



**NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS**

**SCHOOL OF SCIENCES**

**DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS**

**PROGRAM OF POSTGRADUATE STUDIES**

**PhD THESIS**

**Methods for Robust and Energy-Efficient Microprocessor  
Architectures**

**George A. Papadimitriou**

**ATHENS**

**JUNE 2019**





**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ**

**ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ**

**Μέθοδοι για Εύρωστες και Ενεργειακά Αποδοτικές  
Αρχιτεκτονικές Μικροεπεξεργαστών**

**Γεώργιος Α. Παπαδημητρίου**

**ΑΘΗΝΑ**

**ΙΟΥΝΙΟΣ 2019**



# **PhD THESIS**

Methods for Robust and Energy-Efficient Microprocessor Architectures

**George A. Papadimitriou**

**ADVISOR: Dimitris Gizopoulos, Professor UoA**

## **THREE-MEMBER ADVISORY COMMITTEE:**

**Dimitris Gizopoulos, Professor UoA**

**Antonis Paschalis, Professor UoA**

**Yannis Smaragdakis, Professor UoA**

## **SEVEN-MEMBER EXAMINATION COMMITTEE**

(Signature)

(Signature)

**Dimitris Gizopoulos,  
Professor UoA**

**Antonis Paschalis,  
Professor UoA**

(Signature)

(Signature)

**Yannis Smaragdakis,  
Professor UoA**

**Stathes Hadjiefthymiades,  
Professor UoA**

(Signature)

(Signature)

**Dionisios Pnevmatikatos,  
Professor TUC**

**Dimitrios Soudris,  
Professor NTUA**

(Signature)

**Mihalis Psarakis,  
Assistant Professor UniPi**

**Examination Date 20/06/2019**



## **ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ**

Μέθοδοι για Εύρωστες και Ενεργειακά Αποδοτικές Αρχιτεκτονικές Μικροεπεξεργαστών

**Γεώργιος Α. Παπαδημητρίου**

**ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:** Δημήτρης Γκιζόπουλος, Καθηγητής ΕΚΠΑ

### **ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ:**

**Δημήτρης Γκιζόπουλος, Καθηγητής ΕΚΠΑ**

**Αντώνης Πασχάλης, Καθηγητής ΕΚΠΑ**

**Γιάννης Σμαραγδάκης, Καθηγητής ΕΚΠΑ**

### **ΕΠΤΑΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ**

(Υπογραφή)

**Δημήτρης Γκιζόπουλος,  
Καθηγητής ΕΚΠΑ**

(Υπογραφή)

**Γιάννης Σμαραγδάκης,  
Καθηγητής ΕΚΠΑ**

(Υπογραφή)

**Διονύσιος Πνευματικάτος,  
Καθηγητής Πολυτεχνείο Κρήτης**

(Υπογραφή)

**Μιχάλης Ψαράκης,  
Επίκουρος Καθηγητής Πα.Πει**

(Υπογραφή)

**Αντώνης Πασχάλης,  
Καθηγητής ΕΚΠΑ**

(Υπογραφή)

**Στάθης Χατζηευθυμιάδης,  
Καθηγητής ΕΚΠΑ**

(Υπογραφή)

**Δημήτριος Σούντρης,  
Καθηγητής ΕΜΠ**

**Ημερομηνία εξέτασης 20/06/2019**





## ABSTRACT

Technology scaling has enabled improvements in the three major design optimization objectives: increased performance, lower power consumption, and lower die cost, while system design has focused on bringing more functionality into products at lower cost. While today's microprocessors, are much faster and much more versatile than their predecessors, they also consume much power. As operating frequency and integration density increase, the total chip power dissipation increases. This is evident from the fact that due to the demand for increased functionality on a single chip, more and more transistors are being packed on a single die and hence, the switching frequency increases in every technology generation. However, by developing aggressive and sophisticated mechanisms to boost performance and to enhance the energy efficiency in conjunction with the decrease of the size of transistors, microprocessors have become extremely complex systems, making the microprocessor verification and manufacturing testing a major challenge for the semiconductor industry. Manufacturers, therefore, choose to spend extra effort, time, budget and chip area to ensure that the delivered products are operating correctly. To meet high-dependability requirements, manufacturers apply a sequence of verification tasks throughout the entire life-cycle of the microprocessor to ensure the correct functionality of the microprocessor chips from the various types of errors that may occur after the products are released to the market.

To this end, this work presents novel methods for ensuring the correctness of the microprocessor during the post-silicon validation phase and for improving the energy efficiency requirements of modern microprocessors. These methods can be applied during the prototyping phase of the microprocessors or after their release to the market. More specifically, in the first part of the thesis, we present and describe two different ISA-independent software-based post-silicon validation methods, which contribute to formalization and modeling as well as the acceleration of the post-silicon validation process and expose difficult-to-find bugs in the address translation mechanisms (ATM) of modern microprocessors. Both methods improve the detection and diagnosis of a hardware design bug in the ATM structures and significantly accelerate the bug detection during the post-silicon validation phase. In the second part of the thesis we present a detailed system-level voltage scaling characterization study for two state-of-the-art ARMv8-based multicore CPUs. We present an extensive characterization study which identifies the pessimistic voltage guardbands (the increased voltage margins set by the manufacturer) of each individual microprocessor core and analyze any abnormal behavior that may occur in off-nominal voltage conditions. Towards the formalization of the any abnormal behavior we also present a simple consolidated function; the *Severity function*, which aggregates the effects of reduced voltage operation. We then introduce the development of dedicated programs (diagnostic micro-viruses) that aim to accelerate the time-consuming voltage margins characterization studies by stressing the fundamental hardware components. Finally, we present a comprehensive exploration of how two server-grade systems behave in different frequency and core allocation configurations beyond nominal voltage operation in multicore executions. This analysis aims (1) to identify the best performance per watt operation points, (2) to reveal how and why the different core allocation options affect the energy consumption, and (3) to enhance the default Linux scheduler to take task allocation decisions for balanced performance and energy efficiency.

**SUBJECT AREA:** Computer Architecture

**KEYWORDS:** Dependability, Correctness, Post-Silicon Validation, Design Bugs, Address Translation, Energy Efficiency, Voltage Margins



## ΠΕΡΙΛΗΨΗ

Σήμερα, η εξέλιξη της τεχνολογίας επιτρέπει τη βελτίωση τριών βασικών στοιχείων της σχεδίασης των επεξεργαστών: αυξημένες επιδόσεις, χαμηλότερη κατανάλωση ισχύος και χαμηλότερο κόστος παραγωγής του τσιπ, ενώ οι σχεδιαστές επεξεργαστών έχουν επικεντρωθεί στην παραγωγή επεξεργαστών με περισσότερες λειτουργίες σε χαμηλότερο κόστος. Οι σημερινοί επεξεργαστές είναι πολύ ταχύτεροι και διαθέτουν εξελιγμένες λειτουργικές μονάδες συγκριτικά με τους προκατόχους τους, ωστόσο, καταναλώνουν αρκετά μεγάλη ενέργεια. Τα ποσά ηλεκτρικής ισχύος που καταναλώνονται, και η επακόλουθη έκλυση θερμότητας, αυξάνονται παρά τη μείωση του μεγέθους των τρανζίστορ. Αναπτύσσοντας όλο και πιο εξελιγμένους μηχανισμούς και λειτουργικές μονάδες για την αύξηση της απόδοσης και βελτίωση της ενέργειας, σε συνδυασμό με τη μείωση του μεγέθους των τρανζίστορ, οι επεξεργαστές έχουν γίνει εξαιρετικά πολύπλοκα συστήματα, καθιστώντας τη διαδικασία της επικύρωσής τους σημαντική πρόκληση για τη βιομηχανία ολοκληρωμένων κυκλωμάτων. Συνεπώς, οι κατασκευαστές επεξεργαστών αφιερώνουν επιπλέον χρόνο, προϋπολογισμό και χώρο στο τσιπ για να διασφαλίσουν ότι οι επεξεργαστές θα λειτουργούν σωστά κατά τη διάθεσή τους στη αγορά.

Για τους λόγους αυτούς, η εργασία αυτή παρουσιάζει νέες μεθόδους για την επιτάχυνση και τη βελτίωση της φάσης της επικύρωσης, καθώς και για τη βελτίωση της ενεργειακής απόδοσης των σύγχρονων επεξεργαστών. Στο πρώτο μέρος της διατριβής προτείνονται δύο διαφορετικές μέθοδοι για την επικύρωση του επεξεργαστή, οι οποίες συμβάλλουν στην επιτάχυνση αυτής της διαδικασίας και στην αποκάλυψη σπάνιων σφαλμάτων στους μηχανισμούς μετάφρασης διευθύνσεων των σύγχρονων επεξεργαστών. Και οι δύο μέθοδοι καθιστούν ευκολότερη την ανίχνευση και τη διάγνωση σφαλμάτων, και επιταχύνουν την ανίχνευση του σφάλματος κατά τη φάση της επικύρωσης. Στο δεύτερο μέρος της διατριβής παρουσιάζεται μια λεπτομερής μελέτη χαρακτηρισμού των περιθωρίων τάσης σε επίπεδο συστήματος σε δύο σύγχρονους ARMv8 επεξεργαστές. Η μελέτη του χαρακτηρισμού προσδιορίζει τα αυξημένα περιθώρια τάσης που έχουν προκαθοριστεί κατά τη διάρκεια κατασκευής του κάθε μεμονωμένου πυρήνα του επεξεργαστή και αναλύει τυχόν απρόβλεπτες συμπεριφορές που μπορεί να προκύψουν σε συνθήκες μειωμένης τάσης. Για την μελέτη και καταγραφή της συμπεριφοράς του συστήματος υπό συνθήκες μειωμένης τάσης, παρουσιάζεται επίσης σε αυτή τη διατριβή μια απλή και ενοποιημένη συνάρτηση: η συνάρτηση πυκνότητας-σοβαρότητας. Στη συνέχεια, παρουσιάζεται αναλυτικά η ανάπτυξη ειδικά σχεδιασμένων προγραμμάτων (micro-viruses) τα οποία υποβάλλουν της θεμελιώδεις δομές του επεξεργαστή σε μεγάλο φορτίο εργασίας. Αυτά τα προγράμματα στοχεύουν στην γρήγορη αναγνώριση των ασφαλών περιθωρίων τάσης. Τέλος, πραγματοποιείται ο χαρακτηρισμός των περιθωρίων τάσης σε εκτελέσεις πολλαπλών πυρήνων, καθώς επίσης και σε διαφορετικές συχνότητες, και προτείνεται ένα πρόγραμμα το οποίο εκμεταλλεύεται όλες τις διαφορετικές πτυχές του προβλήματος της κατανάλωσης ενέργειας και παρέχει μεγάλη εξοικονόμηση ενέργειας διατηρώντας παράλληλα υψηλά επίπεδα απόδοσης. Αυτή η μελέτη έχει ως στόχο τον εντοπισμό και την ανάλυση της σχέσης μεταξύ ενέργειας και απόδοσης σε διαφορετικούς συνδυασμούς τάσης και συχνότητας, καθώς και σε διαφορετικό αριθμό νημάτων/διεργασιών που εκτελούνται στο σύστημα, αλλά και κατανομή των προγραμμάτων στους διαθέσιμους πυρήνες.

**ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ:** Αρχιτεκτονική Υπολογιστών

**ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ:** Φερεγγυότητα, Ορθότητα, Επικύρωση Σχεδίασης Επεξεργαστών, Σφάλματα Σχεδίασης, Μετάφραση Διευθύνσεων, Ενεργειακή Απόδοση, Περιθώρια Τάσης



στην Αυγή



## **ACKNOWLEDGMENTS**

I would like to express my sincere gratitude to my advisor, Prof. Dimitris Gizopoulos, who has consistently inspired me and provided me with precious suggestions, invaluable advice and most of all enthusiasm. His guidance and encouragement were vital for this thesis to be completed and also for my future research career. I am also infinitely grateful to him for all the patience and tenacity with which he helped me improve my writing and presentation skills. Besides, I have also acquired valuable insights through his teaching, guidance, and collaboration with him during my academic studies and research. I feel truly fortunate to have him as my academic advisor, teacher, and mentor.

I would also like to thank my collaborators and friends from the Computer Architecture Lab, where I am a member since my master thesis, and especially to Sakis Chatzidimitriou for our philosophical and technical debates, exchanges of knowledge and skills, advice, and the excellent cooperation we had during our PhD theses. I also want to thank Ronny Morad, Vitali Sokhin, Anatoly Koyfman, and Tom Kolan from IBM Research in Haifa for the useful discussions during the first years of my PhD thesis, and especially Peter Lawthers from Applied Micro (APM), who has provided me with valuable advice and guidance thanks to his notable professional experience and extensive technical knowledge, which helped enrich my experience. In addition, I would like to thank my dear friend Antonis Tsigkanos for his continuous support and company during the tough times in the pursuit of academic goals.

Last but certainly not least, I would like to thank my family for the support they provided me through my entire life. In particular, appreciation also goes to Avgi for her constant mental support, which always kept me going through the last years of this journey. Completion of this work would not be possible without her love and encouragement.





## LIST OF PUBLICATIONS

### Publications of the PhD Thesis

- [1] **G. Papadimitriou**, A. Chatzidimitriou, and D. Gizopoulos, "Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs", IEEE International Symposium on High-Performance Computer Architecture (**HPCA 2019**), Washington D.C., USA, pp. 133-146, February 16-20, 2019.
- [2] L. Mukhanov, K. Tovletoglou, G. Karakonstantis, **G. Papadimitriou**, A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, and S. Das, Chapter 13: "Improving the Energy Efficiency by Exceeding the Conservative Operating Limits", In book: Hardware Accelerators in Data Centers, Editors: C. Kachris, B. Falsafi, D. Soudris, Springer International Publishing, 2019, DOI: 10.1007/978-3-319-92792-3\_13. (*Book Chapter*)
- [3] M. Kaliorakis, A. Chatzidimitriou, **G. Papadimitriou**, and D. Gizopoulos "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions", IEEE Computer Architecture Letters (**IEEE CAL**), Volume: 17, Issue: 2, pp. 109-112, July 2018.
- [4] K. Tovletoglou, L. Mukhanov, G. Karakonstantis, A. Chatzidimitriou, **G. Papadimitriou**, M. Kaliorakis, D. Gizopoulos, Z. Hadjilambrou, Y. Sazeides, A. Lampropoulos, S. Das, and P. Vo, "Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs", IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (**DSN-W 2018**), Luxembourg, pp. 6-9, June 25-28, 2018.
- [5] **G. Papadimitriou**, A. Chatzidimitriou, M. Kaliorakis, Y. Vastakis, and D. Gizopoulos, "Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs", IEEE International Symposium on Performance Analysis of Systems and Software (**ISPASS 2018**), Belfast, Northern Ireland, United Kingdom, pp. 54-63, April 2-4, 2018.
- [6] G. Karakonstantis, K. Tovletoglou, L. Mukhanov, H. Vandierendonck, D. S. Nikolopoulos, P. Lawthers, P. Koutsovasilis, M. Maroudas, C. D. Antonopoulos, C. Kalogirou, N. Bellas, S. Lalis, S. Venugopal, A. Prat-Perez, A. Lampropoulos, M. Kleanthous, A. Diavastos, Z. Hadjilambrou, P. Nikolaou, Y. Sazeides, P. Trancoso, **G. Papadimitriou**, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, and S. Das, "An Energy-Efficient and Error-Resilient Server Ecosystem Exceeding Conservative Scaling Limits", ACM/IEEE Design, Automation, and Test in Europe (**DATE 2018**), Dresden, Germany, pp. 1099-1104, March 19-23, 2018.
- [7] **G. Papadimitriou**, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs", IEEE/ACM International Symposium on Microarchitecture (**MICRO 2017**), Cambridge, MA, USA, pp. 503-516, October 14-18, 2017.
- [8] **G. Papadimitriou**, M. Kaliorakis, A. Chatzidimitriou, C. Magdalinos, and D. Gizopoulos, "Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors", IEEE International Symposium on On-Line Testing and Robust System Design (**IOLTS 2017**), Thessaloniki, Greece, pp. 51-56, July 3-5, 2017.

- [9] **G. Papadimitriou**, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, G. Favor, K. Sankaran and S. Das, "A System-Level Voltage/Frequency Scaling Characterization Framework for Multicore CPUs", IEEE Silicon Errors in Logic – System Effects (**SELSE 2017**), Boston, MA, USA, March 21-22, 2017. (*Workshop Paper*)
- [10] **G. Papadimitriou**, D. Gizopoulos, A. Chatzidimitriou, and R. Morad "An Agile Post-Silicon Validation Methodology for the Address Translation Mechanisms of Modern Microprocessors", IEEE Transactions on Device and Materials Reliability (**IEEE TDMR**), Volume: 17, Issue: 1, pp. 3-11, March 2017.
- [11] **G. Papadimitriou**, D. Gizopoulos, A. Chatzidimitriou, T. Kolan, A. Koyfman, R. Morad and V. Sokhin, "Unveiling Difficult Bugs in Address Translation Caching Arrays for Effective Post-Silicon Validation", IEEE International Conference on Computer Design (**ICCD 2016**), Phoenix, AZ, USA, pp. 544-551, October 2-5, 2016.
- [12] **G. Papadimitriou**, A. Chatzidimitriou, D. Gizopoulos and R. Morad, "ISA-Independent Post-Silicon Validation for the Address Translation Mechanisms of Modern Microprocessors", IEEE International Symposium on On-Line Testing and Robust System Design (**IOLTS 2016**), Sant Feliu de Guixols, Spain, pp. 72-77, July 4-6, 2016.
- [13] **G. Papadimitriou**, A. Chatzidimitriou, D. Gizopoulos and R. Morad, "Post-Silicon Validation Methodology for the Address Translation Mechanisms of Modern Microprocessors", ACM/EDAC/IEEE Design Automation Conference (**DAC 2016**), Austin, TX, USA, June 5-9, 2016. (*Poster*)

## Other Publications during PhD Thesis

- [14] A. Chatzidimitriou, P. Bodmann, **G. Papadimitriou**, D. Gizopoulos, and P. Rech, "Demystifying Soft Error Assessment Strategies on ARM CPUs: Microarchitectural Fault Injection vs. Neutron Beam Experiments", IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN 2019**), Portland, Oregon, USA, June 24-27, 2019. (*Best Paper Award Candidate*)
- [15] P. Nikolaou, Y. Sazeides, A. Lampropoulos, D. Guilhot, A. Bartoli, **G. Papadimitriou**, A. Chatzidimitriou, D. Gizopoulos, K. Tovletoglou, L. Mukhanov, and G. Karakonstantis, "On the Evaluation of the Total-Cost-of-Ownership Trade-offs in Edge vs Cloud deployments: A Wireless-Denial-of-Service Case Study", IEEE Transactions on Sustainable Computing (**IEEE T-SUSC**), 2019.
- [16] A. Chatzidimitriou, **G. Papadimitriou**, and D. Gizopoulos, "HealthLog Monitor: Errors, Symptoms and Reactions Consolidated", IEEE Transactions on Device and Materials Reliability (**IEEE TDMR**), Volume: 19, Issue: 1, pp. 46-54, March 2019.
- [17] A. Chatzidimitriou, **G. Papadimitriou**, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Assessing the Effects of Low Voltage in Branch Prediction Units", IEEE International Symposium on Performance Analysis of Systems and Software (**ISPASS 2019**), Madison, Wisconsin, USA, pp. 127-136, March 24-26, 2019.
- [18] A. Chatzidimitriou, **G. Papadimitriou**, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Analysis and Characterization of Ultra Low Power Branch Predictors", IEEE International Conference on Computer Design (**ICCD 2018**), Orlando, Florida, USA, pp. 144-147, October 7-10, 2018.

- [19] A. Chatzidimitriou, **G. Papadimitriou**, and D. Gizopoulos, "HealthLog Monitor: A Flexible System-Monitoring Linux Service", IEEE International Symposium on On-Line Testing and Robust System Design (**IOLTS 2018**), Costa Brava, Spain, pp. 183-188, July 2-4, 2018.
- [20] K. Tovletoglou, C. Chalias, G. Karakonstantis, L. Mukhanov, H. Vandierendonck, D. S. Nikolopoulos, P. Koutsovasilis, M. Maroudas, C. Antonopoulos, C. Kalogirou, N. Bellas, S. Lalis, M. Rafique, S. Venugopal, A. Prat-Perez, A. Diavastos, Z. Hadjilambrou, P. Nikolaou, Y. Sazeides, P. Trancoso, **G. Papadimitriou**, M. Kaliorakis, A. Chatzidimitriou and D. Gizopoulos, "An Energy-Efficient and Error-Resilient Server Ecosystem Exceeding Conservative Scaling Limits", Energy-Efficient Servers for Cloud and Edge Computing (**ENeSCE 2017**), Stockholm, Sweden, January 23, 2017.
- [21] S. Tselonis, M. Kaliorakis, N. Foutris, **G. Papadimitriou**, and D. Gizopoulos, "Microprocessor Reliability-Performance Tradeoffs Assessment at the Microarchitecture Level", IEEE VLSI Test Symposium (**VTS 2016**), Las Vegas, NV, USA, April 25-27, 2016.



## ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

Μέχρι και την προηγούμενη δεκαετία, η σχεδίαση των επεξεργαστών βασιζόταν στην κατασκευή ισχυρότερων και περισσότερων (σε πλήθος) πυρήνων με υψηλότερη συχνότητα και υψηλότερη κατανάλωση ισχύος, χρησιμοποιώντας μικρότερα και ταχύτερα τρανζίστορ. Σήμερα, η εξέλιξη της τεχνολογίας επιτρέπει τη βελτίωση τριών βασικών στοιχείων της σχεδίασης των επεξεργαστών: αυξημένες επιδόσεις, χαμηλότερη κατανάλωση ισχύος και χαμηλότερο κόστος παραγωγής του τσιπ, ενώ οι σχεδιαστές επεξεργαστών επικεντρώθηκαν στην παροχή επεξεργαστών με περισσότερες λειτουργίες σε χαμηλότερο κόστος. Ενώ οι σημερινοί επεξεργαστές είναι πολύ ταχύτεροι από τους προκατόχους τους, συνεχίζουν να καταναλώνουν μεγάλη ενέργεια. Τα ποσά ηλεκτρικής ισχύος που καταναλώνονται, και η επακόλουθη έκλυση θερμότητας, αυξάνονται παρά τη μείωση του μεγέθους των τρανζίστορ [29]. Τέτοια ποσοστά ενέργειας μειώνουν την αξιοπιστία των ολοκληρωμένων κυκλωμάτων και το προσδόκιμο ζωής τους, αυξάνουν το κόστος ψύξης, και μάλιστα, δημιουργούν περιβαλλοντικά προβλήματα που προέρχονται κυρίως από τα μεγάλα κέντρα δεδομένων (datacenters). Τα μεγάλα ποσοστά κατανάλωσης ισχύος δημιουργούν επίσης την ανάγκη συχνότερης φόρτισης στις φορητές συσκευές οι οποίες διαθέτουν μπαταρίες περιορισμένης χωρητικότητας. Με το πέρασμα του χρόνου, οι βελτιώσεις στην τεχνολογία των επεξεργαστών θα αρχίσουν να φθίνουν (π.χ., ο λόγος απόδοσης ανά μονάδα ισχύος θα κλιμακώνεται πιο αργά) χωρίς να παρέχουν τελικά οικονομικά αποδοτικές λύσεις στο πρόβλημα της ηλεκτρικής ενέργειας.

Οι σημερινοί επεξεργαστές ενσωματώνουν διάφορους τύπους πυρήνων και λειτουργικών μονάδων (π.χ., κρυφές μνήμες, μονάδες πρόβλεψης διακλαδώσεων, κλπ.), και στοχεύουν στη δυναμική βελτιστοποίηση της απόδοσης αλλά και της ενέργειας. Για να είναι ισχυροί και αποδοτικοί οι σημερινοί επεξεργαστές, κάνουν χρήση εξελιγμένων μηχανισμών για την αύξηση της απόδοσης και της μείωσης της κατανάλωσης ισχύος. Αυτή η πίεση για αύξηση της απόδοσης συνοδεύεται φυσικά και από αύξηση στην ευπάθεια (ή ισοδύναμα μείωση στην αξιοπιστία) των επεξεργαστών καθώς η ποιότητα των ολοκληρωμένων κυκλωμάτων περιορίζεται εξαιτίας: (α) των αυστηρών χρονοδιαγραμμάτων τα οποία ορίζονται για να μειωθεί ο χρόνος που απαιτείται μέχρι το προϊόν να κυκλοφορήσει στην αγορά (άρα και ο χρόνος για τον έλεγχο της αξιοπιστίας του), και (β) των σύγχρονων τεχνικών κατασκευής και της αυξημένης πολυπλοκότητας της σχεδίασης των ολοκληρωμένων κυκλωμάτων, κάνοντάς τα όλο και πιο ευάλωτα στην ακτινοβολία και πιο επιρρεπή σε κατασκευαστικές ατέλειες. Ειδικότερα, οι σύγχρονοι επεξεργαστές αντιμετωπίζουν σοβαρά προβλήματα αξιοπιστίας κατά τη διάρκεια της ζωής τους εξαιτίας: (α) των σφαλμάτων που προέρχονται από την κοσμική ακτινοβολία και από τα φορτισμένα ηλεκτρικά σωματίδια που βάλλουν τα κυκλώματα ακόμα και στο επίπεδο της θάλασσας, (β) της γήρανσης και φθοράς των κυκλωμάτων με την πάροδο του χρόνου, και (γ) των κατασκευαστικών ατελειών που δημιουργούνται κατά την παραγωγή των ολοκληρωμένων κυκλωμάτων.

Οι επεξεργαστές είναι πλέον εξαιρετικά πολύπλοκα συστήματα, καθιστώντας τη διαδικασία επαλήθευσης και επικύρωσης (pre-silicon verification, post-silicon validation) τους σημαντική πρόκληση για τη βιομηχανία των ημιαγωγών. Πάντα προσπαθώντας να παρέχουν υψηλότερες επιδόσεις και ενεργειακή αποδοτικότητα στους τελικούς χρήστες, οι κατασκευαστές επεξεργαστών αναγκάζονται να σχεδιάζουν σταδιακά όλο και πιο πολύπλοκα κυκλώματα, με αποτέλεσμα να απασχολούν πολύ μεγάλες ομάδες επαλήθευσης και επικύρωσης (verification and validation teams) για την έγκαιρη εξάλειψη κρίσιμων σφαλμάτων σχεδιασμού. Για τους παραπάνω λόγους, οι σχεδιαστές επεξεργαστών και κυκλωμάτων χρειάζεται να διασφαλίσουν υψηλά επίπεδα αξιοπιστίας

και ενεργειακής απόδοσης των ολοκληρωμένων κυκλωμάτων πριν αυτά διοχετευθούν στην αγορά.

Πιο συγκεκριμένα, κατά τη διάρκεια σχεδίασης και παραγωγής ενός επεξεργαστή, υπάρχουν δύο πολύ βασικές φάσεις ελέγχου της σωστής λειτουργίας του τσιπ: η φάση της επαλήθευσής του (pre-silicon verification), η οποία πραγματοποιείται κατά τη διαδικασία της σχεδίασης του ολοκληρωμένου κυκλώματος και πριν την παραγωγή του, και η φάση της επικύρωσής του, η οποία πραγματοποιείται αφού ολοκληρωθεί η φάση του σχεδιασμού και παραχθούν τα πρώτα τσιπ. Η φάση της επικύρωσης πραγματοποιείται στο πραγματικό τσιπ. Μόλις η σχεδίαση και η επαλήθευσή της φτάσουν στο τελικό στάδιο και έχουν διορθωθεί τυχόν σχεδιαστικά σφάλματα, οι εταιρίες παράγουν μερικές δεκάδες τσιπ ώστε να ξεκινήσει η φάση της επικύρωσης. Αυτή η φάση είναι η τελευταία και η πιο σημαντική, διότι είναι το στάδιο πριν την μαζική παραγωγή των τσιπ και την διάθεσή τους στην αγορά, και συνεπώς η τελευταία ευκαιρία των κατασκευαστών να διορθώσουν τυχόν λάθη που έχουν απομείνει.

Σκοπός αυτής της διδακτορικής διατριβής είναι να προταθούν πρωτότυπες λύσεις που θα συμβάλλουν στη βελτίωση της αποτελεσματικότητας και επιτάχυνση της φάσης της επικύρωσης ενός οποιουδήποτε επεξεργαστή, όπως επίσης και στο να προταθούν τεχνικές και μέθοδοι οι οποίες συμβάλλουν στην ενίσχυση της ενεργειακής αποδοτικότητας των επεξεργαστών. Εκτιμάται, ότι ο στόχος αυτός επιτυγχάνεται με την ολοκλήρωση αυτής της διατριβής και επιπλέον ένα ακόμη βασικό στοιχείο αυτής της διατριβής είναι ότι προτείνει ένα σύνολο αποτελεσματικών τεχνικών οι οποίες είναι δυνατό να υιοθετηθούν και να εφαρμοστούν άμεσα από τη βιομηχανία. Με άλλα λόγια, μια σημαντική πτυχή της διδακτορικής αυτής διατριβής είναι η έμφαση στη διευθέτηση πραγματικών προβλημάτων της βιομηχανίας (βελτίωση της αποτελεσματικότητας της φάσης της επικύρωσης των επεξεργαστών, καθώς και τη βελτίωση της ενεργειακής κατανάλωσής τους).

Η διαδικασία της επικύρωσης ενός επεξεργαστή κατά τη διάρκεια παραγωγής του (post-silicon validation) αποτελεί αναπόσπαστο και συνεχώς αυξανόμενης σημασίας και διάρκειας τμήμα του κύκλου ζωής ενός επεξεργαστή. Ενδεικτική είναι η συνεχώς αυξανόμενη πίεση που ασκείται στους μηχανικούς υλικού για να ολοκληρώσουν την αποσφαλμάτωση (debug) ενός επεξεργαστή. Στο πλαίσιο αυτό θα πρέπει να ληφθεί υπόψιν η εξαιρετικά μεγάλη πολυπλοκότητα των σύγχρονων επεξεργαστών, σε συνδυασμό με τα στενά χρονικά περιθώρια που τίθενται για την έγκαιρη διάθεση των επεξεργαστών στην αγορά. Η φάση της επικύρωσης έχει ως στόχο να διασφαλίσει ότι τα πρωτότυπα τσιπ που θα παραχθούν συμμορφώνονται με τις προδιαγραφές της αρχικής σχεδίασης του επεξεργαστή, εκτελώντας όσο το δυνατόν περισσότερα προγράμματα δοκιμής (validation tests). Επίσης, επιτρέπει την ανίχνευση λειτουργικών σφαλμάτων που είναι δύσκολο να εντοπιστούν σε προηγούμενες φάσεις ελέγχου, όπως στη φάση της επαλήθευσής του επεξεργαστή πριν την παραγωγή των πρωτότυπων τσιπ (pre-silicon verification), αλλά και των «ηλεκτρικών» σφαλμάτων τα οποία εκδηλώνονται μόνο όταν ο επεξεργαστής βρεθεί σε πραγματικές συνθήκες λειτουργίας. Αποτελεσματικές τεχνικές στο post-silicon validation καθίστανται ακόμη πιο αναγκαίες για την ανίχνευση σφαλμάτων σχεδίασης που παραμένουν μετά το pre-silicon verification και διάφορα ελαττώματα κατασκευής στα πρωτότυπα τσιπ.

Το post-silicon validation σε πραγματικά πρωτότυπα τσιπ προσφέρει πολύ υψηλή απόδοση εκτέλεσης των ελέγχων, πρακτικά πολύ κοντά στις συνθήκες λειτουργίας του ολοκληρωμένου κυκλώματος στο τελικό σύστημα. Έτσι οι ομάδες επικύρωσης προσπαθούν να εκτελέσουν όσο το δυνατόν περισσότερα προγράμματα δοκιμής (validation tests) για να καλύψουν όσο το δυνατό περισσότερες περιπτώσεις ελέγχων (validation coverage) πριν από τη μαζική παραγωγή του ολοκληρωμένου κυκλώματος. Για τον σκοπό αυτό, το post-silicon validation επικεντρώνεται στην εκτέλεση εξαιρετικά

μεγάλου αριθμού παραμετροποιήσιμων τυχαίων προγραμμάτων δοκιμής (parameterized random-generated validation tests). Το σημαντικότερο όμως μειονέκτημα των τυχαίων προγραμμάτων δοκιμής είναι η δυσκολία ελέγχου των αναμενόμενων σωστών αποτελεσμάτων, τα οποία απαιτούνται για να προσδιοριστεί η ορθότητα της παραγωγής των πρωτοτύπων ολοκληρωμένων κυκλωμάτων. Για να ικανοποιηθεί αυτή η απαίτηση, ο εντοπισμός ενός σχεδιαστικού ή κατασκευαστικού σφάλματος προϋποθέτει την εκτέλεση του προγράμματος δοκιμής με τυχειότητα αφενός στο πρωτότυπο μοντέλο του επεξεργαστή και αφετέρου σε έναν αρχιτεκτονικό προσομοιωτή (ο οποίος εκτελεί το ίδιο πρόγραμμα δοκιμής με το πρωτότυπο μοντέλο και εξάγει το σωστό-αναμενόμενο αποτέλεσμα). Στη συνέχεια, τα αποτελέσματα από τις δύο πηγές συγκρίνονται μεταξύ τους έτσι ώστε να προσδιοριστεί η ύπαρξη σφάλματος (διαφορά στα αποτελέσματα) ή η ορθή λειτουργία (ίδια αποτελέσματα). Συνεπώς, η διαδικασία του post-silicon validation περιορίζεται από την απόδοση του αρχιτεκτονικού προσομοιωτή, ο οποίος σε κάθε περίπτωση είναι από 3 έως 6 τάξεις μεγέθους πιο αργός από έναν πραγματικό επεξεργαστή. Για παράδειγμα, περίπου 200 δισεκατομμύρια κύκλοι προσομοιώθηκαν στη διάρκεια πολλών μηνών για τον έλεγχο ορθής λειτουργίας του επεξεργαστή Pentium 4, αλλά αντ' αυτού, στον πραγματικό επεξεργαστή με 1 GHz συχνότητα ρολογιού η ίδια εκτέλεση ολοκληρώθηκε σε 3 λεπτά [91].

Στο πλαίσιο αυτής της διδακτορικής διατριβής παρουσιάζονται δύο θεμελιώδεις μέθοδοι για την επιτάχυνση της διαδικασίας της επικύρωσης (post-silicon validation) των μηχανισμών μετάφρασης διευθύνσεων μνήμης (address translation mechanisms) των σύγχρονων επεξεργαστών, καθώς και για την ανίχνευση σφαλμάτων που είναι δύσκολο να εντοπισθούν με τις υπάρχουσες μεθοδολογίες που χρησιμοποιούνται σήμερα στις κρυφές μνήμες των μηχανισμών μετάφρασης διευθύνσεων μνήμης (address translation caching arrays).

Λόγω της αδιαφανούς λειτουργίας του μηχανισμού μετάφρασης διευθύνσεων ενός επεξεργαστή, η παρατηρησιμότητα και οι καθυστερήσεις ανίχνευσης σφαλμάτων είναι από τα πιο σημαντικά εμπόδια για τον έλεγχο της ορθής σχεδίασης και λειτουργίας του πρωτότυπου τσιπ. Λόγω της ραγδαίας εξέλιξης και χρήσης των εικονικών μηχανών (virtual machines) στους σύγχρονους επεξεργαστές (και της πολυπλοκότητας που αυξάνεται στους επεξεργαστές ώστε να είναι ικανοί να υποστηρίζουν πολλαπλές εικονικές μηχανές), η απόδοση και η ορθότητα των μηχανισμών μετάφρασης διευθύνσεων αποκτούν ιδιαίτερα μεγάλη σημασία σε σύγκριση με το πρόσφατο παρελθόν. Οι μηχανισμοί αυτοί ενός σύγχρονου επεξεργαστή περιλαμβάνουν σύνθετες δομές υλικού και μπορούν να αποτελέσουν κυρίαρχη πηγή σοβαρών διαφυγόντων σφαλμάτων, τα οποία είναι πολύ δύσκολο να ανιχνευθούν.

Σε αντίθεση με άλλες λειτουργικές μονάδες του επεξεργαστή που είναι άμεσα προσπελάσιμες με εντολές μηχανής (καταχωρητές, κρυφές μνήμες, μονάδες περιφερειακών μονάδων ελέγχου, κλπ.), η έξοδος του μηχανισμού μετάφρασης διευθύνσεων (δηλαδή η φυσική διεύθυνση) δεν παρατηρείται σε ορατά σημεία ενός προγράμματος ή στα αρχιτεκτονικά στοιχεία του επεξεργαστή (π.χ., καταχωρητές). Επιπλέον, η διαδικασία μετάφρασης διευθύνσεων περιλαμβάνει διάφορα στάδια και διαφορετικές δομές υλικού (TLBs, MMU caches, κ.λπ.) και σφάλματα σχεδίασης σε κάθε μια από αυτές τις δομές μπορούν να οδηγήσουν σε λανθασμένες μεταφράσεις διευθύνσεων, και συνεπώς σε σοβαρά προβλήματα εκτέλεσης (είτε εσφαλμένα αποτελέσματα είτε μειωμένη απόδοση) των προγραμμάτων αλλά και των εικονικών μηχανών.

Συγκεκριμένα, στις εργασίες [2] και [3] προτείνεται η πρώτη συνεισφορά αυτής της διδακτορικής διατριβής, στις οποίες περιγράφεται μια μέθοδος επικύρωσης του επεξεργαστή, που συμβάλλει στην επιτάχυνση της διαδικασίας ανίχνευσης σχεδιαστικών

σφαλμάτων στους μηχανισμούς μετάφρασης διευθύνσεων των σύγχρονων επεξεργαστών. Αρχικά, προτείνεται ένα ολοκληρωμένο και περιεκτικό σύνολο μοντέλων σφαλμάτων που είναι πιθανόν να εμφανιστούν στους μηχανισμούς μετάφρασης διευθύνσεων μνήμης. Τα προτεινόμενα μοντέλα κατηγοριοποιούν τα αποτελέσματα των λειτουργικών και των «ηλεκτρικών» σφαλμάτων στις δομές υλικού που χρησιμοποιούνται για τη μετάφραση διευθύνσεων. Κατόπιν, προτείνεται μια μέθοδος για την επικύρωση του επεξεργαστή, η οποία είναι ανεξάρτητη από την αρχιτεκτονική συνόλου εντολών (ISA-independent), και στη συνέχεια αξιολογείται ως προς την αποτελεσματικότητά της με τη χρήση των μοντέλων σφαλμάτων που προτάθηκαν. Η προτεινόμενη μέθοδος επιδιώκει να επιταχύνει σημαντικά την διαδικασία ανίχνευσης των σφαλμάτων στους μηχανισμούς μετάφρασης διευθύνσεων και εκμεταλλεύεται πλήρως την απόδοση του πρωτότυπου τσιπ.

Αποδεικνύεται πειραματικά ότι, η προτεινόμενη μέθοδος επιταχύνει την ανίχνευση σφάλματος κατά 5 τάξεις μεγέθους σε σύγκριση με τις κλασσικές τεχνικές επικύρωσης (οι οποίες συγκρίνουν το αποτέλεσμα του πρωτότυπου με την όμοια εκτέλεση σε έναν αρχιτεκτονικό προσομοιωτή στο τέλος της εκτέλεσης του προγράμματος δοκιμής). Στο σημείο αυτό, είναι σημαντικό να τονιστεί ότι για την πειραματική αξιολόγηση της μεθόδου έγινε μια σημαντική ενίσχυση του προσομοιωτή Gem5, στον οποίο υλοποιήθηκε και ενσωματώθηκε η λειτουργία των κρυφών μνημών των μηχανισμών μετάφρασης διευθύνσεων μνήμης (MMU caches), με σκοπό να είναι ικανή η ολοκληρωμένη μίμηση της διαδικασίας της επικύρωσης σε έναν πλήρη σύγχρονο επεξεργαστή και να αποδειχθεί η αποτελεσματικότητα της προτεινόμενης μεθόδου κατόπιν ενδελεχούς πειραματικής αξιολόγησης.

Ως συνέχεια της συνεισφοράς της διατριβής στην περιοχή της επικύρωσης του επεξεργαστή, στην εργασία [4] παρουσιάζεται μια νέα μέθοδος επικύρωσης (συμπληρωματική της προηγούμενης), η οποία εντοπίζει και αποκαλύπτει σπάνια σενάρια σφαλμάτων στις κρυφές μνήμες των μηχανισμών μετάφρασης διευθύνσεων των επεξεργαστών. Αυτές οι κρυφές μνήμες είναι από τις πιο σημαντικές δομές για τη λειτουργικότητα και κυρίως για την απόδοση των σύγχρονων επεξεργαστών, και τα σφάλματα που διαφεύγουν σε αυτές τις κρυφές μνήμες μπορούν να οδηγήσουν σε απρόβλεπτες συμπεριφορές του συστήματος κατά τη διάρκεια κανονικής λειτουργίας (εσφαλμένους υπολογισμούς ή επιβάρυνση στον χρόνο εκτέλεσης). Χρησιμοποιώντας μια περιεκτική πειραματική μελέτη, σε αυτή τη διατριβή παρουσιάζονται και αναλύονται αυτού του είδους τα σπάνια σενάρια σφαλμάτων, και εξηγείται ο λόγος που καθιστά δύσκολο τον εντοπισμό αυτών των σφαλμάτων. Ακόμα κι αν αυτού του είδους τα σφάλματα εκδηλώνονται με την εκτέλεση των κλασσικών μεθόδων επικύρωσης (με τη χρήση προσομοιωτών για τον έλεγχο των αποτελεσμάτων), η ανίχνευσή τους είναι απίθανη. Τέτοιου είδους σφάλματα, και ως εκ τούτου σπάνια, έχουν την ιδιότητα να «καλύπτονται» (masking) κατά τη διαδικασία εκτέλεσης προγραμμάτων επικύρωσης, ή να οδηγούν σε προβλήματα απόδοσης χωρίς να επηρεάζουν τη σωστή εκτέλεση των προγραμμάτων, με αποτέλεσμα οι εκτελέσεις των κλασσικών προγραμμάτων επικύρωσης να μην τα αντιλαμβάνονται και λανθασμένα να αποφαίνονται για την ορθότητα της εκτέλεσης. Η πειραματική μελέτη και αξιολόγηση που παρουσιάζεται στο σχετικό κεφάλαιο της διατριβής, αποδεικνύει ότι, σε αντίθεση με τις κλασσικές τεχνικές επικύρωσης, η μέθοδος που προτείνεται εντοπίζει αποτελεσματικά όλα τα σπάνια σφάλματα που δημιουργήθηκαν και εισήχθησαν στον αρχιτεκτονικό προσομοιωτή.

Και οι δύο μέθοδοι που προτείνονται καθιστούν ευκολότερη και ταχύτερη την ανίχνευση σφαλμάτων κατά τη διάρκεια της φάσης της επικύρωσης του επεξεργαστή. Ωστόσο, κάποια σφάλματα (κυρίως «ηλεκτρικά» σφάλματα) τα οποία είναι πιθανό να διαφύγουν ακόμα και μετά απ' τον πιο αυστηρό έλεγχο, που κάνουν ακόμα και κυκλώματα που



θεωρητικά είναι κατασκευασμένα να λειτουργούν υπό τις ίδιες συνθήκες τελικά να λειτουργούν ορθά κάτω από διαφορετικές συνθήκες, συνήθως ωθούν τους κατασκευαστές ολοκληρωμένων κυκλωμάτων στην υιοθέτηση απαισιόδοξων (υψηλών) περιθωρίων τάσης λειτουργίας (pessimistic voltage margins), τα οποία κατ' επέκταση θυσιάζουν την ενεργειακή απόδοση των προϊόντων.

Επιπρόσθετα, η γήρανση των τρανζίστορ και η δυναμική διακύμανση της τάσης εισόδου και της θερμοκρασίας, που προκαλούνται από διαφορετικές αλληλεπιδράσεις φορτίου εργασίας και μικροαρχιτεκτονικής, έχουν επίσης αντίκτυπο στην υιοθέτηση απαισιόδοξων περιθωρίων τάσης λειτουργίας. Τόσο οι στατικές όσο και οι δυναμικές διακυμάνσεις στην τάση λειτουργίας του επεξεργαστή οδηγούν τους σχεδιαστές επεξεργαστών να εφαρμόζουν αυτά τα απαισιόδοξα περιθώρια τάσης για τη λειτουργία του επεξεργαστή, έτσι ώστε να διασφαλίζεται η σωστή του λειτουργία, ακόμη και στις χειρότερες συνθήκες. Ωστόσο, αυτά τα απαισιόδοξα υψηλά περιθώρια τάσης εμποδίζουν τη χαμηλή κατανάλωση ενέργειας και την υψηλή απόδοση, η οποία μπορεί να επιτευχθεί με τη μείωση της τάσης τροφοδοσίας ή την αύξηση της συχνότητας λειτουργίας, αντίστοιχα. Αρκετές τεχνικές και μέθοδοι σε επίπεδο μικροαρχιτεκτονικής έχουν προταθεί κατά καιρούς για την εξάλειψη ενός υποσυνόλου αυτών των απαισιόδοξων περιθωρίων τάσης που τίθεται από τους σχεδιαστές επεξεργαστών. Ωστόσο, όλες αυτές οι μικροαρχιτεκτονικές τεχνικές αυξάνουν το κόστος της επαλήθευσης και του ελέγχου ορθής λειτουργίας του τσιπ και περιορίζουν την ενσωμάτωση άλλων ή πιο αποδοτικών λειτουργικών μονάδων λόγω του επιπλέον χώρου που καταλαμβάνουν στο τσιπ.

Η εύρεση και η αξιοποίηση των απαισιόδοξων περιθωρίων τάσης προσφέρει μια σημαντική ευκαιρία για ενεργειακά αποδοτικότερη λειτουργία των επεξεργαστών. Το πλήρες δυναμικό εξοικονόμησης ενέργειας μπορεί να εκτεθεί μόνο όταν μετριέται η διακύμανση της τάσης από πυρήνα σε πυρήνα, από τσιπ σε τσιπ και κυρίως με διαφορετικά προγράμματα. Όταν εντοπίζονται όλα αυτά τα επίπεδα διακύμανσης (variation), το λογισμικό του συστήματος (ή ο χρονοπρογραμματιστής) μπορεί να καταναείμει αποτελεσματικά τους πόρους υλικού στις εργασίες λογισμικού που ταιριάζουν στις δυνατότητες του υλικού και στις απαιτήσεις του λογισμικού, όσον αφορά την ενέργεια και την απόδοση. Για παράδειγμα, κάποιες μονάδες υλικού (πυρήνες) χρειάζονται μεγαλύτερη τάση για να λειτουργήσουν σωστά, ενώ κάποιες άλλες μικρότερη. Ως εκ τούτου, προγράμματα που προκαλούν μεγάλες μεταβολές πρέπει να ανατεθούν στα πρώτα ενώ όσα προκαλούν μικρές στα δεύτερα.

Στα πλαίσια αυτής της διδακτορικής διατριβής, και συγκεκριμένα αρχικά στις εργασίες [10] και [11], παρουσιάζεται μια λεπτομερής μελέτη χαρακτηρισμού των περιθωρίων τάσης για εκτελέσεις προγραμμάτων σε ένα πυρήνα (single-core executions), σε επεξεργαστές οι οποίοι βασίζονται στην αρχιτεκτονική ARMv8 και είναι κατασκευασμένοι σε τεχνολογία 28nm. Η βάση αυτής της μελέτης είναι ένα πλήρως αυτοματοποιημένο περιβάλλον σε επίπεδο ενός πλήρους υπολογιστικού συστήματος, το οποίο βασίζεται στον επεξεργαστή X-Gene 2 της εταιρείας Applied Micro (APM – σήμερα ονομάζεται Ampere Computing). Η αυτοματοποιημένη υποδομή που έχει δημιουργηθεί στοχεύει στην αύξηση της απόδοσης των μαζικών εκτελέσεων των προγραμμάτων που απαιτούν πολλαπλές όμοιες εκτελέσεις (δεδομένου ότι η πειραματική διαδικασία γίνεται σε πραγματικό επεξεργαστή, και συνεπώς μεταξύ όμοιων εκτελέσεων μπορεί να υπάρξουν μικροδιαφορές στα αποτελέσματα) σε διάφορα επίπεδα τάσης λειτουργίας και σε όλους τους διαθέσιμους πυρήνες του επεξεργαστή. Η αυτοματοποιημένη διαδικασία χαρακτηρισμού των περιθωρίων τάσης που υλοποιήθηκε στα πλαίσια αυτής της διατριβής απαιτεί ελάχιστη ανθρώπινη παρέμβαση και είναι ικανή να καταγράφει όλες τις πιθανές ανωμαλίες που οφείλονται στην μείωση της τάσης: αναντιστοιχία της εξόδου ενός προγράμματος χωρίς ειδοποίηση σφάλματος υλικού (silent data corruption), διορθωμένα

σφάλματα (corrected errors), μη διορθωμένα (αλλά ανιχνευόμενα) σφάλματα (uncorrected errors), καθώς και σφάλματα του συστήματος (system crashes).

Για την μελέτη και καταγραφή της συμπεριφοράς του συστήματος υπό συνθήκες μειωμένης τάσης, παρουσιάζεται επίσης σε αυτή τη διατριβή μια απλή και ενοποιημένη συνάρτηση: η συνάρτηση πυκνότητας-σοβαρότητας (severity function). Η συνάρτηση πυκνότητας-σοβαρότητας συγκεντρώνει τα αποτελέσματα που προκύπτουν λόγω της λειτουργίας μειωμένης τάσης στους πυρήνες του επεξεργαστή, αναθέτοντας μεμονωμένες τιμές στις διάφορες μη φυσιολογικές συμπεριφορές που εμφανίστηκαν κατά τη διάρκεια του χαρακτηρισμού των περιθωρίων τάσης. Η θεμελιώδης αρχή αυτής της συνάρτησης είναι: όσο χαμηλότερη είναι η τάση, τόσο μεγαλύτερη είναι η τιμή του αποτελέσματος της συνάρτησης πυκνότητας-σοβαρότητας. Η συνάρτηση πυκνότητας-σοβαρότητας εξυπηρετεί στην ταξινόμηση των όμοιων πυρήνων ενός επεξεργαστή για ένα δεδομένο σημείο αναφοράς: οι τιμές που παράγει η συνάρτηση, για διαφορετικούς πυρήνες και διαφορετικές τάσεις, οδηγούν σε διαφορετικές τιμές σοβαρότητας. Για παράδειγμα, ενώ το σύστημα παραμένει ευαίσθητο σε όλο το φάσμα των τιμών τάσης, σε κάποιες τιμές τάσης μπορεί δημιουργεί σφάλματα που ανιχνεύονται από τους μηχανισμούς ανίχνευσης και διόρθωσης σφαλμάτων του υλικού (ECC) ή σε κάποιες άλλες να παράγει αναντιστοιχία της εξόδου ενός προγράμματος χωρίς ειδοποίηση σφάλματος υλικού.

Η λεπτομερής ανάλυση της συμπεριφοράς του επεξεργαστή χρησιμοποιώντας τη συνάρτηση πυκνότητας-σοβαρότητας μπορεί να βοηθήσει στις αποφάσεις ενεργειακής απόδοσης. Επίσης, ο πλήρης χαρακτηρισμός των περιθωρίων τάσης που παρουσιάζεται σε αυτή τη διατριβή (α) επιβεβαιώνει ότι μια διαφορετική μικροαρχιτεκτονική, σχεδίαση κυκλώματος ή τεχνολογία κατασκευής παρουσιάζει διαφορετική μη φυσιολογική συμπεριφορά όταν λειτουργεί πέρα από τις ονομαστικές (nominal) συνθήκες τάσης που έχουν οριστεί από τους σχεδιαστές του επεξεργαστή και (β) μπορεί να χρησιμοποιηθεί αποτελεσματικά για τη στήριξη αποφάσεων σχεδιασμού επεξεργαστών αλλά και λογισμικού, με σκοπό την αξιοποίηση των απαισιόδοξων περιθωρίων τάσης, και συνεπώς, τη βελτίωση της ενεργειακής απόδοσης, διατηρώντας παράλληλα την ορθή λειτουργία και τις επιδόσεις του επεξεργαστή.

Δυστυχώς, οι μελέτες χαρακτηρισμού των περιθωρίων τάσης (όπως αυτή που παρουσιάζεται σε αυτή τη διατριβή) είναι μια χρονοβόρος διαδικασία. Η ακριβής αναγνώριση αυτών των ορίων σε ένα πραγματικό σύστημα απαιτεί μαζική εκτέλεση μεγάλου αριθμού προγραμμάτων σε όλους τους πυρήνες του επεξεργαστή (και όλων των διαφορετικών παραγόμενων τσιπ), για διαφορετικές τιμές τάσης και συχνότητας. Για παράδειγμα, για να προσδιοριστούν τα ασφαλή όρια τάσης, σε κάθε έναν από τους οκτώ πυρήνες του επεξεργαστή X-Gene 2 της Applied Micro (APM) για τη μέγιστη μόνο συχνότητα, χρησιμοποιήθηκαν τα μετροπρογράμματα SPEC CPU2006, και κάθε πείραμα εκτελέστηκε επαναληπτικά 10 φορές (για να περιοριστούν οι μικροδιαφορές που μπορεί να συμβούν μεταξύ όμοιων εκτελέσεων) ξεκινώντας από την ονομαστική τιμή τάσης (980 mV) μέχρι την τιμή τάσης στην οποία το σύστημα δεν ήταν πια λειτουργικό (~ 880 mV). Αυτά τα πειράματα απαιτούσαν περίπου 2 μήνες για έναν πλήρη χαρακτηρισμό των ορίων τάσης για όλους τους πυρήνες ενός τσιπ.

Για την επίτευξη της διαδικασίας του χαρακτηρισμού των περιθωρίων τάσης γρήγορα και αποτελεσματικά, σε αυτή τη διατριβή, και συγκεκριμένα στην εργασία [12], παρουσιάζεται αναλυτικά η ανάπτυξη ειδικά σχεδιασμένων προγραμμάτων (micro-viruses) που αποσκοπούν στο να δημιουργήσουν μεγάλο φορτίο εργασίας στις θεμελιώδεις δομές του επεξεργαστή. Με αυτά τα προγράμματα, μπορούν να χαρακτηριστούν (μεμονωμένα ή σε συνδυασμούς) όλες οι θεμελιώδεις δομές του επεξεργαστή: (α) οι κρυφές μνήμες εντολών και δεδομένων 1<sup>ου</sup> επιπέδου, οι ενιαίες κρυφές μνήμες 2<sup>ου</sup> επιπέδου, και το τελευταίο (3<sup>ο</sup>)

επίπεδο κρυφής μνήμης, και (β) οι δύο πιο θεμελιώδεις μονάδες της διοχέτευσης (η αριθμητική και λογική μονάδα, και η μονάδα κινητής υποδιαστολής).

Αυτά τα προγράμματα εκτελούνται σε πολύ σύντομο χρονικό διάστημα (περίπου σε 3 ημέρες για το συνολικό χαρακτηρισμό των περιθωρίων ασφαλούς τάσης για κάθε μεμονωμένο πυρήνα και για κάθε chip) σε σύγκριση με τα κανονικά μετροπρογράμματα, όπως αυτά της σουίτας SPEC CPU2006 που χρειάστηκαν περίπου 2 μήνες εκτέλεσης για κάθε ένα τσιπ (δηλαδή 6 περίπου μήνες για τα τρία διαφορετικά τσιπ που χρησιμοποιήθηκαν). Ο σκοπός αυτών των ειδικών προγραμμάτων είναι να αποκαλύψουν την διακύμανση των περιθωρίων ασφαλούς τάσης στους διαφορετικούς πυρήνες του επεξεργαστή, και επίσης να συμβάλλουν στη διάγνωση των απρόσμενων συμπεριφορών εκθέτοντας και κατηγοριοποιώντας την ανώμαλη συμπεριφορά κάθε τσιπ (αλλοιώσεις δεδομένων, σφάλματα στις κρυφές μνήμες, κλπ.).

Τέλος, σε αυτή τη διατριβή και συγκεκριμένα στην εργασία [13], πραγματοποιείται ο χαρακτηρισμός των περιθωρίων τάσης σε εκτελέσεις πολλαπλών πυρήνων, καθώς επίσης και σε διαφορετικές συχνότητες, και προτείνεται ένα πρόγραμμα το οποίο εκμεταλλεύεται όλες τις διαφορετικές πτυχές του προβλήματος της κατανάλωσης ενέργειας και παρέχει μεγάλη εξοικονόμηση ενέργειας διατηρώντας παράλληλα υψηλά επίπεδα απόδοσης. Σε αυτό το μέρος της διατριβής, αρχικά εκτίθενται τα απαισιόδοξα περιθώρια τάσης σε εκτελέσεις πολλαπλών πυρήνων δύο επεξεργαστών βασισμένοι στην ARMv8 αρχιτεκτονική (κατασκευασμένοι σε 28nm και 16nm - το X-Gene 2 και X-Gene 3, αντίστοιχα). Η βασική συνεισφορά αυτής της μελέτης συγκριτικά με την προηγούμενη (που αφορούσε τον χαρακτηρισμό μεμονωμένων πυρήνων του επεξεργαστή) είναι ότι αποδεικνύεται πειραματικά πως καθώς ο αριθμός των ενεργών νημάτων/διεργασιών στον επεξεργαστή αυξάνεται, η διακύμανση των περιθωρίων ασφαλούς τάσης στους διαφορετικούς πυρήνες αλλά και των διαφορετικών προγραμμάτων που εκτελούνται έχουν ελάχιστη επίδραση στα περιθώρια ασφαλούς τάσης του επεξεργαστή.

Στη συνέχεια παρουσιάζονται μετρήσεις και αποτελέσματα σχετικά με τη σχέση των περιθωρίων ασφαλούς τάσης και το μέγεθος της διακύμανσης της τάσης σε διαφορετικές συχνότητες λειτουργίας του επεξεργαστή, και αποδεικνύεται ότι σε πολυπύρηνες εκτελέσεις, οι απότομες διακυμάνσεις τάσης (οι οποίες τελικά συμβάλλουν στον προσδιορισμό της τάσης ασφαλούς λειτουργίας) συμβαίνουν ανεξάρτητα από το πρόγραμμα που εκτελείται την κάθε στιγμή. Ωστόσο, για εκτελέσεις σε έναν μόνο ή πολύ λίγους πυρήνες, η μεταβλητότητα των πυρήνων και των προγραμμάτων συνεχίζει να υφίσταται. Κατόπιν, παρουσιάζεται μια εκτενής μελέτη για τον εντοπισμό και την ανάλυση της σχέσης μεταξύ ενέργειας και απόδοσης σε διαφορετικούς συνδυασμούς τάσης και συχνότητας, καθώς και σε διαφορετικό αριθμό νημάτων/διεργασιών που εκτελούνται στο σύστημα, αλλά και κατανομής των προγραμμάτων στους διαθέσιμους πυρήνες. Η ανάλυσή που παρουσιάζεται, αποκαλύπτει ότι, ανάλογα με τα χαρακτηριστικά ενός προγράμματος και τον αριθμό των ενεργών νημάτων, υπάρχει ένας βέλτιστος συνδυασμός τάσης, συχνότητας και κατανομής των προγραμμάτων στους διαθέσιμους πυρήνες για τη βέλτιστη ενεργειακή απόδοση.

Έτσι, εκμεταλλευόμενοι όλες τις παραπάνω παρατηρήσεις, παρουσιάζεται η υλοποίηση ενός προγράμματος παρακολούθησης (monitoring daemon), το οποίο παρακολουθεί τις τρέχουσες διεργασίες του συστήματος και καθοδηγεί τον χρονοπρογραμματιστή του συστήματος ώστε να λάβει τις κατάλληλες αποφάσεις σχετικά με: (α) τον πυρήνα στον οποίο πρέπει να ανατεθεί μια νέα διεργασία και (β) όταν μία ή περισσότερες διεργασίες πρέπει να μεταφερθούν σε άλλους πυρήνες (process migration). Ταυτόχρονα, το πρόγραμμα ρυθμίζει δυναμικά την τάση και την συχνότητα του επεξεργαστή σύμφωνα με τις βέλτιστες πολιτικές ενέργειας-απόδοσης οι οποίες προέκυψαν από την εκτενή μελέτη

που προηγήθηκε. Τέλος, παρουσιάζεται η αξιολόγηση των προτεινόμενων πολιτικών εξοικονόμησης ενέργειας εκτελώντας ένα ρεαλιστικό σενάριο λειτουργίας ενός διακομιστή, το οποίο

- επιλέγει τυχαία τα προγράμματα που θα εκτελεστούν,
- μετακινεί δυναμικά τις τρέχουσες διεργασίες στο σύστημα, και
- ρυθμίζει δυναμικά την τάση και τη συχνότητα.

Αναφέρονται αρκετές συγκρίσεις μεταξύ διαφορετικών περιπτώσεων για να αποδειχθεί η αποτελεσματικότητα της προτεινόμενης πολιτικής, η οποία μπορεί τελικά να πετύχει κατά μέσο όρο 25,2% εξοικονόμηση ενέργειας στο X-Gene 2 και 22,3% στο X-Gene 3, με ελάχιστη μείωση της απόδοσης στο 3,2% για το X-Gene 2 και στο 2,5% για το X-Gene 3 σε σύγκριση με τις αρχικές συνθήκες τάσης και συχνότητας του επεξεργαστή.

Συνοψίζοντας, ο βασικός στόχος όλων των μεθόδων που παρουσιάζονται στη διατριβή αυτή είναι, αφενός μεν να βελτιώσουν και να επιταχύνουν τη φάση της επικύρωσης των επεξεργαστών, και αφετέρου να βελτιώσουν την ενεργειακή απόδοση των σύγχρονων επεξεργαστών.

Η δομή της διδακτορικής διατριβής είναι η ακόλουθη: Το Κεφάλαιο 1 είναι μια εισαγωγή που περιγράφει τα προβλήματα που αντιμετωπίζουν οι σύγχρονοι επεξεργαστές όσον αφορά την κατανάλωση ισχύος αλλά και τη διαδικασία της επικύρωσης. Το Κεφάλαιο 2 περιγράφει τις πρώτες δύο συνεισφορές της διατριβής ως προς την επιτάχυνση και την αποκάλυψη σπάνιων σφαλμάτων σχεδίασης στους μηχανισμούς μετάφρασης διευθύνσεων κατά τη φάση της επικύρωσης του επεξεργαστή. Το Κεφάλαιο 3 περιγράφει την τρίτη συνεισφορά της διατριβής που αφορά τον χαρακτηρισμό των ασφαλών περιθωρίων τάσης σε εκτελέσεις ενός πυρήνα, καθώς και την τέταρτη συνεισφορά που αφορά την παρουσίαση διαγνωστικών προγραμμάτων που στοχεύουν στην επιτάχυνση της διαδικασίας του χαρακτηρισμού των περιθωρίων τάσης. Στο Κεφάλαιο 4 παρουσιάζεται η πέμπτη κατά σειρά συνεισφορά της διατριβής η οποία αναδεικνύει τη συμπεριφορά των προγραμμάτων σε πολυπύρηνες εκτελέσεις σε μειωμένες συνθήκες τάσης, και παρουσιάζεται η ανάπτυξη ενός προγράμματος που στοχεύει στη μείωση της συνολικής ενέργειας κατά τη λειτουργία του επεξεργαστή. Τέλος, το Κεφάλαιο 5 είναι μία ανασκόπηση συμπερασμάτων στα συγκεκριμένα θέματα της διατριβής και μερικές προτάσεις για μελλοντικές ερευνητικές κατευθύνσεις στα θέματα της επικύρωσης και της ενεργειακής απόδοσης των επεξεργαστών.

