is developed for the needs of this thesis. To the best of our knowledge GeST is the first publicly available framework for stress-test generation. The key strengths of GeST are its flexibility and extensibility as it provides an easy interface for using and extending the framework.

Acknowledgements

This PhD thesis would have not been possible without the help of many great people that I had the fortune to meet and work with during the last 7 years. First and foremost, I would like to thank my advisor Yiannakis Sazeides with whom I had the pleasure to work since 2012. Yanos was my advisor during my BSc, MSc and PhD studies. In parallel with my studies, Yanos gave me the opportunity to work for large impact European research projects such as EuroCloud, Harpa and Uniserver. I cannot thank him enough for giving me these opportunities. All these years we had an excellent cooperation. Yanos is characterized by his strong work ethics and tremendous passion for science. These aspects compose the exceptional advisor, scientist and person he is.

I also consider myself extremely fortunate to have cooperated with another remarkable man; my other advisor Shidhartha Das. The first time I met Sid was during my first internship in Arm in 2013. Since then we had a continuous cooperation and the epitome of our efforts is this PhD thesis. Sid was my manager during my last internship in Arm (from Sep 2017 – Dec 2017). The work performed during that internship contributed significantly in delivering this thesis.

Next, a big thank you is devoted to Marco A Antoniadis and Kyriakos Neophytou. With their expertise in antennas and electromagnetic waves, they provided invaluable help and support. They constructed all the antennas used in this thesis and assisted in gathering all the necessary equipment for setting up the EM experimental setup.

I would like to thank people who I cooperated with during my time in Xi Computer Architecture research group. I would like to thank Chrysostomos Nicopoulos with whom we have cooperated closely for the Eurocloud and Harpa projects. I would like to thank Pedro Trancoso; we cooperated for the Uniserver project and I was truly lucky to have him as my professor in various university courses as well. Of course, I would like to thank all the fellow students and researchers that passed from the group: Marios, Panagiota,

Panayiotis, Lorena, Giorgos, Kypros, Georgia and many others. With these people I shared the same lab for many years and their presence made life at work easier and more enjoyable.

Also, I would like to thank my colleagues and friends at Arm Cambridge offices. I would like to thank Emre Özer who contributed in establishing close working relationship between Arm and University of Cyprus. I would like to thank Paul Whatmough, my line manager during my first and second Arm internships who has contributed to the development of the GeST framework. I would like to thank David Bull for being always willing to share his high technical expertise. A big thank you to the people I hanged together with; my fellow Arm internship students and friends Vaibhav, Rohan and Jeremy. Their presence made life and work in Cambridge much more amusing.

Many thanks to the members of the examination committee for their time and their comments that helped improving this thesis. Likewise, special thanks go to all conference and journal reviewers which contributed to polishing this thesis through their feedback.

Many thanks to the institutions that contributed to this thesis: University of Cyprus and Arm. I would like to thank University of Cyprus for providing me the appropriate environment to carry this thesis, and for being my second home for the last eleven years. Also, a big thanks to Arm for providing me the opportunity to work there during three different internships.

Finally, I genuinely express my thankfulness towards my family and Avraamia for encouraging me and standing by my side during this journey. I dedicate this PhD thesis to my grandfather, Zacharias Hadjilambrou, who although passing away he has always been a role model and an inspiration, guiding my way through science. I remember him from a very young age sitting with me and teaching me how to read and write. He was also a holder of a doctoral degree and I am very proud and grateful for achieving to follow his footsteps.

Lalis, Srikumar Venugopal, Arnau Prat-Pérez, Alejandro Lampropulos, Marios Kleanthous, Andreas Diavastos, **Zacharias Hadjilambrou**, Panagiota Nikolaou, Yiannakis Sazeides, Pedro Trancoso, George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Shidhartha Das: An energy-efficient and error-resilient server ecosystem exceeding conservative scaling limits. DATE 2018

11. **Poster:** Harnessing CPU Electromagnetic Emanations for Voltage Noise Characterization, **Zacharias Hadjilambrou** and Yiannakis Sazeides ACM Student Research Competition held in conjunction with Parallel Architectures and Compilation Techniques Conference (PACT18), November 2018
12. **Poster:** Shaving the Safety Margins by Exposing Intrinsic Hardware Heterogeneity **Zacharias Hadjilambrou**, Konstantinos Tovletoglou, Panagiota Nikolaou, Charalambos Chaliou, Dimitrios Nikolopoulos, Pedro Trancoso, Yanos Sazeides and Georgios Karakonstantis, ACACES 2016, July 2016

Contents

Chapter 1. Introduction	1
1.1 Motivation	1
1.2 Contributions	3
1.3 Document Structure	5
Chapter 2. Background and Related Work	6
2.1 CPU Power and Energy-Efficiency Techniques	6
2.1.1 Eliminating Static-Power	7
2.1.2 Eliminating Dynamic Power Consumption (Switching Activity)	8
2.1.3 Eliminating Dynamic Power Consumption (Voltage-Frequency Reduction)	9
2.2 Power-Delivery-Networks	11
2.2.1 PDN Resonance-Frequencies	11
2.2.2 IR drop versus di/dt noise	12
2.3 Voltage Margins	15
2.3.1 V_{MIN} Characterization	16
2.4 Voltage Margin Elimination Techniques	17
2.5 Ldi/dt Noise Mitigation Techniques	18
2.6 Ldi/dt noise on GPUs	20
2.7 EM emanations exploitation	21
2.8 Stress-Tests	21
2.8.1 Performance Stress-Tests	22
2.8.2 Power-Viruses	22
2.8.3 Ldi/dt stress-tests	23
2.9 GA for Stress-Test Generation	24
Chapter 3. GeST Framework	28
3.1 GeST Framework Description	29
3.1.1 GA Engine	29
3.1.2 Inputs	34
3.1.3 Measurement and Fitness Evaluation	38
3.1.4 Output	39
3.2 GeST Evaluation Platforms	40

3.3	Power-Virus Generation	43
3.4	Voltage-Noise Virus Generation	49
Chapter 4.	EM methodology	52
4.1	Required Experimental Apparatus	52
4.2	Relationship Between CPU EM Emanations and On-Chip Voltage Noise	54
4.3	EM Resonance Frequency Detection (Loop Method)	57
4.4	EM Resonance Frequency Detection (Clock Method).....	59
4.5	EM di/dt virus Generation	61
Chapter 5.	Measurement Setup	64
Chapter 6.	PDN Characterization	70
6.1	ARM Cortex-A72	70
6.2	ARM Cortex-A53	72
6.2.1	Simultaneous Voltage Noise Monitoring of Multiple Voltage Domains	75
6.3	AMD Athlon II X4 645	76
6.4	Ampere Computing X-Gene2.....	78
6.5	Ampere Computing X-Gene3.....	79
Chapter 7.	EM based DVS Governors	85
7.1	EM Detection Governor.....	85
7.2	Core Allocation Governor	88
Chapter 8.	Conclusions	94
8.1	Summary.....	94
8.2	Future Work Directions	98
	Bibliography.....	102

List of Figures

Figure 1. Power equations.	7
Figure 2. (a) A simplified model of the PDN [55]. The impedance as seen by the die has multiple resonance frequencies, shown in the frequency-domain response in (b) and time-domain response to a step-current excitation in (c).....	12
Figure 3. Voltage droop and current consumption for a power virus at various CPU frequencies.	14
Figure 4. Voltage droop and current consumption for a dI/dt virus at various CPU frequencies.	14
Figure 5. V_{MIN} characterization flow.....	16
Figure 6. Resonant buildup that leads to very high voltage droops and overshoots during dI/dt virus execution on a x86 CPU. Measured with external Oscilloscope and active differential probe connected to on-package measurement pads.....	24
Figure 7. GeST overview.	30
Figure 8. A Typical GA flow.	31
Figure 9. Demonstration of GA operators.....	32
Figure 10. Example of an instruction definition and its necessary operands.	35
Figure 11. Cortex-A15 power results.	44
Figure 12. Cortex-A7 power results.	44
Figure 13. X-Gene2 temperature results.	45
Figure 14. Complex fitness function rewarding high temperature and instruction simplicity.	47
Figure 15. GeST CPU power virus search on i5-2400.....	48
Figure 16. Power consumption results for Intel i5-2400.....	49
Figure 17. Voltage-noise results on AMD Athlon CPU.	50
Figure 18. V_{MIN} results on AMD Athlon CPU.	51
Figure 19. Experimental setup for the ARM Juno board (left) and AMD desktop CPU (right).....	52

Figure 20. Measured $ S_{11[13]} $ for the square loop antenna indicating a self-resonance around 2.95 GHz.	54
Figure 21. a) Simulated waveforms showing the die voltage (VDIE) and die current (IDIE) in the simplified PDN model in Fig. 2. A pulsing ILOAD triggers the first-order resonance where the AC-component of both VDIE (Vac) and IDIE (Iac) maximize, thus maximizing the radiated EM power. b) Resonant oscillations (close to the resonance-frequency at 67MHz) triggered on an ARM Cortex-A72 cluster (on ARM Juno Platform) causes a corresponding peak in the measured EM power captured on a Spectrum Analyzer.....	56
Figure 22. Comparison of spectrum analyzer readings (left axis) with FFT of OC-DSO voltage readings (right axis) during execution of a dI/dt virus. The two measurements agree as they reveal spikes at the same frequencies.	57
Figure 23. EM resonance frequency exploration for Cortex-A72 PDN with loop method.	58
Figure 24. SCL stimulus reveals a resonance frequency in the range of 66-72MHz with two powered cores (C0C1) and 80-86MHz with one powered core (C0).	59
Figure 25. Resonance-Frequency exploration on Cortex-A72 with clock method.....	60
Figure 26. Resonance frequency exploration on Mali-T622 with the clock method.	61
Figure 27. OC-DSO calibration	65
Figure 28. Voltage waveforms obtained from OC-DSO for 3 different workloads. dI/dt virus causes the largest voltage noise.....	65
Figure 29. The Power-Delivery-Monitor layout on the left and on the right a die photo of a JUNO R1 SoC [73] (Juno R1 uses the Cortex-A57 CPU whereas the Juno R2 uses the Cortex-A72, apart from some differences in CPU micro-architecture the two SoCs are identical).....	67
Figure 30. AMD time-domain supply-voltage measurement setup.	67
Figure 31. Measurement of random signals in the frequency-spectrum of 1 to 3000MHz.	69
Figure 32. EM driven GA run on Cortex-A72. Peak amplitude (left y-axis) and maximum droop / dominant frequency (right y-axis) for the best individual of each GA generation.	70
Figure 33. Voltage droop and V_{MIN} measurements on Cortex-A72.	71
Figure 34. GA search for Cortex-A72 dI/dt virus with 20 loop instructions vs 50 loop instructions.	72

Figure 35. GA EM amplitude driven optimization for Cortex-A53.....	73
Figure 36. V_{MIN} measurements on Cortex-A53.	73
Figure 37. Resonance frequency exploration on Cortex-A53. For four powered cores (C0C1C2C3) the resonance frequency is 76.5MHz.....	74
Figure 38. Simultaneous monitoring of voltage emergencies across multiple voltage domains through EM emanations.....	75
Figure 39. Loop frequency sweep on Athlon II X4 645 reveals a resonance frequency at 78MHz.....	76
Figure 40. GA EM amplitude driven run on AMD CPU.....	77
Figure 41. V_{MIN} and voltage noise measurements on the AMD CPU.....	77
Figure 42. X-Gene2 resonance frequency exploration.....	79
Figure 43. GA search on X-Gene2.....	80
Figure 44. X-Gene2 V_{MIN} results.....	80
Figure 45. Voltage droop of virus vs NAS workloads.....	81
Figure 46. Cycles suffered droop per second versus number of active cores.	82
Figure 47. V_{MIN} measurements on X-Gene 3.....	83
Figure 48. Voltage over time for 3 different workloads (idle, SP and dI/dt virus).	84
Figure 49. DVS governor EM detection setup on X-Gene2.	85
Figure 50. EM predictor flow-chart.	86
Figure 51. EM predictor vs nominal.	87
Figure 52. Different core-allocation classes for 4 active threads. Idle cores and L2 are illustrated with white colour.....	89
Figure 53. Single core and PMD V_{MIN} characterization.	91
Figure 54. Allocation class V_{MIN} characterization.....	91
Figure 55. CPU power consumption over 60 hours workload.....	93
Figure 56. DVS governor vs nominal moving average.....	93

List of Tables

Table I. Comparison of related work on GA frameworks.....	25
Table II. GA parameters.....	34
Table III. GeST evaluation platforms.....	41
Table IV. Instruction breakdown of Cortex-A15 and Cortex-A7 power viruses.....	43
Table V. Power virus, simple power virus and IPC virus comparison.	46
Table VI. Experimental platforms.....	64
Table VII. All X-Gene2 core allocation classes.....	90
Table VIII. dI/dt virus comparison. SL denotes short latency and LL denotes long latency. Voltage margin = Nominal voltage – VirusVmin.....	97

Chapter 1. Introduction

1.1 Motivation

The combination of higher current demand at scaled supply-voltages [66], high operating frequencies, aggressive low-power techniques [39] and increasing core-counts exacerbate supply-voltage noise for CPUs both in mobile [20][46][47] and server/desktop [10][42][49] market segments. Large voltage noise is a threat to robust execution because when the supply voltage drops below a certain threshold, timing violations or bit-flips may occur [10][39][66]. This may lead to silent data corruption (SDC), application or system crashes and general system instability [42][73].

Manufacturers budget voltage margins (or guardbands) to ensure robustness even in the presence of worst-case voltage noise conditions¹. Consequently, production systems are typically operated at a higher supply voltage (and/or lower clock-frequency) than necessary under nominal operating conditions. Accurate determination of voltage margins is critical since optimistic margining (where the added margins are not adequately provisioned for the rare worst-case noise event) can cause abrupt system-failures in the field. Conversely, excessive margining adversely impacts CPU energy-efficiency [9][10][42][44][56][66].

A key aspect of margining production systems is the determination of the worst-case inductive component (referred to as “LdI/dt” or “dI/dt”) of the voltage noise [10]. LdI/dt events are abrupt changes in CPU current demand that cause voltage noise oscillations excited at the Power-Delivery-Network (PDN) 1st order resonance-frequency. The dI/dt voltage-noise component typically dominates over the resistive component (referred to as “IR”) in the PDN of modern computing systems [20][44][55]. In comparison with aperiodic

¹Voltage margins are also necessary for variation effects such as temperature hot-spots, circuit-aging and process-variation [63]. However, system-margins are typically stressed most due to LdI/dt or inductive transients. Their fast-moving nature [21][44] renders them difficult to compensate for using traditional adaptive techniques.

or isolated dI/dt events, periodic current modulations at the 1st order resonance-frequency reinforce the resonant noise even further [63], thereby maximally stressing system-margins.

Commercial Electronic Design Automation (EDA) tools [5][6][15] cannot accurately model the time-varying CPU current due to the complex hardware/software interactions, particularly in multi-core configurations [10]. Consequently, design-time PDN optimization is inadequate and post-silicon characterization is essential for margining production systems [10][47]. Post-silicon characterization typically relies upon synthetic virus workloads, referred to as dI/dt stress tests (dI/dt viruses) [10]. Due to the inherent complexity of manually crafting these tests, previous work [10][41][42] introduced frameworks for automated generation of stress tests based on optimization techniques such as Genetic Algorithms (GA). These approaches rely upon the capability of the platform-under-test to support high-bandwidth monitoring of on-chip voltage rails or direct voltage measurements.

There are two main approaches for direct voltage measurement: 1) specialized on-chip circuitry integrated into the system at design-time [2][31][46][48][53][72] and 2) voltage sense pins located on the package [3][36][42][66] (also known as Kelvin measurement points). Unfortunately, these capabilities are not yet mainstream features, particularly in cost- and resource-constrained mobile platforms. Moreover, on-chip approaches incur the Non-Recurring Engineering (NRE) design-time costs for hardware development and system integration. In cases where the voltage monitor is integrated into the system as a peripheral device, it requires additional software support (in terms of a device driver) to configure, calibrate and query. Alternatively, on-package measurement points directly connected to on-chip voltage rails do not incur design-time NRE overheads. Nonetheless, they require a dedicated pair of Controlled Collapsible Chip Connection (C4) [22] bumps for each voltage-domain. This consumes valuable C4 resources that could otherwise be used for direct power-delivery. Consequently, such support is not usually provided in resource-constrained platforms such as mobile CPUs (e.g. the Cortex-A53 CPU used in this thesis).

In this thesis, we propose an alternative approach for post-silicon PDN characterization. The proposed approach relies upon sensing CPU electromagnetic (EM) emanations using an antenna and a spectrum analyzer connected to the antenna. Compared to direct-measurement, our approach offers the following unique advantages for resonant voltage noise analysis: a) is non-intrusive, as no physical connection to the CPU is required, b) has zero-overhead, as it does not require design time, development effort, on-package and on-chip resources, and c) is cross-platform, as it can be applied to virtually any platform.

Due to its general applicability, we believe that our approach provides a fundamentally new way for benchmarking commercial systems and at the same time democratizes PDN characterization and voltage noise research. Voltage noise visibility is not typically available in motherboards and researchers do not usually have access, when available, to proprietary on-chip voltage noise circuits. Consequently, voltage noise visibility is limited to chips and motherboards that expose high bandwidth voltage measurements points. The proposed EM methodology removes these constraints by allowing basic PDN characterization to be performed on any CPU and motherboard without the need for direct fine-grained voltage measurements.

1.2 Contributions

This thesis explains the theoretical basis and provide conclusive evidence for the correlation between on-chip voltage noise and emanated EM power. Our measurements demonstrate that both on-chip voltage noise and EM signal power are maximized at the 1st-order resonance frequency. We leverage this observation to propose a convenient, zero-overhead, cross-platform and non-intrusive way for PDN characterization. We demonstrate that with the proposed EM approach, it is possible to: a) monitor periodic voltage-noise of large amplitude b) generate dI/dt stress tests within a GA framework that optimizes towards a maximum EM signal amplitude, c) rapidly measure the 1st-order resonance frequency, d)

detect resonance frequency shifts due to capacitance changes in multi-core configurations, e.g. due to dynamically switching on or off cores in a CPU cluster, e) construct Dynamic-Voltage-Scaling (DVS) governors that are based on real-time EM monitoring and the minimum-operational-voltage (V_{MIN}) of the EM dI/dt viruses; these DVS governors provide at least 10% power-savings over nominal voltage settings.

Furthermore, this thesis establishes the cross-platform applicability of the EM approach by successfully applying it to different CPUs and Instruction Set Architectures (ISA). We characterize the PDN for individual CPUs across separate platforms and distinct processor-clusters integrated on the same die. In particular, the EM methodology is applied on five different CPUs: two ARM multi-core CPU clusters (dual-core Cortex-A72 and quad-core Cortex-A53) hosted on a Juno Board [7], one x86-64 AMD desktop CPU (Athlon II X4 645) and two Ampere Computing X-Gene ARM server CPUs (X-Gene2 8-core and X-Gene3 32-core). Thus, the proposed approach is shown to work across CPUs of different market segments (mobile, desktop and server), different ISAs (ARM and x86), different CPU micro-architectures, different technology nodes and on CPUs that do not offer direct voltage measurements such as the Cortex-A53 and the X-Gene2 CPU. The efficacy of the EM approach is validated through direct voltage measurements (where it is feasible) and V_{MIN} determination (minimum stable operational voltage for a given frequency).

Finally, for the experimental needs of this thesis, the GeST (Generating Stress-tests) framework has been developed. GeST is a GA based framework for generating stress-tests. The EM methodology utilizes GeST for generating dI/dt viruses. GeST framework's source code has been released to public. While GA based automatic frameworks for generating stress-tests is not a novel concept, GeST is to the best of our knowledge the first publicly available framework that researchers and practitioners can use for generating stress-tests. The main strengths of GeST are its flexibility and extensibility as GeST can be extended to support virtually any optimization metric, any measurement instrument and any target

platform. This renders GeST an ideal platform for researchers to experiment with and build upon.

1.3 Document Structure

The remainder of this thesis is the following. Chapter 2 discusses the theoretical background of this thesis, this includes among other subjects: a) CPU energy-efficiency, b) PDN and voltage-noise, c) dI/dt and power-viruses, d) V_{MIN} characterization, e) prior work related to exploitation of EM emanations, and f) related work on GA stress-test generation frameworks.

Chapter 3 presents and evaluates the GeST framework. We delve into the architecture, implementation and usage details of the GeST framework. We also demonstrate the framework's effectiveness by generating power, IPC and dI/dt viruses on several CPUs.

Chapter 4 proposes the EM methodology. Specifically, Section 4.1 provides the experimental apparatus required for the proposed approach, Section 4.2 provides both the theoretical explanation and the experimental findings that prove the correlation between inductive PDN voltage-noise and EM radiation. Sections 4.3, 4.4 provide the methodologies for resonance-frequency detection, and, Section 4.5 provides the methodology for dI/dt virus generation where we utilize GeST to conduct a GA search that optimizes towards high EM amplitude.

Chapter 5 presents the experimental platforms where we evaluate the EM methodology. Chapter 6 evaluates the EM methodology by detecting the resonance-frequency and generating dI/dt viruses for the Cortex-A72, Cortex-A53, AMD Athlon II X4 645 CPU, X-Gene2 and X-Gene3 CPUs. Chapter 7 proposes and evaluates on the X-Gene2 CPU two DVS governors based on the EM methodology. Finally, Chapter 8 concludes this thesis.

Chapter 2. Background and Related Work

This Chapter provides this thesis' theoretical background and related work. The Chapter starts by discussing the factors that contribute to the CPU power-consumption and common techniques for improving the CPU energy-efficiency such as power-gating, clock-gating, DVFS, DVS etc. Then it discusses the fundamentals of PDNs and the voltage-noise phenomenon. Voltage-noise is one of the main subjects of this thesis. Therefore, we will delve deeply into related work on mitigating voltage-noise. We also provide background information on stress-tests (dI/dt viruses, power-virus, IPC viruses), V_{MIN} characterization and all the prior work related to GA frameworks for stress-test development.

This thesis harnesses EM emanations for PDN characterization. This approach has been inspired from prior work that exploits EM emanations for other purposes (e.g. performance-profiling, security exploits etc.). Therefore, in this Chapter we also discuss related work in exploiting CPU EM emanations.

2.1 CPU Power and Energy-Efficiency Techniques

Figure 1 illustrates the components of the CPU power-consumption. The CPU power-consumption is attributed to two major factors: a) the static power-consumption, and b) the dynamic power-consumption. The static power contribution comes from the leakage current. Leakage current is always consumed regardless of the CPU activity. In contrast, dynamic power-consumption depends on CPU activity. The dynamic power is composed of: a) the voltage-frequency (these are mentioned together because their values are intertwined) and b) the switching activity. The switching activity is equal to the activity factor multiplied by the capacitance.

A lot of work has been performed for improving the energy-efficiency and reducing the power-consumption of multi-core general-purpose CPUs. The techniques for reducing the power-consumption can be broken down into three categories: a) techniques that deal with static power, b) techniques that deal with dynamic power consumption through switching-activity reduction, and c) techniques that deal with dynamic power consumption through voltage and frequency reduction.

$$P_{\text{static}} = V * I_{\text{leak}}$$

$$P_{\text{dynamic}} = V^2 * F * C * A$$

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}}$$

V (voltage) F (frequency) C (capacitance) A (activity-factor)

Figure 1. Power equations.

2.1.1 Eliminating Static-Power

First, we discuss static power-consumption reduction techniques. Power-gating is probably the most well-known static power elimination technique. Power-gating is performed by detecting idle transistors and turning them off (essentially disconnecting them from the PDN). Power-gating can be applied at the level of a cache line, a functional unit, a whole core or even a whole CPU cluster (e.g. in the case of big. Little chips). Power gating can be detrimental to energy-efficiency if it is not performed at the right time. The gating must be applied long enough to justify the delay and the dynamic-power spent to power-back the transistors. In a typical power-gating implementation a mechanism counts the idle time of a functional unit; if a threshold idle time is reached, then, the functional unit enters a power-gated state. Power-gating has more power saving potential on floating points units compared to integer units. The reason is that integer units are more utilized, hence, they offer less power gating opportunity. Power gating a whole core or CPU cluster can be

instructed from OS using the ACPI interface, whereas power-gating of functional units is performed within the hardware. The power gated states as defined by ACPI interface start from C6 [83] state and above.

Caches are perhaps the most promising component for applying power-gating. Previous work [84] shows that most of the time (80% of workload execution) cache lines are dead. Dead means that they will not be used again. This motivates the concept of cache decay which basically is a mechanism that tries to guess whether a cache line is dead or alive. Dead caches can be power gated to save power. Of-course cache decay again incurs the risk of wrong prediction. A wrong prediction might cause an otherwise unnecessary cache miss that will have detrimental impact on both performance and energy.

The drowsy caches is an alternative approach for reducing static power consumption. This approach may not have the same power saving potential as cache decay but is less prone to detrimental effects. Drowsy caches exploit the fact that to preserve a cache line's state a lower voltage can be applied. Of course, when the cache line must be accessed the normal voltage has to be applied back. But during times when a cache line is not accessed, to save static power the voltage is lowered just enough to preserve the cache line values. The performance penalty of the drowsy cache mechanism is due to the latency to scale back to normal voltage. The performance penalty is very small (1%), therefore, related work proposed to put all the cache periodically regardless of usefulness into the drowsy mode for substantial energy savings [84].

2.1.2 Eliminating Dynamic Power Consumption (Switching Activity)

The next power saving techniques category to be discussed is related to switching activity reduction. Clock-gating is the most common switching activity reduction technique. The motivation behind clock-gating is the following. During a CPU cycle a functional unit might be idle i.e. no instruction is scheduled to execute on that functional unit. Still, switching activity is wasted even when functional units are idle because of the clock signal

causing unwanted bit toggling. Clock-gating prevents unnecessary switching due to clock signal. This is achieved by placing an AND gate between the clock signal and the flip-flops. The AND operation is performed on the clock signal and a control signal. The control signal is equal to 1 to allow the clock to reach the flip flops, unless a functional unit is idle for 1 cycle or more. In that case the control signal is equal to 0 to not allow the clock to reach the flip flops. Clock gating can be performed at various granularities. Clock-gating is commonly used in functional units, the power savings are less compared to power-gating, but clock-gating is more performance friendly since the latency for returning to operational state is negligible. Clock gating can be performed for the whole core in various degrees as defined by ACPI C1-C3 states. Like for the power gating C6 state, OS can orchestrate the core C1-C3 states transitions. Researchers have proposed even more finer grain clock-gating by gating unused bit width of functional units [84]. The motivation stems from the fact that registers are very wide 32 and 64 bit, but most instructions can be satisfied with only 16bits.

2.1.3 Eliminating Dynamic Power Consumption (Voltage-Frequency Reduction)

Next, techniques for reducing voltage and frequency are discussed [84]. Nearly, all modern general-purpose CPUs and GPUs use Dynamic voltage and frequency scaling (DVFS) to save energy. The motivation behind DVFS is that many workloads have significant slack that can be exploited to run the workload at a slower speed and save energy at the same time. Input/output (I/O) and memory latency dominated workloads provide good opportunity for DVFS. Let us take as example a text processor application. This is a relatively easy computational application with a lot of I/O and relatively relaxed response-time requirements (a person types about a character every 200-300ms) that does not justify operating the CPU at the highest CPU frequency all the time. Instead, we can lower the CPU frequency and the voltage to save significant amount of power (recall from Section 2.1 that power depends quadratically on voltage) while keeping a satisfying performance level and

response-time for the user. Like any other power management technique DVFS poses a risk of detrimental impact in performance. To avoid undesired performance reduction DVFS must be used only when substantial compute slack is available. The most common proxy for estimating compute slack and guiding DVFS decisions is the CPU utilization. Low CPU utilization implies opportunity for voltage and frequency reduction. DVFS decisions are usually guided by OS. Hardware usually exposes various voltage-frequency points (also known as performance-points or P-States) to the OS. The OS communicates to the hardware the desired performance level through the ACPI interface. A commonly used OS level DVFS interface is the `cpufreq` Linux utility.

Lowering only the frequency (DFS) (without lowering the voltage as well) can be beneficial in circumstances where the system is thermally constrained or when workload is dominated by main memory latency. But, for non-thermal and non-memory-latency constrained scenarios, performing DFS most probably is not beneficial for energy-efficiency because the power-consumption decrease is proportional to the workload execution time increase (frequency affects both power and performance linearly). Therefore, for energy-savings we usually use DVFS which scales down both voltage and frequency.

Finally, besides DVFS, lowering only the voltage i.e. DVS can be also applied for dynamic power reduction. Lowering the voltage without scaling the frequency reduces the available voltage noise margin. If voltage drops very low error can happen. Therefore, DVS is a less common practice than DVFS. DVS requires either detection, correction and roll-back mechanisms [34] to deal with potential instability due to low voltage [21], or ability to monitor critical path and adjust voltage accordingly [44].

2.2 Power-Delivery-Networks

2.2.1 PDN Resonance-Frequencies

Figure 2 (a) shows a simplistic representation of the PDN of a die-package-PCB system [20][55]. The current demand due to on-chip switching transistors is modelled as a lumped current source, I_{LOAD} . Explicit decoupling capacitors (henceforth, referred to as decaps) and non-switching, but powered-on, transistors act as localized charge reservoirs that provide the high-frequency component of the demand current, I_{LOAD} . The on-chip power-grid resistance is modelled as a lumped resistor, R_{DIE} , connected in series with C_{DIE} . The total die current (I_{DIE}) is sourced through the inductive power-line traces of the package and the PCB, represented by a series R-L (resistor, inductor) equivalent circuit. The discrete decaps on the PCB and package are represented by an ideal capacitance (C_{PKG} , C_{PCB}) in series with its effective series inductance (ESL) and effective series resistance (ESR). Figure 2(b) shows the input impedance of the distributed RLC network as seen from the die. The impedance spectrum shows multiple resonance peaks due to multiple LC-tank circuits. The highest impedance peak, referred to as the 1st-order resonance peak is attributed to the die-capacitance (C_{DIE}) interaction with its counterpart inductance (L_{PKG}). The 1st-order resonance also occurs at the highest frequency (50MHz-200MHz) compared to the 2nd- (~1-10MHz) and 3rd-order (~10KHz) resonances that are due to downstream capacitor networks.

The resonance frequencies also manifest in the time-domain when the PDN is excited by a step-current excitation (Figure 2 (c)). Micro-architectural events such as branch mispredictions [20] can trigger these oscillations in the PDN. Power-supply oscillations of larger magnitudes can be set off within the supply network due to sustained program activity with alternating periods of high-current and low-current consumption within a loop (e.g. due to dI/dt virus) [42][73]. When the frequency of the time-varying current aligns closely with the 1st-order resonance frequency, voltage oscillations are maximized in amplitude.

High voltage oscillations can lead to bit-flips in arrays, timing errors in logic paths [42][65][66][73] and reliability issues due to gate-oxide stress [1][65]. Such periodic events often result in system/application crashes and/or incorrect execution output [42][56].

