The Dissertation Committee for Yazhou Zu
certifies that this is the approved version of the following dissertation:

Active Timing Margin Management to Improve Microprocessor Power Efficiency

Committee:

Vijay Janapa Reddi, Supervisor

Charles R. Lefurgy

Mattan Erez

Lizy K. John

Andreas Gerstlauer

Active Timing Margin Management to Improve Microprocessor Power Efficiency

by

Yazhou Zu

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2018

Dedicated to my loving mother, Zhijie Li

Acknowledgments

As I march towards the end of my doctoral study, I feel more connected to the intriguing campus of The University of Texas at Austin. The past five years at UT is life-changing. Five years in the twenties may be the most precious period of time in a man's life, and I really appreciate the happiness as well as the pains of exploring frontier computer architecture knowledge during this period of time in the lovely city of Austin, TX.

In all cultures around the globe, it is an honorable pursuit for a person to choose truth and knowledge as his/her lifelong career. As a humble young man who just graduated from college, I also had the unsullied dream of being a scholar who will be of use the mankind when I embarked on my Ph.D. journey five years ago. Looking back at this point, I can proudly say it was a truly a tremendously valuable investment that I dedicated myself into UT's rigorous Ph.D. program. Texas is a young place, where the spirit of cowboys striving to survive the wild and to fight for justice is still retained. I am lucky enough to receive an honest and rigorous academic training at the University of Texas, like the immaculate virtue of a newborn child. I hope in the rest of my life I will retain some of the qualities I learned during my Ph.D. study at UT Austin, including the ambition of purse truth and justness, and the habit of being conscientious and honest.

I wish to thank the multitudes of people who have helped me on my Ph.D. journey. First and foremost, I want to give my sincere gratitude to my advisor, professor Vijay Janapa Reddi, who guided me to be a professional computer architect from an immature fresh college graduate. Five years ago, I came to UT as a student primarily trained to develop integrated circuit at Shanghai Jiao Tong University, with the hope of finding an advisor who would help me with my Ph.D. funding, and who would help me find a promising research topic that can lift my system layer up to architecture design, from the low-level circuit implementation. Vijay offered me a precious opportunity the first week I came to Austin. We worked very hard together during my first year at UT, which set a good foundation for my upcoming Ph.D. experience. Vijay is a very responsible mentor, he works hard to make sure all student's Ph.D. research is properly funded so that we students can focus on our work whole-heartedly. Vijay also helps me, as well as other students, find various internship opportunities so that we can witness how the real world works and learn to transition into the professional lifestyle from a student role. I deeply appreciate all the financial and career help Vijay provided me with.

The journey of completing a Ph.D. is not easy. As my mentor, Vijay patiently taught me how to choose research areas that are of value in the future, and how to produce high-quality research papers. His vision, and determination to achieve things inspires me a lot. Along with my Ph.D. journey, I learned that scientific research takes a lot of perseverance. It involves constantly overthrowing our own conjectures, trying out different ideas, and

eventually making things work. In this process, Vijay would always find me the correct people to connect with. I still remember in my first independent research project Vijay repeatedly drove me to IBM research to discuss with Charles when we encountered obstacles. It is in those experiences when I realized that intelligent discussions are key to innovation because it is when one combines knowledge from different areas that inspiration occurs. This idea gradually shaped my mindset beyond just becoming a better technical developer, but to become a more active communicator, which I will forever benefit from.

I want to give special thanks to Charles R. Lefurgy, who played a critical role in my Ph.D. study. My research topic and methodology involves comprehensive hardware measurement and instrumentation, which is very challenging for fresh graduates who have no experience with complex production microprocessors. In my first research projects, Charles patiently guided me on how to get the correct measurement done, how to interpret the data, and where the problem is. I still remember those long phone discussion with Charles in my second year at UT. I remember that when we were submitting my first paper to MICRO in 2015, Charles sat with me late in the night in my cube at the P.O.B building, and we both hadn't had dinner. His humor helped me relieve the pressure of editing the paper, and he patiently helped me pick up the wording and grammar mistakes in my draft. Without Charles's help, we couldn't imagine how I could have initiated my own research, no to mention how I could have progressed towards the finish line. Charles is a wonderful,

nice senior to work with. He loves his research fruition and likes mentoring young students even though he is very busy with his own work at IBM research. Charles sets a good example for me on how to be a nice college in my future career. I feel very fortunate to have the pleasure of working with him in my Ph.D. study.

I would also like to thank all other members of my dissertation committee: Mattan Erez, Lizy John, and Andreas Gerstlauer. Their contentiousness, keen insights, and feedbacks are absolutely invaluable in completing my Ph.D. thesis. In addition, Lizy taught me Computer Performance Evaluation and Benchmarking, and Mattan taught me Computer Architecture Parallelism and Locality; both helped me form a solid foundational computer architecture knowledge. Mattan is a smart, positive, and easy-going mentor who I admire. Andreas makes me realize never feel self-contained and always do things thoroughly when he helped me refine my thesis. I truly appreciate the rigor Andreas possesses. Lizy is always nice and helpful, special thanks to her for serving on my committee. The courses these committee members provide, and the way they mentor their own students make me realize how a top-notch graduate school program functions at UT.

During my time at UT, I also had the privilege to interact with other faculty members, in particular, professor Yale Patt. I had the fortune of following Yale in his micro-architecture course and witnessing how a highly respected scholar emphasizes on teaching, on nurturing seed-like ideas in research, and on keeping an eye on every detail in all the work.

I had four internships during my Ph.D. study, and I want to thank all my mentors during each internship. Sek Chai was my mentor at SRI International. I thank him for opening my eyes to the industry for the first time. Wei Huang and Indrani Paul were my mentors during my part-time internships at AMD Research. I really appreciate the help and convenience they provided me with during my research project here. We together won the IEEE MICRO top picks award for our joint work there, and the work I did with them served as a critical milestone in my Ph.D. journey. Alper Buyuktosunoglu was my mentor during my internship at IBM Thomas.J.Watson research center. He and his colleague's experience and sharp insights into computer architecture truly opened my eyes. Bin Li was my mentor when I interned at Facebook, I thank him for his warmheartedness when providing job-seeking advice for me as I approach my graduation point.

My friends and lab mates at UT Austin are great supporters of mine that let me go through all the setback and challenges in research, and in life. Words cannot describe my gratitude to them. I would like to give special thanks Jingwen Leng, who was like a big brother to me when I first started working with Vijay. Jingwen helped me understand what a Ph.D. program is like at UT ECE and what it means to be a Ph.D. student before I came to Austin. He kept encouraging me when I fell into setbacks and gave me a lot of hands-on advice. I thank him for introducing me to Vijay, without which I couldn't have gone through this invaluable experience. Jingwen deeply influenced me on how to be a helpful and responsible senior student for upcoming

new students, and I always look up to him. I enjoyed a lot of intelligent discussion with Yuhao, not only on computer science research but also in many other areas of life. Yuhao is a clever guy with a lot of curiosity, best of wishes to him on his academic career. I had a good time with Wenzhi Cui and Daniel Richins in the later stage of my study. Wenzhi brought new angles of looking at problems to me, and Daniel helped me edit my last research paper patiently.

I want to express my deepest appreciation to my friends Zhengcheng Tao, Yuanqi Chen, Xiaojun Lin, Jichao Chen, Chuang Wang, as well as those anonymous friends who gave me hope and encouragement over the phone when I was at the bottom. You are all very nice people with warm hearts, and we've had a lot of fun together in our spare time. Wish you all the best wherever you go.

Most importantly, not a single word of this dissertation would happen without the nurture of my loving mother, Zhijie Li. Ph.D. is not only an academic experience but also a life journey where one experience failures and life difficulties. In this process, I gradually grew up and realized the difficulty of raising up a child. In addition, my mother's perseverance of continuing pursuing her literature dream after retirement has truly inspired me. I cannot express how amazed I am when I read the poems she wrote in her spare time. I appreciate my mother not only as a son, but as a researcher and scholar who values knowledge, beauty, and truth. It is easy for people to enjoy the pleasure of life without the pressure of work, yet my mother chooses to read, absorb, and write, simply chasing her youth dream. My mother is a lovely

and wonderful human being, and I am very proud of her. May she continue to have a happy life.

Active Timing Margin Management to Improve Microprocessor Power Efficiency

Publication No. _____

Yazhou Zu, Ph.D.
The University of Texas at Austin, 2018

Supervisor: Vijay Janapa Reddi

Improving power/performance efficiency is critical for today's microprocessors. From edge devices to datacenters, lower power or higher performance always produces better systems, measured by lower cost of ownership or longer battery time. This thesis studies improving microprocessor power/performance efficiency by optimizing the pipeline timing margin. In particular, this thesis focuses on improving the efficacy of *Active Timing Margin*, a young technology that dynamically adjusts the margin.

Active timing margin trims down the pipeline timing margin with a control loop that adjusts voltage and frequency based on real-time chip environment monitoring. The key insight of this thesis is that in order to maximize active timing margin's efficiency enhancement benefits, synergistic management from processor architecture design and system software scheduling are

needed. To that end, this thesis covers the major consumers of pipeline timing margin, including temperature, voltage, and process variation. For temperature variation, the thesis proposes a table-lookup based active timing margin mechanism, and an associated temperature management scheme to minimize power consumption. For voltage variation, the thesis characterizes the limiting factors of adaptive clocking's power saving and proposes application scheduling to maximize total system power reduction. For process variation, the thesis proposes core-level adaptive clocking reconfiguration to automatically expose inter-core variation and discusses workload scheduling and throttling management to control critical application performance.

The author believes the optimization presented in this thesis can potentially benefit a variety of processor architectures as the conclusions are based on the solid measurement on state-of-the-art processors, and the research objective, active timing margin, already has wide applicability in the latest microprocessors by the time this thesis is written.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Power efficiency is critical in designing and operating modern microprocessors. In the early 2000s, microprocessor designers have put low power as a first-order concern in designing digital integrated circuits [76, 80], pointing out that the power density of a microprocessor could exceed that of a nuclear reactor should the clock rate continues the conventional Dennard Scaling [81], making heat dissipation a key constraint for estimating total chip power budget. More recently, as the information ecosystem has shifted towards the cloud-edge paradigm, power efficiency has been classified as a key design constraint by researchers on both ends. For datacenters, higher hardware power efficiency entails a lower total cost of ownership (TCO), which increases the profit margin of an enterprise [3]. For mobile devices, higher efficiency prolongs battery life, which improves user satisfaction [110]. Therefore, improving microprocessor power efficiency is an important goal for computer architecture researchers today.

While there is significant motivation to improve microprocessor power efficiency, the end of Dennard scaling and Moore's law [29, 98] have made our design arsenal increasingly limited to achieve this goal. Specifically, it has

become increasingly difficult to continue pushing more active cores onto the silicon real estate and relying on parallelism for improving performance while keeping power under control, as is the common practice in the past 10 years, leaving aside the difficulty of programming parallel software. As techniques are being exhausted to optimize conventional general purpose hardware, many researchers have turned to application-specific proposals, notably customized hardware design (i.e. accelerators) to reduce the power wastage of data movement and instruction decoding in general purpose chips [79]. While this approach has proven to be successful in key emerging applications domains, e.g., machine learning [17, 49, 16, 22], its design, verification, and manufacturing costs are, however, non-negligible in the fast development cycle of today's technology world. As a result, alternative proposals with lower overhead, wider applicability, and practical power efficiency gains are still highly desirable for chip vendors as well as consumers.

This thesis seeks to improve microprocessor power efficiency by optimizing pipeline's timing margin, a long-neglected, but possibly one of the last opportunities where processor efficiency can yet be improved. The significance of this thesis topic is based on three findings. First, over 10% power or performance gain can be achieved simply by squeezing down modern microprocessor's existing timing margin, based on real hardware measurement on production CPUs and GPUs [89, 56], which proves practical benefit can be realized by working on the excess timing margin. Second, timing margin exists for all processor architectures, from CPUs [88, 89] to GPUs [58, 57, 56], and

inevitably in the upcoming accelerators. This is because all pipelined architectures need to prevent timing failure, caused by chip environment changes, such as unusual temperature, voltage noise, or process corners. Thus, some amount of margin is always required, indicating the opportunity from timing margin is pervasive. Third, to date, there is no comprehensive thesis that systematically studies a practical solution that can effectively reduce timing margin in production processors, and hence the implication for the whole computing system layers are yet to be discovered.

Specifically, this thesis focuses on studying the system-level implication of Active Timing Margin, a young technique proposed to reduce timing margin. Active timing margin uses a hardware loop to adjust chip supply voltage and frequency based on real-time monitored load environment. It has been tested rigorously on various production processors to pass load environment corner cases [54, 14, 100, 35, 13, 104, 103, 31]. Compared with other proposals that try to trim down the timing margin [36, 28, 78, 39, 38, 88, 89, 69, 56, 74], active timing margin's low implementation overhead and execution correctness guarantee make it the more favorable design solution. Thus, understanding how active timing margin behaves and saves timing margin in the field, and trying to maximize its efficiency gain with appropriate management has practical impact for designing future computing systems.

Thesis Statement Active timing margins full efficiency gain can only be unlocked through cross-layer management, covering hardware, platform, and

software configurations. At the hardware level, we need fine-tuning core-level adaptive clocking to address process variability. At the platform level, we need power and temperature management to leverage temperature inversion. At the software level, we need application scheduling to work with voltage variation.

The rest of this chapter is organized as follows. Chapter 1.1 provides an overview of my research contributions. Chapter 1.2 discusses the long-term practical impact of my thesis. Chapter 1.3 outlines the rest of the dissertation and Chapter 1.4 lists previously published materials that this dissertation draws upon.

1.1 Research Contributions

My Ph.D. research's objective is to study and optimize the active timing margin. In this pursuit, the thesis first provides an instrumental explanation over active timing margin's design and working mechanism, including one prevalent design flavor that is based on timing margin sensors that directly measures the saving room and automatically triggers voltage/frequency adjustment, and one alternative lower-cost design that uses environmental sensor such as temperature sensors to correlate the saving potential. Compared against prior proposals that optimize timing margin, the introduction highlights the significance of active timing margin as a practical and convenient solution that effectively captures the power efficiency opportunity in timing margin.

Secondly, the thesis brings the notion that to maximize active timing margin's efficiency enhancement utility, a collaborative hardware/software

design that works in synergy to dynamically and safely provision timing margin is needed. This insight is based on comprehensive hardware measurement and instrumentation, which shows the proposed methods provide over 10% measured power, or performance improvement, a highly lucrative benefit for production chips.

Thirdly, the thesis provides a complete analysis of the different system-level effects for which cross-layer management can help improve active timing margin's gain, at the author's best effort. Because timing margin in modern microprocessor pipeline is designed to combat against a wide variety of system effects, the optimization and management of active timing margin necessitate a solid understanding of all major effects that affect load environments, so that the power efficiency improvement does not hamper pipeline timing correctness. In this effort, my Ph.D. research dissects timing margin into three major components that it protects against - temperature (T), voltage (V), and process (P) variation. Although TVP variation has been thoroughly studied for static margin, their behavior are less understood in active timing margin because the technique is still new and there are not many systems available for study. I take a holistic view across system stack, spanning circuit, architecture, and application to decide for each effect what the appropriate active mechanism is to help active timing margin perform at its best.

- **For temperature variation:** I first propose T_i-states, an active timing margin solution based on the temperature sensor, that leverages a lucrative phenomenon called *temperature inversion* to reduce processor power.

Then, I analyze how to help T_i-states maximize its power reduction benefits, depending on the workload characteristics, the manufacturing technology node, and the chip operating temperature.

T_i-states replaces the conventional static margin design where a worst-case high temperature scenario is guarded against by allocating enough margin with an active timing margin solution that tracks runtime temperature change and the resulting circuit speed variation to adjust the real-time margin dynamically. In particular, T_i-states exploit the highly beneficial temperature inversion effect of CMOS transistors as technology scales down, where circuit speed accelerates significantly under higher temperature. We further show that with T_i-states, runtime processor temperature can be properly managed to maximize chip power reduction, depending on the workload's activity level, and the chip's manufacturing technology.

- **For voltage variation:** I study a production active timing margin system, the POWER7+ multicore, where a responsive per-core hardware feedback loop is implemented that adjusts core frequency and chip supply voltage based on real-time monitored timing margin amount. Through comprehensive hardware measurements, I show that among all voltage noise components, active timing margin deals with the notorious di/dt effects very effectively, which in conventional systems excess margin targeting worse-case di/dt needs to be allocated. However, I find active timing margin falls short in dealing with long-term DC voltage drop, which

is related to workload characteristics and chip-wide multicore activities. To maximize active timing margin's gains, we propose workload mapping management to balance compute loads on different active timing margin "domains". Our management points to power delivery network co-design for active timing margin. We show proper workload mapping can at least double active timing margin's power improvement.

- **For process variation:** I take an alternative perspective and investigate how to make active timing margin automatically push the highest performance out of each core on a multicore system. To achieve this goal, I perform in-depth per-core characterization and instrumentation on POWER7+'s shipped active timing margin design. The instrumentation exposes significant performance variation across cores, which is caused by the process variation of the pipeline itself, the variation of the active timing margin hardware loop, as well as the workloads' impact on DC voltage drop, discovered in the voltage variation research aforementioned. The inter-core performance heterogeneity has previously been hidden by the multicore's default active timing margin setting, which produces uniform frequency target across all cores. Our per-core active timing margin customization automatically brings out the core's highest speed, subject to frequency and application performance variation. To manage the performance variation of the resulting system, we propose a management scheme to improve application performance controllably. Measurement shows the proposed management boosts target application

performance by over 10%.

Put in the system layer context, the contribution of this thesis and the research effort I conducted to arrive at the aforementioned contributions covers circuit, architecture, and software level, illustrated by Figure 1.1.

At circuit and device level: I perform hardware measurement to understand what is the granularity that timing margin sensor measures available margin, and map circuit speed/delay to temperature or voltage variation. In the T_i-states proposal, the characterization of how temperature affects CMOS transistor speed, and hence available margin, is critical in building the active timing margin loop. In the voltage variation study, the dissecting of voltage noise into different components that active timing margin can, or can not deal with is built upon the understanding of how voltage affects circuit speed, using timing margin sensors.

At architecture design level: I propose designs that implement, or helps active timing margin perform at its best. T_i-states are a set of power management states stored as tables in system firmware, which are later indexed using runtime temperature sensor readings. T_i-state is essentially an evolution of classic power management states, such as P-states For voltage variation, the analysis we make points to an alternative power delivery network design which is separated into different domains, with each domain covering a few cores to minimize per-domain DC voltage drop and maximize active timing margin's voltage reduction capability.

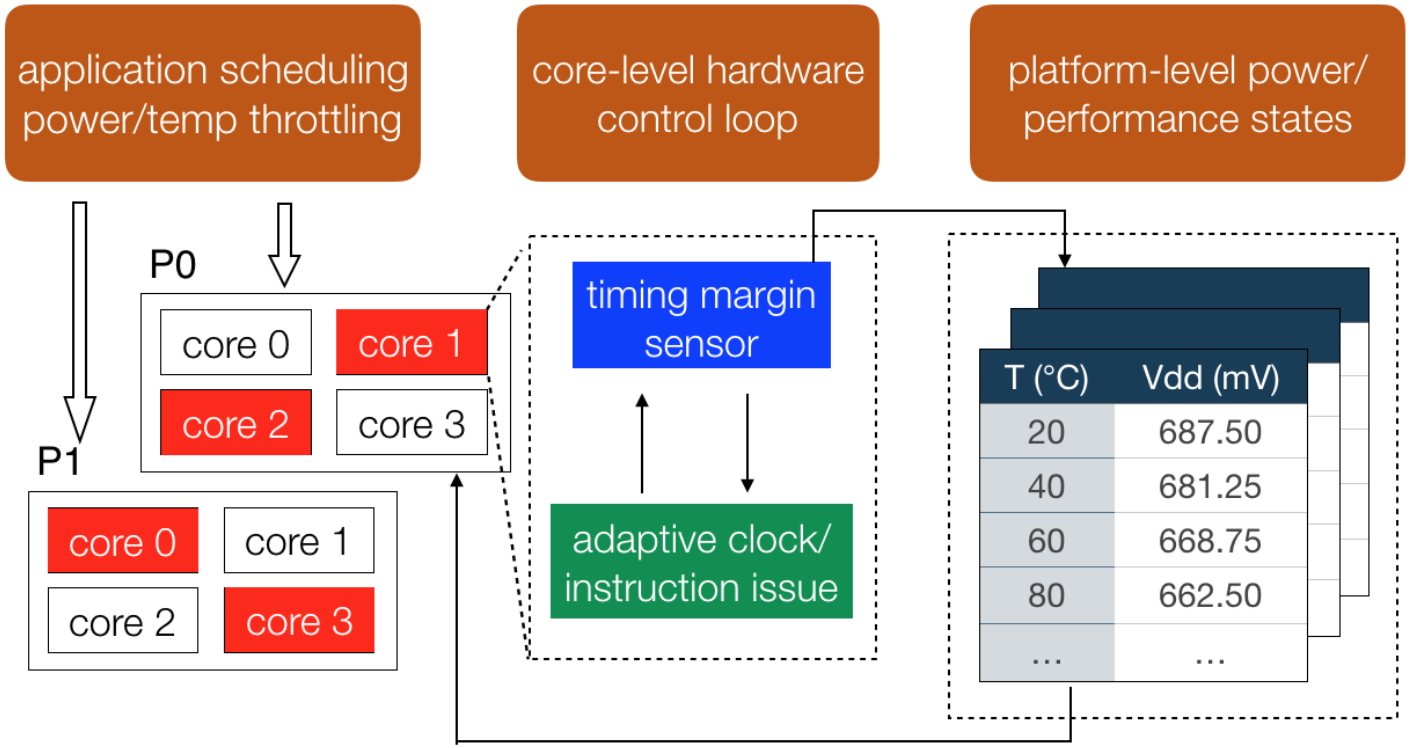


Figure 1.1: Overview of this thesis's cross-layer research on active timing margin management. We design platform-level active timing margin solution for temperature variation, characterizes state-of-the art hardware active timing margin mechanisms that deal with voltage variation, instrument core-level active timing margin fabric to expose process variation, and proposes software scheduling techniques to help these systems achieve the best power-performance gain.

At system software and application level: I propose application mapping and throttling technique on multicore systems to manage a microprocessor's DC voltage drop, with the goal of reducing total processor power or enhancing target application performance. The software solution serves as a complement to the hardware-only active timing margin mechanism and is proven by measurement to double the power reduction, or performance improvement gain.

1.2 Long-term Impact

As Dennard scaling and Moore's law approaching their end, and general-purpose architectures becoming ripe, it is vital that the research contributed to enhancing processor power efficiency have practical long-term impact, or they perish. The long-term impact of my thesis lies in three fundamental aspects:

First, my thesis on optimizing microprocessor timing margin has wide applicability in the semiconductor industry. It is not dependent upon one processor architecture and does not affect and interfaces between hardware and software. T_i-states was carried out on the GPU of an APU System-on-Chip (SoC), while the study on optimizing timing margin for voltage and process variation was conducted on a multicore platform. In principle, any processor architecture can benefit from the insights and proposals in this dissertation, including accelerators, if ultra power efficiency is desired.

Second, the active timing margin technique has seen proven commercial success by the time this dissertation is made. When I first initiated this

research topic five years ago, only a few processors are designed with active timing margin, mostly for experimenting and testing purpose [54, 14]. Today, the latest high-end chips almost all adopt this technique to squeeze out the last bit of power efficiency from silicon [100, 35, 13, 104, 103] because of its effectiveness and convenience for implementation. In this context, our work that tries to optimize active timing margin provides a free extra mile for chip vendors to increase the efficiency gain, proving its long-term impact.

Thirdly, all results and analysis presented in this thesis are acquired from solid, real hardware measurement. Acquiring and interpreting the type of data in this thesis is very difficult, which involves a deep study of a hardware platform's internals. The measurement data not only proves the improvements and insights we made can sustain future work's tests, but also provide trustworthy, valuable guidance for other researchers who need a reference. Thus, this dissertation's impact is of high practicality.

1.3 Dissertation Organization

The rest of my dissertation is organized as follows. Chapter 2 introduces the basics of pipeline timing margin, prior proposals that try to optimize it, and why active timing margin is the design choice today. Chapter 3, Chapter 4, and Chapter 5 describe the proposed active timing margin management schemes for temperature, voltage, and process variation, respectively. The work in these chapters are built upon solid characterization and analysis using hardware measurement, and the proposed work cross architecture and

software design. Chapter 6 provides a retrospective and prospective view of my dissertation work. The retrospective part summarizes the principles distilled from this work on building a maximally efficient active timing margin system; the prospective part suggests next steps for generalizing our work into massive industry adoption.

1.4 Previously Published Material

This dissertation contains materials that are previously published in peer-reviewed conferences and journals:

Chapter 2. The introduction on timing margin sensors, environmental sensors, and the control loop for active timing margin is a collection of the part of the materials published in the following papers: *T_i-states: Processor Power Management in the Temperature Inversion Region.* Yazhou Zu, Wei Huang, Indrani Paul and Vijay Janapa Reddi. In International Symposium on Microarchitecture (MICRO), 2016 [113]; *Adaptive guardband scheduling to improve system-level efficiency of the POWER7+.* Yazhou Zu, Charles R. Lefurgy, Jingwen Leng, Matthew Halpern, Michael S. Floyd and Vijay Janapa Reddi. In International Symposium on Microarchitecture (MICRO), 2015 [115]; *Active Timing Margin Management for Maximizing Multi-Core Efficiency on an IBM POWER7+ Server.* Yazhou Zu, Daniel Richins, Charles R. Lefurgy and Vijay Janapa Reddi. In International Symposium on High Performance Computer Architecture (HPCA) [116].

Chapter 3. The design and management of active timing margin for

temperature variation are based on the following paper: *T_i-states: Processor Power Management in the Temperature Inversion Region*. Yazhou Zu, Wei Huang, Indrani Paul and Vijay Janapa Reddi. In International Symposium on Microarchitecture (MICRO), 2016 [113]. A modified version of this paper is also published in IEEE's annual Top Picks selection: *T_i-states: Power Management in Active Timing Margin Processors*. Yazhou Zu, Wei Huang, Indrani Paul and Vijay Janapa Reddi. In IEEE Micro, June 2017, 37(3):106-114 [114]. I am the lead author of these papers, and I was responsible for proposing the idea, experimenting with it and evaluating the final results.

Chapter 4. The characterization of on-chip voltage noise, as well as its active timing margin management for power saving is based on the following paper: *Adaptive guardband scheduling to improve system-level efficiency of the POWER7+*. Yazhou Zu, Charles R. Lefurgy, Jingwen Leng, Matthew Halpern, Michael S. Floyd and Vijay Janapa Reddi. In International Symposium on Microarchitecture (MICRO), 2015 [115]. I am the lead author of this paper. I conducted all the comprehensive characterization experiments, proposed the idea to improve microprocessor power saving, and performed the associated evaluation.

Chapter 5. The work on active timing margin management for multicore process variation is is based on the following paper: *Active Timing Margin Management for Maximizing Multi-Core Efficiency on an IBM POWER7+ Server*. Yazhou Zu, Daniel Richins, Charles R. Lefurgy and Vijay Janapa Reddi. In International Symposium on High Performance Computer Archi-

tecture (HPCA), 2019 [116]. I am the lead author of this paper. I proposed the idea, performed the design space exploration, and conducted all performance improvement evaluation.

Other publications: During the length of my Ph.D., I've also worked on other related topics and made joint publications in GPU voltage analysis, Integrated Voltage Regulator analysis, etc. The co-authored papers are *GPUVolt: Modeling and characterizing voltage noise in GPU architectures.* Jingwen Leng, Yazhou Zu, Minsoo Rhu, Meeta Gupta and Vijay Janapa Reddi. In International Symposium on Low Power Electronics and Design (ISLPED), 2014 [58]; *GPU voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in GPU architectures.* Jingwen Leng, Yazhou Zu and Vijay Janapa Reddi. In International Symposium on High Performance Computer Architecture (HPCA), 2015 [57]; *Ivory: Early-stage design space exploration tool for integrated voltage regulators.* An Zou, Jingwen Leng, Yazhou Zu, Tao Tong, Vijay Janapa Reddi, David Brooks, Gu-Yeon Wei and Xuan Zhang. In Design Automation Conference (DAC) 2017 [112]; *Efficient and reliable power delivery in voltage-stacked manycore system with hybrid charge-recycling regulators.* An Zou, Jingwen Leng, Xin He, Yazhou Zu, Vijay Janapa Reddi and Xuan Zhang. In Design Automation Conference (DAC) 2018 [111]; *Voltage-Stacked GPUs: A Control Theory Driven Cross-Layer Solution for Practical Voltage Stacking in GPUs.* An Zou, Jingwen Leng, Xin He, Yazhou Zu, Christopher D. Gill, Vijay Janapa Reddi and Xuan Zhang. In International Symposium on Microarchitecture (MICRO),

2018 [1]. As co-authors of these papers, I help brainstorm ideas, carry out necessary experiments with the lead author, and refine the writing of these papers.

Chapter 2

Timing Margin: A Perpetual Role in Modern Microprocessors

This section explains the basics of the microprocessor's timing margin and motivates why active management of timing margin is necessary to improve power efficiency. This section first goes over the necessity of timing margin in modern microprocessors. Then it enumerates the main components involved in timing margin. We end this section with a brief discussion of the working mechanism of active timing margin.

2.1 The Importance of Pipeline Timing Margin

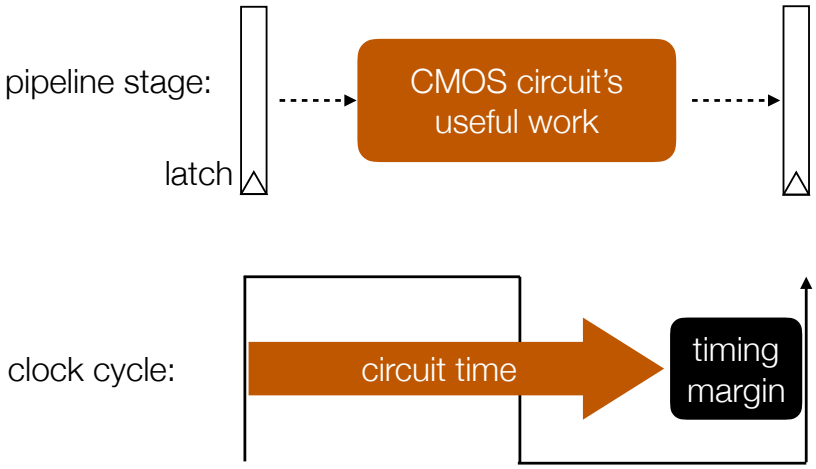
Timing margin is a necessary component in modern microprocessors, whether it is a general purpose CPU, Graphics Processing Unit (GPU), or specialized accelerators like Tensor Processing Unit (TPU) [49].

Almost all today's processors are pipelined for higher instruction throughput and workload performance. All pipeline stages have the same time duration to complete their computing, or circuit toggling tasks, synchronized by a global clock signal. Each cycle, circuits constructed by CMOS transistors take some time to switch and then produce a stable output electric level to be

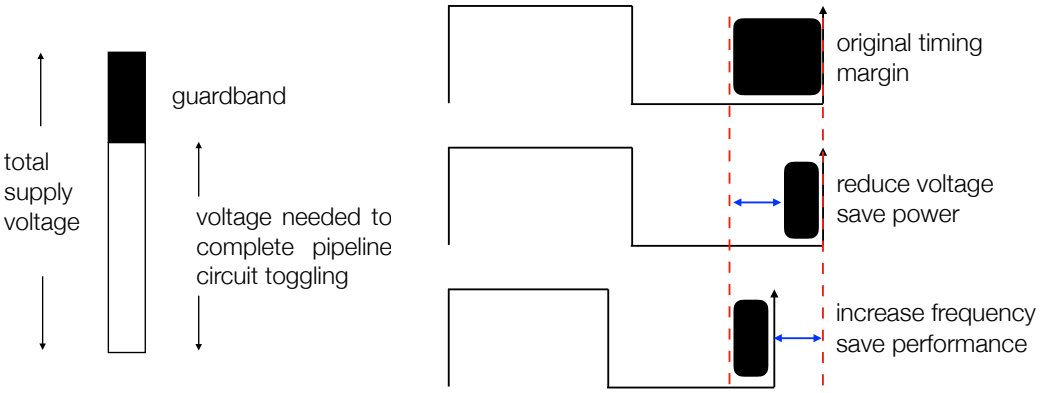
fed into latches, or registers. Ideally, the circuit's switch time can be calculated, or simulated using CMOS device's charge and discharge time formula given certain supply voltage levels and transistor parameters, and pipeline cycle time should be equal to the simulated switch time if all pipeline stages have balanced design and have the same switch time.