# TABLE OF CONTENTS

<b>1. INTRODUCTION .....</b>	<b>41</b>
1.1 Transistor Scaling and Microprocessor Evolution .....	41
1.2 Power Reduction Techniques .....	43
1.3 Supply Voltage Scaling.....	47
1.4 Microprocessors' Dependability Life-Cycle .....	48
1.5 Failures During Microprocessor's Life-Cycle.....	51
1.6 Contribution of This Thesis.....	52
1.7 Thesis Outline.....	56
<b>2. POST-SILICON VALIDATION OF THE ADDRESS TRANSLATION MECHANISMS .....</b>	<b>59</b>
2.1 Address Translation Mechanisms .....	59
2.1.1 Overview .....	59
2.1.2 ATM Organizations .....	60
2.1.3 x86-64 Address Translation .....	61
2.1.4 POWER8 Address Translation.....	62
2.1.5 ARMv8-A Address Translation.....	63
2.1.6 Terminology and Assumptions.....	63
2.2 Reducing the Bug Detection Latency for ATM Post-Silicon Validation.....	64
2.2.1 ATM Bug Models.....	66
2.3 Proposed Method .....	67
2.3.1 Page Table Setup .....	67
2.3.2 Memory Image for ATM Post-Silicon Validation .....	69
2.3.3 Utilizing Unused Bits in Physical Addresses.....	71
2.3.4 ISA-Independent Data Loading in Physical Memory .....	72
2.3.5 Transformation of Validation Tests .....	72
2.3.6 Reducing the Error Detection Latency .....	73
2.4 Experimental Evaluation.....	74
2.4.1 Simulator Setup and Methodology.....	74
2.4.2 Validation Programs Implementation .....	76
2.4.3 Results .....	77
2.5 Bug Coverage and Method Limitations.....	79
2.6 Unveiling Difficult Bugs in Address Translation Caching Arrays .....	79
2.7 Impact of Bugs in Address Translation Caching Arrays.....	80
2.7.1 Motivation .....	80
2.7.2 Evaluating the Impact of Bugs in Address Translation Caching Arrays .....	80
2.7.3 Findings and Observations .....	82
2.8 Generating Validation Tests to Unveil Difficult Bugs in ATCAs.....	84
2.8.1 The Main Concept.....	84
2.8.2 Exercisers.....	85
2.8.3 Checkers .....	86
2.8.4 Example of the Proposed Validation Flow .....	87

<b>2.9</b>	<b>Experimental Evaluation .....</b>	<b>91</b>
2.9.1	Experiment Setup .....	91
2.9.2	Results .....	91
<b>2.10</b>	<b>Related Work .....</b>	<b>93</b>
<b>3.</b>	<b>MEASURING VOLTAGE GUARDBANDS OF SERVER-GRADE ARMV8 CPU CORES .....</b>	<b>95</b>
<b>3.1</b>	<b>System Architecture .....</b>	<b>96</b>
<b>3.2</b>	<b>Automated Characterization Framework Overview .....</b>	<b>98</b>
3.2.1	Initialization Phase .....	100
3.2.2	Execution Phase .....	101
3.2.3	Parsing Phase .....	101
<b>3.3</b>	<b>System Characterization .....</b>	<b>103</b>
3.3.1	Regions of Operation .....	105
3.3.2	$V_{min}$ Experimental Results .....	108
3.3.3	Process Variation .....	109
3.3.4	Abnormal Behaviors below $V_{min}$ .....	110
3.3.5	Severity Function .....	111
3.3.6	Suggestions for Undervolting Effects Mitigation in X-Gene 2-like CPUs .....	117
<b>3.4</b>	<b>Design Enhancements .....</b>	<b>118</b>
<b>3.5</b>	<b>Fast System-Level Voltage Margins Characterization .....</b>	<b>119</b>
<b>3.6</b>	<b>Micro-Viruses Description .....</b>	<b>120</b>
3.6.1	L1 Data Cache Micro-Virus .....	123
3.6.2	L1 Instruction Cache Micro-Virus .....	124
3.6.3	L2 Cache Micro-Virus .....	125
3.6.4	L3 Cache Micro-Virus .....	126
3.6.5	Arithmetic and Logic Unit (ALU) Micro-Virus .....	127
3.6.6	Floating-Point Unit (FPU) Micro-Virus .....	128
3.6.7	Pipeline Micro-Virus .....	128
<b>3.7</b>	<b>Micro-Viruses Validation .....</b>	<b>129</b>
<b>3.8</b>	<b>Experimental Evaluation .....</b>	<b>130</b>
3.8.1	SPEC Benchmarks vs. Micro-Viruses .....	132
3.8.2	Observations .....	135
<b>3.9</b>	<b>Related Work .....</b>	<b>136</b>
<b>4.</b>	<b>BALANCING ENERGY AND PERFORMANCE ON MULTICORE ARMV8 CPUS 139</b>	
<b>4.1</b>	<b>Experimental Setup .....</b>	<b>140</b>
4.1.1	Platforms .....	140
4.1.2	Experimental Configuration .....	140
<b>4.2</b>	<b>Voltage Margins Identification .....</b>	<b>144</b>
4.2.1	Exposing Safe $V_{min}$ Values .....	144
4.2.2	Unsafe Region Investigation .....	147
<b>4.3</b>	<b>Analysis of <math>V_{min}</math> Impact Factors .....</b>	<b>148</b>
4.3.1	Impact of Frequency and Core Allocation on Safe $V_{min}$ .....	148
4.3.2	Impact of the Workload on Frequency and Core Allocation .....	149
4.3.3	Summary of the Factors that Impact the $V_{min}$ .....	152

<b>4.4</b>	<b>Performance and Energy Trade-Offs.....</b>	<b>153</b>
4.4.1	Energy Efficiency .....	154
4.4.2	Combined Energy and Performance Considerations .....	156
<b>4.5</b>	<b>Mitigating Energy: A Real System Implementation .....</b>	<b>156</b>
4.5.1	Online Monitoring Daemon .....	156
4.5.2	Evaluation Results .....	159
<b>4.6</b>	<b>Related Work.....</b>	<b>162</b>
<b>5.</b>	<b>CONCLUSION AND FUTURE WORK.....</b>	<b>165</b>
	<b>ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ .....</b>	<b>167</b>
	<b>ACRONYMS.....</b>	<b>169</b>
	<b>ANNEX I .....</b>	<b>171</b>
	<b>REFERENCES .....</b>	<b>175</b>





## LIST OF FIGURES

Figure 1: CPU transistor count and feature size trend [18].	41
Figure 2: Power densities rising [29].	44
Figure 3: A 45nm Core™ i7 processor (Nehalem) [18].	45
Figure 4: Nehalem power control unit (PCU). The PCU integrates proprietary microcontroller, which shifts control from hardware to embedded firmware. It also contains real-time sensors for voltage, current, power, and temperature [18].	46
Figure 5: Voltage guardband ensures reliability by inserting an extra timing margin. Reduced voltage margins improve total system efficiency without affecting the reliability of the microprocessor.	48
Figure 6: Processor dependability life-cycle.	49
Figure 7: Distribution of failures during microprocessor's life-cycle.	51
Figure 8: Relative cost of finding bugs throughout microprocessor life-cycle [95].	52
Figure 9: Address translation process overview.	60
Figure 10: Paging with 4KB page size in x86-64 ISA (PA=Physical Address, PTE=Page Table Entry, PDE=Page Directory Entry, PDPTE=Page Directory Pointer Entry, PML4=Page Map Level 4).	61
Figure 11: An x86-64 Page Table Entry (PTE).	62
Figure 12: The proposed ATM silicon validation framework.	68
Figure 13: a) A conventional post-silicon validation flow vs. b) the proposed self-checking validation flow.	69
Figure 14: The Validation Program accesses memory through the ATM. The ATM translates the VA to the correct PA and the fetched (pre-stored) data value is returned back to the validation program, which compares it with the expected PA.	70
Figure 15: The Validation Program accesses memory through the ATM. The ATM translates the VA to a wrong PA and the fetched (pre-stored) data value is returned back to the validation program, which compares it with the expected PA. A mismatch occurs indicating the bug.	71
Figure 16: A 64-bit PA and the PFN bits, the Offset bits and the unused bits in (a) POWER8, (b) x86-64, and (c) ARMv8-A.	71
Figure 17: A conventional example of a validation program (left) that checks for correctness at the end of execution vs. the proposed transformations to achieve the self-checking property (right) that checks for correction right after each memory access.	73
Figure 18: a) A conventional validation process with end-of-test checking, and b) the proposed self-checking validation process.	74
Figure 19: Proposed post-silicon validation flow and the experiment setup.	75
Figure 20: Bug detection latency for traditional end-of-test checking methods and the proposed self-checking.	78
Figure 21: Masked vs. Affected entries in ATCAs for instruction pages show the probability of a bug to affect program outcome.	83
Figure 22: Masked vs. Affected entries in ATCAs for data pages show the probability of a bug to affect program outcome.	83

Figure 23: Validation test flow example – Pass; no bug detected.....	88
Figure 24: Validation test flow - Ignored Invalidation in ITLB. ....	89
Figure 25: Validation test flow - Ignored Invalidation in DTLB.....	90
Figure 26: Proposed post-silicon validation flow and the experiment setup.....	92
Figure 27: Bug coverage for the proposed method vs. traditional end-of-test checking techniques. ....	93
Figure 28: X-Gene 2 micro-server power domains block diagram. The outlines with dashed lines present the independent power domains of the chip. ....	97
Figure 29: Margins Characterization Framework Layout. ....	99
Figure 30: Design corners [138].....	104
Figure 31: X-Gene 2 characterization results for 4 SPEC CPU2006 benchmarks ( <i>bwaves</i> , <i>dealII</i> , <i>leslie3d</i> , <i>milc</i> ) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region. ..	106
Figure 32: X-Gene 2 characterization results for 4 SPEC CPU2006 benchmarks ( <i>cactusADM</i> , <i>gromacs</i> , <i>mcf</i> , <i>namd</i> ) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region. ....	107
Figure 33: X-Gene 2 characterization results for 2 SPEC CPU2006 benchmarks ( <i>soplex</i> , and <i>zeusmp</i> ) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region. ....	108
Figure 34: $V_{min}$ results at 2.4 GHz for 10 SPEC CPU2006 programs on 3 different X-Gene 2 chips (TTT, TFF, TSS). ....	109
Figure 35: <i>bwaves</i> benchmark severity on TTT chip cores.....	112
Figure 36: <i>cactusADM</i> benchmark severity on TTT chip cores.....	113
Figure 37: <i>dealII</i> benchmark severity on TTT chip cores. ....	113
Figure 38: <i>gromacs</i> benchmark severity on TTT chip cores. ....	114
Figure 39: <i>leslie3d</i> benchmark severity on TTT chip cores.....	114
Figure 40: <i>mcf</i> benchmark severity on TTT chip cores. ....	115
Figure 41: <i>milc</i> benchmark severity on TTT chip cores.....	115
Figure 42: <i>namd</i> benchmark severity on TTT chip cores.....	116
Figure 43: <i>zeusmp</i> benchmark severity on TTT chip cores. ....	116
Figure 44: <i>soplex</i> benchmark severity on TTT chip cores.....	117
Figure 45: (a) Time needed for a complete system-level characterization to reveal the pessimistic margins for one chip. Programs are executed on individual cores (1T) and on all 8 cores concurrently (8T). (b) Safe $V_{min}$ values and their independence on power consumption. ....	119
Figure 46: A 256KB 32-way set associative L2 cache. ....	126
Figure 47: IPC measurements for both micro-viruses (top) and SPEC CPU2006 benchmarks (bottom). ....	129
Figure 48: Power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks. The upper graph shows the power consumption at nominal	

voltage (980 mV). The lower graph shows the power measurements when the microprocessor operates at 920mV, in order to present the energy efficiency when operating below nominal voltage conditions. Both graphs present single-core executions. .... 130

Figure 49: Power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks. The upper graph shows the power consumption at nominal voltage (980 mV). The lower graph shows the power measurements when the microprocessor operates at 920mV, in order to present the energy efficiency when operating below nominal voltage conditions. Both graphs present results when programs are executed in 8 cores concurrently. .... 131

Figure 50: Detailed comparison of  $V_{min}$  between the 12 SPEC CPU2006 benchmarks and micro-viruses for the TSS chip. .... 132

Figure 51: Maximum  $V_{min}$  among 12 SPEC CPU2006 benchmarks and the proposed micro-viruses for TTT and TFF in PMD domain. .... 133

Figure 52: Maximum  $V_{min}$  among 12 SPEC CPU2006 benchmarks and the proposed micro-viruses for TSS in PMD domain (top graph). The bottom graph shows the maximum  $V_{min}$  of 12 SPEC CPU2006 benchmarks and the proposed L3 micro-virus in SoC domain. .... 134

Figure 53: X-Gene 3 block diagram. .... 141

Figure 54: Power and performance management in X-Gene microprocessors - the Collaborative Processor Performance Control. .... 143

Figure 55: 4 running threads in 2 different core allocation configurations: spreaded & clustered. .... 144

Figure 56: The complete  $V_{min}$  characterization results. This graph presents the X-Gene 2 safe  $V_{min}$  points for all benchmarks with 8 threads, 4 spreaded and clustered threads, and 2 clustered threads in 2.4 GHz, 1.2 GHz, and 0.9 GHz clock frequencies. .... 145

Figure 57: The complete  $V_{min}$  characterization results. This graph presents the X-Gene 3 safe  $V_{min}$  points for all benchmarks with 32 threads, 16 spreaded and clustered threads, and 8 clustered threads in 3.0 GHz, and 1.5 GHz clock frequencies. .... 146

Figure 58: Probability of Failure (pfail) in all voltage levels from nominal level down to the levels of complete failure for different frequency, core allocation, and thread scaling options. .... 148

Figure 59: Voltage droop detections for each program in 2 different voltage droop magnitudes. The top graph presents the droop detections in range between 55mV and 65mV, and the bottom graph presents the droop detections in range between 45mV and 55mV. .... 150

Figure 60: Energy of all benchmarks for 2 different core allocations. The benchmarks on the left of the dashed line are CPU-intensive while the ones on the right are memory-intensive. .... 151

Figure 61: Relative performance of all benchmarks. The y-axis presents the ratio of the execution time of one instance of the single-threaded execution divided by the execution time of multiple instances. .... 152

Figure 63: The magnitude of  $V_{min}$  dependence on frequency, core allocation and workloads. .... 153

Figure 62: L3 Cache access rate per 1M cycles for the 25 benchmarks and the 3 threading configurations (32, 16 and 8 threads). .....	153
Figure 64: Energy consumption in Joules for 8, 4, and 2 threading options for 3 different frequencies (2.4GHz, 1.2GHz, 0.9GHz) of X-Gene 2 (top) and 32, 16, and 8 threading options for 2 different frequencies (3GHz, 1.5GHz) of X-Gene 3 (bottom).....	154
Figure 65: Energy Delay Squared Product (ED2P) for 8, 4, and 2 threading options for 3 different frequencies (2.4GHz, 1.2GHz, 0.9GHz) of X-Gene 2 (top) and 32, 16, and 8 threading options for 2 different frequencies (3GHz, 1.5GHz) of X-Gene 3 (bottom)..	155
Figure 66: Placement flow chart. ....	158
Figure 67: Process handling (Fail-Safe Mechanism). ....	159
Figure 68: Average power for the Baseline and the Optimal configurations in X-Gene 2 during 1-hour execution. ....	160
Figure 69: Average power for the Baseline and the Optimal configurations in X-Gene 3 during 1-hour execution. ....	160

## LIST OF TABLES

Table 1: Comparison of virtual memory features in major ISAs.....	60
Table 2: Published ATM-related bugs from official errata sheet. ....	65
Table 3: Bug coverage of the proposed ATM validation method. ....	74
Table 4: Experimental results on the bare-metal setup in the x86-64 model of the enhanced Gem5.....	77
Table 5: Published ATCA-related bugs from official errata sheets. ....	81
Table 6: Bug Scenarios that cause divergence from the correct behavior and sample activation criterion that can unveil the bug.....	82
Table 7: Example Page Table Entries (Instruction and Data Pages and Frames) for Processes P1 and P2.....	85
Table 8: Number of different type of bugs for each phenomenon.....	93
Table 9: Summary of studies on commercial chips. ....	95
Table 10: Basic Characteristics of X-Gene 2.....	98
Table 11: Experimental effect categorization.....	102
Table 12: Example output of the characterization framework.....	103
Table 13: Design corner checks [138]. ....	105
Table 14: Weights used in our experiments.....	112
Table 15: The X-Gene 2 cache specifications. ....	121
Table 16: ARMv8 instructions used in the L11 micro-virus. The right column presents the encoding of each instruction to demonstrate that all cache block bits get flipped.....	124
Table 17: Basic Parameters of X-Gene 2 and X-Gene 3.....	141
Table 18: Benchmarks description. ....	142
Table 19: Frequency scaling in X-Gene microprocessors. ....	143
Table 20: Correlation of voltage droops magnitude with frequency and core allocation ( $V_{min}$ columns concern X-Gene 3).....	157
Table 21: X-Gene 2 results for the 4 configurations. ....	161
Table 22: X-Gene 3 results for the 4 configurations. ....	161



## **PREFACE**

This thesis was conducted in the Department of Informatics and Telecommunications (Computer Architecture Lab) of the National and Kapodistrian University of Athens during the period 2014-2019.





# 1. INTRODUCTION

## 1.1 Transistor Scaling and Microprocessor Evolution

Gordon Moore famously predicted in his 1965 paper that the number of components per chip would continue to increase every year by a factor of two [14]. The aim of Moore's law is to reduce the cost and the power consumption per integrated circuit component. In 1975, Moore updated his initial prediction stating that components per chip would increase by a factor of two every two years, as a result of the combination of component size scaling and the chip area increase [15]. Back in 1965, the industry was producing chips using a minimum feature size of approximately 50mm with a total of approximately 50 components per chip. The leading chips of today use a minimum feature size of about 7nm and incorporate several billion transistors.

In 1974, Robert Dennard and his colleagues described a scaling methodology for metal-oxide-semiconductor-field-effect-transistors (MOSFETs), which would consistently improve the transistor area, performance and power reduction [16]. The methodology involved the scaling of the transistor gate length, gate width, gate oxide thickness, and supply voltage all by the same scale factor, and increasing channel doping by the inverse of the same scale factor. The result would be transistors with a smaller area, a higher drive current (higher performance) and lower parasitic capacitance (lower active power). This method of scaling MOSFET transistors is generally referred to as "classic" or "traditional" scaling ("Dennard scaling") and was used by the industry very successfully up to the generation of 130nm in the early 2000s [17].

The combination of Moore's law and Dennard's scaling methodology has given the microprocessor industry many generations of smaller and faster transistors resulting in higher performance microprocessors (Figure 1). When gate-oxide scaling began to slow down due to increased gate leakage, classical MOSFET scaling techniques were successfully followed until around the generation of 90nm. The limitation posed by gate leakage became so severe that there was virtually no scaling of gate-oxide thickness from the 90nm to the 65nm generation, and for the high-performance logic process, many

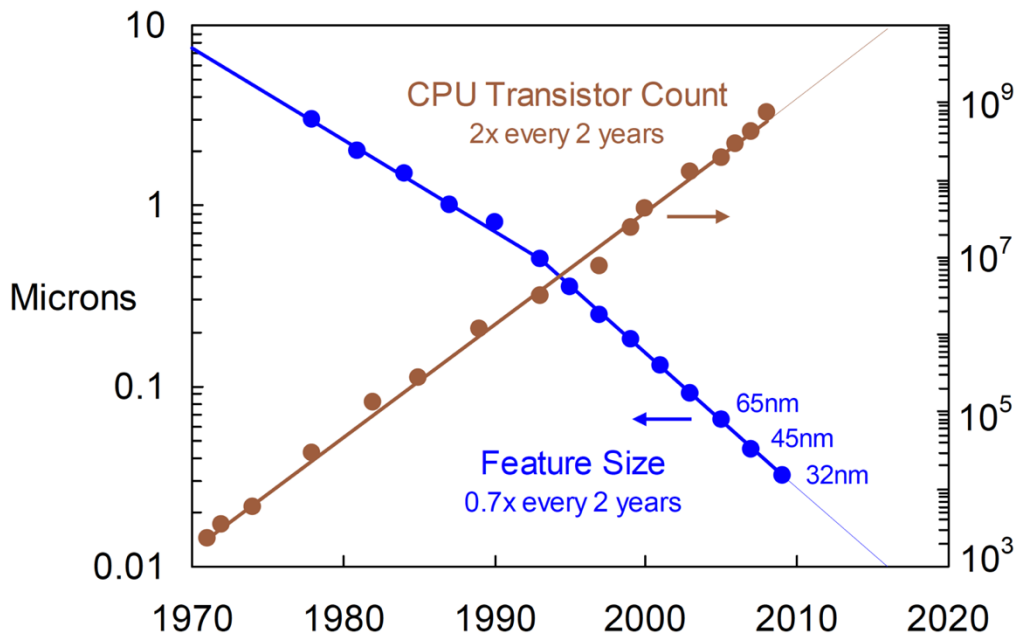


Figure 1: CPU transistor count and feature size trend [18].

companies converged on a  $\text{SiO}_2$  thickness close to 1.2nm. If the thickness of the gate-oxide can no longer be scaled, then critical other MOSFET parameters, such as supply voltage, cannot be further scaled to obtain improved transistor performance [18].

One of the first transistor innovations in the 2000's has been the introduction of strained-silicon technology in 2003 to improve transistor performance for Intel's 90nm microprocessors [19] [20]. The 65nm generation introduced in 2005 further improved these strain techniques to increase the performance of the transistor, although the gate - oxide thickness remained at approximately the same 1.2nm value to prevent increased leakage current [21]. Strained silicon is an example of a revolutionary technology that delivered improved performance without following traditional methods of MOSFET scaling. Moore's law remained valid for some time despite Dennard's scaling end.

Although strained silicon provided valuable performance improvements for generations of 90nm and 65nm, the gate-oxide thickness limit (and corresponding leakage issues) needed to be addressed for subsequent technologies. Intel's 45nm logic technology was the first to introduce high- $\kappa$  dielectric transistors with a metal gate to improve performance and reduce leakage [22] [23]. A dielectric based on hafnium replaced  $\text{SiO}_2$  with a physically thicker gate-oxide that had reduced leakage but a thinner electrical equivalence, which had improved transistor performance. Polysilicon was replaced by special metal-gate materials to further reduce the thickness of electrical oxide to a value of 1.0nm (from 1.2nm). In addition to the performance and leakage benefits of the 45nm high- $\kappa$ /metal-gate (HK+MG) transistors, the decreased electrical thickness also helped to reduce the variability of the transistor's  $V_t$  (threshold voltage); an essential factor in circuits' ability to use scaled devices [24]. These 45nm HK+MG transistors provided an average drive current increase of 30 percent over the previous generation of 65nm. Equivalently, these transistors can reduce sub-threshold ( $I_{\text{off}}$ ) leakage by more than five times.

On the other hand, unlike transistors, interconnections are not getting faster as they scale, so the industry has gradually increased the number of interconnect layers to address this problem. This approach has enabled some layers to use wider / thicker wires for rapid signal propagation, while other layers have a tight pitch for density improvement. Besides, new interconnecting materials were introduced to improve RC delay<sup>1</sup> and current carrying capacity. In the 2010's, copper replaced aluminum as a means of increasing conductivity and improving resistance to electro-migration. In order to reduce wire capacity, a series of dielectrics have been introduced that have both signal speed and active power reduction advantages. Implementing more interconnected layers and improved materials has enabled many generations of scaled interconnections, but interconnections will remain a problem requiring a combination of process, design, and architectural solutions to be overcome.

Intel's 45nm technology uses nine copper interconnect layers, one more than the generation of the previous 65nm. A particular layer to provide low-resistance power routing to minimize voltage droops was introduced in this generation. Intel's 32nm logic technology uses second-generation high- $\kappa$ + metal-gate transistors, fourth-generation strained silicon, nine interconnecting copper layers with low- $\kappa$  dielectrics and minimum pitches scaled up to 70 percent from 45nm [25] [26]. A mixture of increased drive current and reduced gate capacitance increases transistor performance by more than 22 percent

---

<sup>1</sup> The delay in signal speed through the circuit wiring as a result of the resistance (R) and capacitance (C) effects. Resistance is the difficulty an electrical current has in passing through a conducting material, and capacitance is the degree to which an insulating material holds a charge.

compared to 45nm transistors. Over the past several generations, transistor drive currents continued to increase while scaling gate pitch and maintaining constant subthreshold leakage.

Just as transistors evolved more than just scaled dimensions, so did microprocessors evolve. Introduced in 1971, the first microprocessor (Intel 4004) processed 4 bits of data per cycle. It contained 2300 transistors and had a 100kHz clock rate. Microprocessor performance has increased rapidly from that point of departure closely following the predictions of Moore's Law. Scaling technology has made it possible to build more and more complex designs. Integrating multiple pipeline execution engines into microprocessors has enabled microarchitectural innovations to exploit parallelism within a sequential program and allow multiple instructions to be executed in each clock cycle. Speculation was used to get instructions and execute them based on predicting the control flow of the program. Out-of-order and register renaming techniques were used to remove stalls if instructions were not ready for execution. The width of the memory address has increased to 64 bits allowing access to large memories, while integrated multi-level caches have made effective access to memory faster. Multithreading was introduced to enable instructions from independent program threads to efficiently exploit the parallel execution engines of the microprocessor. Microprocessors are now integrating multiple processing cores on a single chip [18].

In addition to enabling advances in microarchitecture, faster transistors have been produced by technology scaling. An essential contributor to microprocessor performance gains was the clock frequency increase. Clock systems evolved from a simple buffer tree to global clock grids driven by a distributed buffer hierarchy. In order to generate multiple clock frequency domains, on-chip PLLs for clock generation were integrated. In order to minimize the skew across the die, active de-skewing schemes using delay lines were used [27]. The combination of more complex microarchitectures and higher operating frequencies has resulted in a primary design constraint for power consumption. To disable idle circuitry and save power, clock systems have evolved to implement clock gating. They also include tuning and debugging circuits, e.g., the Locate Critical Path (LCP) mode, which is based on programmable delay drivers activated in local clock drivers and controlled by configuration registers that allow adjustment of local clock arrival times after the silicon's production [28]. However, power consumption per unit area (power density) has increased significantly in increasing functionality per unit area.

## 1.2 Power Reduction Techniques

Technology scaling has enabled improvements in the three major design optimization objectives: increased performance, lower power consumption, and lower die cost, while system design has focused on bringing more functionality into products at lower cost. While today's microprocessors, are much faster and much more versatile than their predecessors, they also consume much power; in fact, so much power that their power densities and consequent heat generation are rapidly approaching levels comparable to nuclear reactors (Figure 2) [29]. As shown in Figure 2, in 0.5 $\mu$ m technology, microprocessors have surpassed hot-plate power density. The active power scales by ~30-80% (it is not stay constant) for each process generation.

To date, the approach has been to lower voltage with each process generation. But as voltage is lowered, leakage current and energy increase, contributing to higher power. These high-power densities impair the reliability of chips and life expectancy, increase cooling costs, and even raise environmental concerns primarily due to large data centers. Power problems also pose issues for smaller mobile devices with limited battery capacity. While these devices could be implemented using faster microprocessors and larger memories, their battery life would be further diminished. Improvements in microprocessor

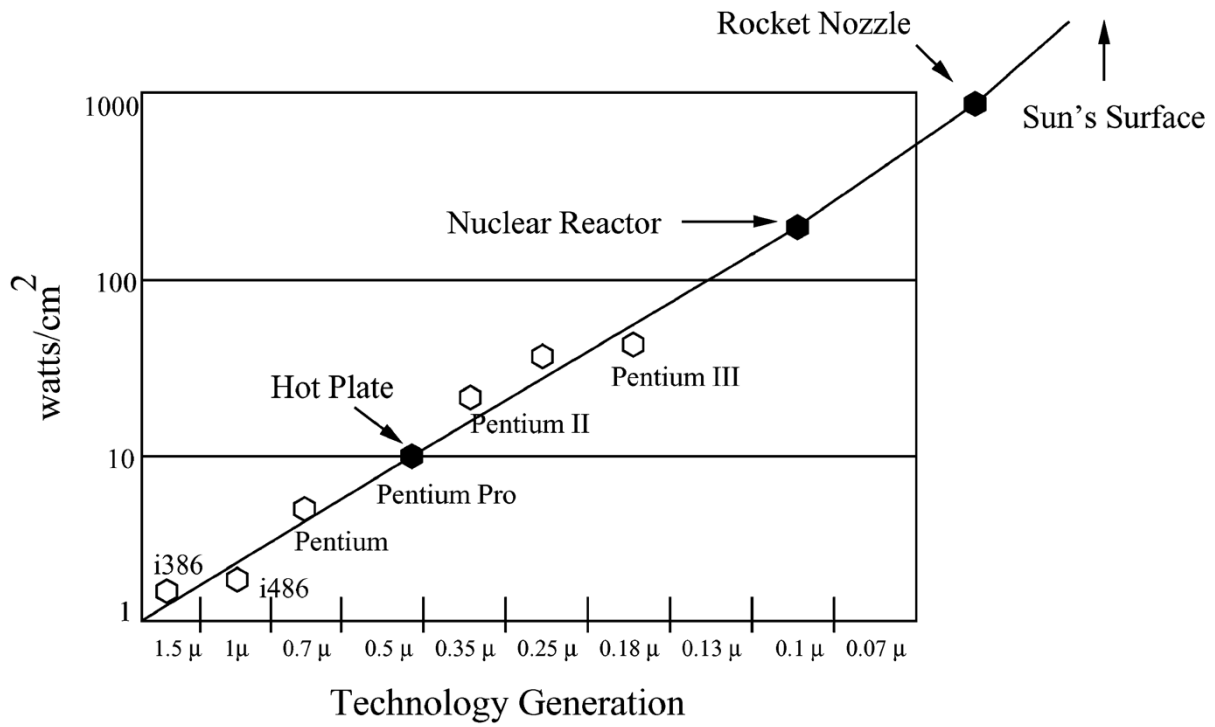


Figure 2: Power densities rising [29].

technology will eventually come to a standstill without cost-effective solutions to the power problem.

Power and energy are commonly defined as the work performed by a system. Energy is the total amount of work performed by a system over some time, whereas power is the rate at which the system performs the work. In formal terms,

$$P = \frac{W}{T} \quad (1)$$

$$E = P * T \quad (2)$$

where  $P$  is power,  $E$  is energy,  $T$  is a specific time interval, and  $W$  is the total work performed in that interval. Energy is measured in joules (J), while power is measured in watts (W) [29].

The concepts: work, power, and energy are used in different contexts differently. *Work* involves activities related to running programs in the context of microprocessors (e.g., subtraction, memory operations, addition), *power* is the rate at which electrical energy is consumed by the microprocessor (or dissipates it as heat) during these activities, and *energy* is the total electrical energy consumed by the microprocessor (or dissipates as heat) over time. This difference between *power* and *energy* is essential because power-reduction techniques do not necessarily reduce energy [29].

For instance, by halving the frequency of the input clock, the power consumed by a microprocessor can be reduced. If the microprocessor, however, takes twice as long to run the same programs, the total energy consumed is the same. Whether power or energy should be reduced depends on the context. Reducing energy is often more critical in data centers because they occupy an area of a few football fields, contain tens of thousands of servers, consume electricity of small cities and utilize expensive cooling mechanisms.

There are two forms of power consumption, *dynamic power consumption* and *static power consumption*. Dynamic power consumption is caused by circuit activity such as input changes in an adder or values in a register. As the following equation shows, the dynamic power ( $P_{dynamic}$ ), depends on four parameters namely, supply voltage ( $V_{dd}$ ), clock frequency ( $f$ ), physical capacitance ( $C$ ) and an activity factor ( $a$ ) that relates to how many  $0 \rightarrow 1$  or  $1 \rightarrow 0$  transitions occur in a chip:

$$P_{dynamic} = aCV^2f \quad (3)$$

An example is the 45nm Intel® Core™ i7 Processor, formerly known as Nehalem (Figure 3). This is a complex system on a chip (SoC) with multiple functional units and multiple interfaces, including four cores, modular L3 cache from 2 to 24MB, integrated memory controller, DDR3 I/O and QPI I/O [30]. There are 11 PLL circuits, 23 master DLL circuits and 5 digital thermal sensors located around the chip providing multiple clocking domains and local control. NMOS sleep transistors are used in the cache to shut off leakage in inactive sub-blocks. They provide a 5x to 10x cache leakage reduction during retention/standby [31]. Power-gate transistors are integrated on the chip to shut off both active and leakage power on cores that are idle. These on-die power gates are enabled by a 45nm process, which uses a 7µm thick top metal layer that provides very low power-distribution impedance combined with ultra-low leakage transistors with low on-resistance.

Nehalem introduces a turbo mode, where core frequency can be boosted in response to workload demands, and thus, dynamically delivering optimal performance and energy efficiency. An integrated Power Control Unit (PCU) incorporates real-time sensors for current/power, voltage and temperature and individually controls settings for each core (Figure 4). An adaptive frequency system adjusts clock frequency upwards during voltage supply spikes and reduces frequency during voltage supply droops, thus allowing the cores to operate at optimal performance and power without excessive guardbanding or adding excessive decoupling capacitance [18] [32].

Several techniques have been proposed over time to reduce the power consumption of the microprocessor chips. These techniques are applied at various levels ranging from

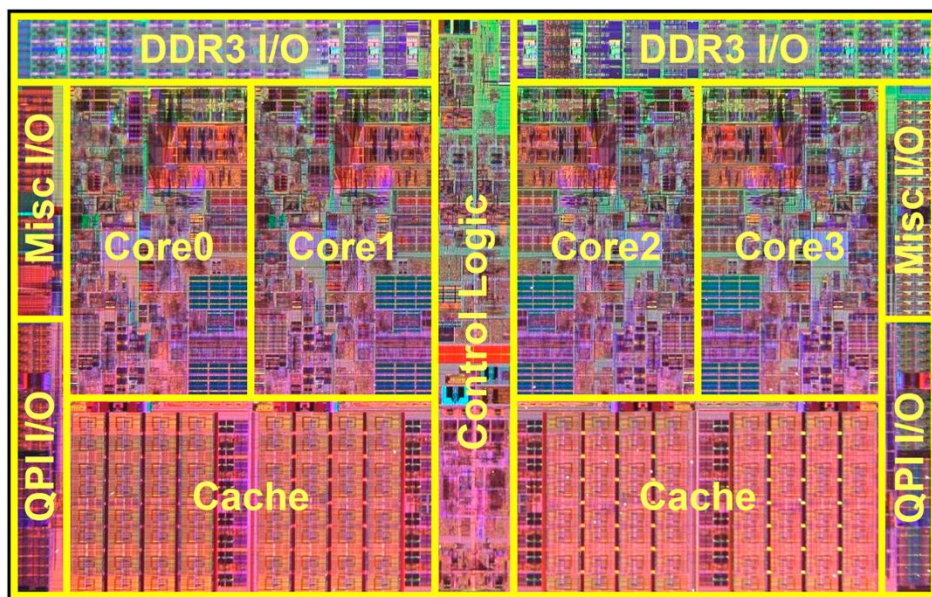
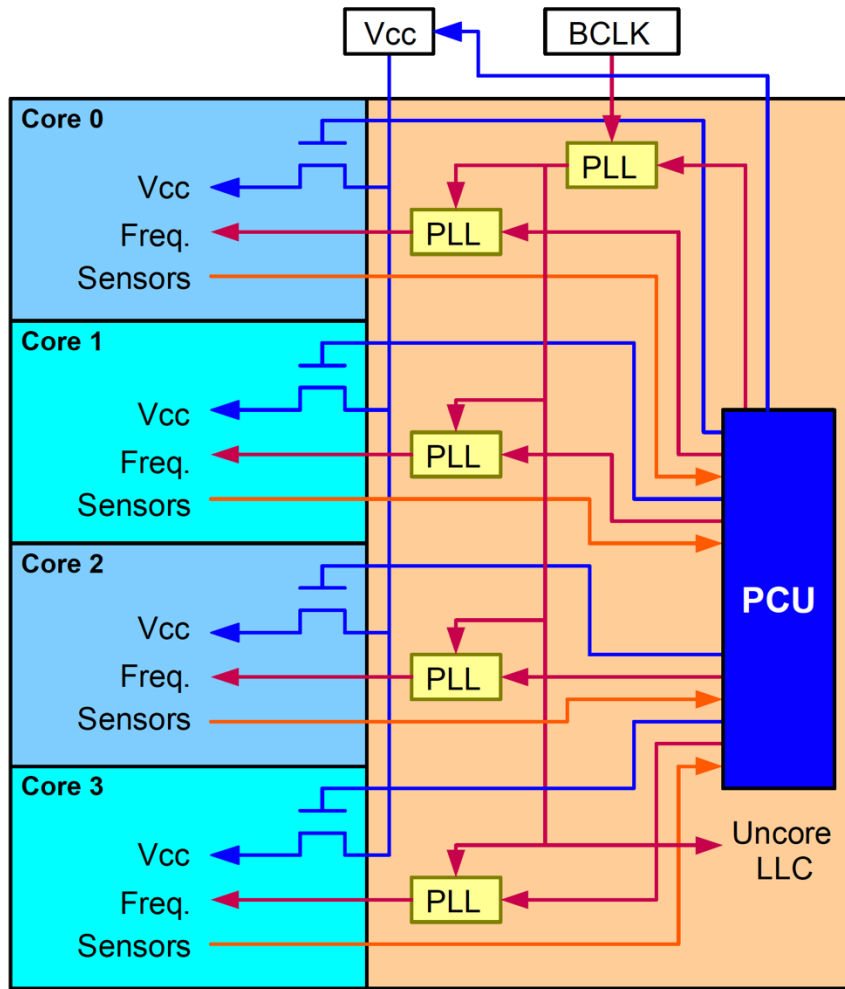


Figure 3: A 45nm Core™ i7 processor (Nehalem) [18].



**Figure 4: Nehalem power control unit (PCU).** The PCU integrates proprietary microcontroller, which shifts control from hardware to embedded firmware. It also contains real-time sensors for voltage, current, power, and temperature [18].

circuits to architectures, architectures to system software, and system software to applications [29].

**Circuit and Logic Level Techniques:** Transistor features and logic design can contribute to reduce the dynamic power consumption by reducing the transistors' width, implementing a different arrangement of transistors in a circuit or rearranging the gates and their input signals. Numerous techniques have been applied in this category, such as transistor sizing [33] [34], transistor reordering [35] [36], technology mapping [37] [38] [39], low-power flip-flops [40] [41] [42] [43] [44], low-power control logic design [45], delay-based dynamic-supply voltage adjustment [46].

**Low-Power Interconnect:** Interconnect heavily affects power consumption since it is the medium of most electrical activity. Efforts to improve chip performance are resulting in smaller chips with more transistors and more densely packed wires carrying larger currents. The wires in a chip often use materials with poor thermal conductivity [47]. The most popular ways of reducing the power consumed in buses contains: Bus Encoding and CrossTalk [48] [49], Low Swing Buses [50], Bus Segmentation [51], Adiabatic Buses [52], Network-On-Chip [53] [54] [55] [56].

**Low-Power Memories and Memory Hierarchies:** There are high-level architectural principles that apply across the spectrum of memories. They attempt to reduce the energy dissipation of memory accesses in two ways, either by reducing the energy dissipated in



a memory accesses, or by reducing the number of memory accesses. Some of the major techniques are: Splitting Memories into Smaller Subsystems [57] [58] [59], Augmenting the Memory Hierarchy with Specialized Cache Structures [60].

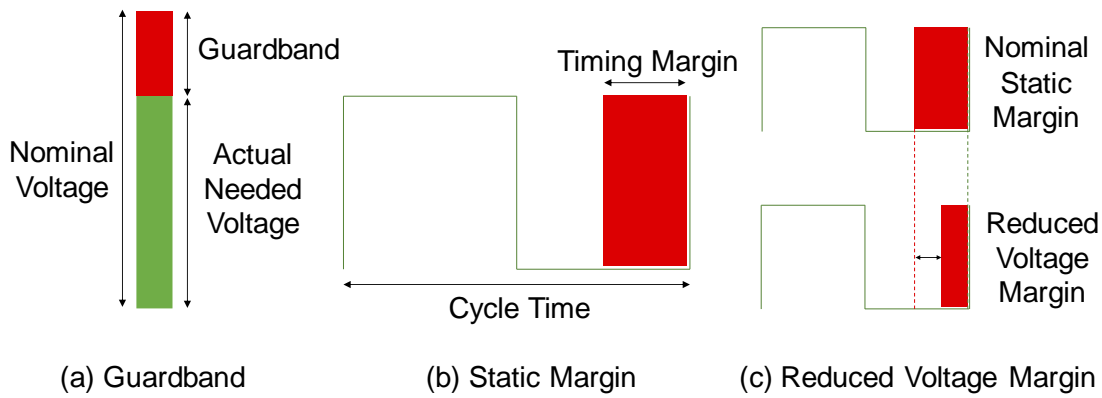
**Low-Power Processor Architecture Adaptations:** Programs exhibit wide variations in behavior. Researchers have been developing hardware structures whose parameters can be adjusted on demand so that one can save energy by activating just the minimum hardware resources needed for the code that is executing. Some of the major techniques are: adaptive caches [61] [62] [63], Adaptive Instruction Queues [64] [65] [66], Algorithms for Reconfiguring Multiple Structures [67] [68] [69] [70] [71] [72] [73] [74] [75] [76].

**Dynamic Voltage and Frequency Scaling:** Dynamic voltage and frequency scaling (DVS) addresses the problem of how to modulate a microprocessor's clock frequency and supply voltage in lockstep as programs execute. The premise is that a microprocessor's workloads vary, and when the microprocessor has less work, it can be slowed down without affecting performance adversely [77]. There are multiple technologies supported by each microprocessor architecture to optimize the power consumption, like the x86's P-states (performance states - optimization of the voltage and clock frequency during operation). More specifically, during the execution of code, the operating system and CPU can optimize power consumption through different P-states. Depending on the requirements, a CPU is operated at different frequencies. P0 is the highest frequency (with the highest voltage).

**Cross-Layer Adaptations:** Power consumption depends on decisions spanning the entire stack from transistors to applications. To this end, there are several holistic approaches that integrate information from multiple levels (e.g., compiler, OS, hardware) into power management decisions. There are already a number of systems being developed to explore cross-layer adaptations, e.g., [78] [79] [80].

### 1.3 Supply Voltage Scaling

Reducing supply voltage is one of the most efficient techniques to reduce the dynamic power consumption of the microprocessor, because dynamic power is quadratic in voltage (as Eq. 3 shows). However, supply voltage scaling increases subthreshold leakage currents, increases leakage power, and also poses numerous circuit design challenges. Process variations and temperature parameters (dynamic variations), caused by different workload interactions are also major factors that affect microprocessor's energy efficiency. Furthermore, during microprocessor chip fabrication, process variations can affect transistor dimensions (length, width, oxide thickness etc. [81]) which have direct impact on the threshold voltage of a MOS device [82]. As technology scales further down, the percentage of these variations compared to the overall transistor size increases and raises major concerns for designers, who aim to improve energy efficiency. This variation is classified as static variation and remains constant after fabrication. Both static and dynamic variations lead microprocessor architects to apply conservative guardbands (operating voltage and frequency settings), as shown in Figure 5a to avoid timing failures and guarantee correct operation, even in the worst-case conditions excited by unknown workloads, environmental conditions, and aging [5] [83]. The guardband results in faster circuit operation under typical workloads than required at the target frequency, resulting in additional cycle time, as shown in Figure 5b. In case of a timing emergency caused by voltage droops, the extra margin prevents timing violations and failures by tolerating circuit slowdown. While static guardbanding ensures robust execution, it tends to be severely overestimated as timing emergencies rarely occur, making it less energy-efficient [84].



**Figure 5. Voltage guardband ensures reliability by inserting an extra timing margin. Reduced voltage margins improve total system efficiency without affecting the reliability of the microprocessor.**

These pessimistic guardbands impede power consumption and performance, and block the savings that can be derived by reducing the supply voltage and increasing the operation frequency, respectively, when conditions permit. Several microarchitectural techniques have been proposed that eliminate a subset of these guardbands (Figure 5c) for efficiency gains over and above what is dictated by the design conservative guardbands. However, all of these techniques are associated with significant design, test and measurement overheads that limit its application in the general case. For instance, in the Razor technique [46], support for timing-error detection and correction has to be explicitly designed into the processor microarchitecture which has significant verification overheads and circuits costs itself. Similarly, in adaptive-clocking approaches [84], extensive test and verification effort is required until the microprocessor is released to the market. Ensuring the eventual success of these techniques requires a deep understanding of dynamic margins and their manifestation during normal code execution.

Revealing and harnessing the pessimistic design-time voltage margins offers a significant opportunity for energy-efficient computing in modern multicore CPUs. The full energy savings potential can be exposed only when accurate core-to-core, chip-to-chip, and workload-to-workload voltage scaling variation is measured. When all these levels of variation are identified, system software can effectively allocate hardware resources to software tasks to provide the best energy efficiency.

As operating frequency and integration density increase, the total chip power dissipation increases. This is evident from the fact that due to the demand for increased functionality on a single chip, more and more transistors are being packed on a single die and hence, the switching frequency increases in every technology generation. However, by developing aggressive and sophisticated mechanisms to boost performance and enhance the energy efficiency in conjunction with the decrease of the size of transistors, microprocessors have become extremely complex systems, making the microprocessor verification and manufacturing testing a major challenge for the semiconductor industry.

#### 1.4 Microprocessors' Dependability Life-Cycle

As the computer architecture field evolves and transistor size shrinks, computer architects are continuously facing more challenges to ensure that the products they deliver meet high-quality standards. Dependability is defined as the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers [85] [86]. A product that fails to meet its reliability requirements could result in a manufacturer's financial loss or even endanger human life if the product is used in safety-critical computing areas such as automotive or aerospace.



Manufacturers, therefore, choose to spend extra effort, time, budget and chip area to ensure that the delivered products are operating correctly. To meet high-dependability requirements, manufacturers apply a sequence of verification tasks throughout the entire life-cycle of the microprocessor to ensure the correct functionality of the microprocessor chips from the various types of errors that may occur after the products are released to the market. These processor life-cycle verification tasks are illustrated in Figure 6 and summarized below:

- Reliability Estimation [221]:** Computer architects evaluate the expected level of reliability of a microprocessor for any fault model (e.g., transient, permanent or intermittent faults) during this task. A microprocessor's early reliability assessment is vital for two reasons. First, to determine the parameters of microarchitectural design that could influence the product's vulnerability in order to meet the defined reliability requirements of the design planning phase. For example, besides its performance, the size and design specifics of the hardware structures can influence the processor's vulnerability. Secondly, to determine the design decisions related to the mechanisms required in the field for error detection and recovery/repair in the presence of faults. These mechanisms may impose significant area, power and performance overheads. Thus, inaccurate assessment of the reliability during the early design phases could easily make the cost of protection unaffordable or lead to an expensive, over-designed chips with unnecessary protection mechanisms. For example, typical memory error protection and detection techniques can have a cost that ranges from 1% to 125% in terms of extra memory capacity for the purposes of protection, depending on the complexity of the protection mechanism [87]. Moreover, detection and protection mechanisms against any fault model must be decided as early as possible in order to avoid costly redesign cycles for late integration of such mechanisms.
- Pre-Silicon Verification [222]:** This verification task is based primarily on simulation using RTL model tests that are compared to golden architectural model

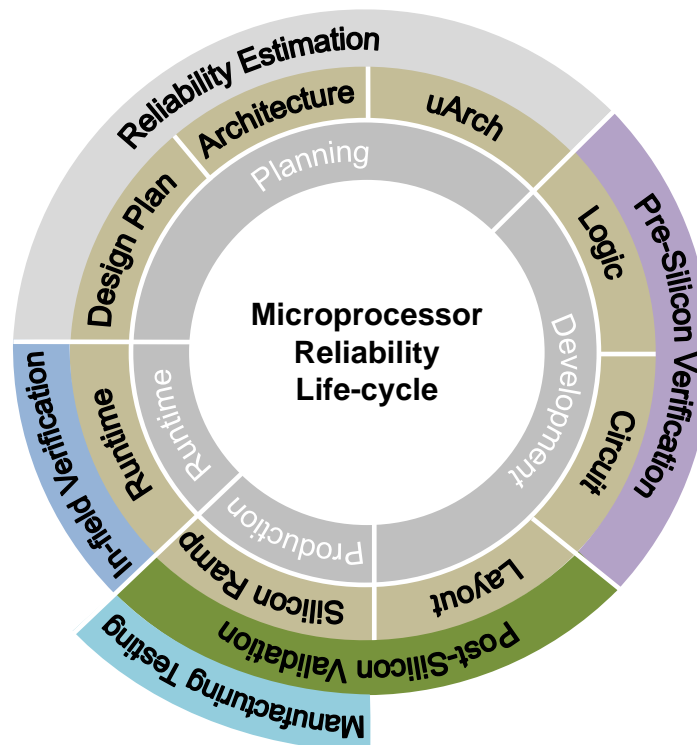


Figure 6: Processor dependability life-cycle.

tests. During this stage, any discrepancies and design bug indicators are identified and fixed. Besides simulation, pre-silicon verification engineers use formal methods based on mathematical evidence to ensure that certain types of design errors are not present. Unfortunately, when they target complex RTL models, these methods lack scalability.

- **Post-Silicon Validation [222]:** This is the design verification task in which the validation and debugging of a new microprocessor design on its first silicon prototype chips take place. The goal of this task is to detect any “difficult” bugs that escaped from the pre-silicon verification task (which typically “catches” less difficult bugs) and to ensure that a chip’s actual silicon implementation matches its specification as was defined in the early design planning phase. When a bug is detected in this stage, the RTL model (or even earlier model than RTL) has to be modified to fix the bug, and the manufacturing process of some prototypes must start again. The re-designing process of the prototypes cannot be repeated too many times (basically due to increased manufacturing costs), but it is limited to a period of couple of months.
- **Manufacturing Testing:** This is the next task after the post-silicon validation task. During manufacturing testing important coverage metrics are used like stuck-at coverage, transition fault coverage, and N-detect coverage. When for each single manufactured chip, a sufficient level of coverage is obtained the chip is ready for market release. Manufacturing testing techniques aim to maximize the coverage of faults while minimizing the time and the resources cost of testing (expensive “big iron” testers are employed in high-volume manufacturing testing – HVM). Traditional manufacturing test approaches are classified into functional and structural test approaches [88]. Functional approaches such as Software-Based Self-Testing (SBST) use on-chip programmable resources to apply the test at - speed and collect memory test responses to make the final pass/fail decision. On the contrary, structural test approaches (such as scan-based testing) use circuit structure knowledge and the corresponding fault model to generate test patterns. Structural testing usually places the design in a specific self-test mode and can result in excessive power consumption and over-testing; hence, the yield loss of structural approaches is higher than the loss of functional methods.
- **In-field Verification:** Protection mechanisms are implemented on microprocessor chips to ensure their functionality after being released on the market and to collect essential information from exposed bugs during pre- or post-silicon validation. The designers are not interacting with the product at this stage; therefore, they should have carefully implemented effective protection mechanisms against the effects of aging and wear-out, failures that may be the result of manufacturing defects that escaped manufacturing testing and also process variations and failures resulting from environmental conditions such as cosmic rays, alpha particles, and electromagnetic interference. Debug and diagnosis mechanisms have also been implemented so that even if a bug or defect cannot be fixed, the designer can understand and collect essential information for future microprocessor releases. Dual- and triple- modular redundancy are some essential protection techniques, but come at very high costs. Also, parity and Error Correction Code (ECC) techniques are necessary to protect buses, memories or other array-based structures of the microprocessor. Finally, there are many proposed techniques to protect SRAM caches [89], pipeline flip-flops and combinational logic [90].

## 1.5 Failures During Microprocessor's Life-Cycle

There are different types of failures that can be introduced during the lifetime of a microprocessor. The distribution and the nature of these failures during the processor dependability life-cycle are presented in Figure 7 and summarized below:

- **Design bugs:** This category contains all the logic, electrical and process-related bugs [91] [92] that are introduced during the design planning and the development phase. The common sources of these bugs are [93]: (i) the limited throughput of the pre-silicon (simulation-based) verification techniques that cannot keep pace with the complexity and the large amount of code that is used to design the modern microprocessors, (ii) the synthesis tools that may impede the accuracy of the synthesized design leading to functional inaccuracies between the intended and the developed design, (iii) the inaccuracies that are created during the place and route process may be leading to some violations of the design specifications during the layout process, and (iv) the combination of process variations and smaller design margins that prevents microprocessors to work at the intended frequency and voltage levels.
- **Manufacturing defects:** This category contains all the manufacturing-related defects that are introduced in the design during the high-volume manufacturing (HVM) phase. These inaccuracies can be the result of optical proximity effects, and processing material defects. Moreover, as the gate oxides have become so thin, transistors functionality can be easily affected by the variations of the current. Most of these defects are detected by dedicated machines (testers) with a very time-consuming procedure; but still some untested defects escape to the field [94].
- **In-field errors:** This category contains all the failures that can be manifested after the chip is released to the market. These malfunctions can be: (i) transient errors that can be created by Single Event Upsets (SEUs), which potentially corrupt the computational logic and the state bits, (ii) intermittent and permanent errors that

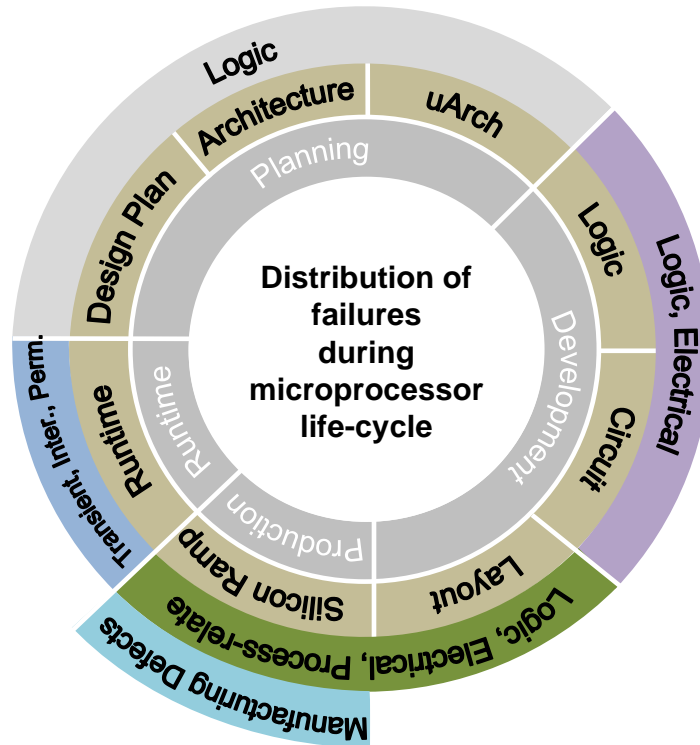


Figure 7: Distribution of failures during microprocessor's life-cycle.

can come from material aging and wear-out effects, or they could have also escaped from the high-volume production testing, and (iii) process variation defects that make microprocessors chips that are designed to be identical to present divergences in terms of performance and power consumption.

For the manufacturers, the detection of any kind of malfunction is of great importance to take place as soon as possible during the life-cycle. The reason is that the cost of product re-design and fix of the detected bug or the addition of any protection mechanism increases significantly throughout the microprocessor life-cycle. Figure 8 illustrates the relative cost of finding bugs for all the phases of the microprocessor's lifetime [95]. The cost to re-start the design planning phase when the manufacturer detects a bug before the start of the layout process ranges to hundreds of dollars, while the cost explodes to more than tens of million dollars when the detection of the bug takes place after the massive release of the product to the market.

## 1.6 Contribution of This Thesis

Microprocessors integrate various types of cores and functional units and are highly adaptive for dynamically optimizing the peak performance, power efficiency, and idle power consumption. Today microprocessors are complex, heterogeneous machines that contain different cores for different types of workloads. This complexity and heterogeneity bring forward the difficult task of design verification and validation. Even for most established microprocessor vendors, the task of verifying a modern microprocessor and ensuring correct operation is increasingly challenging [96]. Always trying to deliver higher performance but also energy efficiency to end-users, manufacturers are forced to gradually design more complex circuits and employ very large verification teams to eliminate all design bugs (if possible) in a timely manner.

The contributions of this thesis belong to two important areas: (a) Post-Silicon Validation for the Address Translation Mechanisms of Modern Microprocessors and (b) Energy Efficiency for Multicore CPUs by Harnessing Pessimistic Voltage Margins. In particular:

**Post-Silicon Validation for the Address Translation Mechanisms of Modern Microprocessors:** Post-silicon validation aims to ensure that the first “real” thing (not models of the design), i.e., the prototype chips, conform to microprocessor design specifications. It allows detection of rare functional design bugs, but also electrical bugs

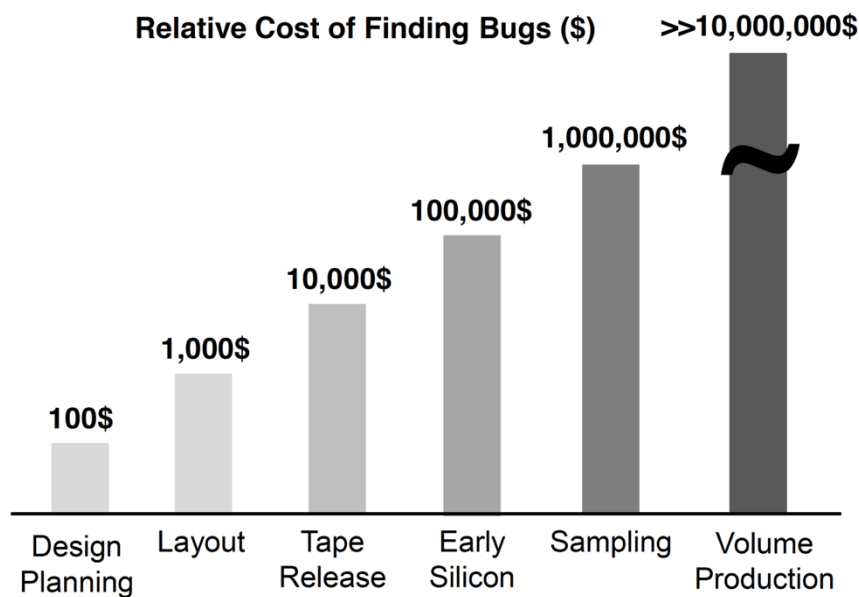


Figure 8: Relative cost of finding bugs throughout microprocessor life-cycle [95].

that manifest themselves only under certain operating conditions, such as thermal effects or process variations [1]. Due to the very high program execution throughput of post-silicon validation (at the speed of the actual prototype chip), the design verification teams attempt to execute as many test programs as possible (such as automatically generated random and directed random test programs). This way, they can obtain the most extensive possible validation coverage (as many potential validation scenarios as possible are adapted) before massive chip production to detect any anomalous behavior [97] [98].

The major downside, however, of random test programs is the difficulty to obtain the expected correct results (the correct output of these programs is unknown because they are randomly generated), which are required to determine the correctness of the output of the prototype chips being validated. To meet this requirement, validation process mainly resorts either to multi-pass consistency end-of-test checking methods (each test-case is executed multiple times ('passes') and the end-of-test values of some system resources (e.g., memory, registers) are compared for consistency) [98], or to golden responses generated by architectural simulators.

However, the throughput difference between native chip execution and simulation (at different levels) is between 3 and 6 orders of magnitude [99]; for example, about 200 billion cycles were simulated in many months for Pentium 4 processor's validation, but instead, in the actual 1 GHz clock processor the execution completed in 3 minutes [91]. Therefore, architectural simulators demand powerful server farms to generate exhaustive simulation traces in parallel, to get the known-correct outcome of test programs [100]. Moreover, both approaches suffer from the same limitation: by detecting a mismatch only at the end of the validation test (after many thousands or millions of executed instructions), debug engineers devote excessive effort to identify the root cause of the bugs by examining long execution traces back in simulators [98] [101].

With the galloping adoption of virtualization today (and the complexity its support adds to ISAs and microarchitectures), the performance and correctness of the address translation mechanisms (ATMs) get more critical. The ATM of a modern microprocessor includes complex hardware structures and can be a predominant source of severe escaped bugs which are, however, very hard to detect as recent reports have shown [102] [103] [104] [105]. Unlike other hardware components (functional blocks, registers, memory subsystem, peripheral control units, etc.) the output of the ATM of a microprocessor (i.e., the physical address) is not observable to any program or architectural visible locations (e.g., architectural register). Because of the "hidden" operation of a microprocessor's address translation subsystem, observability and long bug detection latencies are critical obstacles for its post-silicon validation and debug. Moreover, the address translation process involves several stages and several different hardware structures (i.e., Translation Lookaside Buffers - TLBs), aiming to improve the total microprocessor's performance, and bugs in each of these blocks may lead to wrong address translations, and thus, to unpredictable system behaviors.

To this end, in this thesis we present two contributions on the post-silicon validation of the address translation mechanisms of modern microprocessors. More specifically, in [2] and [3] we present

- a self-checking ISA-independent post-silicon validation method, which accelerates the bug detection process in the address translation mechanisms of modern microprocessors, and
- a comprehensive set of address translation mechanism bug models, which classify the effects of both functional and electrical bugs in the hardware structures employed in address translation.

The method is easily applied to any ISA and we demonstrate that it reduces the bug detection latency by 5 orders of magnitude compared to traditional end-of-test checking techniques (i.e., architectural simulators) by fully resembling a post-silicon validation bare-metal setup. We contribute to the most critical part of functional post-silicon validation, the one based on bare-metal-infrastructure. During functional post-silicon validation, only random-generated test programs and legacy tests are applied, not normal applications. Other post-silicon validation phases focus on full-system configuration with an operating system layer and application software on top of it. We also make an important enhancement of the Gem5 simulator [120] by integrating recently proposed MMU caches [121] [122] to emulate our validation method in a complete modern microprocessor design and demonstrate its effectiveness.

The second contribution of this thesis, which is presented in [4], further enhances the post-silicon validation phase of the ATM by presenting another novel self-checking validation method that unveils and detects rare bug scenarios in Address Translation Caching Arrays (ATCA). ATCAs are among the most important structures for microprocessor functionality and performance and escaped bugs in these arrays can lead to unpredictable system behaviors in the field. Using a comprehensive experimental study, we first present and analyze rare bug scenarios and demonstrate the reason why they are difficult to detect. Our goal is to unveil and detect difficult bugs in ATCAs. Even if bugs manifest themselves by executing traditional validation tests, detecting them is unlikely due to the high possibility of masking during the execution of a traditional validation test. Another reason is that there are bugs in ATCAs that may affect only the performance of the microprocessor and not its functionality. For these reasons, we propose a novel method that guarantees detection of such difficult bugs in the silicon prototype. Our validation method is self-checking (like the previous one) and does not require any hardware instrumentation. We demonstrate that, unlike traditional end-of-test checking techniques, this method effectively detects all the injected bugs we created, by using common use-cases of a real hardware prototype.

Both methods make debugging and diagnosis of a bug easier and also significantly reduce the bug detection latency during the post-silicon validation phase.

### **Energy Efficiency for Multicore CPUs by Harnessing Pessimistic Voltage Margins:**

The third contribution of this thesis is a system-level voltage scaling characterization study for ARMv8-based multicore CPUs manufactured in 28nm ([10] [11]). The primary targets of this study are

1. to identify the pessimistic voltage guardbands (the increased voltage margins set by the manufacturer) of each individual microprocessor core by exposing their safe  $V_{min}$ . Safe  $V_{min}$  is defined as the minimal working voltage of the microprocessor for any workload or operating condition at a specific clock frequency, and
2. to characterize and analyze any abnormal behavior that occur in voltage levels below the safe  $V_{min}$ .

The study's backbone is a fully automated system-level framework built around Applied Micro's (APM) X-Gene 2 micro-server. The automated infrastructure aims to increase the throughput of massive undervolting campaigns that require multiple benchmarks execution at several voltage supply levels of all individual cores. The characterization process requires minimal human intervention and records all possible abnormalities due to undervolting: silent data corruptions (SDC, e.g., program output mismatches without any hardware error notification), corrected errors, uncorrected (but detected) errors (provided by Linux EDAC driver [136]), as well as application and system crashes [10].

Towards the formalization of the any potential behavior in undervolting conditions we also present a simple consolidated function; the *Severity function*. Severity function aggregates the effects of reduced voltage operation in the cores of a multicore CPU by assigning values to the different abnormal observations. The lower the voltage level, the higher the value of the severity function. The severity function assists an undervolting classification of the cores of a CPU chip for a given benchmark: different core, benchmark and voltage values lead to different severity patterns, some with an abrupt increase to the severity (e.g., the benchmark keeps executing correctly until a voltage level at which the system crashes), while others have a “smooth” severity increase while voltage is reduced (the system remains responsive throughout a range of voltage values but it generates ECC errors or produces SDCs). The fine-grained analysis of the behavior of the machine using the severity function can assist energy efficiency decisions for task-to-core allocation by the system software.

Characterization campaigns (like the previous one) with many different benchmarks and for many different microprocessor chips are very time-consuming. The accurate identification of the voltage under-scaling limits in a real multicore system requires massive execution of a large number of real workloads in all the cores of the chip (and all different chips of a system), for different voltage and frequency values. For instance, to identify the  $V_{min}$  of each one of the eight cores of the Applied Micro’s (APM) X-Gene 2 microprocessor, we used the SPEC CPU2006 benchmarks and repeated each experiment 10 times<sup>2</sup> starting from the nominal voltage value (980mV) until their crash voltage value (~880mV). These experiments required about 2 months for a complete characterization for all the cores of one microprocessor chip.

To accelerate the characterization process, we introduce the development of dedicated programs (diagnostic micro-viruses), which are presented in [12] and is the fourth contribution of this thesis. The micro-viruses aim to stress the most fundamental hardware components of the microprocessor aiming to provide quickly the safe  $V_{min}$ . With our diagnostic micro-viruses, we effectively stress (individually or simultaneously) all the main components of the chip:

- a. the caches (the L1 data and instruction caches, the unified L2 caches and the last level L3 cache of the chips) and
- b. the two main functional components of the pipeline (the ALU and the FPU).

These diagnostic micro-viruses are executed in very short time (~3 days for the entire massive characterization campaign for each individual core of one microprocessor chip) compared to normal benchmarks, such as those of the SPEC CPU2006 suite, which need 2 months for a detailed characterization of the same microprocessor chip. The micro-viruses’ purpose is to reveal the variation of the safe voltage margins across cores of the multicore chip and also to contribute to diagnosis by exposing and classifying the abnormal behaviour of each CPU unit (silent data corruptions, bit-cell errors, and timing failures).

The fifth contribution of this thesis, which presented in [13], is to complete the voltage margins characterization for multicore executions and for different clock frequencies. In that part of the thesis, we also present a new software-based scheme for server-grade

---

<sup>2</sup> System-level studies in actual microprocessor chips present a non-deterministic behavior among repeated experiments. To this end, we repeat each and every experiment multiple times to eliminate the divergencies between the same executions and produce safe results.

machines, which provides large energy savings while maintaining high performance levels. Particularly, in this part of the thesis:

- We expose the pessimistic voltage guardbands of two state-of-the-art ARMv8 microprocessors (manufactured in 28nm and 16nm – the X-Gene 2 and X-Gene 3, respectively) to identify the safe  $V_{min}$  points of the CPU chips in multicore executions. We show that as the number of active threads increases, core-to-core and workload-to-workload variability has a minimal impact on  $V_{min}$ .
- We present measurements on the correlation of the safe  $V_{min}$  to the voltage droop magnitude, and show that in multicore executions the emergency voltage droop events occur regardless of the workload. However, for executions in a single or very few cores, core-to-core and workload-to-workload variability exist to a larger extend.
- We perform an extensive study to identify and analyze the tradeoffs between energy and performance at different voltage and frequency combinations, as well as at different thread scaling and core allocation configurations. Our analysis reveals that depending on the coarse-grain characteristics of a program and the number of active threads, there is an optimal combination of voltage, frequency and core allocation for better energy efficiency.
- We also developed a simple online monitoring daemon which monitors the running processes on the system and guides the Linux scheduler to take the appropriate decisions regarding: (a) the core(s) to which a new process should be assigned, and (b) when one or more running processes should be migrated to other cores. At the same time, the daemon dynamically adjusts the V/F settings according to the optimal policies.
- Finally, we evaluate the optimal energy efficient scheme by running the monitoring daemon in a realistic scenario of a server's operation, which (a) randomly selects the issued programs, (b) dynamically migrates the running processes on the system, and (c) dynamically adjusts the voltage and frequency settings for lower energy consumption. We report several comparisons among different configurations to present a detailed evaluation of the optimal scheme, and show that it can achieve on average 25.2% energy savings on X-Gene 2, and 22.3% energy savings on X-Gene 3, with a minimal performance penalty of 3.2% on X-Gene 2 and 2.5% on X-Gene 3 compared to the default voltage and frequency microprocessor's conditions.

## 1.7 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 presents the contributions of this thesis regarding the post-silicon validation of the address translation mechanisms of modern microprocessors. It is divided into two parts: the first part presents a novel method for accelerating the post-silicon validation of the address translation mechanisms, and the second part presents a complementary method aims to detect difficult-to-find bugs in address translation caching arrays.

Chapter 3 and Chapter 4 present the contributions of this thesis regarding the energy efficiency of modern microprocessors. Chapter 3 is divided into two parts: the first part presents an extensive characterization study of three different ARMv8-compliant microprocessor chips in order to expose the pessimistic voltage margins for single-core executions. It also discusses ways that harness the revealed pessimistic voltage margins for increasing the microprocessors' energy efficiency. The second part presents the development of diagnostic micro-viruses, which aim to accelerate the time-consuming



characterization phase. Micro-viruses can successfully accelerate the characterization of pessimistic voltage margins.

Chapter 4 presents essential observations of multicore executions regarding the voltage margins. Based on these observations, in this chapter we present a software-based solution for dynamically adjusting the voltage and frequency levels of the microprocessor, and guiding the Linux scheduler to make optimal energy-efficient decisions.

Finally, Chapter 5 presents the conclusions of this thesis and discusses possible directions for future work.



## 2. Post-Silicon Validation of the Address Translation Mechanisms

### 2.1 Address Translation Mechanisms

#### 2.1.1 Overview

In a modern computing system with virtual memory and virtualization support, the role of address translation is crucial for the correctness and the performance of programs execution. The CPU-memory performance gap is successfully bridged by multi-level caches hierarchies, but the need for frequent virtual-to-physical addresses translation may become a serious performance bottleneck. Every running process generates multiple accesses to its virtual address space (including references to code, stack, and heap segments). If the translation of the virtual addresses to the physical memory space of the system is not fast enough, the execution throughput of the system can dramatically decline. The address translation mechanisms (ATMs) of modern microprocessors

- map the virtual pages (VPs) of software processes to the physical frames (PFs) of the system memory,
- keep track of the access permission rights for each page (user vs. kernel pages, etc.), and
- record the pages' status (accessed, modified, etc.).

Altogether, the mapping, permissions and status information about the virtual pages and the physical frames of the system are stored in page tables (PTs), which can have different organizations and contents in different microprocessors. Every page table contains Page Table Entries (PTEs) for each virtual-to-physical mapping, as well as other attributes (read/write, user/kernel, etc.), which uniquely characterize each frame in the physical memory space. Page tables are stored in the main memory and secondary storage, so when an address translation needs to access them the latency is very high (several hundreds of clock cycles).

Caching of address translations is a very effective way to reduce the very high access latency of PTEs. All microprocessors realize address translation caching through associative Translation Lookaside Buffers (TLBs – which store the most recent VP to PF mappings as well as access permissions and page reference status information. TLBs are accessed through a Virtual Page Number (VPN). On a TLB hit, the translation is promptly available and execution continues. On a TLB miss, the PT is “walked” and the translation is located. When located, the translation is stored in the TLB for future use.

Some processor vendors (e.g., Intel and AMD) design structures that not only cache PTEs from the first level of multilevel page tables (in the TLB), but also cache entries from higher levels of the tree in small per-core Memory Management Unit (MMU) caches [122]. MMU caches are accessed on TLB misses (details in subsection 2.1.3); MMU cache hits enable skipping multiple memory references in the page table walk (reducing the entire walk to just a single memory reference in the best case) [121]. The primary goal of any caching in a microprocessor ATM subsystem is to keep the address translation latency off the critical path of memory access.

Figure 9 shows a high-level diagram for the address translation process of any microprocessor architecture:

- The CPU core generates a virtual address (VA) ①, which consists of a Virtual Page Number (VPN) ② and an Offset ③ (the Offset shows the position of the word in both the virtual page and the physical frame).

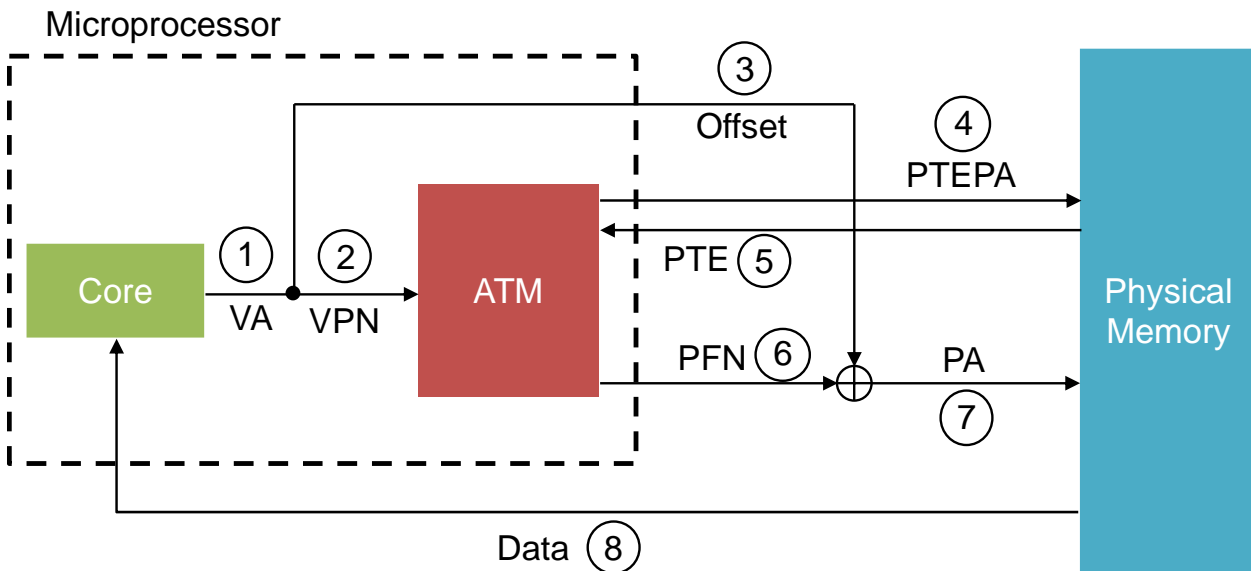


Figure 9: Address translation process overview.

- The ATM calculates the Page Table Entry Physical Address (PTEPA) ④ and fetches the corresponding PTE from the PT in memory ⑤ (this happens when the TLB of the ATM does not already contain the PTE).
- The ATM uses the Physical Frame Number (PFN) ⑥ that the PTE contains along with the same Offset ③ to form the Physical Address (PA) ⑦ and access the memory word ⑧.
- When the ATM's internal TLB already contains the VPN to PFN mapping, steps ④ and ⑤ are skipped and the PA is immediately formed using the TLB contents and the Offset.

### 2.1.2 ATM Organizations

We briefly discuss the ATM hardware specifics of three major microprocessors ISAs: POWER8, x86-64, ARMv8-A. Table 1 summarizes the virtual memory features of the three ISAs. Although the high-level operation of the address translation in all three ISAs is the one that Figure 9 outlines, there are several important differences among them. The purpose of this subsection is twofold: (a) to demonstrate the structure and complexity

Table 1: Comparison of virtual memory features in major ISAs.