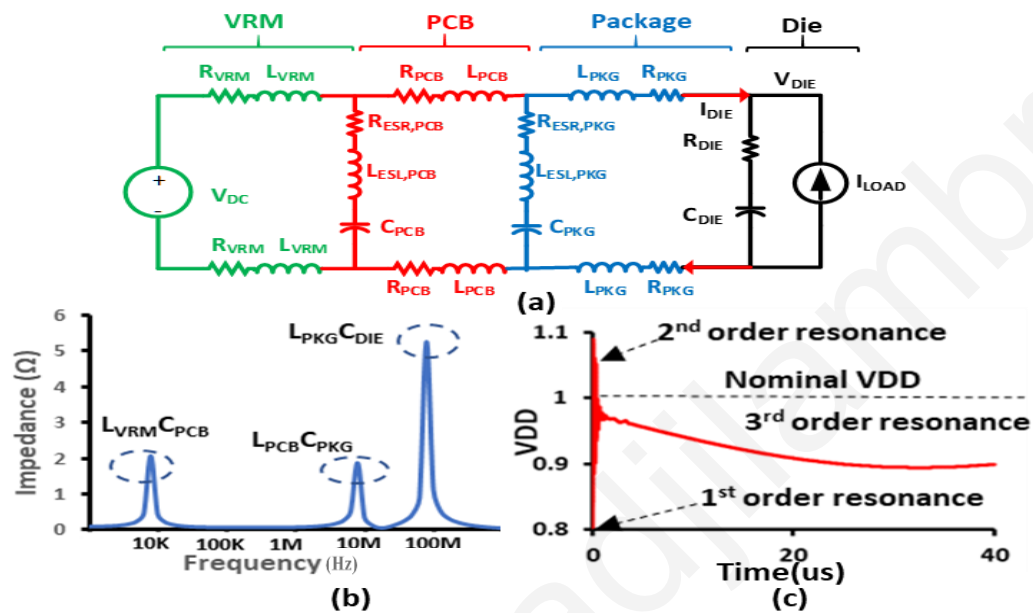


Figure 2. (a) A simplified model of the PDN [55]. The impedance as seen by the die has multiple resonance frequencies, shown in the frequency-domain response in (b) and time-domain response to a step-current excitation in (c)

2.2.2 IR drop versus di/dt noise

This thesis focuses on the di/dt voltage-droop (or AC droop) but the total voltage-droop magnitude does not solely depend on the di/dt effect. The magnitude of a voltage-droop is the sum of the di/dt and the IR droop (or DC droop) [10]. As illustrated in Figure 2 a PDN includes resistive and inductive PDN components. The resistive PDN components (R_{VRM} , R_{PB} , etc.) produce the IR drop, whereas the inductive components (L_{VRM} , L_{PCB} , etc.) produce the di/dt droops. IR drops can be mitigated with techniques that compensate the voltage drop by dynamically adding a voltage-offset during runtime. A well-known technique for dynamic voltage-offsetting is the Load-Line calibration. Load-Line calibration is available on most high-performance PCBs. In contrast to IR droops, di/dt droops due to their fast-

moving nature cannot be compensated with Load-Line calibration. Other more advances techniques must be employed (that are discussed in the next Sections of this Chapter).

IR and dI/dt voltage droops are stimulated by different factors and they are generated by different workloads. IR drop increases proportionally to the CPU power-consumption. Broadly speaking, higher CPU frequency and higher current draw lead to higher IR drop. Contrary, to raise the dI/dt droop, repetitive high current stimulus at a rate equal to the 1st order resonance-frequency is required. Therefore, increasing the CPU frequency while executing a resonant workload will not necessary increase the voltage-droop.

To illustrate these differences we run and measure the voltage droop caused by two different workloads: a) a power-virus that targets maximum power-consumption, and b) a dI/dt virus that targets the PDN resonance-frequency by causing sudden high current-draw at a rate equal to the resonance-frequency. We perform this experiment on a Cortex-A57 CPU hosted on an ARM Juno board. This CPU supports measuring dI/dt voltage-droops through a high-bandwidth On-Chip-Digital-Storage-Oscilloscope (OC-DSO). We run the two workloads for different CPU frequencies. At each frequency we measure the current draw and the voltage-droop. Figure 3 shows the power-virus' results and Figure 4 shows the dI/dt virus' results.

First and foremost, the figures confirm two fundamental expectations: a) that the power-virus draws significantly more current than the dI/dt virus and b) that the current consumption grows linearly to the CPU operating frequency for both workloads. Regarding voltage-droop, the measured voltage-droop for the power-virus shows linear dependence with the CPU frequency. This is the case because the power-virus exercises the IR drop. In contrast, the dI/dt virus measured droop does not show linear dependence to the CPU frequency. The dI/dt virus exercises the dI/dt droop. Hence, the measured voltage droop increases with CPU frequency until the resonance-frequency is achieved. This occurs at 1.2GHz, this is the operating CPU frequency at which the dI/dt virus stresses the resonance-

frequency. Further increase in the CPU frequency moves the virus's stress-frequency beyond the 1st-order resonance. This causes a reduction in the measured voltage droop.

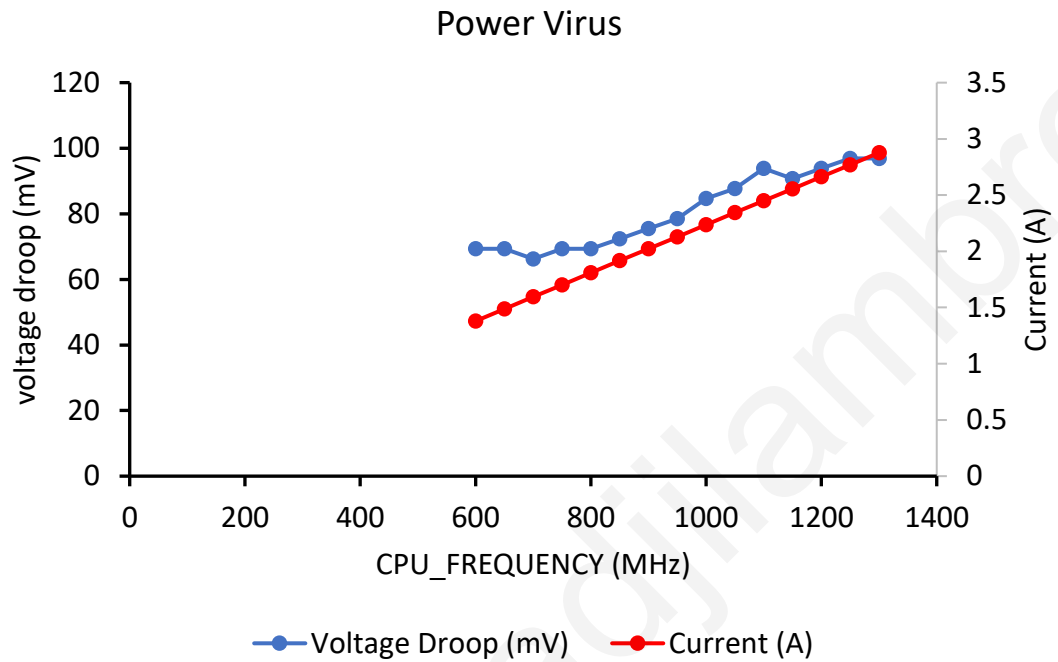


Figure 3. Voltage droop and current consumption for a power virus at various CPU frequencies.

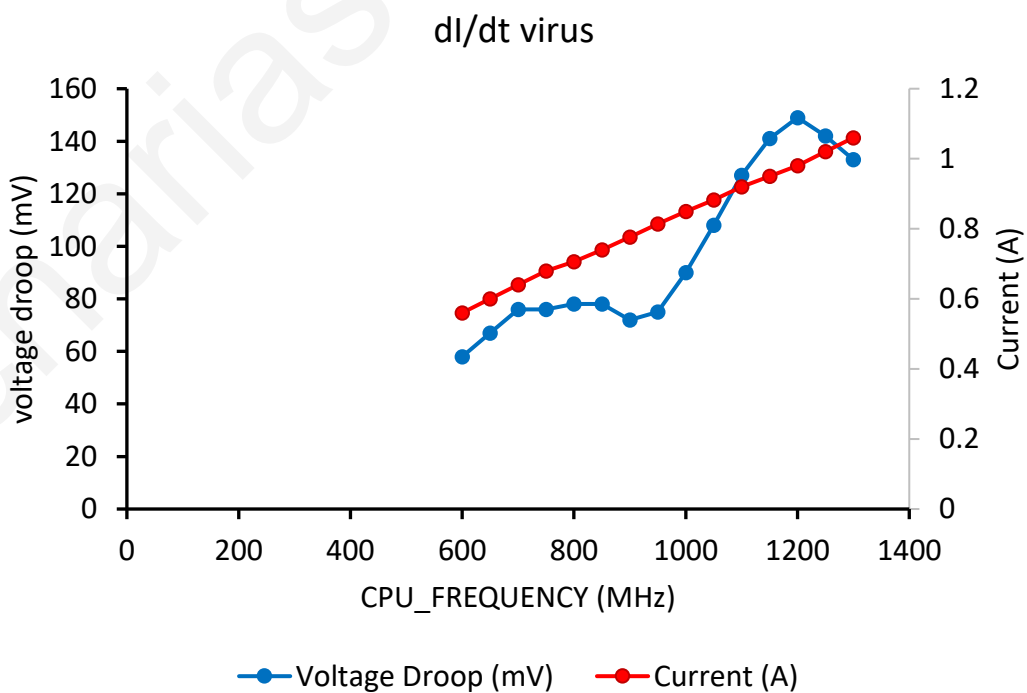


Figure 4. Voltage droop and current consumption for a dI/dt virus at various CPU frequencies.

2.3 Voltage Margins

The classic approach for dealing with voltage-noise is adding a voltage margin i.e. running the chip at a higher voltage to provision for worst case voltage droops [63]. This is very practical for ensuring robust execution, but it leads to unnecessary waste of power. Usually the voltage margins are much more pessimistic than required. Manufacturers are forced to set pessimistic voltage margins because workload variations render the in the field voltage-droop magnitude very hard to predict. Moreover, due to hardware static variations, some parts may have higher tolerance to voltage-droops than others. Therefore, a voltage-margin that ensures robust execution for a slow part, limits the energy-efficiency of a faster part. Ideally, each chip should adopt different margins.

A partial solution to the issue of pessimistic margining is the speed-binning (or frequency-binning). Speed-binning refers to making different product models out of the same chips [63]. The chips are categorized to speed-bins (frequency-bins) after running various representative test workloads on all chips. Broadly speaking the product models will differ on their advertised nominal operating frequency (e.g. Model X runs at 3GHz, Model Y runs at 2.8GHz etc.) and on their price. With speed-binning the faster chips are not forced to work at slower frequencies and this partially solves the energy-inefficiency issue. But speed-binning does not tackle completely the issue of static variations as even in the same frequency-bin static variations still exist.

Constructing dI/dt viruses and characterizing their V_{MIN} is another approach for reducing the pessimistic margins [10]. As we show in this thesis and in prior work [42], the V_{MIN} of a proper dI/dt virus is higher than the V_{MIN} of conventional workloads, thereby, the dI/dt virus's V_{MIN} can be used as a guideline to determine the operating voltage for a given frequency.

2.3.1 V_{MIN} Characterization

V_{MIN} is the minimum voltage at which a chip (CPU, GPU etc.) operates correctly for a given frequency. A chip can have different V_{MIN} per workload as the execution paths and the voltage droop magnitude differs from workload to workload. To determine a workload's V_{MIN} we perform V_{MIN} characterization. This procedure involves running the workload for various voltage values until instability is observed. Typically, the V_{MIN} characterization starts from high voltage and after each successful workload execution, the voltage is lowered in increments of 10mV. The V_{MIN} test stops at the first voltage where instability is observed (SDC, crash etc.). This voltage is tagged as the crash-point. The V_{MIN} is equal to the crash-point plus 10mV (or any other voltage increment that is selected). Figure 5 illustrates the V_{MIN} characterization flow.

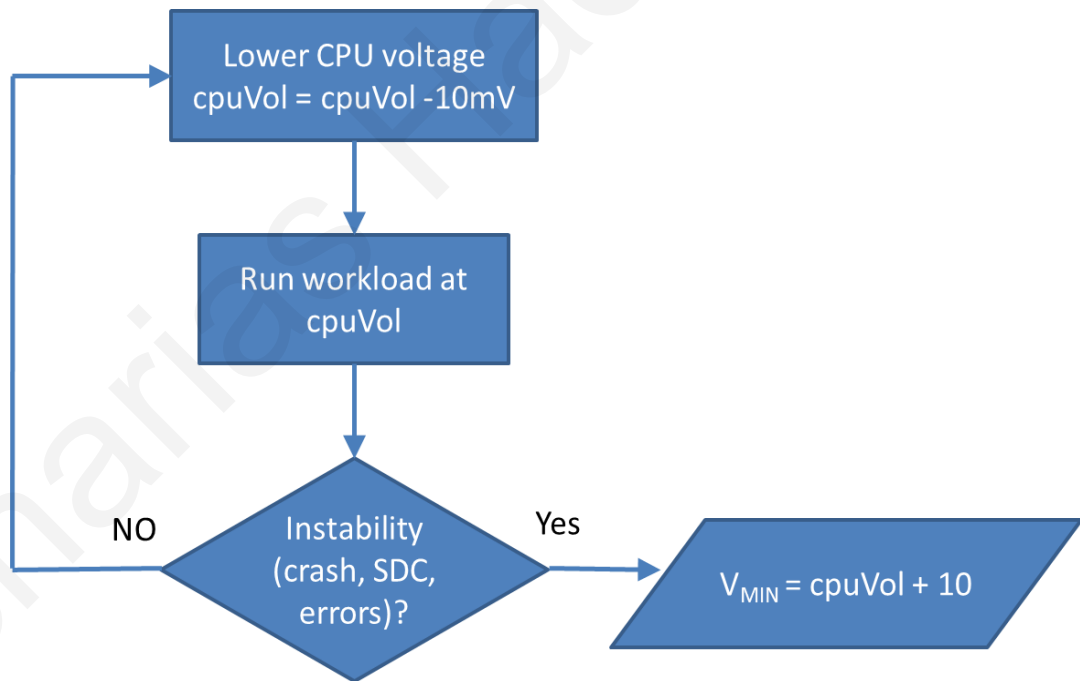


Figure 5. V_{MIN} characterization flow.

2.4 Voltage Margin Elimination Techniques

A voltage margin elimination approach is the critical path monitoring (CPM) [44]. CPMs are on-chip sensors that can measure at real-time the CPU's available timing margin. CPM can be combined with an actuator that can adjust the voltage or operating frequency. For instance, when the available timing margin is high, then the actuator can reduce the operating voltage to reduce the power or increase the CPU frequency to improve the energy efficiency. If the timing margin is low, then the CPU frequency should be decreased, or the voltage should be increased to ensure stability. Even in the presence of this technique, a fixed voltage margin for the worst-case voltage droops must be maintained to ensure stability in production setups [44]. Therefore, in practice this timing guard-band scheme adjusts the voltage (or frequency) depending on workload's voltage noise magnitude to maintain a fixed margin. This is a more energy-efficient approach compared to the conventional approach of maintaining fixed operating voltage, because with fixed operating voltage a lot of power is wasted for common workloads that do not cause high voltage-noise.

A more aggressive approach for voltage margin elimination is Razor [21]. Compared to CPM, Razor can provide higher energy savings, but it also incurs higher area and penalty overhead. Razor is based on running the chip at aggressive voltage/frequency conditions and relying on error-detection and roll-back mechanism to recover the pipeline from errors. Razor replicates critical path flip-flops. One flip-flop operates at aggressive margin conditions and provides the speculative value. The other flip-flop operates at safe conditions and provides the golden reference. If the flip-flop values do not match, then a recovery mechanism for bringing the pipeline to the correct state is started.

2.5 LdI/dt Noise Mitigation Techniques

The most common approach for mitigating dI/dt voltage-noise is to add on-chip capacitance [55]. Capacitance affect voltage-noise in two ways. First, higher capacitance helps in mitigating the voltage droop magnitude. Adding decoupling capacitors is a known design-time technique to reduce voltage droop magnitude [63]. This approach has its limitations though, as cost, area overheads and leakage limit the amount of capacitance that can be added. In general, well connected PDNs that connect many hardware components, and, thus, incorporate higher capacitance are considered beneficial for voltage noise reduction [37]. The second effect that capacitance has on voltage-noise is that higher capacitance has inverse relationship with resonance-frequency [55]. Therefore, the resonance-frequency decreases with more capacitance. In terms of voltage-droop magnitude, shifting the resonance-frequency can be either beneficial or detrimental depending on which frequencies a workload exercise.

The vast majority of voltage droops is caused by periodic activity of loops [33][85]. If these loops happen to match the PDN resonance-frequency, then the voltage noise is amplified even further. Therefore, authors in [58][59][60] examined ways to modulate loop frequency to avoid resonant voltage-noise. The proposed approaches require the capability to monitor current at high bandwidth. Monitoring current is more advantageous over monitoring voltage in this case, because resonant current behavior precedes voltage noise. The authors exploit a phenomenon called “maximum repetition tolerance” which is the maximum resonant current repetitions that can happen before a large voltage emergency is triggered. The maximum repetition tolerance helps preventing unnecessary reactions to voltage droops that are not dangerous and at the same it helps ensuring that a mitigation mechanism will be engaged before it is too late. For mitigating voltage droop when detected, stalling the pipeline by stalling instruction issue or instruction fetch has been proposed by the authors. These mitigation approaches incur a performance penalty each time an

emergency is detected. Other authors [39] have proposed to clock-gate or turn-on functional units on purpose to reduce or increase current consumption. Turn-on can be used to compensate voltage droop and clock-gate to smooth-out voltage overshoot.

The adoption of multi-core CPUs compared to single core CPUs raises the dynamic power variability and hence increases the susceptibility to voltage transients. But at the same time, it gives opportunity for voltage noise mitigation with destructive scheduling [66]. Essentially, some workloads when co-scheduled on different cores of a multi-core CPU have a damping effect on voltage noise. Other workload combinations amplify voltage noise. The key is to find which workload combinations decrease the voltage noise and utilize this information in scheduling decisions. Workloads can have destructive voltage-noise interference because their current consumption does not align. Destructive interference can be also applied within multi-threaded application by misaligning the threads synchronization barriers [49]. Other work performed in a modern multi-core IBM server processor that supports droop mitigation mechanisms has shown that the server efficiency is limited by the IR drop [75]. Therefore, they suggest in multi-socket servers, to evenly balance the workload across the CPU sockets to minimize IR drop for better energy efficiency.

Using ECC feedback as a proxy of voltage noise and emergency conditions has been proposed in [9]. The idea is to lower the voltage until ECC errors start to occur. This approach applies only to chips that have error protected ECC pipelines. Other work shows that predicting safe V_{MIN} through workload performance counters signature is possible [56]. Another prediction scheme [63] proposes voltage noise prediction based on sequences of micro-architectural events and instruction that are likely to cause voltage noise (e.g. branch instructions, pipeline flushes etc.). A micro-architectural structure inspired by branch predictors but for prediction of voltage noise emergency is proposed. This mechanism also relies on a voltage sensor for training the predictor as well as a roll-back mechanism in case

of wrong emergency prediction that causes CPU malfunction. A voltage emergency misprediction in such configuration induces performance penalty.

Voltage droop mitigation circuits have been proposed and implemented on some processors [14][26][31][43][46][62][74]. A common mitigation circuit setup includes: a) an on-die voltage monitor sensor, b) an actuator that mitigates the voltage noise either by pipeline throttling (stall instruction issue or fetch) or reducing the CPU frequency (e.g. adaptive-clocking [14]) and c) a voltage threshold indicating when the actuator should be enabled (also in case of frequency modulation the amount of modulation is expressed). Setting the right threshold can be tricky. A conservative (pessimistic) threshold might lead to performance penalty whereas an optimistic threshold might translate to instability.

2.6 dI/dt noise on GPUs

Besides CPU voltage noise, voltage on GPUs has been studied as well. In fact, GPUs may suffer more from voltage noise due to their higher current consumption. Contrast to CPU research, most GPU voltage noise research has been performed with the help of GPU voltage noise simulators e.g. the GPUVolt simulator [86]. This might be an indication that interfaces for direct voltage noise measurements on GPUs are seldomly found. Generating high inductive noise on GPUs should be easier compared to CPUs. The GPU cores are much simpler compared to complex general-purpose CPU out-of-order superscalar pipelines. Moreover, GPUs have much more support for synchronizing core execution which assists the generation of large dI/dt swings. Therefore, researchers have shown that dI/dt viruses can be achieved by relatively simple manual written code that activates and deactivates functional units with periodicity that matches the resonance-frequency or using instructions that are known to cause low and high-power consumption [70][87].

Similar techniques to CPU have been proposed for voltage noise mitigation on GPUs. GPUs due to high core counts are susceptible to static variations. Researchers have proposed

the use of CPM combined with clock gating to throttle down cores that do not have enough timing margin [87]. Other work reaches to the interesting conclusion that on GPUs normal workloads have higher 2nd order droop compared to 1st order droop [45]. This is the case because the 1st order droop is more sensitive to miss-alignment of cores. In fact, alignment with so many cores (>100) at ~100MHz is hard to achieve. Whereas events as second order droop around 1MHz are aligned easier. They have identified that the main events that cause 1st order droops for GPUs are dispatch stalls and register file current variations. Events that cause 2nd order droops are instruction and data cache miss stalls. For droop mitigation they suggest blocking dispatch unit and delaying system wide activity. Last, related work has examined the GPU behaviour at low voltages. The researchers have shown that performance counters can be utilized for workload V_{MIN} prediction [88].

2.7 EM emanations exploitation

Previous work has exploited EM radiation for various objectives. EM emanations are a known security side channel for snooping information [16][17][18][29]. Other work leverages EM for non-malicious uses. In particular, [68] proposed non-obtrusive software profiling, [76][77] proposed counterfeit detection and [52] proposed a malware detection scheme based on EM emanations. Our work also leverages EM radiation, but for addressing a different problem: voltage noise and PDN characterization in high-performance system-design. Other work [30] proposes architectural and compiler changes to reduce CPU EM interference.

2.8 Stress-Tests

Broadly speaking, stress-tests can be classified into three categories: a) stress-tests that maximize specific micro-architectural (uArch) metrics, such as memory bandwidth, IPC and cache-misses, b) stress-tests that maximize power consumption and temperature, commonly referred to as “power-viruses”, and c) stress-tests that maximize voltage noise,

also known as “voltage-noise viruses” or “dI/dt-viruses”. This work shows that GeST can successfully generate stress-tests for all three categories. In particular, we use the framework to generate stress-tests that maximize CPU IPC, power, temperature and dI/dt voltage-noise. While this work focuses on the CPU there is nothing fundamental that prevents using GeST for other processor components as well, for instance the last level cache (LLC) or for an integrated accelerator. A brief discussion on each of the three stress-tests categories follows.

2.8.1 Performance Stress-Tests

Stress-tests for maximizing specific uArch metrics are mainly useful for performance benchmarking purposes. The AIDA test suite [91] is a good example of benchmarking stress-test software that is commonly used to test desktop and mobile system’s performance. This suite includes various benchmarks to test the performance of specific CPU units (e.g. floating-point unit) and specific functions (e.g. hashing). It also includes memory latency and read/write bandwidth tests as well as specific test benchmarks for GPUs and disks. Besides performance testing, previous work has proposed using stress-tests that target specific CPU parts (ALU, FPU, L1D, L1I, L2 and L3 caches) to characterize the CPU minimum operation voltage (V_{MIN}) [92] and to generate power-models and an energy-per-instruction (EPI) profile [90].

2.8.2 Power-Viruses

Power-viruses maximize both sustained power consumption and heat-dissipation [57]. They are useful for characterizing a system’s power and thermal margins as well as the IR drop. In addition, they can check thermal stability, in particular, of overclocked systems (set to run at a higher than nominal voltage and frequency). Power-viruses usually maximize the micro-architectural activity by issuing many instructions per cycle [40]. Prime95 [61] is a well-known test program that maximizes power consumption and it is often used to check the stability of over-clocked CPUs.

2.8.3 dI/dt stress-tests

Voltage-noise viruses attempt to maximize CPU voltage fluctuations [10][41][42] and they have different characteristics from power-viruses. Rather than keeping a sustained high current (I) consumption, dI/dt stress-tests attempt to cause sudden transition from very low to very high current consumption. Abrupt current increase causes the voltage to drop low. Periodic current surges that match the CPU's PDN 1st order resonance-frequency maximize the CPU voltage droops and overshoots [10][41][42] like shown in Figure 6.

Since low voltage operation may lead to malfunctioning [42][56], dI/dt viruses are very effective timing-error stability-tests. Voltage-noise viruses typically cause higher voltage drop than power-viruses because the dI/dt component dominates over the IR drop. The lowest voltage at which a dI/dt virus runs correctly can provide a good indication of where to set the operating voltage of the CPU (for a given operating frequency).

Typically, a dI/dt virus is a loop of assembly instructions fine-tuned to cause current variations at a rate equal to the PDN's 1st order resonance-frequency. To achieve this the virus should have the following traits: a) the loop should be executed repeatedly, b) a current surge is caused at each loop iteration, and c) the loop iteration length in cycles is equal to the CPU cycles that can fit inside a resonance period ($1/\text{resonance frequency}$). To develop dI/dt viruses high bandwidth voltage measurements are required to measure the maximum voltage droop or the maximum peak-to-peak voltage swing. This is achieved either through external oscilloscope connected to on-package voltage sense points or internal on-chip voltage sensors. Previous work [41][42] has proposed generating dI/dt viruses with a GA that maximizes voltage-noise through on-package voltage sense points. This thesis also uses GA to maximize voltage-noise but through maximizing EM emanations amplitude.

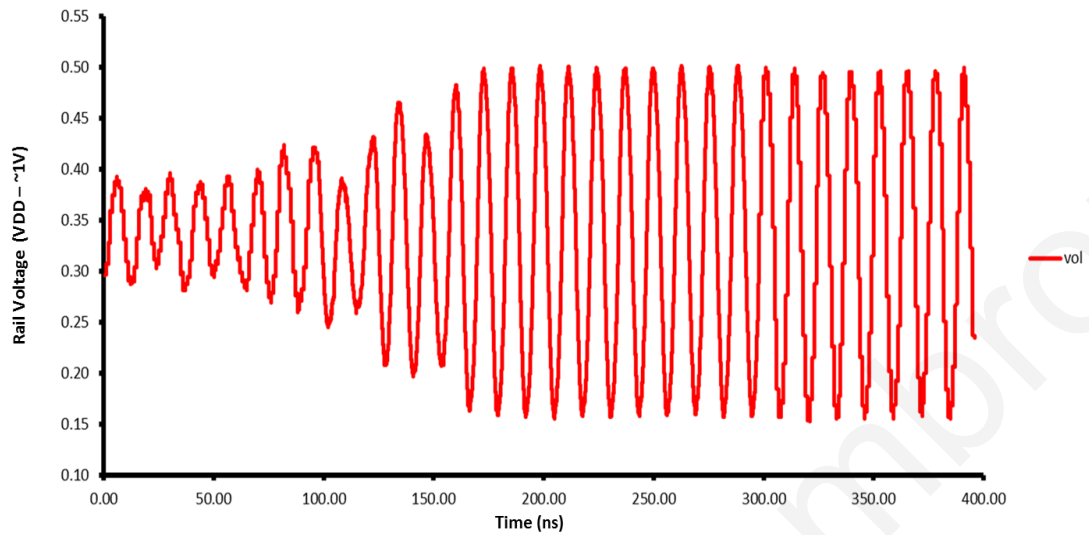


Figure 6. Resonant buildup that leads to very high voltage droops and overshoots during dI/dt virus execution on a x86 CPU. Measured with external Oscilloscope and active differential probe connected to on-package measurement pads.

2.9 GA for Stress-Test Generation

Table I provides an overview of the state-of-the-art GA stress-test generation frameworks. We consider the pairs of works [41][42], and [27][89] as each representing the same framework. Particularly, the work in [41] has evaluated a dI/dt GA framework on a simulated environment and subsequently on real multi-core hardware [42]. Similarly, the work in [89] generates GA power viruses for single-core CPUs and a latter extension on multi-core CPUs [27]. In a different line of work, Joshi et al. [40] evaluated a power-virus GA framework on an Alpha ISA single-core simulator and Polfiet et al. [57] evaluated a power-virus GA framework on real-hardware using x86 multi-cores.

Table I. Comparison of related work on GA frameworks.

Framework	Optimization Type	Optimization Language	Evaluated-On	Metrics Evaluated	Component Stressed	References
AUDIT	Instruction-Level	x86 ISA	Real-Hardware / Simulator	dl/dt	CPU	[41][42]
MAMPO	Abstract-Workload	SPARC ISA	Simulator	power	CPU+DRAM	[27],[89]
Joshi et al.	Abstract-Workload	Alpha ISA	Simulator	power	CPU	[40]
Powermark	Abstract-Workload	C	Real-Hardware	power	Full-System	[57]
GeST	Instruction-Level	ARM,x86	Real-Hardware	dl/dt,power	CPU	this thesis

As shown in Table I, there are two dominant approaches in designing GA frameworks for stress-test generation: a) based on an abstract-workload model and b) based on instruction-level primitives (usually assembly instructions). In the abstract-model frameworks the individual is a vector of workload related parameters such as instruction-mix, register-dependency distance, memory-stride profile, branch transition rates etc. The GA operators are performed on this abstract workload profile. A workload generator stochastically generates the assembly (or higher-level language) code based on the values of the abstract model parameters. On the other hand, for the instruction-level optimizations the individual is the actual source code of the virus. The GA performs the optimization directly on the source-code and has full-control on the instruction-mix, instruction-order and instructions' operands. GeST as presented in Chapter 3 utilizes the instruction-level optimization approach.

An advantage of the abstract workload model is that it reduces the design space. A disadvantage of the abstract model is that it fails in optimizing the instruction order and the instruction opcodes simply because these parameters are out of GA control. Previous work [90] reports that instruction-order can make up to 17% difference in power for the same activity factor and instruction-mix.

Moreover, knobs typically found in abstract-workload frameworks that allow fine tuning memory accesses and branch behavior, through parameters such as memory stride and branch transition rate, seem not so relevant, at least, for high power and dI/dt workloads. As reported in previous work [40][57] and confirmed from this thesis (Section 3.3) power-viruses are characterized by high IPC, very predictable branches and extremely high L1 hit rates. These characteristics can easily be achieved with instruction-level optimization. Regarding dI/dt optimization, all previous work utilized instruction-level optimizations [10][41][42]. This is the case since dI/dt optimization is very sensitive to the workload frequency that must match the PDN resonance frequency. For such optimization search, instruction-order is more important than disruptive events such as cache-misses and branch-misprediction that cause non-determinism and limit the capacity to control the workload frequency [10].

Another design choice of a GA framework is the optimization language. Most frameworks prefer generating assembly code except [57] that prefers a high-level language like C. The advantage of using higher level language is that it makes the framework versatile to the hardware platform of interest. Using a higher-level language makes sense in conjunction with an abstract workload model. For instruction-level optimization this is not so practical because it prevents GA to directly optimize the instruction type mix and order (the final instruction order and types depend on the compiler). For GeST we prefer the assembly instruction level optimization. The versatility of GeST that allows its use with any hardware platform stems from providing an interface to the experimenter to specify the

instructions that will be used in the optimization. Thereby, this allows the experimenter to use GeST to customize and optimize for any ISA.

Finally, another important GA framework aspect is the component it targets. Most works justifiably target the CPU as it is generally accepted that CPU is the most active and power-hungry component. In [57] authors generated full-system stress-tests that also stressed the network-interface-card and hard-disk. This is achieved by adding a thread that sends network packets and a thread that performs disk reads, the invocation frequency of these threads is a parameter of the abstract-workload-profile. GeST is as an instruction-level optimization framework that primarily targets CPU, but it is also applicable to any other component that can be stressed through a stream of instructions. For instance, with GeST is possible to stress LLC or DRAM by instructing the framework to optimize towards cache-misses and providing in the input file load/store instruction definitions with various strides, base memory registers and various min-max immediate values. We are currently investigating such extensions.

Chapter 3. GeST Framework

This Chapter presents and evaluates the GeST framework. GeST is a GA framework that is developed for the requirements of this thesis. GeST source code has been publicly released [78] and a paper dedicated to the framework is published in ISPASS 2019 [79].

GeST, given a user-specified set of assembly instructions and operands, attempts to find the instruction mix, order and operands that maximize a target metric. GeST is extensible as it offers an easy interface to build upon. A user can define the instructions, which the optimization search uses, by only changing input configuration parameters. This renders the framework compatible with any ISA. Moreover, an experimenter can script custom measurement procedures and custom fitness functions (the function that drives the GA optimization) in a plug-and-play fashion using the template measurement and fitness software classes provided in the framework. The user defined measurement scripts and fitness functions are easy to integrate in the framework by simply changing the configuration parameters without performing any change in the framework's core source code. We demonstrate the power of the framework's extensibility and flexibility by: a) generating stress-tests that maximize different target metrics such as power, temperature, and dI/dt voltage-noise, b) using the framework with various measurement procedures and optimization metrics such as software accessible counters (e.g. performance counters) and external instruments (such as oscilloscopes), c) generating stress-tests on mobile ARM and server-grade ARM and x86 CPUs, d) generating stress-tests on bare-metal and OS execution environments, and e) using both simple as well as complex multi-objective fitness functions.