However, in practice the pipeline circuit's switch time have a lot of uncertainty. The uncertainty of pipeline timing can be caused by, for instance, transistor performance variation due to environment temperature variation, unstable supply voltage levels delivered to the transistors, and imperfect transistor size caused by manufacturing lithography, transistor aging, etc. These sources of timing uncertainty make circuit switch time deviate from their simulated normal points, which could make circuits complete their jobs faster, or slower than design simulation. To assure all circuits have plenty of time to complete their toggling, pipeline cycle time is always longer than the theoretical circuity switch time, the added time duration in clock cycle is called *timing margin*, as illustrated in Figure 2.1a.

Timing margin can be implemented as tuning supply voltage higher while leaving frequency target untouched, which makes circuits operator faster and thus leaving margin in the cycle time, or tuning the frequency slower while leaving supply voltage the same, which makes cycle time longer and creating margin. These two methods are equivalent. The former approach is widely known as *voltage guardband* as described in Figure 2.1b. In this thesis, we explore opportunities in both designs, i.e., reduce the voltage to save power in a



(a) Timing margin is the time left in clock cycle after circuit completes its work.



(b) Voltage guardband

(c) Power/Performance saving

Figure 2.1: Timing margin ensures processor execution correctness by allocating extra room in pipeline's clock cycle time. Timing margin can be delivered by providing extra voltage, known as the voltage guardband, or alternatively slowing down frequency. Safely reducing the timing margin can improve power via undervolting, or improve performance via overclocking.

voltage guardband approach, and increase frequency to improve performance.

A good analogy of the microprocessor pipeline's timing margin in everyday life is the relationship between cars and lanes. Ideally, a lane would have the same width as a car if cars can strictly move with the shape of the lane. Yet, in reality, lanes are always much wider than cars because cars often deviate from lane orbits. The deviation uncertainty may be caused by a human driver's improvisation, or the inherent control error of the vehicle (e.g., a mismatch between the left and right tires). The extra space between a lane and a car allows tolerates these errors and make sure no accidents occur. The extra room between lane width and car width works just like how pipeline timing margin protects against circuit's timing uncertainty.

Failing to provide enough timing margin is catastrophic for modern processors as it can lead to pipeline timing errors, causing incorrect application execution results, or even system crash. Circuits need enough time to deliver the correct signal for the next pipeline stage to compute on. With not enough margin, the circuit may not have enough time to produce the correct bit, due to the unusual load environment like extreme temperature environments. The erroneous bits can be meaningless, pointing to a wrong data address, or representing an invalid instruction that cannot be decoded, breaking microprocessor's correct execution stage.

This dissertation addresses the timing margin issue on a processor, primarily CPUs and GPUs. However, timing margin widely exists on other kinds of chips that use pipeline microarchitecture, and are equally important to

guarantee correct chip functioning. One intuitive example is the "rowhammer" in DRAM chips [50]. The authors in [50] found that stressing one row in DRAM chips can cause adjacent row's cells to have erroneous bit storage. The underlying mechanism is because frequency access to a row creates electrical interference to the supply voltage of the adjacent row's wordline, which makes DRAM cell capacitor to leak charge quicker than designed bit retention period. To protect against this situation, DRAMs need to allocate more margin to bit refresh frequency, making sure the cell's charge leakage time does not exceed the threshold for retaining the correct bit in the worst runtime environment. Though DRAM is not exactly the same as a microprocessor's logic circuit structure, this case illustrates the importance of timing margin in pipelined chips.

2.2 What Consumes the Timing Margin?

Timing margin is excessively high in today's processors, costing over 20% extra supply voltage in shipped chips [89, 57]. Our work, along with much prior art try to reduce timing margin magnitude and saves power or performance. The first step to achieve this goal is to understand what timing margin is allocated for, or at runtime what phenomenon makes pipeline circuit time deviates from normal and erodes, or consumes the margin. Although the uncertainty in pipeline circuit timing is caused by parasitic effects in the computer system, they are not completely random. In this dissertation, we dissect the modern processor's timing margin into three main components

follwing prior art's study on static margin, namely temperature, voltage, and process variation (TVP variation [87]). These three effects are deemed the biggest consumers that contribute to timing margin's high amount, and they happen mostly in independent manners which facilitate us to optimize them one by one.

Temperature variation is one important source of timing uncertainty. When temperature changes, transistor performance varies because temperature variation alters the activity level of the particles flowing in CMOS transistor's channel, which changes transistor switch speed and circuit completion time. In practice, processor temperature variation is unavoidable because during workload run the charge and discharge of semiconductor transistors inevitably raise the temperature. Depending on workload intensity, the temperature profile of the chip varies temporarily and spatially, affecting circuit timing. To tolerate timing uncertainty caused by temperature variation, margin must be added in the cycle time. In Chapter 3 we perform an in-depth study on how temperature affects timing margin and propose a feedback loop with corresponding management to combat against it.

Voltage variation is a very dangerous source of timing uncertainty. It is caused by the interaction between the parasitics of the power delivery subsystem of a microprocessor and the processor's varying power draw under workload execution. Figure 2.2 from [58] gives an intuitive overview of the electrical mechanism of voltage variation.

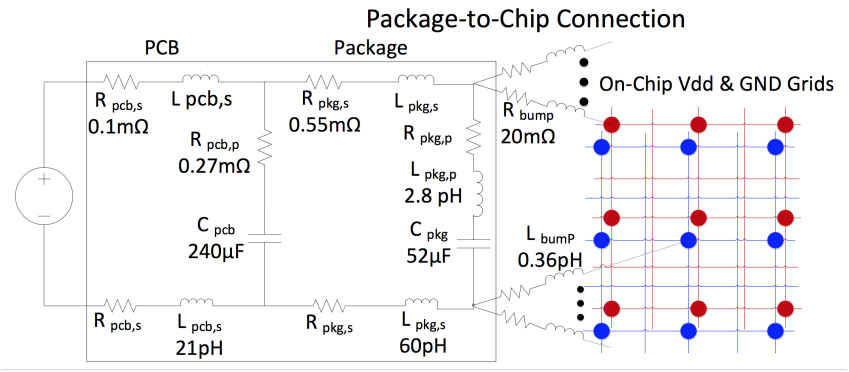


Figure 2.2: An electrical model of a computer's power delivery subsystem, from the voltage regulator module (VRM) to on-chip transistors. Resistive, capacitive, and inductive impedance exist on this path, adding noise to the voltage delivered to transistors which causes timing uncertainty.

The supply voltage delivered to the CMOS transistors contains lots of noise because the path from the source of the voltage supply to the end transistors has electrical parasitics, including resistive, capacitive, and inductive components. Typically, the power supply subsystem can be modeled as four parts: the voltage regulator module (VRM), the printed circuit board (PCB), the package, and on-die power delivery network (PDN). Each part contributes to the total impedance. The impedance exists because the power delivery subsystem is made of real, physical materials - on-chip and off-chip wires have resistance even the magnitude can be small, wires can form loops by chance and create inductive impedance, the alignment between wires and the added decoupling capacitors create capacitance, etc.

Resistive parasitics cause supply voltage's IR drop following Ohm's law. Higher power causes a higher IR drop. Inductive and capacitive parasitics

further worsen supply voltage with the di/dt effects. The di/dt effect happens when there is a rapid change in the current draw or the power consumption of the microprocessor. The di/dt effect happens very rapidly, typically over tens of cycles, yet very rarely. The combined IR drop and di/dt effects make the supply voltage experienced by transistors very noisy, adding uncertainty to circuit timing. In Chapter 4, we perform an in-depth analysis on how state-of-the-art hardware tries to reduce voltage noise and propose management techniques to squeeze out power efficiency from voltage noise.

Process variation is another source of uncertainty if pipeline circuit timing. Unlike temperature and voltage variation which change dynamically during runtime, process variation is a static effect that is formed during chip's manufacturing lithography process. During lithography, transistor performance variation occurs because the lithography instruments cannot perfectly control the various lithography steps, such as etching and doping. Wire width, transistor gate width, and length can be etched with noise. Dopant density can deviate from the ideal density level. All these effects make transistor and wire's performance deviate from the ideal case, and make the speed of different transistors and wires differ. A microprocessor's performance is determined by the slowest part of the chip. The result is that faster circuits are forced to have some amount of timing margin because it is synchronized using the same clock as the slow circuits. In Chapter 5 we devise automatic methods to expose a multicore's performance variation caused by process variation, and propose

management schemes to manage the resulting non-deterministic application performance.

We acknowledge there are other effects that also contribute to the timing margin, such as transistor aging and processor testing inaccuracy. However, these effects are not as strong as the temperature, voltage, and process variation aforementioned in a processor's typical lifetime, and the condition for it to occur is too extreme. For this reason, we leave out these effects in this dissertation.

2.3 The Need for Active Timing Margin

While it is intuitive to allocate timing margin in pipeline cycles to combat various effects that cause circuit timing uncertainty, the specifics of how the amount of margin is calibrated is intricate.

In traditional designs, the margin is estimated during the chip design and testing stage, following a *worst-case* design approach. Under this approach, the amount of timing margin is a static value that is able to tolerate the most extreme conditions that slow down microprocessor circuits, such as very heavy di/dt voltage droops, very large IR drop caused by high power workloads, and unusual operating temperature that degrade transistor performance significantly. Because timing margin is a fixed value in this design, failing to consider all corner cases or allocating margin not conservatively enough may neglect an extreme corner case the user might create to hamper pipeline timing. Thus, the static worst-case timing margin must aggregate corner case of

all effects to determine the total amount of margin is added to the chip.

The cost of the worst-case margining approach, however, is prohibitively high despite its straightforward and low-overhead implementation. Prior art has shown that the extreme conditions that will utilize all the margin happen extremely rarely, for voltage noise the heavy di/dt droops over 10% of the supply V_{dd} happens less than 1% of the time [89, 58], while timing margin must protect against these rare worst cases, leaving the margin unused most of the time. In [56], researchers reported that 20% of the supply voltage of a commercial GPU can be safely reduced without causing program execution errors, which reflects the huge amount of voltage guardband and timing margin in today's chips. The wastage is significant not only because of the power and energy wasted, but also because today's microprocessors are inherently power limited, and wasting power means limiting processor performance. Therefore it is imperative that we investigate what leads to timing uncertainty and consumes timing margin.

Many research efforts have been made to reduce the magnitude of the conventional static worst-case margin, ranging from benchmarking and simulation efforts to understand the worst-case limit [51, 8, 91], microarchitecture analysis to characterize the events that lead to rare extreme noise conditions [78, 37, 38, 88], low-overhead architecture design for error toleration and noise smoothing [39, 88, 57, 28], to software techniques including compilation, scheduling, and runtime management to avoid high timing margin consumption [86, 69, 74, 56]. Many of these works have shed useful insights and are

partly incorporated into the latest designs. However, most of these proposals are not adopted as a whole by industry, either because of their high design overhead, lack of reliability guarantee, or unacceptable performance overhead.

This dissertation concentrates on one particular design flavor widely adopted for reducing timing margin, *Active Timing Margin*. The idea of active timing margin is very intuitive - instead of providing margin for the worst case, active timing margin provides just enough margin under the present condition, and dynamically stretches the margin when emergent event occurs, whether temperature goes to extreme, voltage falls very low, or threads are running on a flow core. Active timing margin relies on environment sensors to check runtime load conditions, including timing margin sensors, temperature sensors, power sensors, etc, rather than performance event counters to predict when a high-stake event may occur as the cost of misprediction can be high.

Figure 2.3 illustrates a high-level design of active timing margin. It uses a control loop between the sensor and the voltage/frequency controller to adjust the timing margin based on real-time monitored load environment. Because active timing margin has very low design and verification overhead, and it has been proven to be effective in mitigating process, voltage, and temperature variation, active timing margin has become the de facto approach for modern chips to reduce margin [54, 14, 100, 35, 13, 104, 103, 113].

Figure 2.4 from [54] illustrates how active timing margin deals with the dangerous di/dt effect. Starting at 1500 ns, a heavy voltage droop caused by strong di/dt effect slows down circuits and necessitates some amount of

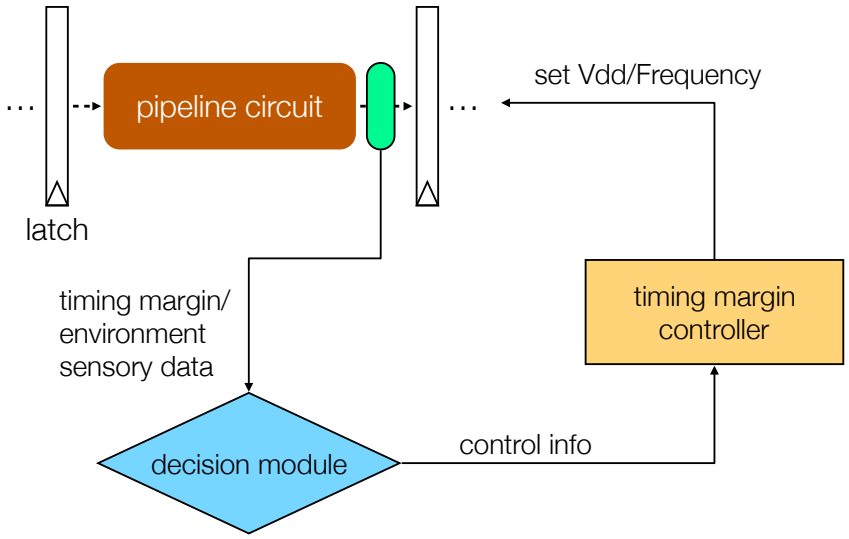


Figure 2.3: Active timing margin is a control loop that detects timing margin and related chip load environment, and accordingly adjust supply voltage or operating frequency in real-time to supply just enough margin.

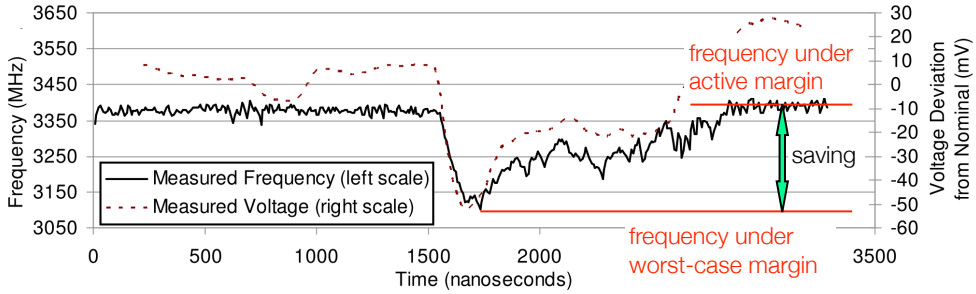


Figure 2.4: Active timing margin protects di/dt effect by making frequency/clock cycle track supply voltage, which improves performance and reduces timing margin wastage.

timing margin to protect against it, losing performance under the same supply voltage. The static margin needs to set at the lowest frequency at 3050 MHz to tolerate the di/dt effect. However, only small voltage ripple occurs on the power delivery network when heavy di/dt effect is over. During these periods, the large timing margin is not needed, yet the static approach still provisions the timing margin set by the worst case, wasting a lot of performance under the same voltage.

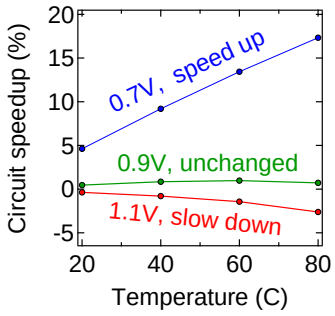
To reclaim the wasted frequency, active timing margin dynamically adjusts clock frequency to match the magnitude of voltage variation. When the di/dt effect occurs, clock frequency ramps down quickly to provide the need timing margin. When there's no di/dt effect, clock frequency stays at a higher level to reclaims the unused timing margin. Overall, the system enjoys a higher performance.

Chapter 3

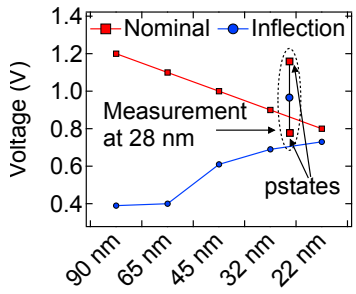
Ti-states: Active Timing Margin Management in the Temperature Inversion Region

Temperature has an intuitive impact on circuit speed, timing, and pipeline timing margin because CMOS transistor performance varies under different chip temperature levels. Conventionally, chip designer's view is that transistor slows down a lot under a higher temperature, so timing margin is set against the worst-case high temperature. However, we find this view no longer holds in today's CMOS technologies owing to an effect called *temperature inversion*. In today's state-of-the-art technology nodes, the temperature inversion effect is the *major temperature-related effect* that changes circuit performance and hence affect pipeline timing margin, and this phenomenon induces high speed variation. Therefore, we devise an active timing margin solution for temperature variation, with a focus on the temperature inversion effect, and explore system-level management scheme to achieve the highest power saving.

Formally, temperature inversion refers to the phenomenon that in certain voltage regions transistors speed up and operate faster at a higher temperature. Figure 3.1a illustrates the temperature inversion effect we measured



(a) Under low voltage, temperature inversion increases circuit performance.



(b) Temperature inversion's inflection voltage approaches nominal supply.

Figure 3.1: Temperature inversion is having more impact on processor performance as technology scales.

on an AMD® A10-8700P processor [70]. It shows the normalized circuit performance under different temperature with respect to a 0°C baseline. At 1.1 V, as temperature increases, circuit performance becomes *slightly* slower at 80°C, as expected from conventional wisdom. However, at 0.7 V circuit becomes much faster as temperature increases to 80°C owning to the temperature inversion phenomenon. Between 1.1 V and 0.7 V there exists a special *inflection voltage* level at 0.9 V where circuit speed remains almost constant at all product specified temperatures.

At a high level, the reason why temperature inversion occurs is a resulting of two fundamentally conflicting effects - when the temperature increases both carrier mobility and threshold voltage decrease. Carrier mobility decrease causes devices to slow down while threshold voltage reduction causes the devices to speed up. Temperature inversion happens in the region where the supply voltage is low enough to make the second factor (i.e., threshold

voltage reduction) dominate, which is 0.7 V in Figure 3.1a. Otherwise, the devices slow down at the higher temperature, degrading performance as in the case of 1.1 V.

In the past, temperature inversion has been safely discounted by processor designers because the nominal supply voltage at which this effect starts to occur is too low in prior technologies. At 250 nm, when temperature inversion was first discovered, the inflection voltage was more than 1.5 V lower than the nominal supply voltage [75, 6, 21]. With such a wide margin of separation, temperature inversion does not interfere with the processor's normal operating voltage region.

However, with technology scaling, today's processors are operating close to the temperature inversion's voltage region. Thus, the impact of this effect can no longer be safely discounted. Figure 3.1b shows a detailed device analysis based on predictive technology models [105, 109]. As technology scales down from 90 nm to 22 nm, the inflection voltage increases with smaller feature sizes. At the 32 nm node, the inflection voltage is predicted to closer to the nominal supply voltage. Scaling into future FinFET and FD-SOI devices with smaller feature sizes, it is likely that temperature inversion will occur for all of a processor's operating voltage range [53, 15].

Silicon measurements performed on the AMD® A10-8700P processor confirm this behavior in practice. At the 28 nm node, the inflection voltage in Figure 3.1b falls within the range of the processor's different P-states. The integrated GPU's highest P-state is only slightly above the inflection point.

The fact that temperature inversion is the major temperature-related effect that varies circuit speed and timing margin today, and the fact that it has been neglected by the architecture community in the past make it imperative to thoroughly investigate the potential implications temperature inversion imposes on timing margin and architecture design. For this reason, we focus on exploiting temperature inversion for actively provisioning timing margin depending on runtime temperature level and transistor temperature inversion intensity. The rest of this section is organized as follows: Chapter 3.1 explains our experimental setup, Chapter 3.2 systematically characterize how temperature affects circuit speed in contemporary microprocessors, Chapter 3.3 proposes our T_i-state solution for active timing margin, Chapter 3.4 discusses how to manage systems equipped with T_i-states, and Chapter 3.5 addresses related work.

3.1 Experimental Setup

In this subsection, we provide an overview of the experimental platform to study temperature inversion, including the chip under study and our temperature control mechanism. In particular, we explain the timing margin sensor we use, which serves as *power supply monitor* in this chip. Timing margin sensor is the key element of our work.

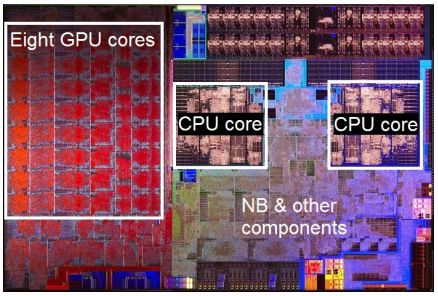


Figure 3.2: Die photo of the A10-8700P SoC.

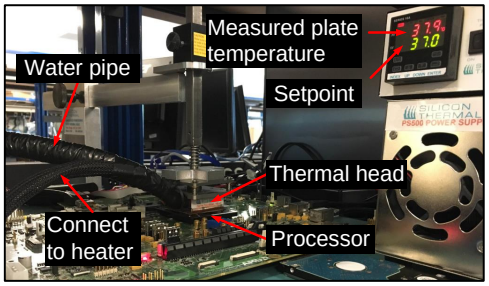


Figure 3.3: Temperature control setup.

3.1.1 AMD® A10-8700P Accelerated Processing Unit

The AMD® A10-8700P Accelerated Processing Unit (APU) is a System-on-Chip manufactured in 28 nm HKMG planar bulk technology. It integrates two CPU core-pairs, eight GPU cores, and other components as shown in Figure 3.2. Each CPU core-pair contains two out-of-order cores that share the front-end and floating point units. Each GPU core includes four 16-lane wide single instruction multiple data (SIMD) units.

We conducted temperature inversion studies on both the CPU and GPU. A separate power delivery network allows us to control the CPU and GPU voltage independently. But in this work, we present the results for the GPU only because the GPU's throughput-oriented architecture allows low-voltage region operation with meaningful and realistic performance. However, because the temperature inversion effect we study depends solely on the supply voltage, and not necessarily the underlying architecture, the analysis and benefits we present on the GPU naturally do extend to the CPU as well.

The GPU clock is set at 300 MHz in the voltage region we explore

around 0.7 V. We pick 300MHz because its associated low voltage is within the temperature inversion region, and makes it possible to explore the potential impact of temperature inversion on future near-threshold technologies. The 300 MHz frequency corresponds to the GPU's lowest P-State, and in practice, we have observed this P-State being exercised frequently during normal workload execution.

We use the ATITool [2] to set the GPU's voltage and frequency over a wide operating range. To measure power, we use a National Instrument's DAQ that reads the GPU's isolated supply voltage rail once every 10 ms.

3.1.2 Temperature Control Setup

To characterize temperature inversion's effect on performance and power under different operating conditions, we have to carefully regulate the processor's on-die temperature. In our work, we generally sweep temperature range from 0℃ to 80℃. This temperature range falls within the product's operating temperature range and does not affect aging significantly.

Figure 3.3 shows our temperature control setup. A thermal head is attached to the processor package. To stabilize the die temperature, which is measured via an on-chip thermal diode, at a user-specified target value, the thermal head's temperature is adjusted every 10 ms. Physically, the thermal head's temperature is controlled via a water pipe and a heater. The water pipe is connected to an external chiller to offer low temperatures while the heater increases temperature to reach the desired temperature setting. Under

feedback control, we see a 2℃ temperature variation on the diode in the worst-case. So, for instance, Figure 3.3 shows the thermal head sets its temperature to 37℃ to let the die temperature stay at 40℃.

3.1.3 Timing Margin Sensors: On-chip Power Supply Monitors (PSMs)

We use power supply monitors (PSMs) [35, 34] to accurately measure circuit speed changes in the chip under different temperature conditions. A PSM is a time-to-digital converter that reflects circuit time-delay or speed in numeric form. Originally designed as a voltage noise sensor, a PSM can sense minute circuit timing changes due to di/dt droops [35]. We use the PSM as a means to characterize circuit performance under temperature variation.

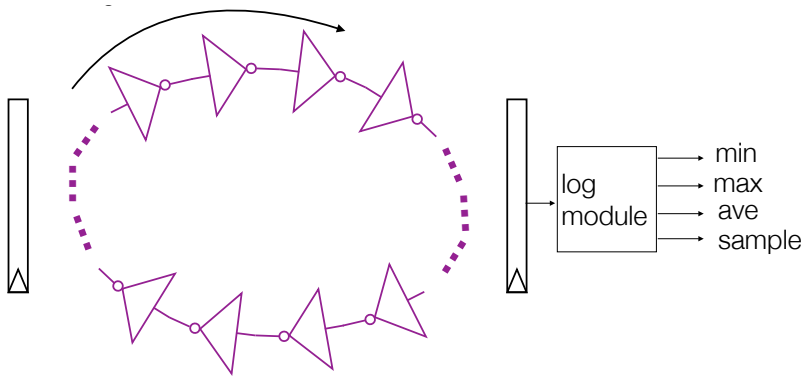


Figure 3.4: Power supply monitors (PSMs) measures pipeline speed/timing margin with an inverter ring. By counting how many inverters an edge has traveled through, the PSM reports a digital value that reflects circuit speed.

Figure 3.4 shows the structure of a PSM. Its core component is a ring oscillator that counts the number of inverters an "edge" has traveled through

in each clock cycle. When the circuit is faster (e.g., under smaller di/dt effects or stronger temperature inversion), an edge can pass more inverters and PSM will produce a higher count output. A supporting module logs ring oscillator's per-cycle output and accumulates the minimum, maximum, and average values over a time.

The A10-8700P processor has ten PSMs in each CPU core-pair and two PSMs in each GPU core, distributed across the cores to account for process variation and spatial differences in di/dt effect. Through measurements we determined that the changes in the different PSM readings under different temperatures are nearly identical, thus we only show the result of one representative PSM in GPU. The results are representative of using other or more than one PSM.

For reasons that prevent us from showing absolute values, we normalize the PSM reading to a reference value measured under 0.7 V, 300 MHz, 0℃, and idle chip condition. We log the minimum, maximum, and average output of all the PSMs.

3.2 Characterizing Timing Margin Under Temperature Inversion Variation

In this section, we first view the timing margin sensor (i.e., PSM) as a normal logic path to understand circuit performance under different temperature environment (Chapter 3.2.1). Then, we use the circuit speed difference to extrapolate how much extra margin can be squeezed out due to timing margin

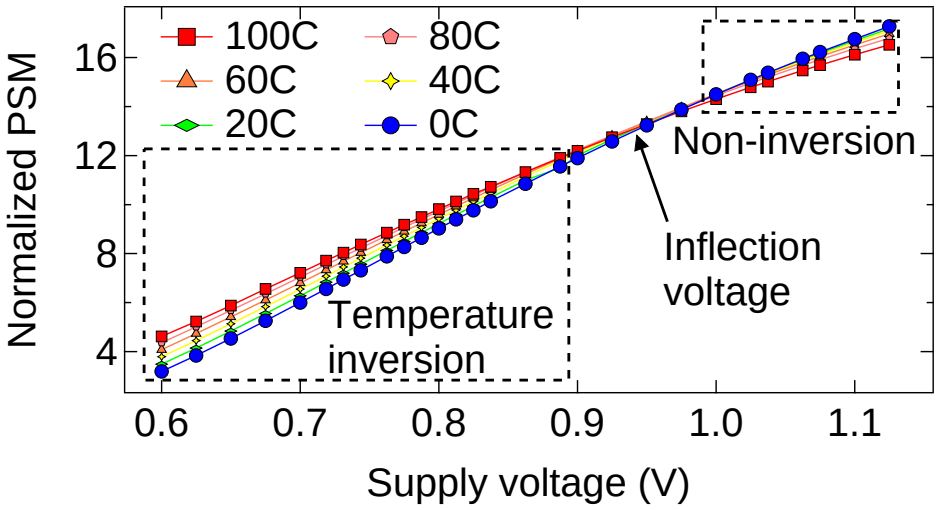


Figure 3.5: Temperature inversion happens below 0.9 V and is progressively stronger when voltage scales down.

change caused by temperature inversion variation (Chapter 3.2.2).

3.2.1 Circuit Speed Variation Under Different Temperature

The PSM by itself is a digital circuit located between the pipeline latches with other normal logic paths [95], and therefore its speed characteristics are representative of a pipeline's overall performance. For this reason, we use the PSM's output to quantify circuit performance across a wide range of different steady-state temperatures.

We keep the chip idle (i.e., the clock is still running) and read the PSM's "average" value to exclude the di/dt effect caused by workload dynamics. Figure 3.5 shows the circuit speed under different supply voltages and die temperatures. Speed is reflected by the PSM's normalized output – higher value implies a faster circuit. At a higher supply voltage, the circuit switches

faster, and the PSM can travel more inverters in a cycle which produces a higher count. The voltage-to-PSM relationship conforms to similar analysis as in [115].

We find that temperature's impact on circuit performance depends on the supply voltage. In the high supply voltage region around 1.1 V, the PSM's reading becomes progressively smaller as the temperature rises from 0℃ to 100℃. The circuit is operating slower at a higher temperature, which aligns with conventional belief [56]. The reason for this circuit performance degradation is that the transistor's carrier mobility decreases at a higher temperature, leading to smaller switch-on current (I_{on}) and longer switch time [105].

Under a lower supply voltage, the PSM's reading increase with higher temperature, which means the circuit switches faster (i.e., the temperature inversion phenomenon). Under temperature inversion the transistor's threshold voltage (V_{th}) decreases linearly as temperature increases [105, 75, 21]. Thus, for the same supply voltage, a lower V_{th} provides more drive current (I_{on}) which makes the circuit switch faster. The speedup effect is more dominant when the supply voltage is low because then the supply voltage is closer to V_{th}.

When the supply voltage is low enough, the speedup contribution from the reduced V_{th}, at some point, will balance out the carrier mobility slowdown. We call this voltage point the *inflection voltage*. The inflection voltage may change from chip to chip due to V_{th} variations, and it can be characterized during the binning process. In Figure 3.5, we show that the tested processor's

inflection voltage is between 0.9 V and 1 V. In this region, the temperature does not have a notable impact on circuit performance. Below the inflection voltage (0.95 V) is the temperature inversion region while above it is the non-inversion region. Half of the GPU's P-states, which range from 0.75 V to 1.1 V, operate in the temperature inversion region.

In Figure 3.5's temperature inversion region, the speed change between any two temperatures increases when the supply voltage scales further away from the inflection point. As voltage scales into the lower voltage region around 0.6 V, the PSM reading varies by more than 40%, indicating the drastic speedup at a higher temperature. As voltage goes lower towards the near-threshold region, the overdrive voltage ($V_{dd} - V_{th}$) becomes small and it is very sensitive to small V_{th} changes. Thus, temperature inversion's V_{th} reduction has a more significant impact on device performance.

Figure 3.6 zooms into the low voltage region between 0.6 V and 0.86 V and has a clearer view of temperature inversion. The figure shows temperature inversion's performance benefit at 100℃ over the 0℃ baseline, and this benefit increases as the supply voltage decreases. Hereon forward we use temperature inversion at 0.7 V as a case study to dive deeper and get more insights. Although we restrict ourselves to this single voltage, there is ample opportunity to demonstrate how temperature inversion may add new ingredients to overall system management.