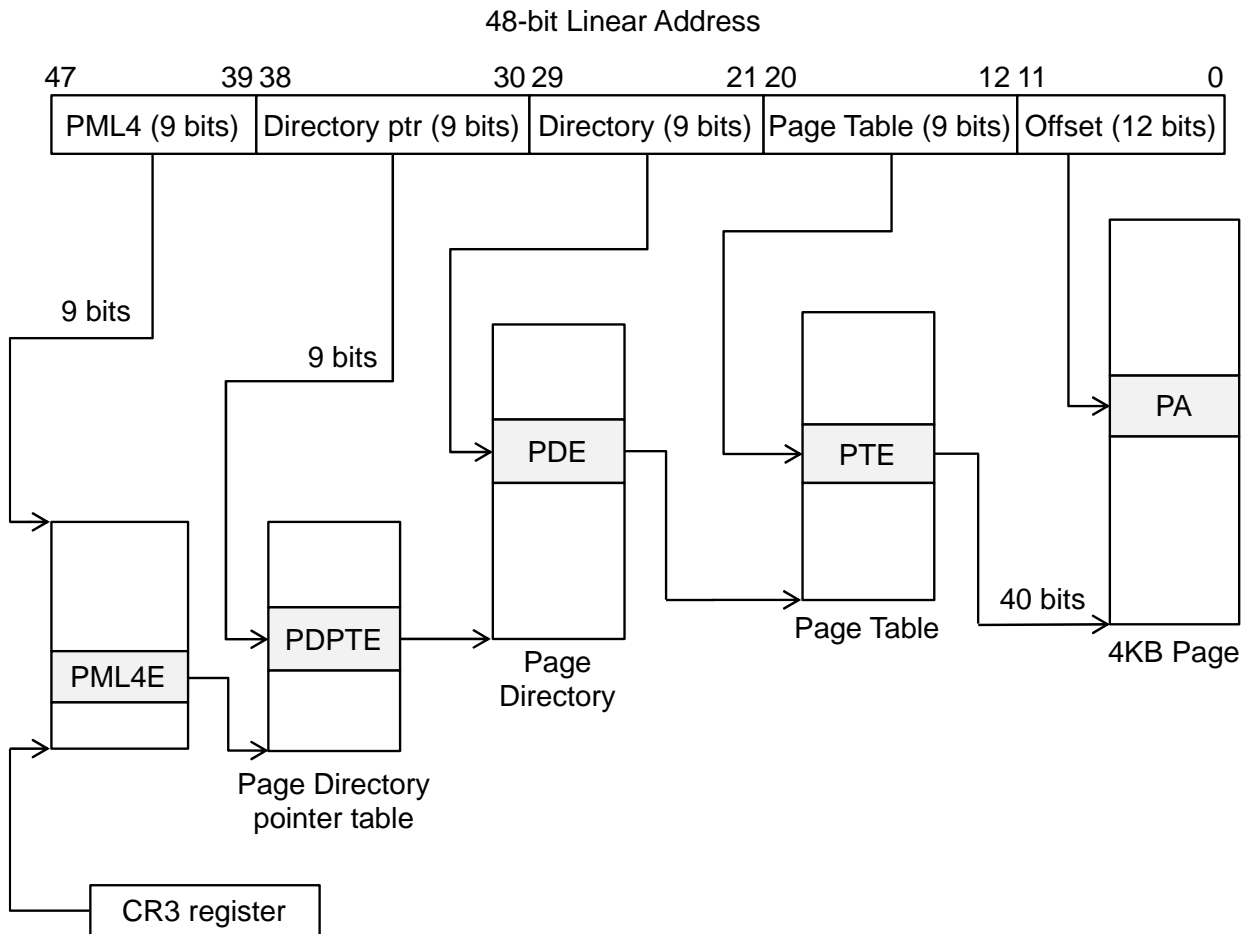
	x86-64	POWER8	ARMv8-A
<b>Address Space Protection</b>	Segments	Segments	Address Space Identifiers
<b>Virtual Address</b>	48-bit	64-bit	48-bit
<b>Page Table Support</b>	Four-level Hierarchical PT	Inverted hashed PT	Four-level Hierarchical PT
<b>TLB Model</b>	L1 Harvard L2 Unified	Single-Level Unified	L1 Harvard L2 Unified
<b>Physical Address</b>	52-bit	60-bit	48-bit
<b>Page Sizes</b>	4KB, 2MB and 1GB	4KB, 64KB or more	4KB, 16KB and 64KB

of the ATMs, and (b) to facilitate understanding of the ATM bug models (sections 2.2.1 and 2.7.3) and the silicon validation methods (section 2.3 and 2.8).

### 2.1.3 x86-64 Address Translation

The address translation mechanism of x86-64 microprocessors is based on the 64-bit extension of the IA-32 architecture (known as *AMD64* or *IA-32e*, for AMD and Intel chips, respectively). The base x86 ISA (Instruction Set Architecture) is a segmented architecture, similar to IBM's POWER8 (next subsection). However, x86-64 uses a flat model, in which segmentation is generally disabled, creating a flat linear-address space. The x86-64 uses 48-bit linear addresses (LAs) and generates up to 52-bit physical addresses (PAs) using three different page sizes: 4KB, 2MB, and 1GB. It translates LAs using a 4-level hierarchical page table as Figure 10 shows (for 4KB pages). Every structure of the page table levels is (at least) 4KB large, each entry is 64-bit long, for a total of (at least) 512 entries in each structure. CR3 register contains the first physical address of the whole page table tree.

The microprocessor generates a 48-bit LA (byte address). The LA's 12 low-order bits is the Offset and the 36 high-order bits are split into 4 groups of 9 bits, each of which is associated with an entry of the 512-entry ( $=2^9$ ) nodes of the corresponding page table level. The pointed entry of a node contains a pointer to the next-level table. The fourth-level table contains a PTE with a 40-bit address, which is concatenated with the 12-bit offset resulting in the final 52-bit PA [112] [122].



**Figure 10: Paging with 4KB page size in x86-64 ISA (PA=Physical Address, PTE=Page Table Entry, PDE=Page Directory Entry, PDPTE=Page Directory Pointer Entry, PML4=Page Map Level 4).**

In x86-64 processors TLBs and MMU caches are used to accelerate the translation process. TLBs cache PTEs (Figure 11) from the first level of multilevel page tables (Page Table). MMU caches contain three different cache levels: the Page Map Level 4 (PML4) cache, which caches PML4Es from the fourth level of multilevel page tables, the Page Directory Pointer (PDP) cache, which caches PDPTEs from the third level of multilevel page tables, and the Page Directory (PD) cache, which caches PDEs from the second level of multilevel page tables.

Software enables paging in x86-64 systems by using the MOV to CR0 instruction to set the CR0.PG bit. Before doing so, software should ensure that control register CR3 contains the physical address of the first paging structure that the processor will use for LA translation and that structure is properly initialized [112].

#### 2.1.4 POWER8 Address Translation

The ATM of the POWER8 architecture microprocessors maps an application's 64-bit Effective Address (EA) onto a global flat 78-bit virtual address (VA) space. Conversion of an EA to a VA is realized after a look-up at the Segment Lookaside Buffer (SLB), which specifies the mapping between Effective Segment IDs (ESIDs) and Virtual Segment IDs (VSIDs) and is managed completely by software. The SLB is a 32-entry-per-thread, fully associative buffer, which can support 256MB or 1TB segment sizes. The SLB entries also contain a group of important fields with information about the type of access, if the segment contains executable instructions or not, the segment size and if it is a valid one.

After a 78-bit VA is generated, it is mapped to a final 60-bit Physical Address (PA) either through the TLB (if it contains the PTE) or through an access to the Page Table. The TLB in POWER8 is a 2048-entry, 4-way set associative buffer, which stores the most recently used translations for both instructions and data. On a TLB miss, the ATM hardware looks up the Hashed Page Table (HTAB)<sup>3</sup>. In order to locate the PTE that contains the

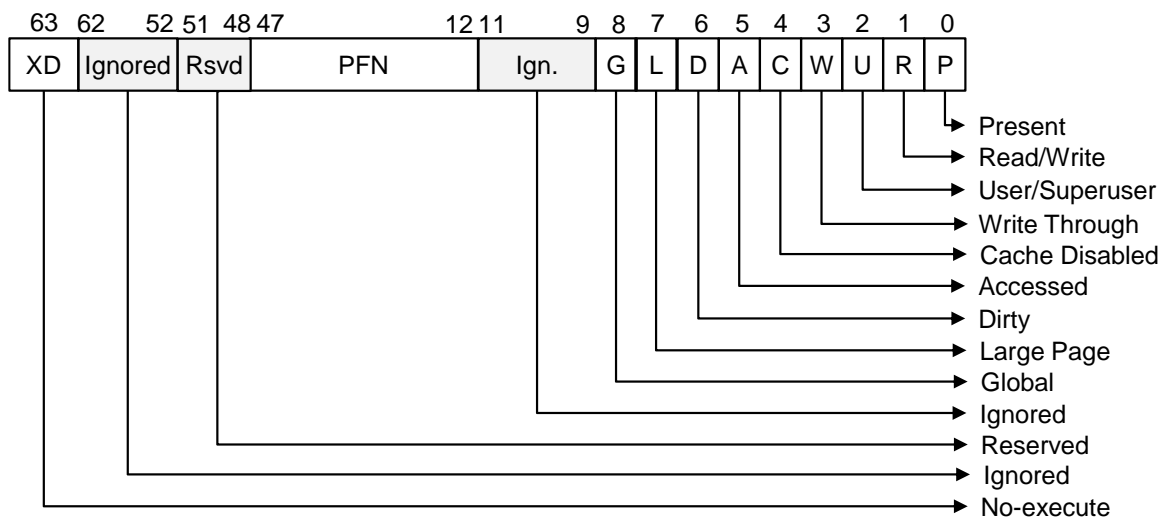


Figure 11: An x86-64 Page Table Entry (PTE).

<sup>3</sup> The HTAB is a variable-sized data structure storing the mapping between VPNs and PFNs, where the PFN of a Physical Frame is the low-order 48 bits of the address of the first byte in the frame. The HTAB contains Page Table Entry Groups (PTEGs). A PTEG contains eight Page Table Entries (PTEs) of 16 bytes each; each PTEG is thus 128 bytes long. PTEGs are entry points for searches of the Page Table [209] [210] [211].

translation of a given VA, up to two hash functions are used. The Primary Hashing uses a 39-bit hash value consisting of the VPN and the Segment Descriptor Register (SDR1), which contains information about the total HTAB's size and the PA of the page table itself. If the Secondary Page Table search is enabled and the Primary Hashing does not lead to the PTE, the Secondary Hashing takes place and the 60-bit PA is formed.

### 2.1.5 ARMv8-A Address Translation

The address translation mechanism of the ARMv8-A architecture has many similarities with the x86-64 model. Its translation system in 64-bit mode supports expanded virtual and physical address spaces, compared to the legacy (32-bit) mode of ARMv8 architecture.

The 64-bit mode of ARMv8-A uses up to 48-bit virtual addresses (VAs) and generates up to 48-bit physical addresses (PAs) using three different page sizes: 4KB, 16KB, or 64KB. It translates VAs using an up to 4-level hierarchical page table. Every structure of the page table levels is (at least) 4KB large, for a total of (at least) 512 entries in each structure. The microprocessor generates a 48-bit VA, whose 12 low-order bits is the Offset and the 36 high-order bits are split into up to 4 groups of 9 bits, each of which is associated with an entry of at most 512-entry ( $=2^9$ ) nodes of the corresponding page table level. The pointed entry of a node contains a pointer to the next-level table. The fourth-level table contains a PTE with an up to 36-bit address, which is concatenated with the 12-bit offset resulting in the final 48-bit PA. In addition, there is a Translation Table Base Register (TTBR), which indicates the base address of the first translation table required for the mapping from VA to PA.

Translation table entries can be cached in a TLB, to reduce the average latency of a memory access by caching the results of translation table walks. In order to eliminate the need for TLB maintenance on context switches, it is possible to distinguish Global pages from Process-specific pages. The Address Space Identifier (ASID) identifies pages associated with a specific process and provides a mechanism for changing process-specific tables without having to maintain the TLB structures. As an example, the Cortex-A57 MPCore has two levels of TLBs. The first level is a fully associative, Harvard-style TLB with 48 entries for instructions and 32 entries for data. The second-level TLB is a unified 4-way set associative structure with 1024 entries.

### 2.1.6 Terminology and Assumptions

Different microprocessor vendors use different terminology for their ATM subsystems. A major difference is the term used for the address that an application produces when executed in the microprocessor. This address is called Linear Address (LA) on x86-64, Effective Address (EA) on POWER8, and Virtual Address (VA) on ARMv8. To avoid confusion and keep the description of our bug models and silicon validation methods uniform, we use a consistent terminology from this point forward.

- For the address produced by the application and is the input address to the ATM, we use the term *Virtual Address* (VA).
- For the final address that is the output of the ATM, we use the term *Physical Address* (PA).

Regarding the different page tables structures, the POWER8 uses an Inverted Page Table (or Hashed Page Table) while x86-64 and ARMv8 use Hierarchical Page Table structures organized in multi-level trees. We use the term Page Table (PT) to refer to any of the organizations.

Finally, for the purposes of our description and without losing the generality of our methodology, we assume that:

- A memory word is 64-bits long (8-bytes).
- A virtual page or physical frame contains 4KB.

## 2.2 Reducing the Bug Detection Latency for ATM Post-Silicon Validation

All major microprocessor vendors regularly extend the ISAs for new functionalities and the microarchitectures with extra hardware support for performance and programmability. Virtual memory is a very mature abstraction that is widely implemented through dedicated ISA and microarchitecture support in most computing systems today. By providing the illusion of an unlimited memory space, it facilitates easier system and application programming and supports efficient system resources protection among tasks [209].

Virtual-to-physical addresses translation is the key step that a microprocessor is required to implement correctly and efficiently for a successful realization of virtual memory. With the galloping adoption of virtualization today (and the complexity its support adds to ISAs and microarchitectures now requiring multi-level address translation), the performance and correctness of address translation get critical. As we described in the previous subsections, larger TLBs and more complex microarchitectural caching structures are used to improve the speed of address translation, which is invoked multiple times as instructions and data are fetched from the memory hierarchy.

The address translation subsystem of a modern microprocessor is one among several complex mechanisms realized in state-of-the-art microarchitectures today. However, address translation (or the virtual memory support hardware in general) is probably the most difficult subsystem for correctness validation [105]. Unlike other hardware components (functional blocks, registers, caches) the output of the address translation mechanism (ATM) of a microprocessor (the physical address) is not observable to architecturally visible locations (registers, memory). Moreover, the address translation process involves several stages (see Figure 10) and can be a predominant source of severe escaped bugs which are, however, very hard to detect [102] [103] [104] [105], as shown in Table 2, which lists escaped ATM bugs into volume production published in official errata sheets from major microprocessor vendors.

As the complexity of microprocessor designs continues to grow, we are seeing an increasing gap between their design and verification. Consequently, effective post-silicon validation techniques are necessary to detect design bugs that remain after pre-silicon verification and manufacturing defects in prototype chips. Post-silicon validation on actual chip prototypes offers very high execution throughput, so design verification teams attempt to execute as many test programs as possible to obtain the largest possible validation coverage before massive chip production. For this purpose, post-silicon validation focuses on extremely large numbers of parameterized random test-programs.

The major downside, however, of random test programs is the difficulty to obtain the expected correct results (the correct output of these programs is unknown because they are randomly generated), which are required to determine the correctness of the output of the prototype chips being validated. To meet this requirement, validation process mainly resorts either to multi-pass consistency end-of-test checking methods (each test-case is executed multiple times ('passes') and the end-of-test values of some system resources (e.g., memory, registers) are compared for consistency) [98], or to golden



**Table 2: Published ATM-related bugs from official errata sheet.**

<b>Prototype</b>	<b>Bug description</b>	<b>Effect</b>
<b>AMD Athlon64/Opteron [107]</b>	Possible use of stale translations even after software has performed a TLB flush.	Unpredictable system failure
<b>IBM PowerPC 750GX/750GL [111]</b>	A mtsr or mtsrin operation, followed closely by an instruction that causes data page address translation, can cause contention for the segment registers, which proceeds using data from the wrong segment register.	Access to incorrect PA or false translation and data access exceptions
<b>Intel® Xeon® Processor 5100 Series [113]</b>	When an unaligned access is performed on paging structure entries, accessing a portion of two different entries simultaneously, the processor may live lock.	The processor may live lock causing a system hang
<b>AMD Athlon64/Opteron [107]</b>	INVLPG instruction with address prefix does not correctly invalidate the requested translation in the TLB.	Unpredictable system failure
<b>Intel® Xeon® Processor E3-1200v3 [115]</b>	The Intel® VT-d supporting the Processor Graphics device may not report ATM faults detected on Display Engine memory accesses when the Context Cache is disabled or during time periods when Context Cache is being invalidated.	Display Engine accesses that fault is correctly aborted but may not be reported in the fault reporting register
<b>AMD K10 (Family 10h) [108]</b>	The processor may use an incorrect cached copy of translation tables when it is in legacy Physical Address Extension (PAE) mode and the guest address translation tables reside in physical page zero.	Unpredictable system behavior
<b>Intel® Xeon® Processor E3-1200v3 [115]</b>	If a logical processor has EPT (Extended Page Tables) enabled, it uses 32-bit PAE paging, and accesses the virtual-APIC page then a complex sequence of internal micro-architectural events may cause an incorrect address translation or machine check on either logical processor.	Uncorrectable TLB error, a guest or hypervisor crash, or other unpredictable system behavior.

responses generated by architectural simulators. However, the throughput difference between native chip execution and simulation (at different levels) is between 3 and 6 orders of magnitude. Therefore, both approaches (they are also known as “end-of-test checking techniques”) suffer from the same limitation: by detecting a mismatch only at the end of the validation test, debug engineers devote excessive effort to identify the root cause of the bug because the mismatch is identified after thousands or millions of executed instructions [98] [101].

To this end, we present and discuss two complementary ISA-independent software-based post-silicon validation methods, which quickly expose difficult-to-find bugs in the address translation mechanisms of modern microprocessors. Both methods make debugging and diagnosis of a bug more effective and significantly reduce the bug detection latency during the post-silicon validation phase.

### 2.2.1 ATM Bug Models

Bugs in a microprocessor's ATM can manifest themselves in arbitrary ways (which is also the case for any other hardware structures). However, the definition of a bounded but comprehensive set of the effects that ATM design bugs can have is an important formulation, which can assess the effectiveness of corresponding post-silicon validation methods, assist engineers to identify the root cause of a bug, and accelerate the validation and debugging process. They can be also used to guide research in microprocessor address translation hardware validation methods allowing the comparison of different approaches.

The bug models are ISA-independent and cover all hardware structures involved in address translation in a modern microprocessor. Our ATM bug models' definitions are the result of the

- published descriptions from errata reports of real microprocessor ATM-related bugs of Intel, AMD and IBM chips ([107] [108] [109] [110] [111] [112] [113] [114] [115] [116]) that slipped in volume production, and
- close interaction with IBM's specialized microprocessor validation teams.

We define a comprehensive set of bug models in ATM hardware, which describe the effect of ATM bugs in any ISA (including x86-64, POWER8 and ARMv8 discussed previously); similarly, bug models for other parts of a microprocessor have been defined in [118] [123] [124].

Each bug model category describes a divergence from the expected behavior, i.e., the effect of a bug. The diverging behavior is attributed to a functional bug or an electrical defect and can appear in any hardware structure or interconnection involved in address translation. The bug modeling categories are enumerated below:

- (1) **Unintended Modification** – Due to a bug, an ATM-related component (or one of its entries) is unintentionally modified. For example, an unintended update of the CR3 register takes place in an x86-64 microprocessor; also, an unintentional update of a TLB entry in the x86-64, POWER8 or ARMv8 architecture belongs to this category.
- (2) **Ignored Modification** – Due to a bug, a modification of an entry of an ATM-related component fails. For example, a group of entries in the Page Table (PT) is updated, but the corresponding TLB invalidation fails and the previous contents of the TLB are kept.
- (3) **Wrong Value** – Due to a bug, a wrong value is written to the correct ATM-related component entry. This bug model supplements the two previous models. The selected entry is correct, but the value written to it is wrong.
- (4) **Dirty Read** – Apart from the updates to registers and entries (Figure 10 and Figure 11) of ATM-related components (TLBs etc.), the ATM hardware also performs reads from these components. A Dirty read happens when the value of a register or an entry is correct, but due to a bug upon a read, the microprocessor reads a wrong value or it reads from a wrong location; but the correct value still exists in the register or entry. A real-life example of this category is a serious bug in Intel®

Xeon® Processor E3-1200v3, in which when 32-bit paging is in use, the processor should use a page directory located at the 32-bit physical addresses specified in bits 31:12 of the CR3; the upper 32 bits of CR3 should be ignored. Due to a bug, the processor uses a page directory located at the 64-bit physical address specified at the 63:12 of CR3 register [115].

- (5) **Timeout** – Due to an error, the microprocessor is trapped to a deadlock or a live lock. For instance, Intel® Xeon® Processor 5100 Series suffers from a potential live lock when an unaligned access is performed on paging structure entries, accessing a portion of two different entries simultaneously [113].
- (6) **Unintended Exception** – Due to an error, an unintended ATM-related exception is raised. For example, a Page-Table Exception Fault is raised, although the required (Page Table Entry) PTE already exists in the Page Table (PT).
- (7) **Ignored Exception** – Due to an error, an exception that should have been raised does not happen. For example, an access rights violation to a physical frame is ignored, and the ATM and permits access to addresses in the frame.
- (8) **Wrong Exception** – Due to a bug, a wrong exception handling routine is invoked.

The eight bug model categories described above are mutually exclusive. The effect of a local bug in any ATM-related structure of the different ISAs is uniquely classified in one of the categories.

## 2.3 Proposed Method

Special purpose verification languages support automatic stimulus generation to enable better specification and design coverage, such as IBM's Genesys-Pro [214] and Synopsys' Vera [215]. These frameworks allow design engineers to express pseudo random test program generation along with complex event scenarios using generic *Test Templates* [216] [217]. A Test Template defines any desired verification scenario, and based on the microprocessor's architecture, the corresponding post-silicon validation programs are generated [212] [213].

Figure 12 shows the proposed framework for post-silicon validation of the ATM hardware. As we explain in the next subsection, our methodology does not require previously generated golden responses by a simulator (a major requirement to facilitate the execution of very large numbers of post-silicon validation programs).

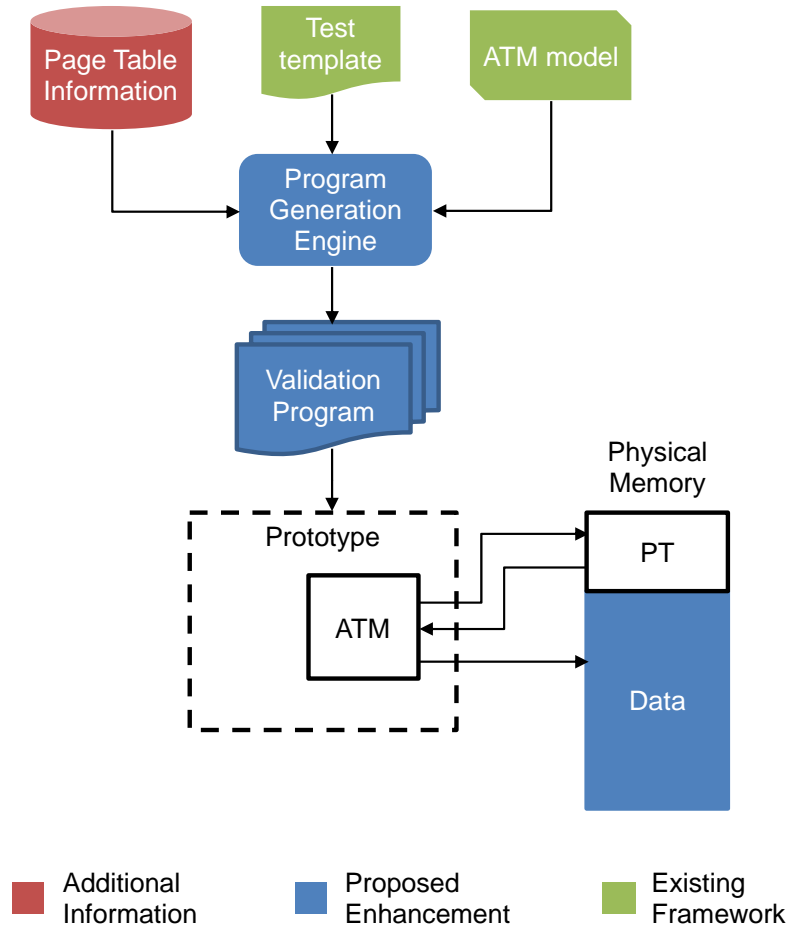
The Program Generation Engine takes as inputs:

- a detailed model of the ATM paths (existing framework),
- a Test Template, which specifies the overall silicon validation plan and describes the test scenarios (existing framework),
- the Page Table information (proposed enhancement), and
- a corresponding memory image (proposed enhancement).

The above scenarios are translated by the Program Generation Engine into complete post-silicon validation programs. The final self-checking validation programs completely stress and validate the ATM hardware of the microprocessor. Subsequently, the validation programs are loaded into the prototype chip (also referred to as the device under validation – DUV) and the validation process begins. In the next subsections we describe the enhancements required in the traditional flow.

### 2.3.1 Page Table Setup

The role of the page table in the proposed method is twofold:



**Figure 12: The proposed ATM silicon validation framework.**

- (a) Post-silicon validation of microprocessor prototype chips is performed for long periods in a bare-metal configuration (e.g., a single-process validation program is executed at a time directly on the silicon chip without any operating system support). To detect bugs in the ATM in a bare-metal configuration we need to construct an initial page table in the system memory, so that the ATM of the microprocessor operates normally (as if the system was in normal mode), and thus, all translation paths of normal system operation are comprehensively excited (coverage). The page table contains the virtual address (VA) to physical address (PA) translations and several status/permission bits required by the generated validation program. By setting the parameters of the validation program generation flow (as presented in subsection 2.3.5), we can both validate the address translation for arbitrary physical memory areas and cover all entries of the ATM hardware structures (TLBs, MMU caches, control registers, etc.).
- (b) We exploit the contents of the page table during validation, since random test generators have knowledge about them (Figure 12). The known page table contents (Figure 13b) are used to provide the expected correct results of ATM (this is the fundamental reason for which our method holds the self-checking property and does not resort to any already known externally derived response), as it is described in the next subsections.

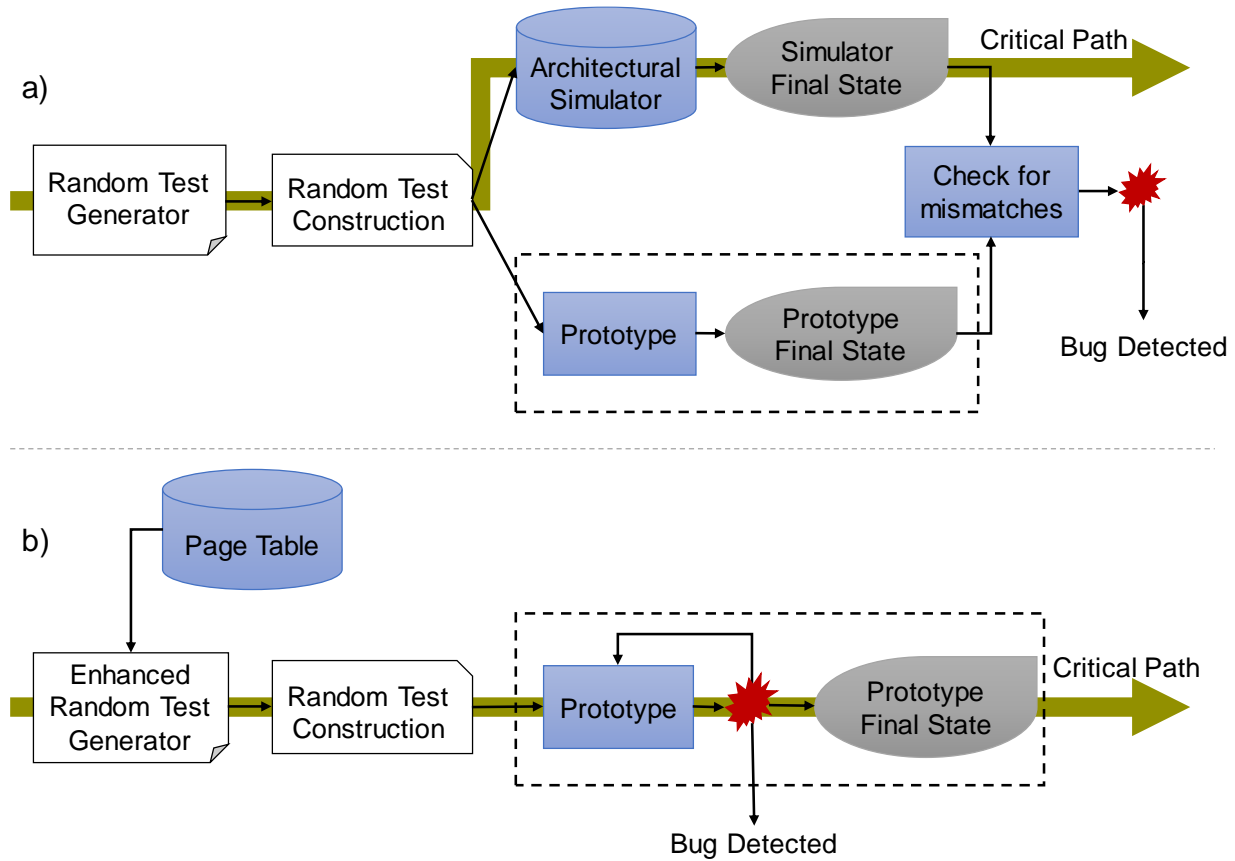


Figure 13: a) A conventional post-silicon validation flow vs. b) the proposed self-checking validation flow.

### 2.3.2 Memory Image for ATM Post-Silicon Validation

Applying existing validation approaches to the ATM hardware would require availability of golden responses (e.g., from architectural simulators or from emulators). In such a setup, the (correctness) checking phase would be performed at the end of the validation test. However, for ATM validation there is *no way for the validation program to have direct access to the generated physical address*, which is the output of ATM for a given virtual address. Figure 13 shows a comparison of a typical validation flow (Figure 13a) and the proposed self-checking flow (Figure 13b). Aiming to improve the observability of ATM hardware and also to reduce the bug detection latency (eliminating the checking phase at the end of the test), we present the following method.

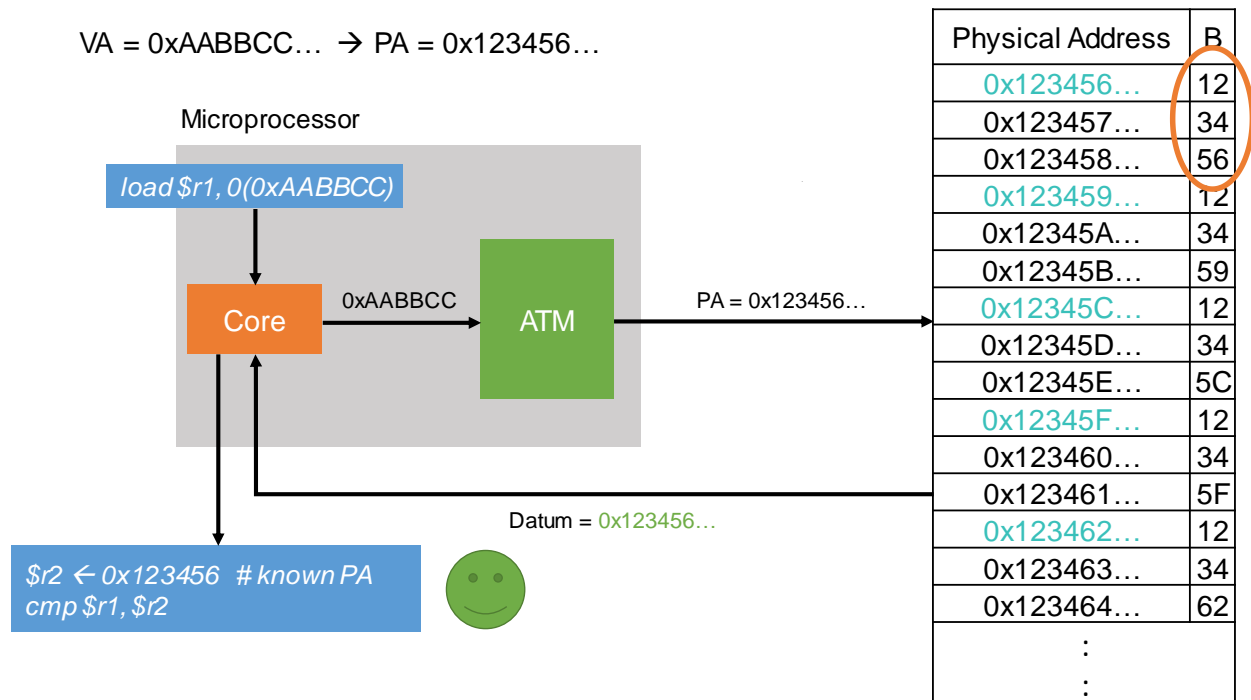
Before the execution of the validation programs, the physical memory locations that the test program will validate (of course these should be different than the ones holding the validation program itself or the page table) *are written with data values equal to their actual physical address* (i.e., every physical address  $A$  in the range being validated contains value  $A$ ).

While storing the desired data in memory, the microprocessor operates in real-addressing mode (details in subsection 2.3.4) during this phase, so that the ATM is disabled and does not impede the proper memory initialization of the desired data (otherwise a bug in ATM could threaten the initial state of the physical memory). This is a key for the proposed methodology, because we must ensure that the initial data in memory must be correct before the validation process begins. Additionally, in most of the cases, the DRAM in a commercial server is ECC protected, so we are sure that the stored data in memory are not jeopardized.

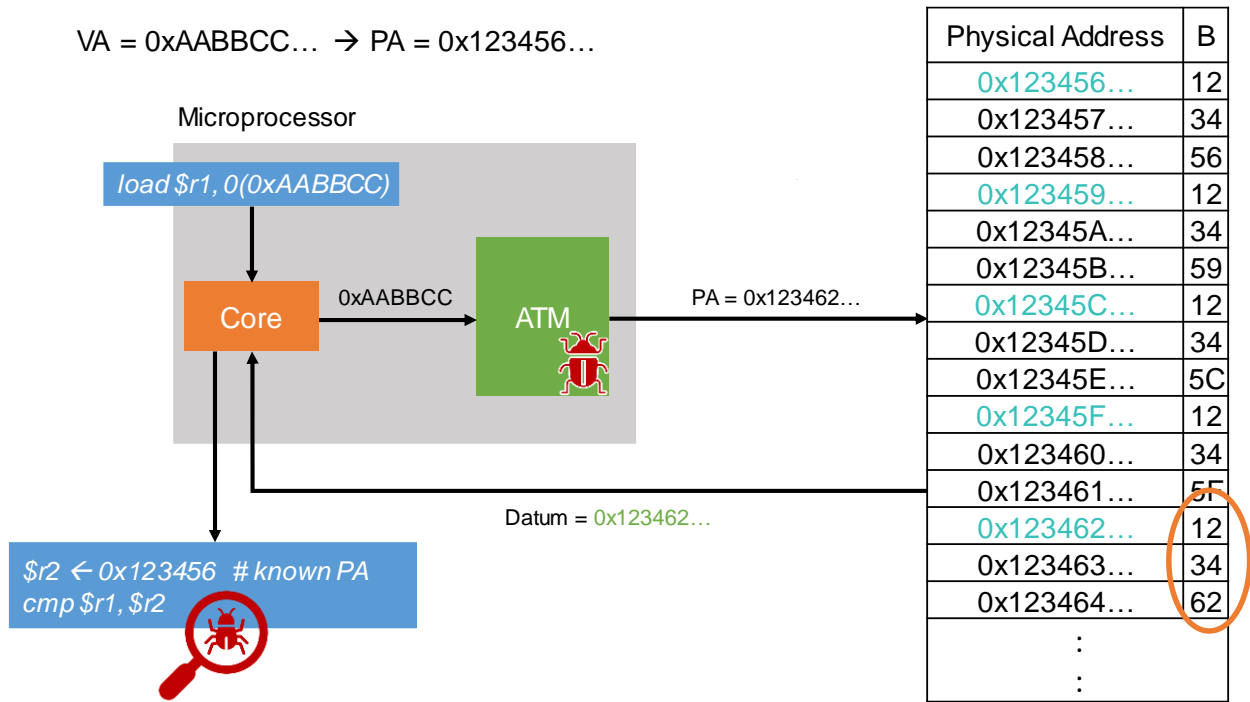
For example, if the physical address 0x123456000 belongs to the area being validated, the data value 0x123456XXX is stored in it. The last 12 bits can have any value since they represent the *Offset* (see Figure 9 and Figure 10). We take advantage of these unused bits to detect ATM bugs related to the control bits (not only to address mappings) of the address translation process (see subsection 2.3.3). Therefore, when for example a virtual address VA=0xAABBCCXXX is mapped to the physical address PA=123456XXX, the fetched datum will be equal to 0123456XXX.

Assume for example that a VA=0xAABBCCXXX is mapped to the PA=0x123456XXX. Figure 14 and Figure 15 outline the proposed validation concept that does not need end-of-test known-correct results to decide if the result is correct. In Figure 14 we can see a correct (bug-free) operation of the validation program, in which the data value returned to the validation program by a load instruction is similar to its physical address. After the datum is fetched from the physical memory it is compared during the execution of the validation program to the known Physical Address that the ATM should generate. Given that the comparison is correct, the validation program continues its execution without any notification of a mismatch. In case of a mismatch in this comparison an ATM bug is detected right after its manifestation. As we can see in Figure 15, the ATM provides a wrong PA (due to a bug in the ATM), and as a result the fetched datum from physical memory is different from its PA. After a mismatch is detected (due to the incorrect comparison), the validation test halts execution and provides specific results to the debugging process (e.g., the correct and faulty PA).

The proposed validation method, therefore, does not need simulated or emulated results as golden responses, which is the case of traditional flow as shown in Figure 13a, or other time-consuming methods that provide the known-correct results at the end of the program, because the physical addresses that the validation programs traverse contain data values equal to the physical address. As a result, if a bug in the ATM occurs, our method can instantaneously detect it, after a few cycles (more details in subsection 2.3.6) – and not at the end of the test after several thousands of clock cycles.



**Figure 14: The Validation Program accesses memory through the ATM. The ATM translates the VA to the correct PA and the fetched (pre-stored) data value is returned back to the validation program, which compares it with the expected PA.**



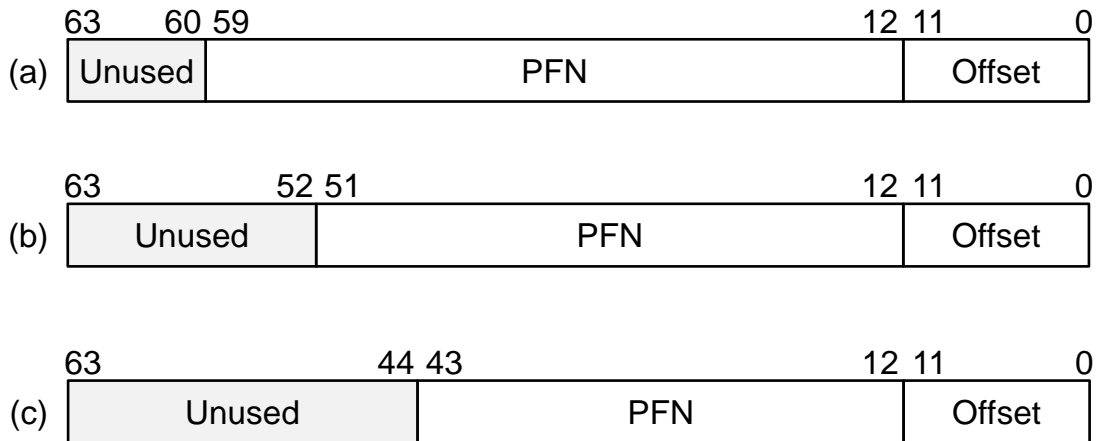
**Figure 15: The Validation Program accesses memory through the ATM. The ATM translates the VA to a wrong PA and the fetched (pre-stored) data value is returned back to the validation program, which compares it with the expected PA. A mismatch occurs indicating the bug.**

### 2.3.3 Utilizing Unused Bits in Physical Addresses

When a microprocessor operates in 64-bit mode, all physical addresses that indicate the stored data in memory are 64-bit long. However, not all available bits of the PA are used. Some of them are used for the Physical Frame Number (PFN) and the 12 low-order bits hold the offset into the frame (for 4KB frames). As shown in Figure 16, the POWER8 ISA uses the middle 48 bits, the x86-64 uses the middle 40 bits and the ARMv8-A uses the middle 32 bits to form the PFN. All three ISAs use the 12 low-order bits for the Offset, and the remaining high-order bits are unused.

The 12-bit Offset does not participate in the address translation process (it is not stored in the PTs, the TLBs or any other ATM structure).

Instead of storing arbitrary values to the 12 low-order bits of the 64-bit physical memory locations (the three X nibbles mentioned above); we use them to store useful information



**Figure 16: A 64-bit PA and the PFN bits, the Offset bits and the unused bits in (a) POWER8, (b) x86-64, and (c) ARMv8-A.**

about the translation process. Apart from checking for correctness the PFN that the ATM produces, it is also mandatory to validate the other available attributes that a PTE contains (control bits). For example, one of the most challenging aspects to validate is a bug that leads to an access to unprivileged locations in memory; when a physical frame is read-only but a write to that frame is not prohibited. Thus, the useful information about those attributes contained in PTE is also pre-stored on each frame to the 12 low-order bits. This information (the correct values of this bits) should also be kept in the validation program similarly to the virtual to physical address translation.

Assuming, for example, the x86-64 PTE organization. Figure 11 above shows the control bits of a PTE. Apart from the PFN, there are also several useful attribute bits that should be validated to ensure coverage of the bug models related to these attribute bits.

### 2.3.4 ISA-Independent Data Loading in Physical Memory

Two mode of memory addressing are typically used: the *real addressing mode*, in which the generated address is mapped one-to-one into a physical location in memory (the ATM is bypassed); and the *protected addressing mode*, in which the ATM is used to translate a virtually generated address to a physical one. A microprocessor boots in real mode, where the ATM is disabled and does not interfere with the microprocessor's functionality. We take advantage of this property to setup the microprocessor and the physical memory for the validation flow.

We set the microprocessor in the real addressing mode to store the desired memory contents to the physical memory, and to set up the page table, the global descriptor table and other fundamental data structures. After this, the kernel switches to the protected mode, and the validation process begins. We analyze this process in subsection 2.4.1, where we present our bare-metal setup and experimental results.

Modern microarchitectures, such as x86-64, ARMv8 and IBM POWER, support the real and the protected memory addressing modes, which are controlled by specific control registers; each microarchitecture has its own control registers and control logic. For example, if the PG bit (bit 31) of CR0 control register is disabled ( $CR0.PG = 0$ ), the Address Translation Mechanism in x86-64 is ignored and the logical processor treats all virtual addresses as if they were physical addresses. In ARMv8, address translation stages are disabled by setting the M bit (bit 0) of a system control register (SCTLR) to 0 ( $SCTLR.M = 0$ ). POWER8 has similar features, which can enable and disable the address translation depending on the microprocessor's operation. POWER8 includes a special Machine State Register (MSR). MSR provides a set of control bits for handling the status of the address translation. As a result, in modern microarchitectures the address translation process is optional and can be enabled or disabled by each microarchitecture's control registers, and thus, our proposed method is ISA-independent.

### 2.3.5 Transformation of Validation Tests

A validation method for a particular hardware structure must fully stress the hardware for all different modes of operation and corner cases to deliver high validation coverage. For the extensive stressing of the ATM hardware our method augments existing random validation tests into new tests, which exploit the self-checking property of our method, and thus, embed our method and accelerate the overall validation flow. The validation programs contain raw information about the virtual-to-physical address mappings and their associated status/permission bits from Page Table, as shown in Figure 12 above. This is required for the realization of the self-checking property, and to improve the observability of the ATM hardware, which is vital for debugging purposes.



As shown in Figure 17, before the execution of each load instruction, validation tests are augmented to store to an architectural register the expected physical address. After the execution of a load instruction, any type of comparison between the expected PA to the fetched datum can be applied (a conditional branch, a compare instruction followed by a jump) in order to check the matching between the expected correct physical address and the fetched datum from a memory location. The comparison should succeed in a correct design because our proposed self-checking method (described in the previous subsections) pre-stores data values equal to their actual physical address, and should fail on an incorrect design. The main role of this program transformation is to decouple the traditional end-of-test checking phase from validation tests that are going to be loaded and executed in the prototype chips and to improve the observability of the ATM hardware.

### 2.3.6 Reducing the Error Detection Latency

Due to the exploitation of the full performance of prototype chips by using the pre-stored physical addresses, control bits and the enhanced validation tests, the elapsed time between bug manifestation and its detection is significantly decreased by our method. As shown in Figure 18a, in a traditional post-silicon validation flow, the detection of a bug occurs when the validation test finishes. Then a comparison between end-of-test golden results and the silicon's final state takes place to ensure correctness. Instead, by using the proposed post-silicon validation method (Figure 18b), the potential error in prototype is detected right after its manifestation, because the results of each individual address translation are known a priori and they have been pre-stored in the memory.

Detecting bugs right after their manifestation and not after the execution of millions of instructions [117] [118], provides a simple and rapid way to determine the divergences from the expected behaviors. Additionally, by recognizing the incorrect PA that manifests the bug and thereby facilitating the localization of the particular bug, when an error in silicon occurs, it is easier for debug engineers to identify the root cause of complex bugs (such as bugs caused by electrical defects).

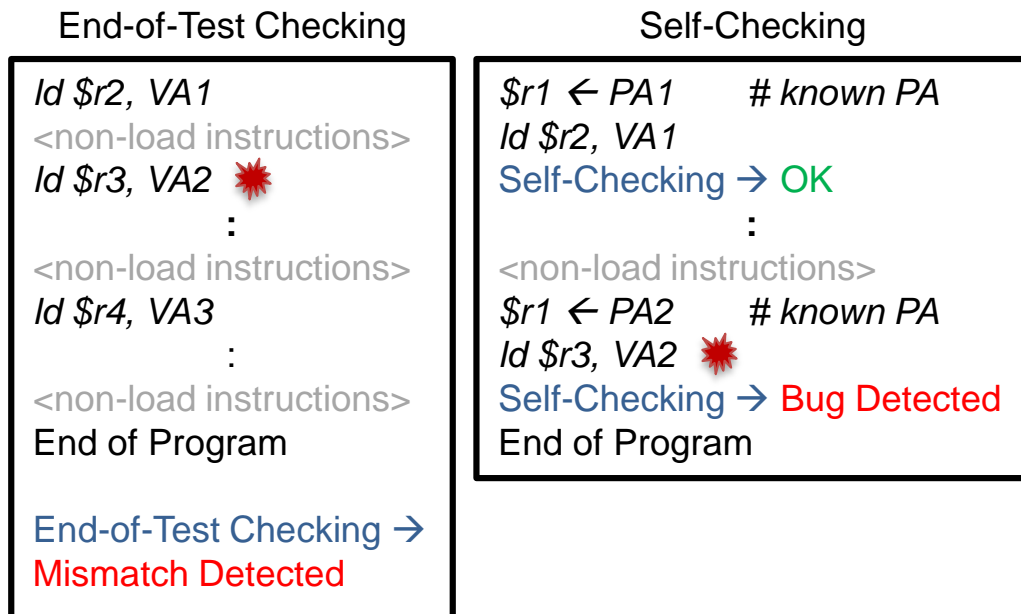


Figure 17: A conventional example of a validation program (left) that checks for correctness at the end of execution vs. the proposed transformations to achieve the self-checking property (right) that checks for correction right after each memory access.

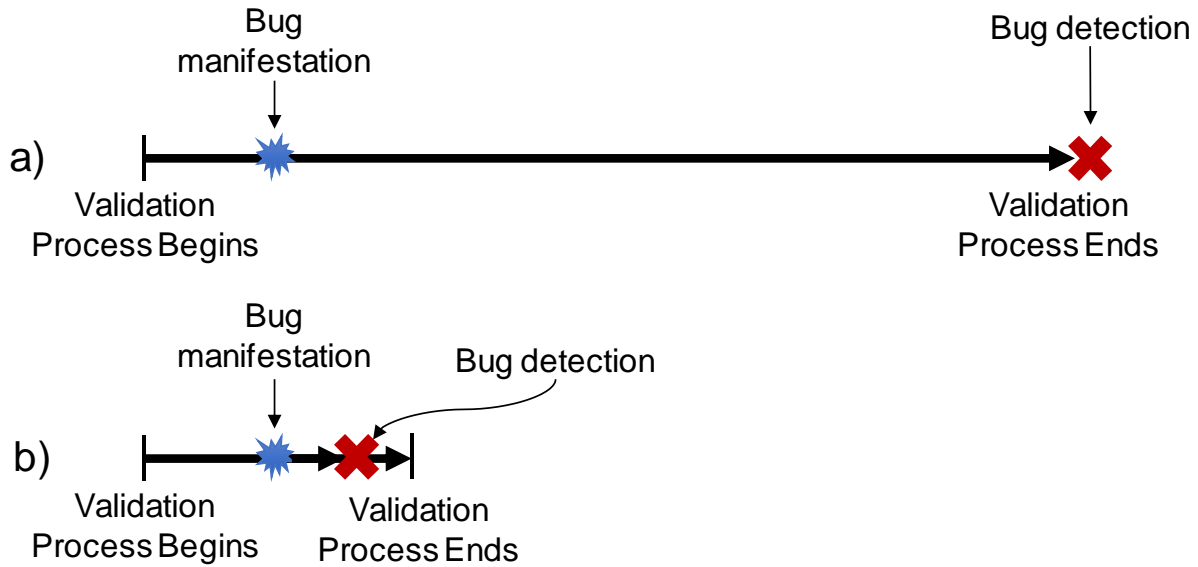


Figure 18: a) A conventional validation process with end-of-test checking, and b) the proposed self-checking validation process.

## 2.4 Experimental Evaluation

### 2.4.1 Simulator Setup and Methodology

The proposed silicon validation methodology for microprocessor ATM subsystems can be adapted to any ISA and specific microarchitecture. When existing validation tests are enhanced with the proposed methodology, very large numbers of validation programs can be applied to the microprocessor under validation without resorting to any golden responses generated by time-consuming simulations.

As Table 3 summarizes, the combination of the original validation programs and the self-checking enhancement we presented, leads to very high coverage of all ATM bug models

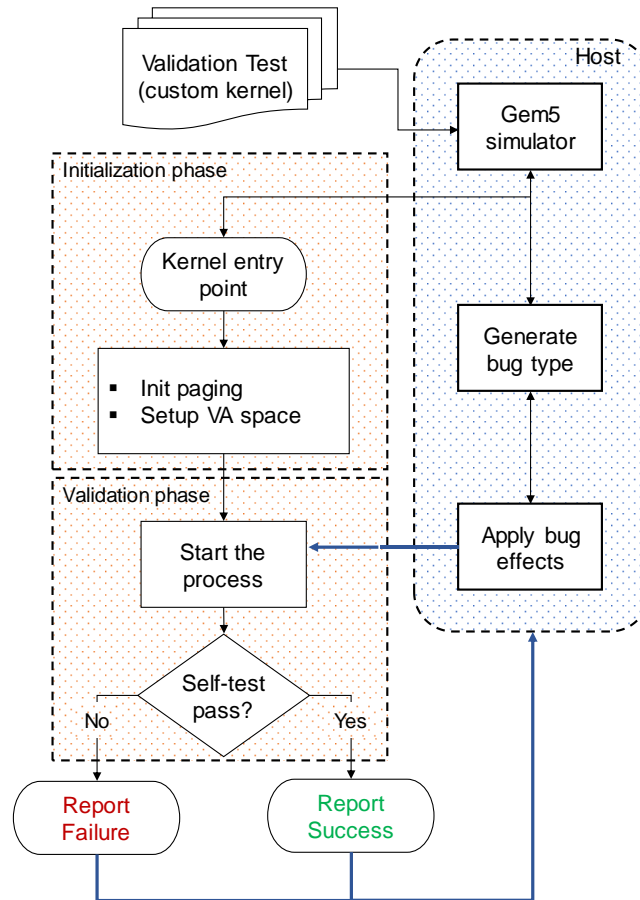
Table 3: Bug coverage of the proposed ATM validation method.

ATM Bug Model	Bug Coverage
Unintended Modification	Detected when entry accessed (self-check mismatch)
Ignored Modification	Detected when entry accessed (self-check mismatch)
Wrong Value	Detected when entry accessed (self-check mismatch)
Dirty Read	Detected (self-check mismatch)
Timeouts	Detected (visible failure)
Unintended Exception	Detected (traditional validation test)
Ignored Exception	Detected (traditional validation test)
Wrong Exception	Detected (traditional validation test)

described in 2.2.1. A well-known difficulty of assessing validation methods is to quantify the coverage of errors that can be detected (mainly electrical defects or rare functional bugs) that could exist in the silicon. Therefore, to demonstrate the effectiveness of our post-silicon validation method and potential usage scenarios we emulate an actual bare-metal silicon validation infrastructure using the x86-64 model of the Gem5 simulator [120], by injecting a large number of bugs that could occur in ATMs, as well as known ATM bugs reported in the official errata sheets.

We implement an important enhancement of the Gem5 simulator. We model MMU caches – a very important modern ATM structure realized in state-of-the-art microprocessors. MMU caches significantly reduce the TLB miss penalty and improve program execution. As shown in Figure 19, we have also modified the Gem5 simulator to support injection of random, non-deterministic bugs (in random clock cycles and in random bits) in the address translation structures of the x86-64 model. With these approaches we resemble a realistic “buggy” microprocessor prototype chip (with many different types of bugs in the ATM). We also developed from scratch a custom minimal kernel to prepare a realistic bare-metal post-silicon validation “environment” and execute our validation programs. The most critical part of functional post-silicon validation is the one based on bare-metal-infrastructure and we contribute to this part.

During functional post-silicon validation only random-generated test programs are applied, not normal applications<sup>4</sup>. Bare-metal modeling on Gem5 was a major part of our work to resemble the real hardware infrastructure that is needed for the method to be applied to actual silicon prototypes. Our kernel initiates all the required procedures to set



**Figure 19: Proposed post-silicon validation flow and the experiment setup.**

<sup>4</sup> Other post-silicon validation phases focus on full-system configuration with an operating system layer and application software on top of it.

up the paging interface required to operate in Long Mode (x86-64) (Figure 19). It also offers a fundamental infrastructure for virtual memory mapping. The initial state of the memory (physical addresses and control bits), which is presented in subsection 2.3.2 is achieved using the simulator's boot loader. A chunk of 32 MB is initialized for the needs of our validation requirements.

### 2.4.2 Validation Programs Implementation

We implemented the validation tests and the “buggy” behavior of the microprocessor as shown in Figure 19 with the following way: we executed the enhanced random-generated validation tests (according to the proposed method); each test targets a unique bug model category. During program execution, we injected incorrect behaviors to the simulator in random clock cycles (one injection per execution) to check if the proposed method is able to detect them. We relied on injection of bugs at random clock cycles in order to achieve the non-deterministic manifestation of errors for a comprehensive evaluation of our method.

We have developed four different validation programs each targeting a different bug model of subsection 2.2.1. Since our method involves a transformation of existing well-established random tests to realize the software-based self-checking approach of our method, we employ random-generated validation programs to traverse as many address translation paths as possible. Our transformation is applied to all programs.

**Unintended Modifications** (*generic; any ISA*) – Validation Program #1 (VP1): The program starts by mapping different virtual pages (VPs) to physical frames (PFs) of the initialized memory. It then goes through the VP and initiates loads on different offsets. It aims to detect *unintended modifications* of the matching TLB entry upon access (right after). The program can only miss the very rare cases when a TLB entry gets corrupted on the last access.

For the validation runs of this experiment, Gem5 was modified to corrupt a TLB entry upon its access, *after* performing a translation, on a rate of 0.01% (an entry corruption every 10,000 accesses), thus emulating the buggy behavior.

**Unintended Modifications** (*x86-64 specific*) – Validation Program #2 (VP2): This is an x86-64 ISA-specific scheme to check whether a TLB flush was unintentionally (due to a bug) initiated. The program first maps a VP to a PF, it requests a TLB flush to update the translation, and then asks for a load on that address, to ensure that the translation is placed on the TLB. It finally remaps the VP to another PF without flushing the TLB.

The TLB should have a stale translation at this point that corresponds to the first mapping. A loop of loads goes through the entire VP and compares the read values with the initial translation. If the TLB gets flushed, the translation will be updated to the new mapping and the self-check comparison will report a mismatch.

For the runs of this experiment, Gem5 was modified to initiate some unintentional (not normal) TLB flushes, thus emulating the buggy behavior.

**Ignored Modifications** (*x86-64 specific*) – Validation Program #3 (VP3): This validation scheme is again x86-64 specific. The program examines if TLB flushes occur upon update of the CR3 register, as described by the ISA specification. The program maps a single VP to a PF of the initialized memory. It then requests a TLB flush and performs 16 loads on different offsets of the mapped page. It compares the values with the expected physical addresses. After the 16 loads, it will re-map the page to a different frame and again request a TLB flush. The process goes through the entire 32MB of initialized memory. If the ATM hardware fails to flush the TLB, it will use a stale translation and it will cause a mismatch on the first load instruction of the new page map (offset 0).

For the runs of this experiment, Gem5 was modified to skip 1% of the requested TLB flushes, thus emulating the buggy behavior.

**Wrong Values** (*generic; any ISA*) – Validation Program #4 (VP4): The program intends to detect all of the wrong values delivered by the page walking process. It starts by mapping different VPs to PFs of the initialized memory. It then performs loads on the mapped VA to compare the loaded value with the requested mapping. If the value matches the expected physical address, it moves on to the next frame until going through all of the 32MB of initialized memory. If no mismatches are found, the program reports success.

### 2.4.3 Results

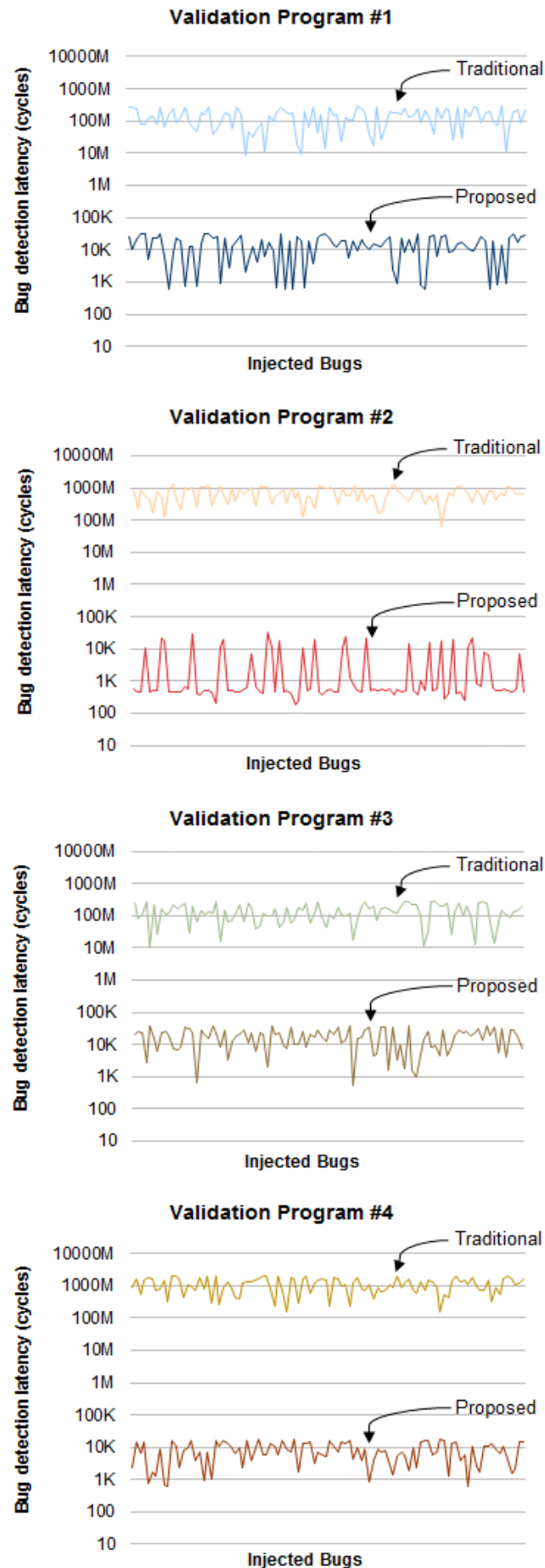
We executed each of the four validation programs 1000 times and the bug occurrence ratio was adjusted to one bug per simulation. Practically, there is no need to inject more than one bug per simulation, because, as we already described, the validation process terminated when the first bug is detected. Table 4 reports: (a) the number of injected bugs (1000 in each case – 4000 bugs total covering all hardware components involved in address translation; second column); (b) the number of virtual pages and physical frames (third and fourth columns) that are covered by the emulated validation process for every different validation program; and (c) the number of self-checking comparisons that each validation program performs (fifth column). The validation programs were also simulated against the golden (unmodified) Gem5 and reported success on the bug-free executions.

Validation programs VP1 and VP2 require a significantly larger number of self-checking comparisons than programs VP3 and VP4. The reason is that programs VP3 and VP4 are designed to detect bugs that are more promptly excited by accesses to the corresponding entries, while programs VP1 and VP2 are designed to detect bugs, which require much longer execution before the modified entry is again accessed. Finally, program VP2 requires a very large number of checks because the entire TLB and the MMU caches are frequently flushed compared to program VP1.

To measure the detection latency in our experiments, we defined a time slot that begins when the bug manifests itself and ends when its detection occurs. Figure 20 presents the bug detection latency for each one of the injected bugs in four validation programs. It demonstrates that, by using our self-checking method, the bug detection latency is around 10,000 cycles (small variations among the different validation programs), while on the other hand the traditional end-of-test validation techniques have, on average, 5 orders of magnitude longer duration (given that the detection occurs at the end of the validation test), taking into account a validation program of 2B cycles long. All graphs of Figure 20 are shown in the same scale to demonstrate the differences among detection latencies for each program category. For instance, the line of the proposed method of validation

**Table 4: Experimental results on the bare-metal setup in the x86-64 model of the enhanced Gem5.**

Test	Total Bugs	Virtual Pages	Physical Frames	Translation Checks
VP1	1000	7,680	7,680	122,880
VP2	1000	1	7,680 (remapped frames)	3,932,160 (checks for stale translation)
VP3	1000	1	7,680 (remapped frames)	7,680
VP4	1000	7,680	7,680	7,680



**Figure 20: Bug detection latency for traditional end-of-test checking methods and the proposed self-checking.**

program VP2 shows increased detection latencies in contrast to validation programs VP3 and VP4, given that this group of bugs (as it is already described) need more checks than the other groups of bugs.

In summary, our experiment evaluation of the method reveals its very fast detection capabilities compared to traditional end-of-test validation approaches as well as the high validation coverage it delivers for the bug models proposed in this work.

## 2.5 Bug Coverage and Method Limitations

Given that the data in memory is pre-stored before the validation process begins their values can be aligned to 8B, 4B, 2B or 1B. If, for example, the pre-stored values stored as 8-byte words, the validation test should access only 8-byte words. Another limitation of the proposed method is related to store operations. If the original ATM validation program performs a store that can affect the pre-loaded data values, the self-checking property is jeopardized (there will always be a mismatch between the physical address and the stored data in that address without a bug). This limitation may mask some cache coherency bugs. Fortunately, this is an easy limitation to avoid by properly guiding the random validation program generation.

Moreover, due to the self-checking nature of the methodology, ATM-related bugs that do not result in a wrong operation (e.g., wrong translation or incorrect privileges and thus exception) but only lead to performance loss (extra execution cycles) *are not detected*. Of course, such bugs are less severe than bugs that affect correctness and are due to the address translation caching arrays (e.g., TLBs) that exist in the ATM to improve performance. In addition to that, several publicly available official errata reports from major microprocessor vendors present severe escaped bugs also in address translation caching arrays (e.g., Translation Lookaside Buffers) that persist across generations. These bugs may affect both the performance and functionality of the microprocessors [113] [116] [125] [126] [127] [128] [129] [130] [131] [132].

In order to face some of the limitations of the proposed method and to enhance the post-silicon validation flow to be able to expose difficult-to-find bugs in address translation mechanisms, and mainly in address translation caching arrays, we present the following method, which is the second major contribution of the thesis in the area of post silicon validation of ATM.

## 2.6 Unveiling Difficult Bugs in Address Translation Caching Arrays

As we previously discussed, the address translation mechanism (ATM) of modern microprocessors is one of several complex mechanisms realized in state-of-the-art microarchitectures today. It is also probably one of the most difficult subsystems to validate for correctness ([102] [103] [104] [105]), due to the intricate functionality provided by its caching arrays. Address translation caching arrays (ATCA) such as Translation Lookaside Buffers (TLBs) and Memory Management Unit (MMU) Caches [122], minimize the address translation latency by significantly reducing multiple memory accesses that aim to translate virtual addresses into physical ones. Escaped bugs in these arrays, which do not necessarily result in a wrong operation but only in significant performance loss, are difficult to detect using traditional validation tests. This occurs because they barely affect the correct functionality, making their occurrence hard to observe. Bugs that manifest themselves only under certain operating conditions and their manifestation does not result in an incorrect architectural level output are massively reported from major microprocessor vendors ([113] [116] [125] [126] [127] [128] [129] [130] [131] [132]), which escaped even from comprehensive post-silicon validation.

To this end, we present the second contribution of this thesis, which describes a post-silicon self-checking validation method that unveils and detects rare bug scenarios in address translation caching arrays (ATCA) and is presented in [4]. ATCAs are among the most important structures for microprocessor functionality and performance. Escaped bugs in these arrays can lead to unpredictable system behaviors in the field. Using a comprehensive experimental study, we present and analyze rare bug scenarios and demonstrate the reasons why they are difficult to detect. Our goal is to unveil and detect difficult bugs particularly in ATCAs. Even if such bugs are excited by executing traditional validation tests, detecting them is unlikely due to the high possibility of masking during the execution of the validation test (as we demonstrate in subsection 2.7.3). For that reason, we present a novel method that detects such difficult bugs in CPU prototypes.

Our validation method is self-checking (like the one presented in the previous sections) and does not require any hardware instrumentation. We demonstrate that, in contrast to traditional validation tests, our proposed method effectively detects all of the injected bugs we created, according to bug scenarios shown later in Table 6. For our experimentation, we use the enhanced the Gem5 simulator with modern MMU Caches, as described before, to have a more detailed representation of a commercial and state-of-the-art microprocessor chip.

## **2.7 Impact of Bugs in Address Translation Caching Arrays**

### **2.7.1 Motivation**

Escaped bugs or faults in performance structures (other than address translation caching arrays), such as branch prediction units (BPU) or prefetchers, lead only to performance degradation and not to incorrect functionality [134]. However, a bug in an ATCA may affect both the performance and the correct functionality of the microprocessor. Massive published errata from microprocessor vendors demonstrated recently that severe escaped bugs in ATCAs can persist across generations (shown in Table 5) [113] [116] [125] [126] [127] [128] [129] [130] [131] [132]. Clearly, traditional post-silicon validation methods are not adequate to detect specific kinds of bugs.

Table 5 lists the most significant hardware bugs that escaped to the market according to official errata documents published from major microprocessor vendors, with a short description of the bug. The last column classifies each bug into a bug scenario described in subsection 2.7.3. The last decade has seen specific types of bugs that affect microprocessor functionality, which persist across newer microprocessor generations. Our experimental study demonstrates and explains why these bugs are difficult to detect.

### **2.7.2 Evaluating the Impact of Bugs in Address Translation Caching Arrays**

It is essential to study the behavior of the microprocessor during the execution of actual workloads to gain a deeper understanding of rare and difficult bugs in ATCAs, and the conditions that prevent these bugs (shown in Table 5) from being discovered during post-silicon validation. For this experimental study, we employed the benchmarks of MiBench suite [135] and the Gem5 simulator [120] in x86-64 mode, which was enhanced with MMU caches [122].

MiBench programs have been used in reliability studies and have significant similarities to SPEC CPU2006 [141] benchmarks in terms of instruction mix and throughput [208], and thus, they are essential programs for these studies. The Gem5 simulator was configured for our experiments to have 64 Instruction TLB (ITLB) entries, 64 Data TLB (DTLB) entries, 2 Page Map Level 4 (PML4) cache entries, 8 Page Directory Pointer (PDP) cache entries and 32 Page Directory (PD) cache entries (for data and instructions). These sizes are based on an Intel Core i7 microprocessor [121]. Our experiments are



**Table 5: Published ATCA-related bugs from official errata sheets.**

Bug Description	Vendor / Year	Phenomenon
Stale data in processor translation cache may result in hang [116].	Intel 2008	Ignored Invalidation
Possible use of stale translations even after software has performed a TLB flush [125] [126] [127].	AMD 2008 - 2011	Ignored Invalidation
The processor may use stale linear addresses translations [128] [129] [130] [131].	Intel 2015, 2016	Ignored Invalidation
The processor may store an incorrect value into bits of 11:0 of linear address field [130] [131].	Intel 2015, 2016	False Hit or False Miss
Some instructions or task switches may not completely invalidate the processor translation cache [116].	Intel 2008	Ignored Invalidation
Stale data may be loaded into the processor's TLB and used for memory operations [116].	Intel 2008	False Mapping

based on bug injection in random clock cycles and in random entry fields, to all available entries of each ATCA structure.

In all ATCAs, there are three different fields for each entry: the virtual address field, which is practically the tag; the data field, with physical address and attributes; and the valid bit, which is cleared upon a requested invalidation, usually when a context switch or a page table modification occurs. For each benchmark, we injected 680 different bugs in the virtual address field (2.6% error margin for 99% confidence level) and 680 different bugs in data field (5% error margin for 99% confidence level). For the valid bit, we injected 65 bugs on average; this depended on the number of invalidations requested by each program's execution.

For the valid bit case, we ran experiments with one benchmark assigned to one core, to study the invalidations due to page table modifications for a single process. Furthermore, we ran two benchmarks also assigned to one core, to force multiple context switches from different processes during the whole execution. Each process is given a fixed time quantum from the Linux kernel scheduler, so each benchmark runs concurrently with another benchmark. The scheduling algorithm depends on the kernel. For our experiments we used the Linux kernel 2.6.22 with  $O(1)$  scheduler. For all (single and pairs of) benchmarks, we counted the total amount of invalidations ( $N$ ) that occur during the execution and then executed each benchmark  $N$  “buggy” times. Each individual “buggy” execution affects only one invalidation in one of the ATCAs. The same procedure was repeated for each different ATCA, one at a time.

To trigger as many potential scenarios as possible in ATCAs, executions for all entry fields contain as much diversity as possible, i.e., we chose the pairs of benchmarks to have similar execution times. This helped ensure that the requested invalidations

occurred from context switches between user processes. For each experiment, we recorded if the bug affects the correct program's outcome (*affected*) or not (*masked*).

We concluded with the following categorization of bug scenarios shown in Table 6. Each bug scenario describes a divergence from the expected behavior. The diverging behavior is attributed to a functional or an electrical bug and can appear in any ATCA structure. The last column of Table 6 describes an example of an activation criterion for each bug scenario.