The rest of the Chapter is organized as follows: Section 3.1 presents GeST architecture and implementation details, Section 3.2 provides the platforms we use to evaluate GeST, Section 3.3 demonstrates the capability of GeST to generate stress-tests that maximize power consumption, temperature and IPC, and, in Section 3.4 we highlight the framework's

capability to generate dI/dt voltage noise stress-tests with the help of an external oscilloscope connected to on-package voltage-sense pins (later in Chapter 4 we use GeST to develop dI/dt viruses by utilizing EM measurements).

3.1 GeST Framework Description

GeST is written in Python 3 and takes as inputs xml files that define configuration parameters. The framework high-level overview is shown in Figure 7. The framework can be broken down into 5 major parts: the inputs, the outputs, the GA engine, the measurement component and the fitness evaluation function. Next, we describe in detail each of these components.

3.1.1 GA Engine

The GA engine is the heart of the GeST framework and coordinates its execution. GAs optimize a target metric by applying operators inspired by natural evolution such as selection of fittest individual for breeding, exchange of genes (crossover) and mutation. Previous work has shown that GAs can generate workloads that stress the system worse or comparably to manually written stress-tests with little human guidance within few hours [27][41][42][57]. Our findings clearly confirm the GA suitability and effectiveness for stress-test generation. A typical GA flow is shown in Figure 8. A short description of each GA step follows:

- **Seed Population:** The first step is to create an initial seed population (generation). The population is a set of assembly instruction sequences. In GA terminology, each sequence of assembly instructions represents an individual of the population. The seed population can be either a new random initial population or a population from a previous GA run. In the case of a random initial population the individuals are randomly generated based on the user-specified instructions, operands and loop-size.

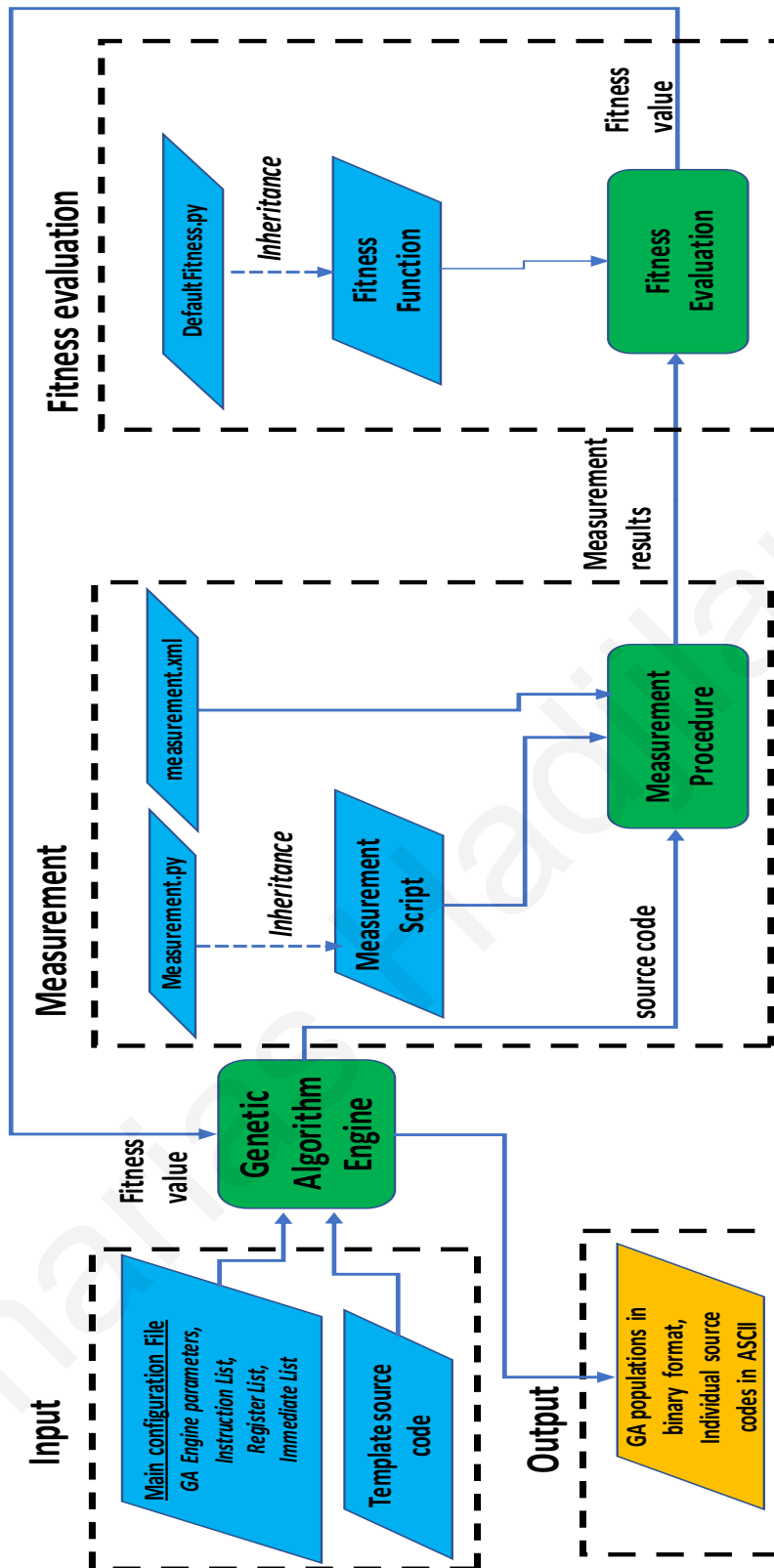


Figure 7. GeST overview.

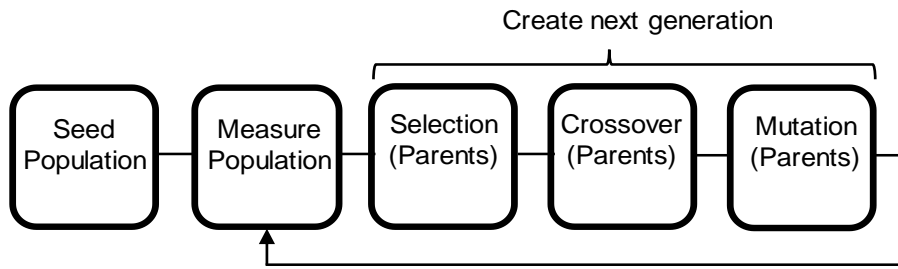


Figure 8. A Typical GA flow.

- **Measure Individuals:** The second step involves compiling each individual, executing the resulting binary, measuring the metrics of interest during the binary execution and assigning a fitness value to the individual. In GeST the user defines the measurement procedure and fitness function as shown in Figure 7.

- **Creating next generation:** The algorithm creates a new population after all individuals are measured. The new population is created by selecting the fittest individuals as parents (e.g. the ones that scored the highest average power), exchanging instructions between the two parents (crossover) and performing mutation. A mutation operation converts an instruction or an instruction-operand (such as a register) into another, with a conversion probability, referred to as the “mutation rate”. For instance, if the mutation rate is equal to 2%, then each instruction has a 2% probability to be mutated.

Figure 9 demonstrates, with the help of an example, how we generate a new population by applying tournament selection, one-point crossover and mutation operators. The procedure demonstrated in the figure is performed repeatedly until the desired population size is reached. Note that for this example each individual consists of only four instructions. First, we randomly pick five individuals from the population and select from them as “parent1” the fittest individual. The same procedure is applied to select “parent2”. Then a random point in the instruction stream is selected for the crossover between the two parents.

In the example the crossover point is the 2nd instruction. This means that the “child1” will inherit the first half from the “parent1” and the second half from “parent2”, while “child2” will inherit the first part from “parent2” and the second part from “parent1”. Finally, the example demonstrates the mutation operator. Mutation can be performed for a whole instruction i.e. the whole instruction is randomly transformed to a new instruction, or an operand of the instruction i.e. an operand is transformed to another operand. For “child1” the r2 register of the SUB instruction transforms to r5, while for “child2” the STR instruction transforms to LSL and the LSL operands are randomly generated.

Table II shows the GA related configuration parameters and their values that we empirically found to work well in our experiments. A key observation from our work is that relatively few instructions are sufficient to stress the CPU. Loop lengths of 50 instructions prove sufficient to cause large power consumption or high IPC. Voltage noise optimization is more sensitive to loop-length because the di/dt noise is highly related to the PDN

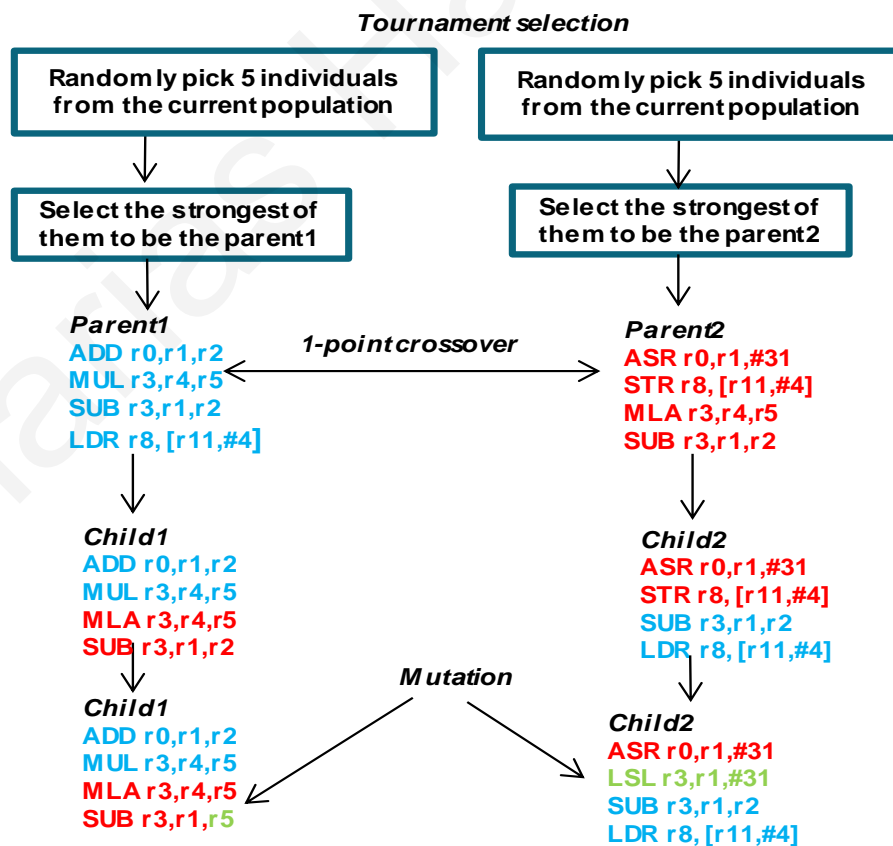


Figure 9. Demonstration of GA operators.

resonance frequency. A rule of thumb that is found to work well for dI/dt noise is to have the loop instruction length equal to $IPC \times \text{clock_frequency} / \text{resonance_frequency}$ (similar to what authors used in [41]). The IPC should be roughly equal to $MAX_THEORETICAL_IPC / 2$. The rationale behind this is that dI/dt should contain low and fast activity phases hence the IPC should be somewhere in the middle (we explain in more detail this heuristic in Section 4.5). In our experience, the aforementioned equation typically results in loop lengths of 15 to 50 instructions. Another recommendation, supported from experimental findings, is that mutation rate should be low enough so that only one or at-most two loop instructions are mutated at a time. Higher mutation rate might impede the GA convergence. So, if the target is one mutated instruction, then for loop lengths of 50 instructions we need 2% mutation rate, for 15 instructions we need 8%.

Also, we have found that optimization search converges faster if children preserve some of the instruction order found in their parents (this is especially true for maximum power and maximum dI/dt search). Hence, to accelerate the GA convergence we prefer one-point crossover that does a better job in preserving the instruction-order of strong individuals compared to uniform-crossover (another well-known crossover operator), where each instruction has an equal probability to be swapped among the parents.

Additionally, we use elitism (automatically promote some of the fittest individuals to the next generation) because according to literature [50] this feature helps GA convergence for most optimization problems. We apply elitism by promoting the fittest individual to the next population. Regarding population size, we use populations of 50 individuals. Authors [50][95] showed that for most optimization problems populations sizes between 50 and 100 individuals are sufficient. Given the relatively high hardware measurement time per individual (in the order of seconds), we choose population size of 50 individuals in order to keep the GA execution time as low as possible. Finally, for selection method we use tournament selection because it is generally considered as one of the most balanced selection

methods in terms of computational efficiency, implementation-ease and GA growth rates [50][96].

Table II. GA parameters.

Parameter	Default Values
population_size	50
Individual Size (number of loop instructions)	15-50
mutation_rate	0.02 - 0.08
crossover_operator	one point crossover
elitism (Best individual promoted to next generation)	TRUE
parent_selection_method	Tournament Selection
tournament_size	5

3.1.2 Inputs

The GeST inputs consists of the main configuration file and the template source code. We describe in detail the format and use of these files. The main configuration file is a xml file that specifies: a) the GA engine related input parameters (population size, mutation rate etc.), b) the instructions and operands used in the GA search, and c) various other parameters, such as, the directory where the results will be saved and the names of the measurement and fitness classes to be used by the GA search. GA engine related configuration parameters (individual size, mutation rate etc.) are explained in Section 3.1.1. The following discussion focuses on how to specify the instructions and operands used by the GA optimization search.

The registers, immediate values and instructions used by the GA optimization are defined in the main configuration file. Figure 10 shows an example of how a user can define an instruction and its required operands. The instruction in the example is the ARM ISA LDR (load from memory). The first required parameter is the instruction name, it is used to

<pre> <operand id="mem_result" values="x2 x3 x4" type="register" > </operand> </pre>	<pre> <operand id="mem_address_register" values="x10" type="register"> </operand> </pre>
<pre> <operand id="immediate_value" min="0" max="256" stride="8" type="immediate" > </operand> </pre>	<pre> <instruction name="LDR" num_of_operands="3" operand1="mem_result" operand2="mem_address_register" operand3="immediate_value" format="LDR op1,[op2,#op3]" type="mem" > </instruction> </pre>

Figure 10. Example of an instruction definition and its necessary operands.

identify the instruction and must be unique. The second parameter is the number of instruction operands. LDR has 3 operands: a) the register where the result will be written, b) the register that holds the base memory address, and c) an immediate value that holds the memory offset. These operands must be separately defined in the same configuration file (also shown in the figure). The instruction definition links to the operand definitions through the operand ids. In our example, the third to fifth instruction-definition parameters define the operand ids which are “mem_result”, “mem_address_register” and “immediate_value”. If the instruction definition contains an undefined operand id, the framework will terminate the execution. In addition, if the instruction definition contains incompatible to the ISA specification operands, then generated instructions sequences that contain this instruction will fail to compile. It is user’s responsibility to provide the right inputs so that the generated instructions sequences will compile and will not crash during execution. The user should define the memory instructions and their operands in a way that prevents random mutation that leads into illegal memory access. To prevent random mutation in our GA runs we set the memory instruction’s base address register to the stack pointer address (this can be done by simply adding an assignment statement in the template source code file before the loop body), then we restrict the memory offset to values that do not exceed the stack size (for

example in Figure 10 the maximum allowed memory offset from the stack pointer address is 256 bytes). With that said, even if some individuals fail due to compilation or execution errors the GA optimization will implicitly discard them because their fitness value will be low. Therefore, as long as these illegal cases are rare, they will not impede GA convergence.

Continuing with the instruction definition parameters, the “format” parameter specifies the instruction format. It prescribes to the framework how the instruction must be printed in the generated output source code. The op1, op2 and op3 keywords in the format specification will be replaced by the corresponding operands. Finally, an instruction type is specified, that is useful for various reasons. For example, it allows analyzing the instruction breakdown of the generated stress-tests in terms of integer, float, SIMD, memory and branch instructions. It is worth noting that through the same instruction specification interface the experimenter can specify both individual instructions as well as whole instructions sequences that will be atomically included in the GA optimization search. One reason for using atomic sequences is for forcing cache-misses. For instance, to cause a cache-miss on an 8-way associative cache, a user can specify 3 atomic sequences each with 3 memory accesses that causes access to same set but to different blocks.

Regarding operand definitions, both register operands and immediate operands require their potential values to be specified. For register type operands the values are specified through the “values” parameter that accepts space separated register names. In Figure 10, the user has specified that the LDR result register can be any one of the x2, x3 or x4 registers. Regarding immediate operands, the potential values of an immediate are expressed through maximum, minimum and stride parameters. In the example the user allows the immediate value to take 33 different values, from 0 to 256 in strides of 8 i.e. 0,8,16,24...256. Essentially, in this example there are 99 possible ways the GA can use the LDR instruction (3 registers for memory result x 1 memory address register x 33 immediate values). The GA randomly generates any one of the 99 possible forms when generating the initial random

population and when performing the mutation operation. As the search is progressing, the GA will converge to the instruction variation that maximizes the target metric. If none of the evaluated instruction's possible variations helps to maximize the fitness value, then the instruction will likely stop appearing in the GA generated source codes. For instance, consider a long-latency instruction like integer division (DIV) used in an IPC maximization search. After, few generations the DIV instruction will most probably be eliminated from the individuals because it does not contribute in generating fit populations.

An operand definition, if desired, can be common for multiple instructions. For instance, the "mem_address_register" and "immediate_value" can be used by other memory instructions that the user may want to define, such as for the ARM ISA instructions LDP, STR, STP. The instruction and operand specification interface can serve one more purpose: as the means to force or explicitly avoid instruction dependencies. For instance, if optimizing for maximum instructions per cycle (IPC) it may be undesirable to have short-latency integer instructions depending on memory loads. Thereby, to avoid integer instructions depending on memory loads the user can specify two disjoint sets of integer register operands, one for memory destinations and one for source operands of all other integer operations.

The GA uses the instruction and operand definitions to generate individuals during the optimization search. These individuals are printed inside a template source code file (the location of the template file is provided in the main configuration file by the experimenter) that will be eventually compiled in a binary. The template source file must contain an empty loop body that is filled with the GA generated individuals. To indicate where the individual will be printed, the string "#loop_code" must be written within the empty loop body. Before compiling an individual, the framework removes the "#loop_code" string and prints the instruction sequence starting from the indicated line. Within the template file the user can also specify some fixed code that can be part of the loop body across all individuals e.g. add

NOP instructions for padding. The template source file may also include user specified initialization code that contains register and memory initialization. We find that register values have considerable effect on power consumption, so they must be initialized judiciously. For this work, we have use checkerboard patterns (e.g. 0xAAAAAAAA) since they increase bit switching that helps in maximizing power or dI/dt voltage-noise. We have observed that checkerboard pattern values may increase the fitness value by approximately 10% compared to setting all bits to high (i.e.. 0xFFFFFFFF).

It is worth mentioning that while this thesis performs GA searches at assembly programming level, the instruction definition interface and the template source file can be also used to perform optimization at a higher-level language (e.g. at a C code level).

3.1.3 Measurement and Fitness Evaluation

Each source code is compiled to a binary and measured on the target machine (the GA framework typically runs on a separate workstation). This procedure typically involves transferring the source file to the target machine, compiling the binary on the machine, running the binary, measuring the metric of interest through a measurement instrument (such as multimeter, oscilloscope or software accessible counters) while the binary is running, and, finally, stopping the binary execution and calculating the fitness value based on the measurements. An abstract Python class, refer to as the “Measurement.py”, provides the template for scripting such measurement procedures. Moreover, the class contains various utility functions that can be useful for scripting these procedures. For instance, the class contains functions for communicating through ssh with the target machine such as copying files over scp and executing any arbitrary command. To create a custom measurement script the user must inherit the Measurement.py class and overwrite the “init” and “measure” functions. The “init” should contain specific to the measurement procedure parameter initializations (e.g. number of active CPU cores, number of measurement samples to take etc.) and the “measure” function defines the actual measurement procedure. The

specific measurement parameters initialized in “init” function should be defined in a xml configuration file (not in the main configuration file). Both the measurement class name and its corresponding configuration file should be specified in the main configuration file. The framework utilizes the Python language capability to dynamically load a class. This means that the user defined class is dynamically loaded by only specifying the class name in the input configuration file. No other change in the source code is required.

Eventually, a fitness value will be given on the generated individual based on the measurement results. This is needed so that the GA can rank the individuals and pick the fittest ones that satisfy the most the optimization goal(s). An individual can have many measurements associated with it, e.g. maximum voltage droop and average power consumption. The framework offers a default fitness class “DefaultFitness.py” that simply uses the first measurement (in the list order) as the fitness function. More complicated fitness functions might be desired, for instance, maximize voltage droop while keeping average power low. The framework offers the user the ability to define such functions by writing a custom class that inherits from “DefaultFitness.py” and overrides the “getFitness” function. Similarly, to the measurement scripts, to use the custom fitness class the user must specify the fitness class name in the main configuration file.

3.1.4 Output

The framework’s output is the source code of all individuals. Each source code is saved in a different file. The name of the file includes: the population number, individual id and an array of measurements. For example, for the individual with id number 10 that belongs to population number 1 and with measured average and peak power of 1.3W and 1.33W respectively the file name would look like this 1_10_1.30_1.33.txt. By default, the first measurement is the fitness value, this naming convention facilitates the quick retrieval of the fittest individual using basic UNIX commands.

Moreover, each GA population is saved in a separate binary file. This binary file contains the source code, the id, the parent ids and the measurement values of each individual. These binary files can be loaded in a Python script for advanced result post-processing. As part of the framework release, there is a Python script that reads the populations in binary format and extracts statistics such as the fitness value of the fittest individual per generation and instruction mix breakdown of fittest individual per generation. Furthermore, the binary population files can be used as seed population for a new GA search (by default a new GA search starts with a randomly generated population). In such case, the user must specify the location of the seed population file in the main configuration file.

Additionally, in the output directory of each GA run the following are saved for record-keeping: the GA source code, the configuration files and the template individual source file used for the run.

3.2 GeST Evaluation Platforms

We use GeST to generate the following viruses: a) power-viruses for ARM Cortex-A15, ARM Cortex-A7, Intel i5-2400 and X-Gene2 CPUs, b) IPC virus for the X-Gene2 CPU and c) a power-virus that targets both power and instruction stream simplicity for the X-Gene2 CPU (henceforth referred to as PowerVirusSimple). For generating PowerVirusSimple we use the GeST capability to work with user defined custom fitness functions.

The characteristics of the evaluated CPUs are shown in Table III. We generate power viruses for ARM Cortex-A15 and Cortex-A7 running on a bare-metal environment (without OS). The chips are hosted on a CoreTile Versatile Express evaluation board. The board offers external measurement points that allow measuring CPU power, current and voltage. We hook an ARM energy-probe on the measurement points to read the power. Next, we generate a power virus and an IPC virus for the Ampere Computing X-Gene2 ARM-based server CPU. This server offers temperature sensor readings accessible through the i2c

Table III. GeST evaluation platforms.

CPU	# of Cores	Board	Environment	Stress-test developed	Measurement Instrument
ARM Cortex-A15	2	CoreTile Versatile Express	Bare Metal	power-virus	ARM energy probe
ARM Cortex-A7	3	CoreTile Versatile Express	Bare Metal	power-virus	ARM energy probe
Ampere X-Gen 2	8	Validation Board	Centos 7.2	power-virus and IPC virus	i2c temperature sensor readings, performance counters
Intel i5-2400	4	HP Business Desktop Computer	Ubuntu 14	power-virus	Likwid-power-meter

interface. We use the i2c interface to generate the power virus by optimizing towards maximum temperature. The IPC virus is generated by monitoring the IPC from the perf Linux utility. On the same system we demonstrate the GeST ability to optimize complex fitness functions (multi-objective) by generating a virus that targets both high temperature and instruction stream simplicity (fewer unique instructions). Lastly, we generate a power-virus on an Intel i5-2400 desktop CPU.

GA searches are performed on a single core. GeST can do multi-core optimizations by launching multiple workload instances but optimizing on single core has the advantage of less measurement variability which helps the GA optimization to converge faster. This is especially true when runs are conducted within an OS environment. Despite the GA search performed on a single core, a virus is tested by running it on all cores. All results reported in this thesis are measured with all cores active with each core running a separate virus instance. The viruses developed in this thesis do not make use of shared resources (e.g. LLC). Hence, the generated viruses scale well with multi-core execution because running multiple virus instances is not causing performance interference. The other workloads used

for comparison purposes are also executed on all cores. For single-thread benchmarks we execute multiple instances and for multi-thread benchmarks (e.g. NAS, Parsec) we execute one instance with multiple threads.

We are aware of a previous work [27] that evaluates a GA framework for power-virus generation on simulated multi-cores and reports significant increase in power-consumption when virus threads access shared memory. This increase in power consumption is attributed to the high engagement of network-on-chip, which in the simulated systems has a large contribution in total power consumption (for some runs more than 33% of the total power). In all CPUs we tested, we have successfully generated effective power/thermal stress-tests that exceed the fitness of the worst-case workload or manually-written stress-test by at least 10% without using shared memory. With that said, memory instructions that access shared memory can be added to the GeST optimization. The user must provide a template file that initializes shared-memory and launches multiple workload threads (in case the shared memory is defined in kernel then multiple process instances instead of threads should also work). Moreover, the user must define in the main configuration file the instructions that access the shared-memory. This important extension is beyond the scope of this thesis.

Regarding framework execution time, the GeST runtime is defined by the following factors: a) time to measure each individual, b) for how many generations the optimization is performed, and c) how many individuals are measured per generation (population size). In our experience GeST produces stress-tests that exceed significantly conventional workloads after 70-100 generations. Given 50 individuals per population and 5 seconds per measurement (which is typical for power optimization) the runtime is approximately 7 hours.

In the framework release we include measurement scripts and fitness functions that can be used for power, IPC, dI/dt noise and instruction-stream simplicity optimization for x86 and ARM ISA.

3.3 Power-Virus Generation

We develop a power-virus for Cortex-A15 and Cortex-A7 in a bare-metal environment. The measurement function for this optimization executes each GA generated binary for few seconds and takes multiple power readings during the binary execution. The fitness function calculates the average value of all power samples. Fittest individuals are considered the ones with the highest average power.

The relative (normalized to coremark benchmark) power-results for Cortex-A15 and Cortex-A7 are shown in Figure 11 and Figure 12 respectively. First, it is worth noting that the GA generated stress-test both on Cortex-A15 and Cortex-A7 cause the highest power

Table IV. Instruction breakdown of Cortex-A15 and Cortex-A7 power viruses.

GA virus	Short Latency Int	Long Latency Int	Float/SIMD	Mem	Branch	Total Loop Instructions
Cortex-A15	4	5	22	18	1	50
Cortex-A7	8	6	16	10	10	50

consumption and surpass the manually written stress-tests (A15manual_stress_test, A7manual_stress_test) as well as conventional bare-metal workloads (coremark, imdct, fdct). This emphasizes the GA's ability to generate worst-case pathological scenarios that are hard for humans to produce. The other interesting observation is that Cortex-A7 GA virus is not a good stress-test for Cortex-A15 and Cortex-A15 virus is not a good stress-test for Cortex-A7. Different CPU designs require different stress-tests to maximize their CPU power consumption. The need for different stress-tests for dissimilar micro-architecture is also evident by the differences in the instruction mix between the Cortex-A15 and Cortex-A7 GA-power-viruses depicted in Table IV. The breakdown is shown in terms of short latency (1 cycle) integer instructions (e.g. ADD, SUB), multi-cycle instructions (e.g. MUL), float or SIMD instructions, memory instruction and branch instructions. Both stress-tests consist of a loop of 50 instructions. The table shows that to raise the Cortex-A7 power

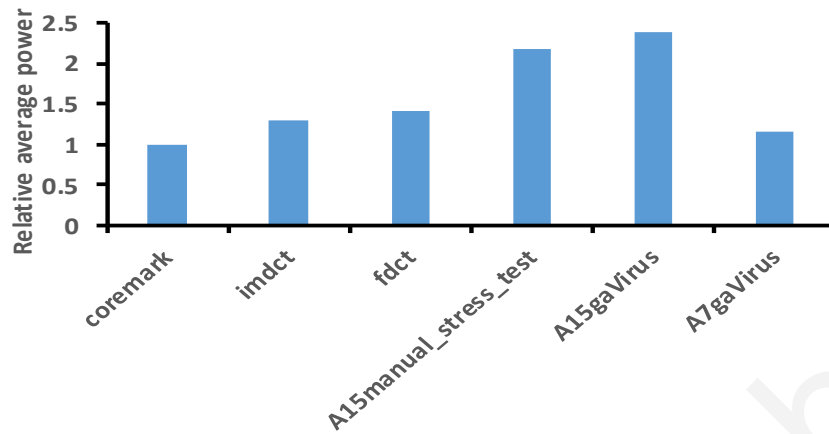


Figure 11. Cortex-A15 power results.

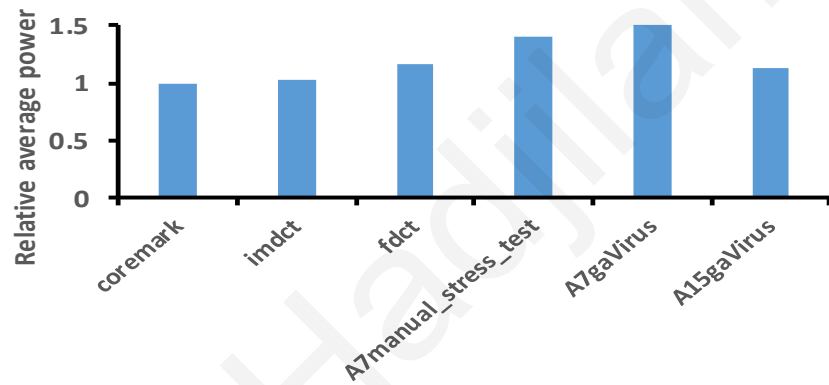


Figure 12. Cortex-A7 power results.

consumption it is important to add a lot of branch instructions (10 instructions out of 50 are branches) while for Cortex-A15 only one branch is used. Also, Cortex-A7 virus prefers slightly shorter latency integer instructions as compared to Cortex-A15 virus. A common observation for both viruses is that floating point/SIMD instructions are dominant.

Next, we test GeST on the X-Gen2 ARM-based server CPU. We generate a virus that maximizes chip temperature (and hence the power) using chip temperature sensor feedback. We compare the temperature of the virus (denoted as powerVirus) with various benchmarks (Parsec and NAS suite) and a virus that maximizes IPC (denoted as IPCvirus) generated with the GA using perf Linux utility. Figure 13 shows the relative (normalized to bodytrack

benchmark) chip temperature. The power virus outperforms all other workloads by reaching the highest chip temperature.