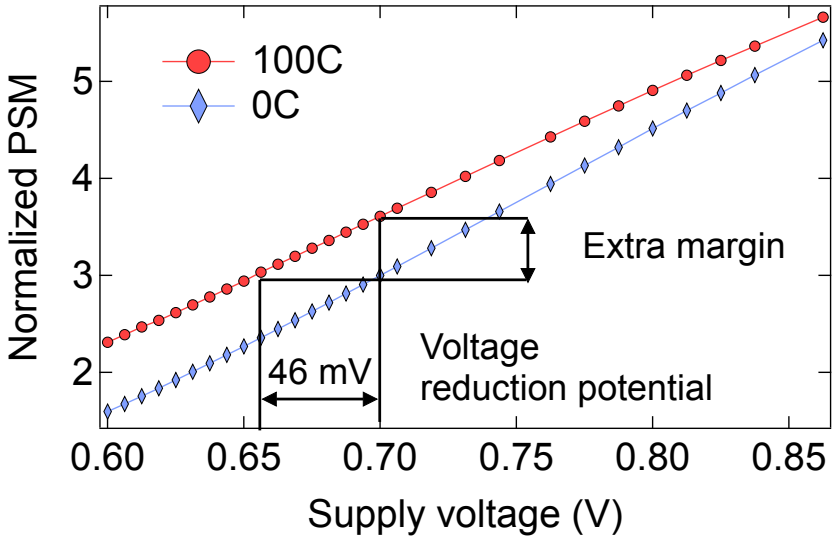


Figure 3.6: Estimating voltage reduction potential based on PSM characterization at different temperatures.

3.2.2 Estimating Active Timing Margin's Undervolting Opportunity

In this subsection, we provide a "design space exploration" of active timing margin's voltage reduction opportunity. When running workloads, chip temperature frequently goes up and speeds up circuits because of temperature inversion. This adds extra timing margin in the pipeline, and the extra margin can be exploited via undervolting.

To determine the amount of extra timing margin that can be exploited, we first need a "baseline margin" where timing margin is not overprovisioned for temperature variation. In other words, the "baseline margin" is the timing margin allocated for the worst-case temperature. It can tolerate all other effects such as di/dt and aging at worst-case temperature, yet circuit speed

cannot be degraded anymore compared to worst-case temperature. When undervolting, it is crucial that the system only reclaims the extra timing margin added from this "baseline margin" and does not reclaims anymore. Otherwise, pipeline timing may fail under some worst-case workloads, such as in the case of voltage stressmarks [51, 8].

We use the timing margin measured at 0℃ as the "golden" reference when reclaiming temperature inversion's extra margin. In other words, the timing margin delivered by our active timing margin scheme should match the "golden" reference. Under this constraint, we can undervolt to maximize power saving.

We choose 0℃ as the reference because under temperature inversion lower temperature degrades circuit performance. Even though 0℃ rarely occurs in desktop, mobile, and datacenter applications, the timing margin still needs to be set to tolerate this worst-case condition. In the industry, 0℃ or below is used as a standard circuit design guideline [44]. In certain scenarios, such as military use, an even more conservative reference of -25℃ is considered [21].

Figure 3.6 shows our estimation process of how much voltage can be reduced via active timing margin. The PSM difference between the high-temperature 100℃ line and the "golden reference" line at 0℃ represents the extra timing margin in the units of inverter delays. In other words, it reflects how much faster the circuits can run at a higher temperature. To bring the faster circuit back to the original speed, supply voltage needs to reduce such

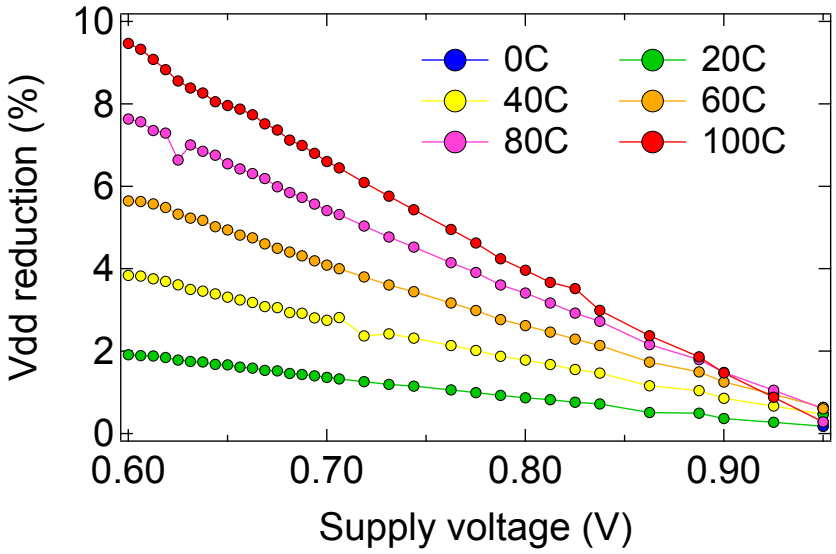


Figure 3.7: Voltage reduction potential is more pronounced in the near-threshold low voltage region.

that under a higher temperature the PSM will ideally read the same value.

We estimate the voltage reduction potential with linear extrapolation. Figure 3.7 shows the estimated opportunity at different temperatures. As supply voltage scales down, the voltage reduction potential goes up almost linearly. Temperature inversion effect is stronger in the lower voltage regions, and hence the greater timing margin opportunity. At 0.6 V and 100℃, the extra timing margin provided by temperature inversion can turn into almost 10% voltage reduction compared to 0℃. As a reference, 5% voltage reduction is considered significant in previous works [104]. At 0.7 V in our study, we can have 1.5% to 7% voltage reduction potential depending on the processor temperature.

3.3 Temperature Inversion States (Ti-States)

Having understood temperature inversion's potential for active timing margin, we propose a systematic method to establish a precise and reliable temperature to voltage mapping that implements active timing margin. The temperature to voltage mapping is discrete as voltage regulator module's output voltage is quantized in small steps [45]. The final mapping is, therefore, in a table format, which we call T_i-states. Similar to the way P-states functions for DVFS, T_i-state is a natural evolution of power management mechanisms for active timing margin.

3.3.1 Methodology to Construct the Ti-States Table

We propose a workload-centric methodology that constructs a set of temperature-voltage states in the inversion region (T_i-states) at test-time. A workload-centric approach ensures T_i-states will work in the face of workload-induced uncertainties like di/dt and IR effects. We use a subset of workloads as the "training" set to first get a tentative temperature-voltage mapping. Then we validate this mapping with another set of "test" workloads to establish the final T_i-state. During training the T_i-state is constructed in a manner that is agnostic to workload-specific settings, so we can be sure our voltage selection will provide enough margin for any workload that is run on the processor.

For each of the training workloads, we first measure their "golden" reference margin at 0℃ under our controlled temperature setup. Then, at the temperature being characterized, we select four candidate voltages. These can-

didate voltage are picked such that they are around the extrapolated voltage value from Figure 3.7. The candidates voltages are chosen such that they are two VRM steps above and two VRM steps below the extrapolated value.

Algorithm 1 T_i-state Construction Methodology

```
1:  procedure GET REFERENCE MARGIN
2:      set voltage and temperature to reference
3:      for each training workload do
4:          workloadMargin ← PSM measurement
5:          push RefMarginArr, workloadMargin
        return RefMarginArr
6:  procedure EXPLORE UNDERVOLT
7:      initVdd ← idle PSM extrapolation
8:      candidateVddArr ← voltage around initVdd
9:      minErr ← MaxInt
10:     set exploration temperature
11:     for each Vdd in candidateVddArr do
12:         set voltage to Vdd
13:         for each training workload do
14:             workloadMargin ← PSM measurement
15:             push TrainMarginArr, workloadMargin
16:         err ← diff(RefMarginArr, TrainMarginArr)
17:         if err < minErr then
18:             minErr ← err
19:             exploreVdd ← Vdd
        return exploreVdd
```

Once we have the set of candidate voltages, we step through each candidate voltage and record the training workloads' timing margin using the PSM at every temperature that is being characterized. The timing margin measured at the candidate voltage is compared against the reference margin. Finally, we select the candidate voltage that has the minimum PSM difference from the golden reference.

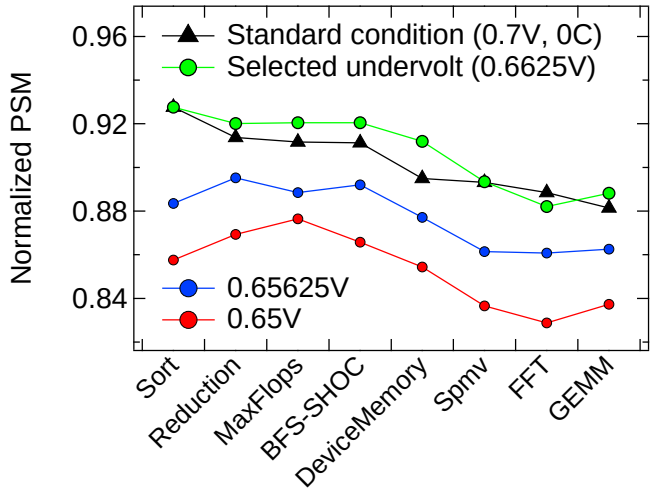


Figure 3.8: Exploring T_i-state at 80℃: we measure the "training" workloads' timing margin, and choose the V_{dd} that best tracks the standard margin.

It is worthwhile to note that on our particular chip the data variation for the 16 PSMs on our GPU is under 2%, so it makes little difference to use worst-case versus average. However, under severe intra-chip variation, transistor's undervolting potential can differ significantly. In that case, worst-case PSMs values need to be used for comparison.

Algorithm 1 summarizes our methodology. Figure 3.8 shows an example at 80℃. At this temperature, Figure 3.7's extrapolated voltage is 0.65625 V. The candidate voltages are 0.6625 V, 0.65625 V, and 0.65 V. Our platform's smallest VRM step is 6.25mV. The original four candidate voltage is capped by a lower hard limit of 0.65 V, and so we cannot set the voltage any lower. Algorithm 1 chooses 0.6625 V as the T_i-state voltage for 80℃ because it has the closest timing margin compared to "golden" reference. Other candidate

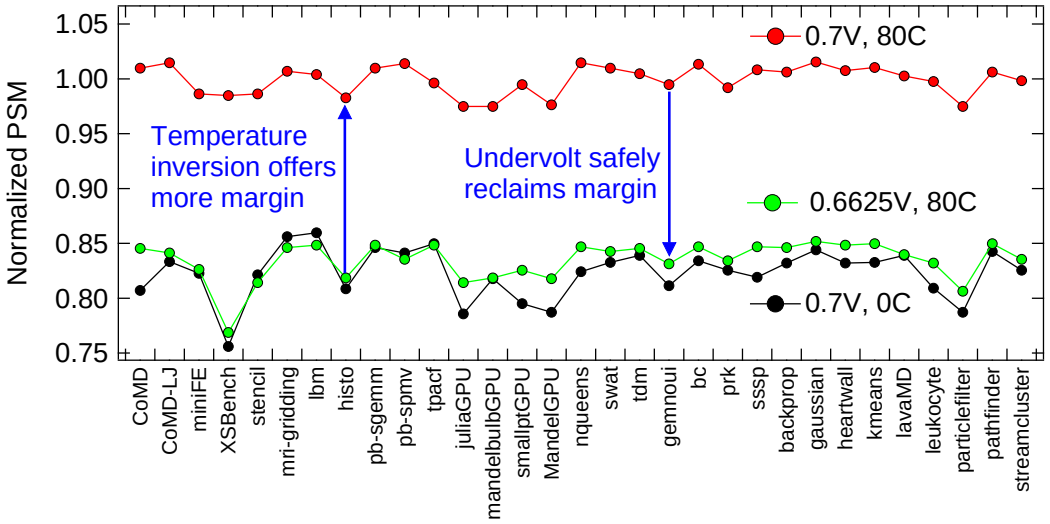


Figure 3.9: T_i-state undervolting decision at 80°C closely tracks the "golden" reference runs' timing margin, which is needed for reliability.

voltages with less timing margin run the risk of hampering the timing safety under potentially worst-case workloads.

Figure 3.9 verifies Algorithm 1's T_i-state selection at 80°C. At 0.7 V, going from 0°C to 80°C offers more than 15% extra timing margin. After voltage reduction, the workload timing margins closely track the golden reference with some workloads showing slightly higher margin.

Figure 3.9 proves yet another important point. It shows that the voltage explored using a small set of training workloads can be safely applied to future unknown workloads. The reason that the approach we present works in practice is because the extra margin that arises from temperature inversion is mainly a device property and it is workload-independent.

	20℃	**40℃**	**60℃**	**80℃**	**100℃**
693.75mV	3.7%	-	-	-	-
687.50mV	**2.2%**	-	-	-	-
681.25mV	8.4%	**2.3%**	-	-	-
675.00mV	13.9%	5.3%	4.9%	-	-
668.75mV	-	9.5%	**2.5%**	-	-
662.50mV	-	13.5%	6.5%	**1.9%**	-
656.25mV	-	-	12.2%	5.6%	9.9%
650.00mV	-	-	-	9.3%	**5.1%**

Table 3.1: PSM error compared to the reference setting for different $<$ $temperature, voltage>$ configurations.

3.3.2 Evaluating Ti-State's Voltage and Power Reduction Effects

Algorithm 1 will repeat the same process at different temperatures. Using results similar to Figure 3.8 and Figure 3.9, our methodology will eventually construct a temperate-voltage pairing table with all the proper T_i-states. Table 3.1 shows the measured results on our A10-8700P processor for 20℃, 40℃, 60℃, and 80℃. For each temperature, there is one voltage that has the smallest deviation from the "golden" reference margin, as highlighted and bolded in the table. These points are selected as the final T_i-states for the processor to use.

T_i-state table construction would add little overhead to existing silicon test procedures. Per-bin or even per-part characterization is already an industry-standard practice, especially for the high-end server market sector. Therefore, we believe that T_i-state table construction is a practical approach.

At runtime, the power management scheme can use temperature sensor

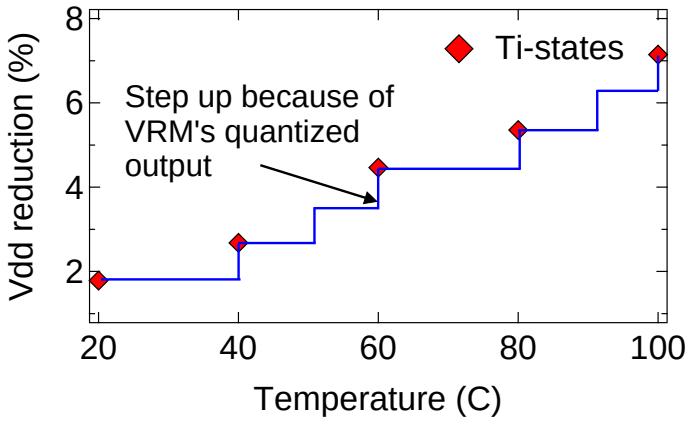


Figure 3.10: V_{dd} reduction due to T_i-states. The line corresponds to the VRM's quantized output values.

data to index into a T_i-state table and determine a suitable supply voltage [95]. In our work and the restricted scope of this paper, T_i-states are constructed for the GPU clock frequency of 300 MHz. In practice, however, the T_i-state table can be constructed across different frequencies, and the power management unit can index into the right table by frequency during runtime.

We use a representative subset of all workloads to evaluate T_i-state's power reduction at different temperatures. We start with Figure 3.10, which shows the V_{dd} reduction at various T_i-states. One temperature range corresponds to one voltage and this is because of the VRM's quantized output. To make the VRM reduce voltage by one step, the temperature has to be high enough to speed up the circuit beyond the current point. Between 20℃ and 40℃, the VRM can reduce V_{dd} by exactly one step, yet from 40℃ to 60℃ there are two VRM steps in between. The results show that V_{dd} reduction

is larger at a higher temperature because the extra timing margin offered by temperature inversion is larger than at a lower temperature.

In Figure 3.11 we compare the average power savings of the various GPU workloads as a result of the V_{dd} reduction at different temperatures. We set the die temperature manually using our temperature control setup to 40℃, 60℃, and 80℃ to mimic the various temperature conditions that the processor typically faces. We manually set the temperature because the GPU on the A10-8700P does not heat up the chip often in the voltage region we study, which limits the temperature range we can use to thoroughly characterize. Therefore, rather than examine the workloads under a "free run," we interject with external temperature control. But on the more high-end and power-hungry server parts, the GPU would hit the higher temperatures we are characterizing.

An added benefit of temperature control is that it facilitates controlled and repeatable experiments. Our choice of temperatures is reasonable because, usually, for a high-end cooling system that has around 0.2℃/W ambient-silicon thermal resistance, a workload consuming 60 W will have a steady state temperature of 40℃. For a less capable 0.5℃/W cooling system the same workload will stabilize around 60℃ [93, 41, 30]. So we cover different cooling options.

Figure 3.11 shows that on average the T_i-states can save 6.2%, 9.5%, 12.2% power at 40℃, 60℃ and 80℃, respectively. The power saving primarily comes from dynamic power reduction. Leakage power consumption also reduces at lower voltages, but only by a little. At each temperature, the relat-

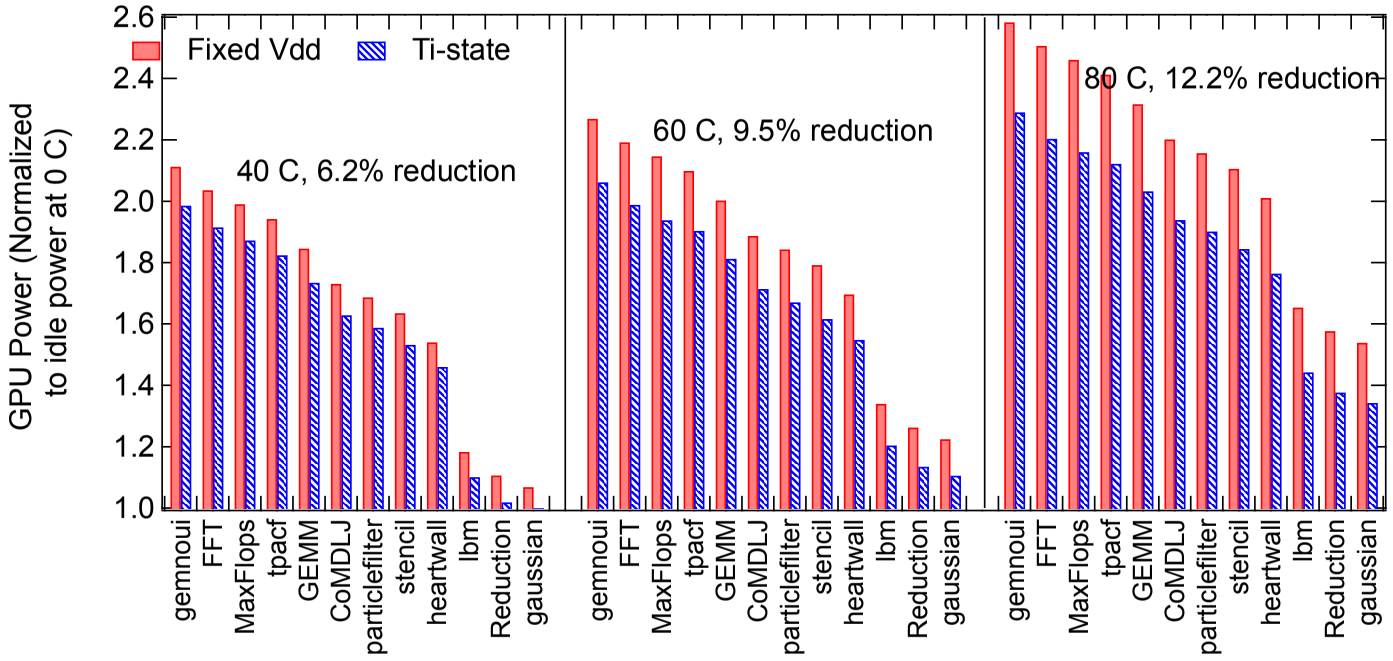


Figure 3.11: Power saving increases at higher temperatures. We mimic workload temperature by externally controlling die temperature to a 40°C – 80°C range. T_i-state's power reduction is independent of the workload activity.

ive power saving does not vary much between different workloads, but this is to be expected because T_i-state's voltage reduction is workload independent. Hence, the relative dynamic power saving for each workload should stay the same for each temperature. In practice, different workloads stabilize at different temperatures at runtime, and T_i-state will reduce the operating voltage accordingly. When the temperature varies under workload phase changes, a VRM can index into T_i-state table in real-time and adjust the supply voltage step by step [95].

3.4 Managing Ti-State Processors in Advanced Technology Nodes

In this section, we compare and contrast the benefits of T_i-state's power savings on traditional planar bulk CMOS versus the more recent FinFET and FD-SOI process technologies. FinFET is already present in latest processors [46, 90] at the time of this proposal, and both technologies will be more broadly adopted in the coming years [107, 72, 61, 62]. Because we do not have access to a FinFET or FD-SOI processor to continue our measurement-based study, we scale our measurement results to these technologies. We first explain our scaling approach for FinFET and FD-SOI, then we detail a careful analysis of T_i-states in these technologies to show that T_i-states may promise an important trade-off between leakage and dynamic power consumption. Finally, we discuss a runtime power management control loop to minimize power consumption under T_i-states (Chapter 3.4.3).

Scaling setting	Leakage power	Dynamic power	Dynamic-leakage Power scale ratio
A	0.1	1.5	15 (aggressive)
B	0.1	1	10 (test-chip [82])
C	0.2	1.5	7.5 (modest)
D	0.2	1	5 (modest)
E	0.2	0.6	3 (conservative)

Table 3.2: FinFET and FD-SOI scaling settings: for completeness, we scale dynamic and leakage power with different factors to cover both aggressive and conservative scenarios.

3.4.1 Scaling to FinFET and FD-SOI

FinFET and FD-SOI technologies can potentially alter high temperature impact total processor power because these technologies' dynamic-to-leakage power ratios are very different from traditional planar bulk CMOS. Here, we set up five reasonable scaling scenarios (ranging from aggressive to conservative leakage reductions) based on lessons from a 14 nm FinFET NTC prototype chip [82] as well as prior report [77]. Compared to 28 nm planar bulk CMOS, FinFET can reduce the off-current (I_{off}) by more than 10× under the same supply voltage for all device types, and FD-SOI can achieve even more leakage reduction. We mimic this scenario as setting B in Table 3.2. Furthermore, the FinFET test chip runs at 650 MHz at 0.55 V [82], over 2× of the 300 MHz frequency we study at 0.7 V. In setting A, we scale dynamic power by 1.5 to simulate possible dynamic power changes.

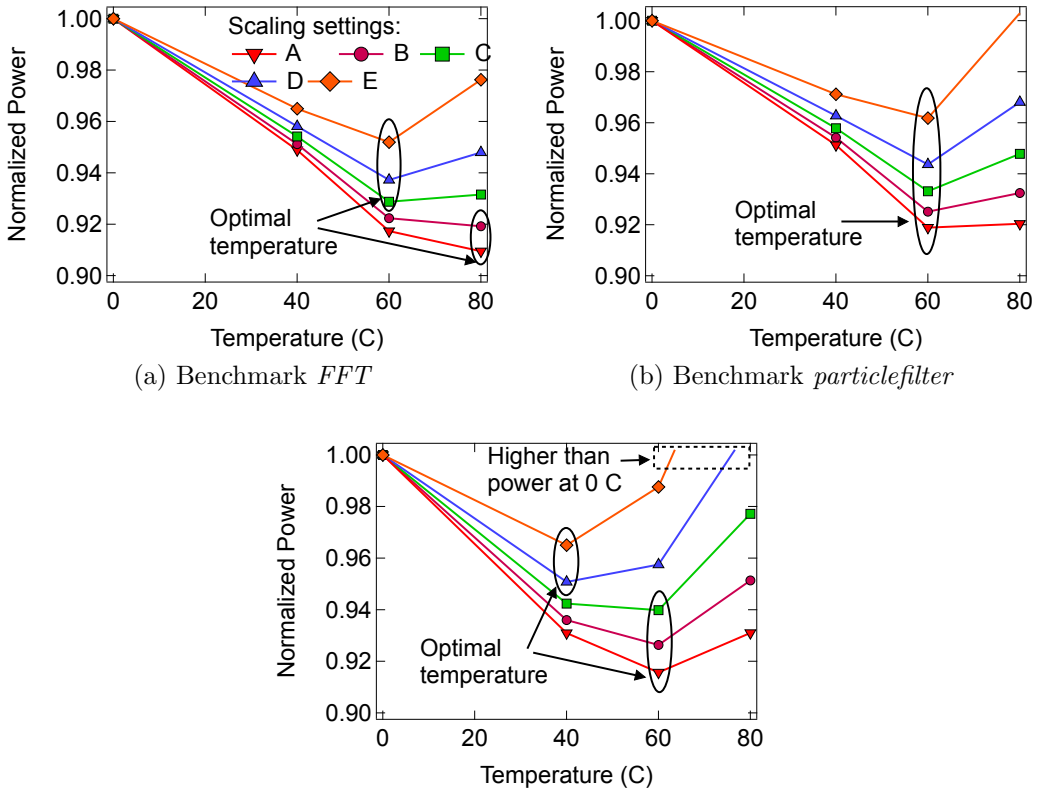
Setting C, D, and E account for possible FinFET threshold voltage engineering by modestly scaling leakage power by 0.2. Setting C mimics a

performance-centric scenario where lower threshold is utilized for higher frequency. We include setting E as a conservative scenario where dynamic power reduces with lower supply voltage. Overall, scaling setting A is an aggressive projection for FinFET, but it is a good example of FD-SOI's application scenario. Setting B reflects FinFET and FD-SOI's leakage power reduction capability, while settings C and D represent FinFET's more realistic use cases.

Temperature inversion will continue to exist in FinFET and FD-SOI. Prior work concludes FinFET processors will entirely work in temperature inversion range [53, 15], and its inflection voltage will be around the same as we measure in 28 nm [53]. Therefore, we assume the same T_i-state's voltage and power reduction capability within these technologies.

3.4.2 T_i-state Power Analysis under FinFET and FD-SOI

Thus far, we have shown the total power savings from T_i-state as a result of voltage reduction under a particular temperature level, which is set by the thermal headset. However, the high temperature still increases leakage power exponentially, especially in planar bulk CMOS technology, which is against the dynamic power savings from T_i-state with voltage reduction at high temperature. The overall effect of these two opposite factors in bulk CMOS is that processors still favor lower temperature for power reduction.



(a) Benchmark *FFT*

(b) Benchmark *particlefilter*

(c) Benchmark *Reduction*

Figure 3.12: Power versus temperature at different scaling factors for different workloads. In FinFET and FD-SOI, T_i-state makes GPU power smaller at high temperature. The optimal temperature is different for the workloads and the different scaling settings, and this is because the ratio of static to dynamic power across the workloads varies.

In FinFET and FD-SOI, the scenario above will fundamentally change. FinFET and FD-SOI have much less leakage power, therefore the leakage power increase has a smaller effect on overall processor power under higher temperature. The opposite side is the more salient dynamic power improvement caused by T_i-state's voltage reduction. These two opposite trends form

a trade-off: an optimal temperature may exist where T_i-state's dynamic power reduction balances leakage power increase at higher temperatures and the overall processor power is minimized. Carefully evaluating this trade-off is crucial for T_i-state to be practical in runtime processor temperature and power management control.

We examine T_i-state's power benefits on FinFET and FD-SOI for three different types of workloads that are representative of different typical dynamic-to-leakage power ratios. The workloads include *FFT*, *particlefilter* and *Reduction*, going from high to low dynamic power consumption. Figure 3.12 shows T_i-state's GPU power under different scaling settings. Power is normalized to 0℃ to show how power scales as temperature increases.

Figure 3.12a shows that when the dynamic power is more dominant in settings *A* and *B* then *FFT* prefers to stay at 80℃. Under more conservative settings where leakage power is higher, the temperature sweet spot drops to 60℃. In these scaling settings, FinFET's leakage power increase beyond 60℃ is more than T_i-state's dynamic power reduction.

For medium dynamic power, Figure 3.12b shows that *particlefilter*'s temperature sweet spot is around 60℃ for the scaling ratios. *Particlefilter*'s dynamic power is not high enough to make T_i-state's power saving override leakage power at 80℃.

In contrast to *FFT* and *particlefilter*, the workload *Reduction* does not consume much dynamic power. Figure 3.12c shows that it prefers to stay at

a lower temperature to minimize leakage power. Its dynamic power occupies a smaller portion of total power, therefore T_i-state's power reduction has a lesser effect. In the optimistic scaling settings A and B, *Reduction*'s sweet spot temperature is 60℃, whereas, in conservative settings D and E, the optimal temperature is at 40℃ to avoid the exponential leakage power at a higher temperature.

In general, Figure 3.12 shows that when leakage power is less prominent (i.e., leakage scaling is more aggressive in Table 3.2), T_i-states have higher power saving and the optimal temperature is also higher. With smaller leakage, dynamic power occupies a larger portion of the total power, which is when T_i-state's improvement has a bigger power saving impact. In the extreme assumption where leakage power is completely agnostic of temperature, T_i-state would prefer to operate at the highest allowed temperature to maximize the magnitude of voltage reduction from temperature inversion.

We also find when the optimal temperature is higher, the corresponding optimal power tends to be lower as well. T_i-state's power saving capability increases with higher temperature. When a workload has a larger share of dynamic power and prefers to run under a higher temperature, T_i-state's higher power saving manifests as total power improvement.

Another observation that we can make from Figure 3.12 is that high-power workloads typically have higher temperature sweet spots. For such workloads, the dynamic power is more dominant than the leakage power. Therefore, in such scenarios, for a given temperature, the percentage of dy-

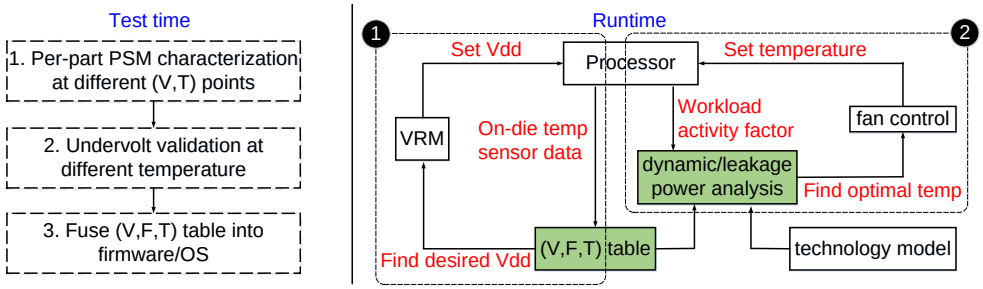


Figure 3.13: T_i-state temperature and voltage control: two loops work in synergy to minimize power. Loop 1 is a fast control loop that uses T_i-state table to keep adjusting voltage in response to silicon temperature variation. Loop 2 is a slow control loop that sets the optimal temperature based on workload steady-state dynamic power profile.

namic power saving from T_i-state contributes more to the bottom-line.

3.4.3 Runtime Temperature Control

We notice that different temperature sweet spots under all workloads and scaling scenarios are essentially a result of processor's dynamic-to-leakage power ratio. To leverage this fact, we propose a set of temperature and voltage control algorithms in Figure 3.13 to steer future FinFET and FD-SOI processors for maximum power efficiency. The solution consists of two stages: test-time and runtime.

At test time, the methodology described in Algorithm 1 establishes T_i-state's temperature-voltage tables. The process starts with characterizing the circuit speed behavior with on-chip timing sensors like the PSM, which are subsequently verified by workload timing margin measurements as we described earlier. The final temperature-voltage table can be fused into firmware

for runtime lookup. For each chip, we envision less than 40 entries to be added in total. Constructing such tables is already in practice [95]. It only extends the existing test flow by a few steps and adds minimal overhead.