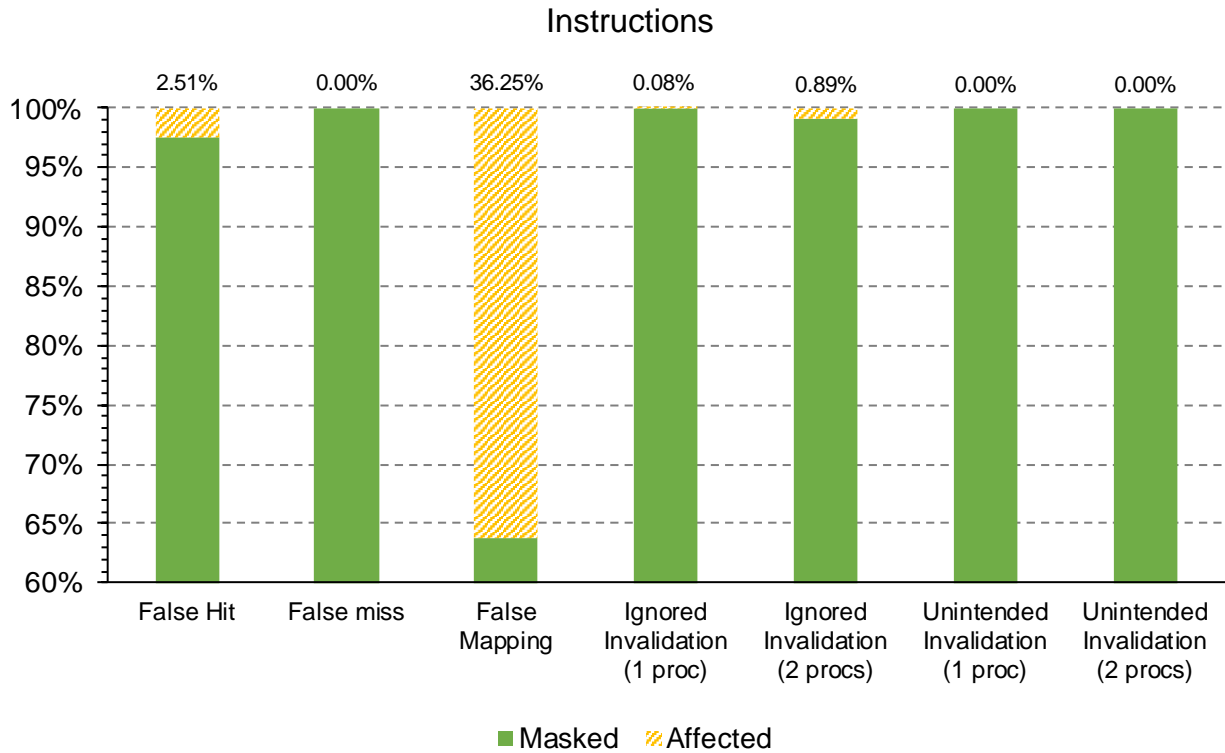
### 2.7.3 Findings and Observations

Although bugs in ATCAs may exist in silicon and can be unveiled during the execution of validation tests, overall it is difficult to detect specific kinds of bugs. This occurs because they may not affect the correct functionality of the executed programs, and in most cases, they do not provide divergences in the states of the silicon. If a bug takes place in an ATCA during the execution of the validation test and gets masked by a valid entry before its access (masked), the silicon output does not differ from the “bug-free” reference model. As a result, the comparison will succeed, and the bug will remain in the silicon prototype.

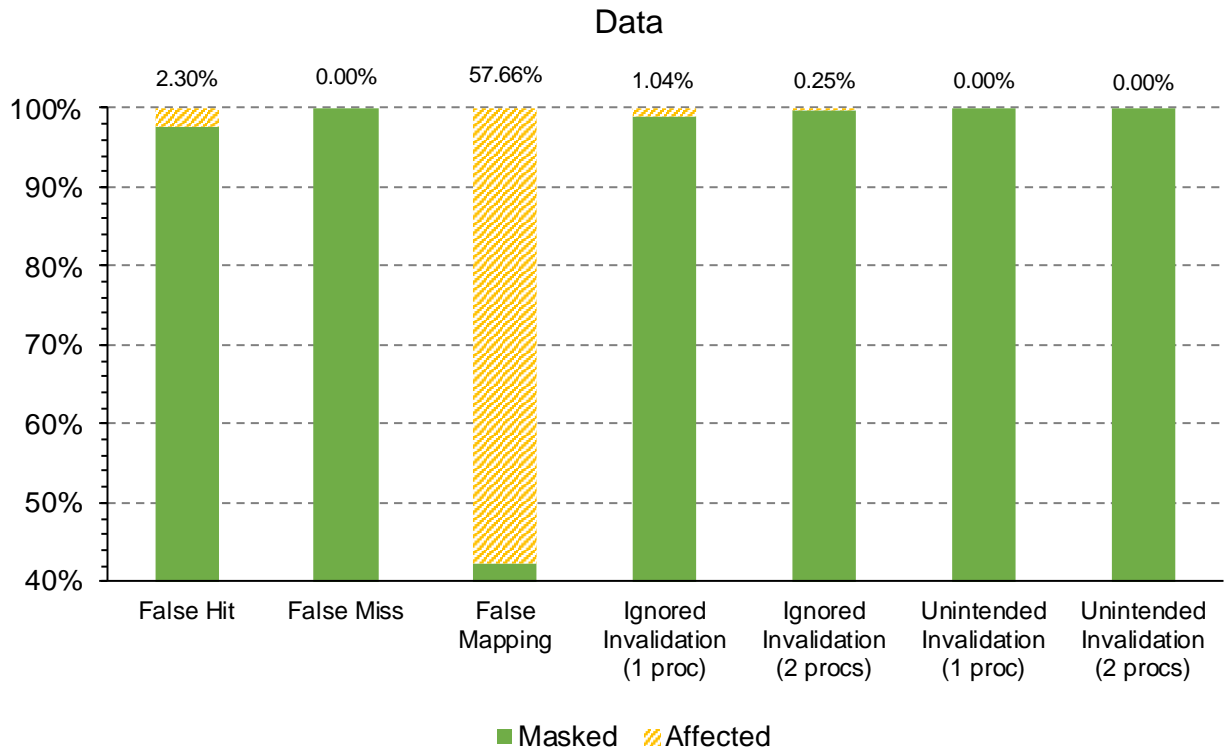
Figure 21 and Figure 22 present the results of our experiments trying to quantify the “difficulty” of detecting these bugs per category. The percentage shown above each stacked bar refers to the probability that a bug can affect the output of the program. The results are categorized into the distinct bug scenarios shown in Table 6. These led us to the following observations: *false mappings* have a high probability of influencing the correct program output, thus, they are likely to be detected by traditional validation tests. On the other hand, the most difficult bugs in ATCAs are the *ignored invalidations* and *false hits*. This is due to their low probability of affecting the correct program outcome, as

**Table 6: Bug Scenarios that cause divergence from the correct behavior and sample activation criterion that can unveil the bug.**

Bug Scenario	Phenomenon	Activation Criterion
A requested translation by the processor matches a wrong cached entry of an ATCA	False Hit	The VA field gets corrupted before its access
A requested translation by the processor matches a correct cached entry of an ATCA but its data field is incorrect	False Mapping	The data field gets corrupted before its access
A requested translation by the processor matches a stale cached entry of an ATCA	Ignored Invalidation	A requested invalidation fails to invalidate that entry
A requested translation by the processor does not match a correct cached entry of an ATCA	False Miss	The VA field gets corrupted before its access
A requested translation by the processor does not match a correct cached entry of an ATCA	Unintended Invalidation	An invalidation occurs to that entry without being requested



**Figure 21: Masked vs. Affected entries in ATCAs for instruction pages show the probability of a bug to affect program outcome.**



**Figure 22: Masked vs. Affected entries in ATCAs for data pages show the probability of a bug to affect program outcome.**

well as the *unintended invalidations* and *false misses*, which can only result in *performance degradation and not incorrect output*. The latter phenomena have zero probability of affecting the program outcome. For example, when an *unintended invalidation* occurs, the CPU will just miss the translation cache (the correct entry exists, but its valid bit is 0), and will walk the page table in memory to fetch the correct translation.

When a bug affects the tag (the Virtual Address field of an ATCA – *false hit* and *false miss*) or the valid bit (*ignored* and *unintended invalidation*) of an entry, it is unlikely to be detected by a traditional validation test. In most cases, the “buggy” entry gets replaced by other valid entries before their next access (masked). The results shown in Figure 21 and Figure 22 demonstrate the most difficult bugs in ATCAs and why major microprocessor vendors publish errata reports with these kinds of bugs.

## 2.8 Generating Validation Tests to Unveil Difficult Bugs in ATCAs

### 2.8.1 The Main Concept

Modern architectures allow multiple processes to execute as if they own the entire virtual address space. More specifically, several processes could write to a memory location at the same virtual address without influencing each other’s results. This is done by virtualizing the address space. Assume, for example, processes P1 and P2 both generate a store to memory location at each one’s virtual address 0x0123XXX. The processor, assisted by the operating system, performs additional adjustments to the address, and maps it to a physical address 0xAABBXXX for P1, and to 0x1122XXX for P2.

The most significant bits of the virtual address (VA) represent the page (called virtual page number - VPN). Those of the physical address (PA) represent the corresponding frame in memory (called physical frame number – PFN). The last 12 bits of both the VA and the PA (in a 4KB page in this example; there is no limitation for other page sizes) represent the *Offset* into the page or the frame respectively, which does not participate in the address translation process. Therefore, different processes can have the same VAs, but, their corresponding physical addresses (PA) may be different. The exception is global pages, which are shared among several processes.

Consider, for example, two running processes on a system, processes P1 and P2, which have the mappings shown in Table 7. We assume for simplicity of the example that the addresses are 16-bits long instead of 64-bits in modern microarchitectures. Both processes have the same virtual address space, but different mappings to the physical memory.

Each validation test consists of two types of processes executed in a row:

- the Exerciser process, and
- the Checker process.

*Exercisers* and *Checkers* are executed on the same core; Exercisers that are executed in different cores can have shared pages (we take advantage of shared pages; the description shown in subsection 2.8.2). However, for the validation of a multicore system, we execute the pair of these processes on all the available cores. Exerciser accesses  $N$  different virtual pages, which are cached in ATCAs, and stops execution. The Checker is then invoked to the same core, so a context switch occurs. Upon the context switch, all entries of ATCAs must get invalidated (except for shared entries), according to the instruction set architecture specification.

The Checker accesses the same  $N$  virtual pages (the accessing order of VPNs does not matter). Given that the Checker accesses the same VAs as the Exerciser, after the context switch, the Checker misses all ATCAs (in a bug-free execution) and must access

**Table 7: Example Page Table Entries (Instruction and Data Pages and Frames) for Processes P1 and P2.**

P1			P2	
Page Type	VPN	PFN	VPN	PFN
Instruction Pages/Frames	0x0123	0xAABB	0x0123	0x1122
	0x1234	0xBBCC	0x1234	0x2233
	0x2345	0xCCDD	0x2345	0x3344
	0x3456	0xDDEE	0x3456	0x4455
Data Pages/Frames	0x4567	0xEEFF	0x4567	0x5566
	0x5678	0xFFAA	0x5678	0x6677
	0x6789	0xFFBB	0x6789	0x7788
	0x7890	0xFFCC	0x7890	0x8899

completely different physical addresses. For example, if an invalidation fails to a DTLB entry (*ignored invalidation*), the Checker will inevitably hit the DTLB, and so it accesses the data of the Exerciser process. If, on the other hand, an invalidation fails to the ITLB, the Checker will execute part of the Exerciser process code.

The same example applies to all other translation caching arrays and entry fields, but with different granularity. The  $N$  parameter is the value of maximum entries of the translation cache with the largest size being validated. For example, if we validate the TLB, then  $N = 64$  (in our setup), and if we validate the PD cache, then  $N = 32$ , and so on (subsection 2.8.2). This procedure continues as is but with a different set of virtual-to-physical address pairs for each core to achieve the desired validation coverage for all ATCAs in the microprocessor.

## 2.8.2 Exercisers

The key idea behind a comprehensive post-silicon validation is to create a complex combination of validation tests with multiple interleaved execution flows and address patterns that exercise the ATCAs, to expose potential corner-case bugs. This is the fundamental role of the Exerciser process. Each of the ATCAs caches both instructions and data in different ways. For effective validation and exercise of all available structures and logic of ATCAs, we present the following exercising methodology.

To exercise instruction entries of an ATCA (e.g., Instruction TLB), the validation test uses conditional or unconditional branches, jumps, and calls to  $N$  different instruction pages. The allocation of instruction entries for ATCAs is achieved by hopping across code page boundaries. Given a wide range of different address patterns and repetitions that are produced, ATCAs can be successfully stressed [100]. The allocated entries for instruction pages are then used by subsequent loads and stores to  $N$  different data pages. These pages are randomly inserted after a jump to a new instruction page, to exercise the data entries of an ATCA (e.g., DTLB).

The Exerciser performs at least one Store operation for each data page; the value stored in physical memory is its own virtual page number. Further, the process ID (PID) (or any

other number that uniquely characterizes the process) is also stored along with the VPN to the same word, if it is a global page. PID information is used to distinguish which process accesses the shared memory location. For example, if the VA = 0x1234XXX is mapped to the PA = 0xAABBXXX, and the current process ID is PID = 001, the value stored (datum) in the physical memory is 0x012345001. As we described above, each VA contains the offset of the page to the least significant bits (e.g., 12 bits for 4KB pages); this offset does not take place in the ATCAs, so there is no need to validate it. As a result, the last 12 bits of the datum word that is stored in memory can be used to store the PID.

The Exerciser also records the number of translation cache hits and misses that occur during its execution by reading the Performance Monitoring Counters (PMC) of the microarchitecture (see for example ANNEX I which presents all the available PMC for ARMv8 architecture). When the Exerciser process begins, it declares two global variables to be shared between the two processes. These variables store the translation cache hits and misses before the Exerciser's execution. At the end of the exercising code, it again reads the PMCs to record; these are the total translation cache hits and misses of the Exerciser ( $N$  misses, 0 hits). The Exerciser is executed as is once more, so the number of misses in the second execution must be 0, and hits must be  $N$  in a bug-free execution.

The role of the Exerciser is twofold:

1. to exercise and stress the ATCAs at all levels (instructions and data) in order to expose as many potential corner cases as possible by accessing a wide range of addresses and address patterns, and
2. to stamp its own physical frames in memory (data and instructions). This is achieved by:
  - a. Using the same virtual page numbers for instruction and data pages as its "partner" Checker process.
  - b. Storing its virtual page number to the corresponding data frame in memory.
  - c. Storing a special number in memory that uniquely identifies itself in the last 12 bits of the datum.
  - d. Recording the total translation cache hits and misses that occur during its execution.

These conditions guarantee that if a bug in ATCAs of any level occurs, the Checker process either executes part of the Exerciser's code, or recognizes that it reads data from the Exerciser. Further, PMCs are used to recognize if a bug affects only performance, for example, unintended invalidations or false misses (during the second execution).

### 2.8.3 Checkers

The Checker process is used to test whether there is a divergence from the correct (bug-free) behavior, while the Exerciser exercises and stresses the ATCAs. It is essential for the Checker to be realized as a different process (so that a context switch occurs) in order to validate the valid bit, and as a simple procedure (in the same process with the Exerciser) at the end of the exercising code (without a context switch) to validate the tag and data fields. There are also alternative techniques that can cause a change to the state of the valid bit, although they are more complicated. The Checker performs load operations to the last  $N$  virtual pages (data and instruction pages), which have been accessed by the Exerciser, and then it checks if the ATCAs operate correctly. Specifically, the Exerciser previously performed  $N$  stores to the memory to  $N$  different data pages, and the Checker performs  $N$  loads from the same VAs. Because the virtual-to-physical address mappings are different between any two processes, the fetched datum must not

be equal to the Checker's current VA. If the fetched datum has the same VA and it is a global page, the Checker also performs a comparison with its PID to the last 12 bits. When the Checker validates the data pages, a mismatch in a comparison indicates the existence of a bug.

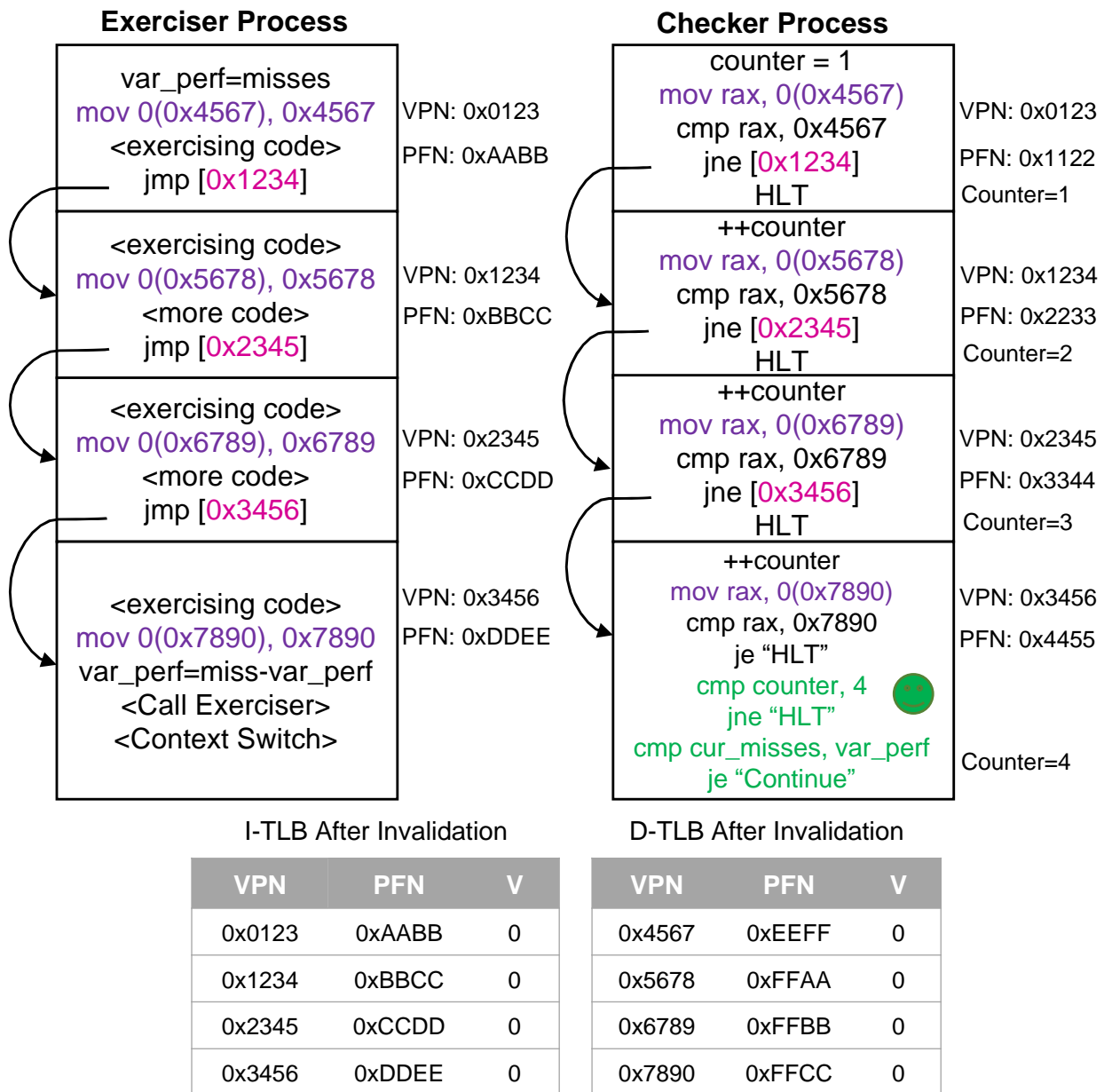
In the validation of instruction pages, consider, for example, that the Exerciser's code (instruction pages) is located in the physical frames (PFN) shown in Table 7. Although the Exerciser and Checker processes have the same VAs, their mappings to the PA are different. When the Checker is invoked after the Exerciser's execution and a bug occurs in ATCAs, it is likely that the Checker will execute part of the code of the Exerciser, if, for example, there was a stale entry in the ITLB. To get knowledge of this, the Checker also has a counter increased by one for each jump to the next instruction page. At the end of the Checker's execution, this counter must have a value equal to  $N$ . If, for example, the counter equals to  $N-1$ , the Checker can recognize that one of its instruction pages has not been executed. Further, the Checker records the total number of translation cache hits and misses from PMCs, as the Exerciser.

Apart from a context switch, which automatically invalidates the ATCAs (by writing the CR3 register in x86 – see details in subsection 2.1.3), there are also implicit instructions that invalidate the ATCAs. This is usually when a page table entry modification is done by the kernel. To avoid a context switch that will again invalidate the ATCAs, the Checker is not a process; it is a procedure at the end of the Exerciser and thereby validates the functionality of such instructions. Similarly, the Checker is used as a procedure to validate false hits and false mappings. When the Checker is a procedure, there is also a need to use different conditions for the comparisons: the *jne* instructions in the code are changed to *je* instructions. Assume, for example, that the Exerciser stops its execution, and the code of the Checker (realized as a procedure now) is executed (without invalidation). Practically, the Checker will load the data from the same VAs to which the Exerciser previously stored its VPNs. If a fetched datum is not equal with its VA, a false hit or a false mapping occurred. The limitation here is that the debug engineers cannot distinguish whether the bug is due to a false hit or due to a false mapping.

#### 2.8.4 Example of the Proposed Validation Flow

In this section, we present some examples of the proposed validation flow (a bug-free flow, an ignored invalidation in ITLB, and an ignored invalidation in DTLB), which consist of the two processes, the Exerciser and the Checker. Assume, for example, that we validate the data and instruction TLBs, and there is no bug in both of them, so the validation test should pass. Moreover, we assume for simplicity that the addresses are 16-bits long (plus a 12-bits offset), the size  $N$  of the TLBs is 4 entries, and the main memory has been initialized with zero values before the validation process begins. We also use the address mappings of Table 7 as a reference.

As shown in Figure 23, the Exerciser process is first invoked for execution. Each block represents an instruction page in memory, with its virtual-to-physical address mappings shown on the right. Each instruction page contains two main instructions: a *store* operation, and a *jump* to stride across instruction pages. Data and instruction caching arrays are exercised concurrently. Each store operation stores its own VPN. For example, the value 0x4567 is stored to VA = 0x4567XXX. Then, there is a jump to the next instruction page, in which the same procedure takes place. The last instruction page records the total number of hits and misses of the Exerciser code, and the Checker is invoked. Upon a context switch, the ATCAs must get invalidated.



**Figure 23: Validation test flow example – Pass; no bug detected.**

After the context switch, as we can see in the TLB entries, all entries are invalidated correctly (all valid bits are zero). Although the Checker accesses the same virtual addresses as the Exerciser, it will miss the TLBs, so the Checker passes the test. The first Checker instruction page initializes a counter, performs a load from the VA = 0x4567XXX and compares the fetched datum. Given that the DTLB is correctly invalidated, this virtual address will miss the TLB, so the fetched datum that returns will be different from the previous store of the Exerciser (practically it will be zero due to the initialization of physical memory with zero values performed before the validation process begins). The process continues with hops across the same virtual addresses of instruction pages as the Exerciser. On the last page, there are also two more comparisons: with the counter (to detect if a bug occurs in an instruction page) and with the total number of cache hits and misses (to detect if there is a bug that affects only performance, such as false miss and unintended invalidation). Given that there is no error in the procedure, the validation test passes.



In an execution with a bug, consider, for example, that an ignored invalidation occurs in ITLB, after the execution of the Exerciser process, as shown in Figure 24. When the Checker attempts to jump to the instruction page with VA = 0x2345XXX, due to the ignored invalidation of that entry in ITLB, it will hit the ITLB; hence, it will execute the corresponding code of the Exerciser process and the counter (that exists in the hashed block) will not be increased. The hashed block in the Checker process is not executed, so the comparison of a counter at the end of the Checker does not match. Therefore, the validation test fails due to the ignored invalidation of the ITLB.

Assume now that an ignored invalidation occurs in DTLB, as shown in Figure 25. As a result, when the Checker process attempts to load a datum from VA = 0x5678XXX, it will go through the cached translation that exists in the DTLB (stale entry), so the datum from PA = 0xFFAAXXX will be fetched. As we can see in the physical memory state, the Exerciser process has stored its virtual addresses as data values into the memory. In a bug-free execution, when the Checker attempts to read data from VA = 0x5678XXX, it must read a zero value (which is the initial value). Instead, due to the ignored invalidation

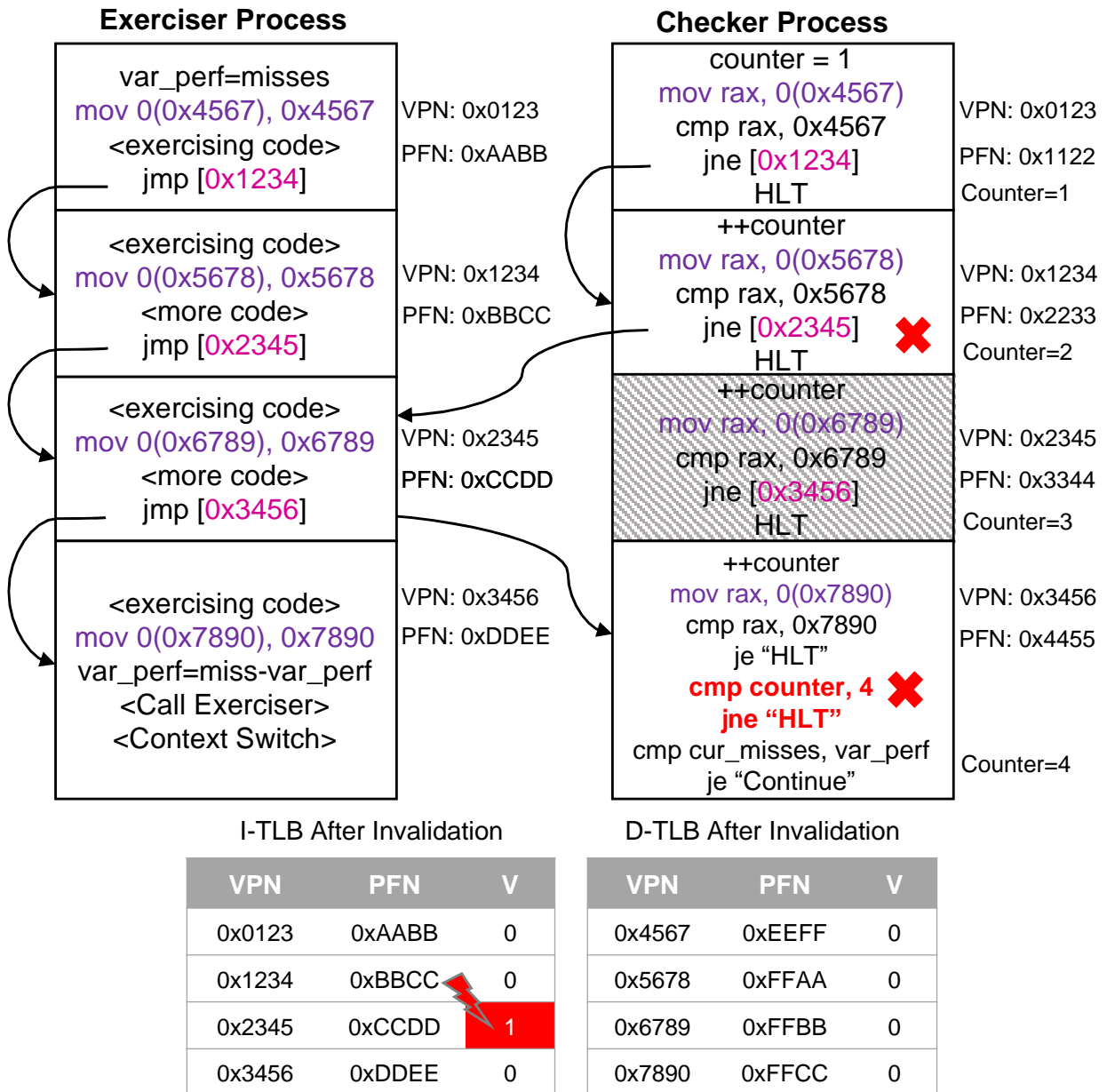


Figure 24: Validation test flow - Ignored Invalidation in ITLB.

in DTLB, it incorrectly reads the value 0x5678. When the Checker process loads the datum stored in VA = 0x5678XXX, it will hit the DTLB due to the ignored invalidation, and the fetched datum will be equal to its virtual address. This means the Checker process reads data from the Exerciser's address space, and the validation test fails. The grayscale hashed blocks in Figure 25 are not executed at all.

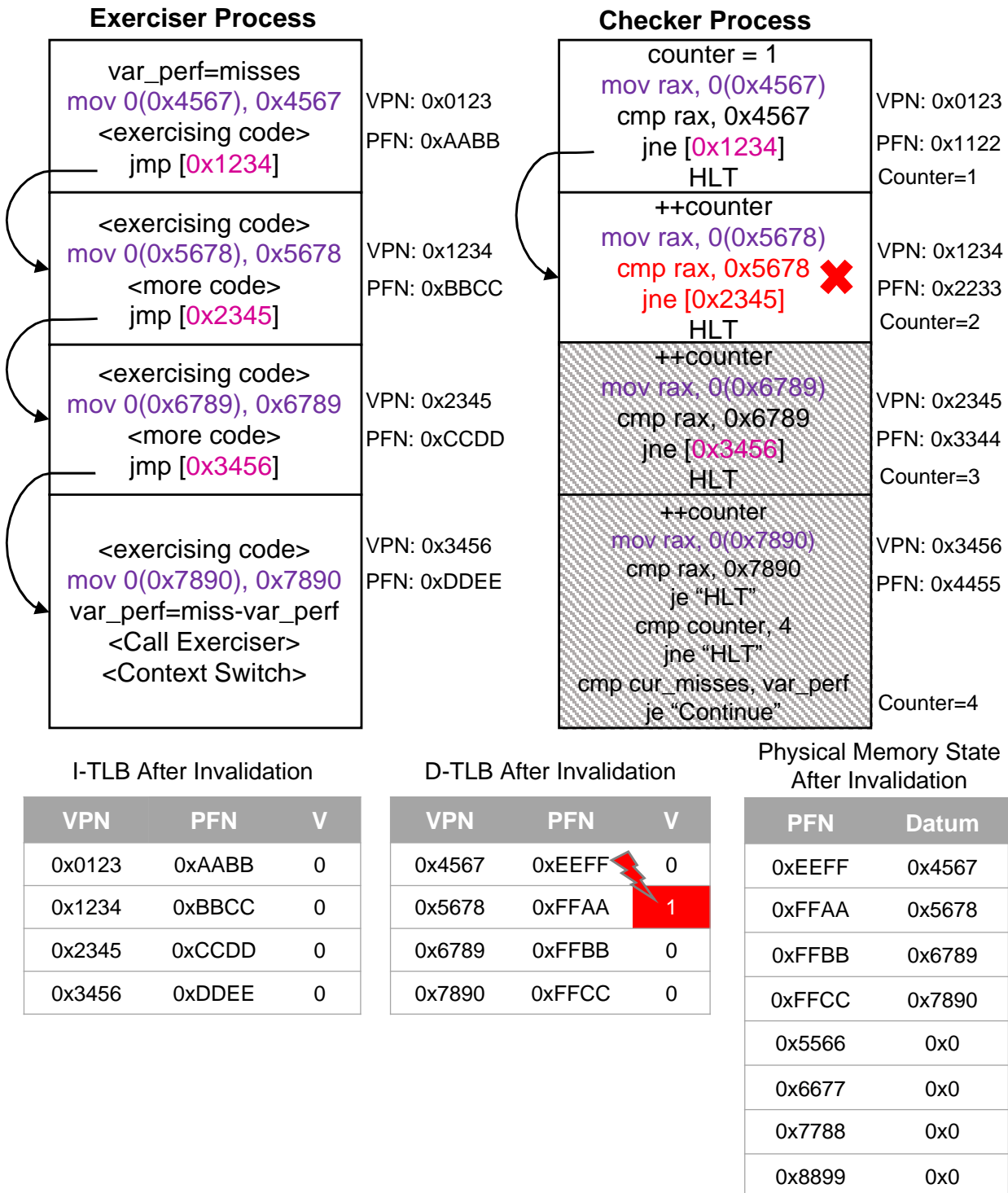


Figure 25: Validation test flow - Ignored Invalidation in DTLB.

## 2.9 Experimental Evaluation

### 2.9.1 Experiment Setup

To evaluate the effectiveness of the proposed post-silicon validation method, we employ random-generated validation programs, each targeting a different bug scenario shown in Table 6. We implemented the test generation using a mix of assembly-level and C programs. To compare the proposed method with a traditional post-silicon validation flow, we developed assembly-level validation programs; these have a number of executed instructions that is comparable to the validation tests of our method.

We configured the Gem5 simulator with 64 ITLB entries, 64 DTLB entries, 2 PML4 cache entries, 8 PDP cache entries, and 32 PD cache entries (for data and instructions); however different sizes do not provide limitations for the proposed method. We modified the Gem5 simulator to support the injection of random, non-deterministic bugs in the address translation caching arrays of the x86-64 model, by covering as many erroneous conditions that may occur in prototype chips as possible. We also modeled the ATCA-related errata described in Table 5. With these two approaches, our simulator resembled a realistic “buggy” microprocessor prototype chip. We also developed a custom minimal kernel to create a realistic bare-metal post-silicon validation “environment” and execute our validation programs.

Bare-metal modeling on Gem5 was a major part of our development work, enabling it to resemble the real hardware infrastructure. Our kernel initiates all the procedures required to set up the paging interface needed to operate in Long Mode (x86-64). It also offers a fundamental infrastructure for the virtual memory mappings of two different processes.

Figure 26 summarizes the proposed post-silicon validation flow as it is modeled in the Gem5 simulator. This procedure can also be integrated in a real prototype chip without modifications in the kernel or validation tests. The kernel is responsible for initiating and managing the processes and their address spaces needed for the validation tests. The host (simulator) generates bugs (by alternate random bits in each field) according to each phenomenon described in Table 6, and applies the bug into a random entry of ATCAs. Furthermore, it records the total amount of translation cache hits and misses. In a real microprocessor chip this is done by reading the performance monitoring counters. At the end of execution, the validation test reports whether the test succeeded or failed.

### 2.9.2 Results

To evaluate the effectiveness of the proposed method, we performed massive injections of random bugs. Each validation test consisted of 200K committed instructions. Table 8 presents the total number of bugs injected in ATCAs applied to each different entry, as well as the ATCA-related bugs presented in official errata sheets. These bugs were applied to all available entries of ATCAs for four different patterns of virtual address spaces.

Figure 27 summarizes the results of our bug injection experiments. Our proposed methodology detected all 2559 bugs injected into the Gem5 simulator (bug coverage 100%). On the other hand, we compared the proposed method to the end-of-test checking techniques, which detected only 255 injected bugs (bug coverage 9.97%). This difference is due to the limitations of these techniques in the validation flow, given that they check if the output of the DUV is equivalent to a golden reference output. In most of the cases (2304 out of 2559 bugs), although a bug is excited, it does not affect the output. For example, as we presented in subsections 2.7.2 and 2.7.3, *false mappings* are easier for a traditional validation test to detect, in contrast to other bug scenarios. Consequently, *false mappings* have significantly higher detection rates.

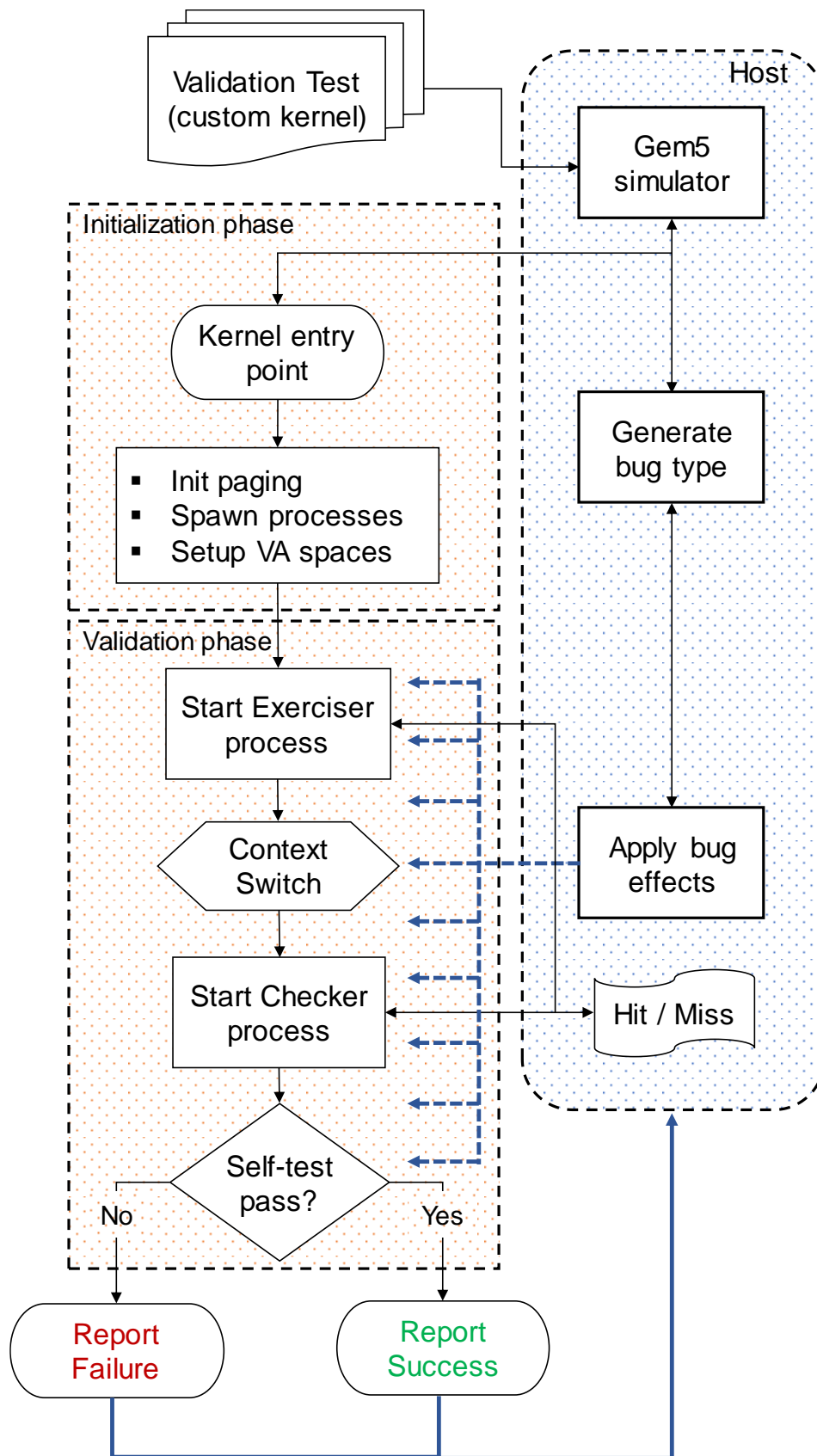
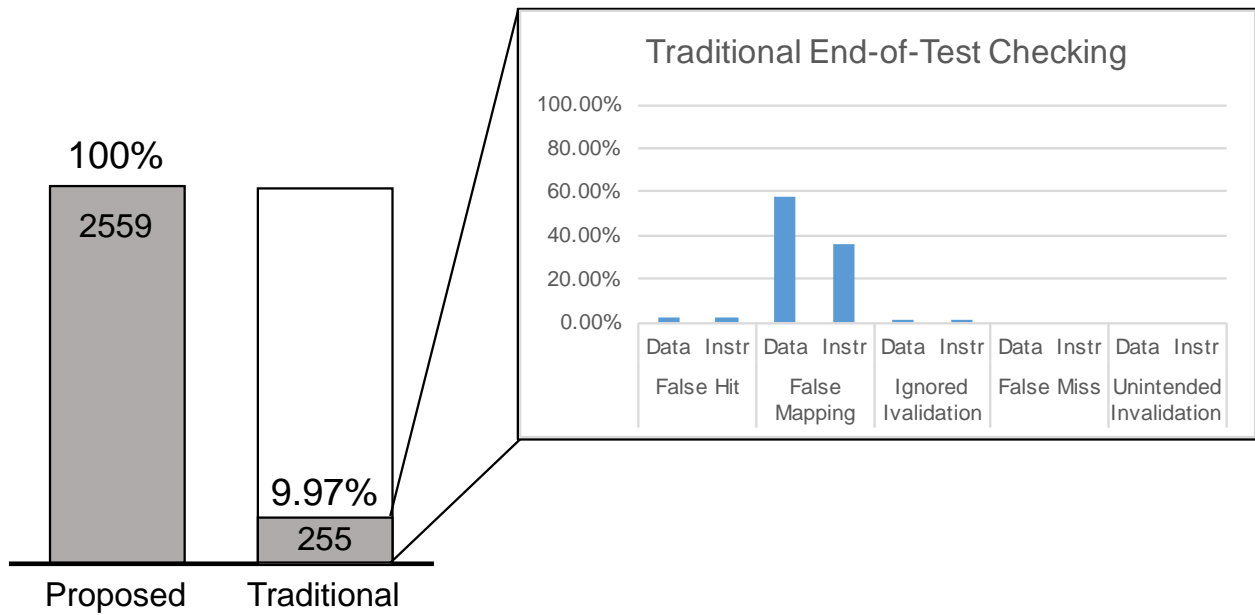


Figure 26: Proposed post-silicon validation flow and the experiment setup.

**Table 8: Number of different type of bugs for each phenomenon.**

Phenomenon	Number of Bugs	
	Random	Errata
False Hit / False Miss	1360	2
False Mapping	680	1
Ignored Invalidation	256	4
Unintended Invalidation	256	–
<b>Total</b>	<b>2552</b>	<b>7</b>

**Figure 27: Bug coverage for the proposed method vs. traditional end-of-test checking techniques.**

## 2.10 Related Work

Recent studies have proposed a comprehensive analysis and classification of bug models for different microprocessor hardware structures [118] [123] [124], but not for the address translation mechanisms. Recent approaches have employed the inherent ISA diversity and reversibility to generate self-checking silicon validation programs for microprocessor cores [212] [219] and other solutions that also target bugs inside the processor cores [117]. These methods alone are not sufficient for detecting the majority of bugs in ATM because the self-checking property is achieved by validation program modifications at the instruction level.

Other works contribute in difficult post-silicon validation issues, such as reducing error detection latencies in in-core bugs and cache consistency, but all of them require hardware modifications and neither method proposes solutions for the critical mechanism of address translation [117] [220]. The work presented in [213] combines self-checking validation programs (which can be generated by any of the previous approaches) with deconfigurable hardware structures to enhance root cause analysis of bugs inside the core, but it is also insufficient for detecting bugs in ATM.



### 3. Measuring Voltage Guardbands of Server-Grade ARMv8 CPU Cores

During chip fabrication, process variations can affect transistor dimensions (length, width, oxide thickness etc. [81]) which have a direct impact on the threshold voltage of a MOS device<sup>5</sup> [82]. As technology scales, the percentage of these variations compared to the overall transistor size increases and raises major concerns for designers, who aim to improve energy efficiency. Devices variation during fabrication known as *static* variation and remains constant during the chip lifetime. On top of that, transistor aging and *dynamic* variation in supply voltage and temperature, caused by different workload interactions, is also of primary importance. Both static and dynamic variations lead microprocessor architects to apply conservative guardbands (operating voltage and frequency settings) to avoid timing failures and guarantee correct operation, even in the worst-case conditions excited by unknown workloads or the operating environment [5] [83]. However, these guardbands impede the power consumption. To bridge the gap between energy efficiency and performance improvements, several hardware and software techniques have been proposed, such as Dynamic Voltage and Frequency Scaling (DVFS) [77]. The premise of DVFS is that the microprocessor's workloads as well as the cores' activity vary. Voltage and frequency-scaling during epochs where peak performance is not required enables a DVFS-capable system to achieve average energy-efficiency gains without affecting peak-performance adversely. However, energy-efficiency gains are limited by the pessimistic guardbands.

Revealing and harnessing the pessimistic design-time voltage margins offers a significant opportunity for energy-efficient computing in multicore CPUs. The full energy savings potential can be exposed only when accurate core-to-core, chip-to-chip, and workload-to-workload voltage scaling variation is measured. When all these levels of variation are identified, system software can effectively allocate hardware resources to software tasks matching the capabilities of the former (undervolting potential of the CPU cores) and the requirements of the latter (for energy or performance). Although characterization studies for CPUs and GPUs have been presented recently [5] [6] [7] [8] [9], they primarily focus on coarse-grained identification of the  $V_{min}$  values, i.e., the voltage level at which no type of anomaly is observed in program execution of a particular core. Furthermore, these studies focus primarily on x86 and POWER-series enterprise-class server systems, whose summary is shown in Table 9; studies on GPU chips have been reported as well.

**Table 9: Summary of studies on commercial chips.**

ISA	Processor	Technology	Reference
POWER 7 / 7+	IBM Power 750, 780	45 / 32 nm	[6] and [84]
IA-64	Intel Itanium 9560	32 nm	[7] and [8]
x86-64	Intel i7-3970X, i5-4200U	32 / 22 nm	[157]
Nvidia Fermi / Kepler	GTX 480, 580, 680, 780	40 / 28 nm	[9]

<sup>5</sup> The threshold voltage, commonly abbreviated as  $V_{th}$ , of a field-effect transistor (FET) is the minimum gate-to-source voltage  $V_{GS(th)}$  that is needed to create a conducting path between the source and drain terminals. It is an important scaling factor to maintain power efficiency.

In this chapter, we present the third contribution of this thesis, which is a detailed system-level voltage scaling single-core characterization study for ARMv8-based CPUs manufactured in 28nm. The study's backbone is a fully automated system-level framework built around Applied Micro's (APM) X-Gene 2 micro-server. The automated infrastructure aims to increase the throughput of massive undervolting campaigns that require multiple benchmarks execution at several voltage supply levels of all individual cores. The automated characterization process requires minimal human intervention and records all possible abnormalities due to undervolting: silent data corruptions (SDC, e.g., program output mismatches without any hardware error notification), corrected errors, uncorrected (but detected) errors (provided by Linux EDAC driver [136]), as well as application and system crashes [10].

Towards the formalization of the behavior in undervolting conditions we also present the definition of a simple consolidated function; the *Severity function*. Severity function aggregates the effects of reduced voltage operation in the cores of a multicore CPU by assigning values to the different abnormal observations. The lower the voltage level, the higher the value of the severity function. The severity function assists an undervolting classification of the cores of a CPU chip for a given benchmark: different core, benchmark and voltage values lead to different severity patterns, some with an abrupt increase to the severity (e.g., the benchmark keeps executing correctly until a voltage level at which the system crashes), while others have a "smooth" severity increase while voltage is reduced (the system remains responsive throughout a range of voltage values but it generates ECC errors or produces SDCs). The fine-grained analysis of the behavior of the machine using the severity function can assist energy efficiency decisions for task-to-core allocation by the system software.

Our comprehensive characterization for ARMv8-based multicore CPUs confirms that a different microarchitecture, circuit design or manufacturing technology exhibits different abnormal behavior when operating beyond nominal voltage conditions. Understanding the behavior in non-nominal conditions is very important for making software and hardware design decisions for improved energy efficiency that preserves correctness of operation. The characterization modeling of our study can be effectively used to support design and system software decisions to harness voltage margins and thus improve energy efficiency while preserving operation correctness.

### 3.1 System Architecture

For the study described in this chapter we use Applied Micro's (APM – now Ampere Computing) X-Gene 2 microprocessor for all of our experiments and results. The X-Gene 2 microprocessor chip consists of eight 64-bit ARMv8 cores. It also includes the Power Management processor (PMpro) and Scalable Lightweight Intelligent Management processor (SLIMpro) to enable breakthrough flexibility in power management, resiliency, and end-to-end security for a wide range of applications. The PMpro, a 32-bit dedicated processor provides advanced power management capabilities such as multiple power planes and clock gating, thermal protection circuits, Advanced Configuration Power Interface (ACPI) power management states and external power throttling support. The SLIMpro, 32-bit dedicated processor monitors system sensors, configure system attributes (e.g., regulate supply voltage, change DRAM refresh rate etc.) and access all error reporting infrastructure, using an integrated I2C controller as the instrumentation interface between the X-Gene 2 cores and this dedicated processor. SLIMpro can be accessed by the system's running Linux Kernel.



X-Gene 2 has three independently regulated power domains (as shown in Figure 28):

1. **PMD (Processor Module) – red hashed line:** Each PMD contains two ARMv8 cores. Each of the two cores has separate instruction and data caches, while they share a unified L2 cache. The operating voltage of all four PMDs together can change with a granularity of 5mV beginning from 980mV. While PMDs operate at the same voltage, each PMD can operate in a different frequency. The frequency can range from 300MHz up to 2.4GHz at 300MHz steps.
2. **PCP (Processor Complex)/SoC – green hashed line:** It contains the L3 cache, the DRAM controllers, the central switch and the I/O bridge. The PMDs do not belong to the PCP/SoC power domain. The voltage of the PCP/SoC domain can be independently scaled downwards with a granularity of 5mV beginning from 950mV.
3. **Standby Power Domain – golden hashed line:** This includes the SLIMpro and PMpro microcontrollers and interfaces for I2C buses.
4. Table 10 summarizes the most important architectural and microarchitectural parameters of the APM X-Gene 2 micro-server that is used in our study.

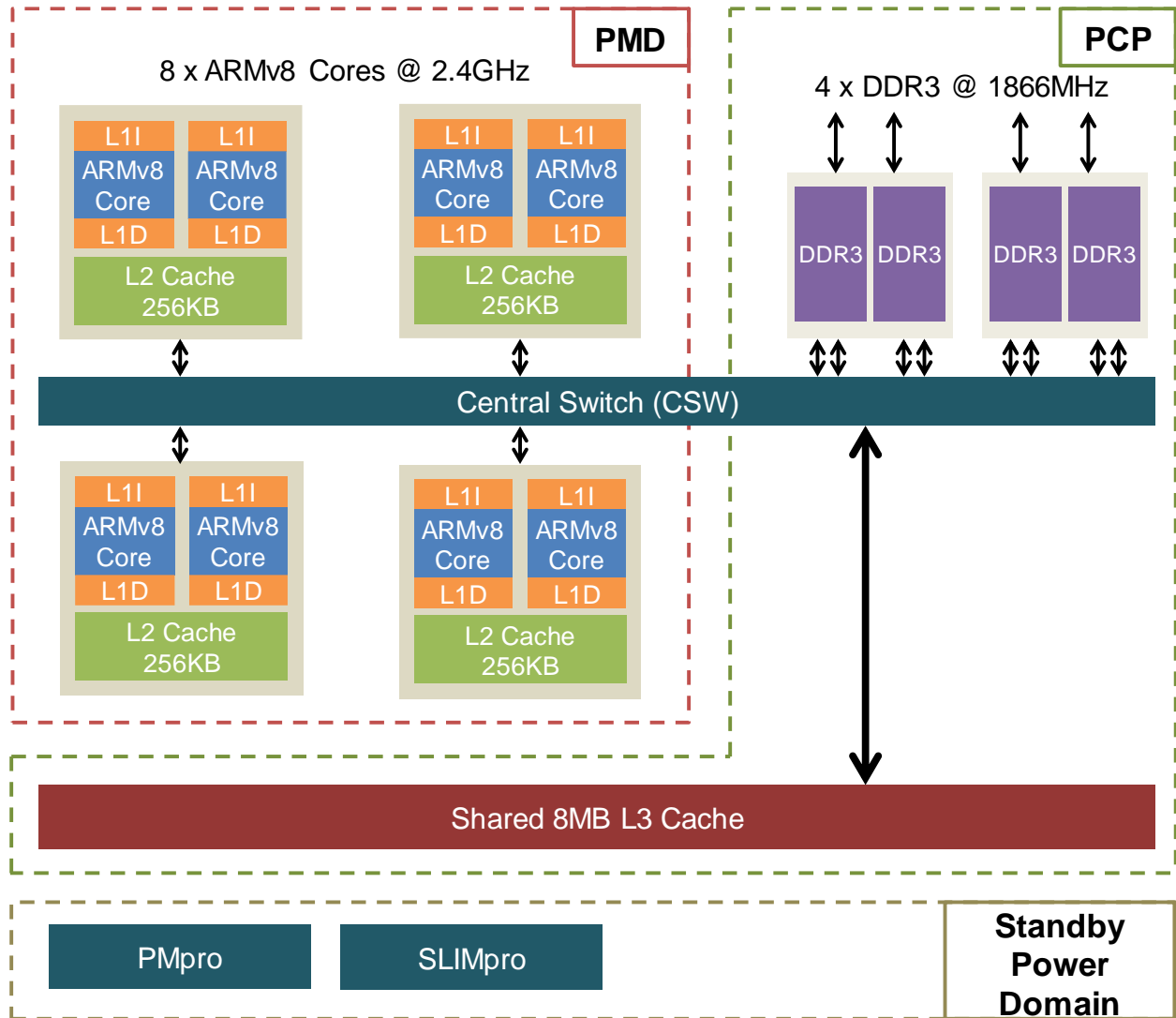


Figure 28: X-Gene 2 micro-server power domains block diagram. The outlines with dashed lines present the independent power domains of the chip.

**Table 10: Basic Characteristics of X-Gene 2.**

Parameter	Configuration
ISA	ARMv8 (AArch64, AArch32, Thumb)
Pipeline	64-bit OoO (4-issue)
CPU	8 Cores, 2.4GHz
L1 Instruction Cache	32KB per core (Parity Protected)
L1 Data Cache	32KB per core (Parity Protected)
L2 Cache	256KB per PMD (SECDED Protected)
L3 Cache	8MB (SECDED Protected)

### 3.2 Automated Characterization Framework Overview

The primary goals of the proposed framework are: (1) to identify the target system's limits when it operates at scaled voltage and frequency conditions, and (2) to record/log the effects of a program's execution under these conditions. The framework provides the following features:

- It compares the outcome of the program with the correct output of the program when the system operates in nominal conditions to record Silent Data Corruptions (SDCs),
- It monitors the exposed corrected and uncorrected errors from the hardware platform's error reporting mechanisms
- It recognizes when the system is unresponsive to restore it automatically,
- It monitors system failures (crash reports, kernel hangs, etc.),
- It determines the safe, unsafe and non-operating voltage regions for each application for all frequencies, and
- It performs massive repeated executions of the same configuration.

The automated framework (outlined in Figure 29) is easily configurable by the user, can be embedded to any Linux-based system, with similar voltage and frequency regulation capabilities, and can be used for any voltage and frequency scaling characterization study.

To completely automate the characterization process, and due to the frequent and unavoidable system crashes that occur when the system operates in reduced voltage levels, we set up a Raspberry Pi board connected externally to the X-Gene 2 board which behaves as a watchdog. The Raspberry is physically connected to both the Serial Port and the Power and Reset buttons of the system board to enable physical access to the system.

We discuss the several challenges that were taken into consideration for a solid development of such a framework.

**Safe Data Collection.** Given that a system operating beyond nominal conditions often has unexpected behaviors (e.g., file system driver failures), there is the need to correctly identify and store all the essential information in log files (to be subsequently parsed and analyzed). The automated framework was developed in such a way to collect and store safely all the necessary information about the experiments.

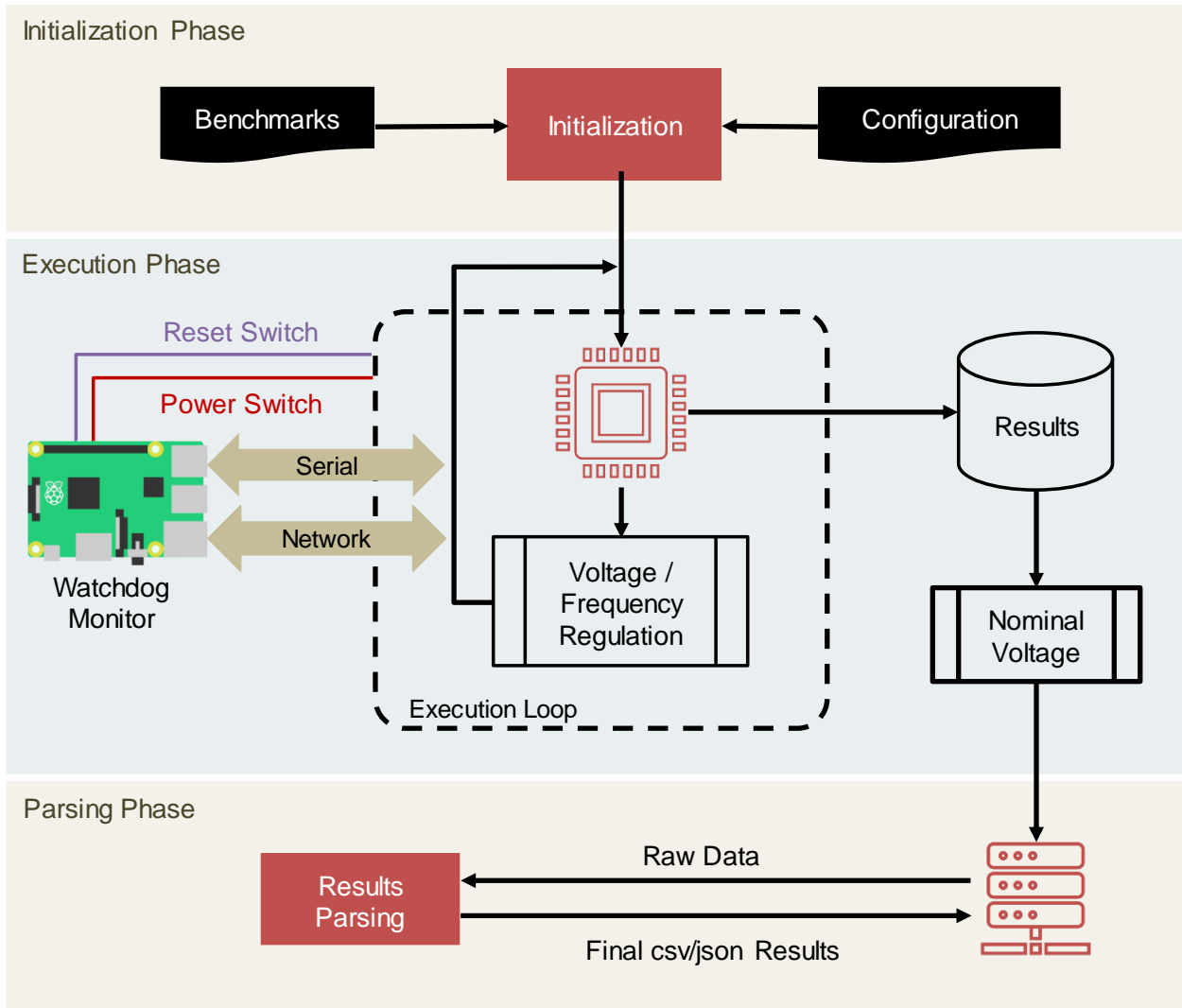


Figure 29: Margins Characterization Framework Layout.

**Failure Recognition.** Another challenge is to recognize and distinguish the system and program crashes or hangs. This is a very important feature to easily identify and classify the final results, with the most possible distinct information concerning the characterization.

**Reliable Cores Setup.** Another major challenge we also face is that the characterization of a system is performed primarily by using properly chosen programs in order to provide diverse behaviors and expose all the potential deviations from nominal conditions. It is thus important to run the selected benchmarks in *reliable cores setup*. This means that the cores, where the benchmark runs, must be isolated and unaffected from the other active processes of the kernel in order to capture only the effects of the desired benchmark.

**Iterative Execution.** The non-deterministic behavior of the characterization results due to several microarchitectural features makes necessary to repeat the experiments multiple times with the same configuration to eliminate the probability of misleading results.

As shown in Figure 29, the proposed framework consists of three phases (Initialization, Execution, Parsing). In the following subsections, we analyze each of these functionalities grouped in the 3 distinct phases of the framework's execution (Initialization, Execution,

Parsing), and describe their detailed implementation and how these challenges were overcome.

### 3.2.1 Initialization Phase

During the *initialization phase*, a user can declare a benchmark list with any input dataset to run in any desirable characterization setup. The characterization setup includes the voltage and frequency (V/F) values under which the experiment will take place and the cores where the benchmark will be run; this can be an individual core, a pair of cores in the same PMD (Processor MoDule – pair of cores), or all of the available eight cores in the microprocessor. The characterization setup depends on the power domains supported by the chip, but our framework is easily extensible to support the power domain features of different CPU chips.

This phase is in charge of setting the voltage and frequency ranges, the initial voltage and frequency values, with which the characterization begins, and to prepare the benchmarks: their required files, inputs, outputs, as well as the directory tree where the necessary logs will be stored. This phase is performed at the beginning of the characterization and each time the system is restored by the Raspberry (for example, after a system crash) in order to proceed to the next run until the entire Execution Phase finishes. Each time the system is restored, this phase restores the initial user's desired setup and recognizes where and when the characterization has been previously stopped. This step is essential for the characterization to proceed sequentially according to user's choice, and to complete the whole Execution Phase.

This phase is also responsible to overcome the challenge of *reliable cores setup that is responsible to ensure the correctness and integrity of our results*. The benchmark must run in an “as bare as possible” system without the interference of any other running process. Therefore, reliable cores setup is twofold: first, it recognizes these cores or group of cores that are not currently under characterization, and migrates all currently running processes (except for the benchmark) to a completely different core. The migration of system processes is required to isolate the execution of the desired benchmark from all other active processes.

Second, given that all the PMDs in the studied system are in the same power domain, they always have the same voltage value (in case this does not hold in a different microarchitecture the proposed framework can be adapted). This means that even though there are several processes run on different cores (not in the core(s) under characterization), they have the same probability to affect an unreliable operation while reducing the voltage. On the other hand, each individual PMD can have different frequency, so we leverage the combination of V/F states in order to set the core under characterization to the desired frequency, and all other cores to the minimum available frequency in order to ensure that an unreliable operation is due to the benchmark's execution only. When for example the characterization takes place in the PMD0 (meaning that the benchmark runs in PMD0; cores 0 and 1), the PMD0 is set to the pre-defined by the user frequency (e.g., the maximum frequency 2.4GHz), and all the other PMDs are set to the minimum available frequency (300MHz in our case). Thus, all the running processes, except for the benchmark, are executed to the reliable-cores setup.

In our setup, we also use a stripped/lightweight Linux Kernel to diminish the unnecessary kernel daemons that the majority of well-known Linux Distributions provide. Thus, the system's running processes and the common power domain of all PMDs, neither affect the benchmarks execution nor can contribute to a system's failure or error event.

### 3.2.2 Execution Phase

After the characterization setup is defined, the automated *Execution Phase* begins. The Execution Phase consists of multiple runs of the same benchmark, each one representing the execution of the benchmark with a pre-defined characterization setup. The set of all the characterization runs running the same benchmark with different characterization setups represents a *campaign*. After the initialization phase, the framework enters the Execution Phase, in which all runs take place. The runs are executed according to user's configuration, while the framework reduces the voltage with a step defined by the user in the initialization phase. For each run, the framework collects and stores the necessary logs at a safe place externally to the system under characterization, which will be then used by the parsing phase.

The logged information includes: the output of the benchmark at each execution, the corrected and uncorrected errors (if any) collected by the Linux EDAC Driver [136], as well as the errors' localization (L1 or L2 cache, DRAM, etc.), and several failures, such as benchmark crash, kernel hangs, and system unresponsiveness. The framework can distinguish these types of failures and keep logging about them to be parsed later by the parsing phase. Benchmark crashes can be distinguished by monitoring the benchmark's exit status. On the other hand, to identify the kernel hangs and system unresponsiveness, during this phase the framework notifies the Raspberry when the execution is about to start and also when the execution finishes.

In the meantime, the Raspberry starts pinging the system to check its responsiveness. If the Raspberry does not receive a completion notification (hang) in the given time (we defined as timeout condition 2 times the normal execution time of the benchmark) or the X-Gene 2 turns completely unresponsive (ping is not responding), the Raspberry sends a signal to the Power Off button on the board, and the system resets. After that, the Raspberry is also responsible to check when the system is up again, and sends a signal to restart the experiments. These decisions contribute to the *Failure Recognition* challenge.

During the experiments, some Linux tasks or the kernel may hang. To identify these cases, we use an inherent feature of the Linux kernel to periodically detect these tasks by enabling the flag "*hung\_task\_panic*" [136]. Therefore, if the kernel itself recognizes a process hang, it will immediately reset the system, so there is no need for the Raspberry to wait until the timeout. In this way, we also contribute to the *Failure Recognition* challenge and accelerate the reset procedure and the entire characterization.

Note that, in order to isolate the framework's execution from the core(s) under characterization, the operations of the framework are also performed in *Reliable Cores Setup*. However, when there are operations of the framework, such as the organization of log files during the benchmark's execution that are an integral part of the framework, and thus, they must run in the core(s) under characterization, these operations are performed after the benchmark's execution in the nominal conditions. This is the way to ensure that any logging information will be stored correctly and no information will be lost or changed due to the unstable system conditions, and thus, to overcome the *Safe Data Collection* challenge.

### 3.2.3 Parsing Phase

In the last step of our framework, all the log files that are stored during the Execution Phase are parsed in order to provide a fine-grained classification of the effects observed for each characterization run. Note that, each run is correlated to a specific benchmark and characterization setup. The categories that are used for our classification are summarized in Table 11, but the parser can be easily extended according to the user's

needs. For instance, the parser can also report the exact location that the correctable errors occurred (e.g., the cache level, the memory, etc.) using the logging information provided by the Execution Phase.

Note that each characterization run can manifest multiple effects. For instance, in a run both SDC and CE can be observed; thus, both of them should be reported by the parser for this run. Furthermore, the parser can report all the information collected during multiple campaigns of the same benchmark. The characterization runs with the same configuration setup of different campaigns may also have different effects with different severity. For instance, let us assume two runs with the same characterization setup of two different campaigns. After the parsing, the first run finally revealed some CEs, and the second run was classified as SDC. At the end of the parsing step, all the collected results concerning the characterization (according to Table 11) are reported in .csv and .json files.

In Table 12 we can see an example output of the characterization framework. Table 12 presents a characterization campaign for 1 run and the whole voltage range from the nominal voltage (980mV) downwards to the minimum voltage in which the microprocessor cannot operate at all. In Table 12 we can see the characterization results for each individual core (it shows the results for cores 0, 2, 4, and 6) and for the maximum frequency (2.4 GHz). With this detailed table we can observe many different aspects, such as the core-to-core variation, the safe  $V_{min}$  for each different core, the produced abnormal behaviors for each core, etc. For example, it is clearly showed that the Core 4 is the most robust Core, while the Cores 0 and 2 are the most sensitive ones, because

**Table 11: Experimental effect categorization.**

Effect	Description
<b>NO</b> (Normal Operation)	The benchmark was successfully completed without any indications of failure.
<b>SDC</b> (Silent Data Corruption)	The benchmark was successfully completed, but a mismatch between the program output and the correct output was observed.
<b>CE</b> (Corrected Error)	Errors were detected and corrected by the hardware.
<b>UE</b> (Uncorrected Error)	Errors were detected, but not corrected by the hardware.
<b>AC</b> (Application Crash)	The application process was not terminated normally (the exit value of the process was different than zero).
<b>TO</b> (Application Timeout)	The application process cannot finish and exceeds its normal execution time (e.g., infinite loop).
<b>SC</b> (System Crash)	The system was unresponsive; meaning that the X-Gene 2 is not responding to pings or the timeout limit was reached.

Table 12: Example output of the characterization framework.

Benchmark_Name				
mV	0	2	4	6
980	NO	NO	NO	NO
:	NO	NO	NO	NO
915	NO	NO	NO	NO
910	SDC	NO	NO	NO
905	SDC	SDC	NO	NO
900	SDC	1 x L1CE SDC	NO	NO
895	1 x L1CE SDC	7 x L1CE SDC	NO	SDC
890	5 x L1CE SC	14 x L1CE 3 x L2CE SDC	NO	SDC
885	-	SC	SDC	3 x L1CE SDC
880	-	-	SDC	13 x L1CE 4 x L2CE SDC
875	-	-	9 x L1CE SDC	5 x L1CE 2 x L2CE 1 x L2UE AC
870	-	-	11 x L1CE 2 x L2CE 1 x L2UE AC	1 x L1CE 1 x L2UE AC
865	-	-	1 x L1CE 1 x L2UE AC	SC
860	-	-	SC	-

they produce higher safe  $V_{min}$  than the Core 4. Moreover, for all cores the first detected abnormal behavior is the SDC.

### 3.3 System Characterization

We study the behavior of 3 different X-Gene 2 chips by using representative benchmarks from the SPEC CPU2006 suite to explore the voltage guardbands for each core of the chip, and thus to detect the safe  $V_{min}$  in which the benchmarks can be executed correctly. We also study any abnormal behavior that can be exposed (SDC, ECC errors, crashes, etc.) below the safe  $V_{min}$  levels, for a comprehensive characterization. We present our findings for three different chips: one typical chip (TTT), and two corner chips (TFF, and TSS).

From the designer's point of view, the collective effects of process and environmental variation can be lumped into their effect on transistors: typical (also called nominal), fast, or slow. In CMOS, there are two types of transistors with somewhat independent

characteristics, so the speed of each can be characterized. Moreover, interconnect speed may vary independently of devices. When these processing variations are combined with the environmental variations, we define design or process corners. The term corner refers to an imaginary box that surrounds the guaranteed performance of the circuits, as shown in Figure 30. The box is not square because some characteristics such as oxide thickness track between devices, making it impossible to find a slow nMOS transistor with thick oxide and a fast pMOS transistor with thin oxide simultaneously.

Table 13 lists a number of interesting design corners. The corners are specified with five letters describing the nMOS, pMOS, interconnect, power supply, and temperature, respectively. The letters are F, T, and S, for fast, typical, and slow. Circuits are most likely to fail at the corners of the design space, so nonstandard circuits should be simulated at all corners to ensure they operate correctly in all cases. Often, integrated circuits are designed to meet a timing specification for typical processing. These parts may be binned; faster parts are rated for higher frequency and sold for more money, while slower parts are rated for lower frequency. In any event, the parts must still work in the slowest SSSSS environment. Other integrated circuits are designed to obtain high yield at a relatively low frequency; these parts are simulated for timing in the slow process corner. The fast corner FFFFFF has maximum speed. Other corners are used to check for races and ratio problems where the relative strengths and speeds of different transistors or interconnect are important. The FFFFS corner is important for noise because the edge rates are fast, causing more coupling; the threshold voltages are low; and the leakage is high [137] [138].