The IPC virus also raises the chip temperature very high (but lower than power virus). IPC virus is expected to cause high temperature because it causes very high CPU activity. It is interesting to understand what characteristics make the power virus cause higher temperatures. Table V provides a comparison of the IPC and the power viruses. The IPC virus achieves 12% higher IPC but also 12% lower power consumption than the power virus. As expected, the IPC virus does not contain any long latency integer instruction. Also, the IPC virus makes moderate use of memory operations. On the other hand, the power virus contains a few long-latency instructions and uses a lot of memory operations. Perhaps the modest use of long-latency instruction helps to increase the power consumption and temperature (which is the goal of the virus) by keeping active the issue queue and the dependency tracking logic. Also, the more frequent engagement of the memory subsystem

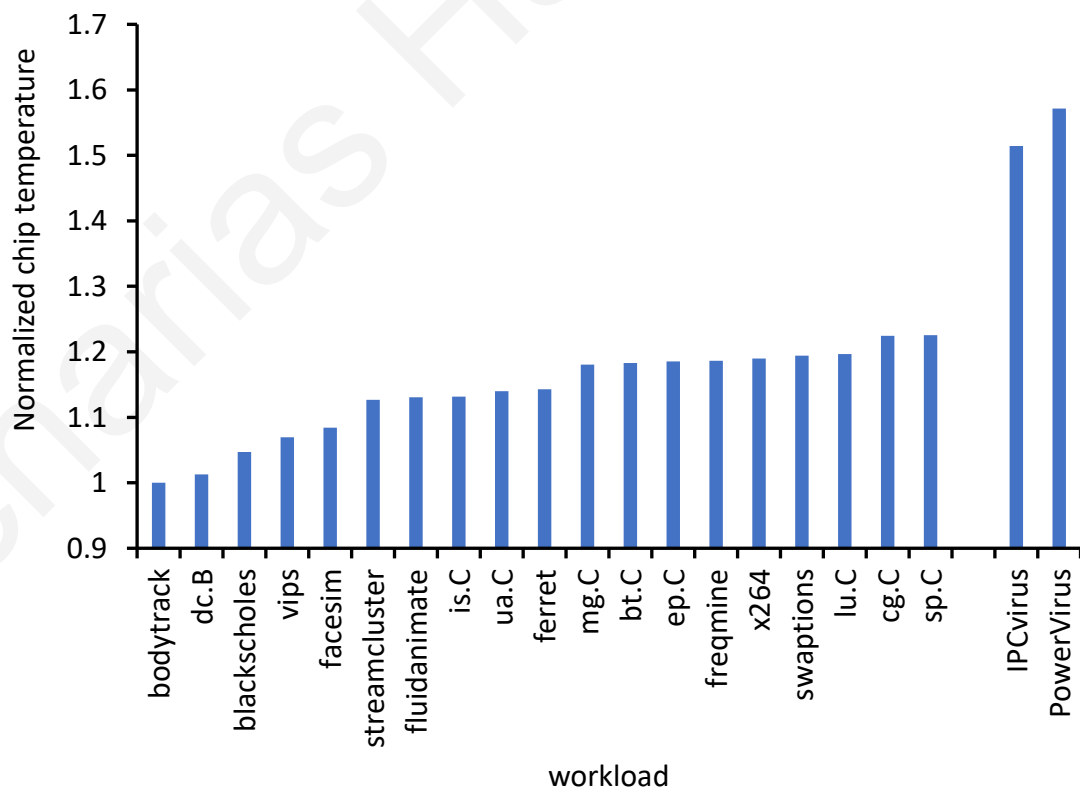


Figure 13. X-Gene2 temperature results.

Table V. Power virus, simple power virus and IPC virus comparison.

GA virus	Short Int	Long Int	Float/SIMD	Mem	Branch	Relative IPC	Relative Plug Power (W)	Relative Chip Temp.	# of Unique Instructions
Power Virus	22	5	9	12	2	1	1	1	21
Power Virus Simple	16	7	13	11	3	0.94	0.99	1	13
IPC virus	26	0	15	6	3	1.12	0.88	0.94	13

results in a higher power consumption. While the use of long-latency operations and many memory instructions increases the temperature, it also reduces the IPC. This highlights an interesting tradeoff that the GA is capable to make to maximize the temperature. This analysis clearly shows that the highest IPC does not automatically convert to highest power consumption and temperature. A recipe for the highest power consumption and temperature (at least for the X-Gene2) seems to be a combination of high IPC (not the highest) with heavy use of memory instructions and modest use of long-latency operations.

Furthermore, we demonstrate the GeST capability to optimize a complex fitness function by generating a power-virus that achieves both high temperature and simplicity in terms of using less unique instructions (unique opcodes). Simplicity of the generated power-viruses is desired for various reasons such as for ease in isolating inefficiencies in initial chip samples, like hotspots, and instructions that are power-intensive. To optimize for both high-temperature and simplicity, we use the GeST interface for scripting custom fitness functions (presented in Section 3.1.3).

We use the equation shown in Figure 14 for calculating the individual's fitness. The fitness can take values from 0 to 1 and the equation has two parts, with both parts contributing equally to the fitness value. The first part rewards high temperature. The

$$F = (M_T - I_T) / (MAX_T - I_T) * 0.5 + (T_I - U_I) / T_I * 0.5$$

Fitness (F), M_T (measured temperature), I_T (idle temperature), MAX_T (max temperature), T_I (total instructions), U_I (unique instructions)

Figure 14. Complex fitness function rewarding high temperature and instruction simplicity.

contribution of the temperature part must be bounded to a 0-1 value range (temperature score), hence, we normalize the measured temperature with the maximum possible temperature. The maximum temperature can be obtained either from a previous GA run or from specifications e.g. TJMAX. An issue with the temperature score is that even during idle operation the temperature is not negligible because of ambient temperature. Thereby, we must subtract the idle temperature to avoid overestimation of the temperature score. The second part of the equation is the simplicity which rewards having less unique instructions. It is also bounded between a 0-1 value range. Assuming individuals of 50 instructions, an individual with 25 unique instructions would be assigned a simplicity score of 0.5 whereas an individual with 15 unique instructions would be assigned a simplicity score of 0.7 (without taking in account the 0.5 weight factor).

We run the GA with the complex fitness function for the same number of populations as the GA that generated the power virus. The characteristics of the fittest individual (powerVirusSimple) are shown in Table V. This virus has very similar characteristics with the original power virus. Specifically, we observe the same characteristics we discussed in the previous paragraphs such as fairly high IPC, significant use of memory and modest use of long latency integer instructions. However, there is also a difference, the new power virus prefers to spend more instruction slots for floating point and long latency instructions at the expense of the short latency instructions. This has an impact on the IPC which is 6% lower compared to the original power virus but this doesn't affect its temperature and power consumption. The simple power virus achieves virtually the same power and the same temperature as the original power virus. The complex fitness optimization is considered

successful as the simple power-virus achieves the same stress-level as the original power-virus while using only 13 unique instructions instead of 21.

Last, we apply GeST to generate a power-virus on Intel i5-2400. Figure 15 shows the GA progression. The GA search converges to a power-virus that consumes 90W CPU package power. We compare the power-virus with the well-known prime95 stress-tests in terms of power consumption. We run both the small (data fits inside cache) and the large (data does not fit inside cache) prime95 tests. Figure 16 shows the power results. The GAvirus consumes 6% more power than the prime95 large test and 8% more power than the prime95 small test.

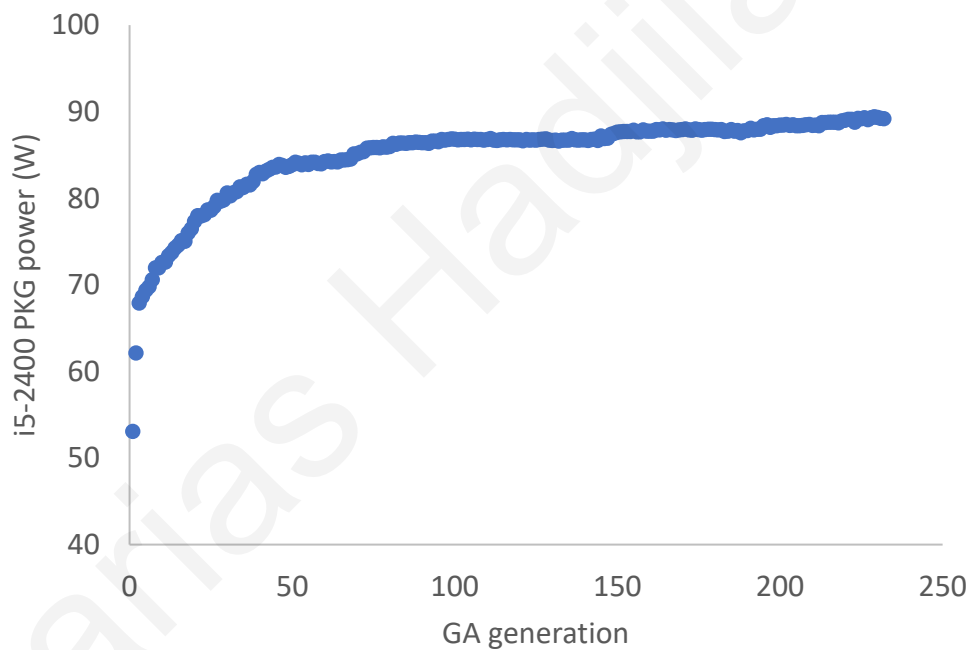


Figure 15. GeST CPU power virus search on i5-2400.

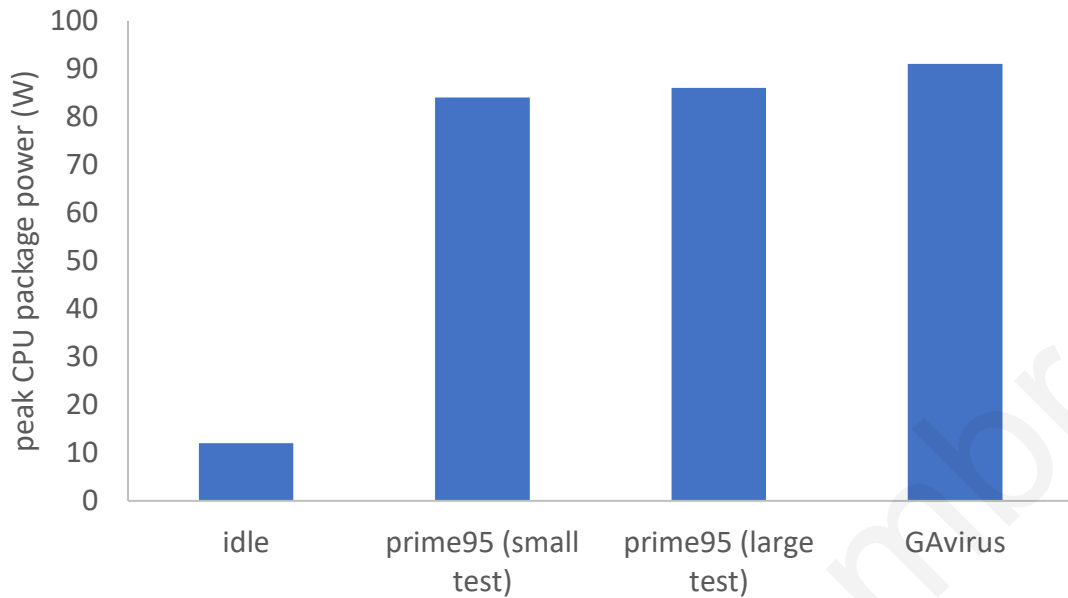


Figure 16. Power consumption results for Intel i5-2400.

3.4 Voltage-Noise Virus Generation

This section demonstrates the capability of GeST to generate voltage noise viruses and consequently stability-tests. For this study we use an AMD Athlon II X4 645 CPU hosted on an Asus M5A78L LE motherboard. This motherboard offers high bandwidth voltage sense points that can be used to monitor voltage noise. This is achieved by connecting an oscilloscope to the sense points through an active differential probe. The GA generates the dI/dt virus by optimizing towards maximum peak-to-peak voltage. The framework runs each GA generated binary for a few seconds. During the binary execution the minimum and maximum voltage observed on the oscilloscope are recorded. The binaries that achieve the highest difference between maximum and minimum recorded voltages are considered the fittest.

Figure 17 shows the max-min voltage noise caused by various workloads compared to the GA generated virus. The GA dI/dt virus clearly outperforms the other workloads including well known stability-tests such as Prime95 and AMD’s own stability test. Since the dI/dt virus causes the highest voltage-noise it should stress the system’s stability better than the other workloads. A good stability-test must have high V_{MIN} . To characterize the V_{MIN} of a workload we run the workload multiple times and each time we lower the operating voltage in steps of 12.5mV. We keep the CPU frequency stable at the nominal value of 3.1GHz. The highest voltage at which a workload executes correctly (without corruption, error messages, crashes) is the workload’s V_{MIN} . Figure 18 shows the V_{MIN} of the various workloads we tested on the AMD CPU. The dI/dt virus is the best stability-test because it causes instability at a higher voltage, even higher than the commonly used AMD stability test and Prime95.

Our results show that workloads designed to draw very high power are not suitable stability-tests as they are not designed to drop the voltage very low and induce timing errors. Prime95 is a workload known to raise the CPU power consumption very high. Such workloads are a good choice for exercising IR drop as well as characterizing thermal

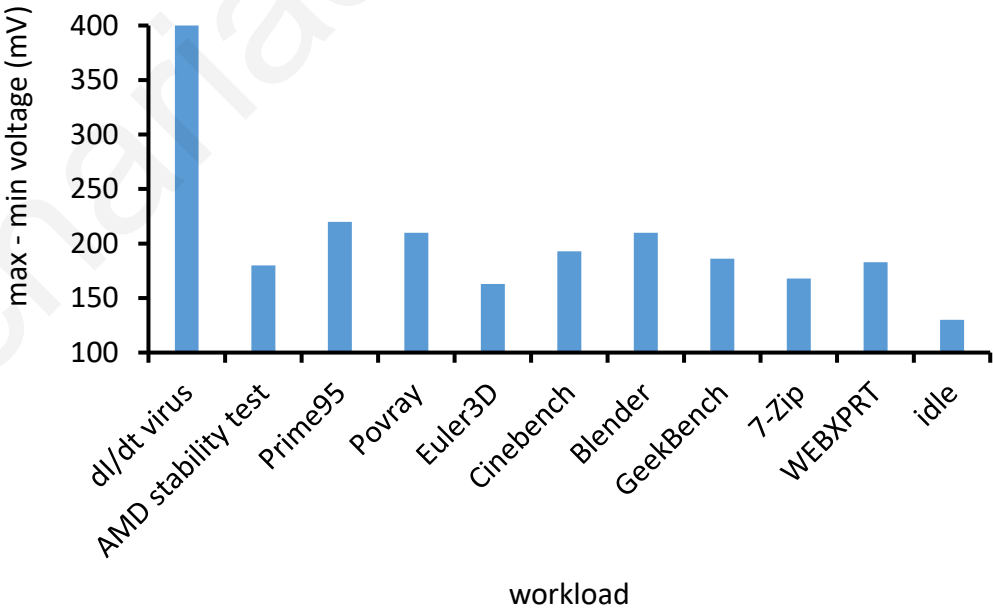


Figure 17. Voltage-noise results on AMD Athlon CPU.

stability and making sure that the temperature will not exceed a critical threshold during normal operation. But they are inadequate for characterizing the susceptibility to timing errors. Also, this is another confirmation that AC di/dt noise dominates over IR drop.

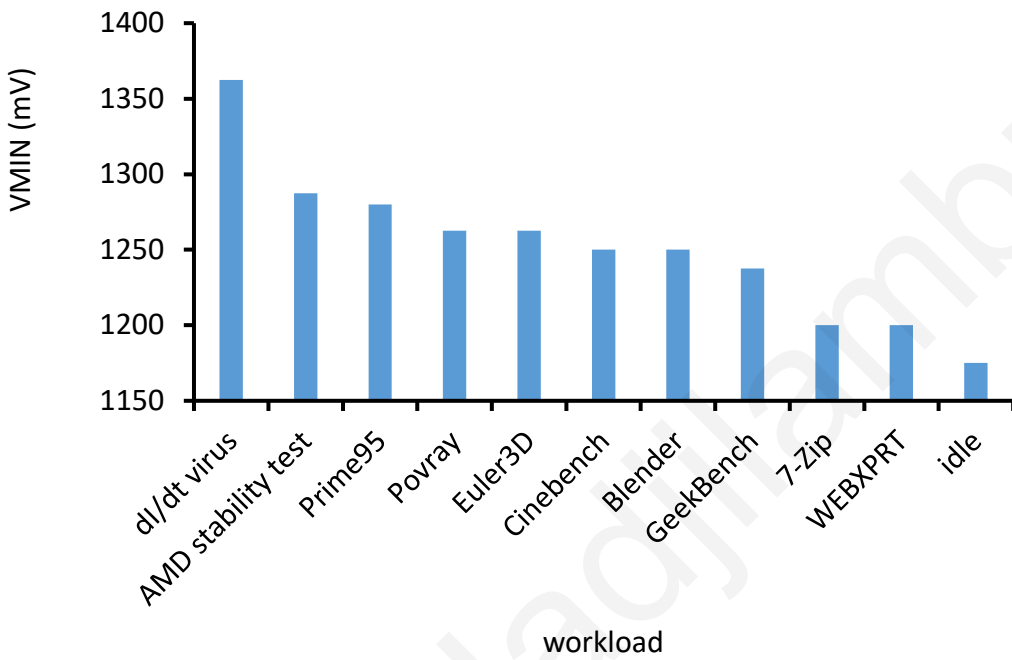


Figure 18. V_{MIN} results on AMD Athlon CPU.

Chapter 4. EM methodology

This chapter presents the EM methodology for PDN characterization. This chapter provides the following: a) the experimental apparatus needed to perform the EM methodology, b) the experimental proof of the correlation between voltage-noise and EM amplitude, c) two EM methodologies for determining the PDN 1st order resonance-frequency, and d) the EM approach for generating dI/dt stress-tests via a GA optimization that maximizes the EM amplitude.

4.1 Required Experimental Apparatus

The EM methodology requires the following components: a) a CPU that will be monitored, b) an antenna to sense the EM emanations that are emitted from the monitored CPU, c) a spectrum analyzer for reading the amplitude and the frequency of the received signals d) a coaxial cable that connects the antenna with the spectrum analyzer, and e) a workstation that is connected with the spectrum analyzer for orchestrating the resonance frequency detection and dI/dt virus generation.

Examples of EM methodology experimental setups are shown in Figure 19. The figure shows both the ARM Juno and AMD desktop PC experimental setups. The spectrum analysers Agilent E4402B (Juno setup) and Agilent N9332C (AMD setup) are used to measure the EM signals. Note that cheaper commercial software-defined radio receivers



Figure 19. Experimental setup for the ARM Juno board (left) and AMD desktop CPU (right).

should also work [68]. Also, instead of a spectrum-analyzer, an oscilloscope can be used for time-domain measurements given that the EM signal amplitude is high enough (typically oscilloscopes have much lower noise floor than spectrum-analyzers).

In the figure we observe that the antennae are placed at a stable position 5-10cm close to the monitored CPUs. We prefer the back-side of the PCB (i.e. back of the CPU socket) due to proximity to the die which translates to stronger received signals. As illustrated in the figure, such approach is particularly convenient for desktop CPUs as the only requirement is to simply remove the tower's back-cover.

It is also possible to capture the EM emanations from the front of the PCB. The received signal can be amplified with a pre-amplifier or with an antenna matching-network that amplifies the signal at the targeted PDN resonance-frequency. Particularly, we have experimented with a matching network that matches the X-Gene2 CPU's PDN resonance-frequency at 150MHz. We confirm that with this matching network we were able to capture strong EM signals from the front of the PCB.

Regarding the antenna selection, is recommended to use an antenna that has a flat response in the frequency range where the PDN resonance-frequency is expected to lie. This is desired to avoid any biases when measuring EM signal amplitude at different frequencies. Such measurement biases are unwanted during resonance frequency-determination and GA dI/dt virus generation.

We crafted a square loop antenna (3 cm side length) as a receiver for the emanated EM radiation (this antenna is used in Figure 19 experimental setups). As shown in the antenna's frequency-response graph in Figure 20, the antenna has a relatively flat and low frequency response from DC until 1.2 GHz, with a self-resonance frequency at 2.95 GHz. This means that the antenna has a flat response at the range where we expect the 1st order resonance-frequency to lie (1-200MHz), hence, this antenna is suitable for the EM methodology. Note that such antenna is very cheap to produce. The fact that applying the EM methodology does

not require spending a significant budget for buying sophisticated commercial antennas (e.g. [80]), can be considered as another advantage of our approach over the state-of-the-art high-bandwidth voltage-measurement approaches.

It is worth saying that a well-matched antenna at 1-200MHz should offer better reception and could enable capturing EM signals from longer distance and more convenient positions. Unfortunately, it is extremely difficult to craft or buy an antenna that is both well-matched and provides flat response in the wide range of 1-200MHz. Therefore, to avoid measurement biases a normalization step that normalizes the measured amplitude at different frequencies based on the antenna's frequency response profile can be used.

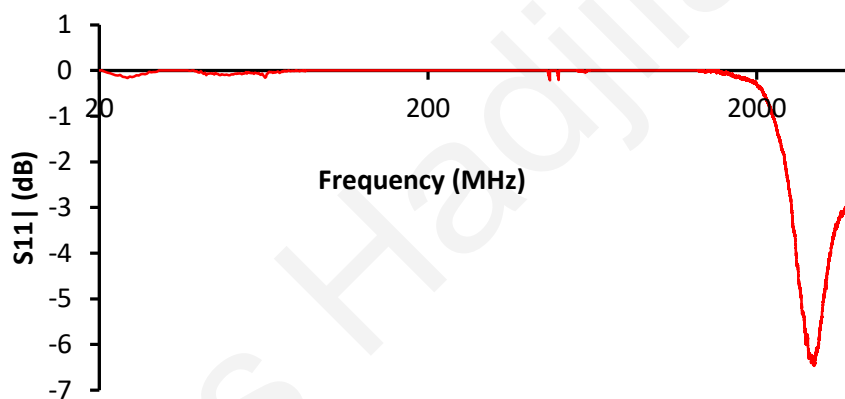


Figure 20. Measured $|S_{11}|$ for the square loop antenna indicating a self-resonance around 2.95 GHz.

4.2 Relationship Between CPU EM Emanations and On-Chip Voltage Noise

It is well-known that metallic conductors act as transmitting antennae that emanate EM radiation under oscillating voltage and current stimulation [38][69]. On-chip interconnections and transistors act as distributed radiating antennae due to time-varying current consumption induced through normal program execution. Simple periodic activity, such as that due to instruction loops, cause periodic variations in CPU power (i.e. sequence

of DIVs followed by ADDs) that manifest as visible spikes in the EM spectrum, at a frequency F equal to $1/T$ (where T is the loop period) [18].

Fundamental antenna theory (say, for a traditional Hertzian dipole) states that the component of the radiated power for the transmitting antenna, at a specific frequency, varies quadratically with the amplitude of the oscillating feed current [38] at the corresponding frequency and the so-called radiation resistance. The radiating resistance of a conductor can be differentiated from its loss resistance, in that the former is a function of the geometry of the conductor and determines the magnitude and the directivity of the radiated power [38]. The loss resistance, in contrast, manifests as ohmic losses dissipated through the conductor. Periodic current load (ILOAD), pulsing at the first-order resonance frequency, can trigger sustained oscillations of large magnitude in VDIE and IDIE.

We simulate the simplified PDN model in Figure 2(a) with a persistently pulsing current excitation (ILOAD) at 80MHz which matches the 1st-order resonance frequency (Figure 2 (b)). This sets off resonant oscillations in the PDN as illustrated by HSPICE [35] simulations in Figure 21 (a). At resonance, both voltage and current oscillations maximize in amplitude. This, in turn, maximizes the radiated EM power from the on-chip distributed antennae, due to the quadratic dependence with oscillatory current amplitude. Therefore, measuring the frequency at which the amplitude of the emanated EM power is maximized directly reveals the 1st-order resonance frequency. We leverage this relationship between radiated EM power and on-chip voltage-noise to maximize the voltage-noise by maximizing the amplitude of EM signals.

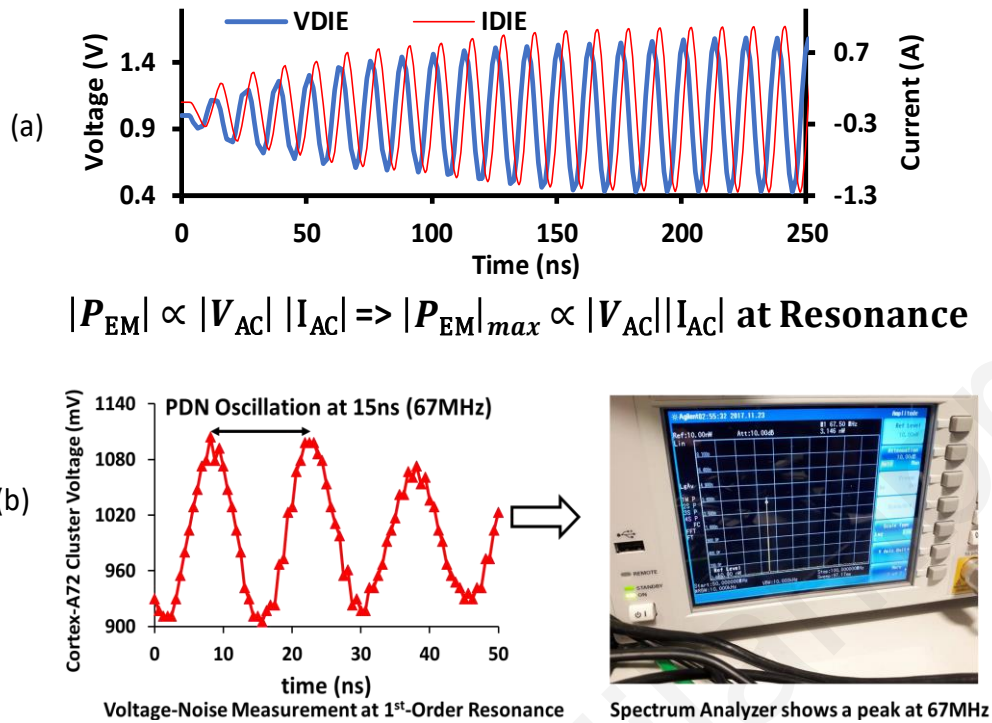


Figure 21. a) Simulated waveforms showing the die voltage (VDIE) and die current (IDIE) in the simplified PDN model in Fig. 2. A pulsing ILOAD triggers the first-order resonance where the AC-component of both VDIE (V_{AC}) and IDIE (I_{AC}) maximize, thus maximizing the radiated EM power. b) Resonant oscillations (close to the resonance-frequency at 67MHz) triggered on an ARM Cortex-A72 cluster (on ARM Juno Platform) causes a corresponding peak in the measured EM power captured on a Spectrum Analyzer.

We experimentally observe the correlation between high-voltage noise and high EM amplitude. We use the experimental setup from Figure 19 to capture the EM signals emitted from the Cortex-A72 CPU while running a dI/dt virus. Simultaneously we capture the time domain supply-voltage oscillations caused by the dI/dt workload with an On-Chip Digital Storage Oscilloscope (OC-DSO) [72] that is integrated in Cortex-A72. We compare the two measurement instruments in Figure 21 (b). The figure shows that resonant voltage oscillations at 15ns on Cortex-A72 captured with the OC-DSO are depicted as a high spectrum spike on the spectrum analyzer at 67MHz. Moreover, we obtain the frequency-domain representation (using the Fast Fourier Transform (FFT) algorithm) of the voltage waveform shown in Figure 21 (b) and we compare it with the spectrum analyzer readings in the range of 10-90MHz in Figure 22. The dominant frequency of both frequency-domain representations is exactly aligned at 67MHz. Moreover, the two instruments agree on other

less dominant spikes as well, such as the virus's base loop frequency (1/loop period) located at 16.66MHz.

We further establish the theory that links CPU EM emanations with on-chip voltage noise in Chapter 6 and Chapter 7 where we show that the proposed EM methodology can determine the 1st order resonance-frequency of a CPU and generate dI/dt viruses that can be used for V_{MIN} determination and for guiding DVS decisions.

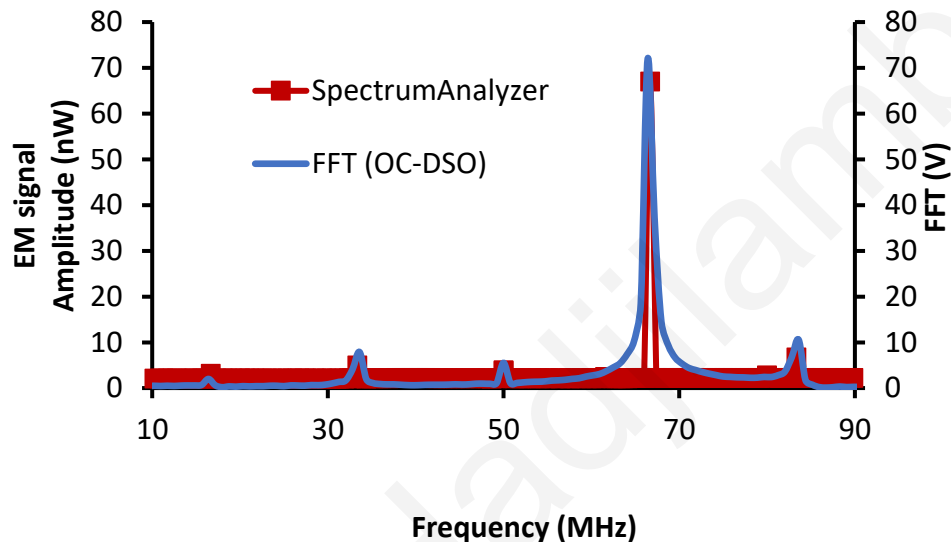


Figure 22. Comparison of spectrum analyzer readings (left axis) with FFT of OC-DSO voltage readings (right axis) during execution of a dI/dt virus. The two measurements agree as they reveal spikes at the same frequencies.

4.3 EM Resonance Frequency Detection (Loop Method)

To quickly identify the resonance-frequency from EM emanations we propose the following procedure. The first step is to manually design a simple instruction loop composed of separate high and low current consuming sequences. This is not meant to be a dI/dt stress test but a loop that causes merely enough current variation to result in a visible EM spike at a frequency equal to the loop frequency (which is equal to the inverse of the loop iteration period). Then, while the loop is running on the CPU, we sweep the CPU frequency to modulate the loop period, and, consequently, the EM spike frequency. The spike amplitude is maximized when the loop frequency matches the resonance frequency because the

fluctuating loop-current will trigger resonant oscillation in the PDN [55]. Therefore, after the frequency sweep is over, the frequency at which the highest EM amplitude occurs reveals the resonance frequency. We refer to this approach as the “loop method”.

We demonstrate the loop method on the Cortex-A72 processor. We use a loop with the high current consuming sequence consisting of eight ADD instructions that are executed in 4 CPU cycles and a low current consuming sequence consisting of a single DIV instruction that takes 4 CPU cycles to execute. The period of execution of the overall loop (with both the high-current and the low-current consuming portions) is 6.6ns at the 1.2GHz CPU frequency (the nominal Cortex-A72 frequency). This corresponds to a loop frequency of 150MHz. To modulate the loop frequency, we sweep the CPU frequency from 1.2GHz down to 300MHz and we record the EM signal amplitude at each loop frequency.

Figure 23 shows the results of the frequency sweep. The amplitude is maximized at around 67-72.5MHz loop frequency when both cores are active (C0C1 curve). When only one core is powered-up (C0 curve) the resonance-frequency increases to approximately 85MHz. This increase is expected due to inverse proportional relationship between resonance-frequency and capacitance [55].

To confirm that the resonance-frequencies for C0 and C0C1 scenarios are correctly identified we use an independent methodology that utilizes a synthetic current load (SCL) circuit integrated on the Cortex-A72 [73]. The SCL allows loading the Cortex-A72 PDN

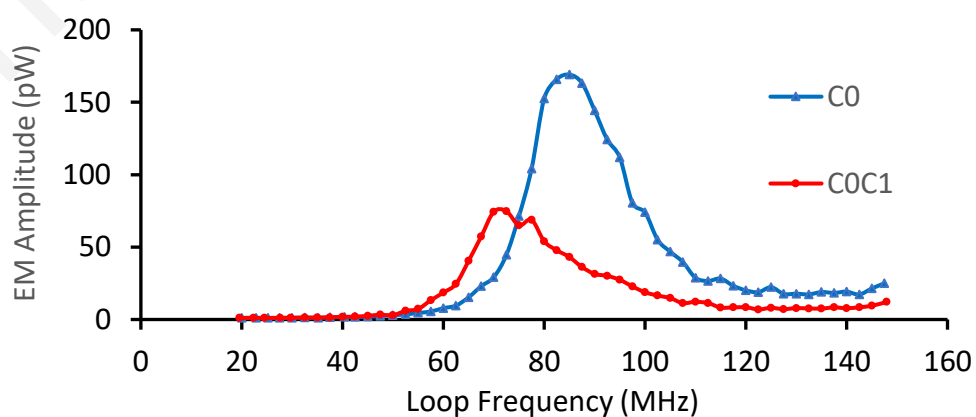


Figure 23. EM resonance frequency exploration for Cortex-A72 PDN with loop method.