At runtime, two loops work in synergy. Loop 1 is a fast loop that addresses quick yet small temperature variations from workload phase changes. It measures silicon temperature and index into T_i-state table in real time to get and set the desired voltage, similar to a typical DVFS table lookup. We envision this loop to occur at millisecond-level granularity, as in with other systems [54]. Loop 2 is a slow control loop that monitors the workload's average activity factor over a longer time period to estimate its dynamic-to-leakage power ratio. This ratio is used to find the optimal temperature in Figure 3.12, and hence discovers the T_i-state's optimal long-term average voltage.

We envision that loop 2 will target the average power savings over a relatively long time (seconds or longer). This is because runtime temperature control by adjusting the cooling system is a relatively slow process. Many of today's workload have steady state behavior suitable for this behavior, such as scientific and deep learning applications, as well as web service workloads that have diurnal patterns [64]. Thus, it is feasible to enable power saving in this scenario.

3.5 Related Work

Temperature inversion has been reported for CMOS devices long before [75, 6, 21, 105]. These works address the reason for this phenomenon, largely at the device level. Recent works study temperature inversion in FinFETs [53, 15]. Our work, however, is the first to systematically measure and characterize temperature inversion under 28 nm process and discuss its implications to the architecture and its power management.

Adaptive voltage setting for temperature variation has been recently proposed [95]. T_i-states work in a similar way to the lookup table that the authors propose. However, our work focuses on the temperature's effect in the inversion region and provides an in-depth analysis, while the solution in [95] mixes process and temperature variation together. Moreover, prior work does not address the implications of temperature control in future technologies, as we do with our FinFET analysis.

Active timing guardband management using on-chip sensors has been recently proposed [54, 115]. These prior works focus mostly on transient di/dt droop and its effect on the timing margin. In contrast, we use PSMs to characterize temperature inversion and its effect on the timing margin. We also study temperature inversion's effect in an integrated manner with di/dt droop and discuss the relationship between the two.

Many papers have addressed architecture-level temperature management [93, 41, 30, 83]. These works try to avoid excess high temperature.

But we demonstrate experimentally how temperature inversion can make high temperature a friendly environment for runtime power management.

Chapter 4

Voltage Noise-aware Scheduling for Power Reduction on Adaptive Clocking Systems

Unlike temperature variation, pipeline circuit's timing uncertainty caused by voltage variation/noise typically happens very fast, at the order of tens of cycles, and have a large magnitude, reaching over 10% of total supply voltage under worst cases, i.e., the di/dt effects [89]. Other effects that make supply voltage deviate from standard also contribute to timing margin, such as the IR drop across the power delivery network (PDN).

To safely combat voltage noise while successfully reducing the margin, active timing margin that dynamically provisions timing margin in response to the degree of voltage noise must act very promptly. A hardware-centric active timing margin solution is a natural fit for this problem, as is the practice in many recent chips [52, 54, 14, 35, 100, 13]. These chips often feature *Adaptive Clocking*, or *Adaptive Instruction Throttling* [104] techniques, where core frequency, or core front-end instruction issue rate is adaptively lowered when a low voltage is detected by sensors. The response time of the control loop is at the level of several cycles, guaranteeing time margin safety.

In this dissertation, we make a detailed, full-system analysis of the ad-

aptive clocking style, active timing margin hardware designed specifically for tolerating voltage noise. Using measurements and running real-world workloads, we study the factors that affect these processors' behavior. Using a fully built production POWER7+ system, we systematically characterize the benefits and limitations of active timing margin in terms of multicore scaling and workload heterogeneity. In our analysis, we cover both active timing margin's undervolting and overclocking modes to fully characterize the system effects under different usage scenarios.

We find when only one core is active, the current hardware active timing margin schemes can efficiently turn the underutilized timing margin into significant power and performance benefits while tolerating voltage swings. However, as more cores are progressively utilized by a multithreaded application, the benefits of active margin begin to diminish in both power and performance improvements. Using POWER7+'s sensor-rich features, we systematically characterize and decompose the on-chip voltage drop that affects the active timing margin's efficiency into its different components, and analyze the root cause of the problem. Under heavy load, the IR drop across the chip and the voltage regulator module's (VRM) loadline effect limit active timing margin's ability to the point of almost no benefit.

The magnitude of the efficiency drop aforementioned, however, varies significantly from one workload to another. Thus, given the workload sensitivity of hardware active timing margin techniques, and the long-term nature of the observed effects, we introduce the notion of voltage noise-aware schedul-

ing. The intent behind our scheduling proposal is to compensate for active margin's inefficiencies in system software. The remainder of this section is structured as follows: Chapter 4.1 provides background for the POWER7+ architecture and its implementation of active timing margin for voltage variation. Chapter 4.2 characterizes active margin's limitations when scaling up the number of active cores under different workload scenarios. Chapter 4.3 analyzes the root cause of the active timing margin's behavior as seen in the previous section. Chapter 4.4 proposes active timing margining scheduling to improve POWER7+'s efficiency when the load is light versus heavy. Chapter 4.5 compares our work with prior work.

4.1 Active Timing Margin in the POWER7+ Multicore Processor

The POWER7+ is an eight-core out-of-order processor manufactured on a 32-nm process. It supports 4-way simultaneous multithreading, allowing a total of 32 threads to execute simultaneously on the system [66]. The server runs Redhat 6.4 operating system, and all workloads are compiled with GCC 4.8.5.

A POWER7+ processor has two main power domains, each with its own on-chip power delivery network (PDN). The V_{dd} domain is dedicated to the logic circuits in the core and caches, and the V_{cs} domain is dedicated to the on-chip storage structures [117, 5]. The PDNs are shared among all eight cores to reduce voltage noise [48]. In our study, we primarily focus on voltage

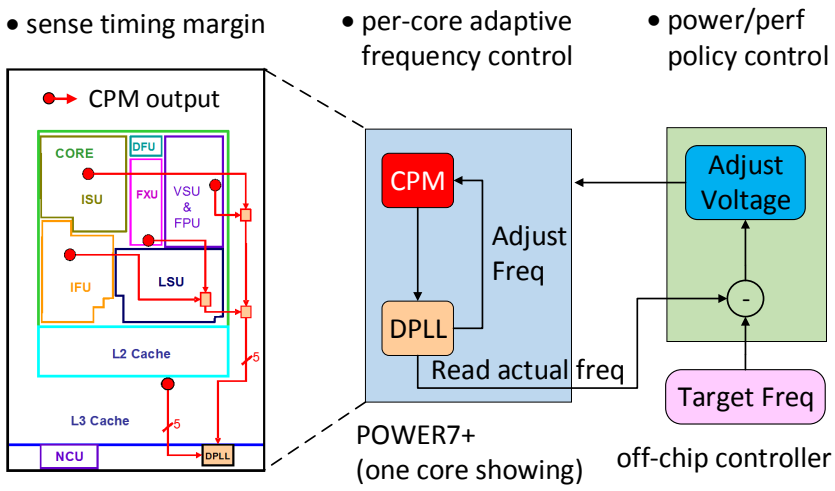


Figure 4.1: In POWER7+, Critical Path Monitor (CPM), Digital Phase Locked Loop (DPLL), and off-chip voltage controller work synergistically to let active timing margin provide just enough margin [55].

noise and power under the logic circuit's power domain as it is the main power consumer.

The processor supports both coarse-grained and fine-grained power management. Coarse-grained power management includes per-core power gating to reduce idle power consumption, and dynamic and voltage frequency scaling (DVFS) which adjusts p-states from 2.1 GHz to 4.2 GHz in 28 MHz steps by controlling V_{dd} with a static timing margin. Fine-grained power management is the active timing margin that further tunes V_{dd} and frequency around each p-state.

POWER7+'s active timing margin features adaptive clocking to tolerate circuit timing emergencies caused by di/dt effects [54, 55, 33]. Although the implementation of active timing margin for voltage noise can vary from one

platform to another [31, 101, 52, 54, 14, 35, 100, 13], the general building blocks and principles largely remain the same, and consists of three parts: (1) the timing margin sensor [25, 26], (2) the adaptive frequency control loop [99], and (3) the overclocking/undervolting policy controller, as depicted in Figure 4.1.

Timing Margin Sensor is the basis of adaptive clocking by monitoring the excess timing margin and driving frequency adjustment. In the POWER7+, the timing margin sensor is implemented as a Critical Path Monitor (CPM). A CPM mimics real circuit delay with a set of synthetic paths and monitors the timing slack after the synthetic paths complete execution. On each cycle, a signal is launched through the synthetic paths and into an inverter chain, similar to the Power Supply Monitor described in Chapter 3.1.3. When the next cycle arrives, the number of inverters the edge has propagated through in the edge detector corresponds to the CPM output, ranging from digital value 0–11 which corresponds to the position of the edge in the inverter chain and directly tells how much margin is available.

Because voltage noise and other effects that cause timing variation has spatial characteristics as illustrated in Figure 4.2, POWER7+ allocate 40 CPMs distributed across the chip to provide chip-wide, cycle-by-cycle timing margin measurement. Each core has five CPMs, integrated inside the instruction fetch unit, instruction scheduling unit, fixed point unit, floating point unit, and last level cache as shown in Figure 4.1. The worst of the five CPM measurements is reported every cycle.

Adaptive Frequency Control Loop is a hardware loop that operates

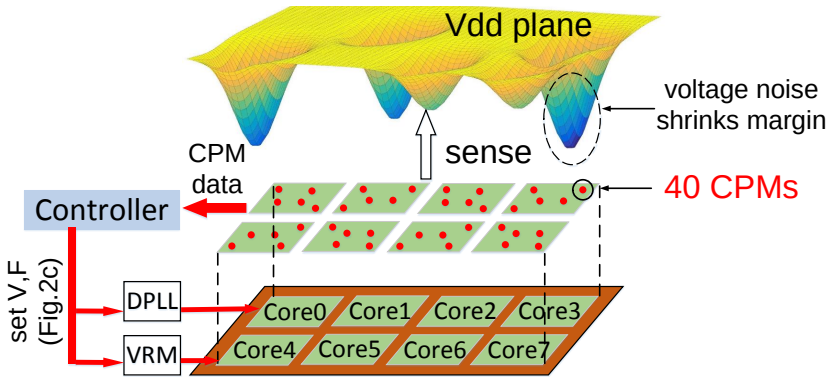


Figure 4.2: Critical path monitors (CPMs) are distributed across the chip to measure spatially variant timing margin consumption, caused by local voltage noise and other system effects.

between the timing margin sensor and an agile clock generator. Each cycle, the measured timing margin is sent to the clock generator, which compares the margin against a preset threshold and adjusts the clock frequency at very fast and short intervals.

In POWER7+, the per-core DPLL frequency control lets the processor tolerate transient voltage droops by reducing clock frequency for each core with no impact on other cores. The DPLLs can rapidly adjust the frequency, as fast as 7% in less than 10 ns [103], while the clock is still active; thus, the processor can tolerate transient voltage droops. Every cycle, the lowest-value CPM in each core is compared against the calibration position. In response, the DPLL will slew the clock frequency up or down to control the timing margin to the calibrated amount.

Off-chip Voltage Control determines whether to turn the reclaimed timing margin into power savings via undervolting or into higher performance

via overclocking based on user preferences. Often the goal is to reach a certain frequency target and convert the remaining timing margin into power savings. In the overclocking mode, the CPM and DPLL hardware form a closed-loop controller. At the fixed nominal voltage, the DPLL continuously adjusts frequency on the basis of the CPM's timing sense to operate at the calibrated timing margin. In the undervolting mode, the firmware observes CPM-DPLL's frequency and over a longer term (32ms) adjusts the voltage to make clock frequency hits the target. In this chapter, we investigate optimizing power in the undervolting mode, while in Chapter 5 we discuss performance management issues in the overclocking mode

4.2 Efficiency Analysis of Active Timing Margin on Multicore

Most prior art studied the benefits of mitigating voltage variation and reducing timing margin at the circuit- [52, 14, 35, 100, 13] and architecture levels [54, 88, 39, 78, 89, 8] using homogeneous single-core workloads. This thesis focuses on understanding the efficiency of active timing margin on a multicore system, specifically as the system activity (i.e., core usage) begins to increase using real workloads.

Using an enterprise-class server (Chapter 4.2.1), we characterize the efficiency of active timing margin at the system level. In particular, we measure, analyze and characterize active timing margin's effectiveness under different architectural configurations and workload characteristics. We make two fun-

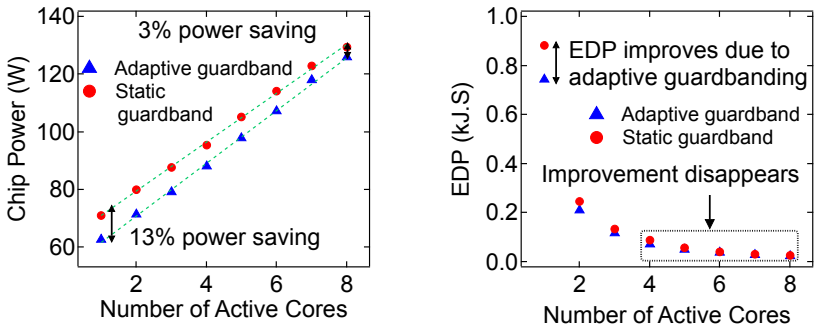
damentally new observations about the effectiveness of active timing margin on a multicore system. First, the efficiency of the active timing margin can diminish as the number of active cores increases (Chapter 4.2.2). Second, the inefficiency is highly subject to workload characteristics (Chapter 4.2.3).

4.2.1 Experimental Infrastructure

We perform our analysis on a commercial IBM Power 720 Express server (7R2) that has two POWER7+ processors on the motherboard. The processors share the main memory and other peripheral resources, such as storage and network. We focus on one of the two processors, although we validated our conclusions by conducting experiments on the other processor as well. Unless stated otherwise, the first processor is configured to idle and runs background tasks. The system runs RedHat Enterprise Linux, configured with 32 GB RAM.

We use PARSEC [11] and SPLASH-2 [106, 10] in this section because they are scalable workloads and we need to the control the applications' parallelism to carefully study the impact of core scaling. The workloads run four threads on each core to maximize hardware utilization.

We characterize the efficiency of active timing margin across two modes of operation: 1) undervolting to reduce power consumption and 2) overclocking to boost performance. Hooks in the firmware let us place the system in either operating mode. The hardware and firmware autonomously select frequency and voltage depending on the configured operation mode.



(a) Power saving.

(b) Energy reduction.

Figure 4.3: Active timing margin can save power effectively. However, the benefits decrease as more cores are used to actively run the application.

4.2.2 Core Scaling

Using *raytrace* from PARSEC (as an example), we show active timing margin's impact on processor's power consumption. We study both average chip power consumption and total CPU energy savings using Figure 4.3. We find that active timing margin is always effective at improving the performance or lowering power consumption. However, it cannot always scale up efficiently with more cores.

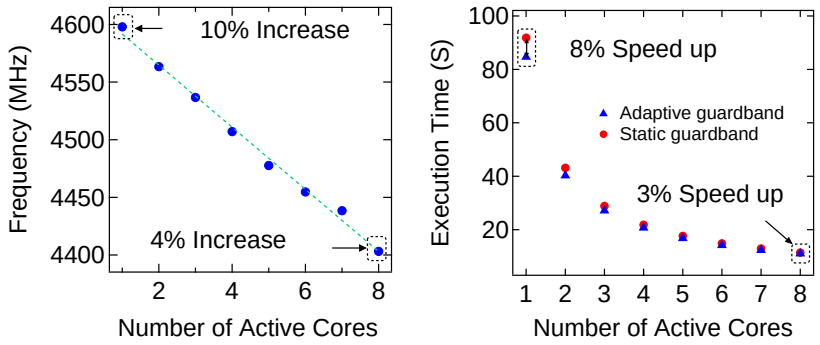
Figure 4.3a shows the program's power consumption as we use more cores, i.e., more threads to process the workload. We measure the microprocessor V_{dd} rail power by reading physical sensors available on the server, which represents most of the total processor power. In the undervolting mode, active timing margin turns the unused margin into energy savings by scaling back the voltage, which reduces unnecessary power consumption. When one core is active and the others are idle, active timing margin reduces the average power

consumption by 13% compared to no active timing margining.

Although active timing margin always saves power, a more important and crucial observation from Figure 4.3a is the decreasing power-saving trend as the number of active cores increases in the system. The power improvement from active timing margin decreases as the parallelism in the workload is (manually) increased, forcing the usage of the additional cores. Although active timing margin can save as much as 13% power when only one core is active, the savings drop sharply to about 3% when the activity scales up to eight cores.

When examining the workload's overall energy-delay product (EDP), Figure 4.3b shows notable energy efficiency improvement when only a small set of cores is actively processing the workload. However, beyond four cores, the improvement drops significantly. When only one core is active, processor energy efficiency improves by as much as 20% compared to using a static margin. But the additional improvement beyond activating more than four cores becomes negligible.

Our observations hold true for frequency-boosting as well. Active timing margin's ability to boost frequency decreases as core counts increase. Figure 4.4 shows experimental results for *lu_cb* from the SPLASH-2 benchmark suite. Compared to using a fixed target frequency of 4.2GHz under a static margin, active timing margin can achieve substantial frequency improvement, as shown in Figure 4.4a. When only one core is actively processing the workload, frequency increases by up to 10% compared to the static margin baseline.



(a) Overclocking mode. (b) Execution time.

Figure 4.4: Active timing margin can improve performance by increasing frequency. However, the overclocking benefits decrease as more cores are used.

However, when all eight cores are running the workload the frequency gain drops to only 4%.

Frequency improvement turns into program execution time speedup, especially for computing-bound workloads. For *lu_cb* the execution speedup varies gradually, decreasing from 8% when only one core is used to 3% when all cores are running the workload. This trend of diminishing benefit as core count scales up is similar to what we observe when the extra guardband is turned into energy savings for this workload.

4.2.3 Workload Heterogeneity

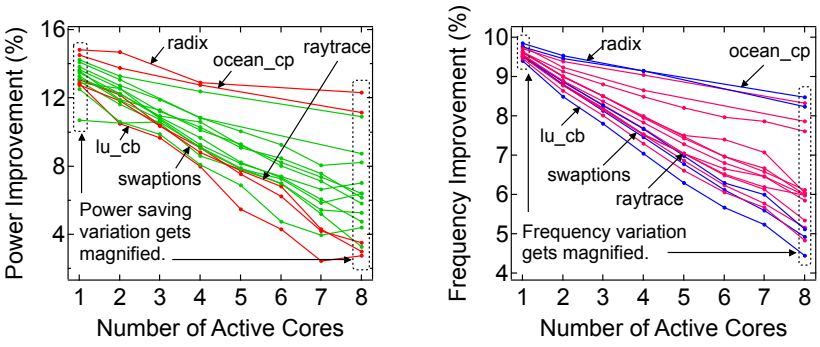
Variations in workload activity (i.e., heterogeneity) are known to strongly impact system performance from cache performance to bandwidth utilization. In this section, we demonstrate workload heterogeneity also impacts active timing margining's runtime efficiency. We focus our analysis on the

architecture-level observations and later in Chapter 4.3 we explore the causes of the observed behaviors.

Figure 4.5 shows the results for power and frequency improvement for all PARSEC and SPLASH-2 workloads compared to the same number of cores active when the active timing margin is disabled. The improvements are with respect to the system using a static guardband. The results are from two experiments, one in which the control loop is operating in energy-saving mode (Figure 4.5a) and the other in which it is operating in frequency-boosting mode (Figure 4.5b). Each line in both figures corresponds to one benchmark.

From Figure 4.5a and Figure 4.5b, we draw four conclusions. First, active timing margining consistently yields improvement, regardless of its operating mode and workload diversity. Across all of the workloads, active timing margining reduces power consumption somewhere between 10.7% and 14.8% and improves processor clock frequency by as much 9.6% on average, when one core is active. Even when all eight cores are active, improvements are at least above 4%. Power-saving improvements are slightly larger than frequency improvements because of the quadratic relationship between voltage scaling and power, as opposed to the linear relationship between frequency and power.

Second, the improvements monotonically decrease as the number of active cores increases. Across all the workloads, we observe a consistent drop in active timing margining's efficiency. The average power efficiency improvement across the workloads drops from 13.3% when one core is active to 10% when two cores are active to 6.4% when all cores are actively processing the workload.



(a) Undervolting mode. (b) Overclocking mode.

Figure 4.5: Improvements reduce at different rates for each of the PARSEC and SPLASH-2 workloads when cores are progressively activated, leading to magnified workload variation when all cores are active.

We observe a similar trend with frequency.

Third, the rate of monotonic decrease for each workload varies significantly. For instance, *radix*'s power improvement drops from 15% when one core is active to around 12% when all eight cores are active. However, in *swaptions*, the improvement drops drastically from 13% to 3%. In the frequency-boosting mode, the decreasing magnitude is slightly smaller, although the variation in improvements is still strongly present. Frequency for *radix* and *ocean_cp* almost remains unchanged at 9%, but the frequency of *lu_cb*, *swaptions* and *raytrace* drops notably from 10% to 4%.

Fourth, regardless of the active timing margining operating mode (i.e., power saving or frequency boosting), workload heterogeneity significantly impacts the mechanism's efficiency when all cores are active. This finding is especially important in the context of enterprise systems because server workloads are ideally configured to fully use all computing resources to reduce the

operator's total cost of ownership (TCO) [4].

In multicore systems that rely on active timing margining, the system's behavior will vary significantly depending on how many cores are being used and what workloads are simultaneously scheduled for execution on the processor. To prove this point, we later discuss the implications of workload co-location using our system. In the future, we suspect workload heterogeneity could be a major source of inefficiency, especially as we integrate more cores into the processor unless we identify the problem's source for mitigation.

4.3 Root-Cause Analysis of Active Timing Margin's Inefficiencies

In this section, we analyze the root cause of active timing margin's inefficiency under increasing core counts and workload heterogeneity to understand how to reclaim the loss in efficiency. We present an approach for characterizing active timing margin's inefficiency using CPM sensors (Chapter 4.3.1). On this basis, we characterize the voltage drop in the chip across both core counts and workloads because the on-chip voltage drop affects active timing margin's efficiency. Our analysis reveals that core count scaling results in a large on-chip voltage drop (Chapter 4.3.2), whereas workload heterogeneity plays a dominant role in affecting the processor's IR drop and loadline (Chapter 4.3.3).

4.3.1 Measuring the On-chip Voltage Drop

We developed a novel approach to capture and characterize active timing margin's behavior using CPMs. We use CPM output to capture the on-chip voltage drop that affects the timing margin, which in turn affects the active timing margin's efficiency. In effect, we use CPMs as "performance counters" to estimate on-chip voltage, similar to how performance counters were first shown to be useful for predicting power consumption [47, 42].

Because timing margin is determined by on-chip voltage, capturing the CPM's output would reflect the transient voltage drops between the VRM output and on-chip voltage. Low on-chip voltage leads to less time for the CPM's synthetic-path edge to propagate through the inverter chain, and thus the CPM will yield a low output value. Under high on-chip voltage, the circuit runs faster, and the CPM yields a higher output.

To read the CPMs, we disable active timing margin because it dynamically adjusts the timing margin to keep the margin small and CPMs constant. The CPMs typically hover around an output value of 2 when active timing margin is active due to CPM calibration. By disabling active timing margin, we allow the CPMs' output values to "float" in response to on-chip voltage fluctuations, and thus we can study how supply voltage affects the behavior of CPMs.

We use the IBM Automated Measurement of Systems for Temperature and Energy Reporting software (AMESTER) [32, 43] to read the CPMs'

output. We record CPM readings under different on-chip voltage levels to determine how CPM responds to different on-chip voltage. AMESTER reads the CPMs at the minimal sampling interval of 32ms, which is restricted by the service processor. AMESTER can read the CPMs in either *sticky mode* or *sample mode*. In the sticky mode, AMESTER reads the worst-case, i.e. smallest, output of each CPM during the past 32 ms, which is useful for quantifying worst-case droops. In the sample mode, AMESTER provides a real-time sample of each CPM, which is useful for characterizing normal operation.

We use CPMs in sample mode to convert their output into on-chip voltage. To minimize experimental variability, we let the operating system run and throttle each core to fetch one instruction every 128 cycles. Figure 4.6 shows the mapping between CPM output and on-chip voltage. We sweep the voltage range for all possible clock frequencies and look at the average output of all 40 CPMs over 12,500 samples, which corresponds to about 1 minute of measurement. Each line corresponds to one frequency setting, and the system default voltage levels at DVFS operating points are highlighted with the marked line. Starting from 2.8 GHz, each diagonal line, as we move to the right, corresponds to a 28 MHz increase in frequency. The rightmost line corresponds to the peak frequency of 4.2 GHz. For any frequency, the CPM value gets smaller as we lower the voltage, confirming the expected behavior that smaller voltages correspond to less timing margin. Also, for a fixed voltage (x-axis), higher frequency yields smaller CPM values (y-axis) because of less cycle time and a tighter timing margin.

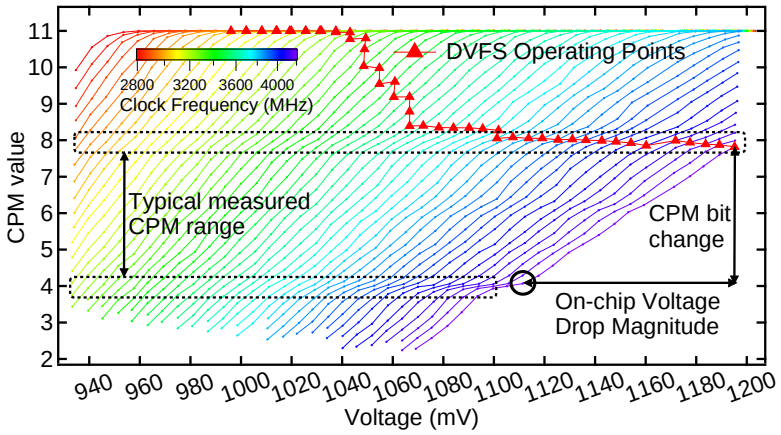
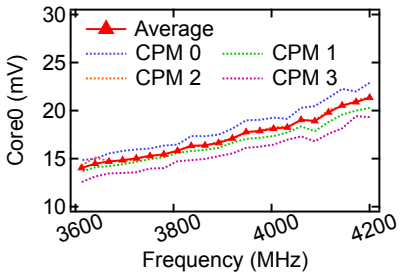


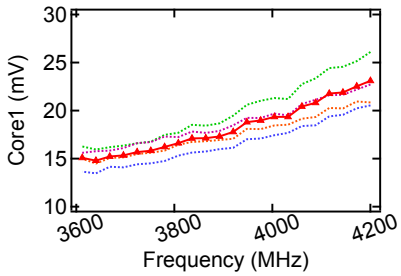
Figure 4.6: Mapping between on-chip voltage and CPM values.

Figure 4.6 lets us establish a direct relationship between CPM and on-chip voltage. We observe a near-linear relationship between the two variables under each frequency. Therefore, with a linear fit, we can determine each CPM bit's significance. On average, one CPM output value corresponds to 21 mV of on-chip voltage. On this basis, we can estimate the magnitude of on-chip voltage drop during any 32 ms interval. For instance, if the measured CPM output drops from eight to four, the estimated on-chip voltage has dropped by 84 mV.

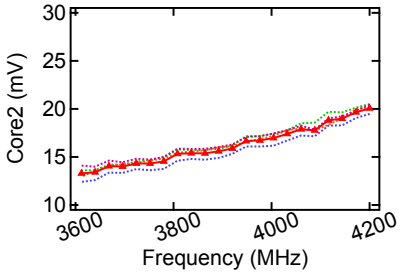
Figure 4.7 shows the sensitivity of the CPMs within each processor core. Although we see a near-linear relationship between frequency and all the CPMs, there is variation among the CPMs in each core and between cores. For instance, CPMs in Core 2, 6, 7 have steadier sensitivity compared to Core 1, 3, 5. The latter have higher distribution across CPMs. We attribute this behavior to process variation and CPM calibration error, as explained by



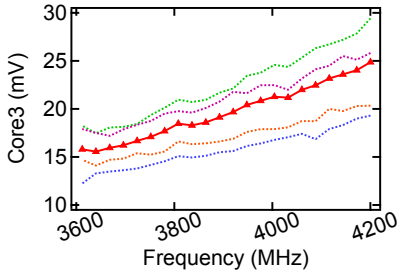
(a) p0 core 0.



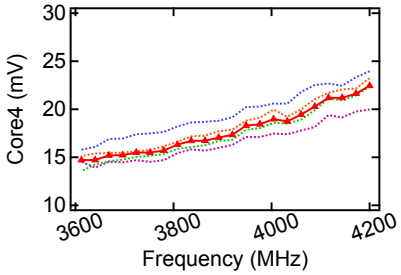
(b) p0 core 1.



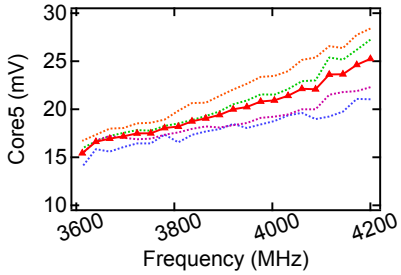
(c) p0 core 2.



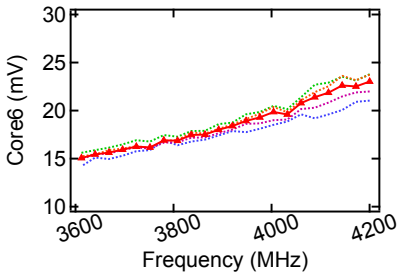
(d) p0 core 3.



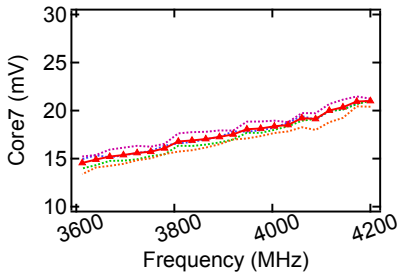
(e) p0 core 4.



(f) p0 core 5.



(g) p0 core 6.



(h) p0 core 7.

Figure 4.7: CPMs can sense the chip supply voltage with a precision of about 21mV per CPM bit at peak frequency.

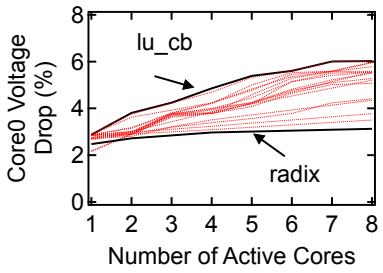
prior work [33].

To ensure the robustness of our measurement results, we considered both repeatability and temperature effects. We repeated our experiment on another socket in the same server, and the result conforms to the same trend shown in Figure 4.6. We observe that chip temperature varies between 27°C at the lowest frequency to 38°C at the highest. Internal benchmark runs show such temperature variation does not have significant influence over CPM readings, and thus we can draw general conclusions from Figure 4.6.

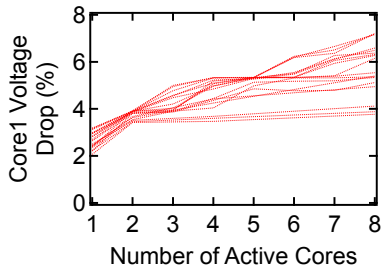
4.3.2 On-chip Voltage Drop Analysis

Using our on-chip voltage drop measurement setup, we quantify the magnitude of the on-chip voltage drop to explain the general core scaling trends seen in Chapter 4.2.2. It is important to understand what factors, and more importantly how those factors, impact the efficiency of active timing margin as more cores are activated.