Usually, the corners are abbreviated to fewer letters. For example, two letters generally refer to nMOS and pMOS. Three refer to nMOS, pMOS, and overall environment. In this work we use three letters; the TTT part is the “normal” part, the TFF is a fast corner part, which has high leakage but at the same time can operate at higher frequency, and the TSS part is a slow corner part which has low leakage and works at lower frequency. All microprocessor chips have maximum frequency equal to 2.4 GHz, however, the TSS chip may have a lower voltage guardband than the TTT and TFF chips due to its lower power leakage.

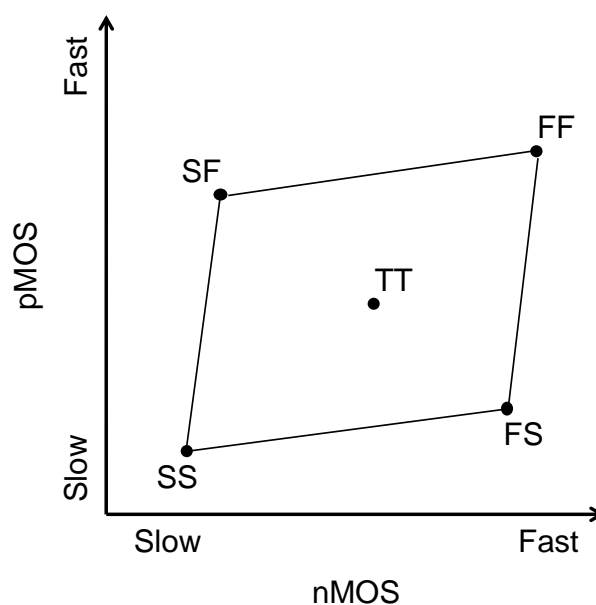


Figure 30: Design corners [138].



Table 13: Design corner checks [138].

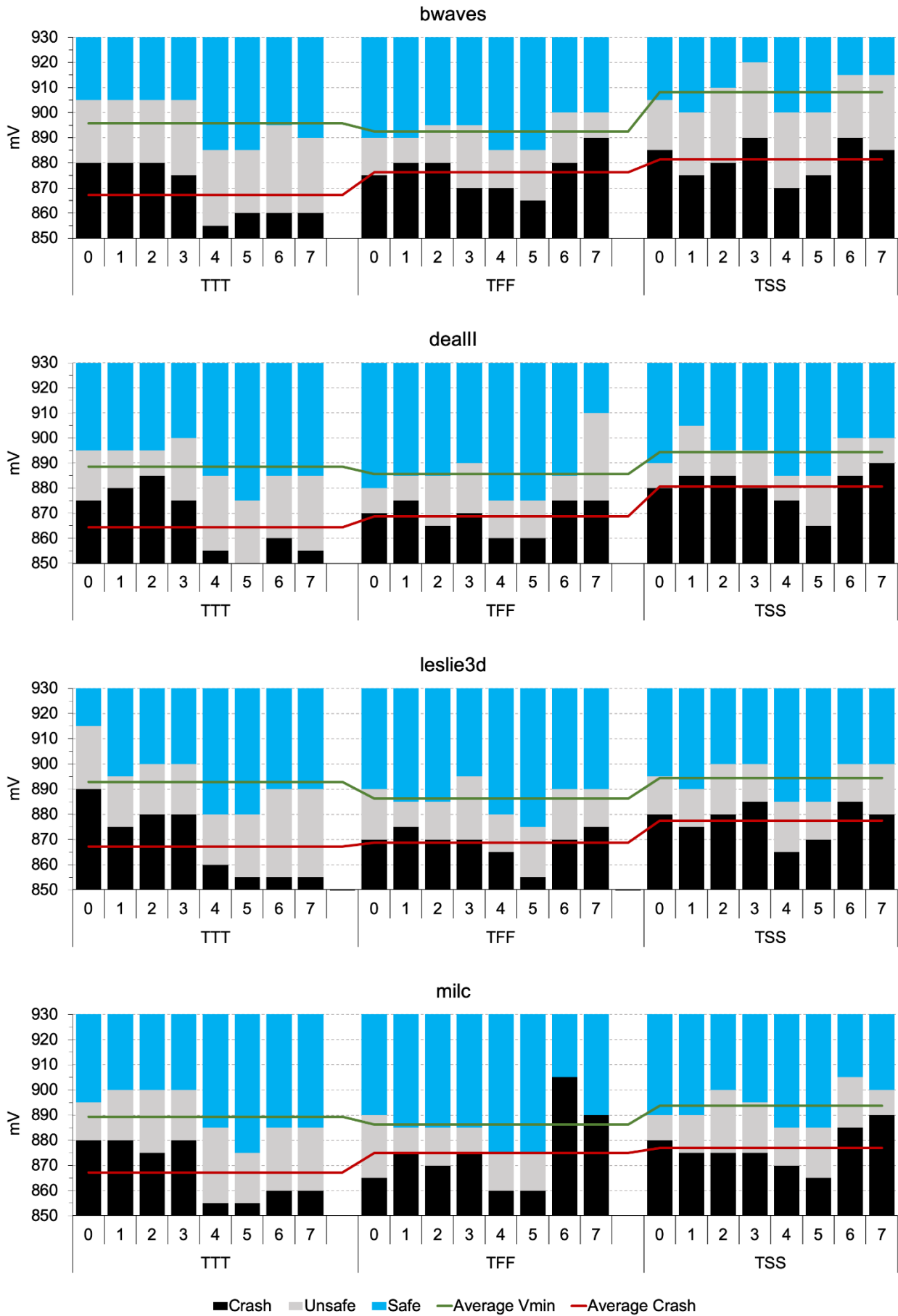
Corner					Purpose
nMOS	pMOS	Wire	Vdd	Temp	
T	T	T	S	S	Timing specifications (binned parts)
S	S	S	S	S	Timing specifications (conservative)
F	F	F	F	F	Race conditions, hold time constraints, noise
S	S	?	F	S	Dynamic power
F	F	F	F	S	Subthreshold leakage noise and power
S	S	F	S	S	Races of gates against wires
F	F	S	F	F	Races of wires against gates
S	F	T	F	F	Pseudo-nMOS and ratioed circuits noise margins, memory read/write, race of pMOS against nMOS
F	S	T	F	F	Ratioed circuits, memory read/write, race of nMOS against pMOS

### 3.3.1 Regions of Operation

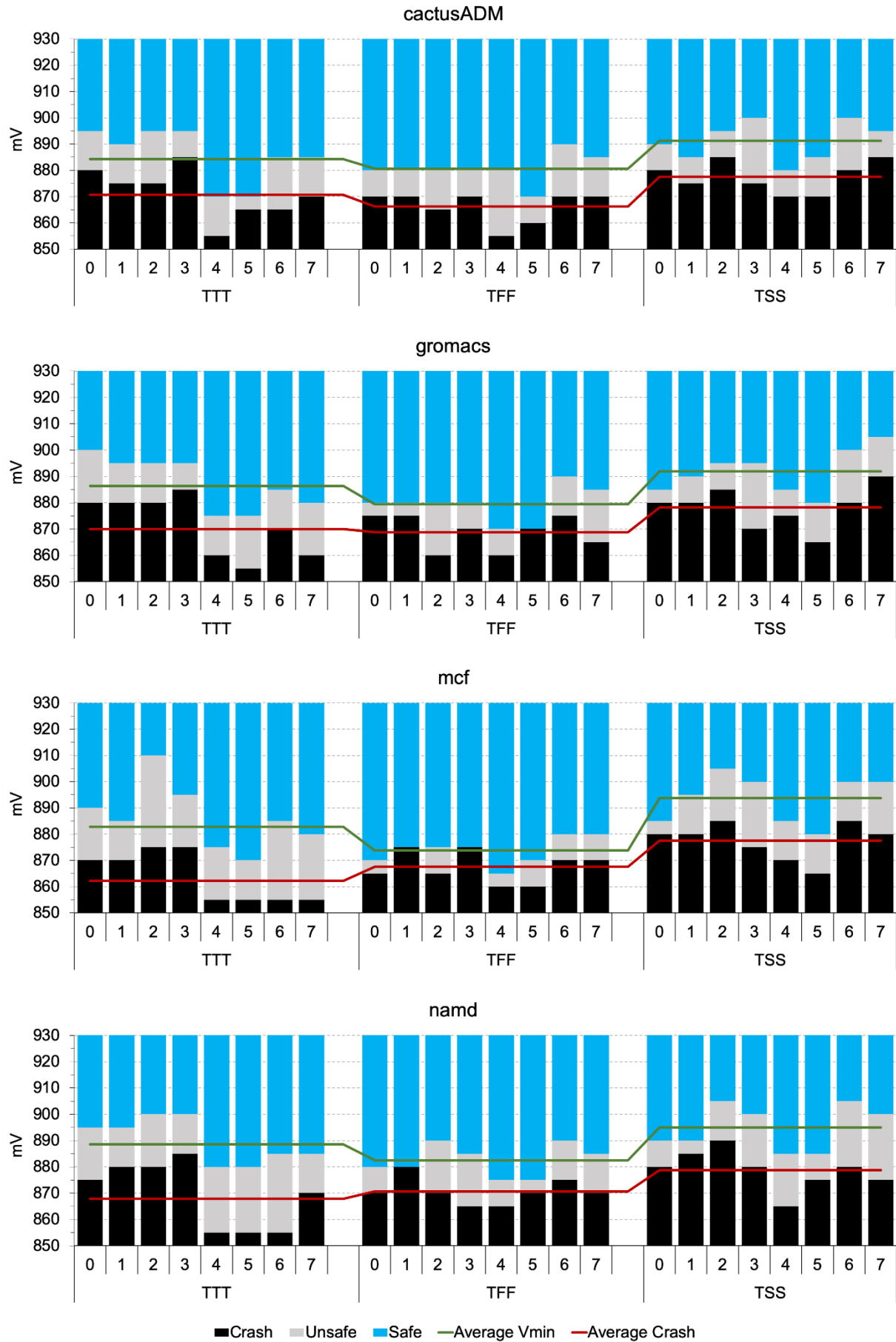
Using the automated framework presented in subsection 3.2, we extensively characterize the three X-Gen2 chips. The characterization process can reveal for each core of the CPU three different regions of operation, when the microprocessor operates beyond nominal voltage conditions. These are the *safe* and *unsafe* operating regions and the region in which the system cannot operate (*crash region*).

To isolate the impact of temperature that can affect our results, apart from the isolation of system processes (see subsection 3.2), we also control the temperature by adjusting the CPU's fan speed accordingly. We stabilize the temperature at 43°C, and thus, all benchmarks complete their execution at the same temperature. In Figure 31, Figure 32, and Figure 33 we present the results for 10 SPEC CPU2006 benchmarks [141]. All programs ran on a single core in each PMD at 2.4 GHz, while the remaining six cores (the other 3 PMDs) reliably operated at 300 MHz (see explanation in subsection 3.2). In order to consider the non-deterministic behavior of such experiments, we ran every undervolting campaign 10 different times (taking into account the long execution times). Figure 31, Figure 32, and Figure 33 present in detail for all benchmarks the highest  $V_{min}$  values and the highest *crash* voltage values of the ten campaigns for the three different chips and all the cores of each chip. In all benchmarks, we can notice the three regions of operation according to the collected results. These regions are:

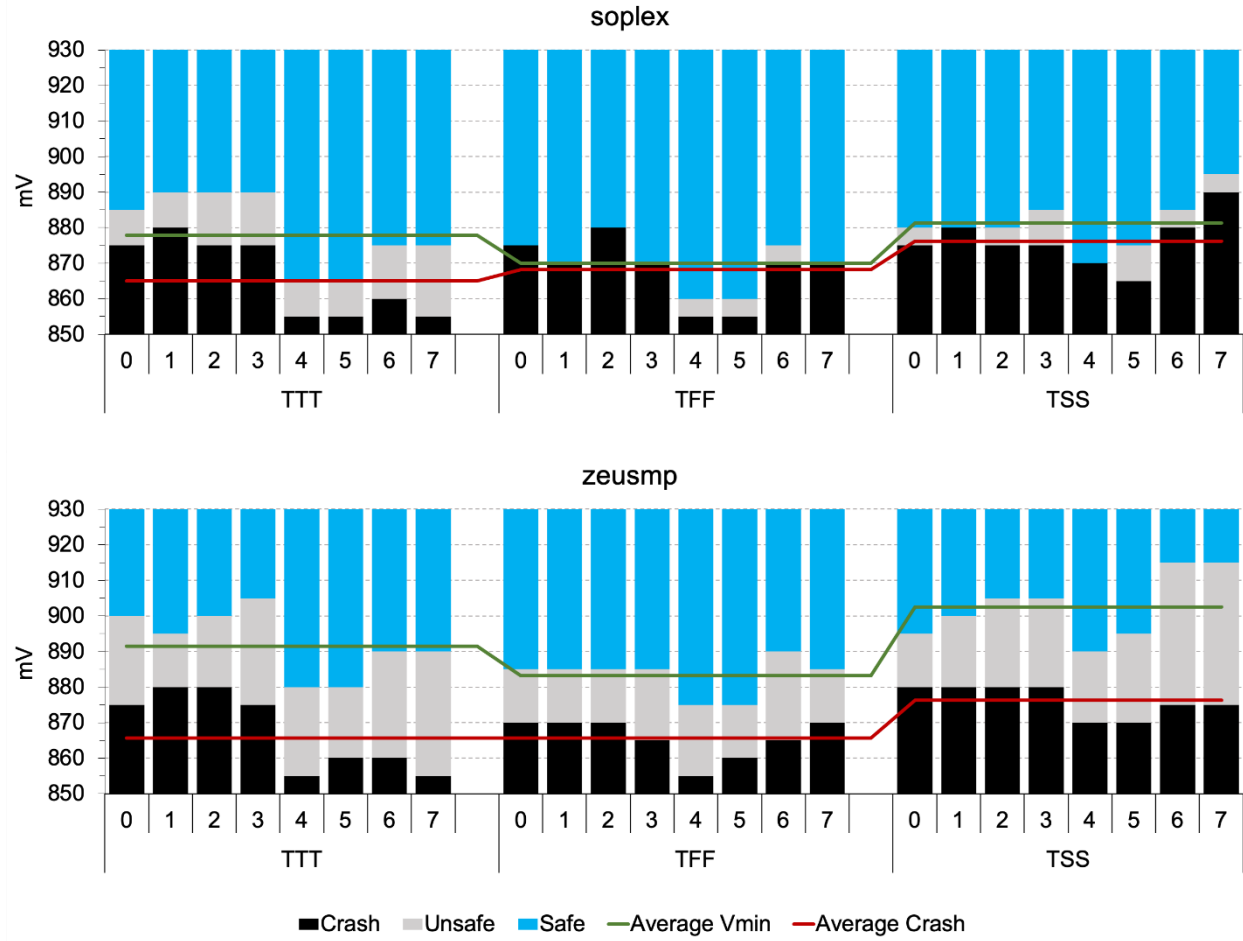
- **Safe region (blue):** The characterization runs that correspond to this region had a normal operation (NO) without any SDCs, errors or crashes.
- **Unsafe region (grey):** The characterization runs that correspond to this region generate an abnormal behavior (SDC, CE, UE, AC) but not a system crash.
- **Crash region (black):** This region includes voltage values in which at least one characterization run led to a system crash.



**Figure 31: X-Gene 2 characterization results for 4 SPEC CPU2006 benchmarks (*bwaves*, *dealll*, *leslie3d*, *milc*) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region.**



**Figure 32: X-Gene 2 characterization results for 4 SPEC CPU2006 benchmarks (*cactusADM*, *gromacs*, *mcf*, *namd*) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region.**



**Figure 33: X-Gen 2 characterization results for 2 SPEC CPU2006 benchmarks (*soplex*, and *zeusmp*) on three different chips (TTT, TFF, TSS). Blue represents the Safe region; grey represents the Unsafe region; and black represents the Crash region.**

### 3.3.2 $V_{min}$ Experimental Results

We experimentally obtain the  $V_{min}$  values of the 10 SPEC CPU2006 benchmarks on the three X-Gen 2 chips (TTT, TFF, TSS), running the entire time-consuming undervolting experiments 10 times for each benchmark. These experiments were performed during 6 months on a single X-Gen 2 machine (for all the 3 microprocessor chips). This part of our study focuses on a quantitative analysis of the safe  $V_{min}$  for different chips of the same architecture in order to expose the potential guardbands of each chip, as well as to quantify how the program behavior affects the guardband and to measure the core-to-core and chip-to-chip variation.

The voltage guardband for each program is the smallest (safe) margin between the nominal voltage of the microprocessor and its  $V_{min}$ . Our single-core experiments were performed in the highest available frequency of the X-Gen 2, which is 2.4 GHz. In that frequency we observed divergences of the  $V_{min}$  values as shown in Figure 34. For a significant number of benchmarks, we can see variations between different programs and different chips. Figure 34 represents all 10 benchmarks for the most robust core for each chip (Core 4 in all three microprocessors chips), and for these programs the  $V_{min}$  varies from 885mV to 865mV for TTT (blue line), from 885mV to 860mV for TFF (orange line) and from 900mV to 870mV for TSS (green line). Considering that the nominal voltage for the X-Gen 2 microprocessors is 980mV, there is a significant reduction of voltage without affecting the correct execution of programs, which is equal to at least 9.7% for the TTT and TFF chip, and 8.2% for the TSS chip. We also notice in Figure 34 that the workload-

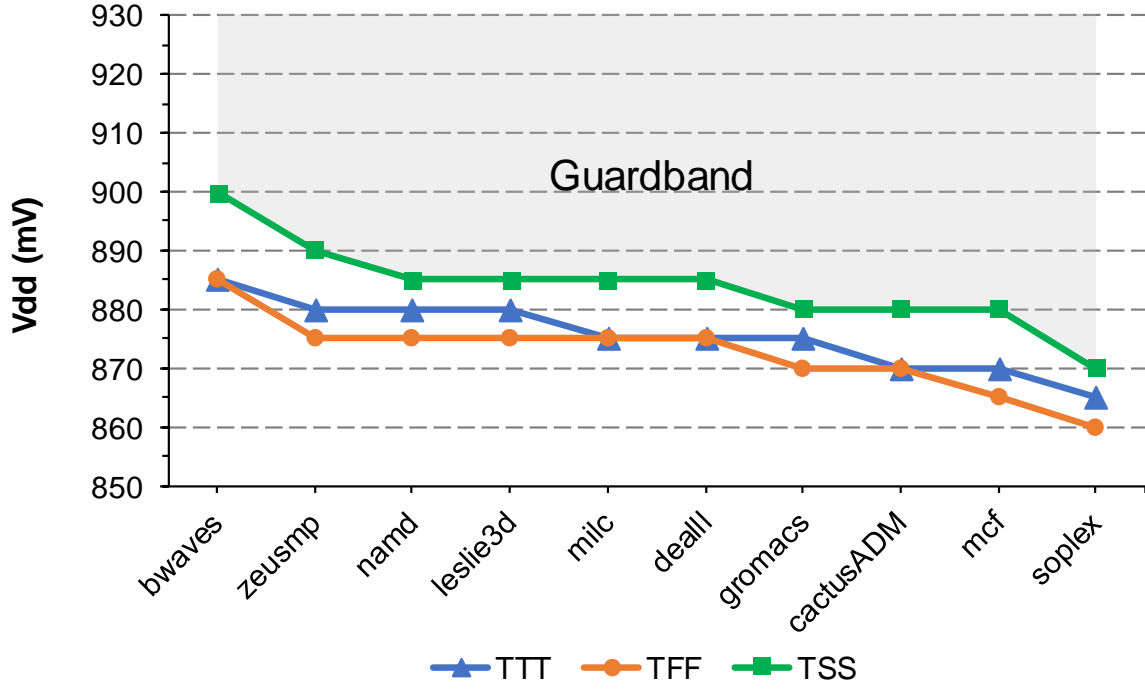


Figure 34:  $V_{min}$  results at 2.4 GHz for 10 SPEC CPU2006 programs on 3 different X-Gene 2 chips (TTT, TFF, TSS).

to-workload variation remains the same across the 3 chips of the same architecture; however, there is a relatively large variation among the chips. This means that there is a program dependency of  $V_{min}$  behavior in all chips.

### 3.3.3 Process Variation

Figure 31, Figure 32, and Figure 33 present the detailed information about the safe  $V_{min}$  for all benchmarks and cores of the three chips, as well as the range of the unsafe region. More specifically, in this section, we discuss the chip-to-chip and core-to-core variation.

Devices and interconnects have variations in film thickness, lateral dimensions, and doping concentrations [139]. These variations can be classified as inter-die (e.g., all the transistors on one die might be shorter than normal because they were etched excessively) and intra-die (e.g., one transistor might have a different threshold voltage than its neighbor because of the random number of dopant atoms implanted).

For devices, the most important variations are channel length  $L$  and threshold voltage  $V_t$ . Channel length variations are caused by photo-lithography proximity effects, deviations in the optics, and plasma etch dependencies. Threshold voltages vary because of different doping concentrations and annealing effects, mobile charge in the gate oxide, and discrete dopant variations caused by the small number of dopant atoms in tiny transistors. Threshold voltages gradually change as transistors wear out.

For interconnect, the most important variations are line width and spacing, metal and dielectric thickness, and contact resistance. Line width and spacing, like channel length, depend on photolithography and etching proximity effects. Thickness may be influenced by polishing. Contact resistance depends on contact dimensions and the etch and clean steps [138].

Process variations can be classified as follows:

- Lot-to-lot ( $L2L$ )

- Wafer-to-wafer (*W2W*)
- Chip-to-Chip, Die-to-die (*D2D*), inter-die, or within-wafer (*W/W*)
- Core-to-Core, Within-die (*WID*) or intra-die

**Chip-to-Chip Variation:** Wafers are processed in batches called *lots*. A lot processed after a furnace has been shut down and cleaned may behave slightly differently than the lot processed earlier. One wafer may be exposed to an ion implanter for a slightly different amount of time than another, causing W2W threshold voltage variation. A die near the edge of the wafer may etch slightly differently than a die in the center, causing chip-to-chip channel length variations. Unless calibrations are made on a per-lot or per-wafer basis, L2L and W2W variations are often lumped into the inter-die variations. Inter-die variations ultimately make one chip faster or slower than another (this phenomenon finally results in Chip-to-Chip variation). They can be handled by providing enough margin to cover 2 or  $3\sigma$  of variation and by rejecting the small number of chips that fall outside this bound, as discussed in section 3.3.

As Figure 31, Figure 32, and Figure 33 show, PMD 2 (cores 4 and 5) is the most robust PMD for all three chips (up to 3.6% more voltage reduction compared to the most sensitive cores). Green line in Figure 31, Figure 32, and Figure 33 presents the average  $V_{min}$ , and the Red line represents the average Crash voltage point for each chip. Thus, we can notice that the TFF chip has lower  $V_{min}$  points than the TTT chip, in contrast to TSS (the chip with lower leakage), which has significantly higher  $V_{min}$  points than the other two chips, and thus, lower power savings. For the unsafe region, on the other hand, we notice only small divergences among the chips.

**Core-to-Core Variation:** Intra-die variations were once small compared to inter-die variations and were largely ignored by digital designers but have become quite important in nanometer processes. Some intra-die variations are spatially correlated; these are called *process tilt*. For example, an ion implanter might deliver a larger dose near the center of a wafer than near the periphery, causing threshold voltages to tilt radially across the wafer. In summary, transistors on the same chip match better than transistors on different chips and adjacent transistors match better than widely separated ones. Intra-die variations are more challenging to manage because some of the millions or billions of transistors on a chip are likely to stray far from typical parameters. This phenomenon finally leads to the Core-to-Core variation.

As shown in Figure 31, Figure 32, and Figure 33, there are significant divergences among cores for the same benchmark due to process variation. Process variations can affect transistor dimensions (length, width, oxide thickness, etc.) which have direct impact on the threshold voltage of a MOS device. More specifically, variation has a minor impact on dynamic energy, but a major impact on static leakage energy [140]. Variation shifts the minimum energy and energy-delay product (EDP) operating points toward a higher supply and threshold voltage, and reduces the potential benefits in operating at these points, and thus, the guardband of each core. This variation among cores of the same chip can result in high energy savings by using the appropriate task scheduling. We present and discuss this method in the following subsection.

### 3.3.4 Abnormal Behaviors below $V_{min}$

Variation can also cause circuits to malfunction, especially at low voltage. Previous studies on Intel Itanium CPUs [7] [8] have shown a large region of voltage values that contains only ECC *corrected* errors during undervolting. By *reducing* the voltage on those chips, the number of corrected errors increases gradually for quite many voltage steps until it exposes other types of abnormal behavior (SDCs, uncorrected errors, crashes). In such systems, ECC corrected errors can serve as proxies for the effects of undervolting.

In contrast to these studies, a major finding of our characterization for ARMv8-compliant multicore CPUs is that *silent data corruptions appear at higher voltage levels than corrected errors alone for any benchmark*. In [7] and [8], the reported range of voltage levels with corrected errors alone offers a significant opportunity for energy savings without jeopardizing correctness of operation. High correctable error rate is helpful to an ECC guided voltage speculation but this is not the case in the APM X-Gene 2 in our case.

To justify the differences between X-Gene 2 and previous studies on Itanium [7] [8], we developed and ran self-tests that separately stress each cache level independently as well as the ALU and FPU (we provide details about the development of these tests later in this thesis in subsection 3.6). Cache tests completely fill the cache arrays and flip all the bits of each cache block to check for cell bit errors during undervolting. ALU and FPU tests perform multiple different concurrent operations in each unit with random values to stress different paths and conditions. Through this component-focused stress process we observed the following: (1) SDCs occur when the pipeline gets stressed (ALU and FPU tests), and (2) the cache bit-cells safely operate at higher voltages (the cache tests crash in much lower voltages than the ALU and FPU tests). This observation leads us to conclude that *the X-Gene 2 is more susceptible to timing-path failures than to SRAM array failures*.

In contrast, ECC corrections appear at a higher voltage on the Itanium compared to SDCs and system crashes. We attribute the increased robustness to timing-failures on the Itanium to circuit-level dynamic-margin mitigation techniques such as the capability to perform continuous clock-path de-skewing during dynamic operation [142]. The X-Gene 2 does not deploy such circuit-level techniques, and thereby, generates SDCs due to timing-path failures. Having the occurrence of *SDCs first*, it is not possible to easily guide the voltage speculation for prediction based on the manifested errors. For that reason, we present the *severity function* both for quantifying the severity and for illustrating the scaling of abnormal behaviors due to voltage reduction. The new metric's contribution is twofold: (1) to aggregate the results produced by multiple runs, and (2) to quantify a microprocessor's ability to operate beyond nominal conditions and especially beyond the safe  $V_{min}$ .

### 3.3.5 Severity Function

Note that each characterization run can manifest multiple effects. For instance, in a run both SDC and CE can be observed; thus, both of them are reported for this run. Due to the non-determinism of the characterization in real hardware, all the information collected during multiple campaigns of the same benchmark (iterative execution) is also reported by our *severity function*. To *quantify* the criticality of the effects of different experimental runs of different campaigns with the same setup, we define the “severity function”  $S_v$ , where  $v$  is the voltage, as follows:

$$S_v = W_{SDC} \cdot \frac{SDC}{N} + W_{CE} \cdot \frac{CE}{N} + W_{UE} \cdot \frac{UE}{N} + W_{AC} \cdot \frac{AC}{N} + W_{TO} \cdot \frac{TO}{N} + W_{SC} \cdot \frac{SC}{N}$$

In this function, the parameters  $SDC$ ,  $CE$ ,  $UE$ ,  $AC$ ,  $TO$  and  $SC$  can take values from 0 to  $N$  ( $N$  is the number of runs at voltage level  $v$ ), and represent the times that the effect appears to these runs (for example, if  $k$  of the  $N$  runs lead to UE, then parameter UE is set to  $k$ ; the actual number of uncorrected errors during each run is not taken into consideration). Parameters  $W_{SDC}$ ,  $W_{CE}$ ,  $W_{UE}$ ,  $W_{AC}$ ,  $W_{AC}$  and  $W_{SC}$  represent “weights” that can be arbitrarily set to characterize the severity of each effect of Table 11. The higher the weight, the more critical the effect is considered by our function, and the main role of these weights is to “translate” the behaviors (SDCs, etc.) to numbers in order to fit to the

equation. We use the values presented in Table 14 as the values for our severity function (but different weight values can be also used according to the importance of each observed abnormal behavior in a particular system study).

Figure 35 to Figure 44 show the severity of all executed benchmarks on TTT chip. As an example, Figure 35 has a significantly large *unsafe region*, and it also provides a smooth gradual increase of severity while the voltage is reduced. These results are derived from 10 executions of the same campaign. According to the severity value for each voltage level, one can decide if and when it is possible to reduce the voltage further (lower than the safe  $V_{min}$  where severity is 0) for more aggressive energy efficiency.

Table 14: Weights used in our experiments.

Weight	Value
$W_{SC}$	16
$W_{AC}$	8
$W_{TO}$	8
$W_{SDC}$	4
$W_{UE}$	2
$W_{CE}$	1
$W_{NO}$	0

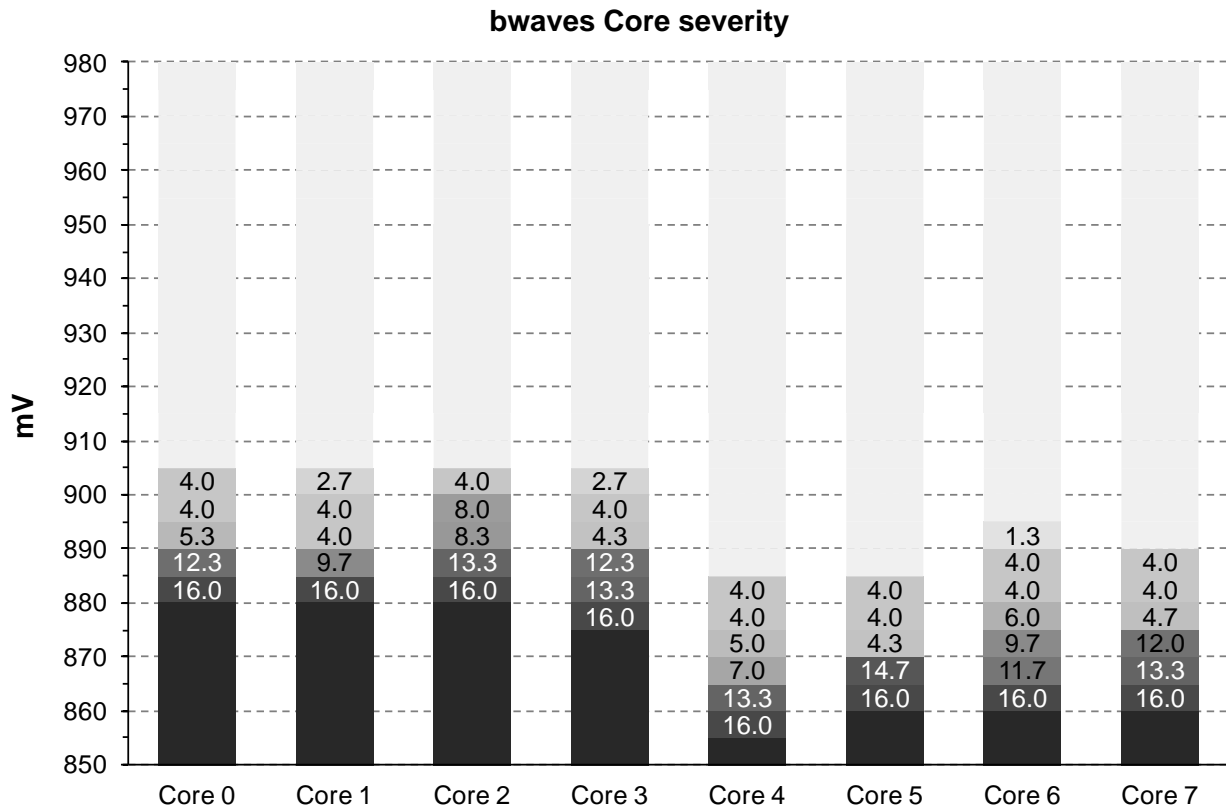


Figure 35: bwaves benchmark severity on TTT chip cores.



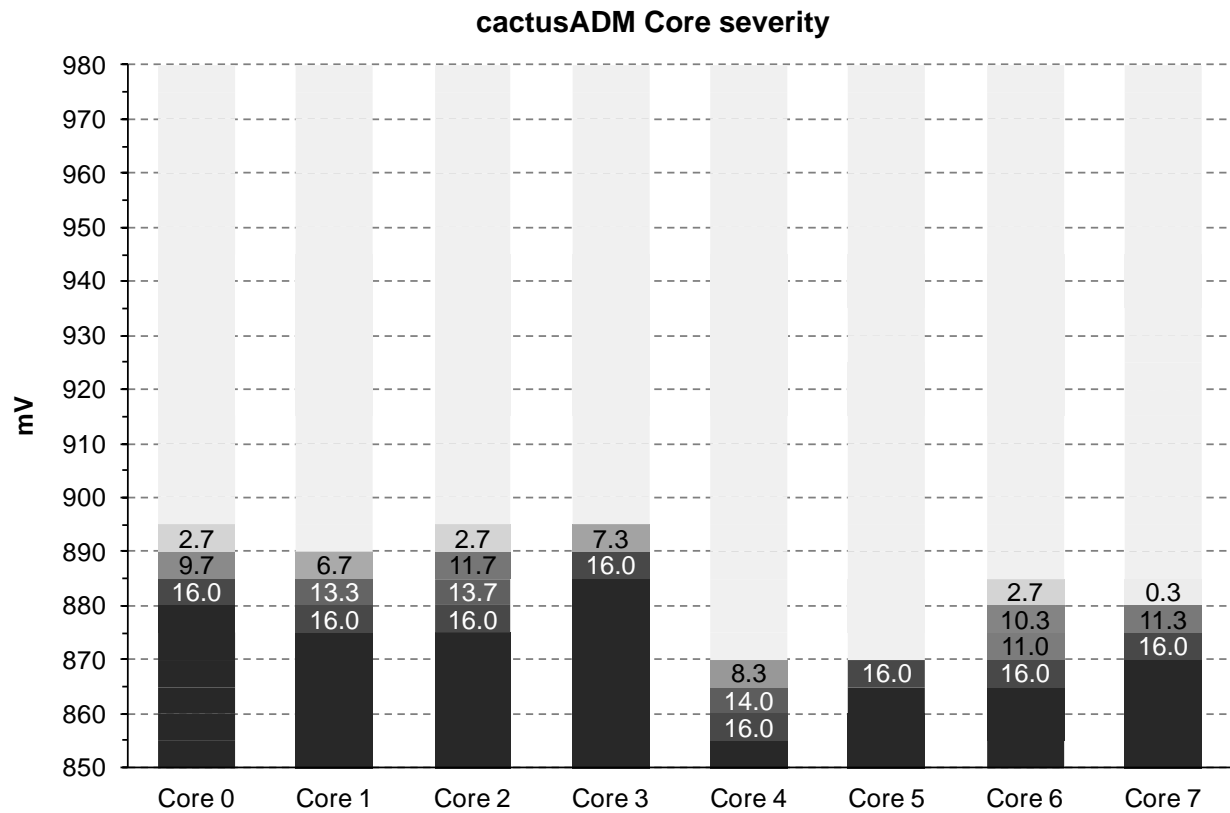


Figure 36: cactusADM benchmark severity on TTT chip cores.

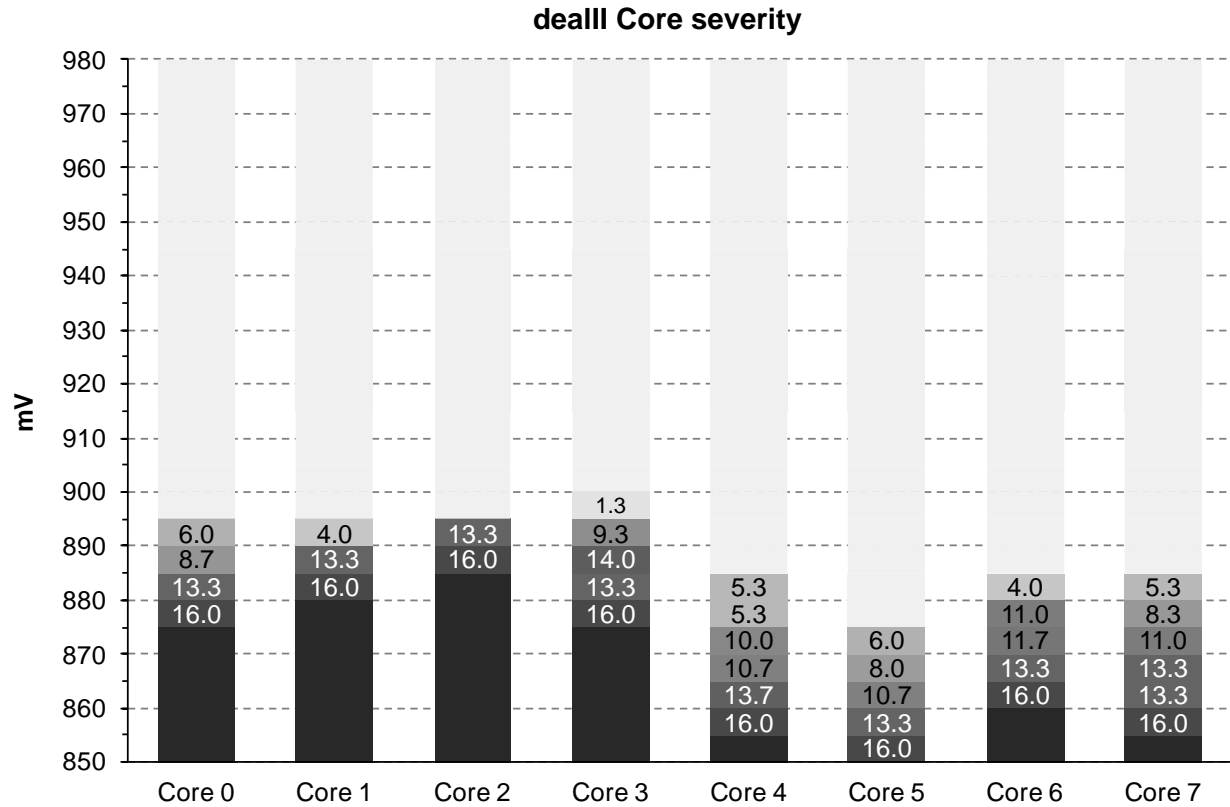


Figure 37: dealll benchmark severity on TTT chip cores.

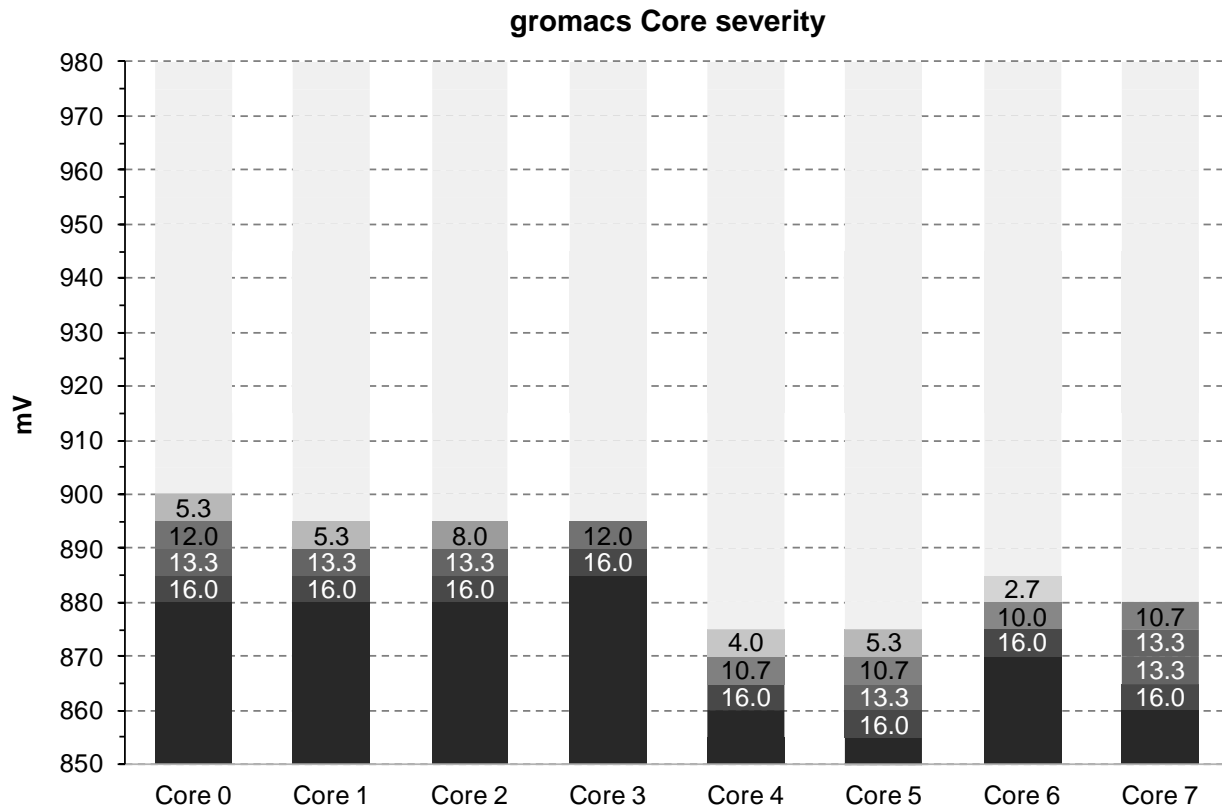


Figure 38: gromacs benchmark severity on TTT chip cores.

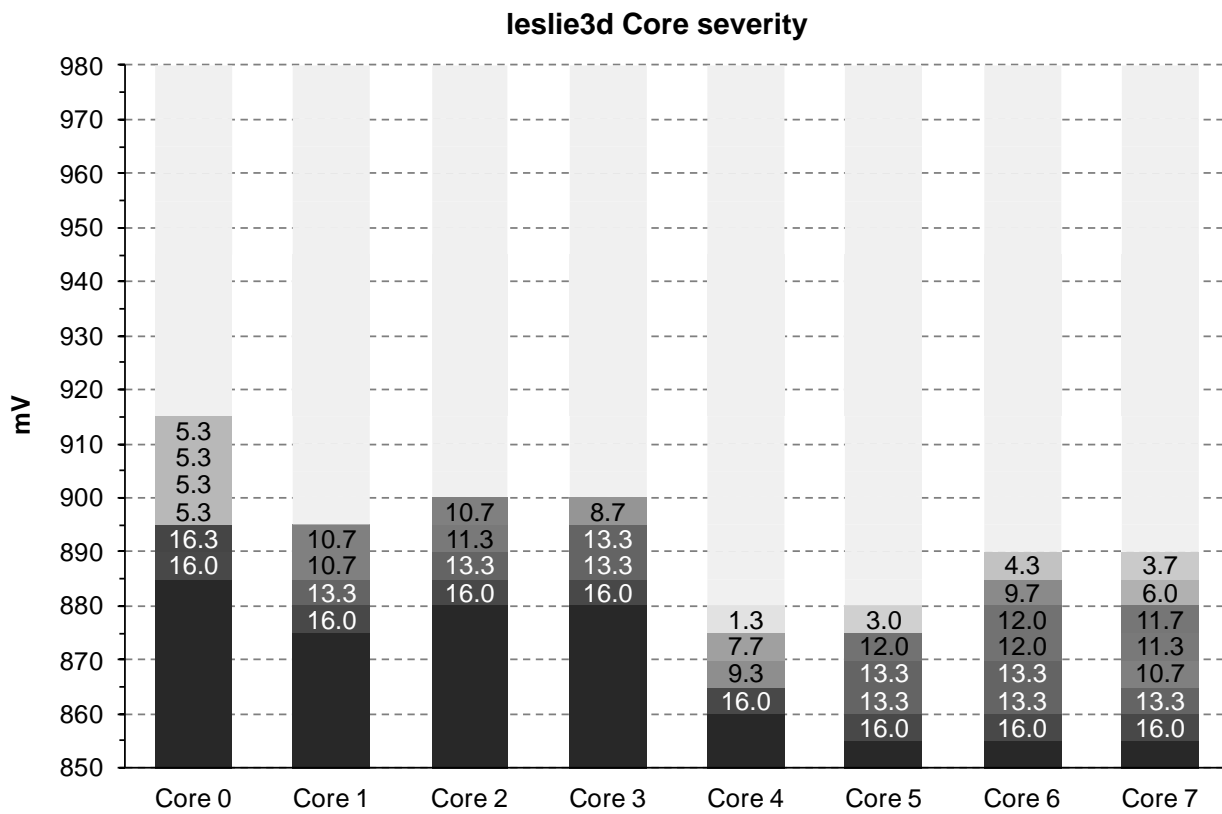


Figure 39: leslie3d benchmark severity on TTT chip cores.

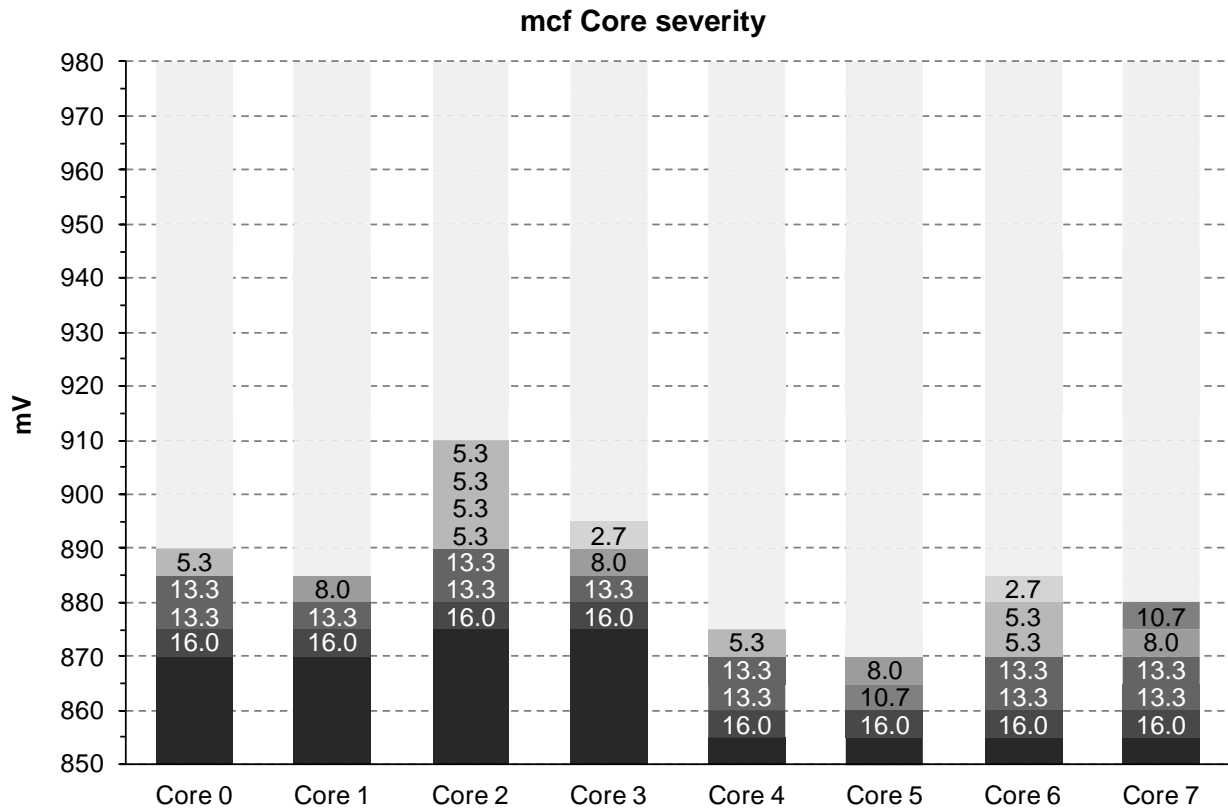


Figure 40: mcf benchmark severity on TTT chip cores.

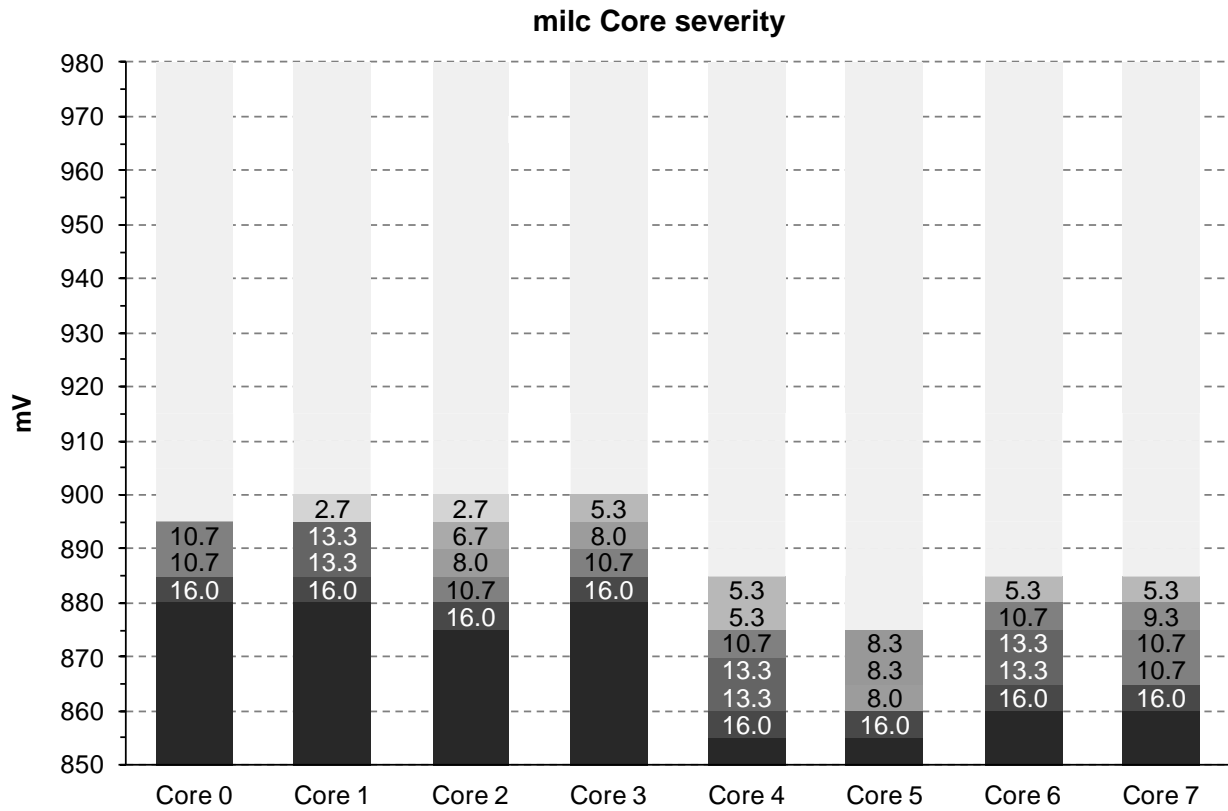


Figure 41: milc benchmark severity on TTT chip cores.

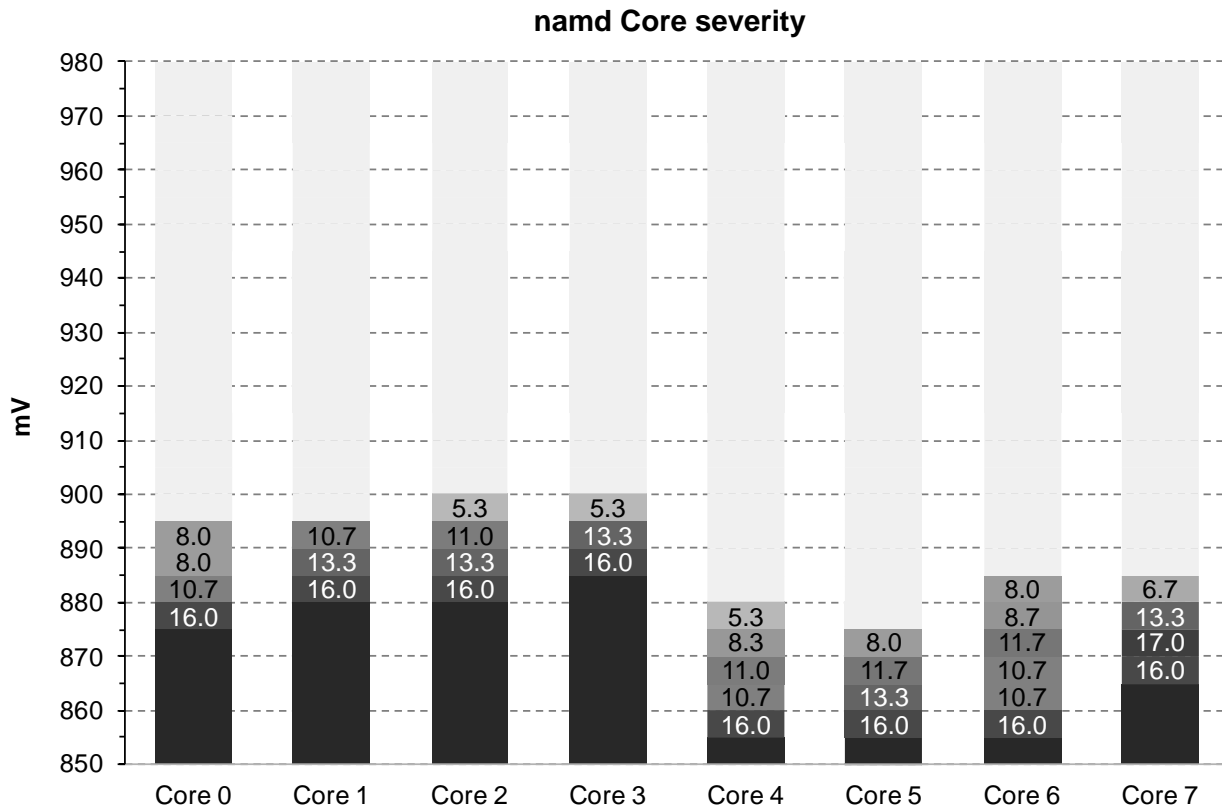


Figure 42: namd benchmark severity on TTT chip cores.

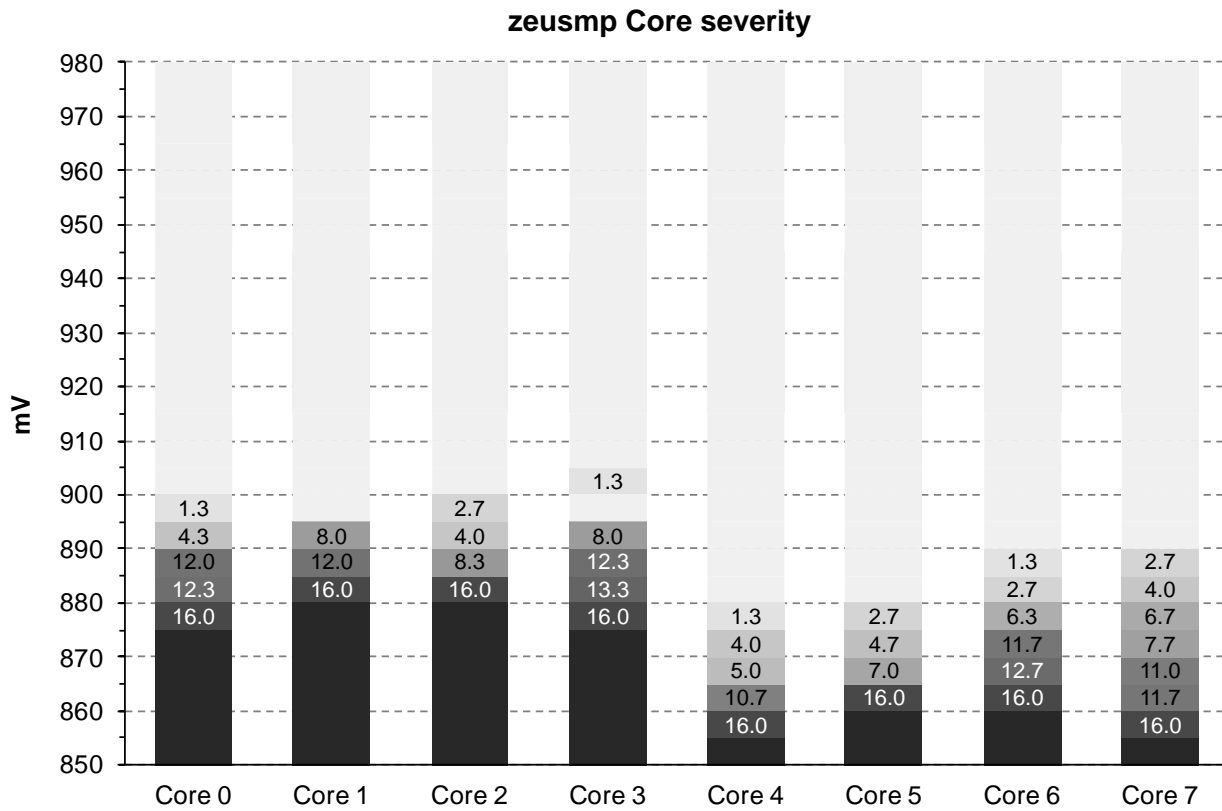
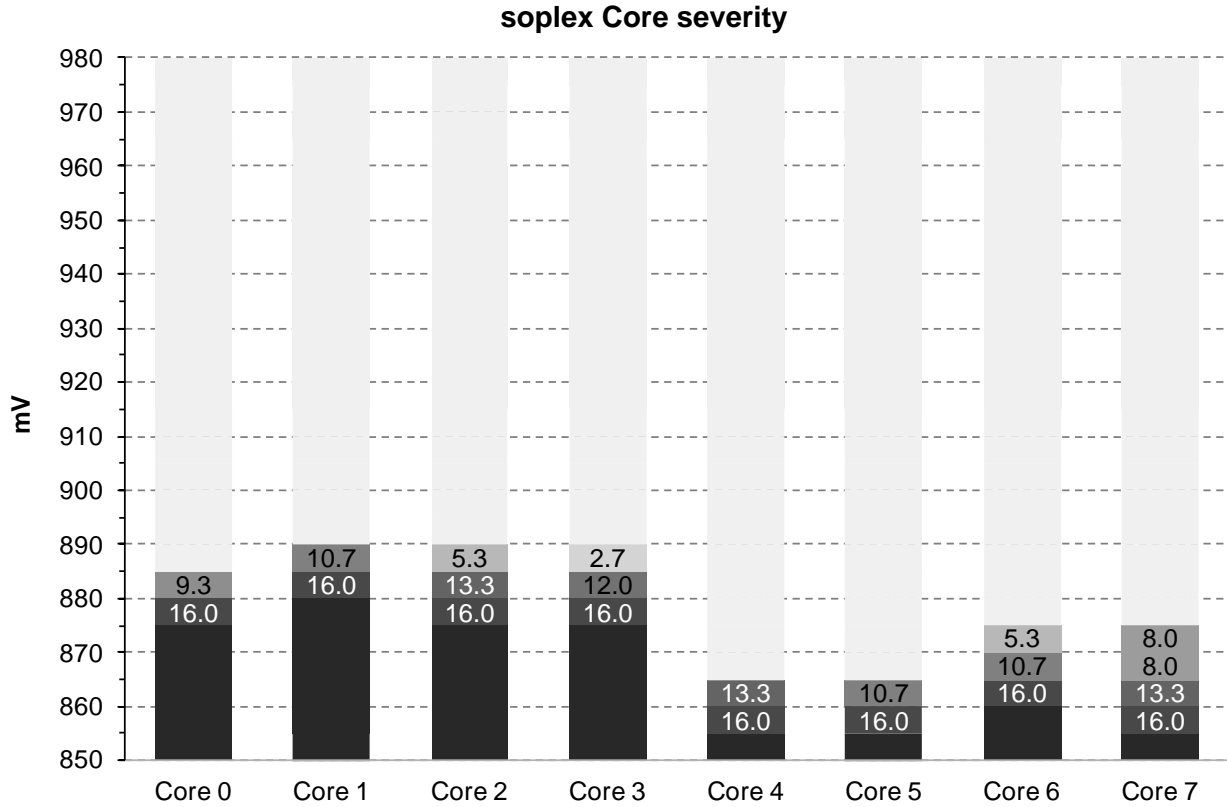


Figure 43: zeusmp benchmark severity on TTT chip cores.



**Figure 44: soplex benchmark severity on TTT chip cores.**

The severity function is essential primarily for speculation, because (1) it collects all needed information in one metric, (2) it incorporates the small divergences across multiple executions (10 executions in our case), (3) it measures the mean severity for each voltage step in each core, and (4) it assigns numbers to behaviors (SDC, CE, etc.), and thus, it is easy to use by any software daemon.

### 3.3.6 Suggestions for Undervolting Effects Mitigation in X-Gene 2-like CPUs

A static  $V_{min}$  point does not contain any information about the severity of operating at voltages below the safe  $V_{min}$ , but severity does so. Therefore, having knowledge about the severity below the safe  $V_{min}$  for each workload, one can decide if it is possible to be more aggressive to set the voltage below the safe  $V_{min}$ , and thus, to save more power.

Depending on the actual characterization findings ( $V_{min}$  and severity) for a CPU core during undervolting, certain hardware-based or software-based mitigation approaches can be employed to maximize the energy savings while preserving the correctness of program execution<sup>6</sup>. The primary aspect that determines the most suitable approach is the *first observed effect* as undervolting goes down the voltage levels. We select the following behaviors using the severity function described in subsection 3.3.5. For each

<sup>6</sup> Our severity metric can be used above existing circuit-based techniques such as adaptive clocking. For instance, in the mechanism proposed in [142] adaptive-clocking can reduce the voltage at which SDCs occur. The frequency with which adaptation is deployed can be an input to our framework, thereby limiting the potential for performance degradation due to excessive deployment of adaptive-clocking induced frequency slowdown.

case we describe the behavior, discuss the severity function values and corresponding mitigation approaches.

- **Nothing abnormal (severity=0).** The voltage range is absolutely safe (above the  $V_{min}$  of a core); no mitigation action is required. System operation in this range is the most conservative option and no mitigation provision is needed. Energy-savings are the minimum.
- **Corrected errors first (severity=1).** This is a voltage range with the behavior as the one observed in [7] [8] for Intel's Itanium (not in our X-Gene 2 machine). In such a case, ECC hardware serves as a proxy for abnormal behavior due to undervolting but program operation is still correct. Significant energy savings can be obtained without any mitigation other than the ECC correction but going further down the voltage is risky.
- **SDCs alone (severity=4) or with corrected and uncorrected errors (severity=5-7).** Voltage ranges with these behaviors generate incorrect program outputs and require extra mitigation approaches. The characterization of our X-Gene 2 system shows that the first abnormal behavior generated by undervolting belongs here for the majority of benchmarks and corrected errors as observed in [7] [8] do not appear first alone in our system. In particular, the cases where SDCs appear alone (severity=4) are the worst ones since there is no indication about the malfunction of the system; these areas should be avoided. When an eventual SDC (output mismatch) is accompanied by corrected or uncorrected error notifications, recovery actions can be employed such as rollback to a previously stored check-point or program re-execution in safe voltage and frequency combinations. There are also many applications that can tolerate SDCs and benefit from the severity function. These applications are (1) approximate computing algorithms, (2) video streaming and other image and video processing, (3) security-oriented applications such as jammer attacks detectors, etc. These applications are tolerant to faults, as they have minor impact on the returned output. For such applications, severity  $\leq 4$  can be used for improving energy efficiency.
- **Application and system crashes (or application timeouts) with or without corrected and uncorrected errors (severity 8-19).** Voltage levels with this behavior (the result of massive hardware malfunction) are well beyond the limits of cores operation in undervolted conditions. Application or system unresponsiveness is systematic in these ranges and unless serious hardware re-design is employed these ranges are unusable.

### 3.4 Design Enhancements

Undervolting characterization studies such as the one we report in this thesis can be used to provide hardware design recommendations for enhancements if the system (or its future revisions) is to be used in scaled voltage conditions for energy efficiency. There are some key hardware design guidelines that our analysis delivers for a system with similar behavior as the X-Gene 2 machine:

- **Stronger error protection.** SECDEC ECC protection at the lower levels of the memory hierarchy does not provide enough protection at lower voltages. If (a) stronger ECC codes are employed [146] [147] and (b) more blocks are protected, SDC behavior with or without errors will have significant probability to be transformed to corrected errors behavior similarly to [7] and [8]. Employing stronger ECC protection has been also reported in [148] for scaled voltage operation.

- **Hardware detectors.** If stronger ECC protection is too costly other types of hardware support can be employed for voltage emergencies detection such as the skitter circuit [149] [150] [151] also cited in [143] or the monitoring circuits used in Power7+ designs [84].
- **Finer-grained voltage domains.** Our characterization study shows that the coarse-grained voltage domains design of X-Gene 2 (a single voltage domain for all 8 cores) reduces the potential of energy savings since the voltage value of the domain is determined by its weakest core (the one with the higher  $V_{min}$ ). If each PMD was designed to operate on a separate voltage domain (similarly to the independent frequency domains per PMD) more aggressive voltage scaling (and energy savings) would be have been possible.

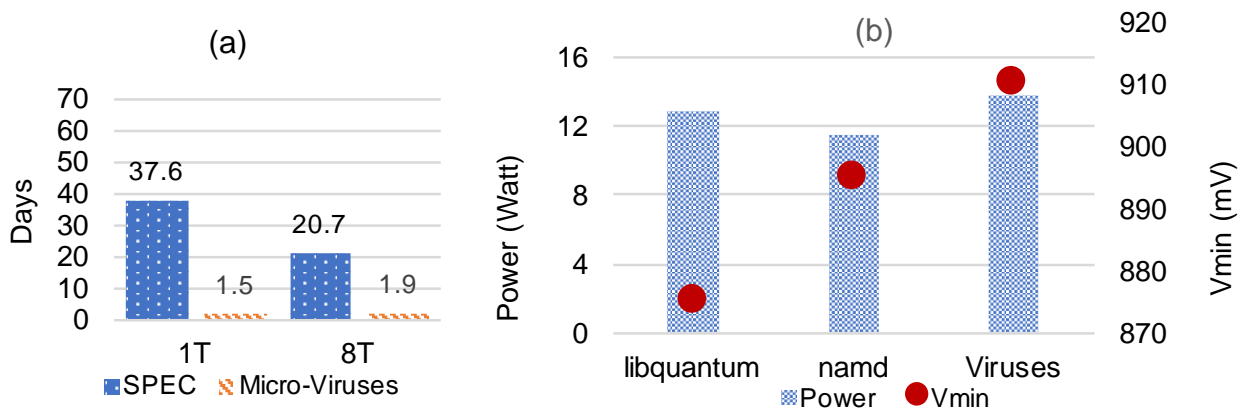
Of course, all the above hardware design modifications have their own design complexity, area and performance implications which must be jointly considered with the potential of energy savings through undervolting.

### 3.5 Fast System-Level Voltage Margins Characterization

The characterization procedure to identify these margins becomes more and more difficult and time consuming in modern multicore CPU chips, as the systems become more complex and non-deterministic and the number of cores is rapidly increasing [152]. In a multicore CPU design, there are significant opportunities for energy savings, because the variability of the safe margins is large among the cores of a chip, among the different workloads that can be executed on different cores of the same chip and among the chips of the same type.

The accurate identification of these limits in a real multicore system requires massive execution of a large number of real workloads (as we have seen in the previous sections) in all the cores of the chip (and all different chips of a system), for different voltage and frequency values. For instance, to identify the  $V_{min}$  of each of the eight cores of the Applied Micro's (APM) X-Gene 2 micro-server that is the experimental vehicle of this study, we used the SPEC CPU2006 benchmarks and repeated each experiment 10 times starting from the nominal voltage value (980mV) until their crash voltage value ( $\sim 880$ mV). These experiments required about 2 months for a complete characterization for all the cores of one microprocessor chip.

Figure 45(a) shows the estimated time for the characterization of one chip when the 12 SPEC CPU2006 benchmarks are used and when the micro-viruses (presented in the next subsections) are used. Note that the difference would have been much larger if all 29



**Figure 45: (a) Time needed for a complete system-level characterization to reveal the pessimistic margins for one chip. Programs are executed on individual cores (1T) and on all 8 cores concurrently (8T). (b) Safe  $V_{min}$  values and their independence on power consumption.**

SPEC CPU2006 benchmarks were used. The excessively long time that SPEC-based or similar characterization takes, forces manufacturers to introduce the same pessimistic guardband for all the cores of the same multicore chips. Clearly, if shorter benchmarks are able to reveal the  $V_{min}$  of each core of a multicore chip (or the  $V_{min}$  of different chips) faster than exhaustive campaigns, finer-grained exploitation of the operational limits of the chips and their cores can be effectively employed for energy-efficient execution of the workloads.

In this section, we introduce the development of dedicated programs (diagnostic micro-viruses), which is the fourth contribution of this thesis and presented in [12]. Micro-viruses aim to stress the fundamental hardware components of three different chips (one “nominal” and two corner parts) of APM’s X-Gene 2 micro-server family, that are ARMv8-based multicore CPUs manufactured in 28nm.

With our diagnostic micro-viruses, we effectively stress (individually or simultaneously) all the main components of the chip:

- c. the caches (the L1 data and instruction caches, the unified L2 caches and the last level L3 cache of the chips) and
- d. the two main functional components of the pipeline (the ALU and the FPU).

These diagnostic micro-viruses are executed in very short time (~3 days for the entire massive characterization campaign for each individual core for each one microprocessor chip) compared to normal benchmarks such as those of the SPEC CPU2006 suite which need 2 months as Figure 45(a) shows. The micro-viruses’ purpose is to reveal the variation of the safe voltage margins across cores of the multicore chip and also to contribute to diagnosis by exposing and classifying the abnormal behaviour of each CPU unit (silent data corruptions, bit-cell errors, and timing failures).

There have been many efforts towards writing power viruses and stress benchmarks. For example, SYMPO [159], an automatic system level max power virus generation framework, which maximizes the power consumption of the CPU and the memory system, MAMPO [160], as well as the MPrime [161] and stress-ng [162] are the most popular benchmarks, which aim to increase the power consumption of the microprocessor by torturing it; they have been used for testing the stability of the microprocessor during overclocking. However, power viruses are not capable to reveal pessimistic voltage margins.