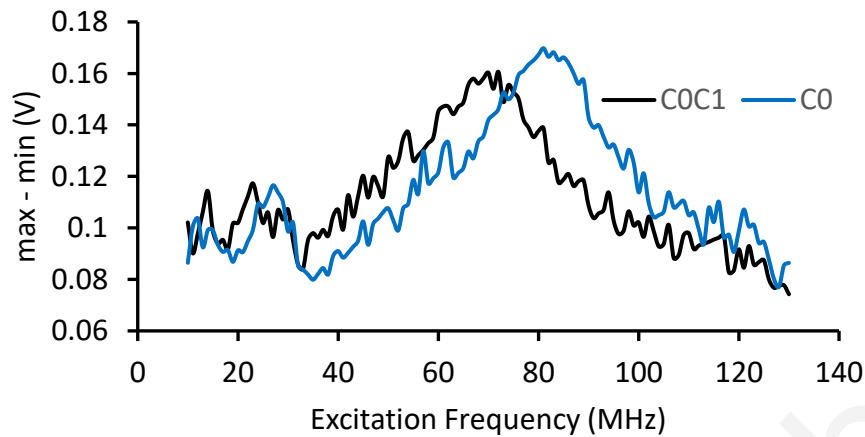


Figure 24. SCL stimulus reveals a resonance frequency in the range of 66-72MHz with two powered cores (C0C1) and 80-86MHz with one powered core (C0).

with artificial current at various frequencies. With the OC-DSO we can measure the PDN response to a SCL current load. We sweep the injected current frequency with SCL from 10MHz to 130MHz and we record the peak-to-peak voltage oscillations at each frequency with the OC-DSO. The highest voltage oscillation reveals the resonance frequency [42][73]. The results of the sweep are shown in Figure 24 according to which the first-order resonance frequency lies in the range between 66-72MHz (we observe a relatively flat frequency response around resonance) when both cores in the cluster are powered up (indicated by the label “C0C1” in the plot). These results agree with EM loop method for resonance-frequency detection. This confirms the effectiveness of the EM approach in identifying the resonance frequency. This approach is applicable to virtually any CPU that allows changing the clock frequency.

4.4 EM Resonance Frequency Detection (Clock Method)

Another approach for PDN resonance-frequency determination is to match the CPU frequency with the PDN resonance-frequency. We refer to this approach as the clock method. This approach does not require executing a specific loop of instructions, the only requirement is to prevent the cores from entering a clock-gated state. This can be easily

achieved by either disabling all C-states or by executing a trivial self-loop on the cores. We demonstrate this method on Cortex-A72. We run a self-loop e.g. “while(true);” while we sweep the CPU frequency from 60MHz to 80MHz. We record the EM amplitude at each frequency. The results are shown in Figure 25 that reveals a peak at 72.5MHz which matches closely the resonance-frequency identified in Figure 23 and Figure 24.

The clock method can be used only with CPUs that support very fine-grain frequency scaling. It can be particularly useful for black box approaches where the experimenter is either not allowed to run anything on the device or executing code on the device is not trivial (e.g. FPGAs, accelerators, GPUs). One use case where this method proved to be handy is for characterizing the Mali-T622 GPU that is hosted on the ARM Juno R2 board. Crafting a loop of instructions that causes an EM spike on Mali-T622 is not a trivial task due to lack of full control regarding to what is executed in the device. We can only program Mali from high-level Open-CL code that is compiled to native instructions through the GPU driver. This introduces a lot of non-determinism and impeded our efforts in applying the loop method on Mali-T622. With the clock method we were able to circumvent this short-coming

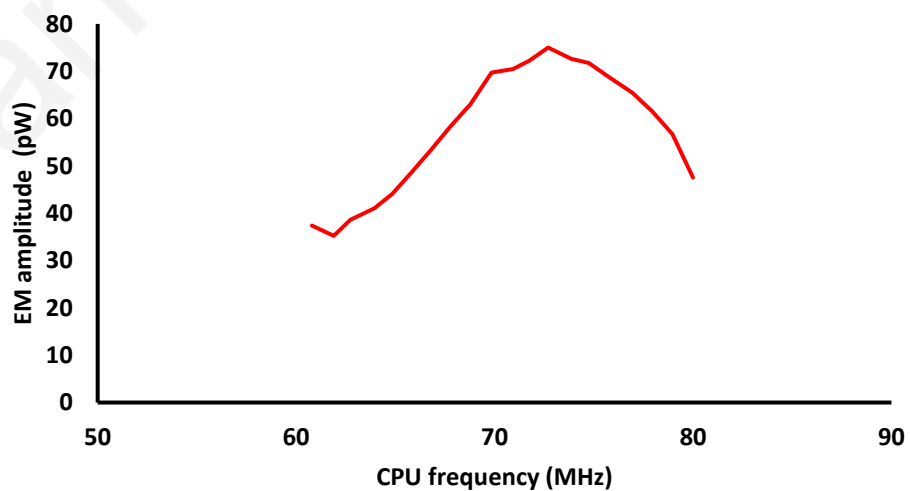


Figure 25. Resonance-Frequency exploration on Cortex-A72 with clock method.

to measure the resonance frequency of the Mali-T622. The results of the clock method on Mali-T622 are shown in Figure 26 and they reveal a resonance frequency at 70MHz.

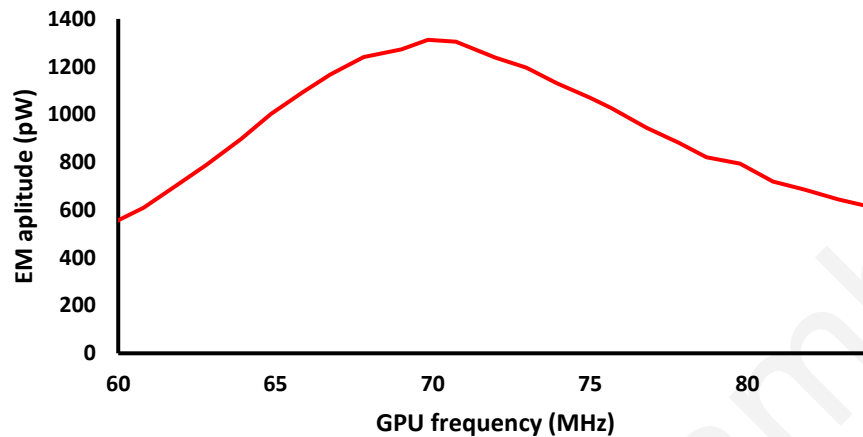


Figure 26. Resonance frequency exploration on Mali-T622 with the clock method.

4.5 EM dI/dt virus Generation

Previous work has proposed automated frameworks based on GA frameworks for generating dI/dt viruses [41][42][73] and power-viruses [27][40][57]. The main difference of this thesis with prior work is the usage of the EM amplitude as an optimization metric to drive the GA dI/dt virus search. We use GeST to drive the GA EM amplitude optimization. This section describes how we use GeST to generate dI/dt viruses by optimizing towards maximum EM amplitude.

For the EM optimization, the metric of interest is maximum EM amplitude at any frequency in the spectrum of 50-200MHz (the spectrum where the 1st order PDN resonance frequency is typically located). Particularly, the fitness function for the EM optimization is the mean root square of 30 EM samples.

Regarding the loop size of the individuals, we use a rule of thumb (heuristic) for picking the optimal loop size that takes in account the ratio of CPU frequency to the PDN resonance frequency and is similar to what authors used in [41]. Particularly, we use the following heuristic equation to estimate the optimal loop size:

$$\text{LOOP_SIZE} = (\text{MAX_THEORETICAL_IPC} / N) * \text{RESONANCE_CYCLES},$$

where $\text{RESONANCE_CYCLES} = \text{CLOCK_FREQ} / \text{RESONANCE_FREQ}$ and N takes values from 2 to 3.

The heuristic is based on the assumption that the following traits are essential to a successful dI/dt virus: a) an IPC that is a fraction of the maximum theoretical IPC of a CPU core because a mix of both short-latency and long-latency stalling instructions [39] (e.g. integer division) are needed to create sudden current surges, b) cause a current surge at each loop iteration, and c) the virus's loop iteration length in cycles to be equal to the CPU cycles that can fit inside a resonance period ($1/\text{resonance-frequency}$). The purpose of the first characteristic is to facilitate the GA convergence towards solutions that cause current surges. This is realized by targeting an IPC that allows both high and low power-phases. The second and third characteristics aim is to increase the likelihood that the current surges will lead to resonant voltage-noise build-up.

To use this heuristic the $\text{MAX_THEORETICAL_IPC}$ and the resonance-frequency must be known a priori. The $\text{MAX_THEORETICAL_IPC}$ can be found either from specs or from a GA optimization search targeting maximum IPC. The PDN resonance can be obtained either by running the GA for few iterations and recording the dominant EM frequency (the frequency with the highest EM amplitude), or through the quick methodologies for resonance frequency detection described in Sections 4.3 and 4.4.

We observe that satisfactory results are obtained after running the GA for at least 60 generations. The algorithm execution is typically limited by the measurement latency per individual. Approximately 18 seconds are needed to take 30 EM measurements which translates to an execution time of ~15 hours for 60 generations (when 50 individuals are considered per generation).

The assembly instructions that are used in the EM GA optimization deliberately target diversity in latency (both single and multi-cycle) and instruction-type (integer, floating-

point, SIMD and load/store) to facilitate rapid convergence (as we present later in Section 8.1. indeed, the dI/dt viruses benefit from instruction type diversity). In particular, for the ARM ISA [8] we use: a) short latency integer instructions such as move (MOV), add (ADD) b) multi-cycle long latency integer instructions such as MUL and DIV, c) floating point equivalents of the above arithmetic instructions, d) equivalent SIMD instructions using SIMD registers, e) unconditional dummy branches pointing to the next instruction (conditional branches are difficult to incorporate as they can introduce non-determinism), and f) load and store memory instructions. For the x86 instruction set, the same instruction mix selection principles as with ARM are used with some minor modifications. Since x86 does not have explicit load-store instructions, memory operations are implemented by using memory address operands for integer instructions. For SIMD operations, SSE2 [54] instructions are used. As shown in the conclusions chapter the viruses make use of nearly all instruction types to maximize voltage noise. This clearly illustrates that it is essential to have diverse set of instruction types to select from during GA optimization. Also, we would like to emphasize that in contrast to a power-virus optimization (such as the ones discussed in Chapter 3), for the EM optimization we deliberately include long-latency low-power instructions (e.g. FSQRT and integer DIV) because they are necessary for creating pipeline stalls that lead to dI/dt events.

Furthermore, we deliberately avoid cache misses due to the timing non-determinism introduced by them. The GA should give preference to instruction sequences with periodic current swings triggering first-order resonant oscillations in the PDN. Thereby, events such as cache misses that introduce time variability should be avoided as they result in significant jitter to the GA algorithm, which in turn impedes its convergence. Nonetheless, memory references, even if they are always hits, are found to be essential for maximizing voltage noise due to engaging the memory subsystem (pipeline resources and L1 cache).

Chapter 5. Measurement Setup

Table VI. Experimental platforms.

MB	CPU	# of Cores	ISA	uArch	Highest Freq., Voltage Point	Technology (nm)	OS	Voltage noise visibility
Juno Board R2	Cortex-A72	2	ARM	Out of Order	1.2GHz, 1V	16	Debian	OC-DSO [72][73]
Juno Board R2	Cortex-A53	4	ARM	In-Order	0.95GHz, 1V	16	Debian	None
Asus M5A78L LE	Athlon II X4 645	4	x86-64	Out of Order	3.1GHz, 1.4V	45	Windows 8.1	On-package pads
Validation Board	Ampere X-Gen 2	8	ARM	Out of Order	2.4GHz, 0.98V	28	Centos 7.2	None
Validation Board	Ampere X-Gen 3	32	ARM	Out of Order	3.0GHz, 0.87V	16	Centos 7.2	On-Chip droop detector

Table VI shows an overview of the platforms where the EM methodology is evaluated. The ARM Juno [7] platform hosts a heterogeneous multiprocessing System-on-Chip (the so-called big.LITTLE configuration) consisting of separate clusters of the dual core Cortex-A72 and a quad core Cortex-A53 [11]. The platform integrates an on-chip power-supply monitor configurable as a digital storage oscilloscope (OC-DSO) [72] that is ideal for validating our proposed EM methodology. The OC-DSO provides fine-grained sampling (up to 1.6GHz bandwidth) of the voltage rails supplying the dual-core Cortex-A72 cluster.

OC-DSO reports 8-bit raw values. To extract voltage readings in millivolts calibration is required. To calibrate the OC-DSO we sweep the supply-voltage from 700mV to 1050mV in steps of 10mV and at each voltage-step we measure the raw counter value. We plot the results of the sweep in Figure 27. The graph shows a linear correlation between voltage and counter values with gradient equal to 6.23 and intercept equal to 493.38. We use the discovered linear equation to convert raw values to millivolts. The capability of OC-DSO to capture voltage noise is illustrated in Figure 28. As expected, the dI/dt virus causes much larger noise as compared to SPEC2006 benchmarks and CPU idle state.

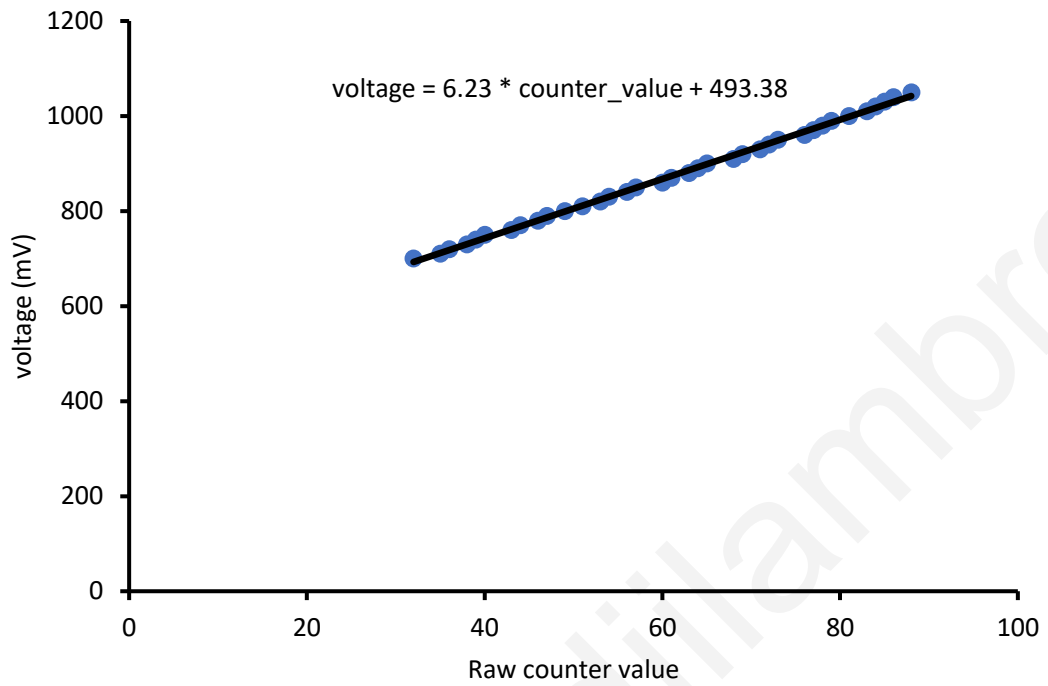


Figure 27. OC-DSO calibration

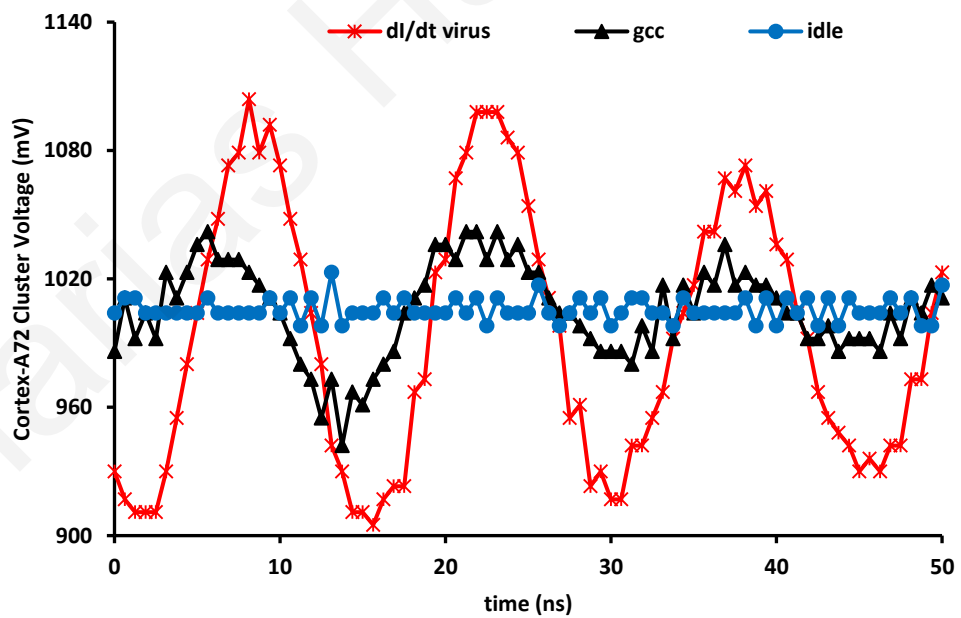


Figure 28. Voltage waveforms obtained from OC-DSO for 3 different workloads. dl/dt virus causes the largest voltage noise.

The JUNO board also offers a synthetic current load (SCL) [73] block integrated in the OC-DSO. The SCL loads the Cortex-A72 PDN with a square-wave current excitation at various frequencies. This is useful for detecting the Cortex-A72 PDN resonance frequency [73] (as shown in Section 4.3) and for validating the EM methodology. Both OC-DSO and SCL are placed in a Power-Delivery-Monitor block that sits on-chip. This is illustrated in Figure 29. The figure shows how small is the OC-DSO. The total size of the Power-Delivery-Monitor macro is approximately $350 \times 310 \mu\text{m}^2$ with a power-consumption of $25 \mu\text{W}$ during waveform capture.

Also, the Juno board runs a Debian OS with a 4.4.0-135-arm64 kernel. The DS-5 debugger [23] is used to access OC-DSO, sweep CPU frequency, change supply-voltage and power-gate both the Cortex-A72 and Cortex-A53 clusters, orchestrated through a system control processor (SCP) that enables this functionality [72]. Please note that the Cortex-A53 cluster does not benefit from the OC-DSO or SCL circuits because it is in a separate voltage domain. Cortex-A53 voltage domain lacks any explicit support for voltage-noise measurement.

For the AMD setup, an Athlon II X4 645 CPU is used that is hosted on an ASUS M5A78L LE motherboard and Windows 8.1 OS. AMD Overdrive application [4] is used to change the voltage and the frequency of the CPU. This application also includes a stability test that is evaluated and compared against the GA generated dI/dt viruses. The motherboard integrates on-package Kelvin measurement pads that enable direct external monitoring of the on-chip voltage rails using differential probes connected to a bench-top oscilloscope (the setup is illustrated in Figure 30).

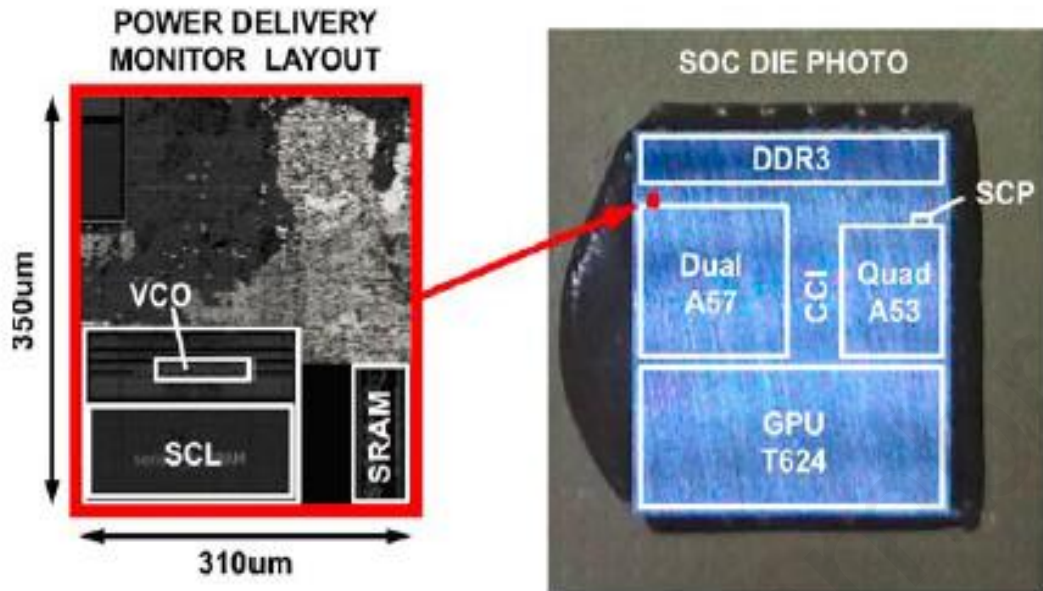


Figure 29. The Power-Delivery-Monitor layout on the left and on the right a die photo of a JUNO R1 SoC [73] (Juno R1 uses the Cortex-A57 CPU whereas the Juno R2 uses the Cortex-A72, apart from some differences in CPU micro-architecture the two SoCs are identical).

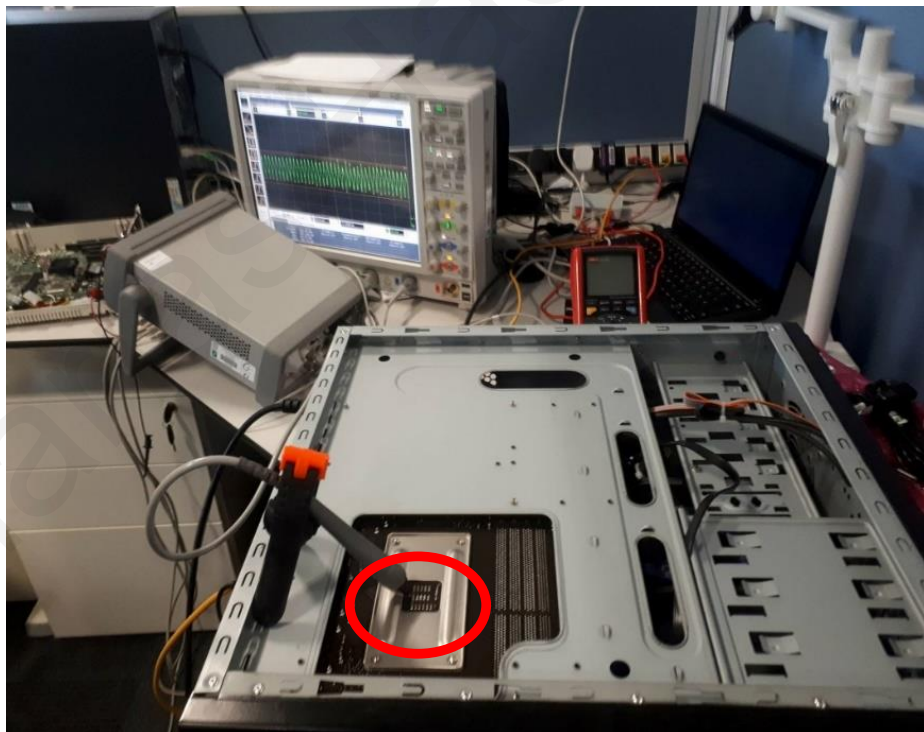


Figure 30. AMD time-domain supply-voltage measurement setup.

The X-Gene2 and X-Gene3 CPUs are hosted on validation boards provided by Ampere Computing. On both boards Centos 7.2 is installed. Also, the boards come with a hardware exposure interface (HEI) provided by Ampere Computing for monitoring various metrics such as CPU temperature and power. The HEI also allows changing the CPU voltage and frequency. The HEI is accessible through i2c Linux utility. Moreover, a voltage droop detector is integrated on X-Gene3 for monitoring dI/dt events. The detector can be programmed on boot to monitor two droop threshold values. The detector reports how many times and for how many cycles a voltage droop that exceeds the threshold values occurred. These metrics are also accessible through the HEI. We use the X-Gene3 droop detector for validating the EM methodology. X-Gene2 does not support any high-bandwidth voltage-measurements for monitoring dI/dt events.

To confirm that our EM generated dI/dt viruses produce large voltage-noise and have high V_{MIN} we compare them against other conventional workloads. We choose workloads from the following benchmark suites: a) SPEC2006, b) SPEC2017 c) NAS and d) common Windows OS workloads (e.g. Blender benchmark). Previous work showed that these workloads expose micro-architectural events that cause voltage-noise [42][65][66] (such as branch miss-predictions, cache-misses), hence, we can use these workloads for our evaluation purposes.

Also, we validate that there are no random signals that will interfere with our measurements. We measure all the random signals emitted in our lab environment in the range of 1 to 3000MHz. Figure 31 shows the result of the sweep. We observe a rather flat EM amplitude across the whole spectrum with a few exceptions. Prominent spikes are observed at 2.4GHz which are attributed to the WIFI internet access that is available in our lab. These signals do not interfere with our experiments as they are located far away from our frequency-spectrum of interest which is located somewhere between 50-200MHz.

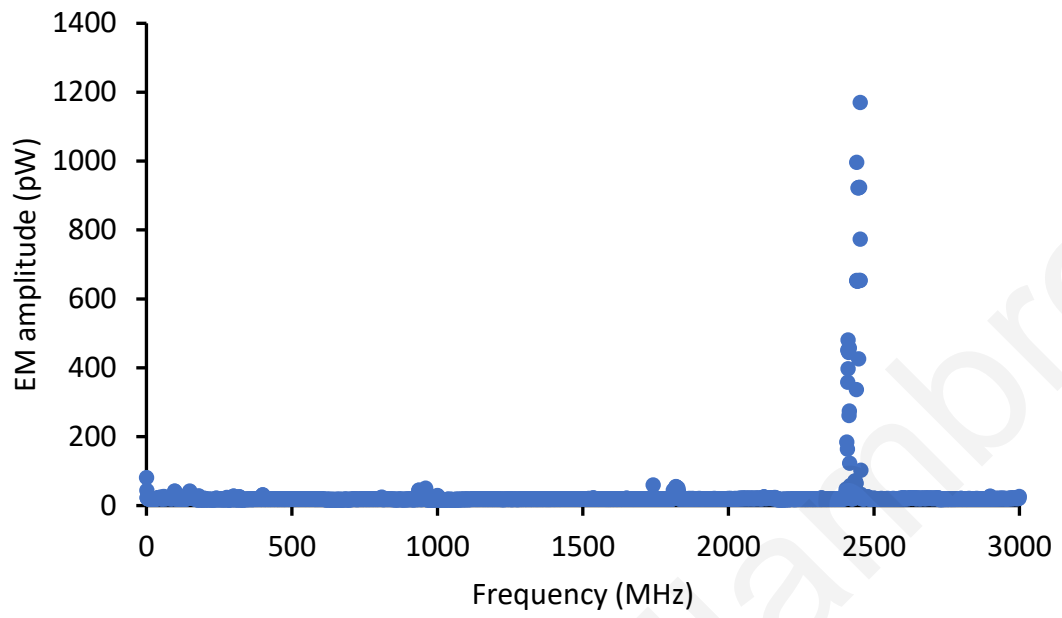


Figure 31. Measurement of random signals in the frequency-spectrum of 1 to 3000MHz.

Chapter 6. PDN Characterization

6.1 ARM Cortex-A72

First, we evaluate the EM methodology on the Cortex-A72 CPU. A GA search is performed with target to produce a stress-test that maximizes the EM amplitude at the resonance-frequency and, hence, generate high voltage-noise. We first determine the loop size using the heuristic proposed in Section 4.5. This requires knowing the resonance frequency, the CPU frequency and the maximum core IPC. As shown in Section 4.3 the resonance frequency of the Cortex-A72 PDN is around 70MHz. The CPU frequency is 1.2GHz and we find with a GA IPC search that the Cortex-A72 can sustain maximum IPC of 3. Given these values, the heuristic suggests a virus with loop size of 20 instructions. Therefore, we perform the GA search on Cortex-A72 with loop size set to 20 instructions.

Figure 32 shows how the EM amplitude, the maximum voltage-droop (measured with OC-DSO [72]) and the dominant frequency (the one with the highest EM amplitude) of the strongest individual of each generation varies as the GA progresses. The signal amplitude

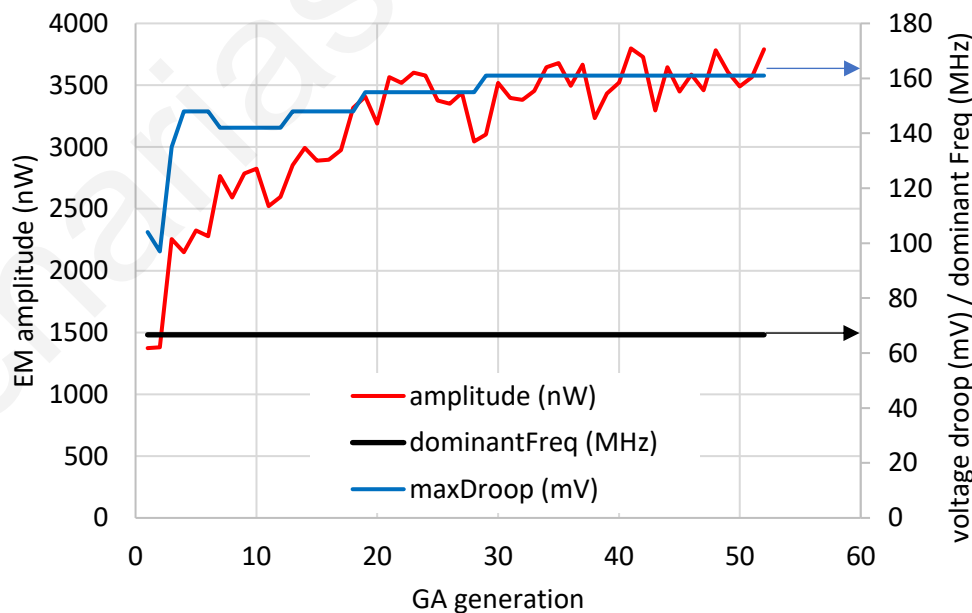


Figure 32. EM driven GA run on Cortex-A72. Peak amplitude (left y-axis) and maximum droop / dominant frequency (right y-axis) for the best individual of each GA generation.

increases from generation to generation during the GA search along with the voltage droop. This shows that the EM generated virus corresponds to a dI/dt virus that maximizes voltage-droop. The virus dominant frequency is 67MHz that is very close to the resonance frequency of the Cortex-A72 identified in Section 4.3.

To confirm that indeed the obtained virus is really causing large voltage noise, more than typical workloads, we perform V_{MIN} testing. Figure 33 compares the voltage-droop and the V_{MIN} of SPEC benchmarks and the dI/dt virus produced by GA search with 20 instructions based on EM emanations. As shown in Figure 33 the virus causes larger voltage droop and has higher V_{MIN} than the SPEC benchmarks. And, it has the same V_{MIN} and voltage-droop as a dI/dt virus generated with a GA search guided from the OC-DSO (denoted as OC-DSO_VIRUS). This is clear indication that the proposed EM based methodology is effective in generating dI/dt viruses.