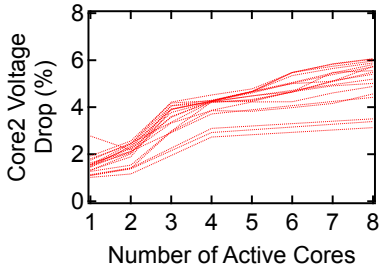
Figure 4.8 shows the measured results for the voltage drop across different cores within the processor, ranging from Core 0 through Core 7. The cores are spatially located in the same order as they appear on the physical processor [117]. The y-axis is the percentage of on-chip voltage drop from the nominal. Given the magnitude of voltage drop and knowledge about the system's nominal operating voltage, we can determine the percentage change. The x-axis indicates the total number of simultaneously active cores, specifically as they are activated in succession from core 0 to 7. Keeping consistent



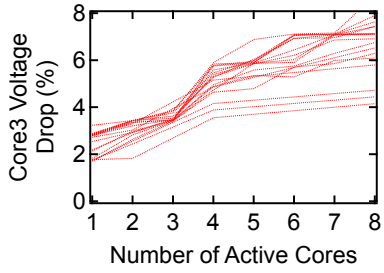
(a) p0 core 0.



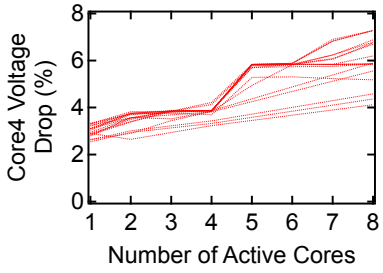
(b) p0 core 1.



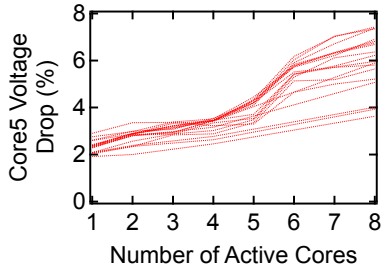
(c) p0 core 2.



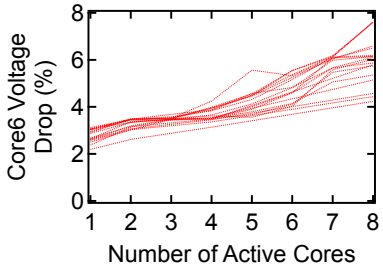
(d) p0 core 3.



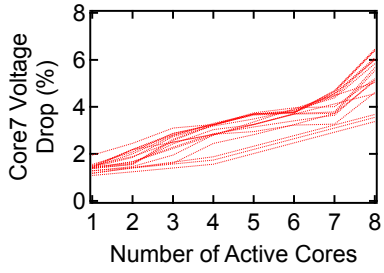
(e) p0 core 4.



(f) p0 core 5.



(g) p0 core 6.



(h) p0 core 7.

Figure 4.8: On-chip voltage drop analysis across cores under different workloads.

with Figure 4.5, each line in the subplots corresponds to one workload from PARSEC and SPLASH-2. Each subplot shows a particular core's characteristics with respect to every other (active or inactive) core in the processor.

Figure 4.8 lets us understand several important factors that affect active timing margin's efficiency. First, voltage drop increases as more cores are activated. For all workloads, voltage drop increases from about 2% to 8% as the number of active cores increases. The trend is similar to the diminishing benefits seen previously in the power and frequency improvement in Figure 4.5. As the magnitude of voltage drop increases, the timing margin decreases and thus active timing margin's efficiency decreases at higher loads.

Second, the increasing on-chip voltage drop trend manifests as chip-wide global behavior because voltage drop affects all cores at the same time, regardless of whether they are idling or actively running a workload. For instance, when cores on the upper row (Core 0 through Core 3) are actively running a workload, they experience a voltage drop. Meanwhile, cores in the bottom row also experience voltage drop even though Core 4 through Core 7 are not running any workloads.

The implications of the second finding are that global effects, such as chip-wide di/dt noise [37, 69, 8] and off-chip IR drop, can affect active timing margin's system-wide power-saving efficiency because active timing margin makes decisions on the basis of the worst-case behavior of all cores. In particular, this behavior impacts the power-saving mode because the processor has a single off-chip VRM that will need to supply the highest voltage to match

the most demanding core's voltage requirement. So, even if some cores are lightly active, the system may have to forgo their active timing margin benefits to support the activity of the busy core(s). In applications where workload imbalance exists, this can become a major efficiency impediment.

Third, the on-chip voltage drop's scaling trend, as the number of active cores increases, tends to differ across cores, indicating that voltage drop has localized behavior in addition to the global behavior described previously. For instance, across all the cores the magnitude of voltage drop shifts upward significantly whenever that particular core is activated. For instance, Core 7's voltage drop increases by 2% when it is activated, as evident in Core 7's voltage drop plot.

More generally, cores that are activated earlier have a higher voltage drop at first, and thereafter their voltage drop begins to saturate and plateau. For instance, Core 0 and Core 1 have a higher voltage drop when Core 0 through Core 3 are activated. These cores' voltage drop increase quickly when the number of active cores is less than four. On the contrary, the voltage drop for Core 4 through Core 7 does not change much while Core 0 through Core 3 are activated, but thereafter their voltage drop increases much more quickly.

Localized effects impact the operation of the per-core frequency-boosting mode. Each POWER7+ core has its own DPLL that can dynamically perform frequency scaling to improve performance when required. However, each core's performance can be boosted only when it is not affected by activity on its neighboring cores. In general, our observations imply that it is easier to

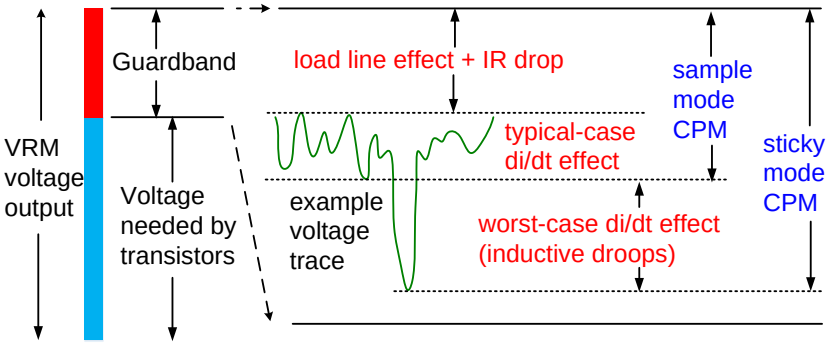


Figure 4.9: Voltage drop component analysis, including di/dt droop, IR drop and the loadline effect.

boost clock frequency and, hopefully, performance – at least for computing-bound workloads – over reducing voltage, because frequency-boosting is largely affected by localized voltage drop. By comparison, the global voltage drop typically tends to have a more pronounced effect on the chip-wide power-saving mode.

4.3.3 Decomposing the On-chip Voltage Drop

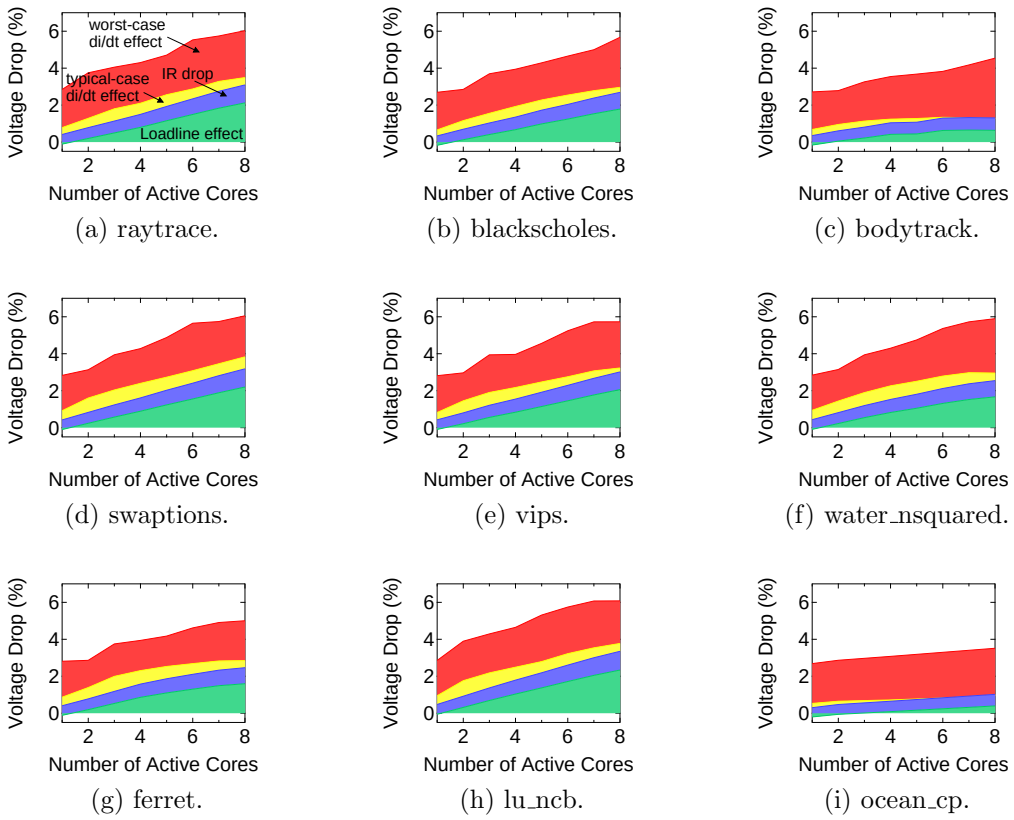
To understand how workload heterogeneity affects the power-saving and frequency-boosting modes when all cores are active, we must understand why the on-chip voltage drop varies significantly from one workload to another with an increasing number of cores. For example, in Figure 4.8 *lu_cb*'s voltage drop increases more quickly compared to *radix*, whose voltage drop does not change much as the number of active cores increases. We decompose the on-chip voltage drop into its three primary components (see Figure 4.9): worst-case di/dt noise, also called voltage droops due to sudden current surges caused

by microarchitecture activities; typical-case di/dt noise due to regular current ripples; and passive voltage drop due to IR drop across the PDN and the loadline effect [54] at the VRM.

We use a mixture of current sensing techniques and CPM measurements to decompose the voltage drop. To measure passive voltage drop (i.e., loadline effect + IR drop), we use VRM's current sensors. The IR drop and loadline effects are quantified using a heuristic equation verified against hardware measurements. The input to the equation is the current going from the VRM into the POWER7+ processor, sampled periodically.

We use CPMs to calculate the magnitude of typical and worst-case voltage noise. To get the typical di/dt value, we put the CPMs in sample mode to get real-time samples of on-chip voltage and subtract the passive component from it which represents static DC voltage drop. To get the worst-case di/dt value, we put the CPMs in the sticky mode to get the largest voltage droop seen in every 32 ms time window and subtract the sampled long-term average on-chip voltage from it.

We select several representative benchmarks from previously discussed data and decompose their on-chip voltage drop into di/dt noise and passive drop in Figure 4.10. The subplots are in the form of a stacked area chart, showing the trend as more cores are progressively activated. Only Core 0 data simplifies the presentation of our analysis, although we have verified that the conclusions described in the following paragraphs hold true for the other cores as well.



(a) raytrace. (b) blackscholes. (c) bodytrack.

(d) swaptions. (e) vips. (f) water_nsquared.

(g) ferret. (h) lu_ncb. (i) ocean_cp.

Figure 4.10: Different components of on-chip voltage drop for some PARSEC and SPLASH-2 benchmarks. In general, as more of the processor's cores are activated, voltage drop increases by varying magnitudes across workloads.

By analyzing the data, we conclude that passive voltage drop, including IR drop across PDN and VRM's loadline is the dominant factor contributing to increasing voltage drop. Intuitively, these two passive effects have the most direct influence over active timing margin's behavior because they always exist steadily during execution as compared to di/dt noise.

As we scale the number of active cores, the worst-case di/dt noise

increases slightly across all of the benchmarks, and typical-case di/dt noise decreases. For instance, the worst-case di/dt noise growth is noticeable in *bodytrack*, *vips* and *water_nsquared*. When multiple cores are active simultaneously, they can have synchronous behavior or random alignment, that can cause large and sudden current swings leading to voltage droops [89, 69, 51]. However, our droop frequency analysis (not shown here) indicates that such large worst-case droops occur infrequently. On the contrary, typical-case di/dt noise gets smaller when core count scales. With more active cores, microarchitectural activities stagger among different cores, which can lead to noise smoothing [69, 89].

Compared to di/dt noise, we find a clear scale-up trend of passive voltage drop from Figure 4.10, and it contributes most to the scale-up of total voltage drop. IR drop and loadline effects increase almost linearly with the number of active cores because the passive voltage drop is caused by processor current draw, which is further determined by chip power. When more cores are used, the whole chip consumes more dynamic power and will lead to higher IR drop and loadline effects.

Because active timing margin can deal with occasional di/dt voltage droops by slowing down frequency quickly, the rare voltage drop caused by this effect does not strongly influence the power-saving and frequency-boosting capability of active timing margin, even though they consume a significant portion of the total voltage guardband. Thus, we believe passive voltage drop is the main source of impact to active timing margin's efficiency.

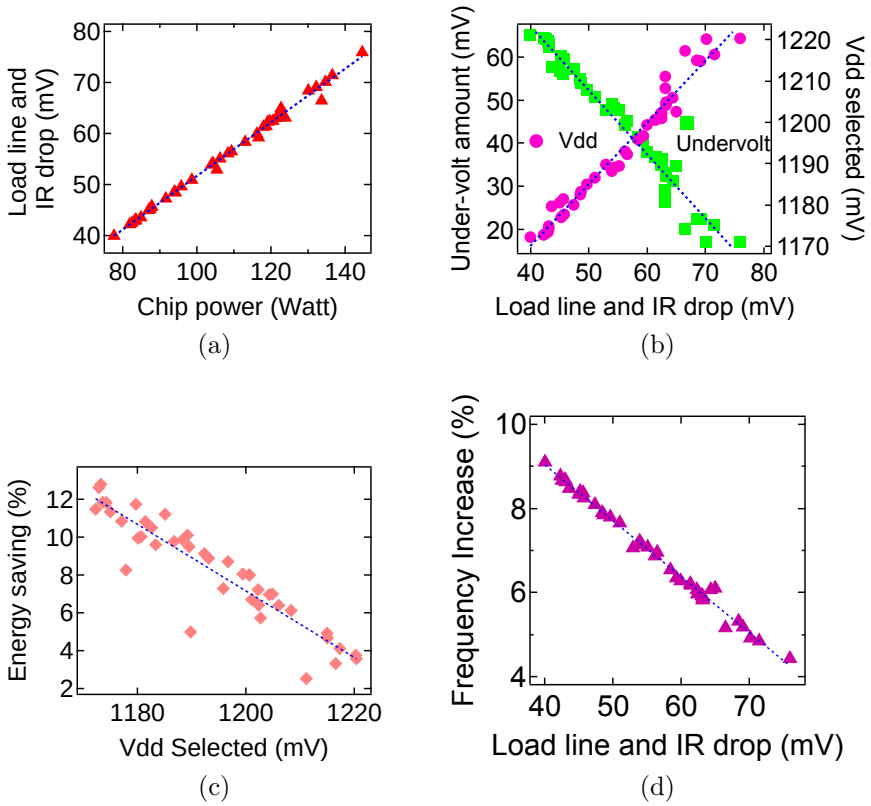


Figure 4.11: Power-intensive workloads induce large loadline and IR drop, which severely limits the active timing margin system's undervolting capability, and thus impacts the system's overall power-saving potential.

We confirm that loadline and IR drop cause active timing margin's inefficiency at full load by quantifying the relationship between their voltage drop under static guardbanding with respect to the system's two optimization modes: power saving (i.e., undervolting) and frequency boosting (i.e., overclocking). Figure 4.11 shows the causal relationship between workload power consumption, loadline and IR drop, and the active timing margin's two modes. To ensure we have enough data points, we consider 27 SPECrate workloads on

top of the existing 17 PARSEC and SPLASH-2 workloads used before. Each point represents the data we experimentally measured for one benchmark.

In Figure 4.11, across all the subfigures, we see a strong correlation between passive voltage drop and the power-saving and frequency-boosting modes. Figure 4.11a shows a strong linear relationship between power and passive voltage drop. Figure 4.11b shows when a workload has a high loadline and IR drop, the voltage guardband is highly utilized, and so active timing margin has less room for undervolting. Thus, the voltage selected by active timing margin is higher. The result is fewer energy savings for high-power workloads, as the data in Figure 4.11c demonstrates. The same holds true for active timing margin's frequency-boosting mode. Here as well, a high loadline and IR drop reduce the timing margin; thus, the DPLL has limited room left to overclock the frequency as shown in Figure 4.11d.

4.4 Voltage Noise-aware Scheduling

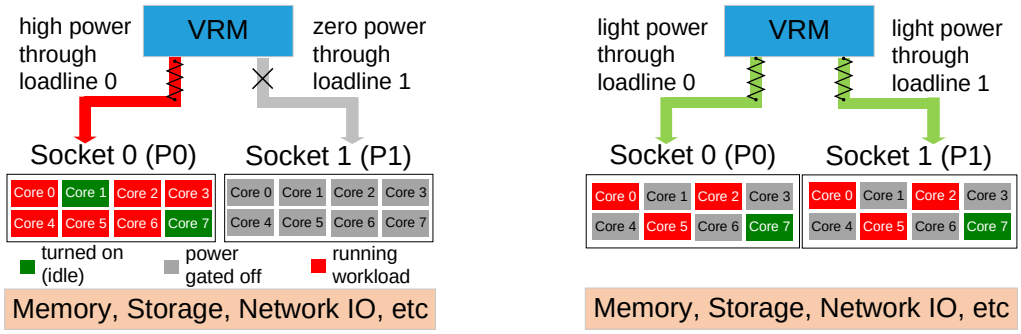
We propose system-level scheduling techniques to improve the benefits of active timing margin. Our scheduler's overarching goal is to minimize the impact that loadline and IR drop have on an active timing margin processor's power and performance efficiency. We demonstrate voltage noise aware scheduling in Chapter 4.4.1, and evaluates its effect in runtime power reduction in Chapter 4.4.2

In a multi-socket server, conventional wisdom says to consolidate workloads onto fewer processors so that the idle processor can be shut down to

eliminate wasted power [71, 64, 59]. However, this principle does not apply to servers with active timing margin and per-core power-gating capability. Our measured results show consolidation actually leads to higher power of these systems. We propose voltage noise-aware scheduling to maximize active timing margin's power-saving benefits for the underlying processors. Compared to workload consolidation, our noise-aware scheduling achieves up to 12% power savings.

4.4.1 Solution for Recovering Multicore Scaling Loss

We use Figure 4.12 to introduce how voltage noise-aware scheduling optimizes workload distribution among a server's VRM-multiprocessor subsystem. In Figure 4.12, multiple processor sockets share a common VRM chip, each with its own power delivery path from the VRM to the die. The VRM can generate multiple V_{dd} levels for different processors, which is nor-



(a) Workload consolidation.

(b) Noise-aware Scheduling.

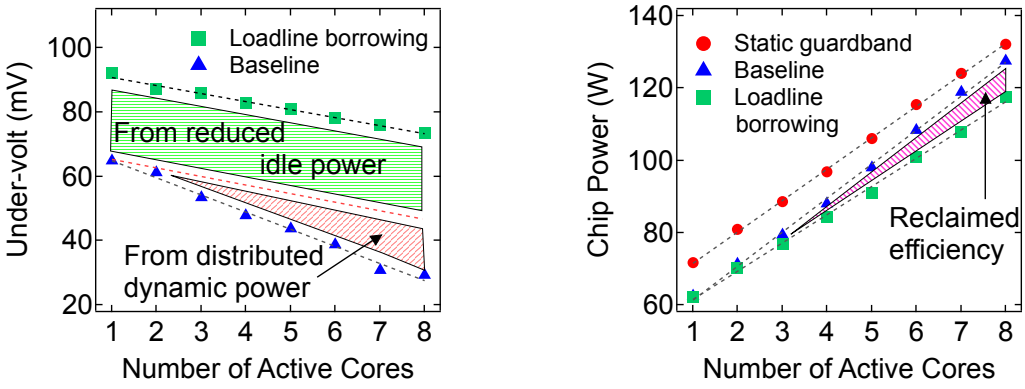
Figure 4.12: Voltage noise-aware scheduling balances workloads across multiple sockets to reduce per-socket voltage drop and create room for active timing margin.

mal for contemporary systems. In the following discussion, we use Figure 4.12a and Figure 4.12b to analyze the scenarios of workload consolidation and noise-aware scheduling and highlight the necessity of considering VRM's role in systems with active timing margin processors. Other components such as memory chips and disks are powered on steadily throughout our analysis.

Figure 4.12a shows a traditional consolidation schedule for a multisocket server. Workloads are all mapped to socket 0 so that socket 1 can be shut down. Because all power goes to socket 0, the passive voltage drop along the power-delivery path from VRM to processor 0 is very high, which limits active timing margin's potential to undervolt.

Voltage noise-aware scheduling balances workloads equally among all available sockets, and power gates off unneeded cores to eliminate idle power consumption. Figure 4.12b illustrates a loadline-borrowing schedule. In Figure 4.12b active cores are distributed to each socket evenly, and each socket power gates off a set of unused cores to achieve the same idle power elimination effect as in a consolidated schedule. In this schedule, each socket draws less power, reducing the passive voltage drop each processor experiences. This allows active timing margin to reduce more voltage from each processor and hence improve total processor power.

We use our two-socket platform to illustrate the benefits of voltage noise-aware scheduling. We compare the case of conventional workload consolidation, which places all loaded cores on one processor as the baseline, to noise-aware scheduling, which balances the loaded core count across both pro-



(a) Undervolt scaling.

(b) Power scaling.

Figure 4.13: Distributing *raytrace* across two processors reduces passive voltage drop, allowing more power saving under high core count.

cessors. We keep eight of the total 16 cores turned on to respond instantly to utilization levels of up to 50%. The remaining eight cores are assumed to be not instantly needed, and therefore are put into a deep sleep (power-gated) state. The power of the on-chip memory controller left powered on is negligible for total processor power across two sockets and total system power.

We run the workloads using one to eight cores. In the conventional case, all of the turned-on cores reside on a single processor. In the noise-aware scheduling case, each processor has four cores that are turned on and active. In either case, we measure and compare the two processors' total chip power.

As an example, Figure 4.13 shows the results for *raytrace* with voltage noise-aware scheduling. Figure 4.13a shows that noise-aware scheduling offers a better undervolting benefit no matter how many cores are used. There are two reasons. First, noise-aware scheduling lets each processor power on fewer cores, which cuts down leakage power, and thus substantially reduces the idle

power. For *raytrace*, less idle power gives 20mV more undervolting benefit when one core is active. Second, balancing application activity (threads) and system requirements (idle cores) across the processors' loadline distributes dynamic power across each processor, which further reduces the passive drop for each processor. When eight cores are active, reduced dynamic power allows an additional 20mV reduction.

Figure 4.13b shows noise-aware scheduling can reduce a significant amount of total chip V_{dd} power. The biggest effect is achieved when more cores are used. In Figure 4.13b noise-aware scheduling reduces power consumption by 1.6%, 4.2% and 8.5% when two, four and eight cores are used, respectively. The result is intuitive because each processor's passive voltage drop is reduced when fewer cores are active. Thus, distributing the workload when more cores are active yields larger benefits.

Loadline-borrowing is suitable only for workload scheduling within a multisocket server. In this setting, all other resources, such as memory, disk, and network I/O, remain active when workloads are consolidated onto a few processors. When workloads are consolidated across multiple servers, the idle power reduction from turning off the used memory and hard drive outweigh active timing margin's processor power savings. In this case, the scheduler will consolidate workloads onto fewer servers first, then on each server noise-aware scheduling can be used to further improve cluster power consumption. We leave this discussion to future studies.

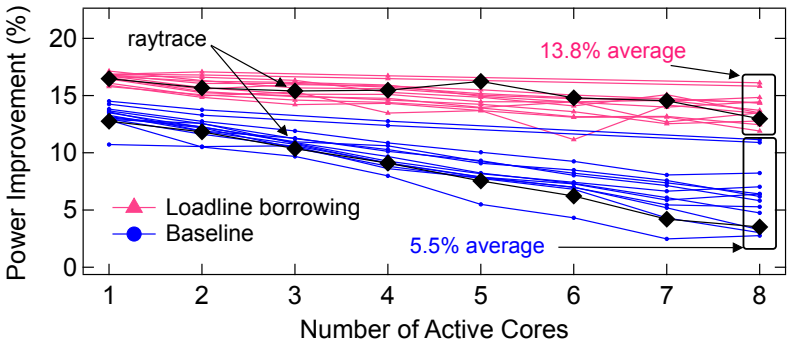


Figure 4.14: Voltage noise-aware scheduling's power and energy improvement under different numbers of active cores. Compared to the baseline, noise-aware scheduling consistently shifts up every workload's power improvement.

4.4.2 Power Reduction Improvement

Current operating systems are unaware and do not incorporate loadline knowledge into process scheduling. We use the Linux kernel's "taskset" affinity mechanism to emulate a schedule that dynamically performs noise-aware scheduling. We evaluate noise-aware scheduling on a wider set of benchmarks including all of PARSEC and SPLASH-2 workloads to capture the general trends. Briefly, the key highlight is that loadline-aware OS-level software scheduling can effectively *double the efficiency* of active timing margin at high core counts.

Figure 4.14 shows active timing margin's scaling power improvement against static guardbanding under workload consolidation and noise-aware scheduling. Ideally, active timing margin's power improvement will not scale down, and it will be identical across workloads. noise-aware scheduling approaches this goal by increasing active timing margin's power-saving capab-

ility for all active cores, shown by the clustered lines at the top of the figure. When fewer cores are active, noise-aware scheduling's power improvement comes mainly from the reduced idle power on each processor. The improvement increases when more cores are active because each chip's dynamic power also reduces when the workload is distributed. Figure 4.14 shows that on average consolidated active timing margin achieves 5.5% power improvement over static guardbanding when eight cores are active, whereas noise-aware scheduling improves by 13.8%, over 50% improvement atop the original system design.

We study more benchmarks along with PARSEC and SPLASH-2, including SPEC CPU 2006 workloads running in the form of SPECrate [18], to further demonstrate noise-aware scheduling's power and energy improvement when all eight cores are active. SPECrate is commonly used to measure system throughput, typical of evaluating performance when running different tasks simultaneously. We use 32 PARSEC and SPLASH-2 threads and eight SPECrate workload copies to match POWER7+'s eight-core architecture. The results are shown in Figure 4.15. On average, noise-aware scheduling achieves 6.2% and 7.7% reduction in processor power and energy, respectively, across the workloads. For power-intensive workloads such as *lu_cb*, noise-aware scheduling can achieve 12.7% power improvement. For all workloads, total server power improves by 2.1% on average.

A handful of benchmarks fall into one of two extremes. On one extreme, some benchmarks that are to the leftmost side on the x-axis, such as *lu_ncb* (not to be confused with *lu_cb*) and *radiosity*, suffer from severe performance

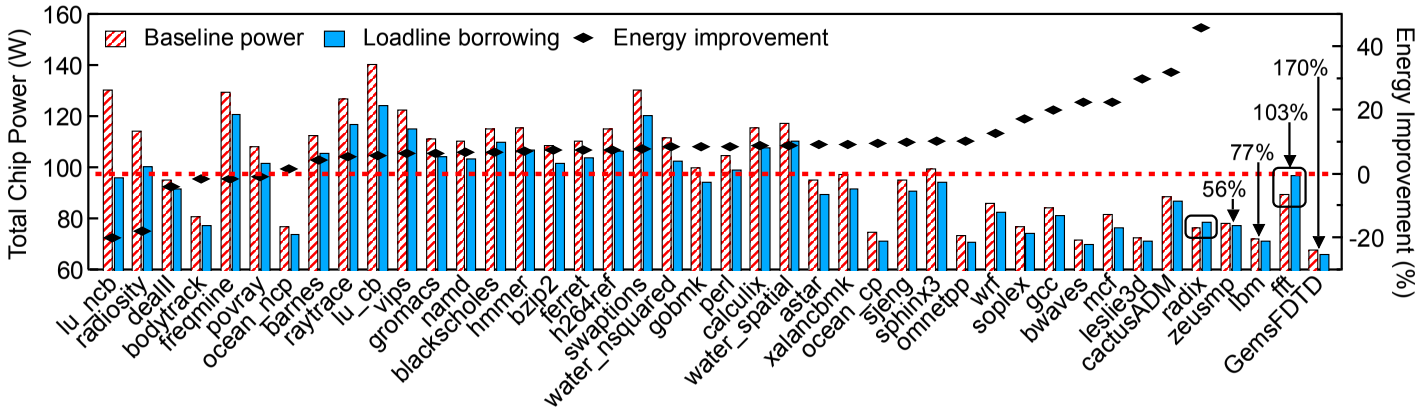


Figure 4.15: Voltage noise-aware scheduling's power and energy improvement when eight cores are active.

loss. Performance decreases by more than 20% due to interchip communication overhead (not shown). This in part leads to reduced core power consumption during noise-aware scheduling (see left y-axis), but the longer execution time negatively offsets the benefit and increases total energy consumption.

On the other extreme, some other benchmarks that are to the rightmost side on the x-axis, such as *radix*, *zeusmp*, *lbm*, *fft* and *GemsFDTD*, experience large performance improvements from load balancing because there is less memory subsystem contention. This performance improvement increases chip activity that could sometimes lead to higher power consumption than the baseline system, such as in the case of *radix* and *fft*. Nonetheless, the improved performance brings about large energy reductions for these workloads, as the right y-axis in Figure 4.15 shows. Improvements range between 50% and 171%.

4.5 Related Work

The di/dt effect and its impact on reliability has been well noted [48, 89, 51, 8]. A plethora of work aims at reducing inductive noise in microprocessors, ranging from the circuit [28, 12], architecture [36, 78, 37, 39, 38, 88, 89, 69, 108] and software [86]. These works usually require intrusive design changes to the hardware [28, 12, 39, 88] and rely on simulation, microarchitecture event detection and activity throttling [36, 78, 38, 88, 86, 69].

Unlike the prior work, we use a measurement-based approach to studying adaptive guardbanding processors [31, 101, 52, 54, 14] that handles droops in a fundamentally new way. Because adaptive guardbanding can effectively

improve efficiency and guarantee reliability at the same time, it has gained more attention recently [35, 100, 13].

Prior work on adaptive guardbanding focuses on voltage droop tolerance and system-efficiency analysis at one core or one processor level [31, 101, 52, 54, 14, 35, 100, 13]. In our work, we showcase adaptive guardbanding's system-level implications for core scaling and workload heterogeneity, and we investigate its root causes. Our analysis incorporates di/dt noise and extends to total on-chip voltage drop. Our multicore di/dt noise characterization confirms prior observations [37, 89, 69]. We also observe mitigated typical-case noise and magnified worst-case noise [69] due to on-chip noise propagation [37, 89]. Because adaptive guardbanding deals with di/dt noise well, further investigation should focus on improving its performance with respect to passive voltage drop.

Chapter 5

Managing Enhanced Performance Variation on Adaptive Clocking Multicore Processors

In this chapter, we discuss how to leverage active timing margin's automatic timing margin tracking ability to expose a multicore's static core-to-core performance heterogeneity caused by process variation, and explore how to manage the dynamically occurring inter-core frequency interference caused by the cores sharing the power delivery on an active timing margin system.

On multi-core and many-core chips, it is critical that we push down timing margin that not only deals with the dynamically occurring effects such as temperature and voltage variation but also covers the core-to-core performance heterogeneity caused by lithography's manufacturing process variation. To investigate this issue, we fine-tune the hardware active timing margin solution designed to cope with voltage noise, such as the adaptive clocking fabric in the POWER7+ multicore in Chapter 4. We study enhancing a multicore's active timing margin capability according to each core's characteristics, as well as the running applications' characteristics. Adaptive clocking's per-core configurable control loop provides a new opportunity to expose the inter-core speed variation and to provide more performance gain than the conventional

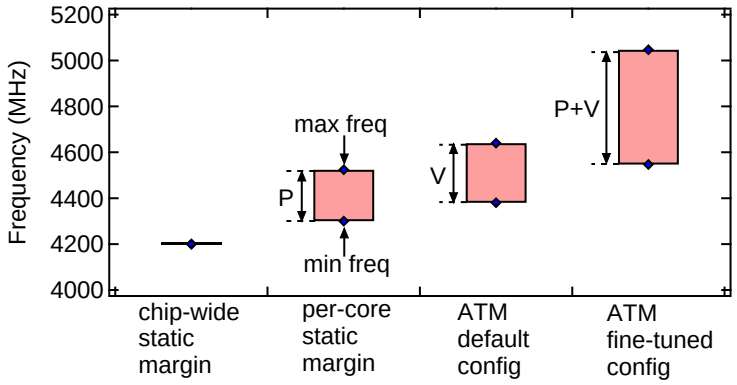


Figure 5.1: Fine-tuning active timing margin (ATM) exposes both process (P) and voltage (V) variation, and improves frequency compared with the default active timing margin configuration and the per-core $<v, f>$ static margin setpoints.

multicore process variation, i.e., calibrating static frequency levels separately for each core [91, 97, 85, 24, 84].