Figure 45(b) shows that the power consumption of a workload is not correlated to the safe  $V_{min}$  (and thus to voltage guardbands) of a core. According to our experiments (presented in sections 3.7 and 3.8), *libquantum* is the most power-hungry benchmark (in the majority of X-Gene 2 cores) among the 12 SPEC CPU2006 benchmarks we used. However, *libquantum*’s safe  $V_{min}$  is significantly lower (20 mV) than the *namd* benchmark, which has lower power consumption.

The purpose of the proposed micro-viruses is to stress individually the fundamental microprocessor units (caches, ALU, FPU) that define the voltage margins variability of the microprocessor (previous studies [7] [8] [11] [46] [84] [157] [158] have shown the importance of these units for the  $V_{min}$ ). We do not aim to reveal the absolute  $V_{min}$  (which can be identified by worst-case voltage noise stress programs). However, we provide strong evidence (IPC and power measurements) that the micro-viruses stress the chips more intensively than the SPEC CPU2006 benchmarks.

### 3.6 Micro-Viruses Description

For the construction of the diagnostic micro-viruses we followed two different principles for the tests that target the caches and the pipeline, respectively. All micro-viruses are



small *self-checking* pieces of code. This means that the micro-viruses check if a read value is the expected one or not. There are previous studies (e.g., [163] and [164]) for the construction of such tests, but they focus only on error detection (mainly for permanent errors), and to our knowledge this is the first study that is performed on actual microprocessor chips; not in simulators or RTL level, which have no interference with the operating system and the corresponding challenges.

In this section, we first present the details of the caches of the X-Gene 2 in Table 15 (the rest important X-Gene 2's specifications were discussed previously in subsection 3.1) and then a brief overview of the challenges for the development of such system-level micro-viruses in a real hardware and the decisions we made in order to develop accurate self-checking tests for the caches and the pipeline.

**Caches:** For all levels of caches the first goal of the developed micro-viruses is to flip all the bits of each cache block from zero to one and vice versa. When the cache array is completely filled with the desired data, the micro-virus reads iteratively all the cache blocks while the chip operates in reduced voltage conditions and identifies any corruptions of the written values, which cannot be detected by dedicated hardware mechanisms of the cache, such as the parity protection that can detect only odd number of flips.

All caches in X-Gene 2 have pseudo-LRU replacement policy. All our micro-viruses focusing on any cache level need to “warm-up” the cache before the test begins, by iteratively accessing the desired data in order to ensure that all the ways of the cache are completely filled and accessed with the micro-viruses’ desired patterns. We experimentally observed through the performance monitoring counters that the safe

**Table 15: The X-Gene 2 cache specifications.**

	<b>L1I</b>	<b>L1D</b>	<b>L2</b>	<b>L3</b>
<b>Size</b>	32 KB	32 KB	256 KB	8 MB
<b># of Ways</b>	8	8	32	32
<b>Block Size</b>	64 B	64 B	64 B	64 B
<b># of Blocks</b>	512	512	4096	131,072
<b># of Sets</b>	64	64	128	4096
<b>Write Policy</b>	-	Write-through	Write-Back	-
<b>Write Miss Policy</b>	No-write allocate	No-write allocate	Write allocate	-
<b>Organization</b>	Physically Indexed Physically Tagged (PIPT)	Physically Indexed Physically Tagged (PIPT)	Physically Indexed Physically Tagged (PIPT)	Physically Indexed Physically Tagged (PIPT)
<b>Prefetcher</b>	YES	YES	YES	NO
<b>Scope</b>	Per Core	Per Core	Per PMD	Shared
<b>Protection</b>	Parity Protected	Parity Protected	ECC Protected	ECC Protected

number of iterations that “warm-up” the cache with the desired data, before the checking phase begins, is

$$\log_2(\text{number of ways})$$

to guarantee that the cache is filled only with the data of the diagnostic micro-virus.

In order to validate the operation of the entire cache array, it is important to perform write/read operations in all bit cells. For every cache level, we allocate a memory chunk equal to the targeted cache size. As the storing of data is performed in cache block granularity, we need to make sure that our data storage is block-aligned, otherwise we will encounter undesirable block replacements that will break the requirement for complete utilization of the cache array.

Assume for example that the first word of the physical frame will be placed at the middle of the cache block. This means that when the micro-virus fills the cache, practically, there will be half-block size words that will replace a desired previously fetched block in the cache. Thus, if the cache blocks are  $N$ , the number of blocks that will be written in the cache will be  $N + 1$  (which means that one cache block will get replaced), and thus, the self-checking property may be jeopardized. To this end, for all cache-related micro-viruses we perform a check at the beginning of the test to guarantee that the allocated array is cache aligned (to be block aligned afterwards).

Another factor that has to be considered in order to achieve full coverage of the cache array, is the *cache coloring* [165]. Unless the memory is a fully associative one (which is not the case of the ARMv8 microprocessors), every store operation is indexed at one cache block depending on its address. For physically indexed memories, the physical address of the datum or instruction is used. However, because the physical addresses are not known or accessible from the software layer, special precautions need to be taken in order to avoid unnecessary replacements. To address this issue, we exploit a technique that is used to improve cache performance, known as *cache coloring* [165]. If the indexing range of the memory is larger than the virtual page, two addresses with the same offset on different virtual pages are likely to conflict on the same cache block (due to the 32KB size of the L1 caches the bits that index the cache occur in page offset, and thus, there is no conflict; this is the case for L2 and L3 caches in our system). To avoid this situation, the indexing range is separated in regions equal to the page size, known as *colors*. It is then enough to use equal number of pages in each color to avoid conflicts. The easiest way to achieve this, is to allocate contiguous physical address range, which is possible at the kernel level using the *kmalloc()* call. The contiguous physical range will guarantee that all the data will be placed and fully occupy the cache, without replacements or unoccupied blocks (see subsections 3.6.1 to 3.6.4).

Another challenge that the micro-viruses need to take into consideration is the interference of the branch predictors and the cache prefetchers. In our micro-viruses, the branch prediction mechanism (in particular the branch mispredictions that can flush the entire pipeline) may ruin the self-checking property of the micro-virus, by replacing or invalidating the necessary data or instruction patterns. Moreover, prefetching requests can modify the pre-defined access patterns of the micro-virus execution.

To eliminate these effects, the memory access patterns of the micro-viruses are modelled using the stride-based model for each of the static loads and stores of the micro-virus. Each of the static loads and stores in the workload walk a bounded array of memory references with a constant stride, larger than the X-Gene 2’s prefetcher stride. In that way, the cache-related micro-viruses are executed *without the interference* of the branch predictor or the prefetcher. We validate this by leveraging the performance counters that

measure the prefetch requests for the L1 and L2 caches and the mispredictions, and no micro-virus counts any event in the related counters.

**Pipeline:** For the pipeline, we developed dedicated benchmarks that stress: (i) the Floating-Point Unit (FPU), (ii) the integer Arithmetic Logical Units (ALUs) and (iii) the entire pipeline using a combination of loads, stores, branches, arithmetic and floating-point unit operations. The goal is to trigger the critical paths that could possibly lead to an error during off-nominal operation voltage conditions.

Generally, for all micro-viruses, one primary aspect that we need to take into consideration is that due to the micro-viruses' execution in the real hardware with operating system, we need to isolate all the system's tasks to a single core. Assume for example that we run the L1 data or instruction micro-virus in Core 0. Each core has its own L1 cache, so we isolate all the system processes and interrupts in the Core 7, and we assign the micro-virus to Core 0. To realize this, we use the *sched\_setaffinity()* call of the Linux kernel to set the process' affinity (execution in particular cores). In such a way, we ensure that only the micro-virus is executed in the desired core each time. We follow the same concept for all micro-viruses, except for L3 cache, because L3 is shared among all cores, so a small noise from system processes is unavoidable.

We developed all diagnostic micro-viruses in C language (except for L1 Instruction cache micro-virus, which is ISA-dependent and is developed with a mix of C and ARMv8 assembly instructions). Moreover, the micro-viruses (except for L1 instruction cache's) check the microprocessor's parameters (cache size, #ways, existence of prefetcher, page size, etc.) and adjust the micro-viruses code to the specific CPU. This way, the micro-viruses can be executed in any microarchitecture and can easily adapted to different ISAs.

### 3.6.1 L1 Data Cache Micro-Virus

For the first level data cache of each core, we defined statically an array in memory with the same size as the L1 data cache. As the L1 data cache is *no-write allocate*, after the first write of the desired pattern in all the words of the structure we need to read them first, in order to bring all the blocks in the first level of data cache. Otherwise, the blocks would remain in the L2 cache and we would have only-write misses in the L2 cache.

Moreover, due to the pseudo-LRU policy that is used in the L1 data cache, we read all the words of the cache:

$$\log_2(\text{number of ways of L1D cache}) = \log_2(8) = 3$$

(three consecutive times) before the test begins, in order to ensure that all the blocks with the desired patterns are allocated in the first level data cache. With these steps, we achieve 100% read hit in the L1 data cache during the execution of the L1D micro-virus in undervolted conditions. The L1 Data micro-virus fills the L1 Data cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property of the micro-virus (correctness of execution is determined by the micro-virus itself and not externally), at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

### 3.6.2 L1 Instruction Cache Micro-Virus

The concept behind the L1 Instruction Cache micro-virus is to flip all the bits of the instruction encoding in the cache block from zero to one and vice versa. In the ARMv8 ISA there is no single pair of instructions that can be employed to invert all 32 bits of an instruction word in the cache, so to achieve this we had to employ multiple instructions. The instructions listed in Table 16 are able to flip all the bits in the instruction cache from 0 to 1 and vice versa according to the Instruction Encoding Section of the ARMv8 Manual [166]

Each cache block of the L1 instruction cache holds 16 instructions because each instruction is 32-bit in ARMv8 and the L1 Instruction cache block size is 64 bytes. The size of each way of the L1 Instruction Cache is  $32\text{KB} / 8 = 4\text{KB}$ , and thus, it is equal to the page size which is 4KB. As a result, there should be no conflict misses when accessing a code segment (see cache coloring previously discussed) with size equal to the L1 Instruction Cache (the same argument holds also for the L1 Data Cache).

The method that guarantees the self-checking property in the L1 Instruction cache micro-virus is the following: The L1 instruction cache array holds 8192 instructions (64 sets x 8 ways x 16 instructions in each cache block = 8192). We use 8177 instructions to hold the instructions of our diagnostic micro-virus, and the remaining 15 instructions ( $8177 + 15 = 8192$ ) to compose the control logic of the self-checking property and the loop control.

More specifically, we execute iteratively 8177 instructions and at the end of this block of code we expect the destination registers to hold a specific “signature” (the signature is the same for each iteration of the same group of instructions, but different among different executed instructions). If this “signature” is distorted, then the micro-virus detects that an error occurred (for instance a bit flip in an immediate instruction resulted in the addition of a different value) and records the location of the faulty instruction as well as the expected and the faulty signature for further diagnosis. We iterate this code multiple times and after that we continue with the next block of code.

As in the L1 Data cache micro-virus, due to the pseudo-LRU policy that is used also in the L1 Instruction cache, we fetch all the instructions

$$\log_2(\text{number of ways of L1I cache}) = \log_2(8) = 3$$

**Table 16: ARMv8 instructions used in the L1I micro-virus. The right column presents the encoding of each instruction to demonstrate that all cache block bits get flipped.**

Instruction	Encoding
<b>add x28, x28, #0x1</b>	1001 0001 0000 0000 0000 0111 1001 1100
<b>sub x3, x3, #0xffe</b>	1101 0001 0011 1111 1111 1000 0110 0011
<b>madd x28, x28, x27, x27</b>	1001 1011 0001 1011 0110 1111 1001 1100
<b>add x28, x28, x27, asr #2</b>	1000 1011 1001 1011 0000 1011 1001 1100
<b>add w28, w28, w27, lsr #2</b>	0000 1011 0101 1011 0000 1011 1001 1100
<b>nop</b>	1101 0101 0000 0011 0010 0000 0001 1111
<b>bics x28, x28, x27</b>	1110 1010 0011 1011 0000 0011 1001 1100

(three consecutive times) before the test begins, to ensure that all blocks with the desired instruction patterns are allocated in the L1 instruction cache. With these steps, we achieve 100% cache read hit (and thus cache stressing) during undervolting campaigns.

### 3.6.3 L2 Cache Micro-Virus

The L2 cache is a 32-way associative PIPT cache with 128 sets; thus, the bits of the physical address that determine the block placement in the L2 cache are bits [12:6] (as shown in Figure 46). Moreover, the page size we rely on is 4KB and consequently the page offset consists of the 12 least significant bits of the physical address. Accordingly, the most significant bit (bit 12) of the set index (the dotted square in Figure 46) is not a part of the page offset. If this bit is equal to 1, then the block is placed in any set of the upper half of the cache, and in the same manner, if this bit is equal to 0, the block is placed in a set of the lower half of the cache. Bits [11:6] which are part of page/frame offset determine all the available sets for each individual half.

In order to guarantee the maximum block coverage (e.g., to completely fill the L2 cache array), and thus to fully stress the cache array, the L2 micro-virus should not depend on the MMU translations that may result in increased conflict misses. The way to achieve this is by allocating memory that is not only virtually contiguous (as with the standard C memory allocation functions used in user space), but also physically contiguous by using the *kmalloc()* function (see cache coloring discussed in section 3.6). The *kmalloc()* function operates similarly to that of user-space's familiar memory allocation functions, with the main difference that the region of physical memory allocated by *kmalloc()* is *physically contiguous*. This guarantees that in one half of the allocated physical pages, the most significant bits of their set index are equal to one and the other half are equal to zero.<sup>7</sup>

Given that the replacement policy of the L2 cache is also pseudo-LRU, the L2 micro-virus needs to iteratively access

$$\log_2(\text{number of ways of L2 cache}) = \log_2(32) = 5$$

(five times) the allocated data array, to ensure that all the ways of each set contain the correct pattern. Furthermore, due to the fact that the L1 data cache has write-through policy and the L2 cache has write allocate policy, the stored data will reside in the L2 cache right after the initial writes (no write backs).

Another requirement for the L2 micro-virus is that it should access the data only from the L2 cache during the test and not from the L1 data cache, to completely stress the former one. We meet this requirement using a stride access scheme for the array with a one-block (8 words) stride. Therefore, in the first iteration the L2 micro-virus accesses the first word of each block, in the second iteration it accesses the second word of each block, and so on. Thus, it always misses the L1 Data cache. By accessing the data using these strides, the L2 micro-virus also overcomes the prefetching requests. Note that the L1 instruction cache can completely hold all the L2 diagnostic micro-virus instructions, so the L2 cache holds only the data of our test.

---

<sup>7</sup> Our Linux kernel was built with the commonly used page size of 4KB; if the page size is 64KB in another CPU, the micro-virus uses standard C memory allocation functions in user space instead of *kmalloc()*, because the most significant bit of the set index would be part of the page offset like the rest of the set index bits.

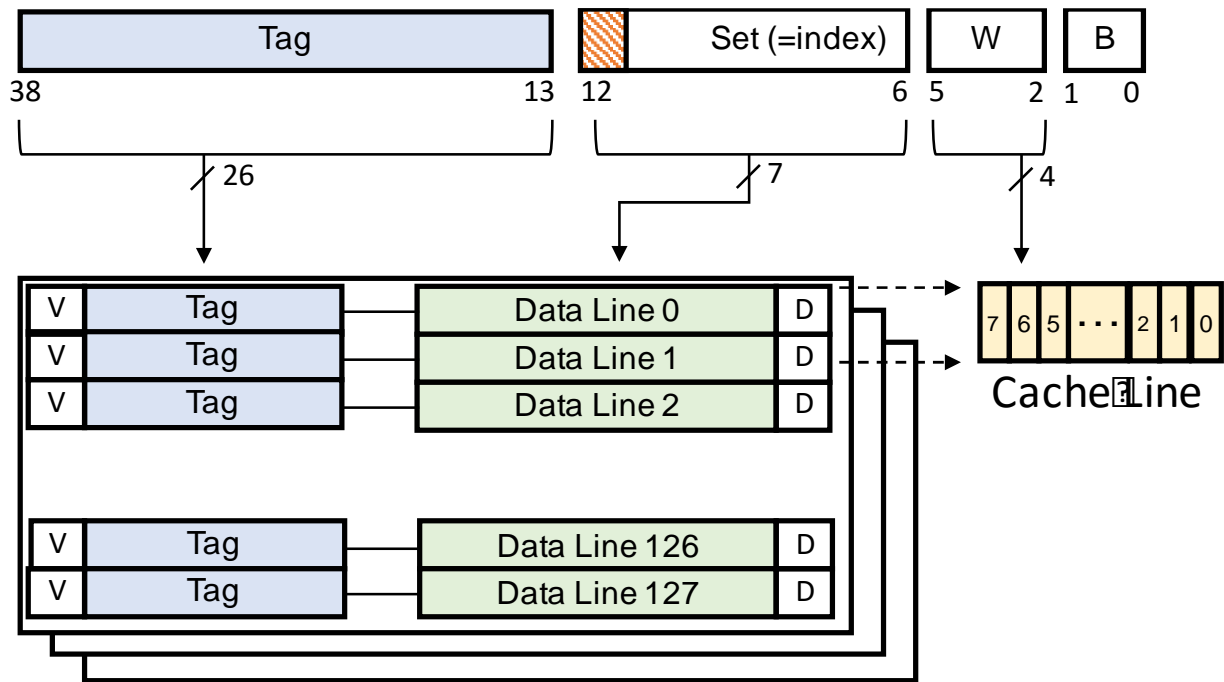


Figure 46: A 256KB 32-way set associative L2 cache.

To validate the above, we isolated all the system processes by forcing them to run on different cores from the one that executes the L2 diagnostic micro-virus, by setting the system processes' CPU affinity and interrupts to a different core, and we measured the L1 and L2 accesses and misses after we have already "trained" the pseudo-LRU with the initial accesses. We measure these micro-architectural events by leveraging the built-in performance counters of the CPU<sup>8</sup>.

The performance counters show that the L2 diagnostic micro-virus always misses the L1 Data cache and always hits the L1 Instruction cache, while it hits the L2 cache in the majority of the accesses. Specifically, the L2 cache has 4096 blocks and the maximum number of block-misses we observed was 32 at most for each execution of the test (meaning 99.2% coverage). In that way, we verify that the L2 micro-virus completely fills the L2 cache.

The L2 micro-virus fills the L2 cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property into this micro-virus, at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

### 3.6.4 L3 Cache Micro-Virus

The L3 cache is a 32-way associative PIPT cache with 4096 sets and is organized in 32 banks; so, each bank has 128 sets and 32 ways. Moreover, the bits of the physical address that determine the block placement in the L3 cache are the bits [12:6] (for choosing the set in a particular bank) and the bits [19:15] for choosing the correct bank.

<sup>8</sup> We developed a kernel module able to provide access to the performance counters (see ANNEX I) from user space. We did not use tools like Perf [144] or PAPI [167] because these tools provide an extra overhead in measurements (+/- 3%), while we need accurate values to validate the proposed micro-viruses in system-level.

Based on the above, in order to fill the L3 cache we allocate physically contiguous memory with *kmalloc()* (as we described previously in subsection 3.6.3).

However, *kmalloc()* has an upper limit of 128 KB in older Linux kernels and 4MB in newer kernels (like the one we are using; we use CentOS 7.3 with Linux kernel 4.3). This upper limit is a function of the page size and the number of buddy system free lists (MAX\_ORDER). The workaround to this constraint is to allocate two arrays with two calls to *kmalloc()* and each array's size should be half the size of the 8M L3 cache. The reason that this approach will result in full block coverage in the L3 cache is that 4MB chunks of physically contiguous memory gives us contiguously the 22 least significant bits, while we need contiguously only the 20 least significant (for the set index and the bank index). Moreover, we should highlight that the L3 cache is as a non-inclusive victim cache.

In response to an L2 cache miss from one of the PMDs, agents forward data directly to the L2 cache of the requestor, bypassing the L3 cache. Afterwards, if the corresponding fill replaces a block in the L2 cache, a write-back request is issued, and the evicted block is allocated into the L3 cache. On a request that hits the L3 cache, the L3 cache forwards the data and invalidates its copy, freeing up space for future evictions. Since data may be forwarded directly from any L2 cache, without passing through the L3 cache, the behavior of the L3 cache increases the effective caching capacity in the system.

Due to the pseudo-LRU policy, similar to the L2, the L3 micro-virus is designed accordingly to perform

$$\log_2(\text{number of ways of L2 cache}) = \log_2(32) = 5$$

(five) sequential writes to cover all the ways before the test begins, and the read operations afterwards are performed by stride of one block (to bypass the L2 cache and the prefetcher, so the micro-virus only hits the L3 cache and always misses the L1 and L2 caches).

The L3 diagnostic micro-virus fills the L3 cache with three different patterns, each of which corresponds to a different micro-virus test. These tests are again the all-zeros, the all-ones, and the checkerboard pattern. To enable the self-checking property, at the end of the test we check if each fetched word is equal to the expected value (the one stored before the test begins).

However, in contrast to the L2 diagnostic micro-virus, in the L3 micro-virus there is no way to prove the complete coverage of the L3 cache in system-level because that there are no built-in performance counters in X-Gene 2 that report the L3 accesses and misses. However, by using the events that correspond to the L1 and L2 accesses, misses and write backs we check that all the requests from the L3 micro-virus miss the L1 and L2 caches, and thus only hit the L3 cache. Finally, we should highlight that the shared nature of the L3 cache forced us to try to minimize the number of the running daemons in the system in order to reduce the noise in the L3 cache from their access to it.

### 3.6.5 Arithmetic and Logic Unit (ALU) Micro-Virus

X-Gene 2 features a 4-wide out-of-order superscalar microarchitecture. It has one integer scheduler and two different integer pipelines:

- a Simple Integer pipeline, and
- a Simple+Complex Integer pipeline.

The integer scheduler can issue two integer operations per cycle; each of the other schedulers can issue one operation per cycle (the integer scheduler can issue 2 simple

integer operations per cycles; for instance, 2 additions, or 1 simple and 1 complex integer operation; for instance, 1 multiplication and 1 addition).

The execution units are fully pipelined for all operations, including multiplications and multiply-add instructions. ALU operations are single-cycle. The fetch stage can bring up to 16 instructions (same size as a cache block) per cycle from the same cache block or by two adjacent cache blocks. If the fetch begins in the middle of a cache block (unaligned), the next cache block will also be fetched in order to have 16 instructions available for further processing, and thus there will be a block replacement on the Instruction Buffer.

To this end, we use NOP instructions to ensure that the first instruction of the execution block is block aligned, so that the whole cache block is located to the instruction buffer each time. For the above microarchitecture, we developed the ALU self-testing micro-virus, which avoids data and control hazards and iterates 1000 times over a block of 16 instructions (that resides in the Instruction buffer, and thus the L1 instruction and data cache are not involved in the stress testing process). After completing 1000 iterations, it checks the value of the registers involved in the calculations by comparing them with the expected values.

After re-initializing the values of the registers, we repeat the same test 70M times, which is approximately 60 seconds of total execution (of course, the number of executions and the overall time can be adjusted). Therefore, we execute code that resides in the instruction buffer for 1000 iterations of our loop and then we execute code that resides in 1 block of the cache after the end of these 1000 iterations. As the instructions are issued and categorized in groups of 4 (X-Gene 2 issues 4 instructions) and the integer scheduler can issue 2 of them per cycle, we can't achieve the theoretical optimal IPC of 4 Instructions per Cycle only with Integer Operations. Furthermore, we try to have in each group of 4 instructions, instructions that stress all the units of all the issue queues like the adder, the shifter and multiplier. Specifically, the ALU micro-virus consists of 94% integer operations and 6% branches.

### 3.6.6 Floating-Point Unit (FPU) Micro-Virus

Aiming to heavily stress and diagnose the FPU, we perform a mix of diverse floating-point operations by avoiding data hazards (thus stalls) among the instructions and using different inputs to test as many bits and combinations as possible. To implement the self-checking property of the micro-virus, we execute the floating-point operations twice, with the same input registers and different result registers. If the destination registers of these two same operations have different result, our self-test notifies that an error occurred during the computations.

For every iteration, the values of the registers (for all of the FPU operations) are increased by a non-fixed stride that is based on the calculations that take place. The values in the registers of each loop are distinct between them and between every loop. Moreover, we ensure that the first instruction of the execution block is cache aligned (as in the ALU micro-virus), so the whole cache block is located to the instruction buffer each time.

### 3.6.7 Pipeline Micro-Virus

Apart from the dedicated benchmarks that stress independently the ALU and the FPU, we have also constructed a micro-virus to stresses simultaneously all the issue queues of the pipeline. Between two consecutive "heavy" (high activity) floating-point instructions of the FPU test (like the consecutive *multiply add*, or the *fsqrt* which follows the *fdiv*) we add a small iteration over 24 array elements of an integer array and a floating-point array.



To this end, during these iterations, the “costly” instructions such as *multiply add* have more than enough cycles to calculate their result, while at the same time we perform load, store, integer multiplication, exclusive or, subtractions and branches. All instructions and data of this micro-virus are located in L1 cache in order to fetch them at the same cycle to avoid high cache access latency. As a result, the “pipeline” micro-virus has a large variety of instructions which stress in parallel all integer and FP units. This micro-virus consists of 65% integer operations and 23.1% floating point operations, while the rest 11.9% are branches.

### 3.7 Micro-Viruses Validation

In the previous section we described the challenges and our solutions to the complex development process of the micro-viruses and how we verified their coverage using the machine performance monitoring counters. However, it is essential to validate the stress and utilization of the micro-viruses on the microprocessor. To this end, we measure the IPC and power consumption for both micro-viruses and SPEC CPU2006 benchmarks. Note that the micro-viruses were neither developed to provide power measurements nor performance measurements.

We present the IPC and power consumption measurements of the micro-viruses only to verify that they sufficiently stress the targeted units. IPC and power consumption along with the data footprints of the micro-viruses (complete coverage of the caches bit arrays; see previous section) are highly accurate indicators of the activity and utilization of a workload on a microprocessor. Figure 47 presents the IPC, and Figure 48 and Figure 49 present the power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks.

As shown in Figure 47, the proposed micro-viruses for fast voltage margins variability identification provide very high IPC compared to most SPEC benchmarks on the target X-Gene 2 CPU. In addition, we assessed the power consumption using the dedicated power sensors of the X-Gene 2 microprocessor (located in the standby power domain; see section 3.1), to take accurate results for each workload. We performed measurements for two different voltage values; at the nominal voltage (980mV) and at 920mV, which is a voltage step that all of the micro-viruses and benchmarks can be reliably executed (without Silent Data Corruptions (SDCs), detected/corrected errors or crashes). Figure 48 and Figure 49 show that the maximum and average power consumptions of the micro-viruses are comparable to the SPEC CPU2006. In the same figure, we can also see the differences concerning the energy efficiency when operating below nominal voltage conditions, which emphasizes the need to identify the pessimistic

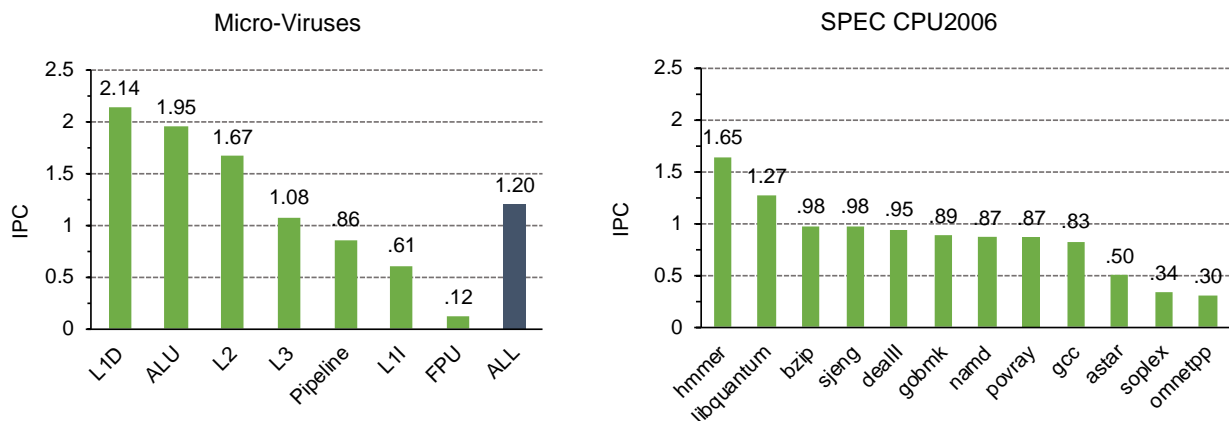
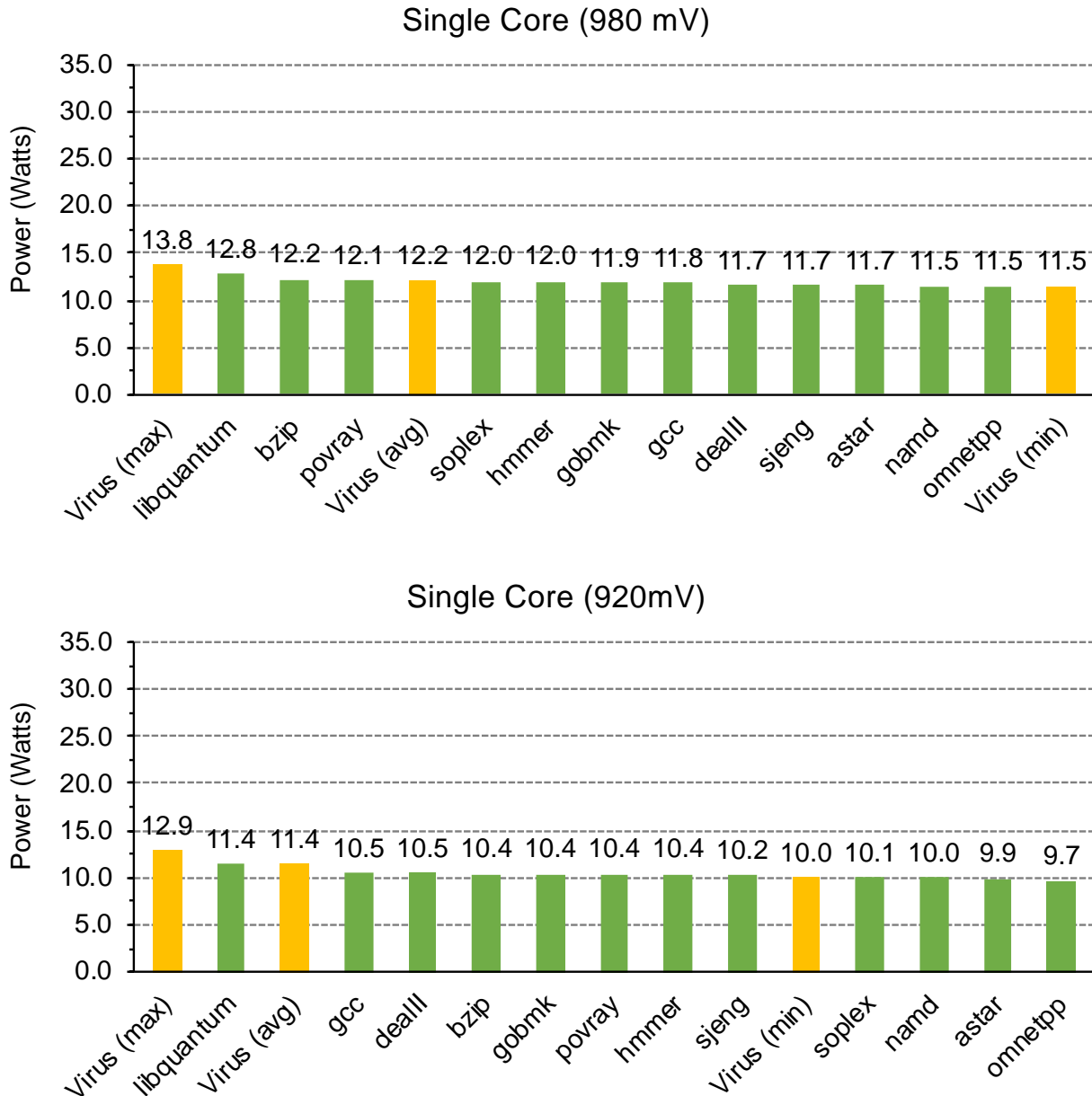


Figure 47: IPC measurements for both micro-viruses (top) and SPEC CPU2006 benchmarks (bottom).

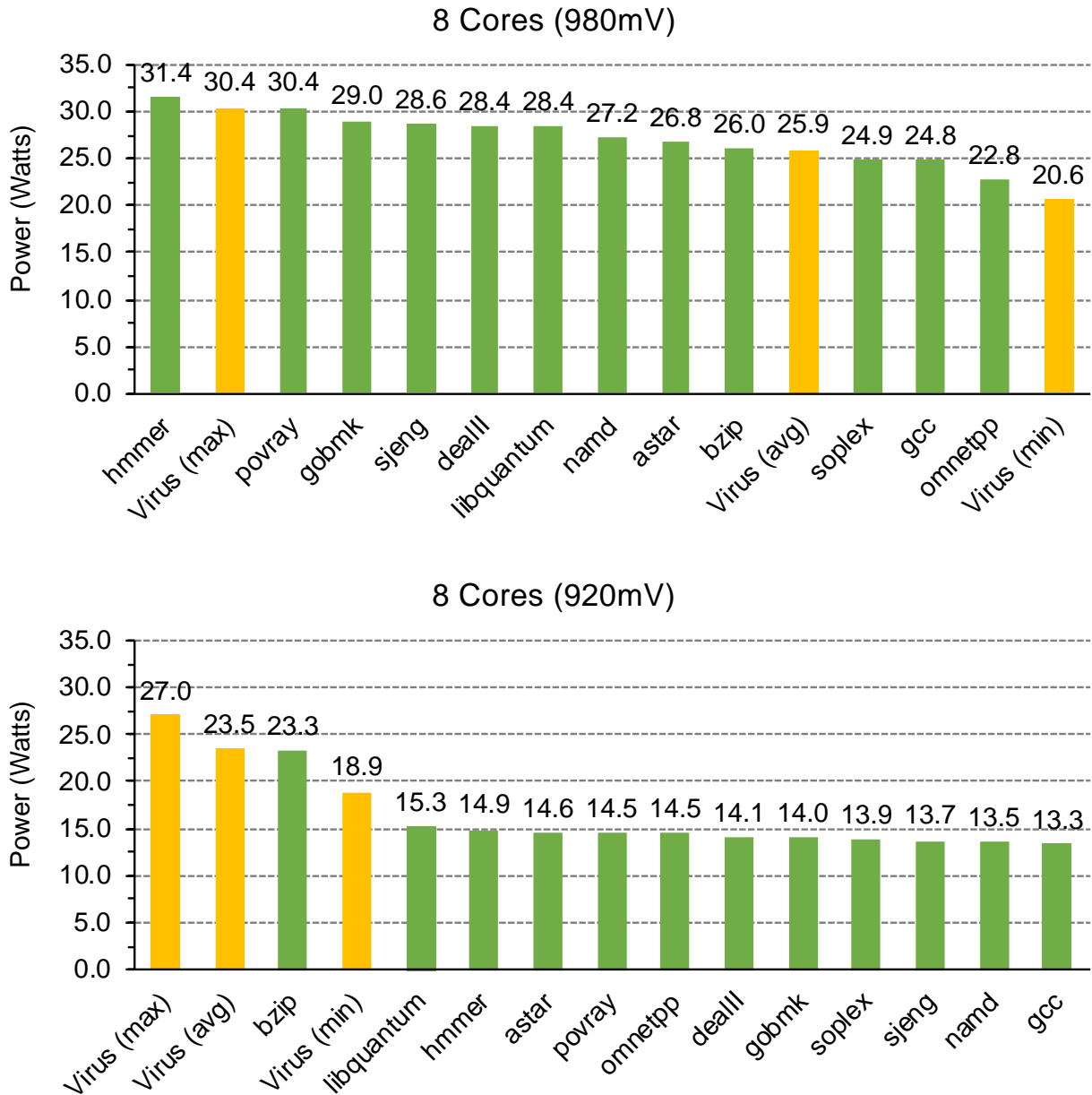


**Figure 48: Power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks. The upper graph shows the power consumption at nominal voltage (980 mV). The lower graph shows the power measurements when the microprocessor operates at 920mV, in order to present the energy efficiency when operating below nominal voltage conditions. Both graphs present single-core executions.**

voltage margins of a microprocessor. As we can see, in the multi-core execution we can achieve 12.6% energy savings (considering that the maximum TDP of X-Gen 2 is 35W), by reducing the voltage 6.2% below nominal, where all of the three chips operate reliably.

### 3.8 Experimental Evaluation

For the evaluation of the micro-viruses' ability to reveal the  $V_{min}$  of X-Gen 2 CPU chips and their cores, we used three different chips: TTT, TFF, and TSS from Applied Micro's X-Gen 2 micro-server family. The TTT part is the nominal (typical) part. The TFF is the fast-corner part, which has high leakage but at the same time can operate at higher frequency (fast chip). The TSS part is also corner part which has low leakage and works at lower frequency. The faster parts (TFF) are rated for higher frequency and usually sold for more money, while slower parts (TSS) are rated for lower frequency. In any event, the



**Figure 49: Power consumption measurements for both the micro-viruses and the SPEC CPU2006 benchmarks. The upper graph shows the power consumption at nominal voltage (980 mV). The lower graph shows the power measurements when the microprocessor operates at 920mV, in order to present the energy efficiency when operating below nominal voltage conditions. Both graphs present results when programs are executed in 8 cores concurrently.**

parts must still work in the slowest environment, and thus, all chips (TTT, TSS, TFF) operate reliably with nominal frequency at 2.4GHz.

Using the I2C controller we decrease the voltage of the domains of the PMDs and the SoC at 5mV steps, until the lowest voltage point (safe  $V_{min}$ ) before the occurrence of any error (corrected and uncorrected – reported by the hardware ECC), SDC (Silent Data Corruption – output mismatch) or Crash. To account for the non-deterministic behavior of a real machine (all of our experiments were performed on the actual X-Gen 2 chip), we repeat each experiment 10 times and we select the execution with *the highest safe  $V_{min}$*  (the worst-case scenario) to compare with the micro-viruses.

We experimentally obtained also the safe  $V_{min}$  values of the 12 SPEC CPU2006 benchmarks on the three X-Gen 2 chips (TTT, TFF, TSS), running the entire time-

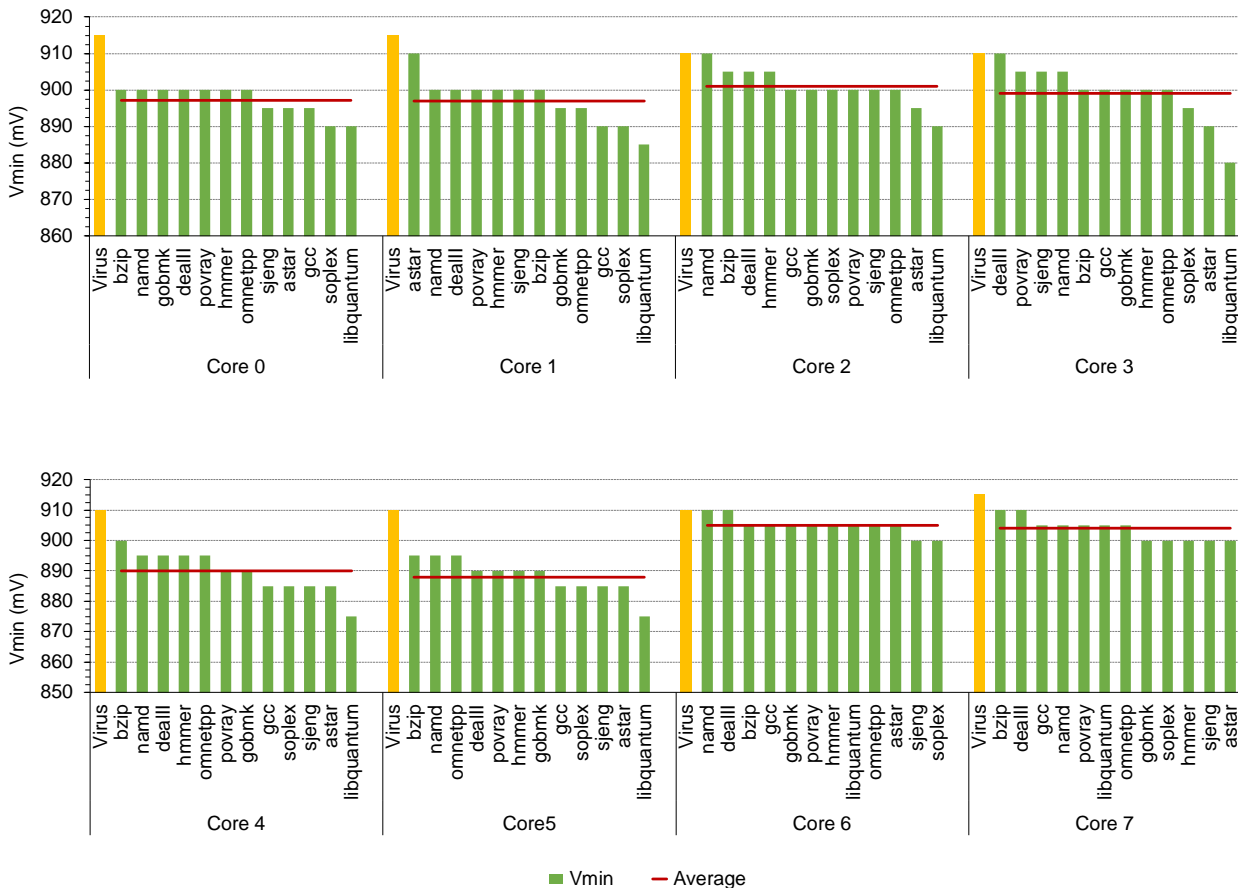
consuming undervolting experiment 10 times for each benchmark. These experiments were performed during a period of 2 months on a single X-Gene 2 machine, that is 6 months for all 3 chips (see Figure 45 for the time needed for one chip). We also ran our diagnostic micro-viruses, with the same setup for the 3 different chips, as for the SPEC CPU2006 benchmarks. This part of our study focuses on:

1. the quantitative analysis of the safe  $V_{min}$  for three significantly different chips of the same architecture to expose the potential guard-bands of each chip,
2. the demonstration of the value of our diagnostic micro-viruses which can stress the individual components, and reveal virtually the same voltage guard-bands compared to benchmarks.

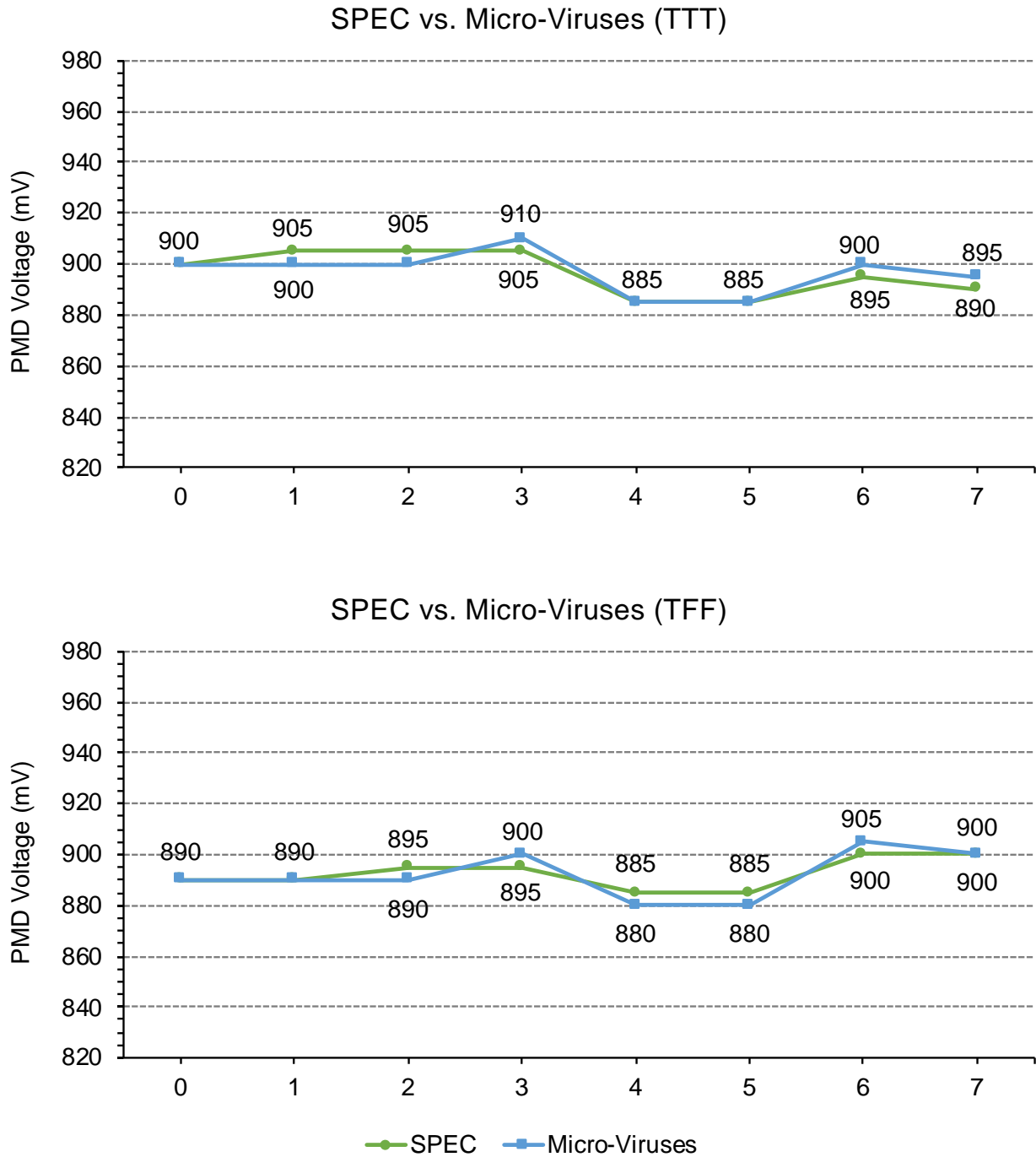
The voltage guardband for each program (benchmark or micro-virus) is defined as the safest voltage margin between the nominal voltage of the microprocessor and its safe  $V_{min}$  (where no ECC errors or any other abnormal behavior occur).

### 3.8.1 SPEC Benchmarks vs. Micro-Viruses

As we discussed earlier, to expose these voltage margins variability among cores in the same chip and among the three different chips by using the 12 SPEC CPU2006 benchmarks, we needed ~2 months for each chip. On the contrary, the same experimentation by using the micro-viruses needs ~3 days (as Figure 45 presents), which can expose the corresponding safe  $V_{min}$  for each core. In Figure 50, Figure 51, and Figure 52 we notice that the micro-viruses provide the same or higher  $V_{min}$  than the benchmarks for 19 of the 24 cores (3 chips x 8 cores). There are a few cases that benchmarks have higher  $V_{min}$  in 5 cores (the difference between them is at most 5mV – 0.5%) but in orders of magnitude shorter time.

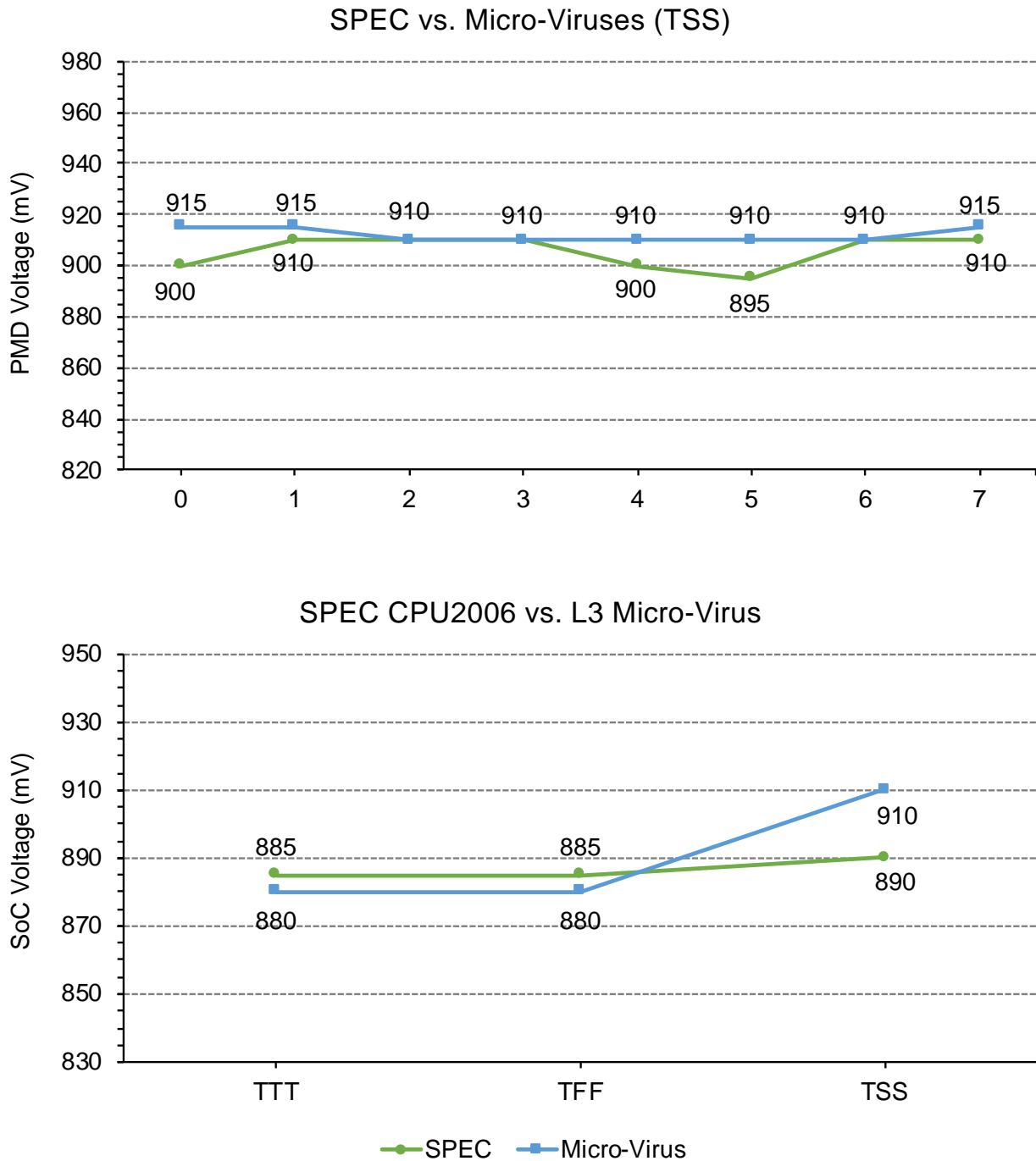


**Figure 50: Detailed comparison of  $V_{min}$  between the 12 SPEC CPU2006 benchmarks and micro-viruses for the TSS chip.**



**Figure 51: Maximum  $V_{min}$  among 12 SPEC CPU2006 benchmarks and the proposed micro-viruses for TTT and TFF in PMD domain.**

Such differences (5mV or even higher) can occur even among consecutive runs of the same program, in the same voltage due to the non-deterministic behavior of the actual hardware chip. This is why we run the benchmarks 10 times and present only the maximum safest  $V_{min}$ . For a significant number of programs (benchmarks and micro-viruses), we can see variations among different cores and different chips. Figure 50 presents the detailed comparison of the safe  $V_{min}$  between the 12 SPEC CPU2006 benchmarks and the micro-viruses for the TSS chip, while Figure 51 and Figure 52 represent the maximum safe  $V_{min}$  for each core and chip among all the benchmarks (blue line) and all micro-viruses (orange line). Considering that the nominal voltage in PMD voltage domain (where these experiments are executed) is 980mV, we can observe that



**Figure 52: Maximum  $V_{min}$  among 12 SPEC CPU2006 benchmarks and the proposed micro-viruses for TSS in PMD domain (top graph). The bottom graph shows the maximum  $V_{min}$  of 12 SPEC CPU2006 benchmarks and the proposed L3 micro-virus in SoC domain.**

the  $V_{min}$  values of the micro-viruses are very close to the corresponding safe  $V_{min}$  provided by benchmarks, but in most cases higher.

The core-to-core and chip-to-chip relative variation among the three chips are also revealed with the micro-viruses. Both the SPEC CPU2006 benchmarks and the micro-viruses provide similar observations for core-to-core and chip-to-chip variation. For instance, in TTT and TFF chip, cores 4 and 5 are the most robust cores. This property holds in the majority of programs but can be revealed by the micro-viruses in several orders of magnitude shorter characterization time.

At the bottom-right diagram of Figure 51 we show the undervolting campaign in the SoC voltage domain (which is the focus of the L3 cache micro-virus). As shown in section 3.1, in X-Gene 2 there are 2 different voltage domains: the PMD and the SoC. The SoC voltage domain includes the L3 cache. Therefore, this graph presents the comparison of the L3 diagnostic micro-virus with the 12 SPEC CPU 2006 benchmarks that were executed simultaneously in all 8 cores (8 copies of the same benchmark) by reducing the voltage only in the SoC voltage domain. In this figure, we also notice that in TTT/TFF the difference of  $V_{min}$  between the benchmark with the maximum  $V_{min}$  and the self-test is only 5mV, while in TSS the micro-viruses reveal the  $V_{min}$  at 20mV higher than the benchmarks. Note that the nominal voltage for the SoC domain is 950mV (while in the PMD domain it is 980mV).

### 3.8.2 Observations

By using the proposed micro-viruses, we can detect very accurately (divergences have short range, at most 5mV; see Figure 51 and Figure 52) the safe voltage margins for each chip and core, instead of running time-consuming benchmarks. According to our experimental study, the micro-viruses reveal higher  $V_{min}$  (meaning lower voltage margin) in the majority of cores in the three chips we used. Specifically, in 19 out of 24 cores in total, the micro-viruses expose higher or the same safe  $V_{min}$  compared to the SPEC CPU2006 benchmarks. For the specific ARMv8 design, we point and discuss the core-to-core and chip-to-chip variation, which are important to reduce the power consumption of the microprocessor.

**Core-to-Core Variation:** There are significant divergences among the cores due to process variation. Process variation can affect transistor dimensions (length, width, oxide thickness etc.) which have direct impact on the threshold voltage of a MOS device, and thus, on the guardband of each core. We demonstrate that although micro-viruses can reveal similar divergences as the benchmarks among the different cores and chips, however, in most of the cases, micro-viruses expose lower divergences among cores in contrast to time consuming SPEC CPU2006 benchmarks. As shown in Figure 51 and Figure 52, our micro-viruses reveal higher safe  $V_{min}$  for all the cores than the benchmarks, and also, we notice that the workload-to-workload differences are up to 30mV. Therefore, due to the diversity of code execution of benchmarks, it is difficult to choose one benchmark that provides the highest  $V_{min}$ . Different benchmarks provide significantly different  $V_{min}$  at different cores in different chips. Therefore, a large number of different benchmarks are required to reach a safe result concerning the voltage margins variability identification. Using our micro-viruses, which fully stress the fundamental units of the microprocessor, the cores guardbands can be safely determined (regarding the safe  $V_{min}$ ) at a very short time, and guide energy efficiency when running typical applications.

**Chip-to-Chip Variation:** As Figure 51 and Figure 52 present for the TTT and TFF chips, PMD 2 (cores 4 and 5) is the most robust PMD for all three chips (it can tolerate up to 3.6% more undervolting compared to the most sensitive cores). We can notice that (on average among all cores of the same chip) the TFF chip has lower  $V_{min}$  points than the TTT chip, in contrast to the TSS chip, which has higher  $V_{min}$  points than the other two chips, and thus, can deliver smaller power savings.

**Diagnosis:** By using the diagnostic micro-viruses we can also determine if and where an error or a silent data corruption (SDC) occurred. Through this component-focused stress process we have observed the following:

- a. SDCs occur when the pipeline gets stressed (ALU, FPU and Pipeline tests), and
- b. the cache bit-cells operate safely at higher voltages (the caches tests crash lower than the ALU and FPU tests).

Both observations show that the X-Gene 2 is more susceptible to timing-path failures than to SRAM array failures. A major finding of our analysis using the micro-viruses for ARMv8-compliant multicore CPUs is that SDCs (derived from pipeline stressing using the ALU, FPU and Pipeline micro-viruses) appear at higher voltage levels than corrected errors when cache arrays get stressed by cache-related micro-viruses. We believe that the reason is that unlike other server-based CPUs (like Itanium), X-Gene 2 does not deploy circuit-level techniques (Itanium performs continuous clock-path de-skewing during dynamic operation [142]), and thereby, when the pipeline gets stressed, X-Gene 2 produces SDCs due to timing-path failures.

### 3.9 Related Work

During the last years, the goal for improving microprocessors' energy efficiency, while reducing their power supply voltage is a major concern of many scientific studies that investigate the chips' operation limits in nominal and off-nominal conditions. In this section, we briefly summarize the existing studies and findings concerning low-voltage operation and characterization studies.

Whilkerson *et al.* [146] go through the physical effects of low-voltage supply on SRAM cells and the types of failures that may occur. After describing how each cell has a minimum operating voltage, they demonstrate how typical error protection solutions start failing far earlier than a low-voltage target (set to 500mV) and propose two architectural schemes for cache memories that allow operation below 500 mV. The word-disable and bit-fix schemes sacrifice cache capacity to tolerate the high failure rates of low voltage operation. While both schemes use the entire cache on high voltage, they sacrifice 50% and 25% accordingly in 500 mV. Compared to existing techniques, the two schemes allow a 40% voltage reduction with power savings of 85%.

Chishti *et al.* [147] propose an adaptive technique to increase reliability of cache memories, allowing high tolerance on multi-bit failures that appear on low-voltage operation. The technique sacrifices memory capacity to increase the error-correction capabilities, but unlike previously proposed techniques, it also offers soft and non-persistent error tolerance. Additionally, it does not require self-testing to identify erratic cells in order to isolate them. The MS-ECC design can achieve a 30% supply voltage reduction with 71% power savings and allows configurable ECC capacity by the operating system based on the desired reliability level.

Bacha *et al.* [7] present a new mechanism for dynamic reduction of voltage margins without reducing the operating frequency. The proposed mechanism does not require additional hardware as it uses existing error correction mechanisms on the chip. By reading their error correction reports, it manages to reduce the operating voltage while keeping the system in safe operation conditions. It covers both core-to-core and dynamic variability cause by the running workload. The proposed solution was prototyped on an Intel Itanium 9560 processor and was tested using SPECjbb2005 and SPEC CPU2000-based workloads. The results report promising power savings that range between 18% to 23%, with marginal performance overheads.

Bacha *et al.* [8] again rely on error correction mechanisms to reduce operating voltage. Based on the observation that low-voltage errors are deterministic, the paper proposes a hardware mechanism that continuously probes weak cache lines to fine-tune the system's supply voltage. Following an initial calibration test that reveals the weak lines, the mechanism generates simple write-read requests to trigger error-correction and is capable to adapt to voltage noise as well. The proposed mechanism was implemented as proof-of-concept using dedicated firmware that resembles the hardware operation on an Itanium-based server. The solution reports an average of 18% supply voltage



reduction and an average of 33% power consumption savings, using a mix set of applications.

Bacha *et al.* [158] exploit the observation of deterministic error distribution to provide physically unclonable functions (PUF) to support security applications. They use the error distribution of the lowest save voltage supply as an unclonable fingerprint, without the typical requirement of additional dedicated hardware for this purpose. The proposed PUF design offers a low-cost solution for existing processors. The design is reported to be highly tolerant to environmental noise (up to 142%) while maintaining very small misidentification rates (bellow 1ppm). The design was tested on a real system using Itanium processor as well as on simulations. While this study serves a different domain, it highlights the deterministic error behavior on SRAM cells.

Duwe *et al.* [148] propose an error-pattern transformation scheme that re-arranges erratic bit cells that correspond to uncorrectable error patterns (e.g., beyond the correctable capacity) to correctable error patterns. The proposed method is low-latency and allows the supply voltage to be scaled further that it was previously possible. The adaptive rearranging is guided using the fault patterns detected by self-test. The proposed methodology can reduce the power consumption up to 25.7%, based on simulated modeling that relies on literature SRAM failure probabilities.

There are several papers that explore methods to eliminate the effects of voltage noise, which however lean closer to the scope of T3.2 and thus, only a very brief reference is included.

Gupta *et al.* [149] and Reddi *et al.* [143] focus on the prediction of critical parts of benchmarks, in which large voltage noise glitches are likely to occur, leading to malfunctions. In the same context, several studies either in the hardware or in the software level were presented to mitigate the effects of voltage [5] [149] [192] [193] [194] or to recover from them after their occurrence [198]. Ketkar *et al.* [187] and Kim *et al.* [188] [189] propose methods to maximize voltage droops in single core and multicore chips in order to investigate their worst-case behavior due to the generated voltage noise effects. To conclude with, the characterization studies of commercial chips in off-nominal voltage conditions are limited to [6] [7] [8] [9] [84] [158].



## 4. Balancing Energy and Performance on Multicore ARMv8 CPUs

In the previous chapter, we discussed that in order to improve the microprocessor's efficiency (in terms of either power or performance), several hardware and software techniques have been proposed, such as Dynamic Voltage and Frequency Scaling (DVFS) [77] as well as several power capping approaches [169]. The ability to cap peak power consumption has recently gained strong interest in several important computing domains (e.g., mobile devices to data centers [170]) since the state-of-the-art high-end microprocessors currently support such features within their power management subsystem. Power capping is realized through power-performance knobs such as DVFS, pipeline throttling or memory throttling [169].

We also presented a comprehensive characterization study, which exposes the pessimistic voltage margins for single-core executions at the maximum frequency of the X-Gene 2 microprocessor. In this chapter, we are based on two recent state-of-the-art ARMv8-compliant multicore CPUs, Applied Micro's X-Gene 2 and X-Gene 3, to present a new software-based scheme for these server-grade machines, which provides large energy savings while maintaining high performance levels. The X-Gene 2 as well as the X-Gene 3 microprocessors are developed for HPC applications. X-Gene 3 specifically (which is the most recent microprocessor of the X-Gene family) has comparable performance to high-end Intel Xeon microprocessors [170]. However, neither X-Gene 2 nor X-Gene 3 microprocessors have predefined power management states as in x86 and IBM POWER architectures. Although X-Gene microprocessors support dynamic frequency scaling, the voltage is always the same for every different frequency step (which is equal to  $V_{nominal}$ ) and can be only explicitly modified.

Particularly, the main contributions of the work presented in this chapter are:

- We expose the pessimistic voltage guardbands of the two state-of-the-art ARMv8 microprocessors of the same family of products (manufactured in 28nm and 16nm – the X-Gene 2 and X-Gene 3, respectively) to identify the safe  $V_{min}$  points of the CPU chips in multicore executions. Note that in the previous chapter we presented an extensive characterization study for single-core executions (where we were aiming to identify the core-to-core variability among others). Having characterizing the multicore executions, in this chapter, we show that as the number of active threads increases, core-to-core and workload-to-workload variability have a minimal impact on  $V_{min}$ .
- We present measurements on the correlation of the safe  $V_{min}$  to the voltage droop magnitude, and show that in multicore executions the emergency voltage droop events occur regardless of the workload. However, for executions in a single or very few cores, core-to-core and workload-to-workload variability exist (as we also presented in the previous chapter).
- We perform an extensive study to identify and analyze the tradeoffs between energy and performance at different voltage and frequency combinations, as well as at different thread scaling and core allocation configurations. Our analysis reveals that depending on the coarse-grain characteristics of a program and the number of active threads, there is an optimal combination of voltage, frequency and core allocation for better energy efficiency.
- We also developed a simple online monitoring daemon which monitors the running processes on the system and guides the Linux scheduler to take the appropriate decisions regarding: (a) the core(s) to which a new process should be assigned, and (b) when one or more running processes should be migrated to other cores.

At the same time, the daemon dynamically adjusts the V/F settings according to the optimal policies.

- Finally, we evaluate the optimal energy efficient scheme by running the monitoring daemon in a realistic scenario of a server's operation, which (a) randomly selects the issued programs, (b) dynamically migrates the running processes on the system, and (c) dynamically adjusts the voltage and frequency settings. We report several comparisons among different configurations to present a detailed evaluation of the optimal scheme, and show that it can achieve on average 25.2% energy savings on X-Gene 2, and 22.3% energy savings on X-Gene 3, with a minimal performance penalty of 3.2% on X-Gene 2 and 2.5% on X-Gene 3 compared to the default voltage and frequency microprocessor's conditions.

## 4.1 Experimental Setup

### 4.1.1 Platforms

This study is performed on two different state-of-the-art ARMv8 micro-processors: Applied Micro's (now Ampere Computing) X-Gene 2 and X-Gene 3, which consist of 8 and 32 64-bit ARMv8-compliant cores, respectively. Both microprocessors offer high-end processing performance and come along with a subsystem that features a Scalable Lightweight Intelligent Management processor (SLIMpro) to enable flexibility in power management, resiliency and end-to-end security for a wide range of applications. The dedicated SLIMpro processor monitors system sensors, configures system attributes (e.g., regulates supply voltage, etc.) and can be accessed by the system's running Linux kernel.

In section 3.1 we present the main characteristics of X-Gene 2 microprocessor. The X-Gene 3 microprocessor is a more powerful, larger scale machine compare to X-Gene 2. Specifically, X-Gene 3 microprocessor has a main power domain that includes the CPU cores, the L1, L2 and L3 cache memories, and the memory controllers, which is called PCP (Processor ComPlex) power domain, as shown in Figure 53, and is the one that consumes the largest part of the overall power consumption. Figure 53 presents the architecture of X-Gene 3, however, the X-Gene 2 has similar structure; the difference is that it has 8 cores instead of 32, and the L3 cache, which is 8MB instead of 32MB, is located in a different domain (see Figure 28 in section 3.1).

The operating voltage of the main power domain can change from 980 mV downwards in X-Gene 2 and from 870 mV downwards in X-Gene 3 by at least 5mV steps. While all the CPU cores operate at the same voltage, each pair of cores (PMD – Processor MoDule) can operate at different frequency in both chips. The frequency can range from 300MHz up to 2.4GHz in X-Gene 2, and from 375MHz to 3GHz in X-Gene 3 (at 1/8 steps of the maximum clock frequency in both microprocessors). Table 17 presents the main characteristics of the two microprocessors. Both platforms run CentOS 7.3 with Linux kernel version 4.11.

### 4.1.2 Experimental Configuration

In this analysis, we use 25 benchmarks from 3 different benchmark suites as shown in Table 18: the NAS Parallel Benchmark Suite v3.3.1 (NPB) [171], the SPEC CPU2006 suite [141], and the PARSEC v3.0 suite [172]. NPB are programs designed to evaluate the performance of parallel supercomputers and have been used in several studies regarding performance and energy efficiency [173] [174]. Given that this study is primarily based on the multicore execution, we use the NPB parallel benchmarks and the PARSEC parallel benchmarks for multi-thread executions, and the SPEC CPU2006

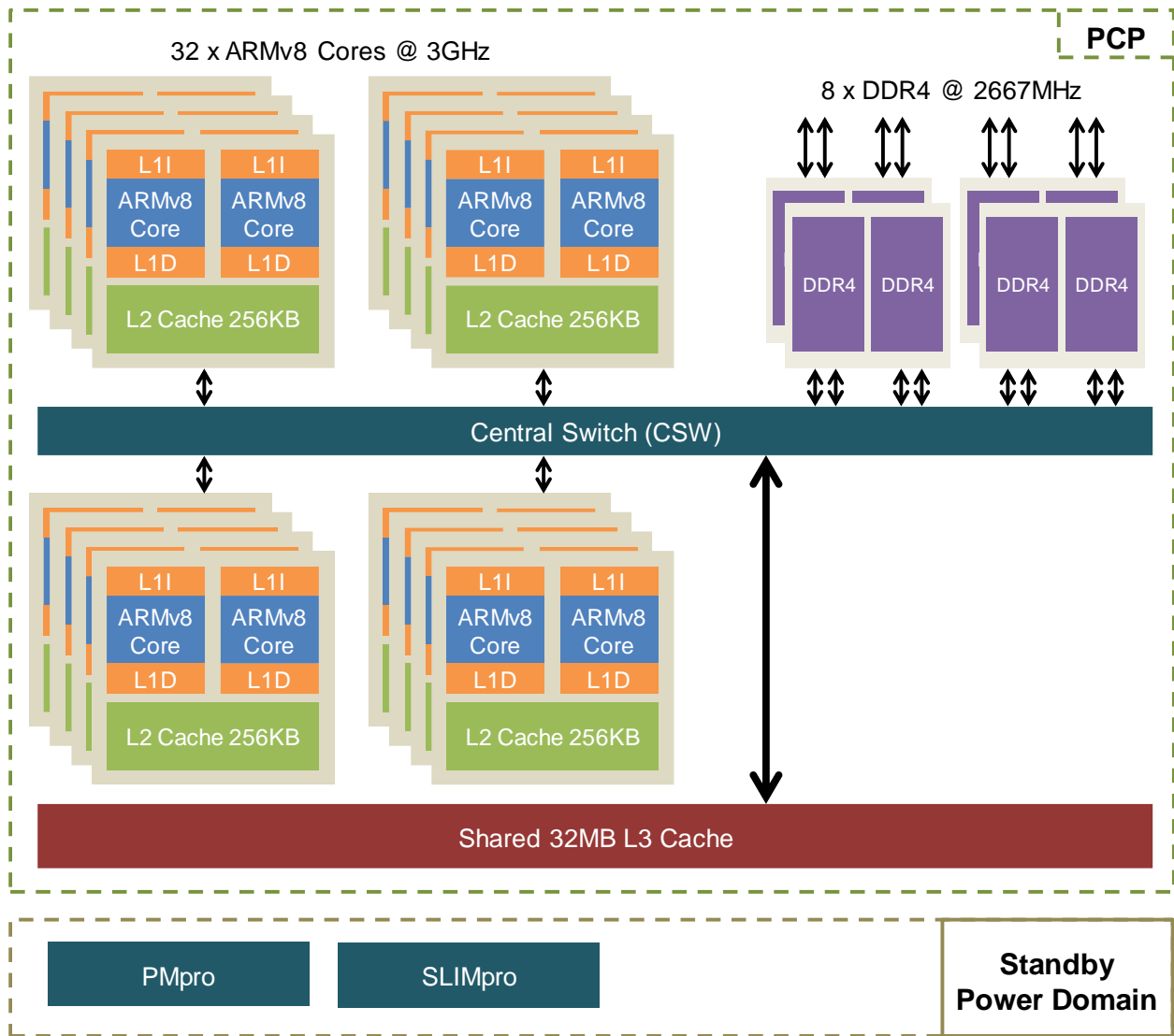


Figure 53: X-Gene 3 block diagram.