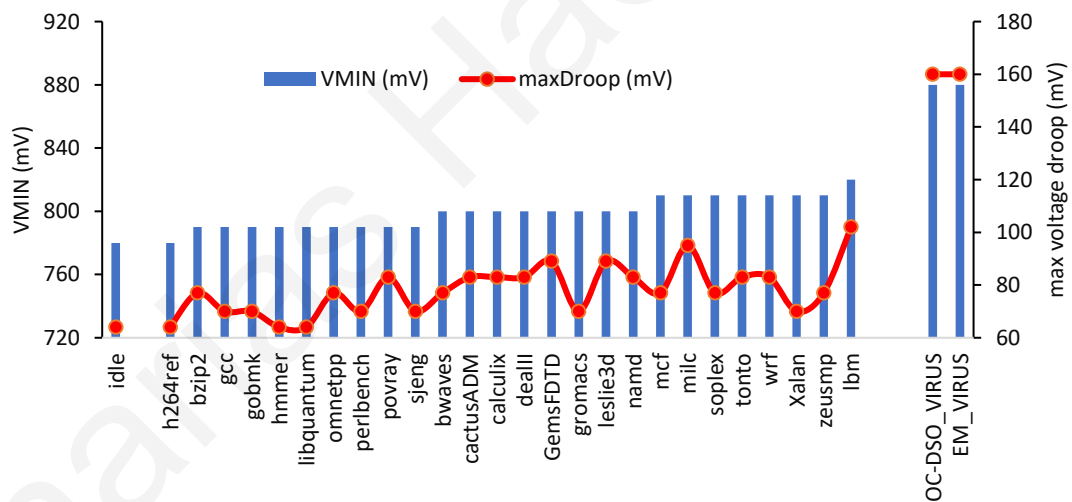


Figure 33. Voltage droop and V_{MIN} measurements on Cortex-A72.

Next, we show the benefit of using the loop size heuristic. We compare how the GA search for Cortex-A72 with 20 loop instructions compares to a GA search with 50 loop instructions. Note that 50 loop instructions is a value that we empirically find to work generally well in most cases. Figure 34 compares the GA progress in terms of maximum voltage-droop between the GA search with 20 instructions and the GA search with 50 instructions. The search with 20 instructions converges much faster to a more powerful

virus. At generation 30, the GA search with 20 instructions causes a maximum voltage droop of 160mV, whereas the GA search with 50 instructions causes a voltage droop of 114mV. Figure 34 also shows that the search with 50 loop instructions even after 60 generations does not produce a virus of equal strength as the search using 20 instructions. This clearly highlights the effectiveness of the loop-size determination heuristic in generating faster stronger dI/dt viruses

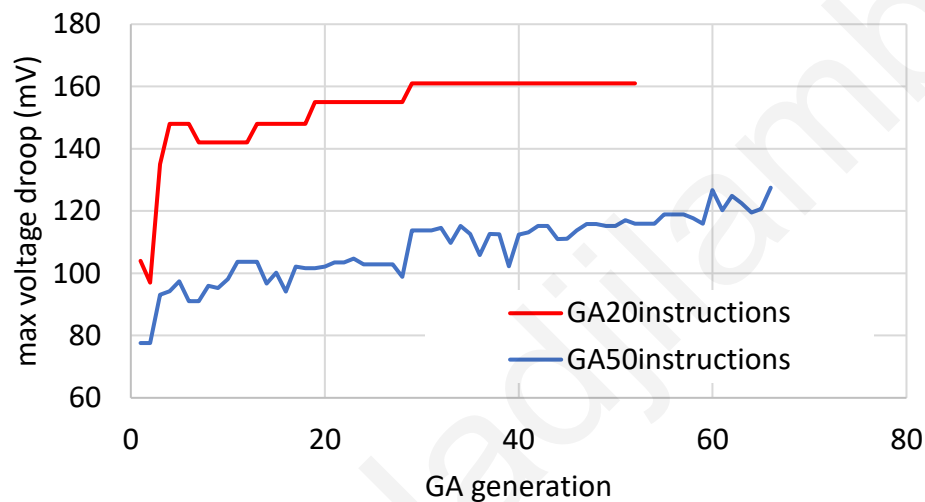


Figure 34. GA search for Cortex-A72 dI/dt virus with 20 loop instructions vs 50 loop instructions.

6.2 ARM Cortex-A53

Cortex-A53 cluster does not provide any support for direct voltage-noise measurements rendering dI/dt virus generation and resonance frequency identification impractical with state-of-the-art means. This section shows that the EM methodology circumvents this shortcoming and obtain a) a virus that stresses voltage margins, and b) the first-order resonance frequency. This underlines the effectiveness and the generality of the proposed methodology.

We conduct a GA optimization run with the objective of obtaining a voltage-noise virus for the Cortex-A53 cluster. Figure 35 shows the inter-generational progression of the GA (left-axis showing received EM-power and the right-axis showing the dominant frequency of the strongest individual per generation). The GA successfully maximizes the EM

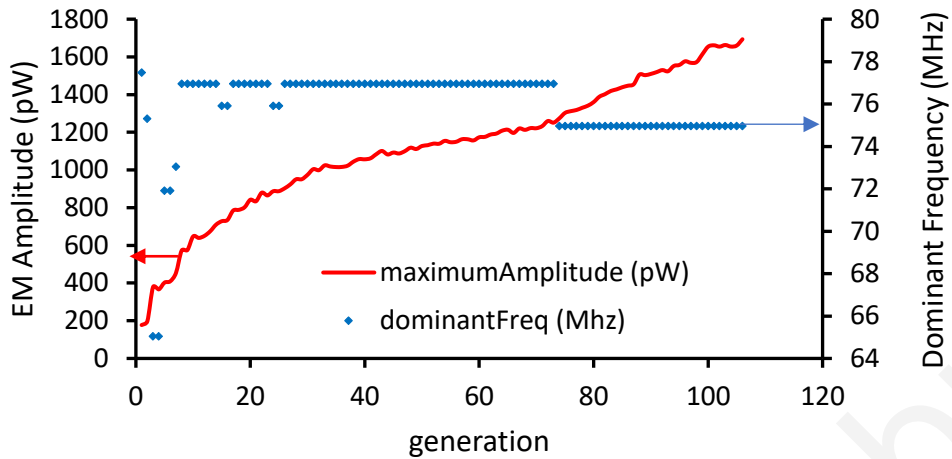


Figure 35. GA EM amplitude driven optimization for Cortex-A53.

amplitude. Since, Cortex-A53 does not support voltage noise measurements to test the effectiveness of the GA we compare the V_{MIN} of the strongest individual across all generations (labelled “EM virus”) against the V_{MIN} of SPEC2006 benchmarks.

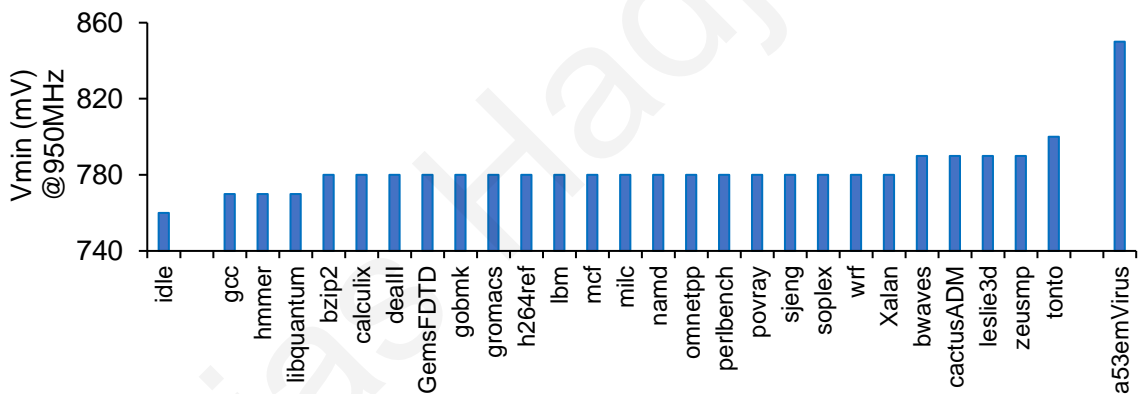


Figure 36. V_{MIN} measurements on Cortex-A53.

Figure 36 shows the V_{MIN} of the EM virus (rightmost) compared to SPEC2006 benchmarks and idle (leftmost). The V_{MIN} is obtained with four active cores at a 950MHz CPU frequency. The V_{MIN} of the generated EM virus stands out (50mV higher) compared to the rest of the benchmarks which demonstrates the effectiveness of the EM approach in generating dI/dt viruses.

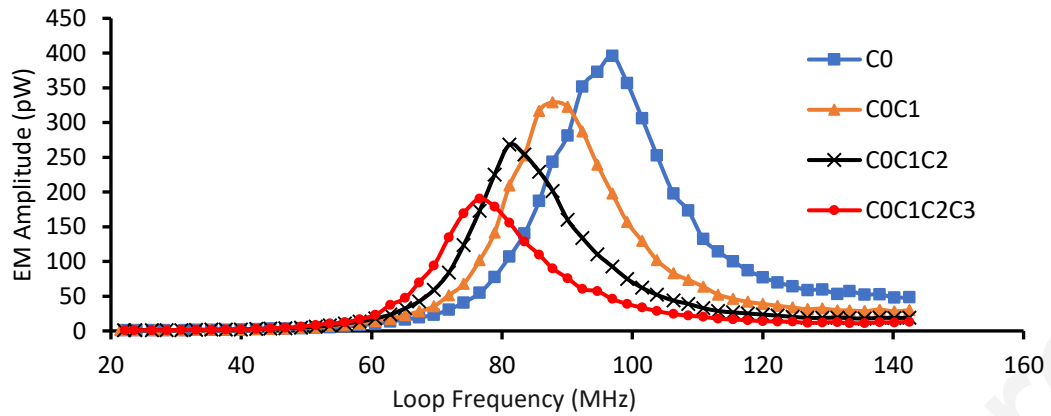


Figure 37. Resonance frequency exploration on Cortex-A53. For four powered cores (C0C1C2C3) the resonance frequency is 76.5MHz.

The GA converges to 75MHz dominant frequency. We use the loop method methodology for resonance-frequency detection (described in Section 4.3) to validate that this is the first-order resonance frequency of the Cortex-A53 cluster. The results of the sweep are shown in Figure 37. For four powered cores (C0C1C2C3 scenario) the sweep reveals a resonance frequency at 76MHz which matches closely the GA results. The agreement of the two independent approaches gives confidence that the resonance frequency is correctly identified.

Furthermore, Figure 37 provides insight about how power-gating can affect significantly the PDN characteristics. The Cortex-A53 quad-core cluster has the highest die capacitance when all four cores are powered up (“C0C1C2C3”). The first-order resonance frequency is inversely proportional to the square-root of the die capacitance [55], hence, the resonance frequency increases from 76.5MHz when all cores are powered up (labelled as “C0C1C2C3”) to 97MHz with just one core powered up (labelled as “C0”). Note that the amplitude of the EM emanations is affected by the number of powered cores in addition to the resonance frequency. Since we kept stable current consumption across all four scenarios by having only the first core active, the EM amplitude (and hence the voltage noise) is maximized in the scenario where the least PDN capacitance is present (“C0”). These results

confirm prior work [37] that shows that with more cores connected under the same PDN, the capacitance increases and voltage noise smooths out.

Moreover, the results indicate that power-saving techniques, such as power-gating individual cores, whilst being beneficial from a leakage perspective, can affect power-delivery adversely. Power-gating not only reduces the available useful capacitance that can mitigate high-magnitude voltage-droops, but also makes the frequency of voltage-noise oscillations higher. This has detrimental implications on voltage-noise mitigation mechanisms such as adaptive-clocking [31][44], that are extremely sensitive to response-latency.

6.2.1 Simultaneous Voltage Noise Monitoring of Multiple Voltage Domains

We next illustrate the capability of the EM based methodology to monitor multiple voltage domains simultaneously. This is impossible with an on-chip or off-chip oscilloscope that has a direct physical probing on a single voltage domain. In contrast, an antenna can detect voltage emergencies happening at the same time on both the Cortex-A72 and Cortex-A53. To demonstrate this capability, we run the Cortex-A72 and Cortex-A53 dI/dt viruses at the same time and capture the spectrum analyzer readings as shown in Figure 38. The frequency-domain signatures of both viruses are clearly visible. This shows that the EM

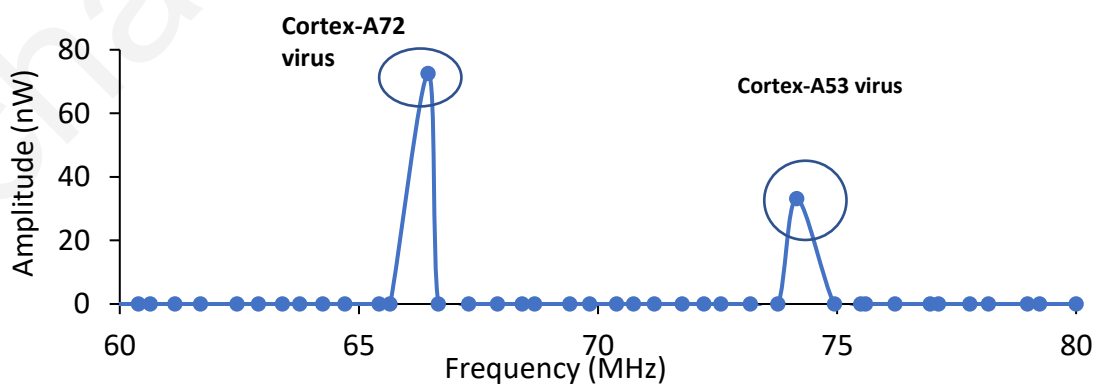


Figure 38. Simultaneous monitoring of voltage emergencies across multiple voltage domains through EM emanations.

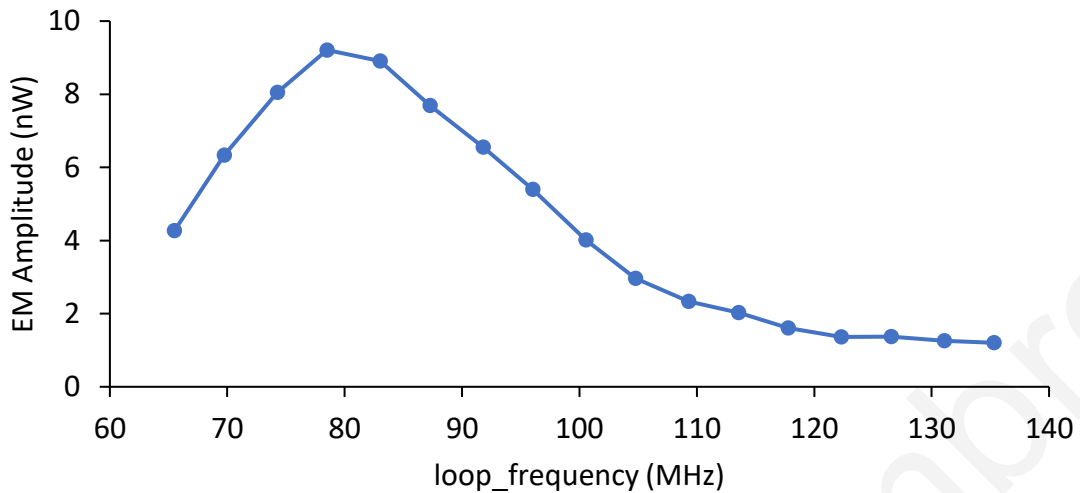


Figure 39. Loop frequency sweep on Athlon II X4 645 reveals a resonance frequency at 78MHz.

methodology offers an effective detection mechanism for voltage-noise oscillations occurring across multiple voltage domains, thereby underlining its applicability to heterogeneous System-on-Chips (SoCs).

6.3 AMD Athlon II X4 645

This section extends the evaluation from low-power mobile CPUs and the ARM ISA to high power x86-64 desktops (AMD Athlon II X4 645). The loop method frequency sweep methodology for finding the resonance frequency (Section 4.3) is performed on the AMD CPU and the results are shown in Figure 39. The sweep reveals the first-order resonance frequency to be at 78MHz. An EM amplitude driven GA run shows excellent agreement converging to nearly the same resonance frequency (77MHz) as shown in Figure 40. The EM amplitude during the GA search follows the same trends as in the Juno board CPUs, it increases with each generation until it eventually converges.

For V_{MIN} comparison, the GA auto-generated EM virus is compared against common Windows (and Desktop CPU) workloads. The benchmark suite includes CPU intensive video rendering workloads such as Blender [12], Cinebench [19], scientific workloads such as Euler 3D [25] and all-around benchmark suites such as WEBXPRT [71] (mimics browser workloads) and GeekBench [28] (set of common workloads e.g. encryption, database

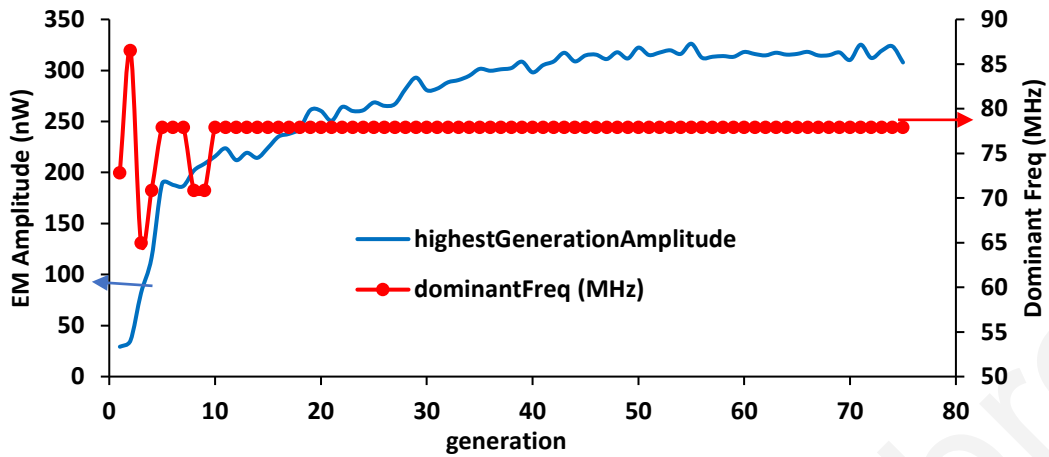


Figure 40. GA EM amplitude driven run on AMD CPU.

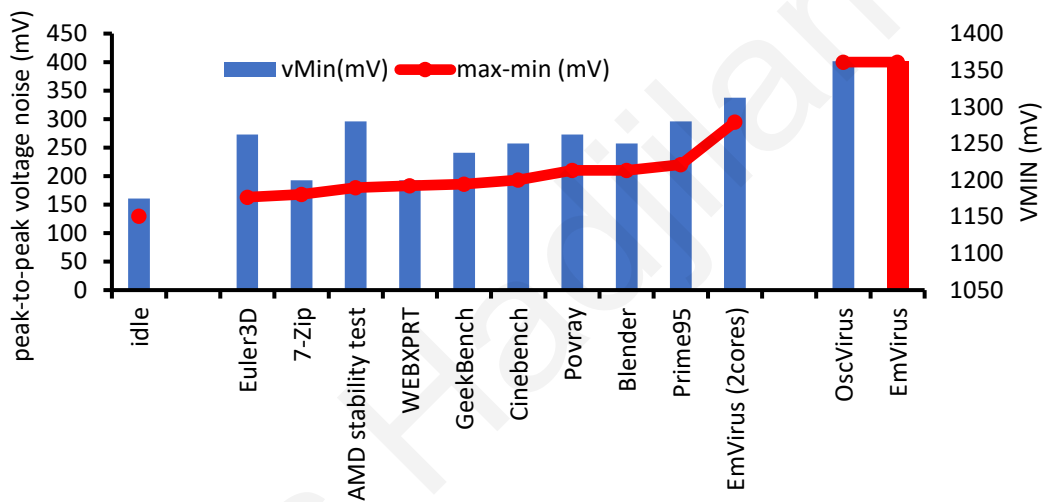


Figure 41. V_{MIN} and voltage noise measurements on the AMD CPU.

queries etc.). Moreover, the EM virus is compared against the well-known Prime95 [61] stability test, AMD’s own stability test application [4], and a GA virus generated through the voltage feedback from on-package Kelvin measurement pads (denoted as OscVirus). We monitor on-die voltage noise using a differential probe connected to an oscilloscope (Figure 30). The V_{MIN} and voltage noise results are shown in Figure 41. Unless noted otherwise, all measurements are with all four cores active.

The GA viruses (EMvirus, OscVirus) cause much higher voltage noise and have higher V_{MIN} as compared to the rest of the workloads. The EM driven GA approach again is effective in generating voltage-noise viruses. The EM virus has a V_{MIN} of 1.3625V, 37.5mV below the nominal voltage at 3.1GHz. It is interesting to point out that the EM based virus

running on only two active cores is more severe than the AMD stability test and Prime95 on four active cores. To gain confidence in the V_{MIN} results we have run the AMD stability test and Prime95 for 24 hours at 1.287V and 1.28V respectively. They both pass the test whereas the EM virus causes immediate system-crash at 1.3V or even higher voltages.

6.4 Ampere Computing X-Gene2

In this section, we evaluate the EM methodology on the X-Gene2 CPU. Compared to the other evaluated CPUs the X-Gene2 has a distinct characteristic that affects the dI/dt virus generation; X-Gene2 cores are clustered into Processor-Modules (PMD). Each PMD includes two cores with a private DL1 cache and a shared L2 cache. The DL1 caches are write-through [82] so each time a write is performed in the DL1 cache, the L2 cache is written as well. Therefore, when two threads run on the same PMD L2 cache contention and interference across threads should be the norm. Hence, the approach of performing the GA search on single core and then testing the resulted virus by running it on all cores does not work for X-Gene2 because execution of multiple instances of a derived virus, one instance in each core, usually suffers from timing interference that impedes resonance built-up. To mitigate this issue, we propose to perform the GA search on a single PMD but with two instances of the virus running, one on each PMD core. With this approach we observe that the GA naturally converges to solutions that do not suffer from contention on the L2 cache and cause high voltage-noise.

We use the loop-size heuristic (Section 4.5) to estimate the number of instructions for the GA optimization on X-Gene2. Recall that to use the heuristic, we need to determine the CPU frequency, the maximum IPC and the PDN resonance-frequency. The nominal CPU frequency is known from the specifications and is equal to 2.4GHz [82]. The maximum IPC is 4 [82] which we confirm with a GA IPC optimization. To find the resonance frequency

we use the loop method for quick resonance-frequency determination described in Section 4.3.

Figure 42 shows the results of the PDN resonance-frequency exploration on X-Gene2. The sweep results suggest a resonance frequency at 150MHz. We observe gaps between the data points in Figure 42 because X-Gene2 supports only discrete CPU frequency values in steps of 300MHz. Based on these values the heuristic suggests using loop size of 25 instructions during the GA optimization. The GA search maximizes the EM amplitude at 150MHz as shown in Figure 43. We confirm the virus effectiveness by performing a V_{MIN} comparison against SPEC2017 and NAS benchmarks. The results in Figure 44 reveal that the GA virus (rightmost) has higher V_{MIN} than the other workloads.

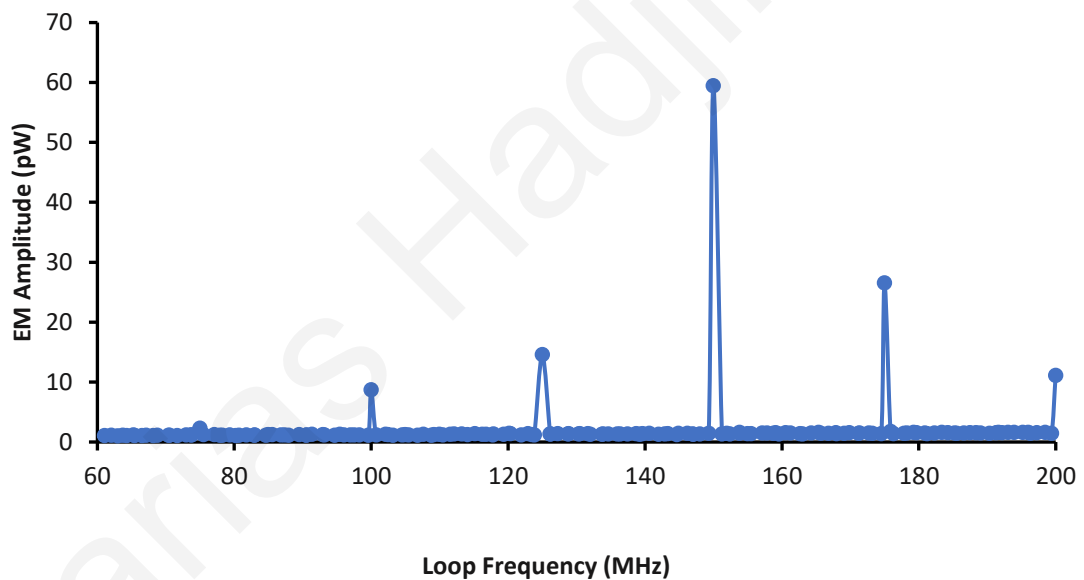


Figure 42. X-Gene2 resonance frequency exploration

6.5 Ampere Computing X-Gene3

We apply the EM methodology on X-Gene3 to generate a dI/dt virus. We validate the effectiveness of the generated virus with both V_{MIN} measurements and voltage droop

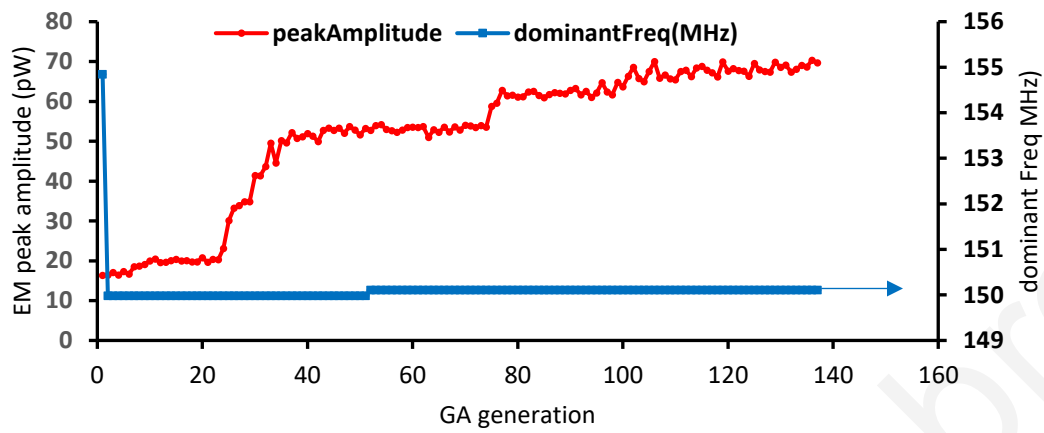


Figure 43. GA search on X-Gene2.

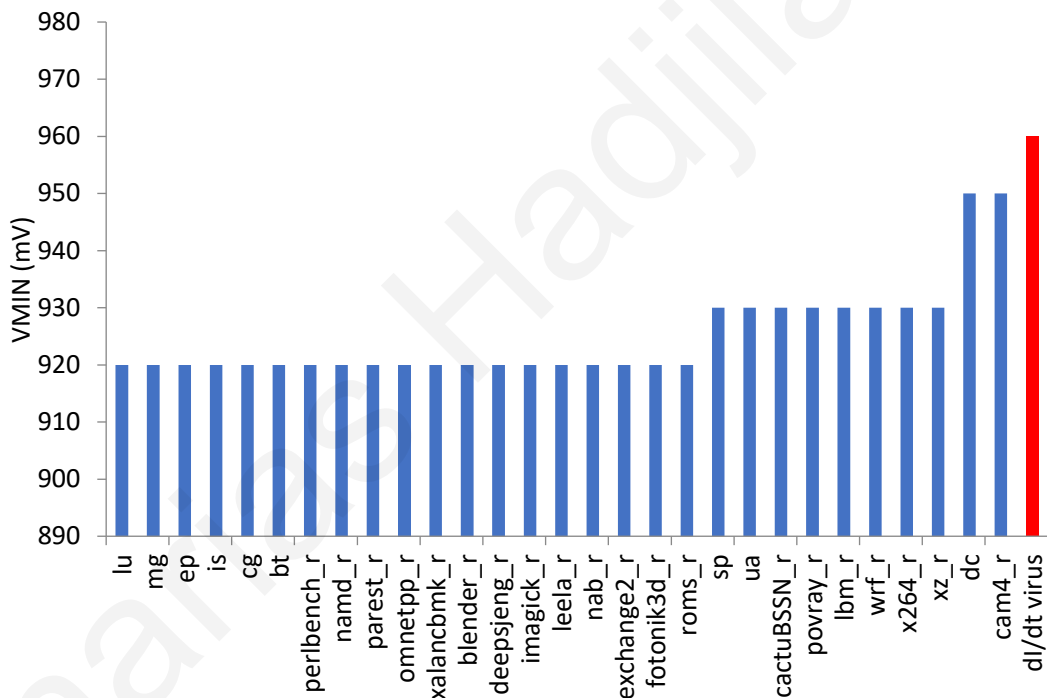


Figure 44. X-Gene2 V_{MIN} results.

measurements using the voltage droop detector circuit that is integrated on the X-Gene 3 chip.

Figure 45 shows the max voltage droop caused by the dl/dt virus compared to NAS parallel benchmarks at 32 active cores and 3GHz CPU frequency. It is shown that the virus is the only workload that causes 80mV voltage droops. Furthermore, the droop detector allows

counting the number of CPU cycles that suffered a voltage droop per second. Figure 46 shows the number of CPU cycles that suffered a 40mV voltage droop per second while running the virus and the SP NAS benchmark (the NAS benchmark with the highest droop count) for a different number of active cores. The first observation is that the virus causes orders of magnitude more droops than the SP benchmark. This confirms that the dI/dt virus behaves as intended by causing periodically voltage droops at a rate equal to the PDN resonance frequency. Another observation is that the droop count increases along with the number of active cores. For instance, the SP benchmarks start to cause droops of 40mV only above 16 active cores. This is expected as with more cores actives more current (I) is drawn, hence, higher voltage droops are generated [42]. We also observe that the SP curve is more flat compare to the dI/dt virus i.e. the droop increase with a higher number of active cores is more apparent for the dI/dt virus. This can be explained by the fact that the virus is a tiny loop of 50 instructions, therefore, during virus execution the cores are more easily aligned, which has a constructive effect on voltage-noise as the number of cores increase.

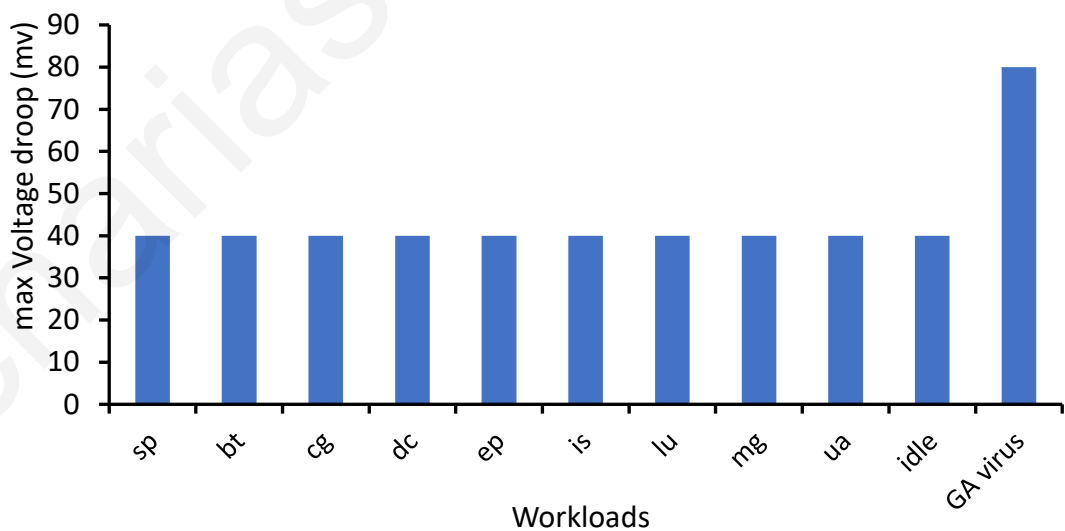


Figure 45. Voltage droop of virus vs NAS workloads.