The conventional approach to expose core-to-core variation uses per-core $<v, f>$ setpoint with static margins and thus requires guarding against worst-case voltage variation, such as the di/dt effect and the DC voltage drop across the chip's power delivery path, each of which can consume 3% of the V_{dd} [115]. But because active timing margin can handle these adaptively, it provides more performance gain by exploiting the inherent inter-core variation in the processor.

Figure 5.1 illustrates the performance enhancement and heterogeneity exposed by "fine-tuning" the adaptive clocking control loop for each core. On the tested POWER7+ platform, we (re)configure active timing margin via its Critical Path Monitors (CPMs). The CPM is the chip's programmable canary

circuit that measures the timing margin [54, 26].Similar interfaces exist on other adaptive clocking systems for test-time calibration of margin measurement accuracy and for configuring margin reduction aggressiveness [68, 104, 7], an example is Power Supply Monitor (PSM) on AMD processors [35]. The figure exposes the pros and cons of different approaches.

Starting with the baseline where there is no active timing margin, under a chip-wide static margin (i.e., first bar), all cores have a fixed frequency of 4.2 GHz. Setting the static margin for each core (second bar) with fixed $<v, f>$ improves performance by exposing the fast cores; we estimate the fastest cores can run around 4.5 GHz, based on prior art's voltage noise characterization [115].

Next, the default active timing margin (third bar) carefully programs each CPM to provide uniform core performance, following the conventional contract between processors and users. When idle, all cores run near 4.6 GHz, higher than static margin's fastest cores because of active timing margin's highly effective mitigation of di/dt effects [54]. However, when high power workloads are run, the induced DC voltage drop across the power delivery grid can create long-term steady degradation of the supply voltage delivered, eroding timing margin and reducing active timing margin's frequency gain [115], which lowers the worst-case performance to around 4.4 GHz. Setting fixed $<v, f>$ points for each core requires that this worst-case be guarded against, whereas active timing margin handles it adaptively and frequency only suffers when power consumption is high.

Fine-tuning (fourth bar) at the per-core adaptive clocking control loop level exposes similar inter-core speed variation as static per-core $<v, f>$ set-points, but it provides much higher performance under typical conditions because of active timing margin's adaptive margin provisioning capability. Fine-tuning active timing margin also removes any margin left not trimmed in the default system, which further pushes processor efficiency to the extreme. For instance, when the chip is idle, power consumption and DC voltage drop is minimal, pushing the fastest core to nearly 5 GHz, 10% higher than the fastest static margin core.

While fine-tuning active timing margin provides high frequency gain, it exacerbates variability and induces performance predictability issues. In the worst case, e.g., when DC voltage drop is maximized when running eight high power `daxpy` threads, the slowest core, which runs at 4.7 GHz under idle conditions, slows down to 4.5 GHz, a 500 MHz drop from the fastest 5 GHz case. Thus, application performance can vary widely, depending on the core chosen for execution and any co-located workloads.

Figure 5.2 shows a POWER7+ processor core's performance under different timing margin settings [92, 32]. We instrument POWER7+'s active timing margin via its Critical Path Monitors (CPMs), a programmable interface of the chip's canary circuit that measures available margin [54, 26]. We illustrate with the inference latency of `SqueezeNet`, a compressed convolutional neural network (CNN). Under conventional static timing, the chip clocks at 4.2 GHz, producing an average inference latency of 80 ms. Under the

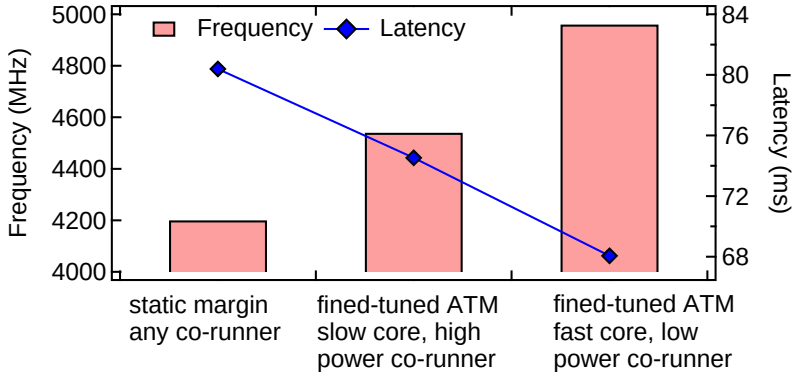


Figure 5.2: `SqueezeNet` inference latency on a POWER7+ core under different timing margin settings. Aggressively fine-tuning active timing margin, and co-locating it with "friendly" low-power applications significantly enhance performance.

chip's default active timing margin, a poorly managed system that co-locates `SqueezeNet` with high-power co-runners such as `daxpy` increases frequency to 4.4 GHz, yielding a limited 7.5% latency improvement. However, customizing each core's active timing margin and wisely managing the system to let `SqueezeNet` run alone boosts core frequency to 5 GHz and reduces latency by 15%, a 2X the performance gain over the default production system.

Inspired by the benefits shown in Figure 5.2, this chapter detail how to fine-tune active timing margin at the core-level to robustly reveal each core's performance limit and to expose inter-core speed differences. We perform extensive hardware measurement to analyze active timing margin's operating limits under different application scenarios, which leads to a low-overhead solution for deploying active timing margin systems with their highest speed at scale, while delivering controllable application performance in the presence

of the exposed process and voltage variation. We present a software solution to actively fine-tune and manage active timing margin. In summary, we make the following contributions:

5.1 Fine-tuning Core-level Active Timing Margin Operation

In our study, we convert all of active timing margin's reclaimed timing margin into frequency and keep V_{dd} unchanged. This process bypasses the restriction on undervolting wherein a chip's worst-case core limits the amount of undervolting. Overclocking allows each core to independently adapt to its conditions and can fully expose a chip's inter-core speed differential, potentially producing more performance benefit. We let active timing margin boost each core's frequency at V_{dd} 1.25 V, the 4.2 GHz P-state.

We explain how to customize a multicore's active timing margin operation to be more aggressive, which extracts more timing margin and increases frequency. Reconfiguring active timing margin's control loop to its operating limit is unexplored before, thus we propose a systematic procedure to characterize how the processor behaves under different scenarios and timing margin reclamation levels. The insights we gain when executing this procedure is instrumental in deploying customized active timing margin systems in production.

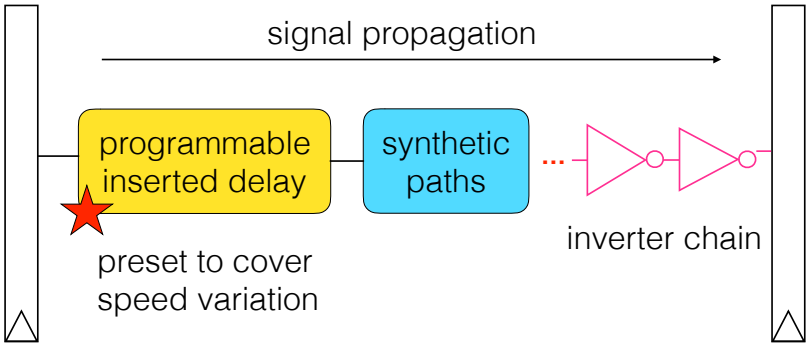


Figure 5.3: CPM has three cascaded parts: programmable inserted delay, synthetic paths, and inverter chain.

5.1.1 Programming Critical Path Monitors to Reconfigure Margin Reclamation

We configure the POWER7+'s Critical Path Monitors (CPMs) to fine-tune active timing margin's margin reclamation behavior. By design, CPMs are programmable to set how aggressively active timing margin trims the margin and, more importantly, to cover speed variation and deliver uniform performance to users. We leverage this interface to fine-tune each core's active timing margin control loop.

Figure 5.3 shows a CPM uses three stages to measure margin [25, 26]: (1) inserted delay, (2) synthetic paths, and (3) an inverter chain. The inserted delay is a configurable circuit. A user can specify the number of inverters a signal passes through to select its timing delay length. The synthetic path simulates a pipeline circuit's delay with a set of paths, including AND, OR, and XOR gates and wires. The final inverter chain quantifies the time left after

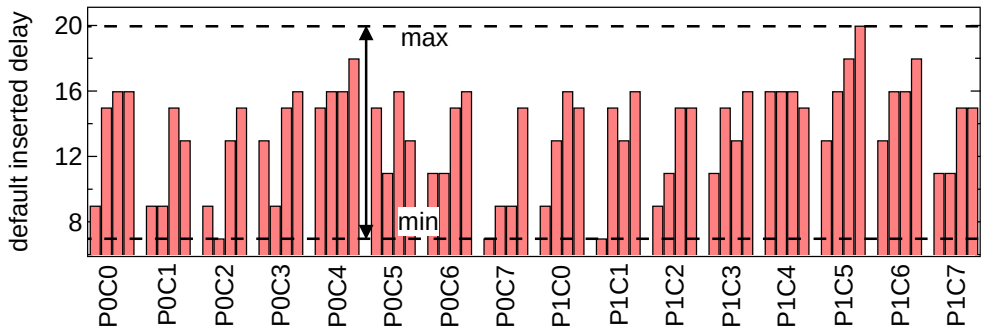


Figure 5.4: Pre-set inserted delay of the CPMs in two POWER7+ chips, grouped by core. There exists wide variation between different CPM sensors.

the signal propagates past the inserted delay and synthetic path by counting the number of inverters a signal passes. The inverter count is a CPM's final output and is sent to the DPLL for clock adjustment.

Before a POWER7+ processor is shipped, each CPM's inserted delay is pre-set at test-time with a default value that serves as extra "protection" for the control loop to function robustly. The pre-set delay makes CPMs report less margin than it could have, leaving some margin not trimmed as protection. The pre-set delay also smooths out the speed differences between different corners of a chip by adding more delay to fast corners in order to fill the empty time after a circuit finishes switching and adding less delay to slow corners.

Figure 5.4 shows the preset inserted delays in each core of the two POWER7+ chips (we exclude CPMs in the LLC because it lies in a different clock domain). Intuitively, each unit of the delay represents some amount of

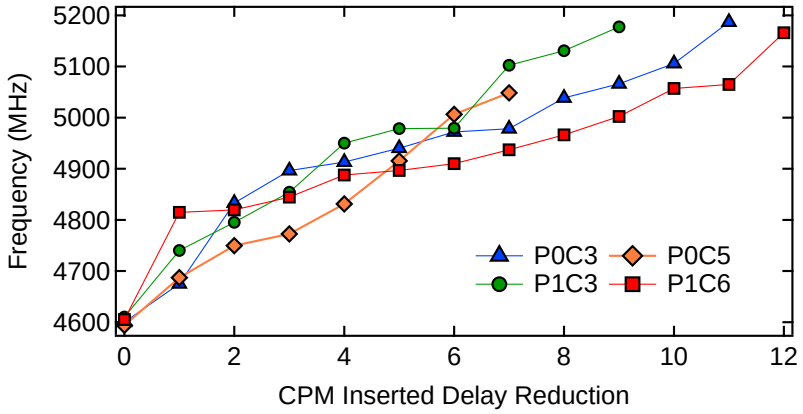


Figure 5.5: Reducing inserted added delay makes the CPM count more time margin after a signal travels through the synthetic path. The DPLL then increases frequency to harvest the excess margin reported by CPM's inverter chain.

timing. Under static margin at 4.2 GHz, reducing the inserted delay by one step lets the CPM detect one to three units more timing margin, equivalent to the speed variation caused by 20-60 mV V_{dd} difference [26, 115]. The magnitude of the preset delay shows the amount of "protection" built into the default active timing margin system. The pre-set inserted delays range from 7 to 20, nearly a 3X range, indicating significant silicon speed variation.

By programming the inserted delay to different values, active timing margin's perception of the amount of available timing margin changes, and thus it is induced to become more or less aggressive in reclaiming timing margin. Figure 5.5 shows, for four example cores (C), across two processors (P) on the same system, how active timing margin converts more margin into frequency as the CPM inserted delay is reduced. The default delay (normalized to 0) makes active timing margin push core frequency to around 4.6 GHz, but

reducing inserted delay (reduction steps beyond 0) pushes frequency to over 5 GHz, a 20% improvement over the static timing margin baseline. Programming the inserted delay to a smaller value (higher delay reduction) decreases the time to the end of the synthetic path, leaving more margin to be counted by the inverter chain. The DPLL loop harnesses the excess margin by overclocking.

Before a POWER7+ processor is shipped, each CPM's inserted delay is configured with some default "protection" delay to keep the CPM timing margin conservative, which guarantees correct active timing margin execution. The protection delay also smooths out the speed differences between different corners of a chip. For the 64 CPMs in our two-socket system (we exclude CPMs in the LLC because it lies in a different clock domain), the protection delays range from 7 to 20, nearly a 3X range, indicating significant silicon speed variation.

In the POWER7+, we configure the inserted delay by programming it with a discrete step count through the server's accompanying service processor. Each step represents some amount of timing delay. Under the static margin at 4.2 GHz, reducing the inserted delay by one step lets the CPM detect one to three units more timing margin, equivalent to the speed variation caused by 20-60 mV V_{dd} difference [26, 115].

We reduce each core's CPM delay from the default amount to increase active timing margin aggressiveness. To simplify the exploration space, we reduce the four CPMs within a core (excluding the LLC CPM) by the same

amount.

5.1.2 Characterizing active timing margin Limits

As shown by Figure 5.5, active timing margin has great potential for more aggressive operation to achieve higher frequency. But to unlock active timing margin's full potential, we need a methodology to characterize the system. Figure 5.6 outlines our procedure.

We profile an active timing margin chip on a per-core basis. System idle is our starting point for the analysis; micro-benchmarks (uBench) cover major paths in a core; and single-threaded benchmarks representing real use cases.

System Idle Running background operating system tasks, an idle system imposes the least stress on the processor. Understanding each core's active timing margin operating limits under system idle provides us with valuable insight into inherent core-to-core differences.

Micro-benchmarks (uBench) Traditionally, micro-benchmarks are used to measure the performance of individual processor modules, such as the branch predictor, floating point unit, and caches. In active timing margin, micro-benchmarks serve an additional purpose because each one primarily touches only one part of the core, avoiding complex microarchitectural interactions. We thus use uBench to get a more comprehensive profile of core-to-core microarchitecture level variation.

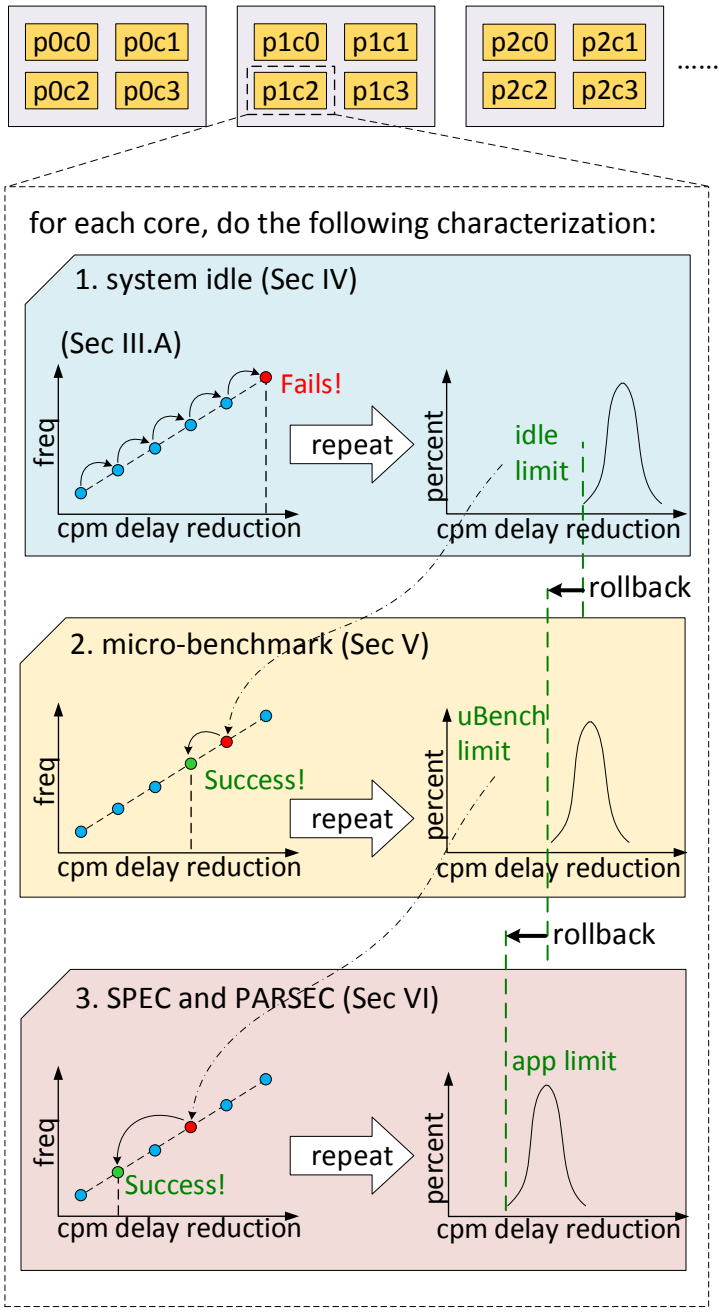


Figure 5.6: Our active timing margin characterization methodology iterates over each core and follows a step-by step approach, going from the simplest system idle scenario to the complex real-world workloads.

Realistic Workloads For the final step, we profile the system with complex applications from SPEC CPU 2017 and PARSEC. These benchmarks cover a wide spectrum of program space in the real world and have diverse architecture behavior [94, 10]; hence they can touch more corner-case timing paths or create more active di/dt effects than uBench, all of which threatens the safe execution of aggressively reconfigured active timing margin. The single-threaded workloads help identify application, chip-wide, and individual core level heterogeneity.

In each of the above setups, failure may occur as a result of timing violation, manifested as an abnormal application termination (e.g., segmentation fault), silent data corruption (SDC), or a system crash. For SDC related error, we rely on SPEC and uBench's inherent result checking tool for guaranteeing execution correctness. All these failures may occur because either the CPM's delay has become so short that it does not capture real circuit delays or system noise events, such as the di/dt effect, overwhelms the control loop's ability to respond in time. Because the effects that cause active timing margin failure might be not fully deterministic, we repeat the profiling in each setup at lest 20 times to produce a distribution of active timing margin operating limits. We expect the distributions to be tight because timing violations will not be entirely random. These distributions provide a holistic view of active timing margin's margin reclamation capability, so we study them from here on.

Our methodology progresses through increasing workload complexity. Thus we often need to roll back the CPM delay setting that was successful

in a previous less complex setup to a less aggressive point, reflecting a workload setup's unique impact on active timing margin's operation. Although the worst-case scenario might determine practical active timing margin reconfiguration in the real world, the middle point analysis shed useful insights on what affects the core-level customization of active timing margin's margin reclamation.

There is no guarantee that a particular circuit path or system noise event will deterministically lead to a timing violation, so we repeat the profiling in each of the above setups at least 20 times to produce a distribution of active timing margin operating limits. On the other hand, the effects that lead to a timing violation are not entirely random. Reconfiguring CPM inserted delay beyond a limit often leads to certain critical paths having much higher probabilities of experiencing timing errors; thus, the resulting distributions of successful CPM delays tend to be very tight. These distributions provide a holistic view of active timing margin's margin reclamation capability, so we study distributions here onward.

A timing violation manifests as an abnormal application termination (e.g., segmentation fault) or a system crash. It happens because either the CPM's delay has become so short that it does not capture real circuit delays, or system noise events, such as the di/dt effect that overwhelms the DPLL.

Our profiling methodology progresses through increasingly complex workloads. Thus we often need to roll back the CPM delay setting to a less aggressive point, reflecting a workload's unique impact on active timing margin's

	P0C0	P0C1	P0C2	P0C3	P0C4	P0C5	P0C6	P0C7	P1C0	P1C1	P1C2	P1C3	P1C4	P1C5	P1C6	P1C7
idle limit	9	8	4	11	10	7	8	2	4	8	5	8	7	5	10	3
uBench limit	9	8	4	10	9	7	8	2	4	8	5	5	6	4	10	2
thread normal	8	7	4	9	8	6	7	2	3	7	5	4	5	3	8	2
thread worst	6	6	3	6	6	5	5	2	3	3	5	3	3	2	6	2

Table 5.1: ATM customization limits under system idle, uBench, and real-world application. Data is collected on two eight-core (C) POWER7+ processors (P). ATM limits are reflected as the number of stepped reduced from CPM's default inserted delay configuration.

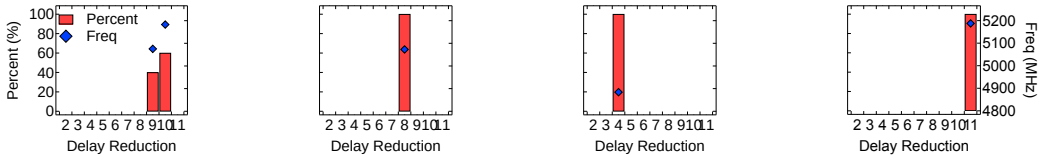
operation.

5.2 Idle System Characterization

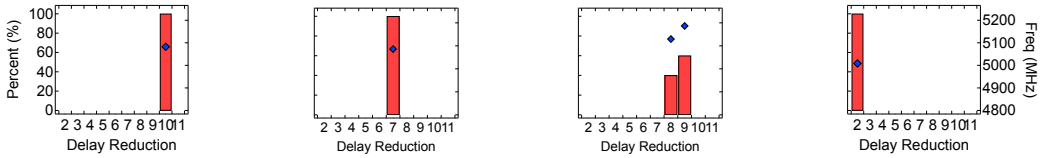
Understanding active timing margin's margin reclamation limits in an idle system sets a starting point for further, more complex analysis. With no application code running, the system exerts minimal stress on active timing margin's reconfigured control loop, enabling us to use active timing margin to expose the silicon's inherent maximum speed.

Running only the operating system, we build a distribution of the most aggressive yet safe CPM configuration points for each core, depicted in Figure 5.7 by the amount of CPM delay reduction from the chip's default setting, along with the resulting frequencies. As expected, the distributions are tight, covering no more than two configurations. Each core's *idle limit* is the lowest (most conservative) CPM delay reduction plotted, e.g. 9 in Figure 5.7a. These are summarized in Table 5.1.

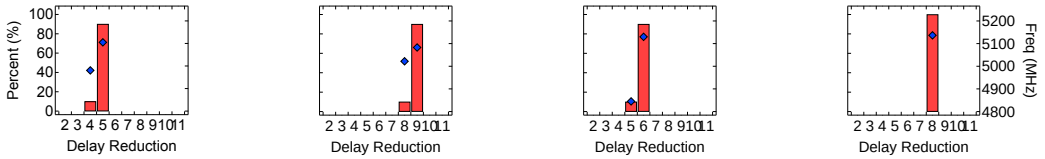
The different core-to-core idle limits reveal lucrative performance potential for aggressive active timing margin customization (Chapter 5.2.1), and



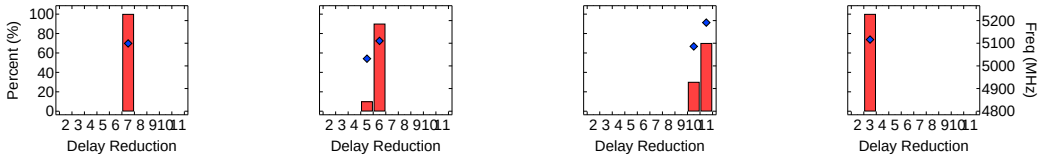
(a) p0 core 0. (b) p0 core 1. (c) p0 core 2. (d) p0 core 3.



(e) p0 core 4. (f) p0 core 5. (g) p0 core 6. (h) p0 core 7.



(i) p1 core 0. (j) p1 core 1. (k) p1 Core 2. (l) p1 Core 3.



(m) p1 Core 4. (n) p1 Core 5. (o) p1 Core 6. (p) p1 Core 7.

Figure 5.7: The limit configuration of each POWER7+ core (i.e., the most aggressive reduction of CPM's inserted delay from its default setting, beyond which ATM operation can cause system failure under idle condition) distributes over a narrow range (red bar, left y axis). The operating frequency at each core's limit delay config is over 4800 MHz, more than 15% higher than static margin's 4200 MHz level (blue mark, right y axis).

the significant core-to-core performance variation (Chapter 5.2.2) which is partly caused by the non-linearity in CPM configuration (Chapter 5.2.3).

5.2.1 Significant Performance Potential

For most cores, the inserted delay can be aggressively reduced, making active timing margin's control loop see more timing margin for reclamation. As Figure 5.7 shows, more than half the cores (e.g., P0C0 and P0C1) can tolerate reductions of at least seven steps of CPM inserted delay, elevating frequencies to over 5000 MHz: a 7% improvement over default active timing margin's 4600 MHz and a 20% improvement over static margin's 4200 MHz baseline, showing customized active timing margin can substantially improve performance.

5.2.2 Exposed Inter-core Frequency Variation

Programming the CPM to change active timing margin operation yields different frequency levels for each core, despite the performance improvement. For instance, at the idle limit P1C2 runs at about 4850 MHz but P0C3 achieves about 5200 MHz. Even within a chip, there is a wide range (e.g., P0C2 and P0C3). The core-to-core frequency variation is essential for application performance management, which we discuss later.

The core to core differences are understood to be a result of manufacturing process variations [24, 84], i.e., some core's circuits are faster due to imperfection in the lithography process. For instance, as Figure 5.7 shows, P0C3 can safely reduce its CPM delay by 11 steps, while P0C7 can only mitigate its delay by two, reflecting the varying amount of timing margin available for reclamation, which is caused by the two cores' speed difference.

However, because on the POWER7+ each core's performance potential is unlocked via active timing margin control loop's automatic harness of available timing margin, the functioning of active timing margin control loop also plays a critical role in the inter-core performance variation.

5.2.3 Nonlinearity of CPM Configuration

The CPM inserted delay's configurable inverter chain is designed to have linear timing delay graduation for timing margin measurement. However, the manufacturing process makes it have non-linear graduation when configured to measure timing margin. The non-linearity magnifies the inter-core performance heterogeneity.

The inserted delay's non-linear configuration manifests as significant idle limit variation between cores. Consider P0C4 and P1C7, which are both able to increase frequency from 4600 MHz to 5100 MHz but do so with very different CPM changes: P0C4 reduces the delay by ten steps, while P1C7 only needs two steps. Hence, although the two cores have similar excess timing margins, P0C4's CPM encodes smaller timing delays in each step than P1C7.

Within each core, CPM's non-linearity makes the timing margin encoded by one CPM delay step vary. Figure 5.5 shows that P1C6's frequency increases by over 200 MHz when going from step zero to one, jumping from the baseline 4600 MHz to over 4800 MHz. But in going from step one to two, there is an almost negligible change in frequency. Similarly, the frequency is nearly unchanged when increasing the CPM delay reduction from step five to

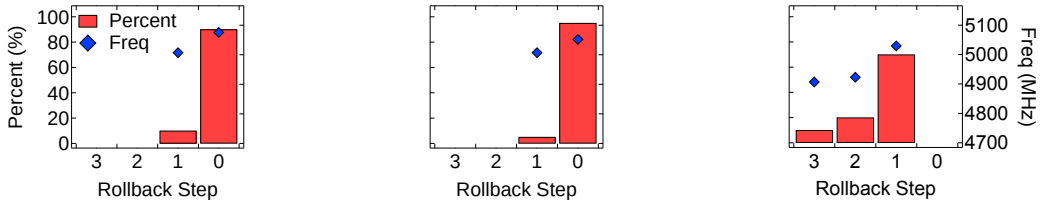
six for P1C3, but reducing the delay by one additional step (i.e., going from six to seven) increases the frequency by over 100 MHz.

As another example, in Figure 5.7k reducing P1C2's CPM delay by six is too aggressive and can crash the system; rolling back its delay by one step ensures safety but at the cost of 300 MHz. P1C1 (Figure 5.7j) similarly needs its CPM delay reduction rolled back by one step (from nine to eight) for safe operation but at the cost of only 100 MHz. Though P1C2 could operate safely at a higher frequency, the large CPM jump forces the 300 MHz drop and amplifies the differences between the two cores.

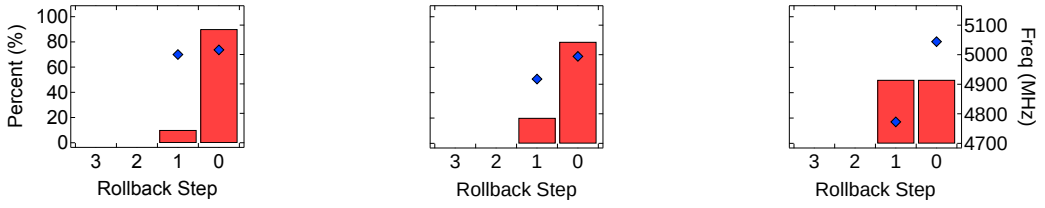
In summary, the non-linearity configuration of the CPM and active timing margin control loop demands that customization of multi-core Active Timing Margin operation be carried out carefully on the per-core basis because no single CPM configuration works uniformly for all cores.

5.3 Micro-bench Characterization

While idle system characterization reveals insights on the performance benefits and the inter-core variation issues of multicore active timing margin customization, it does not evaluate the system's behavior under stress from real-world application codes. Before using more complex applications, we use micro-benchmark (uBench) as a valuable tool that controls program behavior to analyze individual processor components [73]. Because uBench imposes more stress than idling, the CPM configuration tends to be more conservative, creating a practical point for processor deployment in the real world.



(a) p0 core3, `daxpy` (b) p0 core4, `daxpy` (c) p1 core3, `stream`



(d) p1 core4, `stream` (e) p1 core5, `coremark` (f) p1 core7, `coremark`

Figure 5.8: For 6 out of 16 cores, ATM configuration (i.e., CPM's inserted delay setting) needs to be rolled back from its idle limit in order for micro-benchmark (uBench) to run successfully. The FP (`daxpy`), MEM (`stream`), and INT (`coremark`) uBench have similar distribution of their pass config, indicating the core's mismatch between its reconfigured CPM timing measurement and its actual circuit speed. The other 10 cores not shown can run uBench safely at their idle limits.

5.3.1 Workload Selection

We evaluate system behavior under aggressive active timing margin customization using three uBench programs. These programs collectively cover all main parts of the microarchitecture, as well as the dispersed CPMs in a core.

We use `coremark` [27] to stress the core's control, branch, and integer units; `daxpy` to stress the floating point unit; and `stream` [20] for its ability to generate cache misses and exercise the load-store unit. Prior work has used

such benchmarks to exercise the functional units and validate the active timing margin [54, 55]. We check the programs' run result to evaluate processor execution correctness. All incorrect runs manifest as system crashes or abnormal application exits.

Using these benchmarks ensures we challenge a reconfigured active timing margin by touching more paths than system idle. Meanwhile, these uBench programs create little system noise, especially the di/dt effect. They have controlled, smooth program behaviors and avoid complex microarchitectural activity such as periodic pipeline flush, which is the root cause of workload-induced voltage droops [36, 78, 88, 89, 69]. The di/dt effect is dangerous for aggressively reconfigured active timing margin because its fast drooping voltage can prevent the control loop from engaging in time [103], resulting in application failure.