Table 17: Basic Parameters of X-Gene 2 and X-Gene 3.

Parameter	X-Gene 2	X-Gene 3
ISA	64-bit OoO (4-issue)	
Pipeline	ARMv8 (AArch64, AArch32, Thumb)	
CPU	8 cores	32 cores
Core Clock	2.4 GHz	3.0 GHz
L1 Instruction Cache	32 KB per core (Parity Protected)	
L1 Data Cache	32 KB per core (Parity Protected)	
L2 Cache	256 KB per PMD (SECDED Protected)	
L3 Cache	8 MB (SECDED Protected)	32 MB (SECDED Protected)
Technology	28 nm (bulk CMOS)	16 nm (FinFET)
Max TDP	35 W	125 W
Nominal Voltage	980 mV	870 mV

**Table 18: Benchmarks description.**

Name	Description	Suite
<b>CG</b>	Conjugate Gradient	NAS Parallel Benchmarks v3.3.1 [171]
<b>EP</b>	Embarrassingly Parallel	
<b>FT</b>	Discrete 3D FFT	
<b>IS</b>	Integer Sort	
<b>LU</b>	Lower-Upper Gauss-Seidel solver	
<b>MG</b>	Multi-Grid on a sequence of meshes	
<b>namd</b>	Scientific, Structural Biology	SPEC CPU2006 [141]
<b>cactusADM</b>	Physics / General Relativity	
<b>leslie3d</b>	Computational Fluid Dynamics	
<b>deall</b>	Solution of Partial Differential Equations	
<b>bwaves</b>	Computational Fluid Dynamics	
<b>gromacs</b>	Chemistry / Molecular Dynamics	
<b>zeusmp</b>	Physics / Magneto-hydrodynamics	
<b>milc</b>	Physics / Quantum Chromodynamics	
<b>mcf</b>	Combinational Optimization	
<b>swaptions</b>	Uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions	PARSEC v3.0 [172]
<b>blackscholes</b>	Black-Scholes partial differential equation	
<b>fluidanimate</b>	Simulates an incompressible fluid for interactive animation purposes	
<b>canneal</b>	Simulated annealing (SA) to minimize the routing cost of a chip design	
<b>bodytrack</b>	Tracks a human body with multiple cameras	
<b>dedup</b>	Compresses a data stream with a combination of global and local compression	

single-thread benchmarks (both FP and INT class), for evaluating multiple copies of single-threaded executions.

Note that, in a parallel execution with  $N$  threads (assume an NPB or PARSEC program with  $N$  parallel threads), all active threads compute parts of the same work, which is executed once. On the other hand, when we execute  $N$  copies of the same single-threaded benchmark, the microprocessor executes  $N$  times the same work. Therefore, we cannot directly compare the two cases as they refer to different amount of work and thus, the energy values of the single-threaded benchmarks are normalized to the number of running instances in order to deliver a fair comparison between the two groups of programs. For example, if the microprocessor executes  $N$  instances of a single-threaded benchmark for the SPEC CPU2006 suite (e.g., *namd*), the energy will be equal to  $energy\_of\_N\_instances / N$ .

For a comprehensive coverage of as many different execution behaviors as possible, we executed all 25 benchmarks (Table 18) in 3 different threading configurations in both X-Gene 2 and X-Gene 3 systems:

- **Max threads:** In all available cores of each microprocessor (8 cores for X-Gene 2 and 32 cores for X-Gene 3)
- **Half threads:** In half of the cores (4 cores for X-Gene 2 and 16 cores for X-Gene 3), and
- **Quarter threads:** In one quarter of the cores (2 cores for X-Gene 2 and 8 cores for X-Gene 3).

For these 3 different thread-scaling options, we executed the programs at the maximum frequency of each microprocessor (2.4GHz for X-Gene 2 and 3GHz for X-Gene 3), and

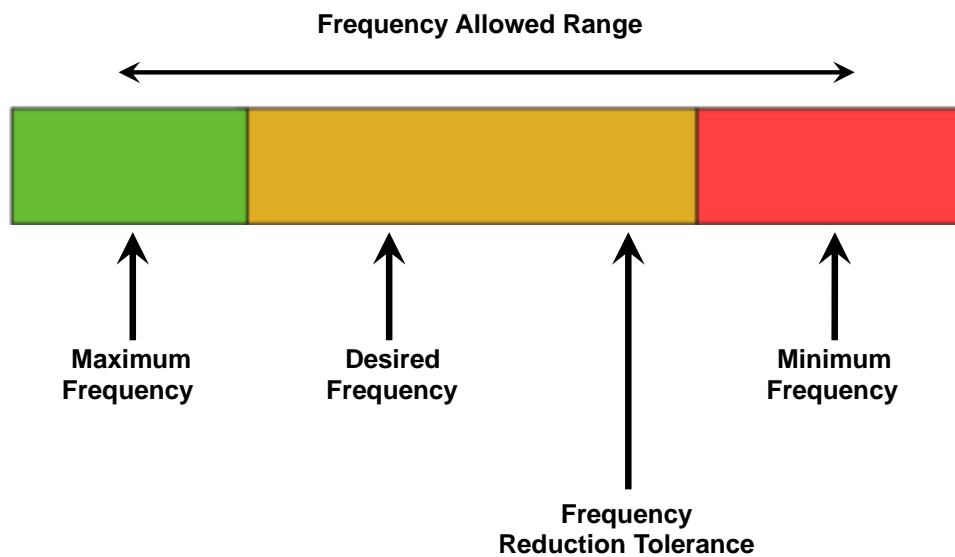
**Table 19: Frequency scaling in X-Gene microprocessors.**

Clock Ratio	Clock Skipping	Clock Division	X-Gene 2 Frequency	X-Gene 3 Frequency
8/8	Yes	No	2.4 GHz	3.0 GHz
7/8			2.1 GHz	2.625 GHz
6/8			1.8 GHz	2.250 GHz
5/8			1.5 GHz	1.875 GHz
<b>4/8</b>	<b>No</b>	<b>Yes</b>	<b>1.2 GHz</b>	<b>1.5 GHz</b>
3/8	Yes	Yes	0.9 GHz	1.125 GHz
2/8			0.6 GHz	0.750 GHz
1/8			0.3 GHz	0.375 GHz

at the half frequency (1.2GHz and 1.5GHz, respectively). It is essential to highlight, however, that clock frequencies larger than the half clock of both microprocessors have similar safe  $V_{min}$  as in the highest clock frequency, and frequencies smaller than the half clock have similar safe  $V_{min}$  as in the half clock. The reason is that both microprocessors support clock skipping and clock division, which, in combination, set the effective frequency of the PMD relative to its clock source, as shown in Table 19. Clock ratios greater or less than 1/2 on the input clock are implemented via clock skipping on the input clock. Clock ratio equal to 1/2 is naturally implemented via clock division on the input clock. For this reason, we do not present any results for the intermediate frequencies because they provide exactly the same  $V_{min}$  points.

Exceptionally, for X-Gene 2 only, we also present results at 0.9GHz, in which we noticed a significant reduction of the  $V_{min}$ , and thus, much larger energy savings compared to 1.2GHz, with minimal impact on performance. The reason is that these micro-servers implement the most recent CPPC (Collaborative Processor Performance Control) power and performance management specification of ACPI 5.1 [175], as shown in Figure 54.

CPPC is a new way to control the performance of cores using an abstract continuous scale in frequency, instead of a discretized P-state scale (as in legacy ACPI). Therefore, during runtime, when there is a request for 1.2GHz (desired frequency), in practice the

**Figure 54: Power and performance management in X-Gene microprocessors - the Collaborative Processor Performance Control.**

actual frequency of the microprocessor is scaled below and above of the 1.2GHz (frequency reduction tolerance), so that it effectively provides an average frequency of 1.2GHz. As a result, the actual frequency properties are limited by the highest frequency setting being used, which in this case is above half (without clock division). This is a frequency interleaving strategy which is provided by the CPPC and cannot be changed by software. Although X-Gene 3 operates also with CPPC specification, we did not observe the same behavior below the 1.5GHz as in X-Gene 2. This is an interesting finding of the characterization part of this work, and thus, we report experiments in X-Gene 2 for three different frequencies that represent all different behaviors: 2.4GHz, 1.2GHz and 0.9GHz and in X-Gene 3 we report our experiments at 3GHz and 1.5GHz.

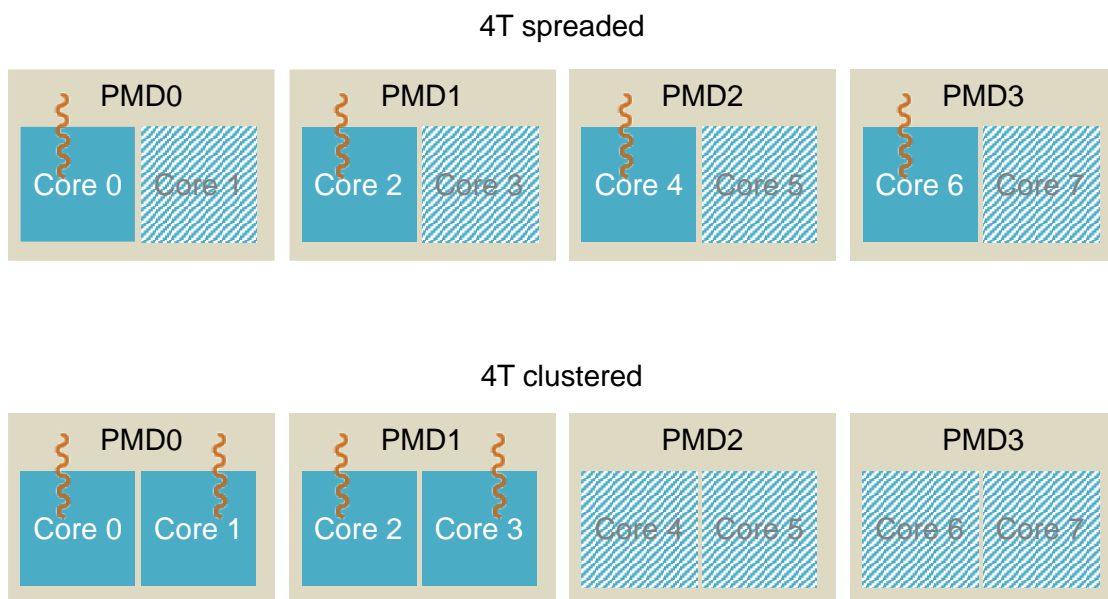
Furthermore, for the needs of our core-allocation analysis, when we execute multicore experiments with the number of threads being less than the available cores of each chip, we characterize different combinations of core allocation, and we have two main categories: *spreaded* threads and *clustered* threads. As shown in Figure 55, spreaded thread configuration refers to the threads running in separate PMDs each, while the clustered thread configuration to the threads running in consecutive cores (both cores of the PMD are occupied).

## 4.2 Voltage Margins Identification

This part focuses on a quantitative analysis of the safe  $V_{min}$  for two micro-servers of the same architecture in order to expose the potential guardbands of each chip, as well as to quantify the factors that determine the  $V_{min}$  of multicore executions.

### 4.2.1 Exposing Safe $V_{min}$ Values

We experimentally obtain the safe  $V_{min}$  values on the two different technology micro-servers: X-Gene 2 and X-Gene 3. For all of our experiments, we consider a voltage level as a safe  $V_{min}$  if the program passes it 1000 times. Safe  $V_{min}$  is the minimal working voltage. Note that we also study the error behavior for each program (subsection 4.2.2) operating below its safe  $V_{min}$  point, but we run it 60 times for each configuration (frequency, core allocation, and thread scaling) through the entire voltage range from the safe  $V_{min}$  until the system crash point.



**Figure 55: 4 running threads in 2 different core allocation configurations: spreaded & clustered.**



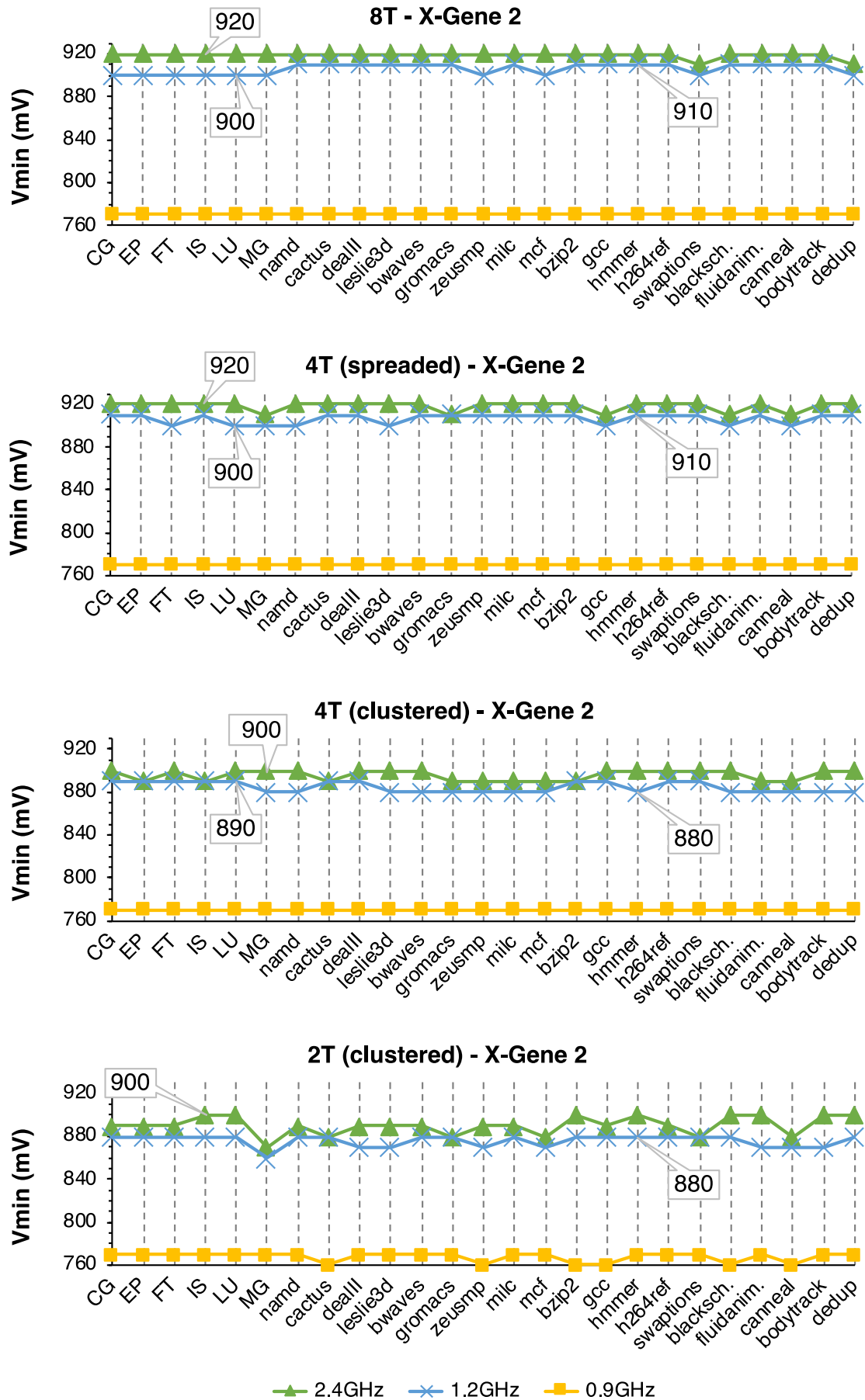
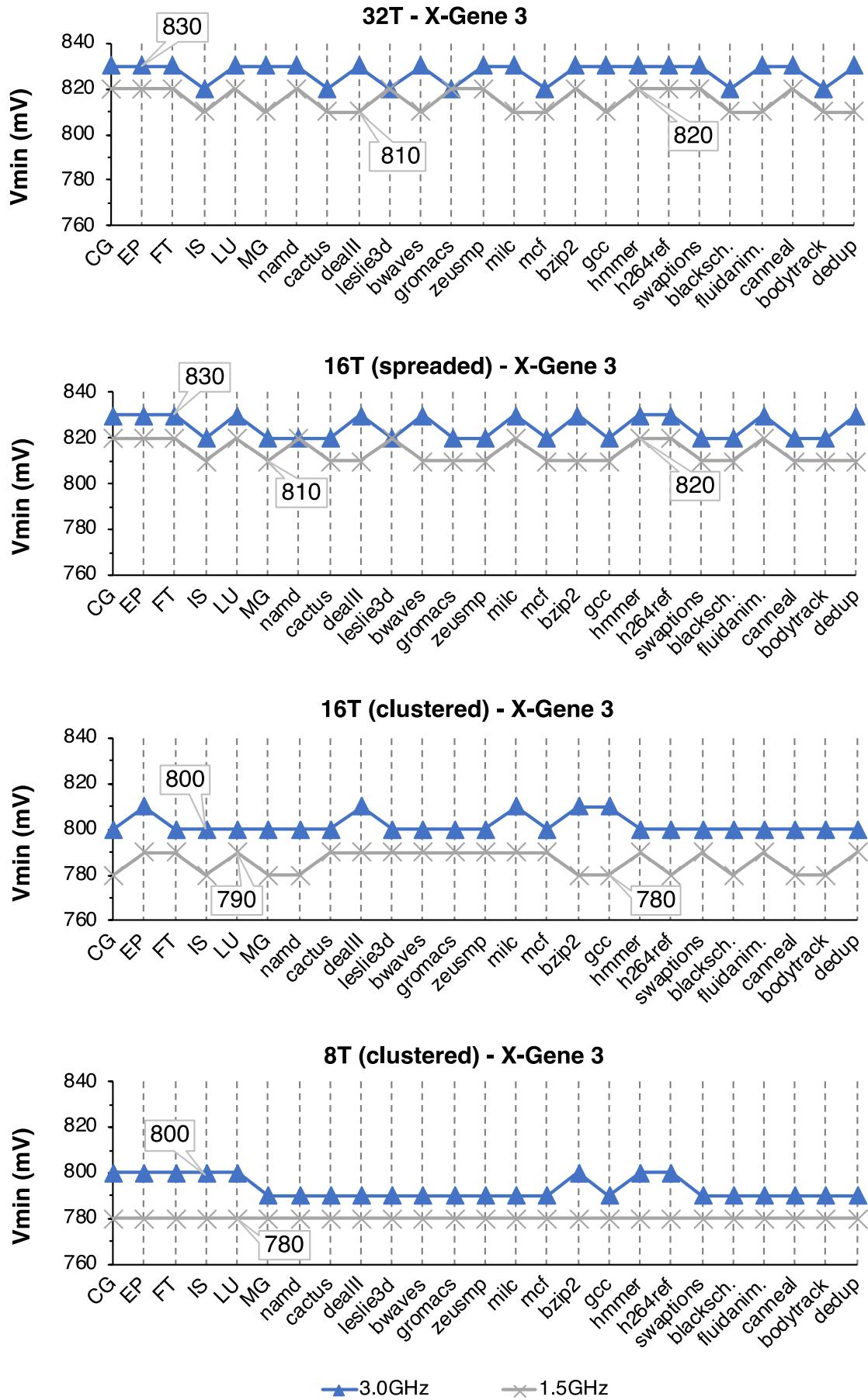


Figure 56: The complete  $V_{min}$  characterization results. This graph presents the X-Gene 2 safe  $V_{min}$  points for all benchmarks with 8 threads, 4 spreaded and clustered threads, and 2 clustered threads in 2.4 GHz, 1.2 GHz, and 0.9 GHz clock frequencies.



**Figure 57: The complete  $V_{min}$  characterization results. This graph presents the X-Gene 3 safe  $V_{min}$  points for all benchmarks with 32 threads, 16 spreaded and clustered threads, and 8 clustered threads in 3.0 GHz, and 1.5 GHz clock frequencies.**

Figure 56 and Figure 57 show the  $V_{min}$  characterization results for the 25 benchmarks on X-Gene 2 and X-Gene 3, respectively. The reported  $V_{min}$  for each program is the lowest (safe) voltage setting where all 1000 executions of each program completed successfully, without hardware errors notification, program output mismatches (silent data corruptions – SDCs) or other abnormal behavior, such as a process timeout, system crash or thread hang. Figure 56 presents the 8-thread, 4-thread and 2-thread executions of the benchmarks in X-Gene 2 for the three different frequencies: 2.4GHz, 1.2GHz and 0.9GHz. Figure 57 presents the  $V_{min}$  results for the same benchmarks on X-Gene 3 with 32, 16, and 8-thread executions for 3GHz and 1.5GHz. Figure 56 and Figure 57 clearly show, that for the same number of threads and at the same frequency, the safe  $V_{min}$  for all 25 benchmarks is virtually the same. There are some cases, where a benchmark has a little lower  $V_{min}$  than the rest, however, the maximum difference is only 10mV or ~1% of the nominal voltage.

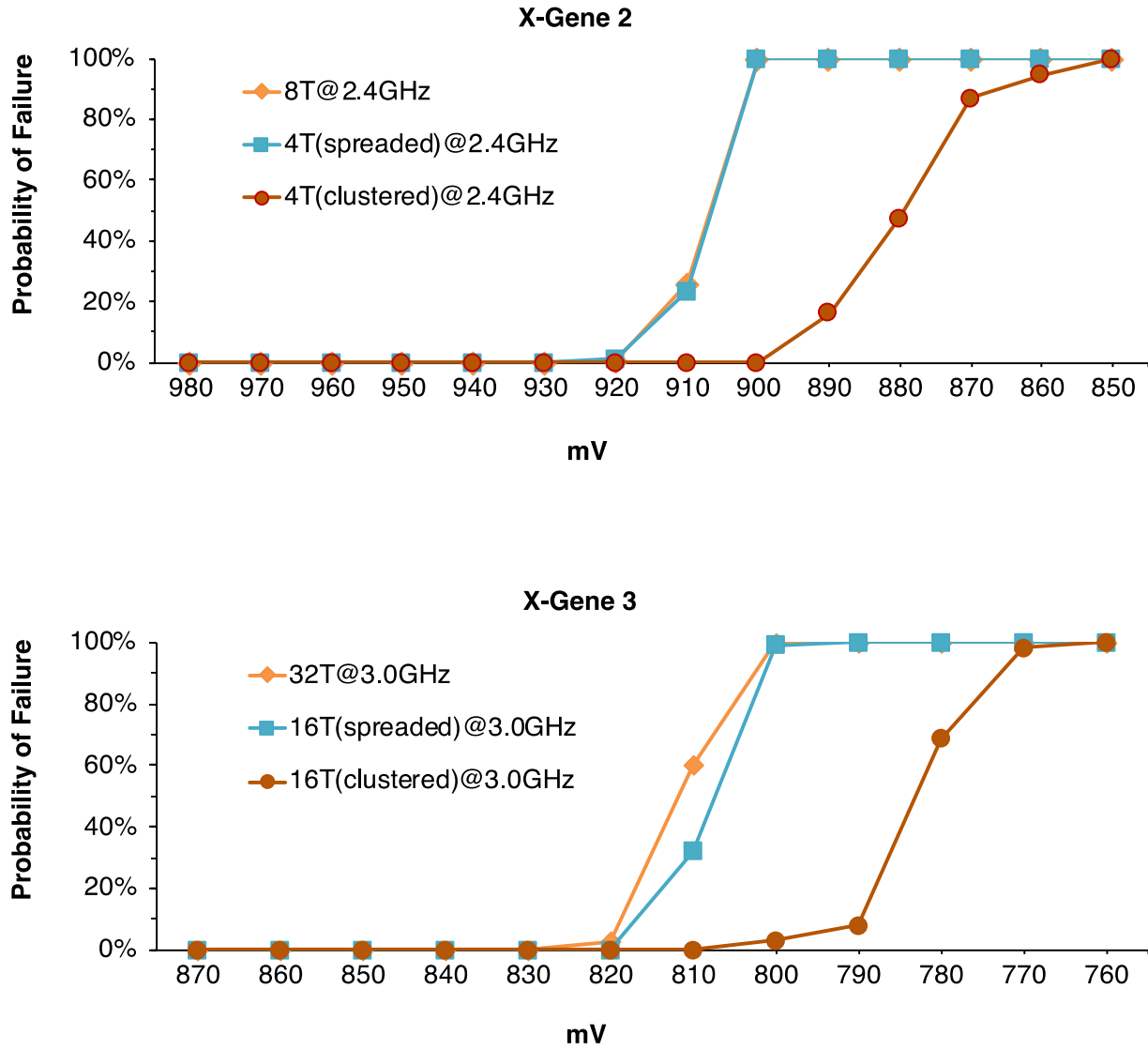
A major finding is that in multicore executions the safe  $V_{min}$  marginally depends on the workload (different program), but heavily depends on the number of active cores and the frequency, as shown in Figure 56 and Figure 57. We discuss the reasons for these differences in Section 4.3.

#### 4.2.2 Unsafe Region Investigation

As we described in the previous subsection, in multicore executions the safe  $V_{min}$  for every program is virtually the same for the same frequency and number of threads. Interestingly, we can notice such a behavior, across different benchmarks, frequencies and number of active threads, also in the region below the safe  $V_{min}$  (the unsafe region, where at least one run, faces an abnormal behavior). Figure 58 shows the cumulative probability of failure ( $pfail$ ) at each voltage level below the  $V_{min}$  (the  $V_{min}$  is the last voltage step with  $pfail=0$ ). Each line of the graph corresponds to the average  $pfail$  of the 25 benchmarks, in two different core allocation and thread scaling options. Similar to Figure 56 and Figure 57, for the same configuration (number of threads and frequency) the workloads have very small differences on the safe  $V_{min}$ , however, for different core allocation options we observe different behavior in  $V_{min}$ . The same pattern appears also in the unsafe region.

Consider for example the configurations with “max threads” and with spreaded “half threads” at maximum frequency (8T @ 2.4GHz and 4T (spreaded) @ 2.4GHz in X-Gene 2, and 32T @ 3GHz and 16T (spreaded) @ 3GHz in X-Gene 3). As shown in Figure 58, these two lines of the graph are virtually identical and have the most severe behavior ( $pfail=100\%$  means that all identical executions failed to complete. On the other hand, a  $pfail=10\%$  means that there are 90% chances for an application to execute correctly in that voltage). On the contrary, if we change the core allocation of “half threads”, we notice that the behavior changes significantly. Consider now, the “clustered” core allocation option (8T @ 2.4GHz and 4T (clustered) @ 2.4GHz in X-Gene 2, and 32T @ 3GHz and 16T (clustered) @ 3GHz in X-Gene 3). Although the frequencies between “max threads” and “half threads” are the same, the  $pfail$  (and also the safe  $V_{min}$ ) are very different (“half threads” configuration has lower safe  $V_{min}$  and  $pfail$  than “max threads”) because the core allocation was changed. We demonstrate in Section 4.3 how this observation is correlated to the voltage droop magnitude.

Based on this experimentation, we conclude that the dominant factors which can affect the safe  $V_{min}$  in multicore executions and also the failure probability, are (a) the frequency, and (b) the core allocation. The workload itself has only a marginal impact on the  $V_{min}$  in multicore execution for the same number of threads in different frequencies (see Figure 56 and Figure 57), however, we can see that lower frequencies have significantly lower safe  $V_{min}$  (and  $pfail$  respectively) for all the benchmarks than at the maximum frequency. For the same number of threads, we also notice that reducing the frequency to the “half



**Figure 58: Probability of Failure (pfail) in all voltage levels from nominal level down to the levels of complete failure for different frequency, core allocation, and thread scaling options.**

speed” we can further decrease the operating voltage by approximately 3%, while the 0.9GHz runs show a significant reduction (approximately 15%) at the  $V_{min}$  due to the clock division that is activated at that frequency (see explanation in subsection 4.1.2). Moreover, using a different core allocation strategy (*clustered* or *spreaded*) for the same number of threads, we can achieve further voltage reduction by 4%.

### 4.3 Analysis of $V_{min}$ Impact Factors

In this section we study the correlation of the contributors in  $V_{min}$  presented before, and identify the best combination of workload characteristics, frequency and core allocation towards the highest energy efficiency. We also discuss how these findings can be exploited in runtime.

#### 4.3.1 Impact of Frequency and Core Allocation on Safe $V_{min}$

Apart from the reduced frequency, in which the operating voltage can be decreased, the second major factor that can reduce the safe  $V_{min}$  is the core allocation. Several studies have exposed the safe  $V_{min}$  for each individual core of the microprocessor, by exploiting the static variation and the workload-to-workload variation [7] [8] [11] [157] [158] [177]. These studies demonstrate that different workloads have different safe  $V_{min}$ , and thus,

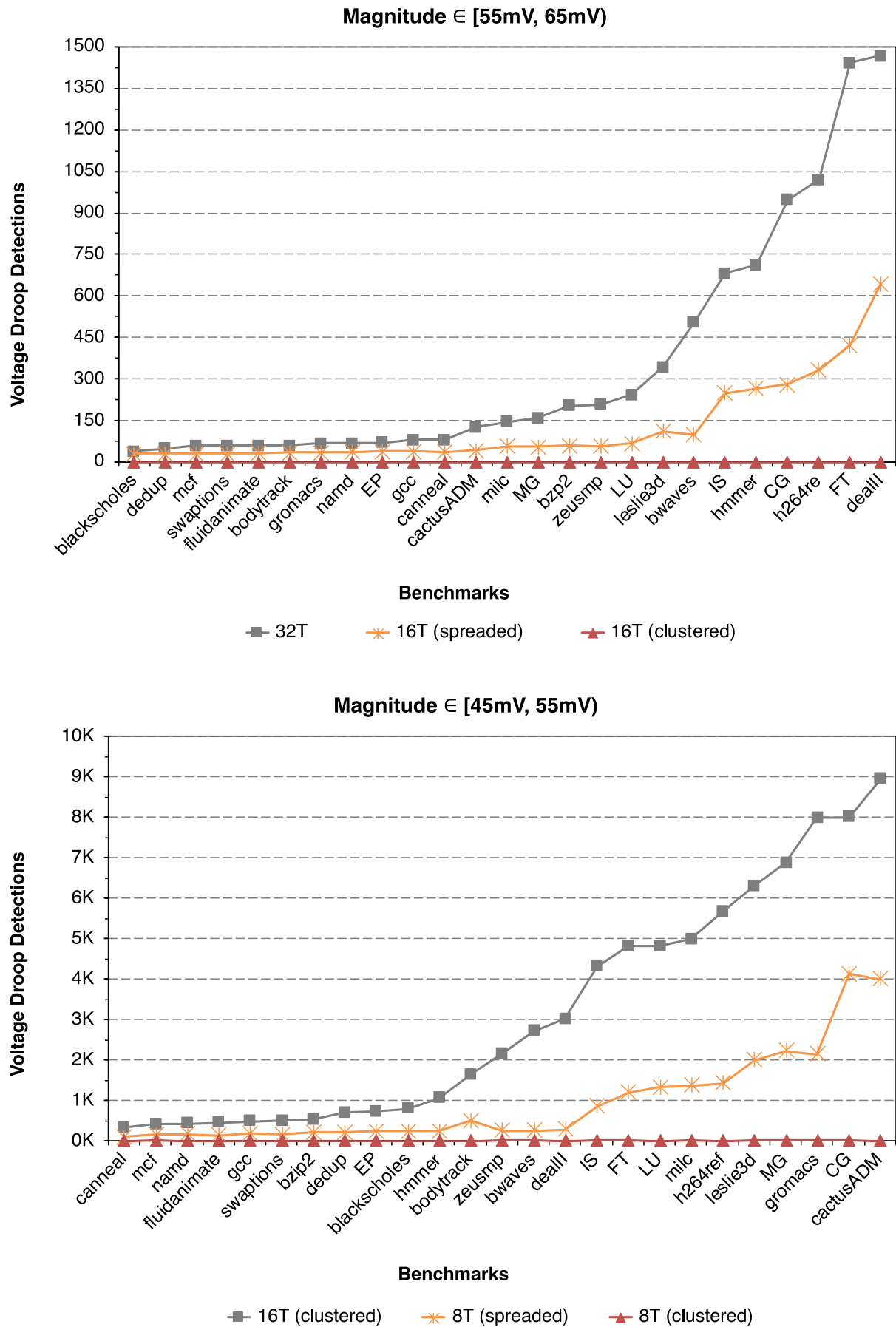
there is an intense research to provide several methods which will be able to predict the safe  $V_{min}$  for each program, dynamically, when the microprocessor operates normally. However, we show that the  $V_{min}$  variation among different workloads (multi-threaded or multiple copies of a single-threaded application) fades away as the number of running threads increases, and the remaining key factors that determine the  $V_{min}$  are the frequency and core allocation. It is known that the larger the number of threads running in the microprocessor, the more noise is generated from voltage droops and the interference of running processes in the system. The findings of our analysis quantify the magnitude of this dependence.

To understand this phenomenon, we study the voltage droop magnitude of the microprocessors for all the different frequency and core allocation configurations, by leveraging the embedded oscilloscope in the X-Gene 3 microprocessor. The X-Gene 3 microprocessor consists of a Voltage Droop Mitigation logic (VDM), which is responsible (if enabled) to detect and respond to occasional bad voltage droop events. The response upon detecting unusually large supply voltage droops is to temporarily reduce the clock frequency by using the same signals that are used for normal clock division and skipping. The goal in adding this feature is to be able to reduce the frequency/voltage guardband need to avoid failure. The VDM has two states: first, it detects the supply droop magnitude, and second, when the voltage droop exceeds a specified threshold, the VDM starts the mitigation sequence to avoid the failure. However, the mitigation sequence can be disabled. For this study, we have disabled the mitigation sequence, while the voltage droop detection remains enabled. With such a way, we are able to set different voltage thresholds in the VDM in order to count the voltage droop magnitude for each benchmark. PMU (Performance Monitoring Unit) counters, which can be accessed by the Perf tool [144], are located in the microprocessor and can be used to monitor the frequency and the magnitude of voltage droop events.

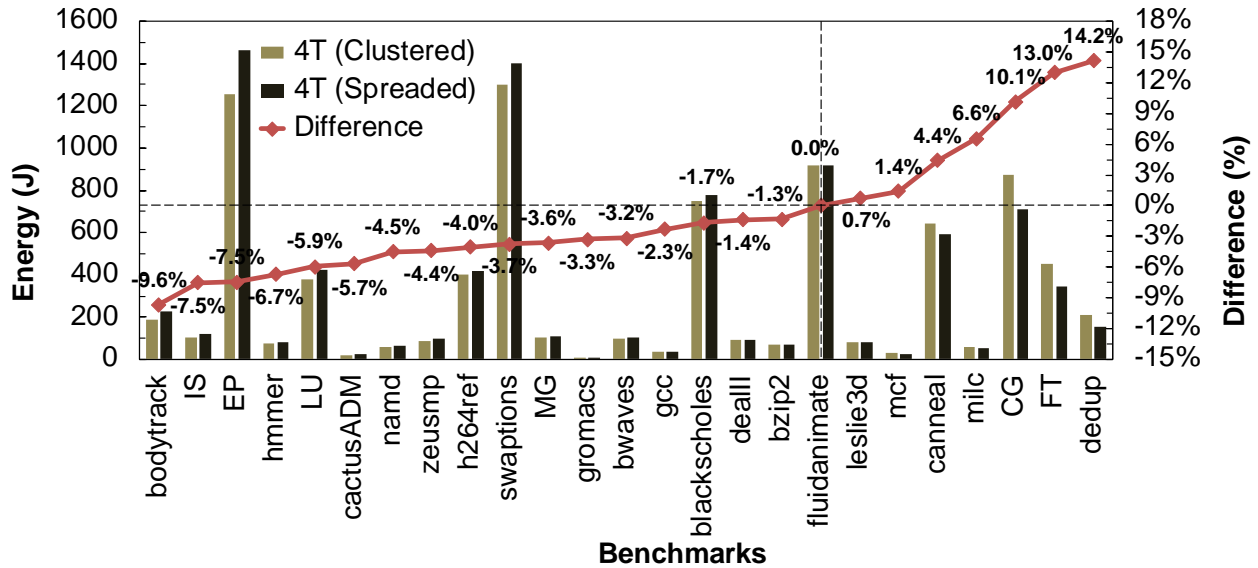
Figure 59 presents two different ranges of voltage droop magnitude when the microprocessor operates at 3GHz: (a) the [55mV, 65mV) in which we present the configurations of all programs that produce voltage droops more than or equal to 55mV and less than 65mV (this voltage range actually corresponds to threshold=5 of the VDM), and (b) the [45mV, 55mV) in which we present the configurations of all programs that produce voltage droops more than or equal to 45mV and less than 55mV (this voltage range actually corresponds to threshold=4 of the VDM). Both graphs show the total number of droop detections per 1M cycles for each program. As we can see in the left graph of Figure 59, the configurations with 32 threads and 16 spreaded threads, which means that all 16 PMDs of the microprocessor running at 3GHz frequency (note that the frequency can be changed per pair of cores – PMD) produce voltage droop magnitude between 55mV and 65mV. However, the configuration of 16 clustered threads (meaning 8 PMDs of the microprocessor running at 3GHz) has almost zero droops in the range of [55mV, 65mV) for all programs. On the other hand, in the right graph of Figure 59, the configurations with 16 clustered threads and 8 spreaded threads (8 PMDs operate at 3GHz frequency in both configurations) produce voltage droop magnitude in the range of [45mV, 55mV). However, the configuration with 8 clustered threads (4 PMDs) has almost zero droops in that range for any program.

#### 4.3.2 Impact of the Workload on Frequency and Core Allocation

Apart from the magnitude of the correlation of the safe  $V_{min}$  to frequency and core allocation, we also quantify the impact of the workload class (CPU-intensive vs. memory-intensive) to: (a) the safe  $V_{min}$ , (b) the workloads' performance, and (c) the energy consumption. Figure 60 shows an illustrative example of the difference in energy of all 25 programs when running at the maximum frequency, with the same number of threads (4



**Figure 59: Voltage droop detections for each program in 2 different voltage droop magnitudes. The top graph presents the droop detections in range between 55mV and 65mV, and the bottom graph presents the droop detections in range between 45mV and 55mV.**



**Figure 60: Energy of all benchmarks for 2 different core allocations. The benchmarks on the left of the dashed line are CPU-intensive while the ones on the right are memory-intensive.**

threads in this case) but in different core allocations (4T clustered vs. 4T spreaded) on the X-Gene 2 (the observation is similar in X-Gene 3). The red line with numbers at the top of each pair of bars shows the energy consumption difference between the two core allocations. We can see that the energy difference between these two configurations varies from -9.6% to 14.2%, depending on the characteristics of each workload. Negative percentages indicate that the spreaded-thread configuration needs higher energy than the clustered-thread configuration, while positive percentages indicate the opposite. In particular, the benchmarks shown at the right side of the dashed line have better energy when their threads are spreaded across the cores, unlike the benchmarks shown at the left side of the vertical dashed line which are more energy efficient when executed in consecutive cores (clustered configuration; see Figure 55). The reason is that the rightmost benchmarks are the most memory-intensive benchmarks, while the leftmost benchmarks are the most CPU-intensive.

As the previous sections present, frequency reduction and the optimal core allocation can enable significant opportunities of lowering supply voltage, however, reduced frequency also translates to degraded performance. Reduced frequency in CPU cores impacts their performance without affecting the lower memory levels (L3 cache and DRAM). This means that a program that is highly computational (CPU-intensive) will be proportionally affected by the reduced frequency. On the other hand, a program that experiences long stalls waiting for the memory to respond will be less affected from the core frequency reduction, as this will be hidden by the long memory delays (memory-intensive programs). The key difference between them is that in CPU-intensive programs the system part that acts as a performance limiter is the CPU core part (pipeline, L1 and L2 caches) while for the memory intensive programs, it is the memory part (L3 and DRAM).

In practice, we can use this property to combine lower frequency and lower voltage with small performance difference for memory intensive workloads, to increase their energy efficiency while still complying with high performance constraints. Previous studies (e.g., [178] and [179]), have exposed the phases of a program which are memory-intensive, and have proposed this feature as a proxy in a single-core microprocessor to guide the DVFS to reduce the frequency in that specific program phases. In this work we further extend this practice to show how this property can be also used to guide core allocation decisions and how this feature impacts the microprocessor's energy and program's performance. In order to identify the class of each program, we follow the method



proposed in [180], to track the access rates of the lower memory hierarchy, and more precisely L3 cache accesses. High L3 access rate means high memory activity in the lower memory hierarchy (memory-intensive program). To find the exact threshold level that separates the two classes, we initially identify what programs are memory intensive and then the L3 access rates of these benchmarks.

Figure 61 presents the performance impact of each program when we introduce contention on the shared CPU resources. We do that by executing multiple copies of the same program on all cores. Programs that are affected the most have high activity on the shared resources (and thus the contention negatively impacts the performance). As an example, we can see the *CG* and *FT*, which are the most memory-intensive benchmarks because their execution time is significantly reduced in a multi-threaded execution compared to the single-threaded one (ratio is much smaller than 1) due to the high contention at the memory system. On the other hand, the *namd* and *EP* are the most CPU-intensive benchmarks because their execution time in multi-threaded execution is virtually the same as in the single-threaded execution (ratio is very close to 1).

We then use the performance monitoring counters of the microprocessor as an indication of the workload class (CPU vs. memory intensive). In particular, we use the L3 Cache (L3C) memory access rate (by monitoring the L2 miss counters), which will allow us to identify the class of each program during runtime. Figure 62 shows the L3 Cache access rate for the 3 different threading configurations of 32, 16, and 8 threads (for this example we used the X-Gene 3 platform; the same behavior occurs in X-Gene 2 also), measured at 3GHz clock frequency. Based on the L3C access rate metric, and also on the experimental analysis of the safe  $V_{min}$ , we found that the threshold which defines the high memory activity is 3K accesses per  $10^6$  cycles. Executions above this threshold are the most memory-intensive, while those below the threshold are the most CPU-intensive.

#### 4.3.3 Summary of the Factors that Impact the $V_{min}$

Figure 63 quantifies the impact on the safe  $V_{min}$  that each factor has on the microprocessors of our analysis. The values shown in this figure are derived from our study in the X-Gene 2 microprocessor, however, the corresponding values for X-Gene 3 are similar. As we can see in Figure 63, the workload variability can affect at most 1% the

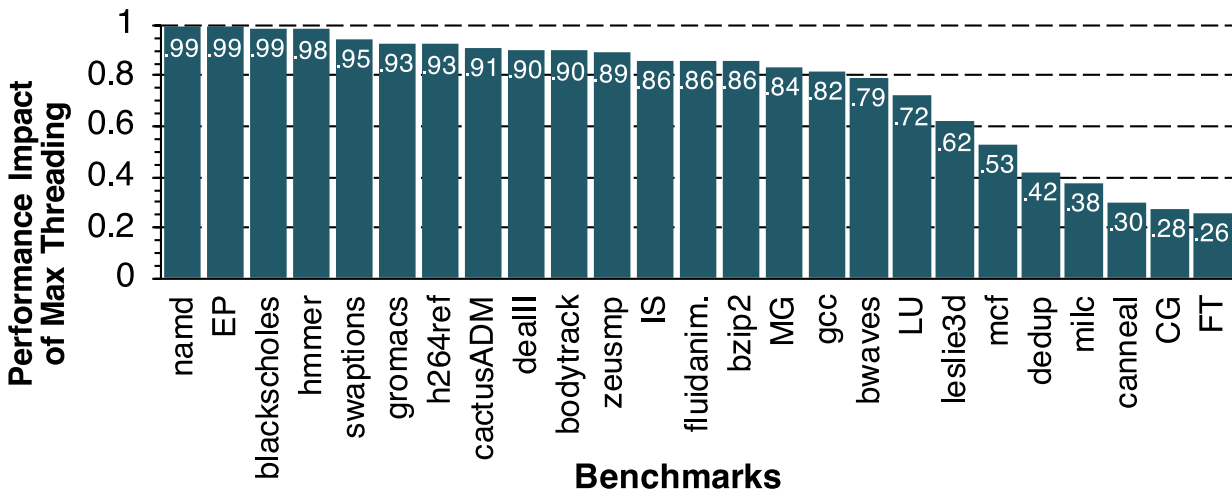


Figure 61: Relative performance of all benchmarks. The y-axis presents the ratio of the execution time of one instance of the single-threaded execution divided by the execution time of multiple instances.



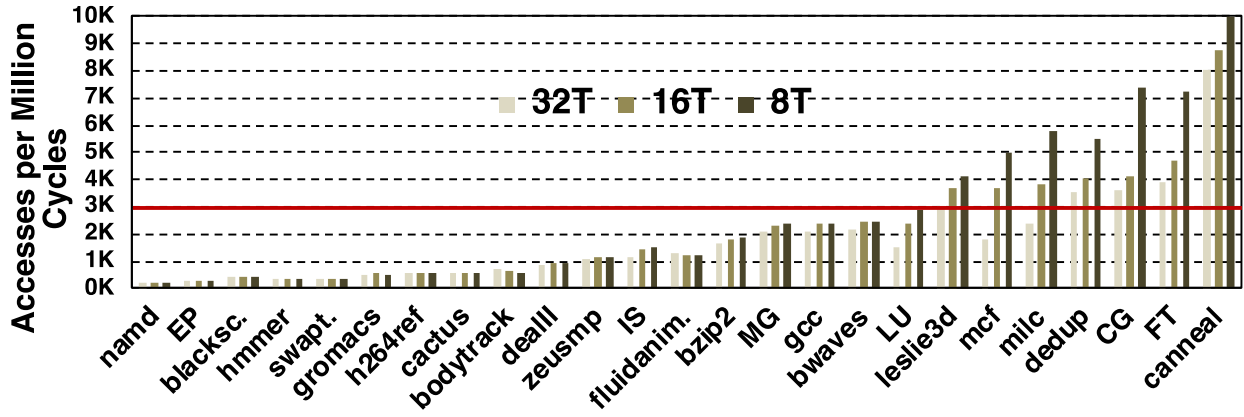


Figure 63: L3 Cache access rate per 1M cycles for the 25 benchmarks and the 3 threading configurations (32, 16 and 8 threads).

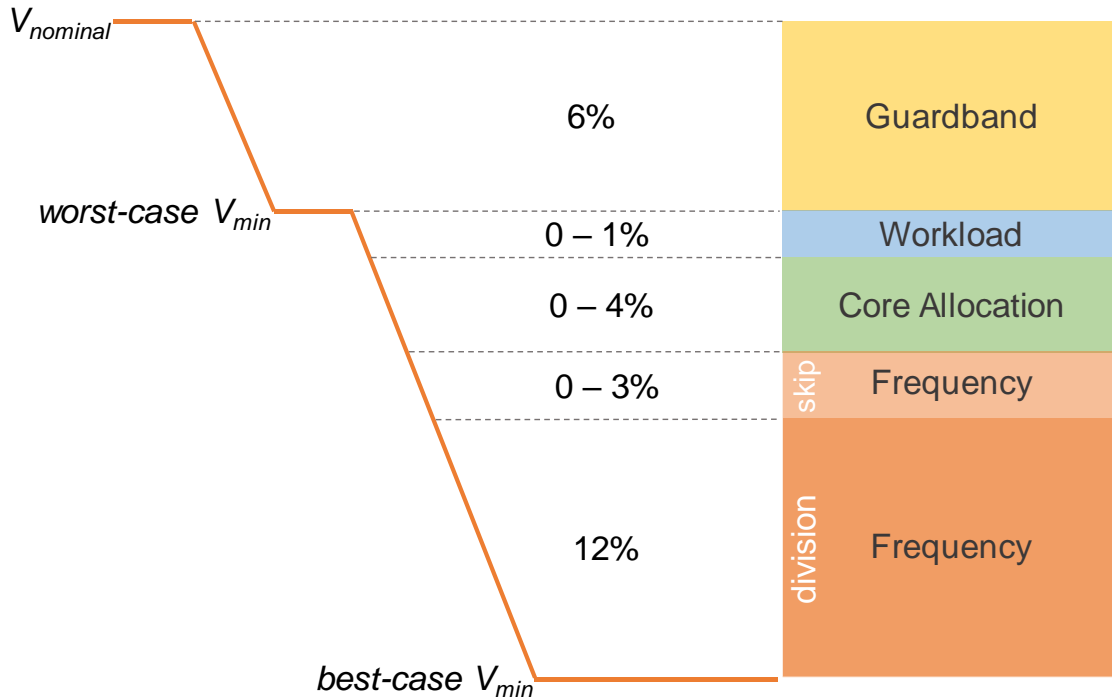


Figure 62: The magnitude of  $V_{min}$  dependence on frequency, core allocation and workloads.

safe  $V_{min}$ , while the core allocation and clock frequency are the major contributors to the safe  $V_{min}$ . The reason is that, as we demonstrated in subsection 4.3.1, frequency and core allocation are the main factors that can affect the voltage droop magnitude. In particular, the largest amount of voltage reduction (12%) is a result of clock division in a specific clock frequency, while just one step further frequency reduction (due to clock skipping) delivers 3% further voltage reduction. The following sections identify the best combinations concerning the voltage, frequency and application characteristics, which can lead to the highest energy savings in CPUs with many cores.

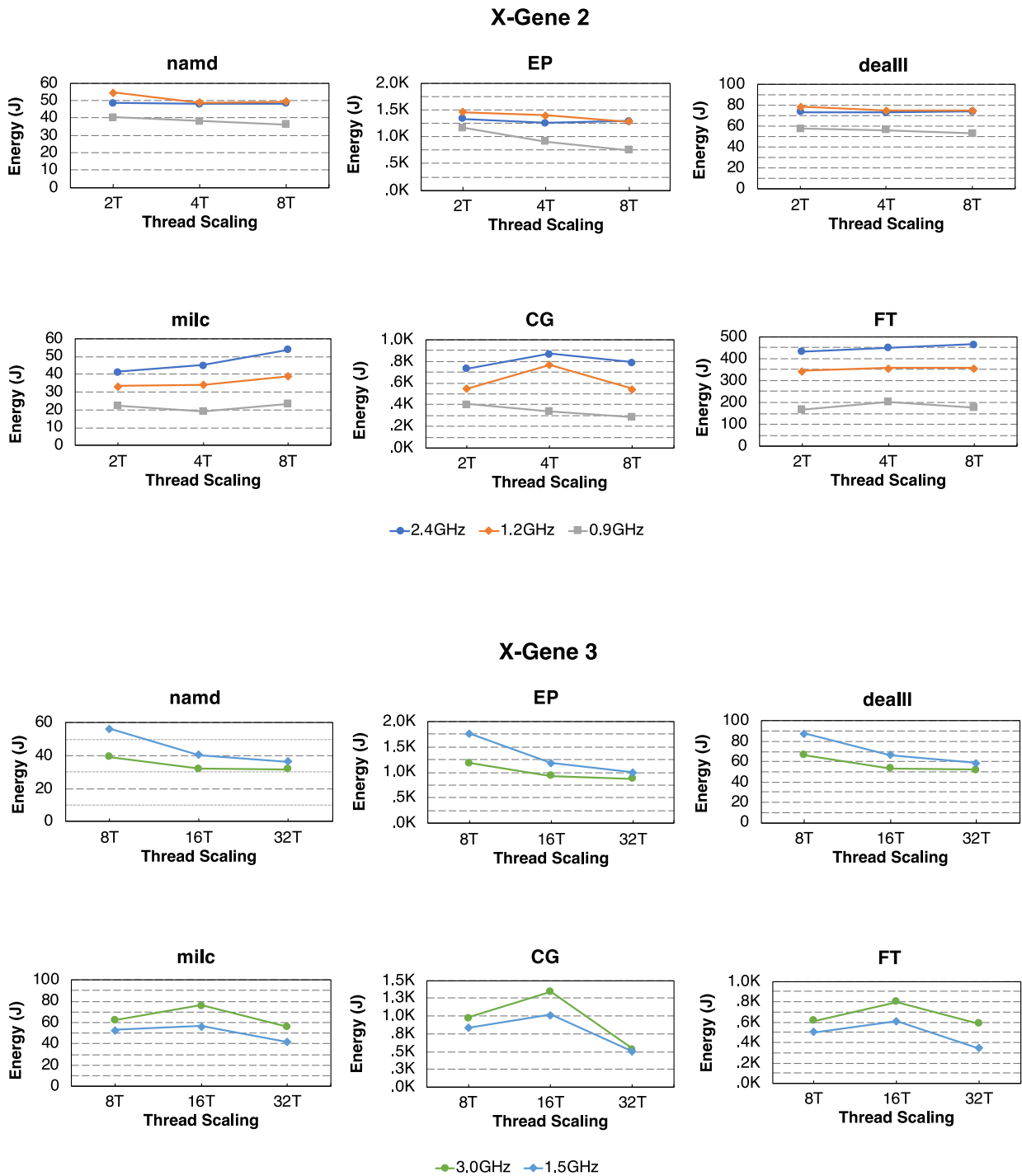
#### 4.4 Performance and Energy Trade-Offs

For a more comprehensive comparison of the different options and configurations we measure and present both the overall energy consumed for each particular workload

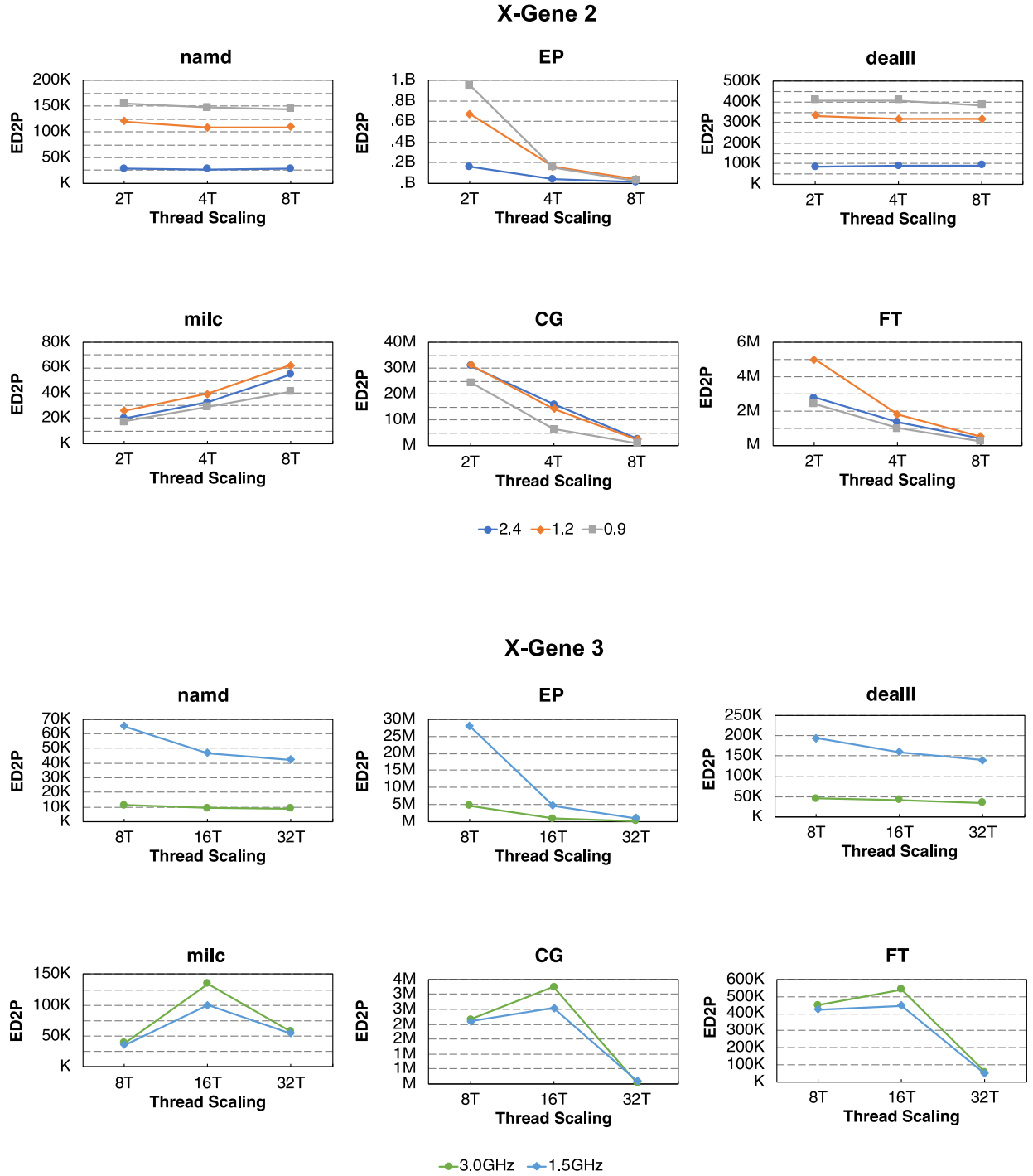
execution, as well as the energy delay squared product (ED2P) metric which combines the energy and the performance of each workload and configuration.

#### 4.4.1 Energy Efficiency

Figure 64 shows the energy consumption for 6 benchmarks (all other benchmarks follow the same pattern as the ones presented) and all configurations of X-Gene 2 and X-Gene 3 systems. The benchmarks are sorted (from left to the right) from the most CPU-intensive to the most memory-intensive ones, as it is also shown in Figure 61 (*namd*, *dealll*, and



**Figure 64: Energy consumption in Joules for 8, 4, and 2 threading options for 3 different frequencies (2.4GHz, 1.2GHz, 0.9GHz) of X-Gene 2 (top) and 32, 16, and 8 threading options for 2 different frequencies (3GHz, 1.5GHz) of X-Gene 3 (bottom).**



**Figure 65: Energy Delay Squared Product (ED2P) for 8, 4, and 2 threading options for 3 different frequencies (2.4GHz, 1.2GHz, 0.9GHz) of X-Gene 2 (top) and 32, 16, and 8 threading options for 2 different frequencies (3GHz, 1.5GHz) of X-Gene 3 (bottom).**

*EP* are the most CPU-intensive benchmarks, while *milc*, *CG*, and *FT* are the most memory-intensive ones).

X-Gene 2 reports significant energy savings for all cases when running at 0.9GHz, which is attributed to the significantly lower safe  $V_{min}$  voltage (as shown in Figure 56 and Figure 57, which is possible due to the clock division) and the lower frequency. For CPU-intensive benchmarks (*namd*, *deaIII*, and *EP*), the frequency reduction (from 2.4GHz to 1.2GHz) does not have an observable impact on the total energy consumption. We can only notice energy improvements for 1.2GHz on the memory-intensive applications (*milc*,

*CG*, and *FT*). Lower CPU speed reduces the performance gap between CPU and memory and leads to a more balanced system. This is why we can gain significant power savings without sacrificing too much performance, and thus achieve better energy efficiency.

Similar observations hold for X-Gene 3. The low-frequency behavior of X-Gene 3 (1.5GHz) matches the one of the 1.2GHz of X-Gene 2. For the CPU-intensive benchmarks the highest frequency (3GHz) gives the best energy (due to the faster execution), but again we can see that memory-intensive applications are more energy efficient in lower frequency. Note that, in X-Gene 3 we do not present results for lower frequencies than 1.5GHz, because as we have described in subsection 4.1.2, frequency settings below 1.5GHz have the same safe  $V_{min}$  as in 1.5GHz due to clock skipping, and thus, there is only performance impact.

#### 4.4.2 Combined Energy and Performance Considerations

Energy consumption itself is a valuable metric that directly translates to cost; however, it occasionally implies very slow system configurations (very low frequencies), which could violate latency and throughput requirements on a server environment. To avoid this bias in the comparisons of different configurations, other metrics such as the energy delay product ( $EDP = E \times D$ ) and the energy-delay squared product ( $ED2P = E \times D^2$ ) have been proposed for high-end systems [181]. Given that our work focuses on server-grade CPUs, we have chosen to present the energy delay squared product (ED2P) for all of our experiments to show a fair comparison among benchmarks, between different microprocessors, and more importantly to provide a fair representation of the relation between energy and performance. In this subsection, we focus on this metric for the comparison of the different X-Gene 2 and X-Gene 3 configurations.

Figure 65 shows the ED2P for 2.4GHz, 1.2GHz and 0.9GHz configurations for the X-Gene 2, and 3GHz and 1.5GHz configurations for the X-Gene 3. In the first 3 benchmarks (*namd*, *dealII*, and *EP*) in both X-Gene 2 and X-Gene 3, which are the most CPU-intensive benchmarks, we can see that the higher the frequency, the most efficient (in terms of ED2P) the configuration (i.e. the blue lines in X-Gene 2 and green lines in X-Gene 3 are always lower in all cases). However, we can see that the trend lines are totally different among the 3 memory-intensive benchmarks (*milc*, *CG*, and *FT*) and the CPU-intensive ones. We can see that the frequency is inversely proportional to ED2P efficiency for all the thread scaling options (grey lines vs. blue lines in X-Gene 2, and blue lines vs. green lines in X-Gene 3). Our analysis shows that the identification of the program class (CPU vs. memory-intensive) in runtime can guide the selection of the optimal system configuration (frequency and threading) to achieve high energy savings without compromising performance.

### 4.5 Mitigating Energy: A Real System Implementation

#### 4.5.1 Online Monitoring Daemon

According to our study and observations, we developed a simple online monitoring daemon which encapsulates all these conditions and constraints as we described in previous sections and guides process placement, core frequency and supply voltage to achieve higher energy savings on both X-Gene 2 and X-Gene 3 platforms. The daemon has two main functionalities: monitoring and placement.

The monitoring part of the daemon acts as a watchdog, which periodically monitors the utilized PMDs (which correspond to the droop magnitude shown in Table 20) and the L3C accesses of each running process (except for the system processes). For each process, it counts the L3C accesses during 1M cycles (this actually varies from 300ms to 500ms in our systems; it depends on the IPC rate of each process) and if the L3C accesses are

**Table 20: Correlation of voltage droops magnitude with frequency and core allocation ( $V_{min}$  columns concern X-Gen3).**

<b>Droop Magnitude</b>	<b>Utilized PMDs</b>	<b>Thread Scaling</b>	<b><math>V_{min}</math> @ 3GHz</b>	<b><math>V_{min}</math> @ 1.5GHz</b>
[ 25mV, 35mV )	1, 2 PMDs	1T, 2T, 4T (clustered)	780 mV	770 mV
[ 35mV, 45mV )	4 PMDs	8T (clustered), 4T (spreaded)	800 mV	780 mV
[ 45mV, 55mV )	8 PMDs	16T (clustered), 8T (spreaded)	810 mV	790 mV
[ 55mV, 65mV )	16 PMDs	32T, 16T (spreaded)	830 mV	820 mV

more than 3K (see Figure 62), then it classifies this process as memory-intensive, otherwise it classifies it as CPU-intensive. Moreover, it classifies the processes according to the utilized PMDs in order to estimate the current  $V_{min}$ .

The second part (Figure 66) of the daemon is the Placement. Figure 66 presents how the system reacts to a process list change and how the placement function of the daemon operates. Placement is the part that places the processes to the CPU cores (or migrate them) and adjusts the voltage and frequencies accordingly. As we presented earlier, each group of core allocation options corresponds to a specific droop magnitude class (Table 20) with a distinct safe  $V_{min}$  for each frequency. Moreover, as shown in Figure 59, for each core allocation option, all programs produce the same maximum droop magnitude. Given that, we do not use any sophisticated mechanism for predicting the safe  $V_{min}$  because the prediction schemes for  $V_{min}$  that have been proposed in the literature are error-prone (e.g., [9] [10] [11] [143] [177] [182] [183] [184] [185]) and can lead to system failures in real microprocessors.

To this end, the daemon has been equipped with a fail-safe mechanism as shown in Figure 67: either before the process(es) are invoked or before the frequency should be increased in one or more PMDs (i.e.: a CPU-bound process), the daemon first increases the voltage to the next safe  $V_{min}$  level (see Table 20) and then, if the voltage can be decreased according to utilized PMDs (this information is provided by the monitoring part), the daemon will set the voltage accordingly (as shown in Figure 66). By following this policy, there is a minimal increase of the total power consumption, however, this guarantees the reliable execution on a real system. The Placement part uses the classification performed by the Monitoring part to guide its decisions, and is invoked upon every process list or classification change.

The online monitoring daemon is minimally intrusive and has no impact on the safe  $V_{min}$ . Its performance overhead is also negligible, as it is only running periodically to read the performance counters and upon every process list change, to invoke the placement process, which has equal impact as a process migration of the Linux kernel. The daemon is invoked only after

- either a new process is issued to the system or when a process finishes its execution (to check if a process migration is required), or
- when a process changes its state (from CPU-intensive to the memory-intensive and vice versa).

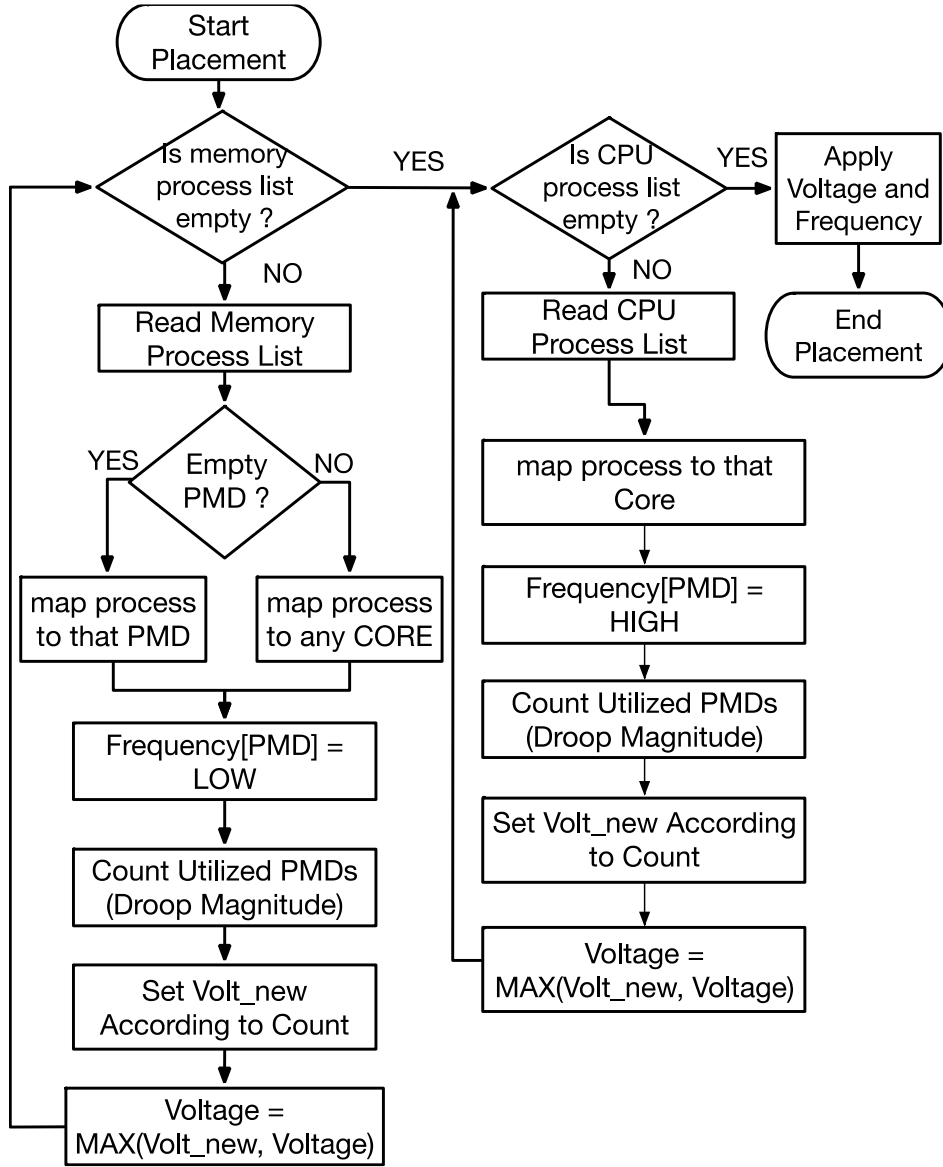
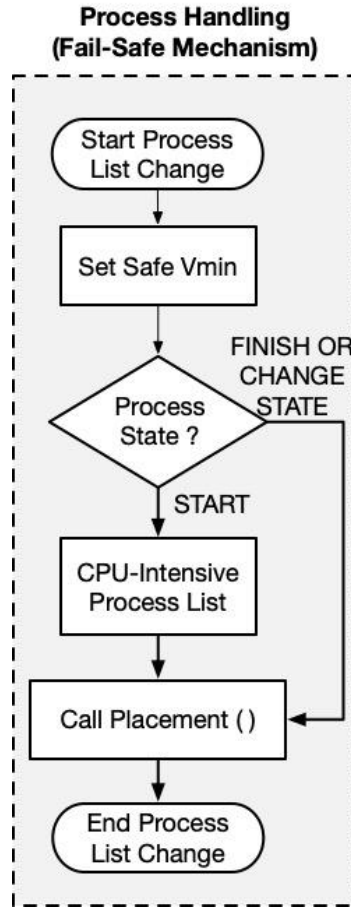


Figure 66: Placement flow chart.

In case (a), it reads the process mapping table and estimates the result, and in case (b), it periodically counts the L3C accesses (as we described in subsection 4.3.1). Note that, in case (b) the utilized PMDs cannot be changed. Utilized PMDs can only be changed when a new process is invoked, or when a process finishes its execution.

To count the L3C accesses, we leverage the built-in performance monitoring counters of the microprocessors. To do so, we developed a kernel module able to provide access to the performance counters from user-space in order to be part of our online monitoring daemon. It is lightweight (thus fast) because it acts in the kernel space and provides near zero overhead to the total microprocessor's operation. We did not use tools like Perf [144] or PAPI [167] because these tools impose an extra overhead in measurements ( $\pm 3\%$ ), while we need very accurate values to take correct decisions. To count the L3C accesses, only one read of one PMU counter and one read of the same register after 1M cycles are required. Afterwards, the kernel module subtracts these two values to produce the final result.