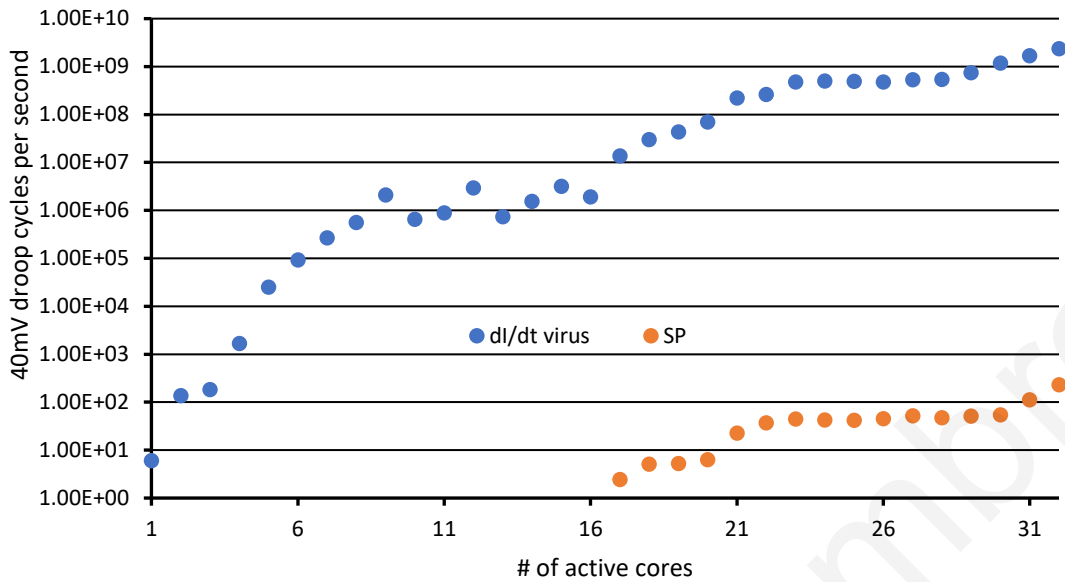


Figure 46. Cycles suffered droop per second versus number of active cores.

Regarding V_{MIN} , as shown in Figure 47 the virus has equal V_{MIN} to the SP benchmark despite (as shown in Figure 45 and Figure 46) virus causing higher voltage-noise (both in terms of quantity and magnitude). A possible explanation for this behaviour might be the Load-Line calibration feature that is supported by the X-Gene3.

Load-Line calibration is the mechanism that automatically adds a voltage-offset to the desired operating voltage (the one set by the user or the manufacturer) to compensate for voltage droops that occur due to changes in CPU power consumption. This mechanism responds to a voltage drop in scales of milliseconds and it addresses the IR drop. It is not intended to address the sudden inductive dI/dt voltage droops, such as the one caused by a dI/dt virus. Despite that, at least in this case, it seems to work as a counter against our X-Gene3 dI/dt virus. This is the case because it happens that the virus causes higher power-consumption than the rest workloads. The dI/dt virus causes on average 20W more CPU power consumption than the NAS benchmarks. This results in more voltage offset being added during virus execution.

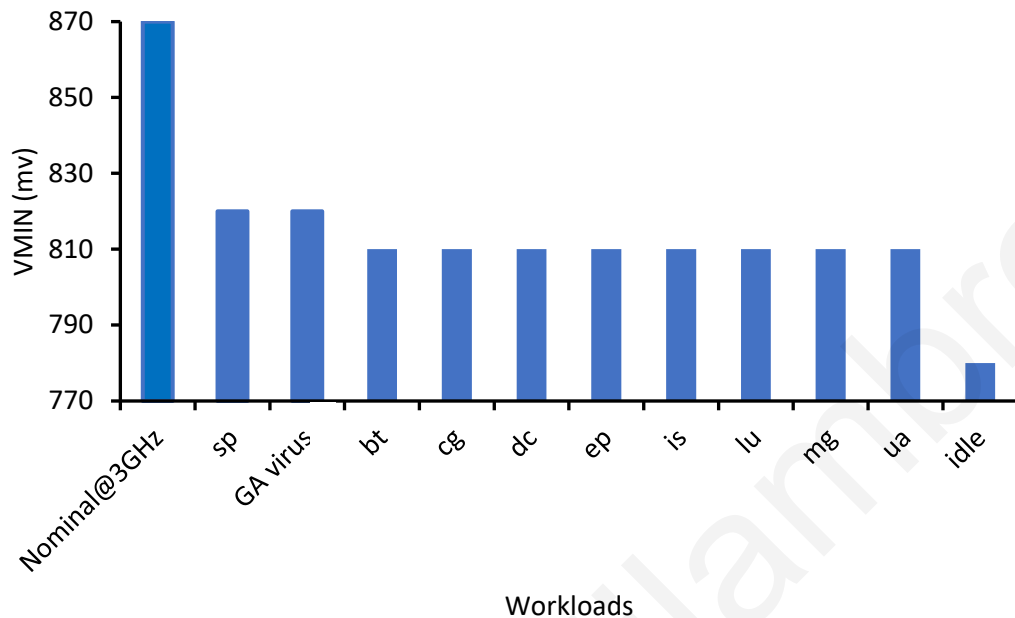


Figure 47. V_{MIN} measurements on X-Gene 3.

Figure 48 show the operating voltage over time for NAS, dI/dt and idle workloads when the voltage regulator is instructed to set the operating voltage to 820mV. We observe that even at idle a small voltage offset of 3mV is added (except for one spike at 824mV that could be correlated with a sudden activity from background system processes). When CPU is not idle (during NAS and dI/dt execution) the Load-Line calibration raises the voltage much higher to compensate for the IR drop (the drop of voltage at the ends signifies the end of execution). Because the dI/dt virus causes higher power consumption, the operating voltage during virus execution is raised from 3 to 6mV higher compared to SP workload. This shows that V_{MIN} testing with Load-Line calibration enabled is not exactly a fair comparison since the actual operating voltage is dependent on the workload power-consumption (instead of just being equal to the voltage asked by the experimenter). Unfortunately, we cannot disable Load-Line calibration to check if the V_{MIN} results will be different. Another implication of these results is that during dI/dt virus development, emphasis must be given not only on dI/dt voltage-droop but on average power consumption

as well. Our results support the following: a) generating voltage-noise viruses with low average power-consumption when Load-Line calibration is present to reduce the automatically added voltage-offset, and b) generating voltage-noise viruses with high average power-consumption when Load-Line calibration is not present to increase the overall voltage droop with higher IR droop.

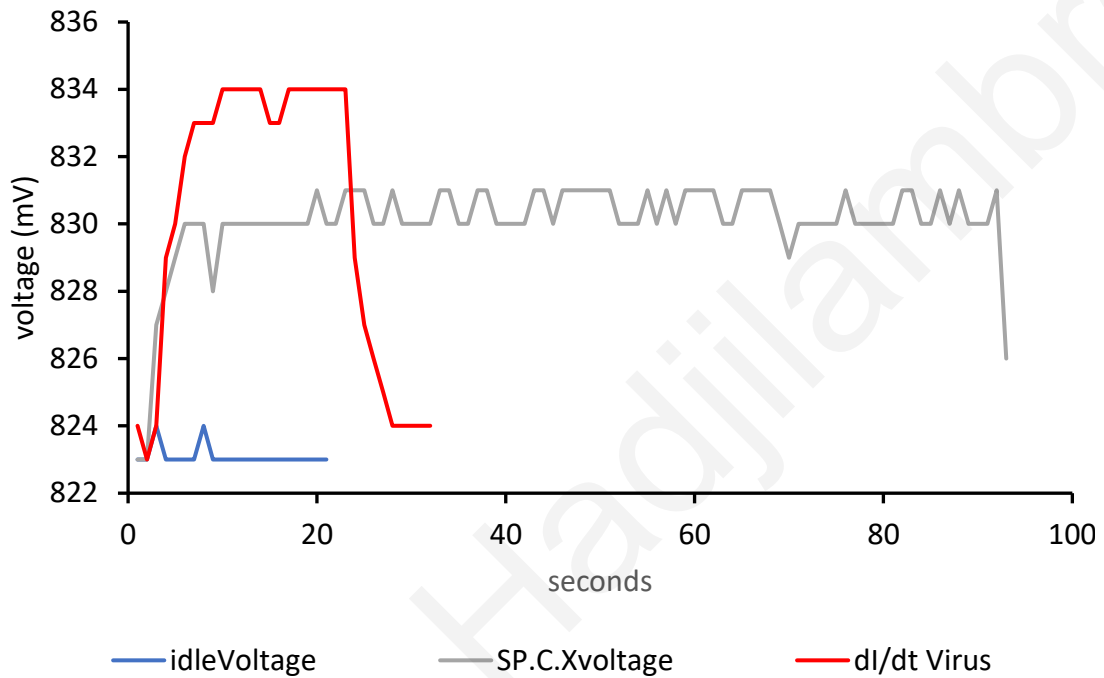


Figure 48. Voltage over time for 3 different workloads (idle, SP and dI/dt virus).

Chapter 7. EM based DVS Governors

7.1 EM Detection Governor

We propose a DVS governor that exploits the correlation between EM amplitude and voltage-noise. The governor detects workloads that cause high-voltage noise by continuously monitoring the EM power levels emitted by the CPU's PDN. If the EM amplitude surpasses a certain threshold the workload is considered unsafe. In that case the governor will set the CPU voltage to a high safe value. Otherwise, if the EM power does not exceed a certain threshold, the governor sets the CPU voltage to a lower more optimistic value.

The DVS governor EM detection setup for a X-Gene2 system is shown in Figure 49. Essentially, is the same EM setup as we shown in Section 4.1 for the Juno board and the AMD desktop CPU. The antenna senses the EM signals and the spectrum analyser monitors EM spikes at the resonance frequency. A workstation is connected to the spectrum analyser

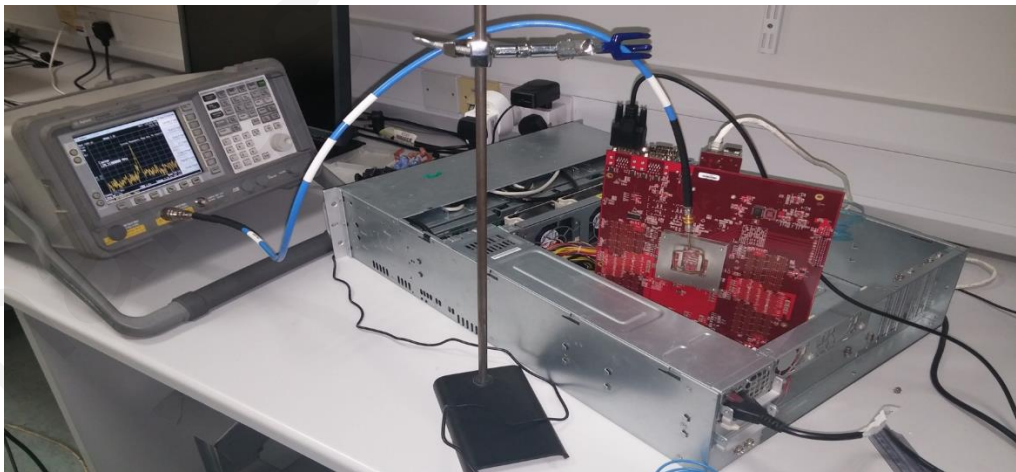


Figure 49. DVS governor EM detection setup on X-Gene2.

(not shown in the picture). The workstation analyses the strength of the signals and if a threshold is surpassed, it will act to enforce stability by changing the voltage to a safe value.

This DVS governor requires the following parameters to work: a) a safe voltage called WorkloadVMIN for safely executing regular workloads (this can be identified through extensive V_{MIN} characterization of regular workloads), b) a safe voltage for running a dI/dt workload which can be equal to the V_{MIN} of a dI/dt virus (we will refer to this as VirusVMIN), c) the threshold of the EM signal amplitude that is considered dangerous (this can be identified by characterizing the EM of a dI/dt virus characterization) and d) the resonance frequency of the CPU's PDN, this is the frequency that the spectrum analyser will monitor.

The predictor algorithm is the following: a) Upon a new workload start set the voltage to VirusVMIN, b) Monitor the workload for few seconds, c) if the EM amplitude doesn't exceed the threshold, then, lower the voltage to the WorkloadVMIN, d) if the EM amplitude exceeds the threshold, then, keep the voltage at the VirusVMIN. These steps are illustrated with a flowchart in Figure 50. In the figure the VirusVMIN is equal to 980mV and the WorkloadVMIN is equal to 930.

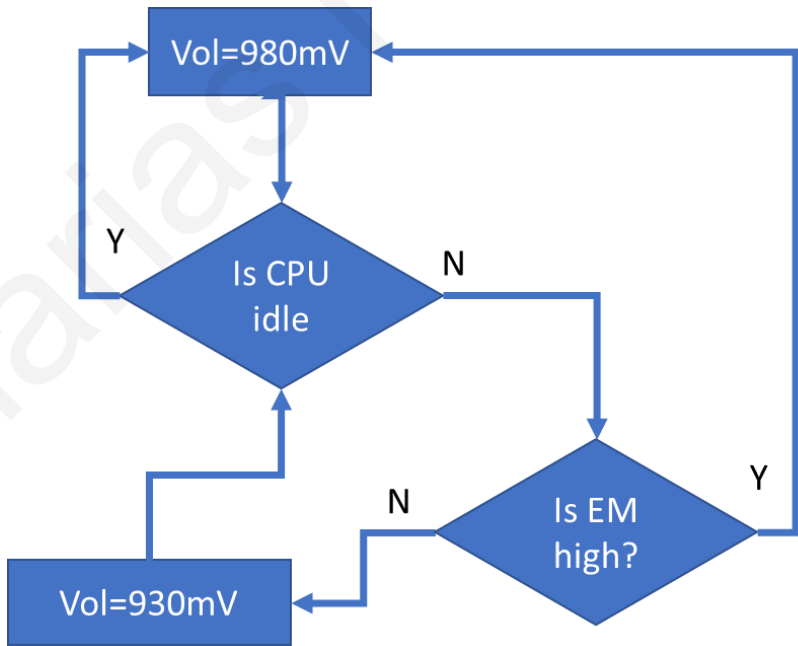


Figure 50. EM predictor flow-chart.

We evaluate the EM detection governor on X-Gene2. We have executed the EM detection governor for 12 hours under a random workload. For virus workloadVMIN, we use 930mV. Figure 51 shows the moving average power consumption of EM governor versus the moving average power consumption of the CPU executing under nominal voltage. The EM governor provides lower power consumption. In total the EM predictor provides 10% power-savings.

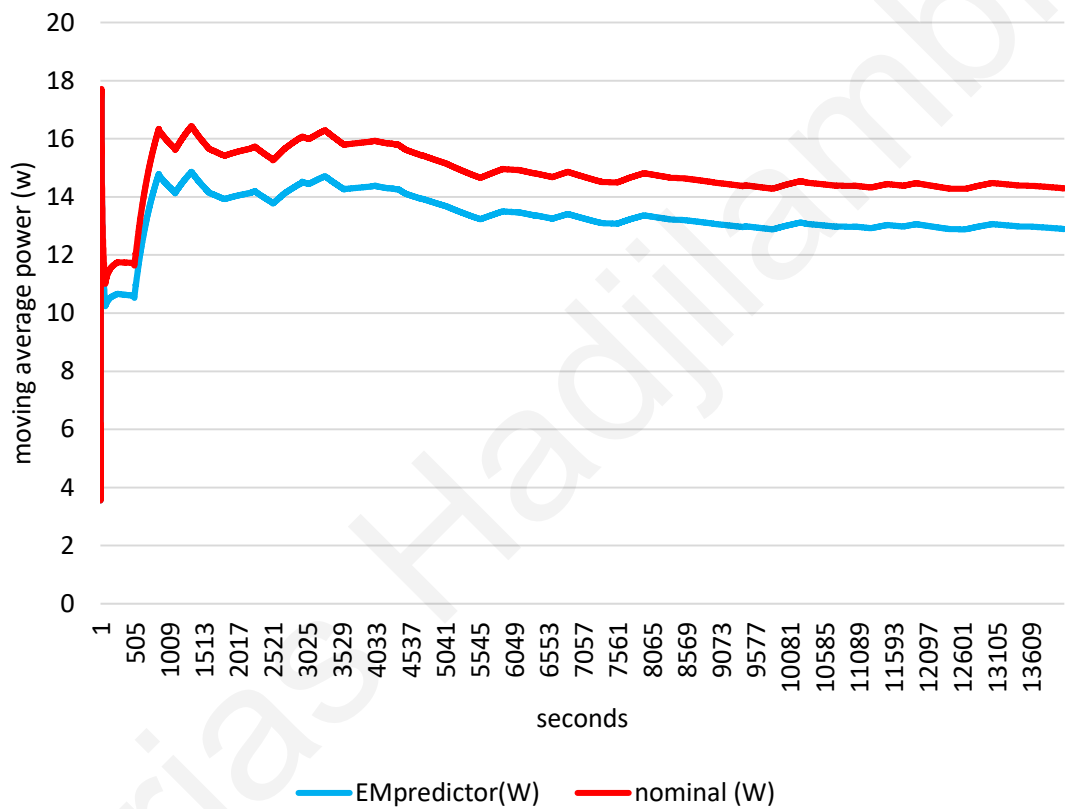


Figure 51. EM predictor vs nominal.

Is worth highlighting that the EM predictor has not yet been practically realised in real deployments. But the effectiveness of this prototype motivates the implementation of this approach in future designs by incorporating both the antenna and the actuator logic inside the chip.

7.2 Core Allocation Governor

The dI/dt viruses produced with the EM methodology can assist in DVS decisions for improving the CPU energy-efficiency. In particular, we propose to characterize offline the V_{MIN} of dI/dt viruses for different core-allocation classes (i.e. for different active cores) and implement a DVS governor that: a) continuously monitors which cores are active, b) identifies the core-allocation class based on the current active cores, c) determines the V_{MIN} of the core-allocation based on the off-line characterization, and d) sets the operating voltage to the V_{MIN} of the current core-allocation.

We evaluate the core-allocation governor on the X-Gene2 CPU. One of the fundamental questions for implementing the core-allocation governor is how many different core allocations are characterized. Broadly speaking the most determining factor for dynamic voltage scaling (DVS) decisions is the number of active cores. More active cores lead to higher current (I) consumption and higher voltage droops, and, it is generally accepted that a system's V_{MIN} is higher when more cores are active [42][49][73]. Therefore, one approach would be to characterize the V_{MIN} of the system under different number of active cores. But in cases where the CPU design introduces asymmetric performance among cores, the number of active cores is not the sole determining factor for the magnitude of the voltage-noise [42]. Examples of such CPU designs are Simultaneous-Multi-Threading processors (SMT) [81], the AMD bulldozer architecture [42] and X-Gene2 [82].

X-Gene2 implements a PMD based architecture. X-Gene2 cores are clustered into Processor-Modules (PMD). Each PMD includes two cores with their private L1 cache and a shared L2 cache. The L1 caches are write-through to the L2 cache. The sharing of the L2 cache causes performance differences when executing two threads on the same PMD compared to executing two threads across two PMDs. Hence, to apply the core-allocation governor on X-Gene2 we must characterize the system V_{MIN} for different number of fully active PMDs (i.e. the PMDs where both cores are active henceforth referred to as fp) and

for different number of half active PMDs (i.e. the PMDs where only one core is active henceforth referred to as hp).

For clarity Figure 52 illustrates different core-allocations classes for 4 threads. In total, X-Gene2 has 14 different allocation classes which are shown in Table VII. For each allocation class, a subset of allocation-instances is shown. Allocation-instances are basically the ids of the active cores that implement an allocation. For example, the combination of cores “0,1,2” implements a 1fp,1hp allocation class. In total, X-Gene2 has 255 allocation-instances (all the possible active core combinations minus one, $2^8 - 1 = 256 - 1 = 255$).

It is also important to consider which cores are going to be characterized for each allocation class. Naïve selection of the cores that will be characterized could result in not optimal power-efficiency or system failures. For instance, applying the V_{MIN} of a 2fp allocation-instance that uses cores 0,1,2,3 to a 2fp allocation-instance that uses cores 4,5,6,7 might be dangerous for system-stability if cores 4,5,6,7 have higher V_{MIN} than cores 0,1,2,3. This issue becomes more important with the increasing CPU core counts that render static variations among cores (in terms of V_{MIN}) a higher concern. Therefore, for each allocation

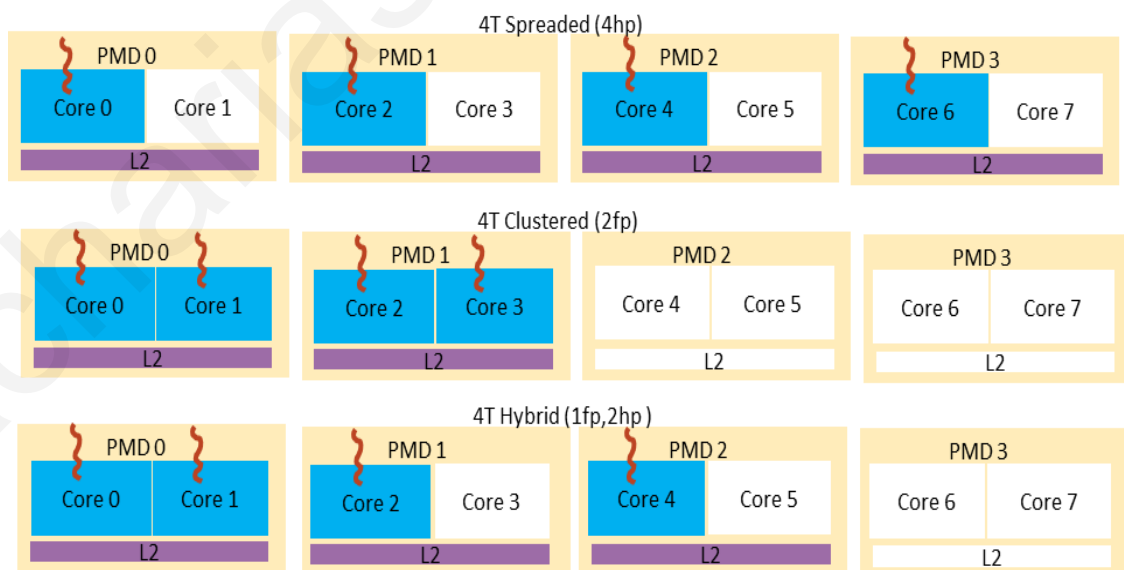


Figure 52. Different core-allocation classes for 4 active threads. Idle cores and L2 are illustrated with white colour.

Table VII. All X-Gene2 core allocation classes.

Allocation Class	Allocation Instances (space separated)	Total number of Allocation Instances
1hp	0 1 2 3 4 5 6 7	8
1fp	0,1 2,3 4,5 6,7	4
2hp	0,2 0,3 0,4 0,5 0,6 0,7 1,2 ...	24
1fp 1hp	0,1,2 0,1,3	24
3hp	0,2,4	32
1fp 2hp	0,1,2,4 ...	48
2fp	0,1,2,3 4,5,6,7	6
4hp	0,2,4,6 1,3,5,7 ...	16
1fp 3hp	0,1,2,4,6 ...	32
2fp 1hp	0,1,2,3,4 ...	24
2fp 2hp	0,1,2,3,4,6 ...	24
3fp	0,1,2,3,4,5 ...	4
3fp 1hp	0,1,2,3,4,5,6 ...	8
4fp (or 8hp)	0,1,2,3,4,5,6,7	1

class we characterize the V_{MIN} of the worst-case allocation instance. The worst-case allocation instance is the instance that consists of the cores and PMDs with the highest V_{MIN} . The V_{MIN} of the worst-case allocation instance will be considered as a safe V_{MIN} for any allocation-instance that belongs to the same allocation class. This approach is based on two expectations: a) the voltage-droop is inherent to the number of active fp and hp (i.e. allocation scenario), b) the core-to-core variations affect the resilience to voltage droops but not the voltage droop magnitude.

We start constructing the DVS governor for X-Gene2 by first finding the dI/dt virus (from Section 6.4) V_{MIN} of all 1hp and 1fp allocation-instances. This is a required first step that enables the identification of the worst-case allocations for the rest 12 allocation classes. The results are shown in Figure 53. The cores 1,6,7 are the most unreliable cores. The PMD runs (both PMD cores active) “0,1”, “2,3”, “4,5”, “6,7” have equal V_{MIN} , and, as expected, they have higher or equal V_{MIN} to the single-core runs. We use these results to construct the worst-case allocations-instances for all rest allocation classes. For example, based on Figure 53 results, the worst-case allocation-instance for the 2hp class are the cores “1,6” (or “1,7”).

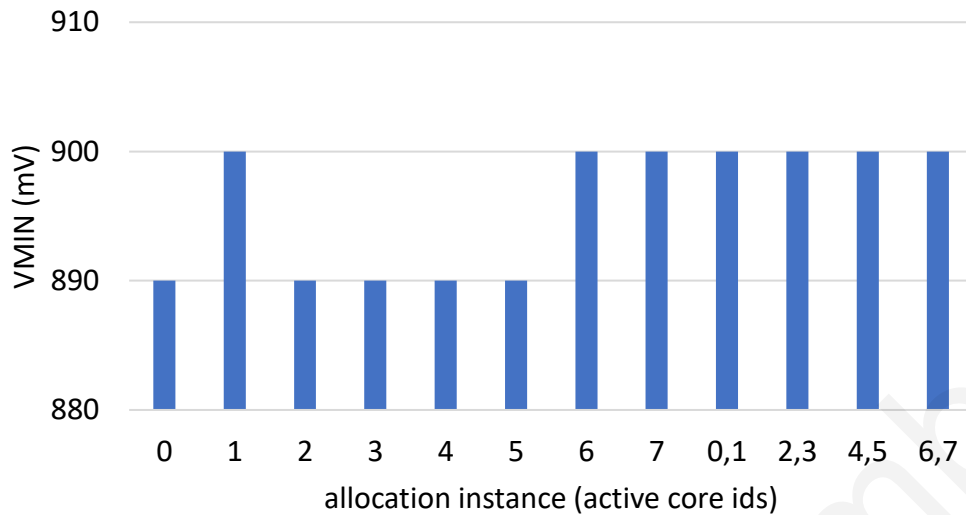


Figure 53. Single core and PMD V_{MIN} characterization.

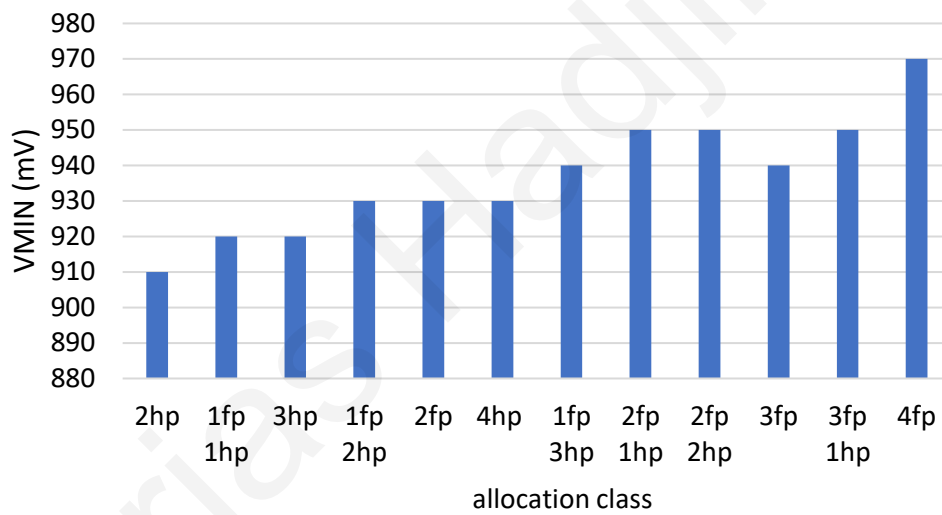


Figure 54. Allocation class V_{MIN} characterization.

In cases of equal V_{MIN} the cores and PMDs are arbitrary selected. For instance, for the 2fp class is not obvious which two PMDs to pick, hence, the worst-case instance is arbitrary selected.

We perform dI/dt virus V_{MIN} characterization for each worst-case topology instance and the results are shown in Figure 54. The classes are sorted based on the number of active cores, and, the results show a rather monotonical trend with increasing number of cores,

which is expected. These V_{MIN} values have a 10mV safety margins added on top of the V_{MIN} i.e. the V_{MIN} values shown on the graph are equal to crash voltage plus 20mV. This safety margin is added to cover any potential inaccuracies in the V_{MIN} results. Such inaccuracies may arise from various experimental limitations such as limited characterization time.

With the allocation class V_{MIN} characterization finished, the DVS governor has all the necessary information for taking DVS decisions. The governor essentially will perform the following steps when CPU is running a workload: a) check which cores are active (i.e. the allocation instance, b) if the V_{MIN} of the allocation-instance is known e.g. in the case a 1hp or 1fp allocation-instance, then we set the voltage to that value otherwise, c) determine to which allocation class the allocation instance belong and set the voltage according to Figure 54 V_{MIN} results.

We test the DVS governor by running the dI/dt virus for all 255 allocation-instances. We have not observed any instability during the experiment execution. Furthermore, we evaluate the DVS governor with SPEC_rate 2017 and NAS benchmarks. We create a random workload by randomly generating: a) the benchmark to execute, and b) the allocation-instance i.e. the cores that will run the benchmark. For SPEC_rate each core runs a different benchmark instance whereas for NAS benchmarks each core executes a different benchmark thread. The total execution time for the random workload is 62 hours. We compare the power consumption of this random workload under: a) the nominal voltage, b) the proposed DVS governor.

The power of the CPU voltage domain over time for the two scenarios is shown in Figure 55. As expected, execution time is not affected because we alter only the CPU voltage. The workload changes are depicted in the figure by the observed power variations. The full random workload consists of a mix of high-power, medium and low-power runs. The average CPU utilization (not shown in the graph) is 50%. The DVS governor consistently

provides lower power than operating 24/7 under nominal voltage. This is illustrated more clearly in Figure 56 , which shows the moving average power consumption.

At the end of the workload, the DVS governor provides 10% power-savings compared to nominal execution. The average voltage during DVS governor execution is 928mV (not shown in graphs). The success of the core-allocation governor supports two conclusions: a) the robustness and accuracy of the EM approach in generating proper dI/dt viruses and b) the worst-case noise is inherent to the design and not to core-to-core variations.

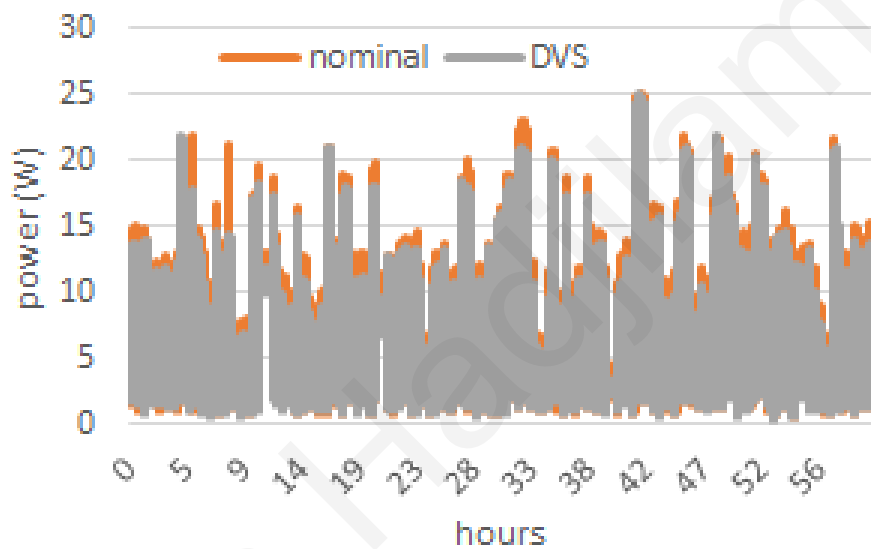


Figure 55. CPU power consumption over 60 hours workload.