5.3.2 What Makes Some Cores Fail?

We start the uBench characterization from the idle limit because it is the point that sustains stable system state. If this initial starting point fails, the CPM inserted delay is rolled back to have a longer timing delay to make active timing margin harness timing margin more conservatively until the program runs correctly. We find most cores' idle limits sustain correct uBench execution, which entails they can safely accommodate the major paths activated by the instructions used by uBench programs.

For the server's two physical processors, uBench characterization ex-

poses six cores that fail for the three programs. Figure 5.8 shows the distributions of reintroduced delays for these cores, using the "rollback steps" relative to the idle limit, which reflects the stress impact from uBench program execution compared with system idle. For those six cores, rollback ranges from one to three steps and sustains all uBench workloads.

All three programs, despite their different characteristics, show similar behaviors on the six problematic cores. The implication is that the microarchitecture blocks that limit active timing margin fine-tuning are the common structures used by all programs, such as instruction fetch and scheduling, rather than specific modules stressed by each application (e.g., FP unit). We also find *uBench limit* sustains voltage and power stress-test, which will be detailed later in this paper. We, therefore, use the *uBench limit* as a reference point for further characterization using realistic applications.

5.4 Realistic Workload Characterization

To run real applications, a production active timing margin system today adds some amount of protection margin to CPM's uBench limit configuration [54]. To conservatively guarantee execution correctness, the added margin can be up to 50% of the static guardband. But this leaves room for improvement as demonstrated by the 2X frequency gain during our system idle characterization.

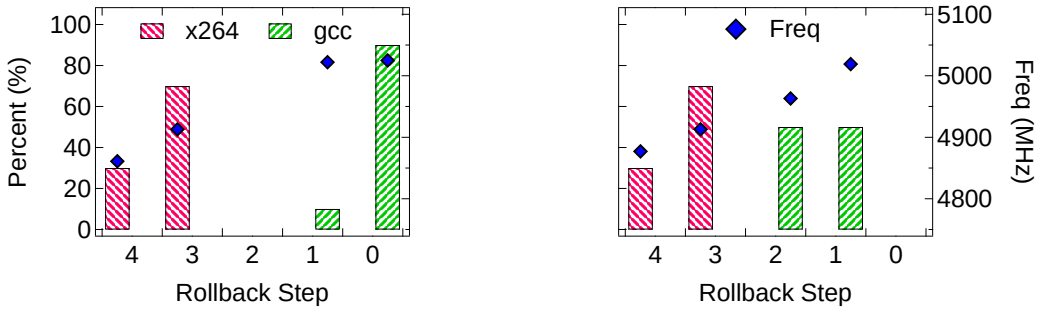
However, adding additional guardband as a conservative precaution ig-

nores the application-dependent behavior and can waste valuable performance benefit. In this section, we profile with a variety of integer and floating point workloads from SPEC CPU 2017 [19] and PARSEC 3.0 [11]. We use these workloads because their result-checking tool provides a convenient method for checking execution correctness. Understanding per core active timing margin operating limits under these heterogeneous workloads offer helpful insights for deploying aggressively customized active timing margin chips in real-world use cases.

To understand all system factors that impact an aggressively fine-tune active timing margin processor, we profile with a variety of integer and floating point workloads from SPEC CPU 2017 [19] and PARSEC 3.0 [11]. These realistic workloads provide helpful insight for deploying aggressively fine-tuned active timing margin chips in real-world use cases. They often have more complicated code patterns that may touch corner timing paths in a core, or introduce complex microarchitectural behaviors that can lead to severe di/dt effects, both of which threaten to violate the aggressively tuned CPM configuration after uBench profiling, even though the uBench limits already ensure the active timing margin control loop protects major core paths.

5.4.1 Application Heterogeneity

Figure 5.9 shows `x264` often requires significant CPM delay rollback from the uBench limit, whereas `gcc` needs relatively little, allowing active timing margin to more aggressively boost frequency. The rollback reflects an ap-



(a) P0C3 (b) P1C6

Figure 5.9: `x264` stresses active timing margin more heavily and needs a more conservative CPM configuration compared to `gcc`, as indicated by the larger CPM rollback that is required for `x264` over `gcc`.

plication's unique system noise effects. Configuring active timing margin for the worst application in all cases, e.g., `x264`, wastes active timing margin's margin reclamation potential when running more benign workloads. This is the approach taken by today's deployed active timing margin processors, which still rely on a safety margin as large as 50% of the original static guardband [54]. This is the case for today's active timing margin processors deployed into the field which still rely on some safety margin, approximately 50% of the original static guardband [54].

To get a complete picture of the behavior of aggressively configured active timing margin cores on different workloads, we profile CPM rollback from the uBench limit for all $< app, core >$ pairs in Figure 5.10. We use the weighted average CPM rollback as it quantifies the application's unique stress level. Two applications may have quite a different delay reduction distributions even when they show the same lower bound in their CPM delay profile.

From the individual rows in Figure 5.10, we see that each workload imposes a different amount of stress but does so consistently across cores. For instance, `x264` and `ferret` needs much more conservative active timing margin setting than `gcc` and `leela`, indicating these workloads have exerted a higher pressure on active timing margin's control loop.

We classify the workloads as "heavy," "medium," or "light" as shown in Table 5.2. "Heavy" workloads pose the greatest threat to aggressively reconfigured active timing margin and often force a rollback of CPM inserted delay for more conservative operation. In contrast, "light" applications exert little pressure on active timing margin and often need no rollback from the uBench limit. The "medium" workloads show more sensitivity to a core's active timing margin control loop.

In Table 5.1, *thread-worst* is the worst CPM configuration limit of all workloads and represents the most severe application stress in our profiling. The *thread-normal* is less conservative and lets most medium, and light applications safely pass. From our realistic single-threaded workload profiling, we draw the following two key insights:

From the individual columns in Figure 5.10, we see that different cores exhibit varying levels of "robustness", where we define robustness as the immunity to CPM rollback from the core's CPM uBench limit. The cores on the right of Figure 5.10 has the highest robustness, requiring the least rollback across all applications, indicating their active timing margin control loops can deal with the system effects of any application. We anticipate they will con-

stress level	benchmark
heavy	x264, exchange2, ferret
medium	perlbench, xalancbmk, xz, facesim, omnetpp, mcf, bodytrack, dedup
light	gcc, bodytrack, deepsjeng, leela, freqmine, barnes, streamcluster, fluidanimate, fft, blackscholes

Table 5.2: Benchmark classification based on their stress level to aggressively configured active timing margin.

tinue to be robust on untested applications since the profiled workloads already cover different behaviors [94].

The reason why certain applications and cores are more vulnerable after aggressive active timing margin customization is a combination of the core's inherent speed and the running application's characteristics. We conducted a best-effort static instruction analysis on the applications and concluded that more detailed insight into the running instructions is needed to predict each application's best-fit CPM setting on each core. For instance, `gcc` covers a much richer set of instructions than `exchange2`, likely touching more corner timing paths, yet stresses active timing margin much less. As another example, `x264` has similar performance counter profiles as `leela`, but their rollback requirements differ substantially. We, therefore, defer the root-cause analysis and the prediction of applications' heterogeneous CPM configuration to future work and focus on the variations already exposed.

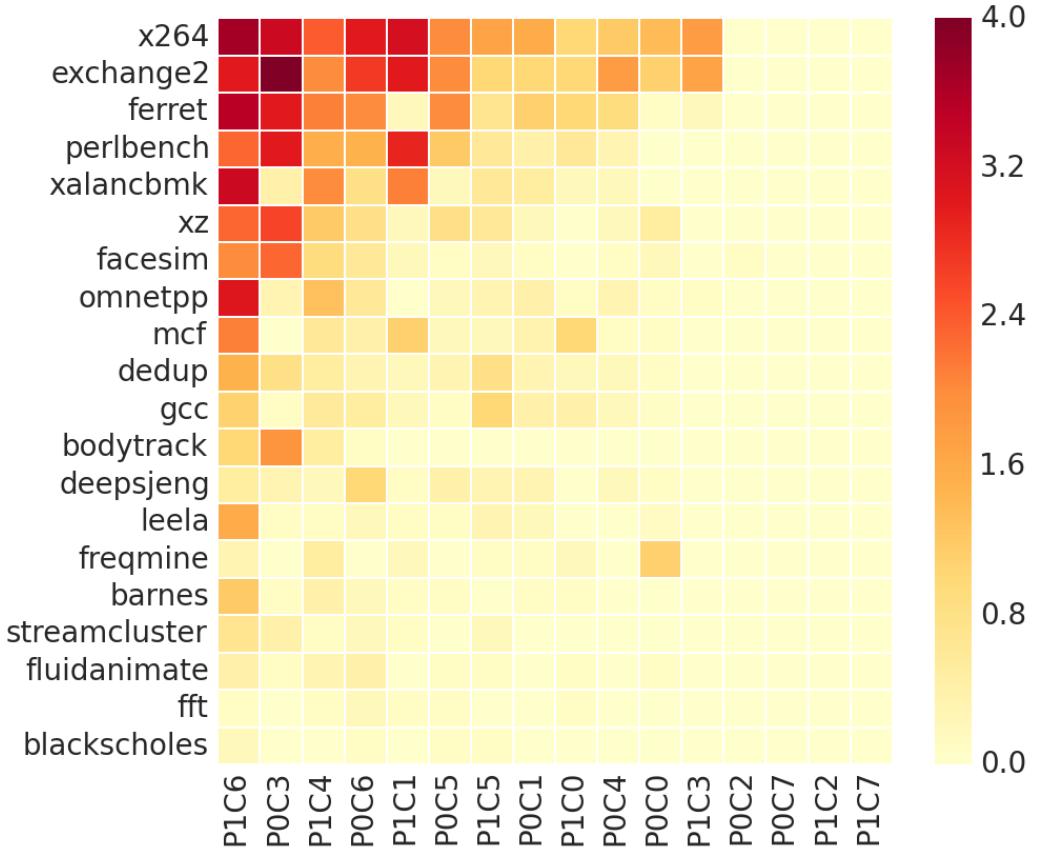


Figure 5.10: Application's average CPM delay rollback from the core's uBench limit. The top workloads stress active timing margin heavily and need more delay rollback for less aggressive margin reclamation.

5.4.2 Core Robustness Heterogeneity

Cores have varying levels of "robustness" to application heterogeneity, where we define robustness as the immunity to CPM rollback from the core's inherent speed (the uBench limit profile). From the columns in Figure 5.10, the cores on the right exhibit the highest robustness, requiring the least rollback across all applications, indicating their active timing margin control loops can

deal with the system effects of any application.

Figure 5.11 sorts cores' average rollbacks across all workloads. The rightmost cores, P0C7, P1C2, and P1C7 are immune to workload effects, flawlessly executing all applications at their uBench limit. We anticipate they will continue to be robust on untested applications since the profiled workloads already cover various behaviors [94]. These "robust cores" can be relied upon in a production environment to execute any application. Among the robust cores, P1C7, however, is notable because its CPM delay was rolled back from the idle test to the uBench test, significantly reducing its frequency to a rather conservative 4800 MHz, possibly accounting for its apparent robustness. Contrariwise, P0C7 remains robust even at its CPM delay from the idle test. As such, there is no clear correlation between a core's speed and its active timing margin robustness.

Figure 5.11 also summarizes different cores' frequency variation under the profiled scenarios. At the uBench limit configuration, core-to-core speed varies by 300 MHz from the fastest, P0C6, to the slowest, P1C7. The speed gap shrinks to 200 MHz at the thread-worst limit, caused by CPM delay rollback of the non-robust active timing margin cores. Nevertheless, the non-uniform core frequency is still impressive and deserves proper management. The arithmetic mean frequency is 4908 MHz, under thread-worst setting, and the standard deviation is 63.5 MHz.

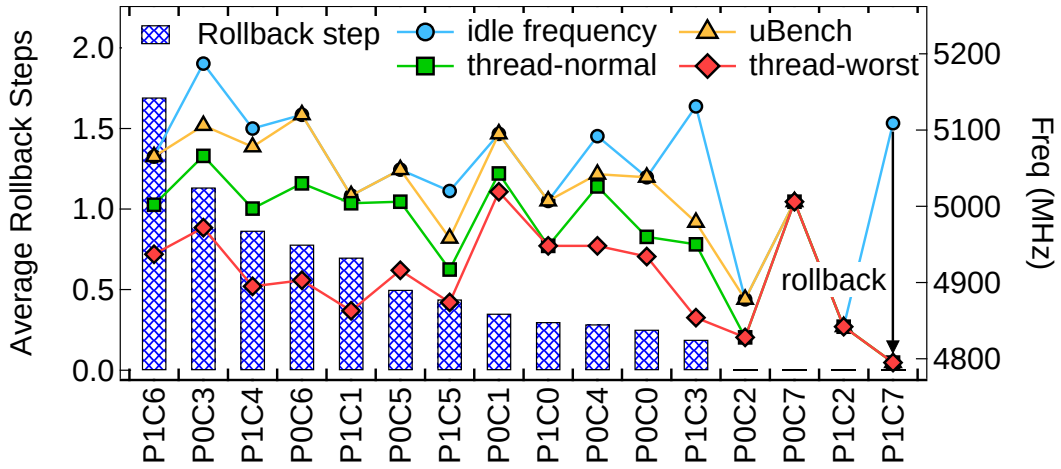


Figure 5.11: Aggressively configured active timing margin cores exhibit different CPM rollback steps and frequencies when running realistic workloads.

5.5 Managing Fine-tuned Active Timing Margin

In this section, we discuss how to deploy and manage a fine-tuned active timing margin system into the field in the presence of significant variability. Fine-tuning improves application performance because frequency is higher. However, pushing active timing margin to its operation limit requires execution correctness guarantee, and the varying frequency levels of different cores and application scenarios create obstacle for the processor to delivering a promised performance level to end users. Hence, we discuss how to determine CPM settings for each core to robustly expose variation, and show how to schedule and throttle co-running workloads to deliver predictable performance for latency sensitive critical applications.

5.5.1 Deploying Fine-tuned Active Ting Margin Configuration

The insights we gather while analyzing the operating limits of a fine-tuned active timing margin system under idle, micro-benchmarks, and realistic workload scenarios are useful for understanding the performance opportunity from exposed inter-core variability, but overhead of our procedure is too high for finding a processor's fine-tuned configuration in a real-world deployment.

Because programs have heterogeneous CPM settings on different cores, one might try to predict each application's best CPM setting on each core. However, such a prediction scheme would demand essentially perfect prediction accuracy because any misprediction can lead to system failure or incorrect execution. Achieving this accuracy is difficult because it relies on deep knowledge of a program's di/dt behavior as well as the circuit paths touched by the program, all of which derives from the dynamic instruction streams and may incur high overhead [88]. We leave CPM prediction for future work.

Rather than predict CPM settings, we propose a test-time stress-test procedure to identify active timing margin fine-tuning limits while maintaining a correctness guarantee. The approach and evaluation presented here is an example of the process we recommend, and not meant to be literally the exact steps to follow. For instance, the stressmarks we use in the paper are different from what we use in production. Nonetheless, the general approach we discuss is useful.

During test-time, we iterate over each core and run worst-case work-

loads, such as a di/dt stressmark [51, 8], power stressmark [9], and ISA test suites to create high voltage noise and high operating temperatures and to cover all potential circuit paths. The combined stress-test finds each core's limit active timing margin configuration, providing a guarantee of correctness for any realistic workload because, by definition, a stress-test pushes the system beyond the requirements of any other workload.

In our work, we try our best to create a stress-test with a voltage virus that repeatedly and synchronously throttles all cores' instruction issue rate to operate only one out of every 128 cycles while simultaneously running 32 `daxpy` threads. The `daxpy` workloads create high power consumption, raising chip power to 160 W and temperature to 70℃; the issue throttling creates a synchronous power surge across the chip, inducing concurrent di/dt effects from adjacent cores, representing worst-case voltage noise [54, 103]. We recognize that better power stressmarks can be constructed using more systematic procedures [9], but we do not expect power and temperature to be the limiting factors for active timing margin operation because these are long-term effects and are well within active timing margin control loop's nanosecond-level response time. Though the realistic workload characterization in Chapter 5.4 covers a variety of instructions, in practice, chip vendors have tailored ISA verification suites that provide wider coverage and execute in less time.

On the two tested POWER7+ chips, the *thread worst* CPM configurations sustain correct execution under all our stressmarks. To provide additional correctness guarantees, the vendor can optionally rollback the stress-

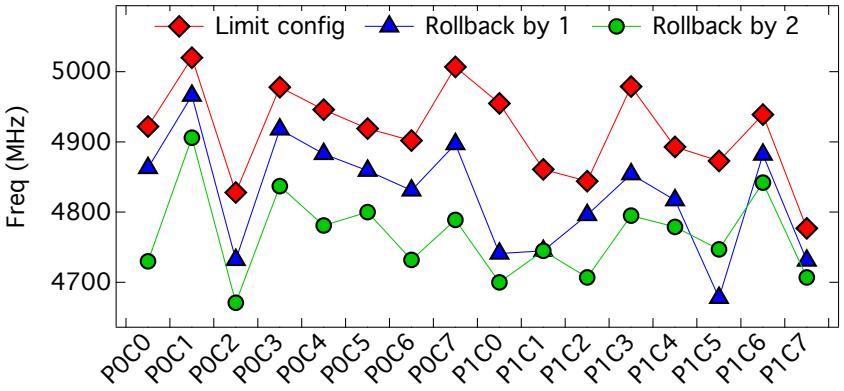


Figure 5.12: To ensure execution correctness, fine-tuning active timing margin goes through worst-case stress-test during test time. Vendors can optionally roll back stress-test active timing margin configurations, providing additional safety guarantee. Either way, speed variability is exposed.

test-determined active timing margin limit by several steps.

Figure 5.12 shows the core frequencies across the two POWER7+ chips after executing the above test-time procedure. At their limit, P0C1 and P0C7 have over 200 MHz speed differential. Rolling back each core's CPM from the limit by one or two steps keeps the same inter-core variation trend and provides an additional safety guarantee. In the management scheme we propose, we will use the limit *thread-worst* configuration, though the conclusions we present and the scheme we propose can be applied to more conservative (rolled back) configuration points.

5.5.2 Per-core Frequency Predictor

To manage active timing margin's performance variability, we first develop a predictor that informs frequency and performance for a candidate ap-

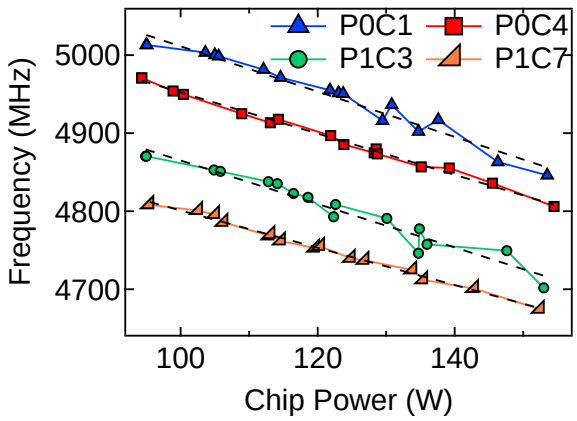


Figure 5.13: After active timing margin customization, core frequency can be predicted with a fitted linear model, following Equation. 5.1.

plication schedule. We develop this predictor by modeling each core's runtime average frequency $\overline{f}$, as a linear function of the transistors' supply voltage, V_{chip}. Among different dynamic effects, long-term stable effects such as temperature variation and DC voltage drop caused by high power determines core frequency —infrequent, transient di/dt events are handled transparently by the active timing margin control loop.

Because past research has shown that speed is only modestly affected by temperature [113], we base our model strictly on IR voltage drop. Subtracting the IR voltage drop, which is proportional to current and hence power consumption, we derive a linear relationship between active timing margin's dynamic frequency and the chip's total power consumption as shown in Equation. 5.1. The value b represents a core's static CPM setting, while $k' \cdot \overline{P}$ represents the dynamic variation, dominated by the IR voltage drop.

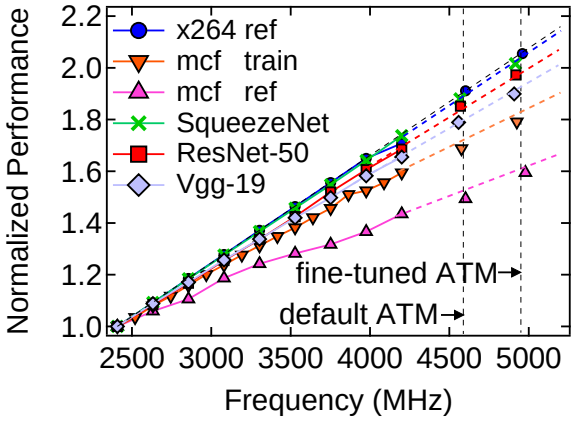


Figure 5.14: Single-thread application performance can be predicted linearly using core frequency.

$$\begin{aligned} \overline{f} &= k \cdot \overline{V_{chip}} = k \cdot (V_{vrm} - R \cdot \overline{I}) \\ &= k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}}) \\ &= -k' \cdot \overline{P} + b \end{aligned} \tag{5.1}$$

Figure 5.13 shows the linear model fitted for each core's customized CPM configuration. The measured data points align with Equation. 5.1's predictions. Each additional watt degrades the frequency by about two MHz. In practice, each core stores its frequency prediction model and the model is indexed by the chip's total power consumption during job scheduling and runtime.

5.5.3 Delivering Critical App's Performance

Frequency directly affects application performance. Figure 5.14 shows application performance scales linearly with frequency, with different coeffi-

cients depending on the workload's memory behavior. A memory-bound workload, such as `mcf`, enjoys less performance improvement from higher frequency compared with a compute-bound workload like `x264` because cache misses limits the compute throughput. We, therefore, build a performance predictor for each application, using frequency as the input. In this way, thread performance on each core can be inferred by the chip's total power, using each core's frequency predictor as the intermediate step.

On a customized active timing margin system, each application's performance depends on both the core it runs on as each core has different CPM configuration which leads to varying frequency levels, and the applications running on other neighboring cores, as all applications contribute to the chip's total power which in turn affects each core's frequency through the DC voltage drop on the shared power delivery path. For some `critical` applications that the users are interested in, it is crucial that they get mapped to the customized cores that are high performance and robust. Meanwhile, it is also crucial that the co-located `background` applications are adequately managed so that the total chip power does not exceed the level that hampers critical app's core frequency. To handle this issue, we propose a scheme to selectively throttle `background` application performance to control total chip power, and indirectly frees up frequency potential for `critical` applications.

We use the applications in Table 5.3 for evaluation. The `critical` workloads are user-facing and require high performance for lower latency. They include deep learning inference (CNN, RNN, and LSTM models), ob-

Mem behavior	Critical	Background
Intensive	resnet, vgg, ferret, fluidanimate	mlp, gcc, facesim, lu_cb, streamcluster, etc.
Non-intensive	squeezenet, seq2seq, babi, bodytrack, vips	blackScholes, x264, swaptions, raytrace,

Table 5.3: Classifying critical and background applications, based on their memory subsystem interference behavior.

ject detection, real-time image processing, content similarity search, etc. The `background` workloads can tolerate lower performance and include workloads such as stock price estimation, 3D image rendering, compression, compilation, and machine learning training. For our work, we focus on the performance issue caused by the active timing margin system's shared power delivery problem and excludes performance interference from the memory subsystem which is a general issue for all multicores. Thus, we avoid co-locating two memory-intensive workloads at the same time to simplify the problem.

Figure 5.15 outlines our management scheme. It takes into account the core-to-core performance and robustness variation as characterized in Chapter 5.4, and the inter-core application power interference on the power delivery subsystem. First, the user selects how he/she would like to set different core's CPM. The default policy uses the chip's thread-worst CPM configuration as shown in Table 5.1, obtained through our earlier characterization.

The default thread-worst policy represents a balanced trade-off between performance and reliability. Most likely workloads can execute correctly while

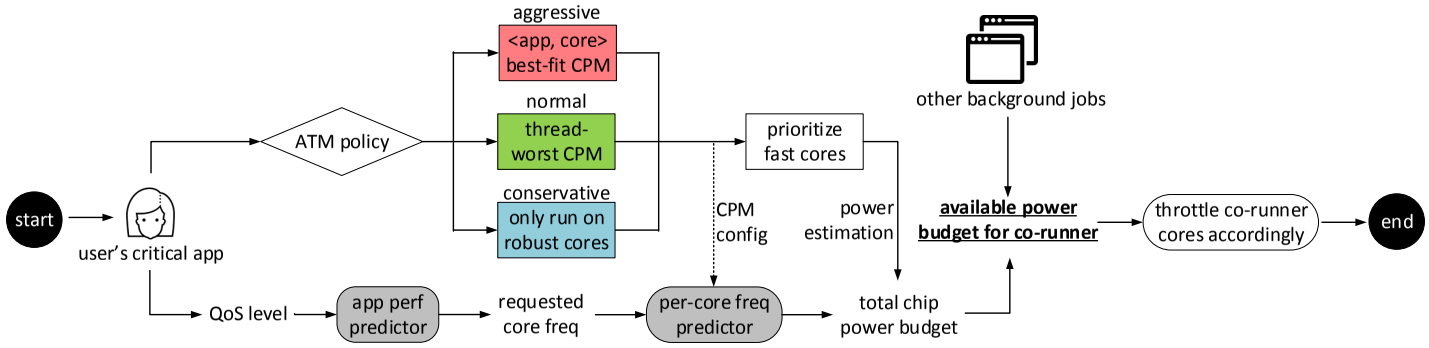


Figure 5.15: Managing a customized active timing margin system needs to integrate the per-app performance predictor and per-core frequency predictor, so that `critical` application performance can be satisfied by using faster cores and maintaining an estimated chip power budget.

still providing better performance. The `critical` and `background` workloads all execute correctly under thread-worst.

For higher performance, the user selects an "aggressive" governor, which chooses an application's most aggressive CPM configuration that guarantees correct execution. In the current approach, this can be achieved by repetitive profiling an application's CPM limits in a tier of testing servers before shipping the application to production server clusters. For most medium and light workloads in Table 5.3, thread-normal represents the high-performance policy.

For higher robustness, the user can select a "conservative" governor, which only schedules `background` workloads onto robust cores picked by active timing margin characterization. The robust cores are scarce and may not provide the highest performance, but they have the highest guarantee of correct execution. The conservative policy is best for unknown applications or when application correctness is paramount.

The operating system then automatically sets each core's CPM setting according to user-selected policy. The faster cores after CPM customization are selected for running `critical` application. In parallel with CPM reconfiguration, the scheme reads user-specified QoS target for the `critical` application and infers the chip power needed to meet the performance goal using per-application performance predictor and per-core frequency predictor. To meet the QoS goal, total chip power under `critical` and co-running `background` workloads cannot exceed the calculated power budget.

The manager subtracts the estimated power of the `critical` workloads from the total chip power budget to get the power envelope available for the co-running `background` jobs. The `background` jobs can then be scheduled to the same chip under this envelope by carefully tuning their power consumption. On POWER7+ where V_{dd} is shared for all cores, we adjust power consumption by changing core frequency. Depending on the power envelope, we can 1) allow workloads to use aggressive active timing margin that has the highest frequency, 2) set cores to different DVFS states' frequency levels or 3) use power-gating to disable cores.

5.5.4 Performance Improvement

We evaluate our solution(s) against the static margin and the default active timing margin. Some customers turn off active timing margin for predictability. Hence the static margin is one of the fair baselines we compare with for evaluation. The system is running the stock DVFS operating system governors that already strive to improve system efficiency. Therefore, our results include that comparison implicitly. Since active timing margin systems are still new and rare, there is little other prior work to compare against directly.

Our evaluation is carried out when all cores are scheduled to run an application. In practice, power gating idle cores off when not enough workloads are available can further free up chip power and boost the performance of target workload [115]. For all our tests, die temperature is maintained under 70℃,

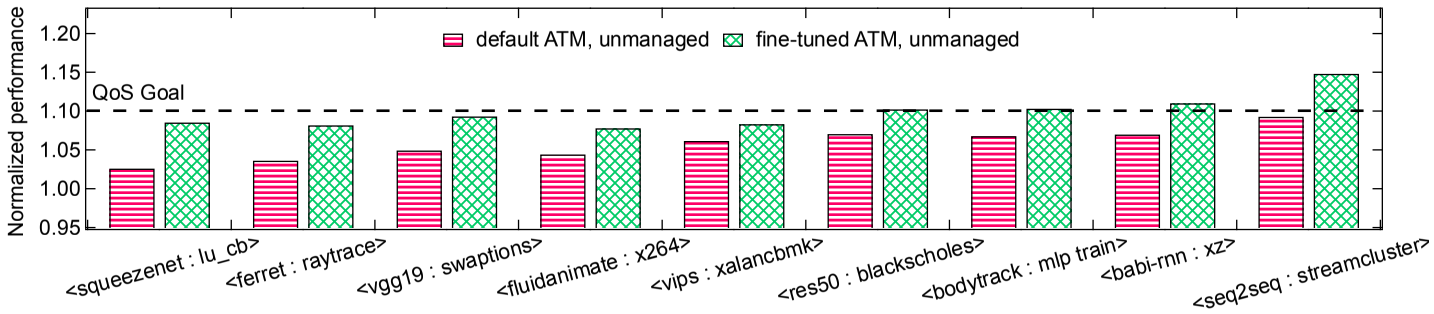
and no side effect on-chip cooling is observed.

Figure 5.16 summarizes the performance benefit of managing an aggressively customized active timing margin system. To highlight frequency interference's impact, we use one core to run `critical` application, which is a natural fit for many applications, such as LSTM and RNN inference. We co-locate all `critical` and `background` applications on processor 0 (P0) of our two-socket server.

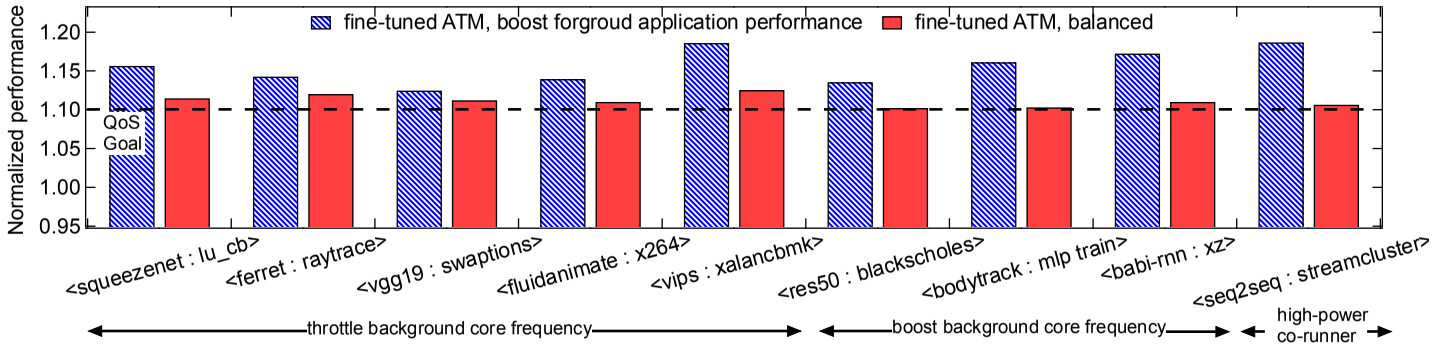
Under static margin, the default DVFS governor makes POWER7+ processors clock at fixed 4.2 GHz to run applications, providing predictable but low performance.

In Figure 5.16a, the default active timing margin improves performance uniformly for all cores, not with the highest efficiency. An unmanaged system ignores the sensitivity of core frequency to total chip power. active timing margin may be indiscriminately activated on all cores, both for `critical` and `background` workloads, which significantly raises total chip power, eroding timing margin and reducing all cores' frequency, thereby diminishing the `critical` application performance. This unmanaged system still increases frequency thanks to active timing margin's harnessed margin, but the improvement is restricted to only 6.1% on average.

Aggressively customized active timing margin provides more frequency gain, but an unmanaged system prevents the processor from providing maximum benefit. Compared with the default active timing margin, an unmanaged



(a) Fine-tuning core-level ATM enhances application performance compared to the default ATM, but scheduling workloads to the slow core and co-locating high-power jobs limit the performance benefits.