**Figure 67: Process handling (Fail-Safe Mechanism).**

#### 4.5.2 Evaluation Results

For the purposes of our experimental evaluation, we also developed a “workload generator” which creates a typical server workload from a “pool” of programs (which includes all the 29 SPEC CPU2006 and the 6 NPB benchmarks; in total 35 different programs). The generator can generate workloads of configurable duration by randomly selecting benchmarks from this pool and randomly defining the timeslot in which each benchmark will be invoked. The workload includes heavy load periods, average load periods and light periods, including also a few idle periods, resembling a typical server workload on a given time window. The generator is configured to guarantee that the number of active processes is never more than the available cores of the microprocessor. The generated workload can be then invoked multiple times, allowing multiple experiments under the same load conditions, using different policies or configurations. Our exploration has revealed potential energy efficiency improvements by adjusting the CPU voltage, frequency and the core allocation. In this subsection we present the results for a simple realistic experimentation we performed in our two systems. To provide detailed results for a comprehensive analysis, we ran 4 different configurations for the same workload sequence:

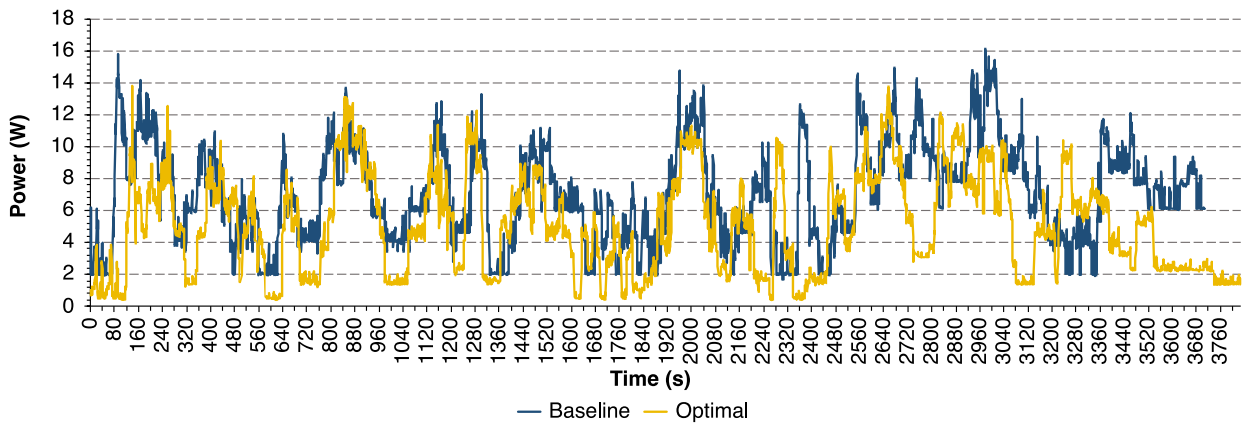
- **Baseline:** in this configuration we run the workload sequence with the default microprocessor’s frequency scaling and scheduler settings (ondemand governor is enabled).
- **Safe  $V_{min}$ :** in this configuration we change the nominal voltage of the microprocessor to the safe  $V_{min}$ , according to Table 20, (again with ondemand

governor enabled). With this configuration we evaluate the impact of pessimistic voltage guardbands in energy.

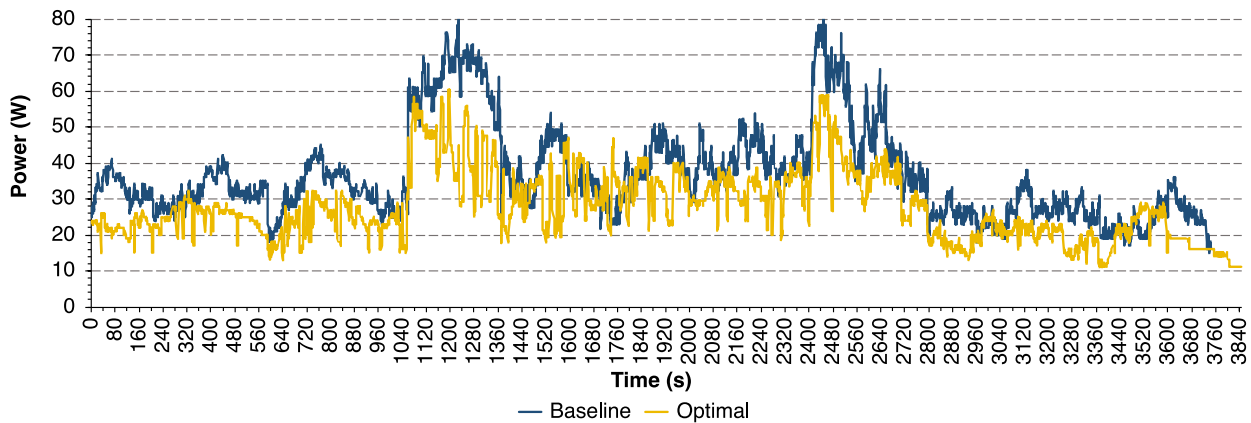
- **Placement:** in this configuration we run the simple online monitoring daemon to perform the proposed frequency and core allocation (the ondemand governor is now disabled), but keeping the voltage at its nominal value. With this configuration we evaluate the impact of the proposed frequency and core allocation options in energy and performance.
- **Optimal:** in this configuration we run the simple online monitoring daemon and adapting the voltage and frequency settings (with ondemand governor disabled). This is the best scenario which integrates all the conditions targeting the best energy efficiency.

Figure 68 and Figure 69 show the average power for a 1-hour execution of randomly generated workload for the default microprocessor's settings (Baseline) and the Optimal scheme, for X-Gene 2 and X-Gene 3, respectively. We generated 2 workloads, one for X-Gene 2 using a maximum constraint of 8 cores, and one for X-Gene 3 with a constraint of 32 cores. Each workload was executed under both Baseline and Optimal settings.

In Figure 68 and Figure 69 that compare the Baseline and the Optimal configuration, we can see that the average power consumption for the optimal configuration compared to the default one (for the same 1-hour workload) is significantly reduced for both



**Figure 68: Average power for the Baseline and the Optimal configurations in X-Gene 2 during 1-hour execution.**



**Figure 69: Average power for the Baseline and the Optimal configurations in X-Gene 3 during 1-hour execution.**



Table 21: X-Gene 2 results for the 4 configurations.

	Baseline	Safe $V_{min}$	Placement	Optimal
Time (s)	3707	3707	3829	3829
Avg. Power (W)	6.90	6.10	5.46	5.00
Energy (J)	25578.30	22612.07	20906.34	19145.00
Energy Savings	--	11.6%	18.3%	<b>25.2%</b>
ED2P (workload)	$351 \times 10^9$	$311 \times 10^9$	$307 \times 10^9$	$281 \times 10^9$
ED2P Savings	--	11.6%	12.8%	<b>20.1%</b>

Table 22: X-Gene 3 results for the 4 configurations.

	Baseline	Safe $V_{min}$	Placement	Optimal
Time (s)	3748	3748	3846	3846
Avg. Power (W)	36.49	32.51	30.78	27.63
Energy (J)	136773.26	121847.48	118379.88	106283.56
Energy Savings	--	10.9%	13.4%	<b>22.3%</b>
ED2P (workload)	$19 \times 10^{11}$	$17 \times 10^{11}$	$17 \times 10^{11}$	$15 \times 10^{11}$
ED2P Savings	--	10.9%	8.9%	<b>18.2%</b>

microprocessors. As presented in these figures, the system has a load with phases of high utilization and others with low utilization, resembling a typical server workload. We can see that some peaks reach the maximum capability of the system, indicating that the system was occasionally pushed towards its limits.

In Table 21 and Table 22 we compare the 4 different configurations described above. We can see a significant reduction in the total energy: 25.2% in X-Gene 2 (Table 21) and 22.3% in X-Gene 3 (Table 22). This was achieved without compromising the performance of CPU-intensive programs, while slightly reducing the performance of memory-intensive programs, as described in Section 4.2, targeting the highest ED2P efficiency. The use of ED2P metric guarantees that the selected policies do not violate high performance constraints and at the same time, significantly reduce the energy consumption.

The Optimal scheme with the online monitoring daemon did also slightly shift the completion time of the workload as a result of the small performance impact on some memory-intensive programs (as described in Section 4.2). The total time was shifted by 3.2% in X-Gene 2 and 2.5% in X-Gene 3. We can also notice in Table 21 and Table 22

that the frequency and core allocation options (Placement) are the major contributors on the total energy reduction (18.3% in X-Gene 2 and 13.4% in X-Gene 3), compared to the voltage reduction with the *ondemand* governor (Safe  $V_{min}$ ). In any case, the Optimal scheme can achieve more than 22% energy savings with a minimal performance penalty.

## 4.6 Related Work

**Characterization:** Bacha *et al.* [7] [8] focus on monitoring the hardware-reported errors manifested in the caches of a commercial Intel Itanium processor during the execution of benchmarks in off-nominal voltage conditions. Authors in [12] [157] [176] [177] [186] measure single-core voltage margins in several commercial microprocessor chips to study not only the pessimistic voltage guardbands of the chips, but also the core-to-core and chip-to-chip variations for single-core executions.

Sasaki *et al.* [180], study the prevalence of power capping when multiple processes in a multicore microprocessor compete for power, while the power management system attempts to mitigate the contention (reduce the power consumption) by slowing down the processor. Several characterization studies have been presented for off-nominal voltage conditions operation of commercial microprocessor chips with up to 8 cores (e.g., [6] [7] [8] [9] [11] [12] [84] [145] [167] [176] [186]).

In this work we present the characterization of multi-threaded workloads on high-end microprocessors with significantly larger core counts (up to 32-core). To our knowledge, the observations we make for the safe  $V_{min}$ , voltage droop magnitude and the reduced workload variation in actual hardware (not simulated models) of 8-core and 32-core microprocessors, have not been presented in the literature before.

**Voltage Noise Characterization and Mitigation:** Ketkar *et al.* [187] and Kim *et al.* [188], [189] propose methods to maximize voltage droops in single core and multicore chips in order to investigate their worst-case behavior due to the generated voltage noise effects. Bertran *et al.* [152] present a voltage noise characterization study in multi-core microprocessors, in which they use a systematic methodology to generate noise stressmarks.

Studies of Gupta *et al.* [190] and Reddi *et al.* [143] focus on the prediction of critical execution points of benchmarks, in which large voltage noise glitches are likely to occur, leading to system malfunctions. In the same context, several studies either in the hardware or in the software level have been presented to mitigate the effects of voltage noise [5] [84] [149] [191] [192] [193] [194] [195] [196] or to recover from them after their occurrence [197]. Some works [198], [199], [200], and [201] focus on the development of electrical simulation framework for power-delivery analysis Reddi *et al.* [5] show that different workloads have different voltage margins and different voltage droop activity based on a 2-core microprocessor.

Unlike these previous studies, in this work we are based on 8-core and 32-core microprocessors and show that as the number of active cores increases (4 or more active cores), the emergency voltage droops are massive and lead to workload-independent  $V_{min}$ .

**Workload Scheduling and DVFS:** Energy Aware Scheduling (EAS) project [202], is trying to solve power-management limitations on big.LITTLE designs by balancing the load across all CPUs, while saving power by scaling down the frequency of the CPUs or idling them. Li and Martinez [203] optimize a parallel workload running on a simulated 16-core microprocessor by dynamically changing the number of processors and the V/F levels. Isci *et al.* [204] consider a simulated 4-core microprocessor with per-core DVFS and examine different DVFS policies for high performance and power efficiency.

Their solutions are not scalable to large systems, and to this end, Teodorescu and Torrellas [156] consider a simulated 20-core microprocessor, again assuming a per-core DVFS approach, to propose variation-aware algorithms for power management. However, in this work we show with actual measurements on multicore microprocessors that, as the number of active threads increases, there is a minimal variation across different workloads and cores. Vega *et al.* [205] propose a coordinated power management algorithm, which dynamically detects situations in which the program may be bound by single-thread-performance or throughput and actuates the power management accordingly.

Miftakhutdinov *et al.* in [206] propose a low-cost mechanism that accurately predicts the performance impact of frequency scaling in the presence of a realistic memory system. Castillo *et al.* in [207] propose a hardware mechanism that dynamically adapts the computational power of a task depending on its criticality. Authors in [178] [179] have exposed the phases of a program which are memory intensive and proposed this feature as a proxy in a single-core microprocessor to guide the DVFS to reduce the frequency in that specific program phases.

Unlike these previous studies, in this work, we are based on real implementations of 8-core and 32-core microprocessors, whose frequency can be set per pair of cores and all cores have the same voltage, and present a different approach for V/F settings, according to workload characteristics (not program phases) and voltage droop magnitude, for the dynamic scheduling and migration of the programs on the available microprocessor cores.



## 5. CONCLUSION AND FUTURE WORK

In this thesis, we presented two complementary methods to accelerate the post-silicon validation phase of modern microprocessors and to guarantee their energy efficiency. More specifically, we presented our contributions on post-silicon validation of the address translation mechanisms of modern microprocessors. We presented a comprehensive set of bug models, which correspond to the address translation mechanisms and classify the effects of both functional and electrical bugs in the hardware structures employed in address translation. We then presented and described an ISA-independent software-based post-silicon validation method, which contribute to accelerate the bug detection process in the address translation mechanisms of modern microprocessors and fully exploits the silicon's performance. The method is easily applied to any ISA. We demonstrated that our method reduces the bug detection latency by 5 orders of magnitude compared to traditional end-of-test checking techniques by fully resembling a post-silicon validation bare-metal setup. Afterwards, we presented another novel post-silicon self-checking validation method, which complements the first one. This post-silicon validation method aims to unveil and detect rare bug scenarios in address translation caching arrays (ATCA). ATCAs are among the most important structures for microprocessor functionality and performance and escaped bugs in these arrays can lead to unpredictable system behaviors. By using a comprehensive experimental study, we presented and analyzed rare bug scenarios and why they are difficult to detect. The primary contribution here was that even if the bugs manifest themselves by executing traditional validation tests, detecting them is unlikely due to the high possibility of masking during the execution of the validation test.

We then presented a detailed system-level voltage scaling characterization study for single-core executions in ARMv8-based multicore CPUs manufactured in 28nm. The study's backbone was a fully automated system-level framework aims to increase the throughput of massive undervolting campaigns that require multiple benchmarks execution at several voltage supply levels of all individual cores. Towards the formalization of the behavior in undervolting conditions we also presented a simple consolidated function; the *Severity function*, which aggregates the effects of reduced voltage operation in the cores of a multicore CPU by assigning values to the different abnormal observations. Aiming to accelerate the time-consuming characterization process, we also introduced the development of dedicated programs (diagnostic micro-viruses) that aim to stress the fundamental hardware components of APM's X-Gene 2 micro-server family and provide quickly the safe  $V_{min}$  values for each core. These diagnostic micro-viruses are executed in very short time (~3 days for the entire massive characterization campaign for each individual core of one microprocessor chip) compared to normal benchmarks, such as those of the SPEC CPU2006 suite, which need 2 months. The micro-viruses' purpose is to reveal the variation of the safe voltage margins across cores of the multicore chip and also to contribute to diagnosis by exposing and classifying the abnormal behaviour of each CPU unit (silent data corruptions, bit-cell errors, and timing failures).

The final contribution of this thesis in the area of energy-efficiency was a detailed system-level voltage scaling characterization study for multicore executions in two recent ARMv8-based multicore CPUs manufactured in 28nm and 16nm. Based on this study, we discussed several important observations and presented a new software-based scheme for these server-grade machines, which considers all the diverse aspects of the increased energy consumption and provides large energy savings while maintaining high performance levels. Particularly, we showed that the proposed software scheme can achieve on average 25.2% energy savings on X-Gene 2, and 22.3% energy savings on

X-Gene 3, with a minimal performance penalty of 3.2% on X-Gene 2 and 2.5% on X-Gene 3 compared to the default voltage and frequency microprocessor's conditions.

The research outcomes of this thesis can be useful to several future directions. Future systems architectures must be designed to facilitate hardware validation. In the post-silicon validation domain, future research should focus on the automation and standardization of the design bug detection and root-cause analysis process. More specifically, in this thesis we demonstrated the effectiveness of software-based self-checking techniques in accelerating the post-silicon validation phase of the address translation mechanisms. This may be an indication that future microprocessors should devote valuable silicon estate in hardware hooks that enable the at-speed, low-cost testing. In the energy-efficiency domain, future research should focus on building microarchitectural solutions, which will be able to mitigate the emergency voltage conditions that lead the microprocessor designers to overestimate the voltage margins. Moreover, another interesting future direction will be a comprehensive comparison among different architectures and microarchitectures of heterogeneous platforms, regarding the voltage reduction tolerance for different types of applications. Such a study could be very useful especially on large datacentres that use heterogeneous distributed computing systems that consist of contemporary general-purpose processors (CPUs), general-purpose graphic processing units (GP-GPUs), and field-programmable gate arrays (FPGAs). The utilization of heterogeneous architectures poses several challenges, primarily due to increased power consumption, and thus, voltage reduction could be deliver significant energy savings.

**ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ**

<b>Ξενόγλωσσος όρος</b>	<b>Ελληνικός Όρος</b>
Architecture	Αρχιτεκτονική
Benchmarks	Μετροπρογράμματα
Bug	Σφάλμα
Cache	Κρυφή Μνήμη
Circuit	Κύκλωμα
Complexity	Πολυπλοκότητα
Consumption	Κατανάλωση
Core	Πυρήνας
Design	Σχεδίαση
Efficiency	Αποδοτικότητα
Energy	Ενέργεια
Frequency	Συχνότητα
Hardware	Υλικό
Kernel	Πυρήνας
Microarchitecture	Μικροαρχιτεκτονική
Microprocessor	Μικροεπεξεργαστής
Operating System	Λειτουργικό Σύστημα
Out-of-Order	Εκτέλεση εκτός σειράς
Performance	Απόδοση
Pipeline	Διοχέτευση
Power	Ισχύς
Process	Διεργασία
Register	Καταχωρητής
Scaling	Κλιμάκωση
Simulator	Προσομοιωτής
Software	Λογισμικό
Thread	Νήμα
Undervolting	Μείωση της τάσης
Validation	Επικύρωση
Voltage	Τάση
Voltage Guardband	Περιθώριο τάσης





**ACRONYMS**

ACPI	Advanced Configuration and Power Interface
ALU	Arithmetic and Logic Unit
ATCA	Address Translation Caching Arrays
ATM	Address Translation Mechanisms
BPU	Branch Prediction Unit
CE	Corrected Error
CPPC	Collaborative Processor Performance Control
CR0	Control Register 0
CR3	Control Register 3
DTLB	Data Translation Lookaside Buffer
DUV	Design Under Verification
DVFS	Dynamic Voltage and Frequency Scaling
EAS	Energy Aware Scheduling
ECC	Error Correction Code
ED2P	Energy-Delay Squared Product
EDP	Energy-Delay Product
FPU	Floating-Point Unit
IPC	Inter-Process Communication
IPC	Instruction per Cycle
ISA	Instruction Set Architecture
ITLB	Instruction Translation Lookaside Buffer
MMU	Memory Management Unit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSR	Machine State Register
OoO	Out of Order
PA	Physical Address
PCU	Power Control Unit
PD	Page Directory
PDP	Page Directory Pointer
pFail	Probability of Failure
PFN	Physical Frame Number
PID	Process Identification
PMC	Performance Monitoring Counter

PMD	Processor MoDule
PML4	Page Map Level 4
PMpro	Power Management processor
PMU	Performance Monitoring Unit
PT	Page Table
PTE	Page Table Entry
RAM	Random Access Memory
SDC	Silent Data Corruption
SECDED	Single-Error Detection / Double-Error Correction
SLIMpro	Scalable Lightweight Intelligent Management processor
SoC	System on Chip
TLB	Translation Lookaside Buffer
UE	Uncorrected Error
VA	Virtual Address
VPN	Virtual Page Number

**ANNEX I**


---

**Event-id Description**


---

0x000	Instruction architecturally executed, condition code check pass, software increment
0x001	L1 Instruction cache refill
0x002	L1 instruction TLB refill
0x003	L1 data cache refill
0x004	L1 data cache access
0x005	L1 data TLB refill
0x008	Instruction architecturally executed
0x009	Exception taken
0x00A	Instruction architecturally executed (condition check pass) -Exception return
0x00B	Instruction architecturally executed (condition check pass) - Write to CONTEXTIDR
0x010	Mispredicted or not predicted branch speculatively executed
0x011	Cycle
0x012	Predictable branch speculatively executed
0x013	Data memory access
0x014	L1 instruction cache access
0x016	L2 data cache access
0x017	L2 data cache refill
0x018	L2 data cache write-back
0x019	Bus access
0x01A	Local Memory Error
0x01B	Operation speculatively executed
0x01C	Instruction architecturally executed (condition check pass) - Write to translation table base
0x01E	Counter chain
0x040	L1 data cache access - Read
0x041	L1 data cache access - Write
0x042	L1 data cache refill - Read
0x048	L1 data cache invalidate 0x04C L1 data TLB refill – Read
0x04D	L1 data TLB refill - Write
0x050	L2 data cache access - Read
0x051	L2 data cache access - Write
0x052	L2 data cache refill - Read
0x053	L2 data cache refill - Write
0x056	L2 data cache write-back - victim
0x057	L2 data cache write-back - Cleaning and coherency
0x058	L2 data cache invalidate
0x060	Bus access - Read
0x061	Bus access - Write
0x062	Bus access - Normal, cacheable, sharable
0x063	Bus access - Not normal, cacheable, sharable
0x064	Bus access - Normal
0x065	Bus access - Peripheral
0x066	Data memory access – Read
0x067	Data memory access - write
0x068	Unaligned access - Read

0x069	Unaligned access - Write
0x06A	Unaligned access
0x06C	Exclusive operation speculatively executed - Load exclusive
0x06D	Exclusive operation speculative executed - Store exclusive pass
0x06E	Exclusive operation speculative executed - Store exclusive fail
0x06F	Exclusive operation speculatively executed - Store exclusive
0x070	Operation speculatively executed - Load
0x071	Operation speculatively executed - Store
0x072	Operation speculatively executed - Load or store
0x073	Operation speculatively executed - Integer data processing
0x074	Operation speculatively executed - Advanced SIMD
0x075	Operation speculatively executed - FP
0x076	Operation speculatively executed - Software change of PC
0x078	Branch speculative executed - Immediate branch
0x079	Branch speculative executed - Procedure return
0x07A	Branch speculative executed - Indirect branch
0x07C	Barrier speculatively executed - ISB
0x07D	Barrier speculatively executed - DSB
0x07E	Barrier speculatively executed - DMB
0x081	Exception taken, other synchronous
0x082	Exception taken, Supervisor Call
0x083	Exception taken, Instruction Abort
0x084	Exception taken, Data Abort or SError
0x086	Exception taken, IRQ
0x087	Exception taken, FIQ
0x08A	Exception taken, Hypervisor Call
0x08B	Exception taken, Instruction Abort not taken locally
0x08C	Exception taken, Data Abort or SError not taken locally
0x08D	Exception taken, other traps not taken locally
0x08E	Exception taken, IRQ not taken locally
0x08F	Exception taken, FIQ not taken locally
0x090	Release consistency instruction speculatively executed - Load Acquire
0x091	Release consistency instruction speculatively executed - Store Release
0x100	Operation speculatively executed - NOP
0x101	FSU clocking gated off cycle
0x102	BTB misprediction
0x103	ITB miss
0x104	DTB miss
0x105	L1 data cache late miss
0x106	L1 data cache prefetch request
0x107	L2 data prefetch request
0x108	Decode starved for instruction cycle
0x109	Op dispatch stalled cycle
0x10A	IXA Op non-issue
0x10B	IXB Op non-issue
0x10C	BX Op non-issue
0x10D	LX Op non-issue
0x10E	SX Op non-issue
0x10F	FX Op non-issue
0x110	Wait state cycle
0x111	L1 stage-2 TLB refill
0x112	Page Walk Cache level-0 stage-1 hit

0x113	Page Walk Cache level-1 stage-1 hit
0x114	Page Walk Cache level-2 stage-1 hit
0x115	Page Walk Cache level-1 stage-2 hit
0x116	Page Walk Cache level-2 stage-2 hit



## REFERENCES

- [1] P. Patra, "On the cusp of a validation wall," *IEEE Design & Test of Computers*, vol. 24, no. 2, pp. 193–196, Feb. 2007 [Online]. Available: <http://dx.doi.org/10.1109/MDT.2007.54>
- [2] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, and R. Morad, "ISA-independent post-silicon validation for the address translation mechanisms of modern microprocessors," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016 [Online]. Available: <http://dx.doi.org/10.1109/IOLTS.2016.7604675>
- [3] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, and R. Morad, "An Agile Post-Silicon Validation Methodology for the Address Translation Mechanisms of Modern Microprocessors," *IEEE Transactions on Device and Materials Reliability*, vol. 17, no. 1, pp. 3–11, Mar. 2017 [Online]. Available: <http://dx.doi.org/10.1109/TDMR.2016.2636880>
- [4] G. Papadimitriou, D. Gizopoulos, A. Chatzidimitriou, T. Kolan, A. Koyfman, R. Morad and V. Sokhin, "Unveiling difficult bugs in address translation caching arrays for effective post-silicon validation," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016 [Online]. Available: <http://dx.doi.org/10.1109/ICCD.2016.7753339>
- [5] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2010.35>
- [6] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in POWER7," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*, 2011 [Online]. Available: <http://dx.doi.org/10.1145/2155620.2155622>
- [7] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors," in *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13*, 2013 [Online]. Available: <http://dx.doi.org/10.1145/2485922.2485948>
- [8] A. Bacha and R. Teodorescu, "Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2014.54>
- [9] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe limits on voltage reduction efficiency in GPUs," in *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2830772.2830811>
- [10] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, G. Favor, K. Sankaran and S. Das, "A system-level voltage/frequency scaling characterization framework for multicore CPUs," in *13th IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE '17)*, Boston, MA, USA, 2017.
- [11] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing voltage margins for energy efficiency in multicore CPUs," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-50 '17*, 2017 [Online]. Available: <http://dx.doi.org/10.1145/3123939.3124537>
- [12] G. Papadimitriou, A. Chatzidimitriou, M. Kaliorakis, Y. Vastakis, and D. Gizopoulos, "Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018 [Online]. Available: <http://dx.doi.org/10.1109/ISPASS.2018.00014>
- [13] G. Papadimitriou, A. Chatzidimitriou, and D. Gizopoulos, "Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2019.00033>
- [14] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, 1965, pp. 114–117 [Online]. Available: <http://dx.doi.org/10.1109/jproc.1998.658762>
- [15] G. Moore, "Progress in Digital Integrated Electronics," *IEEE Int'l Electron Devices Meeting Technical Digest*, 1975, pp. 11–13.
- [16] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974 [Online]. Available: <http://dx.doi.org/10.1109/JSSC.1974.1050511>

- [17] M. T. Bohr and I. A. Young, "CMOS Scaling Trends and Beyond," *IEEE Micro*, vol. 37, no. 6, pp. 20–29, Nov. 2017 [Online]. Available: <http://dx.doi.org/10.1109/MM.2017.4241347>
- [18] M. Bohr, "The new era of scaling in an SoC world," in 2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers, 2009 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2009.4977293>
- [19] T. Ghani *et al.*, "A 90nm high volume manufacturing logic technology featuring novel 45nm gate length strained silicon CMOS transistors," in IEEE International Electron Devices Meeting 2003 [Online]. Available: <http://dx.doi.org/10.1109/IEDM.2003.1269442>
- [20] S. E. Thompson *et al.*, "A 90-nm Logic Technology Featuring Strained-Silicon," *IEEE Transactions on Electron Devices*, vol. 51, no. 11, pp. 1790–1797, Nov. 2004 [Online]. Available: <http://dx.doi.org/10.1109/TED.2004.836648>
- [21] P. Bai *et al.*, "A 65nm logic technology featuring 35nm gate lengths, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57  $\mu\text{m}/\text{sub } 2/$  SRAM cell," in IEDM Technical Digest. IEEE International Electron Devices Meeting, 2004. [Online]. Available: <http://dx.doi.org/10.1109/IEDM.2004.1419253>
- [22] K. Mistry *et al.*, "A 45nm Logic Technology with High-k+Metal Gate Transistors, Strained Silicon, 9 Cu Interconnect Layers, 193nm Dry Patterning, and 100% Pb-free Packaging," in 2007 IEEE International Electron Devices Meeting, 2007 [Online]. Available: <http://dx.doi.org/10.1109/IEDM.2007.4418914>
- [23] C. Auth *et al.*, "45nm High-k + metal gate strain-enhanced transistors," in 2008 Symposium on VLSI Technology, 2008 [Online]. Available: <http://dx.doi.org/10.1109/VLSIT.2008.4588589>
- [24] K. J. Kuhn, "Reducing Variation in Advanced Logic Technologies: Approaches to Process and Design for Manufacturability of Nanoscale CMOS," in 2007 IEEE International Electron Devices Meeting, 2007 [Online]. Available: <http://dx.doi.org/10.1109/IEDM.2007.4418976>
- [25] S. Natarajan *et al.*, "A 32nm logic technology featuring 2nd-generation high-k + metal-gate transistors, enhanced channel strain and 0.171  $\mu\text{m}^2$  SRAM cell size in a 291Mb array," in 2008 IEEE International Electron Devices Meeting, 2008 [Online]. Available: <http://dx.doi.org/10.1109/IEDM.2008.4796777>
- [26] Y. Wang *et al.*, "A 4.0 GHz 291Mb voltage-scalable SRAM design in 32nm high-k metal-gate CMOS with integrated power management," in 2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers, 2009 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2009.4977505>
- [27] S. Tam, S. Rusu, U. Nagarji Desai, R. Kim, Ji Zhang, and I. Young, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000 [Online]. Available: <http://dx.doi.org/10.1109/4.881198>
- [28] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs, "The Implementation of the 65nm Dual-Core 64b Merom Processor," in 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, 2007 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2007.373610>
- [29] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195–237, Sep. 2005 [Online]. Available: <http://dx.doi.org/10.1145/1108956.1108957>
- [30] R. Kumar and G. Hinton, "A family of 45nm IA processors," in 2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers, 2009 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2009.4977306>
- [31] K. Zhang *et al.*, "SRAM design on 65nm CMOS technology with integrated leakage reduction scheme," in 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525) [Online]. Available: <http://dx.doi.org/10.1109/VLSIC.2004.1346592>
- [32] N. Kurd, J. Douglas, P. Mosalikanti, and R. Kumar, "Next generation Intel® micro-architecture (Nehalem) clocking architecture," in 2008 IEEE Symposium on VLSI Circuits, 2008 [Online]. Available: <http://dx.doi.org/10.1109/VLSIC.2008.4585952>
- [33] P. I. Péntzes, M. Nyström, and A. J. Martin, "Transistor sizing of energy-delay-efficient circuits," in Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems - TAU '02, 2002 [Online]. Available: <http://dx.doi.org/10.1145/589411.589439>



- [34] J. Ebergen, J. Gainsley, and P. Cunningham, "Transistor sizing: how to control the speed and energy consumption of a circuit," in 10th International Symposium on Asynchronous Circuits and Systems, 2004. Proceedings. [Online]. Available: <http://dx.doi.org/10.1109/ASYNC.2004.1299287>
- [35] E. Kursun, S. Ghiasi, and M. Sarrafzadeh, "Transistor level budgeting for power optimization," in SCS 2003. International Symposium on Signals, Circuits and Systems. Proceedings (Cat. No.03EX720) [Online]. Available: <http://dx.doi.org/10.1109/ISQED.2004.1283660>
- [36] A. K. Sultania, D. Sylvester, and S. S. Sapatnekar, "Transistor and pin reordering for gate oxide leakage reduction in dual  $T_{ox}$  circuits," in IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. [Online]. Available: <http://dx.doi.org/10.1109/ICCD.2004.1347927>
- [37] D. Chen, J. Cong, F. Li, and L. He, "Low-power technology mapping for FPGA architectures with dual supply voltages," in Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays - FPGA '04, 2004 [Online]. Available: <http://dx.doi.org/10.1145/968280.968297>
- [38] H. Li, S. Katkoori, and W.-K. Mak, "Power minimization algorithms for LUT-based FPGA technology mapping," ACM Transactions on Design Automation of Electronic Systems, vol. 9, no. 1, pp. 33–51, Jan. 2004 [Online]. Available: <http://dx.doi.org/10.1145/966137.966139>
- [39] N. Dragone, R. A. Rutenbar, L. R. Carley, and R. Zafalon, "Low-power technology mapping for mixed-swing logic," in ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No.01TH8581) [Online]. Available: <http://dx.doi.org/10.1109/LPE.2001.945420>
- [40] A. G. M. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in Proceedings of the 2000 international symposium on Low power electronics and design - ISLPED '00, 2000 [Online]. Available: <http://dx.doi.org/10.1145/344166.344540>
- [41] B.-S. Kong, S.-S. Kim, and Y.-H. Jun, "Conditional-capture flip-flop for statistical power reduction," IEEE Journal of Solid-State Circuits, vol. 36, no. 8, pp. 1263–1271, 2001 [Online]. Available: <http://dx.doi.org/10.1109/4.938376>
- [42] N. Nedovic, M. Aleksic, and V. G. Oklobdzija, "Conditional techniques for low power consumption flip-flops," in ICECS 2001. 8th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.01EX483) [Online]. Available: <http://dx.doi.org/10.1109/ICECS.2001.957596>
- [43] P. Zhao, T. K. Darwish, and M. A. Bayoumi, "High-performance and low-power conditional discharge flip-flop," IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 12, no. 5, pp. 477–484, May 2004 [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2004.826192>
- [44] R. P. Llopis and M. Sachdev, "Low power, testable dual edge triggered flip-flops," in Proceedings of 1996 International Symposium on Low Power Electronics and Design [Online]. Available: <http://dx.doi.org/10.1109/LPE.1996.547536>
- [45] F. Gao and J. P. Hayes, "ILP-based optimization of sequential circuits for low power," in Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03. [Online]. Available: <http://dx.doi.org/10.1109/LPE.2003.1231850>
- [46] D. Ernst, N. Sung Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," In Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36), 2003. [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2003.1253179>
- [47] K. Banerjee and A. Mehrotra, "Global (interconnect) warming," IEEE Circuits and Devices Magazine, vol. 17, no. 5, pp. 16–32, 2001 [Online]. Available: <http://dx.doi.org/10.1109/101.960685>
- [48] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 3, no. 1, pp. 49–58, Mar. 1995 [Online]. Available: <http://dx.doi.org/10.1109/92.365453>
- [49] C. N. Taylor, S. Dey, and Y. Zhao, "Modeling and minimization of interconnect energy dissipation in nanometer technologies," in Proceedings of the 38th conference on Design automation - DAC '01, 2001 [Online]. Available: <http://dx.doi.org/10.1145/378239.379060>
- [50] H. Zhang and J. Rabaey, "Low-swing interconnect interface circuits," in Proceedings of the 1998 international symposium on Low power electronics and design - ISLPED '98, 1998 [Online]. Available: <http://dx.doi.org/10.1145/280756.280876>
- [51] W.-B. Jone, J. S. Wang, H.-I. Lu, I. P. Hsu, and J.-Y. Chen, "Design theory and implementation for low-power segmented bus systems," ACM Transactions on Design Automation of Electronic

- Systems, vol. 8, no. 1, pp. 38–54, Jan. 2003 [Online]. Available: <http://dx.doi.org/10.1145/606603.606606>
- [52] V. Lyuboslavsky, B. Bishop, V. Narayanan, and M. J. Irwin, “Design of databus charge recovery mechanism,” in Proceedings of 13th Annual IEEE International ASIC/SOC Conference (Cat. No.00TH8541) [Online]. Available: <http://dx.doi.org/10.1109/ASIC.2000.880750>
- [53] H. Zhang, M. Wan, V. George, and J. Rabaey, “Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs,” in Proceedings. IEEE Computer Society Workshop on VLSI '99. System Design: Towards System-on-a-Chip Paradigm. [Online]. Available: <http://dx.doi.org/10.1109/IWV.1999.760456>
- [54] W. J. Dally and B. Towles, “Route packets, net wires,” in Proceedings of the 38th conference on Design automation - DAC '01, 2001 [Online]. Available: <http://dx.doi.org/10.1145/378239.379048>
- [55] M. Sgroi *et al.*, “Addressing the system-on-a-chip interconnect woes through communication-based design,” in Proceedings of the 38th conference on Design automation - DAC '01, 2001 [Online]. Available: <http://dx.doi.org/10.1145/378239.379045>
- [56] H. Wang, L.-S. Peh, and S. Malik, “Power-driven design of router microarchitectures in on-chip networks,” in Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-'36), 2003. [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2003.1253187>
- [57] V. De La Luz, M. Kandemir, and I. Kolcu, “Automatic data migration for reducing energy consumption in multi-bank memory systems,” in Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324), 2002 [Online]. Available: <http://dx.doi.org/10.1109/DAC.2002.1012622>
- [58] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, “Power aware page allocation,” in Proceedings of the ninth international conference on Architectural support for programming languages and operating systems - ASPLOS-IX, 2000 [Online]. Available: <http://dx.doi.org/10.1145/378993.379007>
- [59] K. Ghose and M. B. Kamble, “Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation,” in Proceedings of the 1999 international symposium on Low power electronics and design - ISLPED '99, 1999 [Online]. Available: <http://dx.doi.org/10.1145/313817.313860>
- [60] J. Kin, Munish Gupta, and W. H. Mangione-Smith, “The filter cache: an energy efficient memory structure,” in Proceedings of 30th Annual International Symposium on Microarchitecture [Online]. Available: <http://dx.doi.org/10.1109/MICRO.1997.645809>
- [61] M. Powell, Se-Hyun Yang, B. Falsafi, K. Roy, and N. Vijaykumar, “Reducing leakage in a high-performance deep-submicron instruction cache,” IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 9, no. 1, pp. 77–89, Feb. 2001 [Online]. Available: <http://dx.doi.org/10.1109/92.920821>
- [62] C. H. Kim and K. Roy, “Dynamic  $V_{th}$  scaling scheme for active leakage power reduction,” in Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition [Online]. Available: <http://dx.doi.org/10.1109/DATE.2002.998265>
- [63] S. Kaxiras, Zhigang Hu, and M. Martonosi, “Cache decay: exploiting generational behavior to reduce cache leakage power,” in Proceedings 28th Annual International Symposium on Computer Architecture [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2001.937453>
- [64] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, and D. Albonesi, “An Adaptive Issue Queue for Reduced Power at High Performance,” in Power-Aware Computer Systems, Springer Berlin Heidelberg, 2001, pp. 25–39 [Online]. Available: [http://dx.doi.org/10.1007/3-540-44572-2\\_3](http://dx.doi.org/10.1007/3-540-44572-2_3)
- [65] R. I. Bahar and S. Manne, “Power and energy reduction via pipeline balancing,” in Proceedings of the 28th annual international symposium on Computer architecture - ISCA '01, 2001 [Online]. Available: <http://dx.doi.org/10.1145/379240.379265>
- [66] D. Folegnani and A. González, “Energy-effective issue logic,” in Proceedings of the 28th annual international symposium on Computer architecture - ISCA '01, 2001 [Online]. Available: <http://dx.doi.org/10.1145/379240.379266>
- [67] C. J. Hughes, J. Srinivasan, and S. V. Adve, “Saving energy with architectural and frequency adaptations for multimedia applications,” in Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2001.991123>
- [68] R. Sasanka, C. J. Hughes, and S. V. Adve, “Joint local and global hardware adaptations for energy,” in Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X) - ASPLOS '02, 2002 [Online]. Available: <http://dx.doi.org/10.1145/635508.605413>

- [69] C. J. Hughes and S. V. Adve, "A formal approach to frequent energy adaptations for multimedia applications," in Proceedings. 31st Annual International Symposium on Computer Architecture, 2004 [Online]. Available: <http://dx.doi.org/10.1145/1028176.1006713>.
- [70] A. Iyer and D. Marculescu, "Power aware microarchitecture resource scaling," in Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001 [Online]. Available: <http://dx.doi.org/10.1109/DATE.2001.915023>
- [71] A. Iyer and D. Marculescu, "Microarchitecture-level power management," IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 10, no. 3, pp. 230–239, Jun. 2002 [Online]. Available: <http://dx.doi.org/10.1109/TVLSI.2002.1043326>
- [72] M. Huang, J. Renau, Seung-Moon Yoo, and J. Torrellas, "A framework for dynamic energy efficiency and temperature management," in Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33, 2000 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2000.898071>
- [73] M. C. Huang, J. Renau, and J. Torrellas, "Positional adaptation of processors: application to energy reduction," in 30th Annual International Symposium on Computer Architecture, 2003. Proceedings. [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2003.1206997>
- [74] D. Ponomarev, G. Kucuk, and K. Ghose, "Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources," in Proceedings. 34th ACM/IEEE International Symposium on Microarchitecture. MICRO-34 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2001.991108>
- [75] D. H. Albonesi *et al.*, "Dynamically tuning processor resources with adaptive processing," Computer, vol. 36, no. 12, pp. 49–58, Dec. 2003 [Online]. Available: <http://dx.doi.org/10.1109/MC.2003.1250883>
- [76] S. Dropsho *et al.*, "Integrating adaptive on-chip storage structures for reduced dynamic power," in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2002 [Online]. Available: <http://dx.doi.org/10.1109/PACT.2002.1106013>
- [77] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," In Proceedings of the 2010 international conference on Power aware computing and systems (HotPower'10). USENIX Association, Berkeley, CA, USA, 2010.
- [78] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian, "Integrated power management for video streaming to mobile handheld devices," in Proceedings of the eleventh ACM international conference on Multimedia - MULTIMEDIA '03, 2003 [Online]. Available: <http://dx.doi.org/10.1145/957013.957134>
- [79] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03, 2003 [Online]. Available: <http://dx.doi.org/10.1145/945445.945460>
- [80] Y. Fei, L. Zhong, and N. K. Jha, "An energy-aware framework for coordinated dynamic software management in mobile computers," in The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, (MASCOTS 2004), 2004 [Online]. Available: <http://dx.doi.org/10.1109/MASCOT.2004.1348285>
- [81] F. Salehuddin, I. Ahmad, F.A. Hamid, A. Zaharim, A. Maheran, A. Hamid, P. S. Menon, H. A. Elgomati, and B. Y. Majlis, "Optimization of process parameter variation in 45nm p-channel MOSFET using L18 Orthogonal Array," In Proceedings of IEEE International Conference on Semiconductor Electronic (ICSE '12), 2012 [Online]. Available: <http://dx.doi.org/10.1109/SMElec.2012.6417127>.
- [82] W. Schemmert and G. Zimmer, "Threshold-voltage sensitivity of ion-implanted m.o.s. transistors due to process variations," Electronics Letters, vol. 10, no. 9, p. 151, 1974 [Online]. Available: <http://dx.doi.org/10.1049/el:19740115>
- [83] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie, "Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor," in 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, 2007 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2007.373412>
- [84] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling to improve system-level efficiency of the POWER7+," in Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2830772.2830824>

- [85] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004 [Online]. Available: <http://dx.doi.org/10.1109/TDSC.2004.2>
- [86] D. Gizopoulos and S. Mukherjee, "Guest Editors' Introduction: Special Section on Dependable Computer Architecture," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 3–4, Jan. 2011 [Online]. Available: <http://dx.doi.org/10.1109/TC.2011.8>
- [87] Y. Luo et al., "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014 [Online]. Available: <http://dx.doi.org/10.1109/DSN.2014.50>
- [88] L. T. Wang, C. Stroud, N. Toubia, "System-on-Chip test architectures: Nanometer design for testability", Elsevier, 2008 [Online]. Available: <http://dx.doi.org/10.1016/B978-0-12-373973-5.X5001-3>
- [89] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, 2009 [Online]. Available: <http://dx.doi.org/10.1145/1669112.1669126>
- [90] S. Raasch, A. Biswas, J. Stephan, P. Racunas, and J. Emer, "A fast and accurate analytical technique to compute the AVF of sequential bits in a processor," in *Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48*, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2830772.2830829>
- [91] B. Bentley, "Validating the intel pentium 4 microprocessor," in *Proceedings of the 38th conference on Design automation - DAC '01*, 2001 [Online]. Available: <http://dx.doi.org/10.1145/378239.378473>
- [92] K. Constantinides, O. Mutlu, and T. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2008.4771798>
- [93] Y.-C. Hsu, F. Tsai, W. Jong, and Y.-T. Chang, "Visibility enhancement for silicon debug," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006 [Online]. Available: <http://dx.doi.org/10.1109/DAC.2006.238670>
- [94] J. M. Rabaey, "Digital integrated circuits: a design perspective", Prentice-Hall, Inc., 1996.
- [95] D. Gizopoulos, "Advanced electronic testing: challenges and methodologies", Springer, 2006.
- [96] I. Wagner, *An Effective Verification Solution for Modern Microprocessors*. PhD thesis, University of Michigan, 2008.
- [97] A. Nahir, M. Dusanapudi, S. Kapoor, K. Reick, W. Roesner, K.-D. Schubert, K. Sharp and G. Wetli, "Post-Silicon Validation of the IBM POWER8 Processor," in *Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, 2014 [Online]. Available: <http://dx.doi.org/10.1145/2593069.2593183>
- [98] A. Adir, M. Golubev, S. Landa, A. Nahir, G. Shurek, V. Sokhin and A. Ziv, "Threadmill," in *Proceedings of the 48th Design Automation Conference on - DAC '11*, 2011 [Online]. Available: <http://dx.doi.org/10.1145/2024724.2024916>
- [99] J. Goodenough and R. Aitken, "Post-silicon is too late avoiding the \$50 million paperweight starts with validated designs," in *Proceedings of the 47th Design Automation Conference on - DAC '10*, 2010 [Online]. Available: <http://dx.doi.org/10.1145/1837274.1837279>
- [100] H. Rotithor, "Postsilicon validation methodology for microprocessors," *IEEE Design & Test of Computers*, vol. 17, no. 4, pp. 77–88, 2000 [Online]. Available: <http://dx.doi.org/10.1109/54.895008>
- [101] D. Josephson, "The good, the bad, and the ugly of silicon debug," in *Proceedings of the 43rd annual conference on Design automation - DAC '06*, 2006 [Online]. Available: <http://dx.doi.org/10.1145/1146909.1146915>
- [102] A. Adir, R. Emek, Y. Katz, and A. Koyfman, "DeepTrans - a model-based approach to functional verification of address translation mechanisms," in *Proceedings. 4th International Workshop on Microprocessor Test and Verification - Common Challenges and Solutions* [Online]. Available: <http://dx.doi.org/10.1109/MTV.2003.1250255>
- [103] A. Adir, L. Fournier, Y. Katz, and A. Koyfman, "DeepTrans - Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms," in *2006 IEEE International High-Level Design Validation and Test Workshop*, 2006 [Online]. Available: <http://dx.doi.org/10.1109/HLDVT.2006.319971>

- [104] Y. A. Katz, A. Koyfman and E. Tsanko, "Method of Verification of Address Translation Mechanisms", US Patent No. 8245164.
- [105] A. Koyfman, A. Adir, R. Emek, Y. Katz and M. Vinov, "Modeling Language and Method for Address Translation Design Mechanisms in Test Generation", US Patent No. 7370296.
- [106] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov and A. Ziv, "Genesys-pro: innovations in test program generation for functional processor verification," IEEE Design & Test of Computers, vol. 21, no. 2, pp. 84–93, Mar. 2004 [Online]. Available: <http://dx.doi.org/10.1109/MDT.2004.1277900>
- [107] AMD. Revision Guide for AMD Athlon64 and AMD Opteron Processors. Publication 25759, Revision 3.79, July 2009.
- [108] AMD. Revision Guide for AMD Family 10h Processors. Technical Report 41322, Revision 3.92, March 2012.
- [109] IBM PowerPC 750FX and 750FL RISC Microprocessor Errata List DD2.X, version 1.4, Feb. 2008.
- [110] Intel Corporation. Intel Core Duo Processor and Intel Core Solo Processor on 65nm Process Specification Update. Technical Report 309222-020, June 2009.
- [111] IBM. IBM PowerPC 750GX and 750GL RISC Microprocessor Errata Notice DD1.X, version 1.8, Jan. 2006.
- [112] Intel 64 and IA-32 Architectures Software Developer's Manual, Order Number: 325462-053US, January 2015.
- [113] Intel Corporation. Intel® Xeon® Processor 5100 Series Specification Update. Document Number 313356-026, Dec. 2012.
- [114] Intel Corporation. Intel® Xeon® Processor 3400 Series Specification Update. Document Number 322373-009, May 2010.
- [115] Intel Corporation. Intel® Xeon® Processor E3-1200 v3 Series Specification Update. Document Number 328908-011, April 2015.
- [116] Intel Corporation. Intel® Pentium® 4 Processor Specification Update. Document Number 249199-071, Aug. 2008.
- [117] T. Hong, Y. Li, S. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner and S. Mitra "QED: Quick Error Detection tests for effective post-silicon validation," in 2010 IEEE International Test Conference, 2010 [Online]. Available: <http://dx.doi.org/10.1109/TEST.2010.5699215>
- [118] D. Lin, T. Hong, F. Fallah, N. Hakim, and S. Mitra, "Quick detection of difficult bugs for effective post-silicon validation," in Proceedings of the 49th Annual Design Automation Conference on - DAC '12, 2012 [Online]. Available: <http://dx.doi.org/10.1145/2228360.2228461>
- [119] S. Ray, J. Yang, A. Basak, and S. Bhunia, "Correctness and security at odds," in Proceedings of the 52nd Annual Design Automation Conference on - DAC '15, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2744769.2754896>
- [120] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and D. A. Wood, "The gem5 simulator," ACM SIGARCH Computer Architecture News, vol. 39, no. 2, p. 1, Aug. 2011 [Online]. Available: <http://dx.doi.org/10.1145/2024716.2024718>
- [121] A. Bhattacharjee, "Large-reach memory management unit caches," in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46, 2013 [Online]. Available: <http://dx.doi.org/10.1145/2540708.2540741>
- [122] T. W. Barr, A. L. Cox, and S. Rixner, "Translation caching," in Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10, 2010 [Online]. Available: <http://dx.doi.org/10.1145/1815961.1815970>
- [123] D. Chatterjee, A. Koyfman, R. Morad, A. Ziv, and V. Bertacco, "Checking architectural outputs instruction-by-instruction on acceleration platforms," in Proceedings of the 49th Annual Design Automation Conference on - DAC '12, 2012 [Online]. Available: <http://dx.doi.org/10.1145/2228360.2228531>
- [124] C.-H. Hsu, D. Chatterjee, R. Morad, R. Ga, and V. Bertacco, "ArChiVED: Architectural checking via event digests for high performance validation," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, 2014 [Online]. Available: <http://dx.doi.org/10.7873/DATE.2014.330>
- [125] AMD Revision Guide for AMD Family 11h Processors, Rev. 3.00, 41788, July 2008.

- [126] AMD Revision Guide for AMD Athlon™ 64 and AMD Opteron™ Processors, 25759, Rev. 3.79, July 2009.
- [127] AMD Revision Guide for AMD Family 11h Processors, Rev. 3.10, 41788, December 2011.
- [128] 5th Generation Intel® Core™ Processor Family, Intel® Core™ M Processor Family, Mobile Intel® Pentium® Processor Family, and Mobile Intel® Celeron® Processor Family, Specification Update, October 2015, Revision 015, Reference Number 330836-015.
- [129] Mobile 4th Generation Intel® Core™ Processor Family, Mobile Intel® Pentium® Processor Family, and Mobile Intel® Celeron® Processor Family, Specification Update, November 2015, Revision 028, Reference Number 328903-028.
- [130] Intel® Xeon® Processor E3-1200 v3 Product Family Specification Update January 2016, Reference Number 328908-014US.
- [131] Intel® Xeon® Processor E7-8800/4800 v3 Product Family, Specification Update, March 2016, Reference Number 332317-007US.
- [132] Intel® Core™ Duo Processor and Intel® Core™ Solo Processor on 65 nm Process Specification Update June 2009 Revision 020.
- [133] S. Ray, J. Yang, A. Basak, and S. Bhunia, "Correctness and security at odds," in Proceedings of the 52nd Annual Design Automation Conference on - DAC '15, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2744769.2754896>
- [134] N. Foutris, D. Gizopoulos, J. Kalamatianos, and V. Sridharan, "Assessing the impact of hard faults in performance components of modern microprocessors," in 2013 IEEE 31st International Conference on Computer Design (ICCD), 2013 [Online]. Available: <http://dx.doi.org/10.1109/ICCD.2013.6657044>
- [135] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538) [Online]. Available: <http://dx.doi.org/10.1109/WWC.2001.990739>
- [136] The Linux Kernel Documentation (Parent Directory), Retrieved 2017 from <https://www.kernel.org/doc/Documentation>.
- [137] K. L. Shepard, V. Narayanan, and R. Rose, "Harmony: static noise analysis of deep submicron digital integrated circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 8, pp. 1132–1150, 1999 [Online]. Available: <http://dx.doi.org/10.1109/43.775633>
- [138] N. H. E. Weste and D. M. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective," 4<sup>th</sup> Ed., Addison-Wesley, 2010.
- [139] K. Bernstein, K. Carrig, C. Durham, P. Hansen, D. Hogenmiller, E. Nowak, and N. Roher, High Speed CMOS Design Styles. Springer US, 1999 [Online]. Available: <http://dx.doi.org/10.1007/978-1-4615-5573-5>
- [140] R. Rao, A. Srivastava, D. Blaauw, and D. Sylvester, "Statistical estimation of leakage current considering inter- and intra-die process variation," in Proceedings of the 2003 international symposium on Low power electronics and design - ISLPED '03, 2003 [Online]. Available: <http://dx.doi.org/10.1145/871506.871530>
- [141] J. L. Henning, "SPEC CPU2006 benchmark descriptions," ACM SIGARCH Computer Architecture News, vol. 34, no. 4, pp. 1–17, Sep. 2006 [Online]. Available: <http://dx.doi.org/10.1145/1186736.1186737>
- [142] R. J. Riedlinger *et al.*, "A 32nm 3.1 billion transistor 12-wide-issue Itanium® processor for mission-critical servers," in 2011 IEEE International Solid-State Circuits Conference, 2011 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2011.5746230>
- [143] V. J. Reddi, M. S. Gupta, G. Holloway, G.-Y. Wei, M. D. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in 2009 IEEE 15th International Symposium on High Performance Computer Architecture, 2009 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2009.4798233>
- [144] Perf: Linux Profiling with Performance Counters. Retrieved 2017 from [https://perf.wiki.kernel.org/index.php/Main\\_Page](https://perf.wiki.kernel.org/index.php/Main_Page).
- [145] S. Sundaram *et al.*, "Adaptive Voltage Frequency Scaling Using Critical Path Accumulator Implemented in 28nm CPU," in 2016 29th International Conference on VLSI Design and 2016 15th

- International Conference on Embedded Systems (VLSID), 2016 [Online]. Available: <http://dx.doi.org/10.1109/VLSID.2016.106>
- [146] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," in 2008 International Symposium on Computer Architecture, 2008 [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2008.22>
  - [147] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42, 2009 [Online]. Available: <http://dx.doi.org/10.1145/1669112.1669126>
  - [148] H. Duwe, X. Jian, D. Petrisko, and R. Kumar, "Rescuing Uncorrectable Fault Patterns in On-Chip Memories through Error Pattern Transformation," in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016 [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2016.61>
  - [149] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies," in Proceedings of the 2007 international symposium on Low power electronics and design - ISLPED '07, 2007 [Online]. Available: <http://dx.doi.org/10.1145/1283780.1283808>
  - [150] R. Franch, P. Restle, N. James, W. Huott, J. Friedrich, R. Dixon, S. Weitzel, K. Van Goor, and G. Salem, "On-chip Timing Uncertainty Measurements on IBM Microprocessors," in 2008 IEEE International Test Conference, 2008 [Online]. Available: <http://dx.doi.org/10.1109/TEST.2008.4700707>
  - [151] Phillip J. Restle, Robert L. Franch, Norman K. James, William V. Huott, Timothy M. Skergan, Steven C. Wilson, Nicole S. Schwartz, Joachim G. Clabes. "Timing uncertainty measurements on the Power5 microprocessor," in 2004 IEEE International Solid-State Circuits Conference (IEEE Cat. No.04CH37519) [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2004.1332740>
  - [152] R. Bertran *et al.*, "Voltage Noise in Multi-Core Processors: Empirical Characterization and Optimization Opportunities," in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2014.12>
  - [153] S. Herbert and D. Marculescu, "Variation-aware dynamic voltage/frequency scaling," in 2009 IEEE 15th International Symposium on High Performance Computer Architecture, 2009 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2009.4798265>
  - [154] N. Kapadia and S. Pasricha, "VARSHA: Variation and Reliability-Aware Application Scheduling with Adaptive Parallelism in the Dark-Silicon Era," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, 2015 [Online]. Available: <http://dx.doi.org/10.7873/DATE.2015.0454>
  - [155] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-Picking: Exploiting Process Variations in Dark-Silicon Homogeneous Chip Multi-Processors," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013, 2013 [Online]. Available: <http://dx.doi.org/10.7873/DATE.2013.023>
  - [156] R. Teodorescu and J. Torrellas, "Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors," in 2008 International Symposium on Computer Architecture, 2008 [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2008.40>
  - [157] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, C. Magdalinos, and D. Gizopoulos, "Voltage margins identification on commercial x86-64 multicore microprocessors," in 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS), 2017 [Online]. Available: <http://dx.doi.org/10.1109/IOLTS.2017.8046198>
  - [158] A. Bacha, and R. Teodorescu, "Authenticache: Harnessing cache ECC for system authentication," in Proceedings of the 48th International Symposium on Microarchitecture - MICRO-48, 2015 [Online]. Available: <http://dx.doi.org/10.1145/2830772.2830814>
  - [159] K. Ganesan, J. Jo, W. L. Bircher, D. Kaseridis, Z. Yu, and L. K. John, "System-level max power (SYMPO)," in Proceedings of the 19th international conference on Parallel architectures and compilation techniques - PACT '10, 2010 [Online]. Available: <http://dx.doi.org/10.1145/1854273.1854282>
  - [160] K. Ganesan and L. K. John, "MAximum Multicore POwer (MAMPO)," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11, 2011 [Online]. Available: <http://dx.doi.org/10.1145/2063384.2063455>
  - [161] Mersenne Prime Search Software, Prime95, Version 28.10, <https://www.mersenne.org/download/>
  - [162] Stress-ng benchmark. Retrieved July 30, 2017 <http://kernel.ubuntu.com/~cking/stress-ng/>

- [163] G. Theodorou, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Software-Based Self-Test for Small Caches in Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1991–2004, Dec. 2014 [Online]. Available: <http://dx.doi.org/10.1109/TCAD.2014.2363387>
- [164] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda, "Microprocessor Software-Based Self-Testing," *IEEE Design & Test of Computers*, vol. 27, no. 3, pp. 4–19, May 2010 [Online]. Available: <http://dx.doi.org/10.1109/MDT.2010.5>
- [165] R. E. Kessler and M. D. Hill, "Page placement algorithms for large real-indexed caches," *ACM Transactions on Computer Systems*, vol. 10, no. 4, pp. 338–359, Nov. 1992 [Online]. Available: <http://dx.doi.org/10.1145/138873.138876>
- [166] ARM Cortex-A Series: Programmer's Guide for ARMv8-A, v1.0, March '15. [http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A\\_v8\\_architecture\\_PG.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf)
- [167] S. Browne, C. Deane, G. Ho, P. Mucci, "PAPI: A Portable Interface to Hardware Performance Counters," *Proceedings of Department of Defense HPCMP Users Group Conference*, June, 1999.
- [168] H. David, E. Gorbato, U. R. Hanebutte, R. Khanaa, and C. Le, "RAPL," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design - ISLPED '10*, 2010 [Online]. Available: <http://dx.doi.org/10.1145/1840845.1840883>
- [169] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th annual international symposium on Computer architecture - ISCA '07*, 2007 [Online]. Available: <http://dx.doi.org/10.1145/1250662.1250665>
- [170] L. Gwennap, *Performance Arms X-Gene 3 for Cloud*, 2017 (Accessed: July 25, 2018). <http://www.linleygroup.com/uploads/x-gene-3-for-cloud.pdf>
- [171] NAS Parallel Benchmarks Suite, v3.3.1. <https://www.nas.nasa.gov/publications/npb.html>.
- [172] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, 2008 [Online]. Available: <http://dx.doi.org/10.1145/1454115.1454128>
- [173] B. Lepers, V. Quema, and A. Fedorova, "Thread and memory placement on numa systems: Asymmetry matters," in *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '15*, (Berkeley, CA, USA), pp. 277–289, USENIX Association, 2015.
- [174] M. Curtis-Maury, *Improving the Efficiency of Parallel Applications on Multithreaded and Multicore Systems*. PhD thesis, Virginia Tech, 2008.
- [175] *Advanced Configuration and Power Interface (ACPI) Specification*, 2014 (Accessed: Aug 1, 2018). [https://www.uefi.org/sites/default/files/resources/ACPI\\_5\\_1release.pdf](https://www.uefi.org/sites/default/files/resources/ACPI_5_1release.pdf)
- [176] G. Karakonstantis, K. Tovletoglou, L. Mukhanov, H. Vandierendonck, D. S. Nikolopoulos, P. Lawthers, P. Koutsovasilis, M. Maroudas, C. D. Antonopoulos, C. Kalogirou, N. Bellas, S. Lalis, S. Venugopal, A. Prat-Perez, A. Lampropoulos, M. Kleanthous, A. Diavastos, Z. Hadjilambrou, P. Nikolaou, Y. Sazeides, P. Trancoso, G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, and S. Das, "An energy-efficient and error-resilient server ecosystem exceeding conservative scaling limits," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018 [Online]. Available: <http://dx.doi.org/10.23919/DATE.2018.8342175>
- [177] M. Kaliorakis, A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 109–112, Jul. 2018 [Online]. Available: <http://dx.doi.org/10.1109/LCA.2018.2798604>
- [178] C. Isci, G. Contreras, and M. Martonosi, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, 2006 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2006.30>
- [179] Q. Wu *et al.*, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," in *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)* [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2005.7>
- [180] H. Sasaki, A. Buyuktosunoglu, A. Vega, and P. Bose, "Characterization and mitigation of power contention across multiprogrammed workloads," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2016 [Online]. Available: <http://dx.doi.org/10.1109/IISWC.2016.7581266>
- [181] D. M. Brooks *et al.*, "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000 [Online]. Available: <http://dx.doi.org/10.1109/40.888701>



- [182] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based GPU design space exploration," in 2012 IEEE International Symposium on Performance Analysis of Systems & Software, 2012 [Online]. Available: <http://dx.doi.org/10.1109/ISPASS.2012.6189201>
- [183] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and Use of Linear Regression Models for Processor Performance Analysis," in The Twelfth International Symposium on High-Performance Computer Architecture, 2006. [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2006.1598116>
- [184] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in Proceedings of the 12th international conference on Architectural support for programming languages and operating systems - ASPLOS-XII, 2006 [Online]. Available: <http://dx.doi.org/10.1145/1168857.1168881>
- [185] A. Biswas, N. Soundararajan, S. S. Mukherjee, and S. Gurumurthi, "Quantized AVF: A means of capturing vulnerability variations over small windows of time," in Silicon Errors in Logic - System Effects (SELSE), 2009.
- [186] K. Tovletoglou, L. Mukhanov, G. Karakonstantis, A. Chatzidimitriou, G. Papadimitriou, M. Kaliorakis, D. Gizopoulos, Z. Hadjilambrou, Y. Sazeides, A. Lampropoulos, S. Das, and P. Vo, "Measuring and Exploiting Guardbands of Server-Grade ARMv8 CPU Cores and DRAMs," in 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2018 [Online]. Available: <http://dx.doi.org/10.1109/DSN-W.2018.00013>
- [187] M. Ketkar and E. Chiprout, "A microarchitecture-based framework for pre- and post-silicon power delivery analysis," in Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42, 2009 [Online]. Available: <http://dx.doi.org/10.1145/1669112.1669136>
- [188] Y. Kim and L. K. John, "Automated di/dt stressmark generation for microprocessor power delivery networks," in IEEE/ACM International Symposium on Low Power Electronics and Design, 2011 [Online]. Available: <http://dx.doi.org/10.1109/ISLPED.2011.5993645>
- [189] Y. Kim et al., "AUDIT: Stress Testing the Automatic Way," in 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, 2012 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2012.28>
- [190] M. S. Gupta, V. J. Reddi, G. Holloway, Gu-Yeon Wei, and D. M. Brooks, "An event-guided approach to reducing voltage noise in processors," in 2009 Design, Automation & Test in Europe Conference & Exhibition, 2009 [Online]. Available: <http://dx.doi.org/10.1109/DATE.2009.5090651>
- [191] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2003.1183526>
- [192] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in 2012 39th Annual International Symposium on Computer Architecture (ISCA), 2012 [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2012.6237022>
- [193] M. D. Powell 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, 2003. ISLPED '03. [Online]. Available: <http://dx.doi.org/10.1109/LPE.2003.1231866>
- [194] J. Leng, Y. Zu, and V. J. Reddi, "GPU voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in GPU architectures," in 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), 2015 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2015.7056030>
- [195] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu, "Core tunneling: Variation-aware voltage noise mitigation in GPUs," in 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2016.7446061>
- [196] R. R. Thomas, N. Sedaghati, and R. Teodorescu, "EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures," in 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016 [Online]. Available: <http://dx.doi.org/10.1109/ISPASS.2016.7482076>
- [197] M. S. Gupta, K. K. Rangan, M. D. Smith, Gu-Yeon Wei, and D. Brooks, "DeCoR: A Delayed Commit and Rollback mechanism for handling inductive noise in processors," in 2008 IEEE 14th International

- Symposium on High Performance Computer Architecture, 2008 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2008.4658654>
- [198] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. M. Bull, "14.6 An all-digital power-delivery monitor for analysis of a 28nm dual-core ARM Cortex-A57 cluster," in 2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers, 2015 [Online]. Available: <http://dx.doi.org/10.1109/ISSCC.2015.7063026>
- [199] P. N. Whatmough, S. Das, and D. M. Bull, "Analysis of adaptive clocking technique for resonant supply voltage noise mitigation," in 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2015 [Online]. Available: <http://dx.doi.org/10.1109/ISLPED.2015.7273502>
- [200] S. Das, P. Whatmough, and D. Bull, "Modeling and characterization of the system-level Power Delivery Network for a dual-core ARM Cortex-A57 cluster in 28nm CMOS," in 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2015 [Online]. Available: <http://dx.doi.org/10.1109/ISLPED.2015.7273505>
- [201] P. N. Whatmough, S. Das, Z. Hadjilambrou, and D. 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, vol. 52, no. 6, pp. 1643–1654, Jun. 2017 [Online]. Available: <http://dx.doi.org/10.1109/JSSC.2017.2669025>
- [202] V. Wool, EAS – Energy Aware Scheduler: An unbiased look, (Accessed: Oct 11, 2018). <https://elinux.org/images/6/69/Eas-unbiased1.pdf>.
- [203] J. Li and J. F. Martinez, "Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors," in The Twelfth International Symposium on High-Performance Computer Architecture, 2006. [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2006.1598114>
- [204] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), 2006 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2006.8>
- [205] A. Vega, A. Buyuktosunoglu, H. Hanson, P. Bose, and S. Ramani, "Crank it up or dial it down," in Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46, 2013 [Online]. Available: <http://dx.doi.org/10.1145/2540708.2540727>
- [206] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting Performance Impact of DVFS for Realistic Memory Systems," in 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, 2012 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2012.23>
- [207] E. Castillo *et al.*, "CATA: Criticality Aware Task Acceleration for Multicore Processors," in 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016 [Online]. Available: <http://dx.doi.org/10.1109/IPDPS.2016.49>
- [208] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos, "Differential Fault Injection on Microarchitectural Simulators," in 2015 IEEE International Symposium on Workload Characterization, 2015 [Online]. Available: <http://dx.doi.org/10.1109/IISWC.2015.28>
- [209] B. Jacob and T. Mudge, "Virtual memory: issues of implementation," Computer, vol. 31, no. 6, pp. 33–43, Jun. 1998 [Online]. Available: <http://dx.doi.org/10.1109/2.683005>
- [210] IBM PowerPC® Microprocessor Family: The Programming Environments Manual for 64-bit Microprocessors v3.0, 2005.
- [211] IBM Power ISA™ v2.07, 2013.
- [212] N. Foutris, D. Gizopoulos, M. Psarakis, X. Vera, and A. Gonzalez, "Accelerating microprocessor silicon validation by exposing ISA diversity," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11, 2011 [Online]. Available: <http://dx.doi.org/10.1145/2155620.2155666>
- [213] N. Foutris, D. Gizopoulos, X. Vera, and A. Gonzalez, "Deconfigurable microprocessor architectures for silicon debug acceleration," in Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13, 2013 [Online]. Available: <http://dx.doi.org/10.1145/2485922.2485976>
- [214] A. Adir *et al.*, "Genesys-pro: innovations in test program generation for functional processor verification," IEEE Design & Test of Computers, vol. 21, no. 2, pp. 84–93, Mar. 2004 [Online]. Available: <http://dx.doi.org/10.1109/MDT.2004.1277900>
- [215] F. Haque, J. Michelson, and K. Khan, "The Art of Verification with Vera", Verification Central, 2001.

- [216] M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, and M. Vinov, "Industrial experience with test generation languages for processor verification," in Proceedings of the 41st annual conference on Design automation - DAC '04, 2004 [Online]. Available: <http://dx.doi.org/10.1145/996566.996578>
- [217] L. Fournier, Y. Arbetman, and M. Levinger, "Functional verification methodology for microprocessors using the Genesys test-program generator. Application to the x86 microprocessors family," in Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078) [Online]. Available: <http://dx.doi.org/10.1109/DATE.1999.761162>
- [218] Y. Zu, W. Huang, I. Paul, and V. J. Reddi, "Ti-states: Processor power management in the temperature inversion region," in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016 [Online]. Available: <http://dx.doi.org/10.1109/MICRO.2016.7783758>
- [219] I. Wagner and V. Bertacco, "Reversi: Post-silicon validation system for modern microprocessors," in 2008 IEEE International Conference on Computer Design, 2008 [Online]. Available: <http://dx.doi.org/10.1109/ICCD.2008.4751878>
- [220] A. DeOrio, I. Wagner, and V. Bertacco, "Dacota: Post-silicon validation of the memory subsystem in multi-core designs," in 2009 IEEE 15th International Symposium on High Performance Computer Architecture, 2009 [Online]. Available: <http://dx.doi.org/10.1109/HPCA.2009.4798278>
- [221] S. Mukherjee, "Architecture Design for Soft Errors (1st ed.)", Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
- [222] I. Wagner, V. Bertacco, "Post-Silicon and Runtime Verification for Modern Processors (1st ed.)", Springer Publishing Company, Incorporated, 2010.