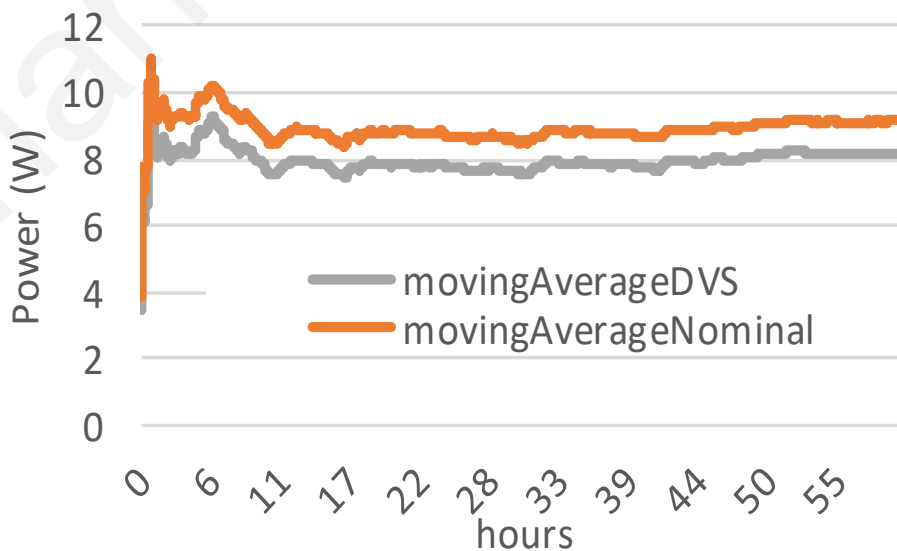


Figure 56. DVS governor vs nominal moving average.

Chapter 8. Conclusions

8.1 Summary

This thesis proposes a novel methodology for post-silicon dI/dt stress-test generation and resonance-frequency detection based on sensing modulations in CPU EM emanations. The proposed approach has the advantage of being non-intrusive: a) to system-software, b) to hardware and c) does not incur design-time overheads and complexities. The basic premise for this methodology is the presence of a correlation between the radiated EM power and on-chip voltage noise. The experimental analysis clearly establishes this correlation. Additionally, we demonstrate the generality of the proposed approach by successfully applying it to different CPUs to generate voltage-noise viruses for them and to obtain their PDN's 1st order resonance frequency.

Our experimental results support that the EM methodology can improve the CPU energy-efficiency by enabling operation outside of the nominal voltage margins. The V_{MIN} of the dI/dt viruses produced with the EM methodology provides a good indication of a system's safe V_{MIN} . We repeatedly verified this claim through V_{MIN} measurements. For all experimental platforms used in this thesis the V_{MIN} of the dI/dt viruses is higher or equal to the V_{MIN} of the conventional workloads. Furthermore, for a 60-hour workload we demonstrate that a DVS governor that scales the voltage according to the dI/dt virus V_{MIN} for various core allocation scenarios can ensure robust execution and provide 10% power-savings compared to nominal execution.

We also demonstrate how the EM setup can be utilized during live operation to detect high voltage droops, react to them using a mitigation mechanism, and ultimately prevent their negative consequences. We evaluate this approach on a simple prototype setup with significant limitations in terms of reaction time. Still, we can guarantee safe and energy-efficient operation for a 12-hour workload composed of conventional benchmarks and dI/dt

viruses. These results strongly encourage further development of this approach towards a product-grade solution that might be integrated into real systems.

Besides the EM methodology, this thesis has another significant contribution. This thesis delivers GeST, a GA based framework for automatic stress-test generation that was developed for the needs of this thesis. While GA based automatic frameworks are not a novel concept, to the best of our knowledge there is no publicly available framework that researchers and practitioners can use. The framework presented in this thesis has been successfully demonstrated in industrial platforms and has been used for various research publications [20][72][73][93][94].

The key strengths of the GeST framework are its flexibility and extensibility as it provides an easy interface to the experimenter that can be used for building upon the framework. We demonstrate the flexibility and the effectiveness of the framework by generating, among other, power and dI/dt stress-tests (viruses) on various CPUs with simple and complex fitness functions. The generated viruses stress the system more than conventional workloads and manually written stress-tests. While this thesis demonstrates GeST on real hardware, there is no fundamental restriction that prevents the framework from being used for pre-silicon stress-test generation in conjunction with accurate power, temperature, performance and voltage-noise models/simulators.

Finally, we conclude this Section with some observations we gather from all the CPUs that we have applied the EM methodology. We discuss cross-platform findings to provide insight on the generated viruses. In particular, we focus the discussion on the measured resonance frequencies, the potential for energy-efficiency improvements, the efficacy of the GA-optimization and the implications of instruction mix in virus generation.

Table VIII provides a comparison about the viruses generated by GA for the different platforms in terms of average instructions per cycle (IPC), instruction loop frequency, dominant frequency (the one where the highest EM amplitude is observed), voltage margin

(the difference between the nominal voltage and virus V_{MIN}), energy-efficiency improvement potential through the elimination of voltage margins and instruction-mix breakdown.

The first-order resonance frequency of processors is typically in the range between 50-200MHz [55][63] which is confirmed by our experimental results. According to Table VIII the lowest resonance is observed at 66MHz (Cortex-A72) and the highest at 150MHz (X-Gene2). Regarding the V_{MIN} of the dI/dt viruses, the viruses exhibit between 20 to 75mV higher V_{MIN} compared to standard benchmarks or previously proposed stress tests (e.g. Prime95) and, hence, can be used to determine better operating points. The Cortex-A72 and Cortex-A53 on the Juno platform can benefit considerably from margin elimination (the estimated V_{MIN} is at least 120mV and 150mV respectively, lower than nominal voltage specifications).

Another interesting insight from Table VIII is that that the dominant frequency (at which highest voltage oscillations occur) does not have to be equal the instruction loop frequency (1/loop period). Cortex-A53 virus has long loop periods that includes faster periodic events that stress the 1st order resonant frequency (e.g. Cortex-A53 has 6 times slower loop frequency than dominant frequency). In contrast, the other viruses have equal dominant and loop frequencies. Is worth mentioning that Cortex-A53 is the only examined CPU that does not benefit from the loop size heuristic. We empirically find that the fittest virus is generated with a loop-size of 50 instructions that does not adhere to the heuristic. Cortex-A53 is also the only in-order CPU in this study. This might suggest that in-order CPUs require slightly different loop-size heuristic, or different parameter values e.g. different target IPC.

Table VIII. dI/dt virus comparison. SL denotes short latency and LL denotes long latency. Voltage margin = Nominal voltage – VirusV_{min}.

CPU	Loop instructions	IPC	Loop period (ns)	Loop Freq (1/loop period) (MHz)	Dominant Freq (MHz)	Voltage margin (mV)	Instruction Type Mix							
							Branch (ARM Only)	SL int Register only	LL int Register only	SL int Mem (x86 only)	LL int Mem (x86 only)	Float	SIMD	MEM (ARM Only)
A72	21	1.2	15	66.66	66.66	120	0%	35%	10%	-	-	25%	25%	5%
A53	50	0.69	81.17	12.32	74.95	150	0%	20%	8%	-	-	42%	24%	6%
X-Gene2	25	1.6	6.66	150	150	20	0%	68%	8%	-	-	4%	4%	16%
X-Gene3	50	2	8	125	125	50	0%	38%	6%	-	-	14%	14%	28%
AMD	50	1.32	13.00	76.92	76.92	37.5	0%	24%	8%	30%	2%	10%	26%	-

Table VIII also shows the instruction breakdown of the viruses. All instruction types, apart from branch instructions are used in the instruction-mix of the viruses. Typically, a virus requires a combination of high-current and low-current-consuming instructions to create modulations in CPU current demand that can match the PDN's 1st order resonance frequency. Single-cycle instructions and those that engage the memory sub-system typically increase current consumption in the pipeline due to higher switching activity. The ARM viruses use plenty of short latency operations whereas the AMD viruses include many short latency integer instructions with operands in memory (denoted as SL-int-Mem). X-Gene CPUs use more memory instructions compared to the rest ARM CPUs. This seems to be attributed to the fact that memory-store instructions on X-Gene CPUs draw more power compared to other CPUs due to the simultaneous engagement of both L1 and L2 cache (recall from Section 6.4 that DL1 on X-Gene CPUs is write-through).

Longer latency instructions are found in all the viruses as they create explicit pipeline stalls/interlocks that reduce current consumption. For stalling the SIMD/floating point functional units we have observed by code inspection that viruses tend to use long latency instructions like FSQRT (square root).

8.2 Future Work Directions

For future work, we aim to extend our methodology to GPU PDNs, complementing recent studies on GPU voltage noise [70][86]. To the best of our knowledge we are not aware of any work that has conducted voltage-droop measurements on real hardware. Most GPU related work uses simulators [86] and models to study voltage-noise. Hence, conducting real-hardware voltage-noise measurements on GPU alone is a prominent research direction. It is worth examining how observations made with real measurements compare against the observations made from studies that are performed with a simulator. Furthermore, applying successfully the EM methodology for dI/dt virus generation on GPUs

will be another significant achievement. In this thesis we already show some promising results by successfully measuring the Mali GPU resonance-frequency in Chapter 4.

Synchronization of dI/dt events for high core counts (8 and above) in general-purpose CPUs is another very interesting future work direction. There is no known method that guarantees thread synchronization in a tangible amount of time, for high core counts in an OS environment. Previously proposed probabilistic approaches for guaranteeing thread-alignment work well for up to 4 threads [42]. Synchronizing dI/dt events at high core counts can lead to security, robustness and reliability issues at nominal settings. Also, we plan to apply the EM methodology on newer Intel and AMD CPUs e.g. Intel Skylake (and above) and AMD Ryzen architectures. Since these CPUs are widely used in the PC and server market segments, conducting voltage-noise research on these CPUs is of great importance.

Another possible research direction is to exploit the interplay of IR drop and inductive dI/dt noise for generating the best voltage-noise viruses. It is known that these two can have additive effect. Fine tuning GeST to achieve both high power and high dI/dt noise is a receipt for creating the ultimate voltage-noise virus for cases where the system does not utilize IR drop compensation. Contrary, if the system employs IR drop compensations e.g. Load-Line-Calibration (such as X-Gene3 Section 6.5) a better receipt seems a dI/dt virus with the lowest possible power-consumption. In general, the GeST framework provides a solid base to build upon and conduct research. In this thesis we have thoroughly tested the framework for CPU based optimizations such as power-viruses and dI/dt viruses. GeST is fairly extensible and can be easily used or extended to generate stress-tests that stress specific CPU components (e.g. viruses that maximize cache-misses or branch mispredictions) or non-CPU parts such as GPUs. Some preliminary results we have on GPU power-viruses look promising. A challenge of generating stress-tests with the EM approach for individual CPU components (e.g. floating-point units) instead of the whole CPU is to filter out sources of EM emanations

that do not belong to the target component. Examples of such sources are fetching and decoding pipeline activity which are necessary steps for executing any CPU instruction.

Investigating the trade-offs between voltage noise and power-gating in the presence of having some cores active and some cores idle in a multi-core CPU is also an interesting work direction. There is an interesting inter-play that we believe has not been comprehensively examined yet. Power-gating disables idle cores, therefore, it reduces overall power consumption by reducing static power. This is desirable, but power gating can also cause undesirable effects. Particularly, as shown in Section 6.2 power gating can amplify voltage-noise with the following means: a) power-gating reduces capacitances, hence, voltage-droop magnitude might be increased, b) power-gating reduces capacitances, therefore, shifts resonance-frequency to a higher value; if the activity of the active cores happens to match the new resonance-frequency then voltage-noise is amplified. Hence, instead of applying power-gating, in some cases it might be more power-efficient to keep the powered cores up and simply lower the operating voltage (this is feasible since voltage-noise will be smaller). There is one more aspect to consider though. In some cases, power-gating can alleviate the voltage droop. If the active cores run a workload that matches the 1st order resonance-frequency, then power-gating can alleviate voltage-noise by moving away the resonance-frequency from the workload activity. A prominent future work is to explore these trade-offs and propose a scheme that decides to power-gate or not idle cores based on the active cores' periodic activity.

Regarding the EM methodology itself, a promising extension is to perform time-domain measurements (instead of frequency-domain as performed for this thesis). With time-domain measurements it might be possible to capture aperiodic-events such as single one-off voltage droops. Currently, the inability to capture one-off events is a disadvantage of the EM methodology compared to the traditional high-bandwidth measurement tools. Moving

the EM methodology to time-domain measurements is a promising future work direction for removing this limitation.

Other general work directions for EM methodology are: a) secure-system design where on-the-fly PDN characterization can be utilized to thwart malicious side-channel attacks, and b) development of an EM based PDN characterization procedure that is integrated in high-end products that can help improve their quality and energy efficiency.

Bibliography

- [1] Alam, M., B. Weir, and A. Silverman. "A future of function or failure?[CMOS gate oxide scaling]." *IEEE circuits and devices magazine* 18, no. 2 (2002): 42-48.
- [2] Alon, Elad, Vladimir Stojanovic, and Mark A. Horowitz. "Circuits and techniques for high-resolution measurement of on-chip power supply noise." *IEEE Journal of Solid-State Circuits* 40, no. 4 (2005): 820-828.
- [3] AMD, <http://support.amd.com/TechDocs/31412.pdf>
- [4] AMD overdrive, <https://www.amd.com/en/technologies/amd-overdrive>
- [5] Ansys, <https://www.ansys.com/en-gb/products/electronics/ansys-hfss>
- [6] Apache, <https://www.apache-da.com/products/sentinel/sentinel-psi>
- [7] ARM,
[http://infocenter.arm.com/help/topic/com.arm.doc.100114_0200_03_en/arm_versatile_express_juno_r2_development_platform_\(v2m_juno_r2\)_technical_reference_manual_100114_0200_03_en.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.100114_0200_03_en/arm_versatile_express_juno_r2_development_platform_(v2m_juno_r2)_technical_reference_manual_100114_0200_03_en.pdf)
- [8] ARM V8 ISA,
http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf
- [9] Bacha, Anys, and Radu Teodorescu. "Using ECC feedback to guide voltage speculation in low-voltage processors." *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014.
- [10] Bertran, Ramon, Alper Buyuktosunoglu, Pradip Bose, Timothy J. Slegel, Gerard Salem, Sean Carey, Richard F. Rizzolo, and Thomas Strach. "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities." In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 368-380. IEEE, 2014.
- [11] big.LITTLE Whitepaper ARM,
https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
big.LITTLE Whitepaper ARM,
https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
- [12] Blender, <https://www.blender.org/>
- [13] Bockelman, David E., and William R. Eisenstadt. "Combined differential and common-mode scattering parameters: Theory and simulation." *IEEE transactions on microwave theory and techniques* 43, no. 7 (1995): 1530-1539.

- [14] Bowman, Keith A., Carlos Tokunaga, Tanay Karnik, Vivek K. De, and James W. Tschanz. "A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance." *IEEE Journal of Solid-State Circuits* 48, no. 4 (2013): 907-916.
- [15] Cadence, https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ic-package-design-analysis/sigrity-systemsi-technology-ds.pdf
- [16] Callan, Robert, Alenka Zajic, and Milos Prvulovic. "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events." *Microarchitecture (MICRO)*, 2014 47th Annual IEEE/ACM International Symposium on. IEEE, 2014.
- [17] Callan, Robert, Alenka Zajic, and Milos Prvulovic. "FASE: finding amplitude-modulated side-channel emanations." *Computer Architecture (ISCA)*, 2015 ACM/IEEE 42nd Annual International Symposium on. IEEE, 2015.
- [18] Callan, Robert, Nina Popovic, Alenka Zajic, and Milos Prvulovic. "A new approach for measuring electromagnetic side-channel energy available to the attacker in modern processor-memory systems." In *Antennas and Propagation (EuCAP)*, 2015 9th European Conference on, pp. 1-5. IEEE, 2015.
- [19] Cinebench, <https://www.maxon.net/en/products/cinebench/>
- [20] Das, Shidhartha, Paul Whatmough, and David Bull. "Modeling and characterization of the system-level Power Delivery Network for a dual-core ARM Cortex-A57 cluster in 28nm CMOS." *Low Power Electronics and Design (ISLPED)*, 2015 IEEE/ACM International Symposium on. IEEE, 2015.
- [21] Das, Shidhartha. "Razor: A Variation-Tolerant Design Methodology for Low-Power and Robust Computing", Doctoral Dissertation, University of Michigan, 2009.
- [22] DeHaven, Keith, and Joel Dietz. "Controlled collapse chip connection (C4)-an enabling technology." In *Electronic Components and Technology Conference, 1994. Proceedings., 44th*, pp. 1-6. IEEE, 1994.
- [23] DS-5 debugger, <https://developer.arm.com/products/software-development-tools/ds-5-development-studio/ds-5-debugger/overview>
- [24] Ernst, Dan, Shidhartha Das, Seokwoo Lee, David Blaauw, Todd Austin, Trevor Mudge, Nam Sung Kim, and Krisztian Flautner. "Razor: circuit-level correction of timing errors for low-power operation." *IEEE Micro* 24, no. 6 (2004): 10-20.
- [25] Euler3d benchmark. www.caselab.okstate.edu/research/euler3dbenchmark.html

- [26] Fischer, Tim, Jayen Desai, Bruce Doyle, Samuel Naffziger, and Ben Patella. "A 90-nm variable frequency clock system for a power-managed itanium architecture processor." *IEEE Journal of Solid-State Circuits* 41, no. 1 (2006): 218-228.
- [27] Ganesan, Karthik, and Lizy K. John. "MAXimum Multicore POver (MAMPO): an automatic multithreaded synthetic power virus generation framework for multicore systems." *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011.
- [28] GeekBench, <https://www.geekbench.com/>
- [29] Genkin, Daniel, Lev Pachmanov, Itamar Pipman, and Eran Tromer. "Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation." In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 207-228. Springer Berlin Heidelberg, 2015.
- [30] Gorman, Daphne I., Matthew R. Guthaus, and Jose Renau. "Architectural opportunities for novel dynamic EMI shifting (DEMIS)." *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017.
- [31] Grenat, Aaron, Sanjay Pant, Ravinder Rachala, and Samuel Naffziger. "5.6 adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor." In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 106-107. IEEE, 2014.
- [32] Gu, J., Eom, H. and Kim, C.H., 2007, June. A switched decoupling capacitor circuit for on-chip supply resonance damping. In *VLSI Circuits, 2007 IEEE Symposium on* (pp. 126-127). IEEE.
- [33] Gupta, Meeta Sharma, Krishna K. Rangan, Michael D. Smith, Gu-Yeon Wei, and David Brooks. "Towards a software approach to mitigate voltage emergencies." In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pp. 123-128. IEEE, 2007.
- [34] Gupta, Meeta S., Krishna K. Rangan, Michael D. Smith, Gu-Yeon Wei, and David Brooks. "DeCoR: A delayed commit and rollback mechanism for handling inductive noise in processors." In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pp. 381-392. IEEE, 2008.
- [35] HSPICE circuit simulation tool, <https://www.synopsys.com/verification/ams-verification/circuit-simulation/hspice.html>
- [36] Intel, <http://download.intel.com/design/mobile/datashts/31407804.pdf>

- [37] James, Norman, Phillip Restle, Joshua Friedrich, Bill Huott, and Bradley McCredie. "Comparison of split-versus connected-core supplies in the POWER6 microprocessor." In Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International, pp. 298-604. IEEE, 2007.
- [38] Jordan, Edward C., and K. G. Balmain. "EM Waves & Radiating Systems." (2006).
- [39] Joseph, Russ, David Brooks, and Margaret Martonosi. "Control techniques to eliminate voltage emergencies in high performance processors." In High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. The Ninth International Symposium on, pp. 79-90. IEEE, 2003.
- [40] Joshi, Ajay M., Lieven Eeckhout, Lizy K. John, and Ciji Isen. "Automated microprocessor stressmark generation." In High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on, pp. 229-239. IEEE, 2008.
- [41] Kim, Youngtaek, and Lizy Kurian John. "Automated di/dt stressmark generation for microprocessor power delivery networks." In Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design, pp. 253-258. IEEE Press, 2011.
- [42] Kim, Youngtaek, Lizy Kurian John, Sanjay Pant, Srilatha Manne, Michael Schulte, William Lloyd Bircher, and Madhu Saravana Sibi Govindan. "AUDIT: Stress testing the automatic way." In Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on, pp. 212-223. IEEE, 2012.
- [43] Kurd, Nasser A., Subramani Bhamidipati, Christopher Mozak, Jeffrey L. Miller, Timothy M. Wilson, Mahadev Nemani, and Muntaquim Chowdhury. "Westmere: A family of 32nm IA processors." In Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International, pp. 96-97. IEEE, 2010.
- [44] Lefurgy, Charles R., Alan J. Drake, Michael S. Floyd, Malcolm S. Allen-Ware, Bishop Brock, Jose A. Tierno, and John B. Carter. "Active management of timing guardband to save energy in POWER7." In Microarchitecture (MICRO), 2011 44th Annual IEEE/ACM International Symposium on, pp. 1-11. IEEE, 2011.
- [45] Leng, Jingwen, Yazhou Zu, and Vijay Janapa Reddi. "GPU voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in GPU architectures." High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. IEEE, 2015.
- [46] Mair, H.T., Gammie, G., Wang, A., Lagerquist, R., Chung, C.J., Gururajarao, S., Kao, P., Rajagopalan, A., Saha, A., Jain, A. and Wang, E., 2016, January. 4.3 A 20nm 2.5 GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC

- performance. In Solid-State Circuits Conference (ISSCC), 2016 IEEE International (pp. 76-77). IEEE.
- [47] Mair, H., Wang, E., Wang, A., Kao, P., Tsai, Y., Gururajaroo, S., Lagerquist, R., Son, J., Gammie, G., Lin, G. and Thippana, A., 2017, February. 3.4 A 10nm FinFET 2.8 GHz tri-gear deca-core CPU complex with optimized power-delivery network for mobile SoC performance. In Solid-State Circuits Conference (ISSCC), 2017 IEEE International (pp. 56-57). IEEE.
- [48] Mansuri, M., Casper, B. and O'Mahony, F., 2012, June. An on-die all-digital delay measurement circuit with 250fs accuracy. In VLSI Circuits (VLSIC), 2012 Symposium on (pp. 98-99). IEEE.
- [49] Miller, Timothy N., Renji Thomas, Xiang Pan, and Radu Teodorescu. "VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors." In Computer Architecture (ISCA), 2012 39th Annual International Symposium on, pp. 249-260. IEEE, 2012.
- [50] Mitchell, Melanie. An introduction to genetic algorithms. MIT press, 1998.
- [51] National Instruments drivers, <http://www.ni.com/downloads/drivers/>
- [52] Nazari, Alireza, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. "EDDIE: EM-Based Detection of Deviations in Program Execution." In Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 333-346. ACM, 2017..
- [53] O'Mahony, F., 2013 February. Tutorial 6 - On-chip voltage and timing-diagnostic circuits. In Solid-State Circuits Conference (ISSCC), 2013 IEEE International. IEEE.
- [54] Oracle x86 assembly language reference manual, https://docs.oracle.com/cd/E18752_01/html/817-5477/docinfo.html
- [55] Pant, Sanjay. "Design and Analysis of Power Distribution Networks in VLSI Circuits." (2008).
- [56] Papadimitriou, George, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Peter Lawthers, and Shidhartha Das. "Harnessing voltage margins for energy efficiency in multicore CPUs." In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 503-516. ACM, 2017.
- [57] Polfliet, Stijn, Frederick Ryckbosch, and Lieven Eeckhout. "Automated full-system power characterization." IEEE Micro 31.3 (2011): 46-59.
- [58] Powell, M. D., & Vijaykumar, T. N. (2004, June). Exploiting resonant behavior to reduce inductive noise. In Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on (pp. 288-299). IEEE.

- [59] Powell, Michael D., and T. N. Vijaykumar. "Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage." In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp. 72-83. IEEE, 2003.
- [60] Powell, Michael D., and T. N. Vijaykumar. "Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise." In *Proceedings of the 2003 international symposium on Low power electronics and design*, pp. 223-228. ACM, 2003.
- [61] Prime 95, <https://www.mersenne.org/download/>
- [62] Ravezzi, Luca, and Hamid Partovi. "Clock and synchronization networks for a 3 GHz 64 Bit ARMv8 8-core SoC." *IEEE Journal of Solid-State Circuits* 50.7 (2015): 1702-1710.
- [63] Reddi, Vijay Janapa, and Meeta Sharma Gupta. "Resilient architecture design for voltage variation." *Synthesis Lectures on Computer Architecture* 8.2 (2013): 1-138.
- [64] Reddi, Vijay Janapa, Meeta S. Gupta, Glenn Holloway, Gu-Yeon Wei, Michael D. Smith, and David Brooks. "Voltage emergency prediction: Using signatures to reduce operating margins." In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 18-29. IEEE, 2009.
- [65] Reddi, Vijay Janapa, Meeta S. Gupta, Krishna K. Rangan, Simone Campanoni, Glenn Holloway, Michael D. Smith, Gu-Yeon Wei, and David Brooks. "Voltage noise: Why it's bad, and what to do about it." In *5th IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE)*, Palo Alto, CA. 2009.
- [66] Reddi, Vijay Janapa, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D. Smith, Gu-Yeon Wei, and David Brooks. "Voltage noise in production processors." *IEEE micro* 31, no. 1 (2011): 20-28.
- [67] Sathé, V. and Das, S. Taming the Dark Horse: Voltage-Margin Minimization for Modern "Real-World" Energy-Efficient Computing. Tutorial in *IEEE Design Automation Conference (DAC)*, Austin, TX, June 2016.
- [68] Sehatbakhsh, Nader, Alireza Nazari, Alenka Zajic, and Milos Prvulovic. "Spectral profiling: Observer-effect-free profiling by monitoring EM emanations." In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1-11. IEEE, 2016.
- [69] Stutzman, Warren L., and Gary A. Thiele. *Antenna theory and design*. John Wiley & Sons, 2012.
- [70] Thomas, Renji, Naser Sedaghati, and Radu Teodorescu. "EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures." *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 2016.

- [71] Webxprt, <http://www.principledtechnologies.com/benchmarkxprt/webxprt>
- [72] Whatmough, Paul N., Shidhartha Das, Zacharias Hadjilambrou, and David M. Bull. "14.6 An all-digital power-delivery monitor for analysis of a 28nm dual-core ARM Cortex-A57 cluster." In Solid-State Circuits Conference-(ISSCC), 2015 IEEE International, pp. 1-3. IEEE, 2015.
- [73] Whatmough, Paul N., Shidhartha Das, Zacharias Hadjilambrou, and David M. Bull. "Power Integrity Analysis of a 28 nm Dual-Core ARM Cortex-A57 Cluster Using an All-Digital Power Delivery Monitor." *IEEE Journal of Solid-State Circuits* 52, no. 6 (2017): 1643-1654.
- [74] Xu, J., Hazucha, P., Huang, M., Aseron, P., Paillet, F., Schrom, G., Tschanz, J., Zhao, C., De, V., Karnik, T. and Taylor, G., 2007, February. On-die supply-resonance suppression using band-limited active damping. In Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International (pp. 286-603). IEEE.
- [75] Zu, Yazhou, Charles R. Lefurgy, Jingwen Leng, Matthew Halpern, Michael S. Floyd, and Vijay Janapa Reddi. "Adaptive guardband scheduling to improve system-level efficiency of the POWER7+." In Proceedings of the 48th International Symposium on Microarchitecture, pp. 308-321. ACM, 2015.
- [76] Stern, Andrew, Ulbert Botero, Bicky Shakya, Haoting Shen, Domenic Forte, and Mark Tehranipoor. "EMFORCED: EM-based Fingerprinting Framework for Counterfeit Detection with Demonstration on Remarked and Cloned ICs." In 2018 IEEE International Test Conference (ITC), pp. 1-9. IEEE, 2018.
- [77] He, Kai, Xin Huang, and Sheldon X-D. Tan. "EM-based on-chip aging sensor for detection and prevention of counterfeit and recycled ICs." In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 146-151. IEEE Press, 2015.
- [78] GeST a framework for generating stress-tests, <https://github.com/toolsForUarch/GeST>
- [79] Hadjilambrou, Zacharias, Shidhartha Das, Paul N. Whatmough, David Bull, and Yiannakis Sazeides. "GeST: An Automatic Framework For Generating CPU Stress-Tests." In 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 1-10. IEEE, 2019.
- [80] AARONIA, "Datasheet: Rf near field probe set dc to 9ghz." <http://www.aaronia.com/Datasheets/Antennas/RF-Near-Field-Probe-Set.pdf>, accessed April 6, 2016
- [81] Tullsen, Dean M., Susan J. Eggers, and Henry M. Levy. "Simultaneous multithreading: Maximizing on-chip parallelism." *ACM SIGARCH computer architecture news*. Vol. 23. No. 2. ACM, 1995.

- [82] Singh, Gaurav, Greg Favor, and Alfred Yeung. "Appliedmicro x-gene2." In 2014 IEEE Hot Chips 26 Symposium (HCS), pp. 1-24. IEEE, 2014.
- [83] Rogers, Aaron, et al. "The Core-C6 (CC6) sleep state of the AMD Bobcat x86 microprocessor." Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design. ACM, 2012.
- [84] Kaxiras, Stefanos, and Margaret Martonosi. "Computer architecture techniques for power-efficiency." Synthesis Lectures on Computer Architecture 3.1 (2008): 1-207.
- [85] Reddi, Vijay Janapa, Simone Campanoni, Meeta S. Gupta, Michael D. Smith, Gu-Yeon Wei, David Brooks, and Kim Hazelwood. "Eliminating voltage emergencies via software-guided code transformations." ACM Transactions on Architecture and Code Optimization (TACO) 7, no. 2 (2010): 12.
- [86] Leng, Jingwen, et al. "GPUVolt: Modeling and characterizing voltage noise in GPU architectures." Low Power Electronics and Design (ISLPED), 2014 IEEE/ACM International Symposium on. IEEE, 2014.
- [87] Thomas, Renji, et al. "Core Tunneling: Variation-aware voltage noise mitigation in GPUs." High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. IEEE, 2016.
- [88] Leng, Jingwen, Alper Buyuktosunoglu, Ramon Bertran, Pradip Bose, and Vijay Janapa Reddi. "Safe limits on voltage reduction efficiency in GPUs: a direct measurement approach." In Proceedings of the 48th International Symposium on Microarchitecture, pp. 294-307. ACM, 2015.
- [89] Ganesan, Karthik, Jungho Jo, W. Lloyd Bircher, Dimitris Kaseridis, Zhibin Yu, and Lizy K. John. "System-level Max Power (SYMPO)-A systematic approach for escalating system-level power consumption using synthetic benchmarks." In 2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 19-28. IEEE, 2010.
- [90] Bertran, Ramon, Alper Buyuktosunoglu, Meeta S. Gupta, Marc Gonzalez, and Pradip Bose. "Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks." In 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 199-211. IEEE, 2012.
- [91] AIDA64 system stability test, <https://www.aida64.com/>
- [92] Papadimitriou, G., Chatzidimitriou, A., Kaliorakis, M., Vastakis, Y., & Gizopoulos, D. (2018, April). Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs. In Performance Analysis of Systems and Software (ISPASS), 2018 IEEE International Symposium on (pp. 54-63). IEEE.

- [93] Hadjilambrou, Zacharias, Shidhartha Das, Marco A. Antoniadis, and Yiannakis Sazeides. "Leveraging CPU Electromagnetic Emanations for Voltage Noise Characterization." In 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 573-585. IEEE, 2018.
- [94] Konstantinos Tovletoglou, Lev Mukhanov, Georgios Karakonstantis, Athanasios Chatzidimitriou, George Papadimitriou, Manolis Kaliorakis, Dimitris Gizopoulos, Zacharias Hadjilambrou, Yiannakis Sazeides, Alejandro Lampropulos, Shidhartha Das, Phong Vo: Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs. DSN Workshops 2018: 6-9
- [95] De Jong, Kenneth Alan. "Analysis of the behavior of a class of genetic adaptive systems." (1975).
- [96] Goldberg, David E., and Kalyanmoy Deb. "A comparative analysis of selection schemes used in genetic algorithms." In Foundations of genetic algorithms, vol. 1, pp. 69-93. Elsevier, 1991.