(b) In a fine-tuned ATM multicore, scheduling workloads to the fast core, and throttling co-located workload power can boost performance by 15.4%. With proper management, a 10% performance improvement goal is guaranteed for `critical workloads` while `background workloads` performance is maximized.

Figure 5.16: `Critical` application performance under different timing margin and scheduling settings; shown as $< critical : background >$ pairs. The result is normalized to the performance under static margin's fixed 4.2 GHz point, and the goal is to achieve 10% improvement over static margin.

processor system may carelessly assign the slowest core after CPM reconfiguration to a `critical` job, limiting the peak performance that can be achieved. The unmanaged system may also let all co-located `background` workloads run under their highest frequency, increasing total chip power and reducing `critical` workload frequency. However, in this scenario `critical` applications still enjoy 10.2% improvement over static margin because customizing active timing margin unlocks substantial frequency gain.

In Figure 5.16b, a managed active timing margin system can opt to maximize the performance of `critical` applications. Specifically, `critical` applications get assigned to the fastest cores, and `background` application power is minimized by applying the lowest p-state. In this way, `critical` application frequency is maximized, at the cost of `background` workload performance. On average, `critical` workload performance improves by 15.2%.

Alternatively, a managed active timing margin system can opt to balance `critical` and `background` jobs by letting `critical` applications just meet their performance goal, and maximizing `background` performance under that promise. Suppose the user targets 10% performance improvement for a `critical` workload over the static margin run, our managed system then throttles `background` core frequencies with the minimal amount to control total chip power, letting the frequency of the core running `critical` workload reach the level that delivers target performance. Compared with the schedule that maximizes `critical` application performance, the managed schedule on average doubles the frequency of cores running `background` workloads, which

is estimated to provide over 50% performance background application performance improvement.

In Figure 5.16 the performance of squeezenet, ferret, vgg19, and fluidanimate exceeds the 10% improvment target when the chip aims at maximizing their performance. However, their performance drops below the target when the chip puts all cores into customized active timing margin states. A balanced point can be obtained by controlling background workload frequency. In this case, the frequency of co-located lu_cb, raytrace, swaptions, and x264 is set to the 4.2 GHz p-state.

On the contrary, seq2seq outperform the 10% improvement goal when its co-located streamcluster runs under customized active timing margin. This is because streamcluster consumes little power even when the frequency is high. The extra available power budget can be exploited by swapping streamcluster with a more power-hungry co-runner, lu_cb, with core frequency properly throttled.

The other critical and background workloads combinations meet the QoS target when active timing margin is aggressively customized for all cores. The high-frequency gain of active timing margin customization provides this benefit. For these cases, no core throttling needs to take place.

In summary, core-level active timing margin customization and active timing margin-aware application power management provide 5% to 10% steady performance improvement over the original active timing margin system. This

result is notable because the improvement comes on a production-grade system where even a 1% performance gain is considered significant.

5.6 Related Work

There is a plethora of work on process variation, inter-core performance heterogeneity, and multicore scheduling [60, 91, 97, 85, 24, 84]. We leverage active timing margin's capability of tracking a core's inherent speed and do not need prior knowledge on the core's max frequency. Our proposal for core-level active timing margin customization conveniently expose the inter-core performance heterogeneity and help users leverage it.

Prior art has shown multi-core performance interference through the memory subsystem [67, 96, 23, 65, 102, 63]. Our work is the first to show that on an active timing margin system, the shared power delivery subsystem introduces a new dimension of resource contention, and proper management is required to reap active timing margin's full performance benefits in a predictable way.

Chapter 6

Conclusion

This chapter provides the conclusion of my dissertation work. The retrospective part (Chapter 6.1) summarize my Ph.D. research work and distill some of the important lessons learned during this process. The prospective part (Chapter 6.2) envisions how to apply and generalize this dissertation's research effort into more general computing systems, and provides the author's own remark on how computing will move forward, and how to steer one's research focus as well as the career path at the time this dissertation is written.

6.1 Retrospective

My dissertation provides comprehensive, measurement-based analysis of a microprocessor's timing margin characteristics under different environmental variation, namely temperature, voltage, and process. The data and insights presented in this thesis is extracted all using in-silicon sensor measurement, thus providing critical guidance on what causes the timing margin to be overprovisioned, how to reclaim it using active timing margin style solution, and how to design system from software to hardware in order to help active timing margin performs the best.

The timing margin problem is admittedly rooted in the microprocessor's circuit level and even device level behavior. Yet, this dissertation shows that to extract the full efficiency, architecture and software level co-design and management that impacts hardware behavior, specifically power consumption which indirectly affects chip temperature and voltage loss, are of significant benefits. Specifically:

For temperature variation (Chapter 3), we identify the huge timing margin slack caused by the significant circuit timing variation in the temperature inversion region and propose a table lookup named based feedback loop, i.e., T_i-states, for active timing margin. We note the time scale of temperature variation is typically at the order of ms, so the table storage and lookup action can be put in off-chip hardware, or in system software, such as the OS or device driver. Furthermore, we find that for the system that employs T_i-states, high workload temperature can reduce total system power, by balancing leakage power increase with dynamic power decrease. Thus, whole-system management of processor temperature can be in place to reduce total chip power.

For voltage variation/noise, hardware, or circuit level solution is mandatory to deal with the fast-occurring nature of di/dt effects, as is the case of the adaptive clocking system we study in Chapter 4 and Chapter 5. For these systems, we conduct in-depth measurement to understand the mitigation of voltage noise, after a decade of meaningful investigation of its architecture-level causes [36, 78, 37, 39, 38, 88, 89, 69, 108], and find that, similar to the case of temperature variation, longer-term IR voltage loss caused by ap-

plication power consumption limits the efficiency improvement we can gain. We propose load-balanced application scheduling to help individual processors achieve its best power saving.

For process variation, it has been long known that each individual core has their own operating frequency [60, 91, 97, 85, 24, 84], although providing per-core frequency points in a multicore induces substantial test effort and performance variation issues, which is also the reason why existing multicores all employ uniform frequency determined by the slowest core. We explore leveraging the core-level adaptive clocking loop to track individual core's speed, which not only frees up frequency from runtime effects that occasionally erode the timing margin such as temperature and voltage variation but also brings up the fast cores which are suffering from the excess margin, dictated by the slow cores. We further propose application scheduling and throttling mechanism to manage performance variation, in the presence of static frequency heterogeneity caused by core-to-core process variation, as well as the runtime frequency variation caused by power delivery system sharing on these active timing margin systems.

6.2 Prospective

This dissertation provides an in-depth study on timing margin and its optimization across system stack. Based on commercial hardware measurement, the insights presented thus render itself useful for industrial practice. Although the active timing margin techniques we study are explored on CPU

and GPU architectures, they ideally apply to any other platforms. To facilitate ease of adoption, future work can be carried out to put the research fruition in this dissertation into more detail for robust production adoption.

For T_i-states, an automatic procedure to build the temperature to voltage conversion table can be implemented and tested. It is worthwhile to understand the max within-die temperature gradient due to local hotspots and its impact of voltage reduction magnitude. It is also worthwhile to understand how different circuit cell types affect the T_i-state tables as their threshold voltage varies. For latest 7nm technology, a thorough evaluation is needed to understand to the trade-off between leveraging the device's low leakage power for frequency and performance enhancement, or power reduction which exposes space for temperature management in synergy with T_i-states.

For adaptive clocking, or adaptive instruction issue system, an automatic procedure is also needed to speedup per-core timing margin sensor calibration for identifying the safe customization point for ultimate performance, as proposed in Chapter 5.5.1. With shared power delivery, application scheduling and throttling are needed for performance management. Alternatively, architecture-level re-design of on-chip power delivery network can also reduce inter-core interference from IR voltage drop loss. Combined with Integrated Voltage Regulators (IVRs), the complete design space is yet to be covered.

Active timing margin management for voltage and process variation can be effectively combined with adaptive clocking's programmable interface, as discussed in Chapter 5.1. However, unified timing margin optimization that

additional incorporates temperature variation does not exist yet. The speedup effect of temperature inversion provides new opportunity for chip power and performance optimization. Unlike the management for voltage variation, where power reduction is favored to reduce static DC voltage drop, temperature inversion may favor high power scenarios that increase chip temperature and opens up more timing margin slack, as Chapter 3.4 projects. The trade-off between circuit speedup caused by temperature inversion, DC voltage variation, and leakage power is yet to be explored.

Moving forward, the era of general purpose processor performance benefits through Moore's law and Dennard scaling has undoubtedly ended. The future of computing system enhancement will depend on domain-specific accelerator hardware design, accompanying software toolchain design, and convenient tools for low-cost, fast prototyping and testing [40]. The timing margin optimizations proposed in this dissertation can be embedded into the resulting system, should ultra power/performance efficiency is demanded. The author believes as a computer architect, identifying key application domains, and shipping the accompanying hardware-software system that is of economic value is a major task for the coming decade.

Bibliography

[1] Zou An, Jingwen Leng, Xin He, Yazhou Zu, Christopher D Gill, Vijay Janapa Reddi, and Xuan Zhang. Voltage-stacked gpus: A control theory driven cross-layer solution for practical voltage stacking in gpus. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[2] ATI. Ati tool. `http://www.techpowerup.com/atitool/`, 2006. [Online; accessed 11-Oct-2018].

[3] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 2013.

[4] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE computer*, 2007.

[5] John Barth, Don Plass, Erik Nelson, Charlie Hwang, Gregory Fredeman, Michael Sperling, Abraham Mathews, William R Reohr, Kavita Nair, and N Cao. A 45nm soi embedded dram macro for POWER7 32mb on-chip l3 cache. In *International Solid-State Circuits Conference (ISSCC)*, 2010.

[6] A Bellaouar, A Fridi, MI Elmasry, and K Itoh. Supply voltage scaling for temperature insensitive cmos circuit operation. *IEEE Transactions on Circuits and Systems II: Analog and Digital signal processing*, 1998.

[7] C Berry, J Warnock, J Badar, DG Bair, E Behnen, B Bell, A Buyuktosunoglu, C Cavitt, P Chuang, O Geva, et al. IBM z14 design methodology enhancements in the 14-nm technology node. *IBM Journal of Research and Development*, 2018.

[8] Ramon Bertran, Alper Buyuktosunoglu, Pradip Bose, Timothy J Slegel, Gerard Salem, Sean Carey, Richard F Rizzolo, and Thomas Strach. Voltage noise in multi-core processors: Empirical characterization and optimization opportunities. In *Proceedings of the International Symposium on Microarchitecture(MICRO)*, 2014.

[9] Ramon Bertran, Alper Buyuktosunoglu, Meeta S Gupta, Marc Gonzalez, and Pradip Bose. Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks. In *International Symposium on Microarchitecture (MICRO)*, 2012.

[10] Christian Bienia, Sanjeev Kumar, and Kai Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2008.

[11] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li.

The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel architectures and compilation techniques (PACT)*, 2008.

[12] David Blaauw, Sudherssen Kalaiselvan, Kevin Lai, Wei-Hsiang Ma, Sanjay Pant, Carlos Tokunaga, Shidhartha Das, and David Bull. Razor ii: in situ error detection and correction for pvt and ser tolerance. 2008.

[13] Keith Bowman, Sarthak Raina, Todd Bridges, Daniel Yingling, Hoan Nguyen, Brad Appel, Yesh Kolla, Jihoon Jeong, Francois Atallah, and David Hansquine. A 16nm auto-calibrating dynamically adaptive clock distribution for maximizing supply-voltage-droop tolerance across a wide operating range. In *International Solid-State Circuits Conference (ISSCC)*, 2015.

[14] Keith A Bowman, Carlos Tokunaga, Tanay Karnik, Vivek K De, and Jim W Tschanz. A 22nm dynamically adaptive clock distribution for voltage droop tolerance. In *IEEE Symposium on VLSI Circuits (VLSIC)*, 2012.

[15] Ermao Cai and Diana Marculescu. Tei-turbo: temperature effect inversion-aware turbo boost for finfet-based multi-core systems. In *International Conference on Computer-Aided Design (ICCAD)*, 2015.

[16] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Pun-

eet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *The International Symposium on Microarchitecture (MICRO)*, 2016.

[17] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International conference on Architectural support for programming languages and operating systems (ASPLOS)*, 2014.

[18] Standard Performance Evaluation Corporation. Spec cpu 2006. `https://www.spec.org/cpu2006/`, 2006. [Online; accessed 11-Oct-2018].

[19] Standard Performance Evaluation Corporation. Spec cpu 2017. `https://www.spec.org/cpu2017/`, 2017. [Online; accessed 11-Oct-2018].

[20] McCalpin John D. Stream. `https://www.cs.virginia.edu/stream/`. [Online; accessed 11-Oct-2018].

[21] Ali Dasdan and Ivan Hom. Handling inverted temperature dependence in static timing analysis. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2006.

[22] Jeff Dean, David Patterson, and Cliff Young. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 2018.

[23] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. 2014.

[24] Saurabh Dighe, Sriram Vangal, Paolo Aseron, Shasi Kumar, Tiju Jacob, Keith Bowman, Jason Howard, James Tschanz, Vasantha Erraguntla, Nitin Borkar, et al. Within-die variation-aware dynamic-voltage-frequency scaling core mapping and thread hopping for an 80-core processor. In *International Solid-State Circuits Conference (ISSCC)*, 2010.

[25] Alan Drake, R Senger, Harmander Deogun, G Carpenter, Soraya Ghiasi, Tuyet Nguyen, N James, M Floyd, and Vikas Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *International Solid-State Circuits Conference (ISSCC)*, 2008.

[26] Alan J Drake, Michael S Floyd, Richard L Willaman, Derek J Hathaway, Joshua Hernandez, Crystal Soja, Marshall D Tiner, Gary D Carpenter, and Robert M Senger. Single-cycle, pulse-shaped critical path monitor in the POWER7+ microprocessor. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*. IEEE, 2013.

[27] EEMBC. Coremark. `https://www.eembc.org/coremark/`. [Online; accessed 11-Oct-2018].

[28] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level

timing speculation. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2003.

[29] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *ACM SIGARCH Computer Architecture News*, 2011.

[30] Songchun Fan, S Zahedi, and B Lee. The computational sprinting game. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[31] Tim Fischer, Ferd Anderson, Ben Patella, and Sam Naffziger. A 90nm variable-frequency clock system for a power-managed itanium®-family processor. In *International Solid-State Circuits Conference (ISSCC)*, 2005.

[32] Michael Floyd, Malcolm Allen-Ware, Karthick Rajamani, Bishop Brock, Charles Lefurgy, Alan J Drake, Lorena Pesantez, Tilman Gloekler, Jose A Tierno, Pradip Bose, et al. Introducing the adaptive energy management features of the POWER7 chip. *IEEE Micro*, 2011.

[33] MS Floyd, A Drake, NS Schwartz, RW Berry, CR Lefurgy, M Ware, K Rajamani, V Zyuban, R Willaman, and RM Zgabay. Runtime power reduction capability of the IBM POWER7+ chip. *IBM Journal of Research and Development*, 57(6):2–1, 2013.

[34] Kevin Gillespie, Harry R Fair, Carson Henrion, Ravi Jotwani, Stephen Kosonocky, Robert S Orefice, Donald A Priore, Jonathan White, and Kathryn Wilcox. Steamroller: An x86-64 core implemented in 28nm bulk cmos. In *International Solid-State Circuits Conference (ISSCC)*, 2014.

[35] Aaron Grenat, Sanjay Pant, Ravinder Rachala, and Samuel Naffziger. Adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor. In *International Solid-State Circuits Conference (ISSCC)*, 2014.

[36] Ed Grochowski, David Ayers, and Vivek Tiwari. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2002.

[37] Meeta S Gupta, Jarod L Oatley, Russ Joseph, Gu-Yeon Wei, and David M Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2007.

[38] Meeta S Gupta, Vijay Janapa Reddi, Glenn Holloway, Gu-Yeon Wei, and David M Brooks. An event-guided approach to reducing voltage noise in processors. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2009.

[39] Meeta Sharma Gupta, Krishna K Rangan, Michael D Smith, Gu-Yeon Wei, and David Brooks. Decor: A delayed commit and rollback mechanism for handling inductive noise in processors. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[40] John Hennessy and David Patterson. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development.

[41] Wei Huang, Shougata Ghosh, Siva Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on Very Large Scale Integration Systems (VLSI)*, 2006.

[42] Wei Huang, Charles Lefurgy, William Kuk, Alper Buyuktosunoglu, Michael Floyd, Karthick Rajamani, Malcolm Allen-Ware, and Bishop Brock. Accurate fine-grained processor power proxies. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2012.

[43] IBM. Automated measurement of systems for temperature and energy reporting. `https://github.com/open-power/amester`, 2015. [Online; accessed 11-Oct-2018].

[44] Intel. Fpga timing model. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01139-timing-model.pdf`. [Online; accessed 11-Oct-2018].

[45] Intel. Intel vrm design guideline. https://www.intel.com/content/www/us/en/power-management/voltage-regulator-module-enterprise-voltage-regulator-down-11-1-guidelines.html, 2009. [Online; accessed 11-Oct-2018].

[46] Intel. Intel 22nm finfet technology. http://www.intel.com/content/www/us/en/silicon-innovations/intel-22nm-technology.html, 2011. [Online; accessed 11-Oct-2018].

[47] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2003.

[48] N James, Phillip Restle, Joshua Friedrich, Bill Huott, and B McCredie. Comparison of split-versus connected-core supplies in the power6 microprocessor. In *International Solid-State Circuits Conference (ISSCC)*, 2007.

[49] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture (ISCA)*, 2017.

[50] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits

in memory without accessing them: An experimental study of dram disturbance errors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014.

[51] Youngtaek Kim, Lizy Kurian John, Sanjay Pant, Srilatha Manne, Michael Schulte, William Lloyd Bircher, and Madhu Saravana Sibi Govindan. Audit: Stress testing the automatic way. In *International Symposium on Microarchitecture (MICRO)*, 2012.

[52] Nasser Kurd, Jonathan Douglas, Praveen Mosalikanti, and Rajesh Kumar. Next generation intel® micro-architecture (nehalem) clocking architecture. In *IEEE Symposium on VLSI Circuits (VLSIC)*, 2008.

[53] Woojoo Lee, Yanzhi Wang, Tiansong Cui, Shahin Nazarian, and Massoud Pedram. Dynamic thermal management for finfet-based circuits exploiting the temperature effect inversion phenomenon. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[54] Charles R Lefurgy, Alan J Drake, Michael S Floyd, Malcolm S Allen-Ware, Bishop Brock, Jose A Tierno, and John B Carter. Active management of timing guardband to save energy in POWER7. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.

[55] Charles R Lefurgy, Alan J Drake, Michael S Floyd, Malcolm S Allen-Ware, Bishop Brock, Jose A Tierno, John B Carter, and Robert W

Berry. Active guardband management in POWER7+ to save energy and maintain reliability. *IEEE Micro*, 2013.

[56] Jingwen Leng, Alper Buyuktosunoglu, Ramon Bertran, Pradip Bose, and Vijay Janapa Reddi. Safe limits on voltage reduction efficiency in gpus: a direct measurement approach. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015.

[57] Jingwen Leng, Yazhou Zu, and Vijay Janapa Reddi. Gpu voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in gpu architectures. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[58] Jingwen Leng, Yazhou Zu, Minsoo Rhu, Meeta Gupta, and Vijay Janapa Reddi. Gpuvolt: Modeling and characterizing voltage noise in gpu architectures. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.

[59] Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2014.

[60] Xiaoyao Liang, Ramon Canal, Gu-Yeon Wei, and David Brooks. Process variation tolerant 3t1d-based cache architectures. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007.

[61] C-H Lin, B Greene, S Narasimha, J Cai, A Bryant, C Radens, V Narayanan, B Linder, H Ho, A Aiyar, et al. High performance 14nm soi finfet cmos technology with 0.0174μm 2 embedded dram and 15 levels of cu metallization. In *International Electron Devices Meeting (IEDM)*, 2014.

[62] Q Liu, B DeSalvo, P Morin, N Loubet, S Pilorget, F Chafik, S Maitrejean, E Augendre, D Chanemougame, S Guillaumet, et al. Fdsoi cmos devices featuring dual strained channel and thin box extendable to the 10nm node. In *International Electron Devices Meeting*, 2014.

[63] Qiuyun Llull, Songchun Fan, Seyed Majid Zahedi, and Benjamin C Lee. Cooper: Task colocation with cooperative games. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2017.

[64] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the International Symposium on Computer Architecuture (ISCA)*, 2014.

[65] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: improving resource efficiency at scale. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2015.

[66] Stelios Manousopoulos, Miquel Moreto, Roberto Gioiosa, Nectarios Koziris, and Francisco J Cazorla. Characterizing thread placement in the IBM

POWER7 processor. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2012.

[67] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.

[68] Alex Mericas. Performance characteristics of the power8 processor. In *IEEE Hot Chips(HCS)*, 2014.

[69] TN Miller, R Thomas, Xiang Pan, and R Teodorescu. Vrsync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.

[70] Benjamin Munger, David Akeson, Srikanth Arekapudi, Tom Burd, Harry R Fair III, Jim Farrell, Dave Johnson, Guhan Krishnan, Hugh McIntyre, Edward McLellan, et al. Carrizo: A high performance, energy efficient 28 nm apu. *Journal of Solid-State Circuits (JSSC)*, 2016.

[71] Preeti U. Murthy. Overview of the current approaches to enhance the linux scheduler. In *Linux Foundation Collaboration Summit*, 2013.

[72] S Natarajan, M Agostinelli, S Akbar, M Bost, A Bowonder, V Chikarmane, S Chouksey, A Dasgupta, K Fischer, Q Fu, et al. A 14nm

logic technology featuring 2 nd-generation finfet, air-gapped interconnects, self-aligned double patterning and a 0.0588 μm 2 sram cell size. In *International Electron Devices Meeting (IEDM)*, 2014.

[73] George Papadimitriou, Athanasios Chatzidimitriou, Manolis Kaliorakis, Yannos Vastakis, and Dimitris Gizopoulos. Micro-viruses for fast system-level voltage margins characterization in multicore cpus. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018.

[74] George Papadimitriou, Manolis Kaliorakis, Athanasios Chatzidimitriou, Dimitris Gizopoulos, Peter Lawthers, and Shidhartha Das. Harnessing voltage margins for energy efficiency in multicore cpus. In *International Symposium on Microarchitecture (MICRO)*, 2017.

[75] Changhae Park, Jay P John, Kevin Klein, Jim Teplik, Jim Caravella, Jim Whitfield, Ken Papworth, and Sunny Cheng. Reversal of temperature dependence of integrated circuits operating at very low voltages. In *International Electron Devices Meeting (IEDM)*, 1995.

[76] Massoud Pedram and Jan M Rabaey. *Power aware design methodologies*. Springer Science & Business Media, 2002.

[77] Bertrand Pelloux-Prayer, Milovan Blagojević, Olivier Thomas, Amara Amara, Andrei Vladimirescu, Borivoje Nikolić, Giorgio Cesana, and Philippe Flatresse. Planar fully depleted soi technology: The convergence of high performance and low power towards multimedia mobile

applications. In *IEEE Faible Tension Faible Consommation (FTFC)*, 2012.

[78] Michael D Powell and TN Vijaykumar. Pipeline damping: A microarchitectural technique to reduce inductive noise in supply voltage. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.

[79] Wajahat Qadeer, Rehan Hameed, Ofer Shacham, Preethi Venkatesan, Christos Kozyrakis, and Mark Horowitz. Convolution engine: Balancing efficiency and flexibility in specialized computing. *Communications of the ACM*, 2015.

[80] Jan Rabaey. *Low power design essentials*. Springer Science & Business Media, 2009.

[81] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolic. *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002.

[82] Ravinder Rachala, Stephen Kosonocky, Khaled Heloue, Rajashekhar Bochkar, Aparna Tula, Miguel Rodriguez, and Erhan Ergin. Design methodology for operating in near-threshold-computing (ntc) region. In *Design Automation Conference (DAC)*, 2016.

[83] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P Pipe, Thomas F Wenisch, and Milo MK Martin. Compu-

tational sprinting. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[84] Krishna K Rangan, Michael D Powell, Gu-Yeon Wei, and David Brooks. Achieving uniform performance and maximizing throughput in the presence of heterogeneity. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[85] Krishna K Rangan, Gu-Yeon Wei, and David Brooks. Thread motion: fine-grained power management for multi-core systems. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009.

[86] Vijay Janapa Reddi, Simone Campanoni, Meeta S Gupta, Michael D Smith, Gu-Yeon Wei, David Brooks, and Kim Hazelwood. Eliminating voltage emergencies via software-guided code transformations. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2010.

[87] Vijay Janapa Reddi and Meeta Sharma Gupta. Resilient architecture design for voltage variation. *Synthesis Lectures on Computer Architecture*, 2013.

[88] Vijay Janapa Reddi, Meeta Sharma Gupta, Glenn Holloway, Gu-Yeon Wei, Michael D Smith, and David Brooks. Voltage emergency prediction: Using signatures to reduce operating margins. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[89] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D Smith, Gu-yeon Wei, and David Brooks. Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2010.

[90] Samsung. Samsung 14nm finfet. `https://www.samsungfoundry.com/foundry/homepage/anonymous/technology12inch14nm.do?_mainLayOut=homepageLayout&menuIndex=020102`, 2012. [Online; accessed 11-Oct-2018].

[91] Smruti R Sarangi, Brian Greskamp, Radu Teodorescu, Jun Nakano, Abhishek Tiwari, and Josep Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[92] Balaram Sinharoy, R Kalla, WJ Starke, HQ Le, R Cargnoni, JA Van Norstrand, BJ Ronchetti, J Stuecheli, J Leenstra, GL Guthrie, et al. IBM POWER7 multicore server processor. *IBM Journal of Research and Development*, 2011.

[93] Kevin Skadron, Mircea R Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2004.

[94] Shuang Song, Reena Panda, Joseph Dean, and Lizy K John. Wait of a decade: Did spec cpu 2017 broaden the performance? In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[95] Sriram Sundaram, Sriram Samabmurthy, Michael Austin, Aaron Grenat, Michael Golden, Stephen Kosonocky, and Samuel Naffziger. Adaptive voltage frequency scaling using critical path accumulator implemented in 28nm cpu. In *Proceedings of the International Conference on VLSI Design (VLSID)*, 2016.

[96] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The impact of memory subsystem resource sharing on datacenter applications. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2011.

[97] Radu Teodorescu and Josep Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2008.

[98] Thomas N Theis and H-S Philip Wong. The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering*, 2017.

[99] José Tierno, Alexander Rylyakov, Daniel Friedman, Ann Chen, Anthony Ciesla, Timothy Diemoz, George English, David Hui, Keith Jenkins,

Paul Muench, et al. A dpll-based per core variable frequency clock generator for an eight-core POWER7 microprocessor. In *Symposium on VLSI Circuit Digest of Tech Papers*, 2010.

[100] Carlos Tokunaga, Joseph F Ryan, Charles Augustine, Jaydeep P Kulkarni, Yi-Chun Shih, Stephen T Kim, Rinkle Jain, Keith Bowman, Arijit Raychowdhury, Muhammad M Khellah, et al. A graphics execution core in 22nm cmos featuring adaptive clocking, selective boosting and state-retentive sleep. In *International Solid-State Circuits Conference (ISSCC)*, 2014.

[101] James Tschanz, Nam Sung Kim, Saurabh Dighe, Jason Howard, Gregory Ruhl, Sriram Vangal, Siva Narendra, Yatin Hoskote, Howard Wilson, Carol Lam, et al. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *International Solid-State Circuits Conference (ISSCC)*, 2007.

[102] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2015.

[103] Christos Vezyrtzis, Thomas Strach, I Pierce, Jen Chuang, Preetham Lobo, Richard Rizzolo, Tobias Webel, Pawel Owczarczyk, Alper Buyuktosunoglu, Ramon Bertran, David Hui, Susan Eickhoff, Michael Floyd, Gerald Salem, Sean Carey, Stelios Tsapepas, and Phillip Restle. Droop

mitigation using critical-path sensors and an on-chip distributed power supply estimation engine in the z14 enterprise processor. In *International Solid-State Circuits Conference-(ISSCC)*, 2018.

[104] T Webel, PM Lobo, R Bertran, GM Salem, M Allen-Ware, R Rizzolo, SM Carey, T Strach, A Buyuktosunoglu, C Lefurgy, et al. Robust power management in the IBM z13. *IBM Journal of Research and Development*, 2015.

[105] David Wolpert and Paul Ampadu. Temperature effects in semiconductors. In *Managing Temperature Effects in Nanoscale Adaptive Systems*. Springer, 2012.

[106] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1995.

[107] Shien-Yang Wu, Colin Yu Lin, MC Chiang, JJ Liaw, JY Cheng, SH Yang, Ming Liang, Tadakazu Miyashita, CH Tsai, BC Hsu, et al. A 16nm finfet cmos technology for mobile soc and computing applications. In *International Electron Devices Meeting (IEDM)*, 2013.

[108] Runjie Zhang, Ke Wang, Brett H Meyer, Mircea R Stan, and Kevin Skadron. Architecture implications of pads as a scarce resource. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014.

[109] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on Electron Devices*, 2006.

[110] Yuhao Zhu. *Energy-efficient mobile Web computing*. PhD thesis, 2017.

[111] An Zou, Jingwen Leng, Xin He, Yazhou Zu, Vijay Janapa Reddi, and Xuan Zhang. Efficient and reliable power delivery in voltage-stacked manycore system with hybrid charge-recycling regulators. In *Proceedings of the Design Automation Conference (DAC)*, 2018.

[112] An Zou, Jingwen Leng, Yazhou Zu, Tao Tong, Vijay Janapa Reddi, David Brooks, Gu-Yeon Wei, and Xuan Zhang. Ivory: Early-stage design space exploration tool for integrated voltage regulators. In *Proceedings of the Design Automation Conference (DAC)*, 2017.

[113] Yazhou Zu, Wei Huang, Indrani Paul, and Vijay Janapa Reddi. Tistates: Processor power management in the temperature inversion region. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[114] Yazhou Zu, Wei Huang, Indrani Paul, and Vijay Janapa Reddi. Tistates: Power management in active timing margin processors. *IEEE Micro*, 37(3):106–114, 2017.

[115] Yazhou Zu, Charles R Lefurgy, Jingwen Leng, Matthew Halpern, Michael S Floyd, and Vijay Janapa Reddi. Adaptive guardband scheduling

to improve system-level efficiency of the POWER7+. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015.

[116] Yazhou Zu, Daniel Richins, Charles R Lefurgy, and Vijay Janapa Reddi. Active timing margin management for maximizing multi-core efficiency on an IBM POWER7+ server. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[117] V Zyuban, SA Taylor, B Christensen, AR Hall, CJ Gonzalez, J Friedrich, F Clougherty, J Tetzloff, and R Rao. IBM POWER7+ design for higher frequency at fixed power. *IBM Journal of Research and Development*, 2013.

Vita

Yazhou Zu was born in the city of Changzhou, Jiangsu Province, China, in November 1990. He received the Bachelor of Engineering degree in Microelectronics from Shanghai Jiao Tong University, Shanghai, China in 2013. In his senior year of undergraduate study, Yazhou worked as a research intern in the Polytechnique Montral and Shanghai Jiao Tong University, sponsored by China Scholarship Council's State Fund Undergraduate Scholarship Award and Mitacs Canada's Globalink Research Internship Award. During this period, Yazhou worked with professor Brunild Sanso and Yongxin Zhu on server farm power management in virtualized datacenters.

Yazhou is now a graduate researcher and Ph.D. candidate at the Electrical and Computer Engineering Department in The University of Texas at Austin, Austin, TX, working with professor Vijay Janapa Reddi. He is on track to graduate in Oct 2018. During his Ph.D. study, Yazhou was a research intern/Co-op engineer at SRI International, AMD Research, IBM Thomas J. Watson Research Center, and Facebook Inc.

Email address: yazhou.zu@utexas.edu

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.